FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Geheime Kommunikation im Internet

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Johanna Ullrich, BSc
Matrikelnummer 0725189

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dipl.-Ing. Mag.rer.soc.oec. Dr. techn. Edgar Weippl
Zweitbetreuung: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby

Diese Dissertation haben begutachtet:

Davide Balzarotti	Günther Pernul

Wien, 12. August 2016

Johanna Ullrich

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Secret Communication in the Internet

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Dipl.-Ing. Johanna Ullrich, BSc
Registration Number 0725189

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dipl.-Ing. Mag.rer.soc.oec. Dr. techn. Edgar Weippl
Second advisor: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby

The dissertation has been reviewed by:

Davide Balzarotti

Günther Pernul

Vienna, 12th August, 2016

Johanna Ullrich

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Johanna Ullrich, BSc
Hauptstraße 44, 2481 Achau

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 12. August 2016

Johanna Ullrich

Abstract

Secret communication characterizes clandestine approaches of communication: *Covert channels* conceal a communication's mere existence, *side channels* are unintended by the sender, and *obfuscation* conceals sender and/or receiver or hinders their correlation. The ability to establish such secret communication provides a powerful instrument to adversaries; attacks involving secret communication encompass in general three steps: (1) the *development of the secret communication channel*, (2) the *extraction of information* using this channel and finally (3) *exploitation* of the gained information to cause further harm. Hitherto, research concentrates on the first aspect – channel development – and assesses channel capacities to evaluate a channel's impact on security. The more capacity, the more dangerous a channel is considered. In some scenarios, a single bit of transmitted data however suffices, whereas in other situations a high-capacity channel is useless due to an overall lack of sensitive data. Hence, it is more promising to include the latter two aspects, and ask for the information gained by an adversary as well as the advantages she takes from this information. This line of action also implies that secret communication must not be considered separately from its context.

In this thesis, we strive to advance research through the development of attack paths including all steps from channel development to exploitation in order to improve the understanding of secret communication and its impact on security. For the context, we choose two contemporary scenarios in computer science, cloud computing and the Internet Protocol version 6 (IPv6). While the first is a recently introduced operating model that provides new functionality by reusing existing technology, the latter is a novel technology replacing its predecessor with (almost) the same functionality and is going to affect all Internet users – consciously or unconsciously – in the long run. We develop two full attacks per context; our results emphasize that secret communication serves both, benign and malicious, goals.

Kurzfassung

Geheime Kommunikation bezeichnet heimliche Arten der Datenübertragung: *Covert Channels* verbergen die Existenz von Kommunikation, ein *Side Channel* ist vom Sender unerwünschte Kommunikation, und *Obfuscation* verschleiert Sender und/oder Empfänger bzw. verhindert deren Korrelation. Für Angreifer ist geheime Kommunikation ein mächtiges Werkzeug, und deren Angriffe bestehen üblicherweise aus drei Schritten: (1) die *Entwicklung des geheimen Kommunikationskanals*, (2) die *Informationsgewinnung mithilfe des Kanals* sowie (3) die *Ausnutzung* der erlangten Information um weiteren Schaden anzurichten. Bisherige Arbeiten betrachten überwiegend den ersten Aspekt, die Entwicklung von Kanälen. Um deren Einfluss auf die Security abschätzen zu können, werden Übertragungskapazitäten beurteilt. Je mehr Kapazität ein Kanal hat, als desto gefährlicher gilt er. Es gibt jedoch Szenarien, in denen ein einziges Bit an übertragener Information ausreicht. Andererseits können Kanäle mit hoher Kapazität nutzlos sein, wenn es keine sensiblen Daten, die verraten werden können, gibt. Deshalb ist es versprechender auch die beiden anderen Schritte miteinzubeziehen, und nach der Information, die ein Angreifer erlangen kann, sowie deren Nutzen für den Angreifer zu fragen. Diese Vorgehensweise impliziert auch, dass geheime Kommunikation nicht unabhängig vom Anwendungsszenario betrachtet werden kann.

In dieser Arbeit streben wir danach, das Verständnis für geheime Kommunikation und deren Einfluss auf die Security zu erweitern. Wir entwickeln Angriffspfade, die alle drei Schritte von der Entwicklung bis zur Ausnutzung umfassen. Als Rahmen wählen wir zwei gegenwärtige Szenarien der Informatik, nämlich Cloud Computing sowie das Internet Protocol version 6 (IPv6). Cloud Computing ist ein Betriebsmodell, das neue Funktionalität für eine breite Kundenbasis durch Wiederverwendung von vorhandener Technologie bietet. IPv6 ist hingegen eine neue Technologie, die ihren Vorgänger mit mehr oder weniger derselben Funktionalität ersetzen soll, und langfristig alle Internetnutzer (ob bewusst oder unbewusst) betreffen wird. Wir entwickeln zwei vollständige Angriffe pro Szenario, und unsere Ergebnisse zeigen, dass geheime Kommunikation in guter, aber auch in böser Absicht genutzt werden kann.

Acknowledgements

Writing a PhD thesis is an endeavor, and impossible without support. First and foremost, I am grateful to my advisor Edgar Weippl; you have provided me a stimulant environment at SBA Research for diving into the world of academia, given me the freedom to follow my own research ideas and had faith in my capabilities despite my anything but strong background in security. Lest we forget, Edgar introduced me to Tanja Zseby who supported me in improving my scientific skills and taught me to always aim high. Thank you, Tanja, for being a great role model over the last years. I owe gratitude to the competent, and caring senior researchers I met – Joachim Fabini, Artemios Voyiatzis, Dimitris Simos and Christian Rossow. Further, I thank my reviewers Davide Balzarotti and Günther Pernul for reviewing the script, providing feedback and enabling (and motivating) me to bring the thesis to its final state.

My colleagues at SBA have enriched my life and made the office hours a more (or should I say most?) pleasant time; together, we tackled the really though issues – what to do for lunch (on every workday!), turn the AC on and/or open the windows (not simultaneously!), where do tumblers vanish to and why do the tiny ones remain behind (future work...), why are there only three-pronged forks in the kitchen, what are alternative uses for space blankets (sweeping under!) and how could we get rid of the Bitcoin miner's noise. Thank you, Adrian, Aljosha, Georg, Peter, Sebastian and all the others.

Thanks to my closest friends Katharina, Alexandra, Sabine and Julia for all the joint adventures. I enjoy every second with you; next spring (and Tichy) is waiting to come. On a daily basis, the effort of forcing me to relax was taken by Retriever Harvey and recently by Čuvač Pixel; my bowwows literally never cared whether I have to perform measurements or to write a paper. The most important is yet to come, my precious husband Philippe. Thanks for your love, encouragement and support over the last decade; it was not always easy with me. Without you, I would not be where I am today; you are not only the love of my life, but also my mentor; I learned a lot from you!

In contrast to previous works, e. g., by Martina Lindorfer, this work has been written in the abstinence of a substance called coffee; admittedly, other dark substances containing caffeine have been involved.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Transmission of data over the Internet follows a regular pattern. A sender intends to send a message to a receiver, and embeds data in packets. These packets do not only hold the payload, but also packet headers. The latter contain information that is needed for packet delivery, and designates the path to the receiver. Finally, the receiver re-extracts the data from the packets upon receipt, and processes it according to its needs. Transmitted data might be encrypted, and thus only accessible by the receiver. Nevertheless, it remains clear who intends to communicate with whom, who is the sender/receiver, what remains header data for the purpose of packet delivery, and what is the actual data – albeit an outsider might be hindered to access it as a consequence of encryption.

In contrast, secret communication characterizes more clandestine approaches of communication. Clandestineness manifests in multiple ways, and results in distinct deviations from the classic pattern of communication. *Covert channels* are intended by both, sender and receiver; but exploit a resource that is unintended for communication as the communication partners aim to conceal the communication's mere existence from observers. *Side channels* are unintended by the sender, and arise from specific implementation characteristics. The third concept is *obfuscation*; obfuscation intends to conceal sender and/or receiver, or to hinder their correlation; an observer is however aware that communication is prevalent. Summarizing their difference from the classic pattern, covert channels trick observers to believe in the non-existence of communication, side channels lack the intention for data transmission and obfuscation attempts to hide who is talking with whom. Secret communication's goals are manifold, and not necessarily of bad faith as emphasized by the following enumeration: industrial espionage, whistleblowing, censorship evasion, exchange of illegal or regime-critical content, superficial compliance to cryptography laws, gain of transmission capacity, compliance checking, reconnaissance, data extraction from compromised hosts or malware communication.

The Internet evolves, and so does secret communication; among recent changes in the Internet are cloud computing and the transition to the Internet Protocol version 6 (IPv6). The first describes a novel operational model characterized by unprecedented flexibility. A cloud provider leases out infrastructure, platforms or software to a large customer base on a pay-per-use basis. Even though utilized technologies are not novel themselves, the potential for secret communication in cloud computing is considered to be elevated as former implicit security assumptions, e. g., with regard to perimeter protection, are now violated. Whereas, the latter is a new protocol replacing today's Internet Protocol version 4 (IPv4) to overcome address shortage. While providing (almost) no additional functionality in comparison to its predecessor, it is going to affect all Internet users in the long run – consciously or unconsciously. Especially its increased address length forms a sound substrate for secret communication; this channel is especially worthwhile as addresses must not be simply scotched; they are a required for successful packet delivery.

1.2 Problem Statement/Aim of this Work

The Internet is subject to continuous further development; and secret communication evolves in step. The introduction and advancement of technologies bears chances for new approaches of secret communication, but might also supersede legacy channels; however, there will always be some sort of secret communication available. Secret communication is however not an end in itself, but rather establishes a basis for later attacks. A full attack is characterized by the following steps:

1. *Channel Development:* The adversary discovers potential for a new approach of secret communication, and develops the channel accordingly. At this stage, the adversary does not necessarily have to interact with the later victim(s); she might alternatively test the channel in a test setup.

2. *Information Extraction:* An adversary uses the previously developed channel in order to gain information from the victim. Such information might be cryptographic keys, but also Internet Protocol (IP) addresses, type and version of operation systems, availability of a certain file or local proximity.

3. *Exploitation:* The adversary investigates the gained information and exploits the latter in order to perform an attack. Attacks are diverse, and range from unauthorized access to encrypted messages using the gained key to launching denial-of-service attacks targeted to the revealed characteristics of the victim.

Hitherto, research, e. g., [1], concentrates on the first aspect, channel development. A channel's mere existence does however not necessarily point to a lack of security; it is their potential of leaking sensitive information and the latter's exploitation that transforms secret communication into a security risk. To evaluate a channel's impact on security, existing approaches assess channel capacities; the more capacity, the more dangerous

a channel is considered. For example, the *Orange Book* by the *US Department of Defense* defines a channel with more than 100 bit/s as *"high"* [2]. In some scenarios, a single bit of transmitted data suffices to cause harm, whereas in other situations a high-capacity channel is useless due to an overall lack of sensitive data. It thus remains more promising to include all three steps of an attack into consideration and to ask the following: *Which information is gained by an adversary exploiting secret communication, and which advantages does the adversary take from this information?* This line of action further implies that secret communication is dependent on the case of application, and cannot be considered separately from its context. Based on these insights, it is then possible to identify a channel's impact on security, and whether mitigation becomes necessary.

The thesis at hand overcomes this gap and focuses on the latter two aspects – extraction of information, and turning it into successful attacks. Thereby, the development of channels becomes a (clearly necessary) precondition. Due to the vast subject, the thesis' scope is limited with respect to (i) the context as well as to (ii) distinct means of secret communication. With respect to the first, we investigate secret communication in cloud computing on the one hand, and secret communication in the Internet Protocol version 6 (IPv6) on the other hand. There is a key difference between cloud computing and IPv6 with respect to novelty of technology. The first reuses available technology in order to form a novel operational model and additional functionality; the latter means replacement of technology without changing key functionality. Finally, side channels are unintended by the sender, and thus do not require the latter's active participation in communication. For this reason, we consider them a more likely attack model and dedicate the practical part of this thesis entirely to side channels.

1.3 Contributions

The contribution of this thesis is fourfold, and highlighted in the following paragraphs. The first two paragraphs consider cloud computing, the latter IPv6 networking.

We exploit side channels to investigate the role of firewalls at major Infrastructure-as-a-Service (IaaS) cloud providers; our results shed light on firewall details that are hidden by the respective providers and hence not accessible by average tenants. The developed test tool is of importance as the cloud as a whole – and in consequence also its firewalls – remain a black box for customers; but the latter have substantial interest in more details, e. g., for reasons of risk analysis, troubleshooting or compliance checking, and the official documentation is rare, barely scratching the surface.

We develop side channels revealing all configuration parameters related to the Xen hypervisor's rate limits – a functionality to throttle a virtual instance's networking in order to guarantee fair bandwidth distribution among neighbors. Their actual impact on security is dependent on the adversary's advantage; thus, we show how these side channels are further developed in an actual denial-of-service attack exploiting the same mechanism. The denial-of-service attack causes up to 88.3 percent of packet drops, or up to 13.8

seconds of packet delay. Our results show that rate limits – originally intended as a protection mechanism – snap back and become attack vectors themselves. As Xen is used in major cloud providers, a high number of virtual instances is potentially vulnerable.

We investigate a new method for IPv6 reconnaissance, i. e., the discovery of previously unknown victims, that is based on rule mining. Therefore, we develop an automatic way for the extraction of implicit address patterns in a data set of addresses or, to put it in another way, these addresses form a side channel revealing implicit address patterns. Using the gained patterns, we generate potential addresses for active probing (scanning). The fundamental idea behind this approach lies in the observation that administrators do not select random addresses when migrating their services to IPv6, but rather rely on patterns. Our approach finds significantly more addresses than manual pattern-based approaches; however, it is limited by random-appearing addresses generated by the IPv6 Privacy Extension.

We prove that the IPv6 Privacy Extension's randomness is not sufficient. An adversary is able to predict a host's future temporary interface identifiers[1] that are generated by the IPv6 Privacy Extension once the algorithm's internal state is known, and is further able to synchronize to this internal state by exploiting the victim's previous interface identifiers as a side channel. In turn, the adversary is able to perform address-based correlation of different transactions and infer (private) details about people's Internet behavior. The IPv6 Privacy Extension is generally held to be insusceptible against any kind of pattern exploitation, and thus also against address correlation. This aspect of our research not only disproves this belief, but also highlights that current IPv6 address formats do not protect user privacy on a sufficient scale.

1.4 Structure of this Work

The remainder of this thesis is structured as follows. In Chapter 2, we define underlying terminology with respect to secret communication. Part I investigates such patterns of communications in cloud computing: Chapter 3 contains a survey of available approaches of secret communication in cloud computing and a classification thereof. Chapter 4 investigates the role of firewalls in *IaaS* clouds, and draws conclusions on their functionality as well as location in clouds using side channels. Finally, Chapter 5 shows how the protection measure of rate limiting becomes an actual attack vector; we develop a side channel and a denial-of-service attack exploiting these rate limits.

Part II addresses secret communication with respect to IPv6. In Chapter 6, we provide background on this protocol, and a survey on the latter's security and privacy short-comings. Further, we systematize this short-comings and allot appropriate countermeasures. In Chapter 7, we investigate IPv6 reconnaissance that exploits implicit

[1] The IPv6 Privacy Extension aims to protect privacy by regularly changing the address, and defines an algorithm for the generation of interface identifiers that are combined with the advertised network prefix to form temporary IPv6 addresses.

patterns from a data set of addresses. The latter approach comes to a grief with the IPv6 Privacy Extension that pretends to generate random addresses that are totally unrelated to each other. In Chapter 8, we overcome this belief and prove that an adversary is able to predict all future identifiers of a host once she has synchronized to the algorithm's internal state; synchronization is feasible by exploiting former interface identifiers as a side channel.

In Chapter 9, we conclude on both aspects of this work and provide an outlook on future work. The research that is presented within this thesis has further been published in the following peer-reviewed conferences and workshops:

- J. Ullrich and E. Weippl, "The Beauty or The Beast? Attacking Rate Limits of the Xen Hypervisor," in European Symposium On Research in Computer Security (ESORICS), 2016. (Chapter 5)
- J. Ullrich, T. Zseby, J. Fabini, and E. Weippl, "Secret Communication in Clouds: A Survey," IEEE Communications Surveys & Tutorials, 2016. (under Submission after Major Revision) (Chapter 2 and 3)
- J. Ullrich, J. Cropper, P. Frühwirt, and E. Weippl, "The Role and Security of Firewalls in Cyber-Physical Cloud Computing," EURASIP Journal on Information Security, 2016. (Extended Version of the Conference Paper) (Chapter 4)
- J. Ullrich and E. Weippl, "Privacy is Not an Option: Attacking the IPv6 Privacy Extension," in International Symposium on Research in Attacks, Intrusions, and Defenses (RAID), 2015. (Chapter 8)
- J. Ullrich, P. Kieseberg, K. Krombholz, and E. Weippl, "On Reconnaissance with IPv6: A Pattern-Based Scanning Approach," in International Conference on Availability, Reliability and Security (ARES), 2015. (Chapter 7)
- J. Cropper, J. Ullrich, P. Frühwirt, and E. Weippl, "The Role and Security of Firewalls in IaaS Cloud Computing," in International Conference on Availability, Reliability and Security (ARES), 2015. (Chapter 4)
- J. Ullrich, K. Krombholz, H. Hobel, A. Dabrowski, and E. Weippl, "IPv6 Security: Attacks and Countermeasures in a Nutshell," in USENIX Workshop on Offensive Technologies (WOOT), 2014. (Chapter 6)

CHAPTER 2

Background

This chapter provides background for the remainder of this thesis. In Section 2.1, we define fundamental terms of cloud computing; discuss its key technologies and introduce the general challenge of cloud computing security. Further, we describe participating stakeholders. In a second step, we focus on patterns of secret communication in Section 2.2; in particular, we define covert channels, side channels as well as obfuscation and compare them with each other. Then, we describe the potential of secret communication in cloud environments in Section 2.3. Finally, we discuss application scenarios, both typically considered benign and malicious, in which such communication channels are used for the sake of secrecy and relate them to covert channels, side channels as well as obfuscation, see Section 2.4.

2.1 Cloud Computing At A Glance

In this section, we define cloud computing and present service models that assign responsibility to different stakeholders in manifold ways. These definitions, however, are rather vague from a technical point of view. Thus, we review the key technologies that lay the foundation for cloud computing, and highlight security challenges in cloud computing despite its lacking novel technologies. Finally, we shed light on different roles and entities in cloud computing.

2.1.1 Defining the Cloud and Cloud Service Models

The term cloud computing emerged in the late 2000s [3], and rather describes a mode of operation that combines several other technologies than a novel technology itself [4]. Thus, definitions are diverse [5], and rather comprehensive. The definition from the *National Institute of Standards and Technology (NIST)* appears to be the most popular, and sees cloud computing as *"a model for enabling ubiquitous, convenient, on demand*

network access to a shared pool of configurable computing resources (e. g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [6].

Further, NIST defines five essential cloud characteristics:

- Cloud customers have to provision resources in an "*on-demand self-service*" way leading to 24/7 service "*without requiring any human intervention*" [6].
- As the services are provisioned over the network, "*broad network access*" [6] is an inevitable pre-requisite and a lack of networking leads to full outage, i. e., the Internet is a critical infrastructure.
- "*Resource pooling*" [6], frequently also called multi-tenancy, means sharing of resources like storage, processing or network bandwidth with other customers.
- Customers do not only provision resources on their own, they can also easily scale them leading to "*rapid elasticity*" [6]. Customer needs are rather small in comparison to the size of professional data centers and thus the latter "*often appear to be unlimited*" [6].
- *Metering* is used for resource control and optimization of the providers' infrastructure as well as billing.

Real-world cloud computing encompasses a broad range of applications: social networking, tax and health applications, storage solutions, virtual machine rentals, and many more. Generally, these services are classified into the three service models *SaaS*, *PaaS* and *IaaS* [6]. We added *StaaS* as a fourth service model due to its characteristics. The service models assign responsibilities to customer and cloud provider in different combinations.

- In *Software as a Service (SaaS)*, the customer uses "*provider's applications running on a cloud infrastructure*" [6] by means of a web browser or a certain client. Examples are *Twitter*[1] and *Google Cloud Messaging*[2].
- *Platform as a Service (PaaS)* clouds provide a platform including programming languages, libraries, etc. to run "*consumer-created or acquired applications*" [6]. An example is *Google App Engine*[3].
- In *Infrastructure as a Service (IaaS)*, the cloud provides resources to the customers. The latter are able to "*run arbitrary software, which can include operating systems and applications*" [6] on this resources. Examples are *Amazon Elastic Compute Cloud (EC2)*[4] and *Google Compute Engine*[5].

[1] www.twitter.com
[2] developers.google.com/cloud-messaging/
[3] cloud.google.com/appengine/
[4] aws.amazon.com/de/ec2/
[5] cloud.google.com/compute/

- *Storage as a Service (StaaS)* offers synchronization into the cloud and a possibility for storing backups. While some see it as a specialization of *IaaS* due to storage provision [7], its aspect of offering a certain client for easy up- and downloading tends to be *SaaS*. For the purpose of this thesis, we use *StaaS* as a service model sui generis. A real-world example is *Dropbox*[6].

2.1.2 Key Technologies and the Challenge of Cloud Security

From a technology perspective, the definition of cloud computing appears vague. Following the idea of cloud computing as an operational model and a confluence of existing technologies, [7] highlights the following economic, societal and technological shifts as relevant for the establishment of clouds.

- *The spread of devices:* Smart phones, tablets, or laptops allow access to clouds in a variety of situations and places.
- *The trend towards browser interfaces or thin clients:* Shifting heavy computing to the server (and further into the cloud) requires less performance of the access device.
- *The provisioning of services over the network:* Fast broadband access is inevitable therefore and a lack of networking leads to service outage.
- *The dropping of hardware prices:* This paved the way for data centers consisting of inexpensive and inter-connected servers and storage devices.
- *The sharing of data centers:* The data centers are rarely used by a single organisation, but shared among different customers. Virtualization technologies abstract computer resources to *"provide a dedicated resource view for customers"* [7].
- *The development of Application Programming Interfaces (APIs):* This allows "*self-provisioning and programmatic control*" [7].

Still, one might ask for the reasons of seemingly excessive engagement with the cloud. Cloud computing is certainly a tremendous economic success story: This year's global market size is estimated to $96.98 billions, and its annual growth rate to 9.14 % [8]. Market leader *Amazon* alone generates an annual $6 billion revenue [9]. By 2014, 69 % of enterprises had an application or infrastructure in the cloud, another 18 % planned to do so within the following year [10]. Beyond, cloud services for end users have large user bases. *Dropbox* claims to have 400 million registered users by 2015[7], *Evernote* more than 100 millions[8] and *Spotify* 75 millions[9]. Moving into the cloud is primarily an economic

[6] www.dropbox.com
[7] http://techcrunch.com/2015/06/24/dropbox-hits-400-million-registered-users/
[8] https://blog.evernote.com/blog/2014/05/13/evernote-reaches-100-million-users/
[9] https://press.spotify.com/at/information/

decision: Cloud computing does not require upfront capital investment for infrastructure, and typically provides (almost) immediate access. Flexibility allows to start with little resources, and increase later if needed. It unfolds its full effect with the creation of new services based on existing ones, and is a key enabler of many novel services due to low economic risk. Low entry costs further enable small companies access to computing facilities that were previously only accessible by large players [7, 11].

Despite lacking of novel technology, there is a decent technical aspect in the deployment of cloud computing: The combination of several existent technologies into a new operation model implies that they are now used in an environment with different characteristics. As a consequence, several (formerly) basic assumptions might be broken and partial redesign required. Among them are:

- Traditional enterprise architectures follow a zoned approach. Internal is considered as benign, external as potentially malicious and thus perimeter protection is applied. Potentially sensitive data is now travelling the Internet on its way to the cloud or back to the customer. Even the cloud-internal traffic is not necessarily benign as the cloud-internal network is shared among (potentially malicious) customers. A traditional zoned approach becomes infeasible.
- Moving to the cloud changes infrastructure from a white box to a black box. An operator knows the details of his infrastructure and thus its advantages as well as its disadvantages, or is at least able to find out. Cloud providers however see their internal structure as their company secret, and disclose only a limited view thereof. This means that the customers are not fully aware of the provider's intentions and vice versa; and their combination might introduce risks that neither of the two is aware of.
- The customer is not solely unaware of the infrastructure, but also dependent on the cloud provider's offer. While a minor change in configuration might be easy in a self-operated infrastructure, it is almost impossible to do so in the cloud. In the worst case, a provider has to be replaced by another. However, migration is another challenge as there are barely any standards.

Such broken assumption due to a changed use of technology might negatively impact non-functional requirements. Among them is security, and it is thus of utmost importance to address this issue. Indeed, concerns of security is considered the major obstacle by enterprises when moving to the cloud [10].

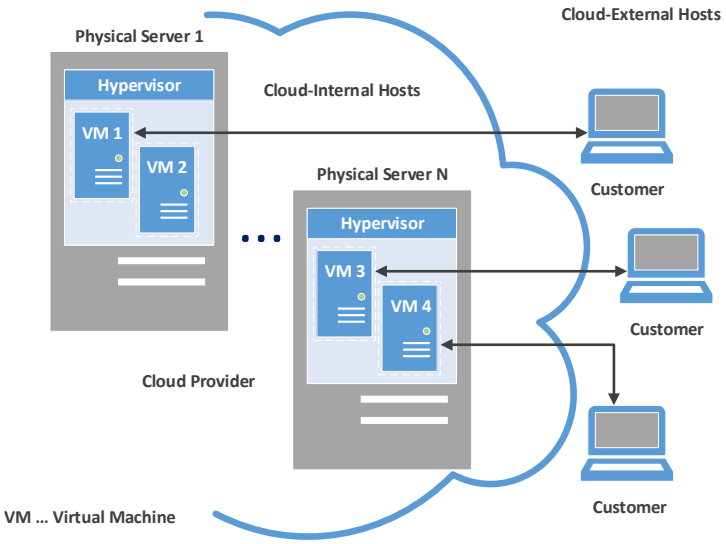


Figure 2.1: Roles in Cloud Computing

2.1.3 Roles and Entities in Cloud Computing

The definition of service models in cloud computing has already revealed two roles – the cloud provider and the customer. These and other roles are defined for the purpose of this thesis based on descriptions found in the literature [12, 13, 14, 3, 7, 15, 6], and are also depicted in Figure 2.1.

- A *cloud provider* develops, operates and offers cloud services. Others are able to access these services via the Internet, and use them for their own purposes. The offered services follow one of the cloud service models. Major cloud providers not solely offer a single service, but a variety thereof and also span among more than a single service model.
- A *cloud customer* is somebody who accesses and uses a cloud service; cloud customers are also referred to as *cloud users* or *consumers*. Typically, they pay the cloud provider for the service. Customers might be enterprises or private individuals.
- *Virtual machines* pretend to be real computers, but are virtual representations thereof. They are a consequence of virtualization technology that enables different cloud customers to utilize the same hardware simultaneously.
- The *physical server* is the hardware that is abstracted by means of virtualization. Atop runs a *hypervisor* (or *virtual machine monitor*). This is a software that

manages virtual machines. Multiple virtual machines might reside on the same physical server and share its hardware.

- A *neighbor* is a virtual machine that resides on the same physical server as another virtual machine, i. e., they share a server's hardware resources. Neighbors are *co-resident* to each other; the situation is referred to as *co-residency*.

- A *node* is an entity that is connected to the network – independently whether its role is data redistribution or being a data end point. The term includes virtual machines, but also non-virtualized (ordinary) computers, e. g., adversaries that reside outside the cloud. We distinguish *cloud-based nodes* from *cloud-external nodes*. Cloud-based nodes reside in a cloud, while external nodes are outside and reside at an arbitrary place on the Internet. Customers access the cloud typically by means of external nodes, e. g., laptops, smart phones, etc.

2.2 Secret Communication

Data transmission over networks typically follows a regular pattern: A sender wants to transmit a message to a receiver, and thus embeds data in packets. Packets do not only hold a payload but also a header containing information for delivery. The receiver extracts the data from the packets and processes the data according to its needs. For protection, transmitted data might be encrypted so that only the intended receiver can access the actual content. Still, it remains clear who intends to communicate with whom, who is the sender/receiver, what remains control data for purposes of delivery and what is the delivered data (albeit an outsider might not access it due to encryption).

This pattern is contrasted by the concept of secret communication that deviates from the classic pattern insofar as communication takes a more clandestine way. Clandestineness manifests through stakeholder's nescience of certain aspects of communication or even its mere existence. We identified three means of such secret communication, namely covert channels, side channels and obfuscation, that are defined and discussed in this section.

2.2.1 Covert Channels

The *United States Department of Defense* standard 5200.28-STD defines a covert channel as "*any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy*" [2]. Literature highlights further aspects: A covert channel "*exploits a shared resource*" [16], and the used channel "*is not designed to be a communication mechanism*" [16] or "*contrary to design*" [17]. Typically it is of "*malicious or unwanted nature*" [1] and "*can be used to leak information*" [18].

If communication partners want to prevent unauthorized parties from eavesdropping on transmitted data, end-to-end encryption is a practical countermeasure. Covert channels go beyond insofar as their intention is concealment of the communication's mere existence from third parties [1, 19], and are applied in case an observer should not even know

that communication is on-going. They exploit senders' degrees of freedom [20] that are also accessible to receivers. Traditional network-based covert channels use for example unused or vaguely defined bits; values that are chosen by senders without having strict criteria (e. g., identification numbers, fragmentation offsets or hop counts); values that are typically not parsed by receivers (e. g., timestamps); checksums in case the payload can be modified in a way to correspond to the checksum value; and timing as IP-based traffic shows indeterministic behavior [1].

Observers are usually unaware of the fact that some of these characteristics can be used to communicate. Nevertheless, even if the method is known, the content of the communication should remain confidential by using encryption and ideally should not be distinguishable from typical (e.g. random) values used in such fields. So the secrecy of the communication should additionally lie in the knowledge of a secret key and not solely in the unawareness of external observers of potential communication channels [21]. Therefore classical cryptographic methods should be applied before a message is encoded in a covert channel.

Cryptography differentiates between symmetric and asymmetric algorithms [22]. Symmetric cryptography uses a single secret key that is used by the sender to encrypt, and by the receiver to decrypt the message accordingly. In contrast, asymmetric cryptography operates with two keys – a private key, and a public key per communication partner. While the first is kept private by all means, the public key is made available to the general public. A sender encrypts a message with the public key, and only the receiver is able to regain the message by decryption with the private key. Asymmetric approaches rely on trapdoor functions, i. e., functions that are computationally inexpensive in one direction but only solvable in the other direction if additional knowledge (private key) is known.

The differences between symmetric and asymmetric cryptography have an impact on covert channel application scenarios. An autocratic regime might force communication partner A within its jurisdiction to reveal the secret key. In a symmetric key setting the regime is able to decrypt all traffic of both communication partners A and B, even if communication partner B is outside its area of influence. If asymmetric cryptography is used, and the regime gets hold of the private key of A (communication partner in its jurisdiction), but does not know the private key of the communication partner outside B, it can decrypt incoming messages to A (encrypted with the public key of A) but has no possibility to decrypt outgoing traffic from A to B (encrypted with the public key of B). Nevertheless, a regime in possession of a private key of A is able to sign messages and pretend to be A. Asymmetric encryption appears further suitable for encrypting unidirectional data extraction, e. g., from several compromised hosts, via covert channels to one data collector. Whenever data is ready to be delivered to a data collector it is encrypted with the same public key and the reporting hosts do not need a key pair on their own. Beyond, an administrator discovering compromization of his own host (and the public key used for data extraction) is not able to decrypt messages that have been sent from his host, neither to decrypt communication of hosts that are still compromised. However, asymmetric cryptography bears the drawback of being computationally more

expensive than symmetric approaches and thus might be more conspicuous for the owner of a compromised host. Furthermore, if signing or mutual authentication is needed, the reporting hosts need to create keypairs.

Authenticated and eavesdropping-secure key exchange is a vital part of symmetric cryptography, and a major challenge. The Diffie-Hellman protocol is nowadays the major approach to overcome this issue on a non-tap-proof communication link, and is based on the concept of finite cyclic groups [23]. Diffie-Hellman (and also other key exchange protocols like Needham-Schroeder [24]) require a two-way data exchange, i. e., a bidirectional channel – an assumption that does not necessarily hold for covert channels. In unidirectional covert channel settings, the communication partners might perform an out-of-band key exchange. For example, Diffie-Hellman could be performed over regular networking instead of the covert channel. Such an unexpected key exchange might, however, appear illegitimate to an observer, and might be a hint for covert communication. This especially holds for countries prohibiting cryptography, see application scenario *Superficial Compliance to Cryptography Laws* in Section 2.4, as a Diffie-Hellman key exchange is an unambiguous characteristic of cryptography.

With respect to covert channels in cloud computing we further have to highlight two distinct aspects: First, a shared resource in the context of covert channels as highlighted in [16] does not necessarily mean resource sharing in the sense of cloud computing. A shared resource of a covert channel is one that both sender and receiver can access, i. e., read and/or write. However, this resource does not necessarily have to be physically shared in the cloud computing's sense of resource-sharing. I. e., sender and receiver need not be co-resident virtual machines (neighbors) and compete for the resource, as packet header fields can be used as shared resource, too.

Second, an observer that is looking for suspicious traffic has to reside on the communication path between sender and receiver. For determination of the non-existence of suspicious traffic, the observer has to control all alternative paths that the communication might take. Depending on the observer's power this appears to be a minor or major challenge. In cloud computing however such a potential observer is always present with the cloud provider. In dependence of the service model, the provider controls the underlying infrastructure including computing, storage and network facilities, virtualization, the operating system, the platform and/or the applications and traffic has to pass in any case.

2.2.2 Side Channels

Side channels root in the field of cryptographic engineering and *"exploit characteristic information extracted from the implementation of the cryptographic primitives and protocols. This characteristic information can be extracted from timing, power consumption, or electromagnetic radiation features"* [25]. The classic way of exploiting a side channel is the extraction of a secret key. A recent prominent example is the extraction of the

RSA private key from noise that emerges from a laptop during the performance of cryptographic algorithms [26].

But even beyond breaking cryptography, side channels are prevalent whenever an implementation's behavior reveals systems internals that should be kept secret. A well-known side channel is operating system detection (fingerprinting): Although protocols like IP are standardized [27], stack implementations show (subtle) differences in behavior and allow to determine a host's operating system. For example, operating systems initialize the IP *Time to Live (TTL)* field with different values. Adding the measured hop distance to the received TTL allows to draw conclusions on the remote host's operating system [28].

In comparison to the classic communication pattern, side channels are a side effect of the system architecture or implemented algorithms and unintended by the sender. Such channels can leak (confidential) information, and transmitted data is neither encrypted nor otherwise protected due to the channel's unplanned nature.

Cloud computing adds new aspects to such channels' application: Cloud providers conceal their infrastructure and configuration following a security-by-obscurity concept. In-depth and verifiable knowledge on cloud internals remains widely inaccessible for customers, and this black-box approach impedes checks on the provider's compliance with service level agreements. In such scenarios, side channels can be used as a source of information gathering that allow plausibility checks. For example, cloud providers offer dedicated instances[10], i. e., physical servers that just run instances of a single customer to mitigate the threat of co-residency. A customer might use a side channel checking for co-residency as a defensive tool and verify whether there is a stranger's virtual machine on the same physical server [29]. This example also emphasizes a modified standing of side channels. In cloud computing, side channels are not exclusively means of attacking and thus evil, but also serve the benign purposes, e. g., protection of customers against a typically far more powerful provider that might silently disobey service level agreements. This modification goes hand in hand with a change in perspective from building systems as in traditional engineering to discovering phenomena in a way that is comparable to the natural sciences.

2.2.3 Obfuscation

Obfuscation aims at anonymous communication by concealment of sender/receiver or hindering their correlation by third-parties [1]. Communication is not fully covert: Observers are generally aware that nodes are participating in communication and use a certain method of obfuscation [30]. However, they are not able to correlate sender and receiver and/or identify them. This assumption holds even in case the observer joins this technique of obfuscation. Observers might further be unable to read the transmitted content due to encryption.

The most prominent example is *Tor* [30]. Based on onion-routing, packets are detoured over relays that are provided by volunteers. Relays decrypt packets to infer the next

[10] `https://aws.amazon.com/de/blogs/aws/amazon-ec2-dedicated-instances/`

	Covert Channel	Side Channel	Obfuscation
Intention	yes	no	yes
Hiding Technique	resource unintended for communication	none	large set of nodes or users
Intermediate nodes	optional	optional	required
Content Protection	encryption	none	encryption
Suspicion Level	medium	low	high

Table 2.1: Classification of Secret Communication Variants

address and deliver to the next hop. Although aware of the communication, a third party cannot compromise anonymity by simple actions like the provision of single relays. Obfuscation in general relies on confusion due to the Internet's extent, requires high traffic and many users/participating nodes for successful concealment.

With respect to the classic communication pattern that has been presented at the beginning of this section, obfuscated communication is intended, as with covert channels. An observer however is aware that communication is going on, but cannot find out who is communicating with whom. An observer is also unable to decode transmitted data. A pattern of obfuscation is the involvement of intermediate nodes: Sender and receiver seem to maintain a connection to this intermediate, but as a number of nodes do the same, correlation becomes more difficult.

Clouds appear to be a sound substrate for obfuscation - less due to technology than their impact on economy and society. First, obfuscation depends crucially on the number of participants, and cloud services typically have a large user base. As networking is a prerequisite of cloud computing many participants imply much traffic that can be used to hide. Second, this traffic appears in-dubious as the majority of people use the cloud service as intended. Finally, services with a vast user base are unlikely to be blocked. For example, countries applying Internet censorship refrain from blocking clouds as they fear negative impact on commerce and society. This enables activities to counter censorship by moving content to the cloud or using the cloud as a relay [31, 32].

2.2.4 Comparison

We defined three means of secret communication, namely covert channels, side channels and obfuscation, and discussed them with respect to classic communication patterns and their application in clouds. Table 2.1 summarizes differences between these three (general) kinds of communication on the basis of five discovered major characteristics:

- *Intention for communication* describes whether the data is transmitted by the

sender on purpose. This is the case for communication via covert channels and obfuscation techniques, as the communication partners aim to exchange information. In contrast, side channels leak information unintentionally; the sender might not even know that it is transmitting data and provides sensitive information to third parties.

- *Hiding technique* describes the method that hides the exchanged information from potential observers: As side channels are unintended, they do not hide either. Covert channels hide by using a shared resource which is not intended for communication. Examples are CPUs or caches in a system, header fields or packet timing in network protocols. Obfuscation exploits a large set of nodes or users to hide.

- *Intermediate nodes* reside between sender and receiver on the communication path. Obfuscation requires intermediate nodes for concealment. For example, sender and receiver might maintain connections to the same intermediate node, but correlation of these parties is complicated as they are lost in the multitude of other connections. This intermediate node is used as a reflection point for information. An alternative is the redirection of traffic over a number of relays. Covert and side channel do not necessarily require an intermediate node. It is worth noting, however, that intermediate nodes might be detrimental to the quality of side and covert channels, for instance by rewriting packet headers or changing packet timing.

- *Content protection* refers to techniques that might be used to prevent others from accessing transmitted information. Covert channels and obfuscation might use encryption, side channels do not include any – again due to their unintended nature. Senders might attempt to close side channels when becoming aware of their existence.

- *Suspicion level* describes the degree to which an observer suspects that communication is taking place. Low means that an observer is unaware of the communication in general, medium means an observer may suspect communication, but cannot find details, and high refers to an observer being (quite) sure that there is communication but cannot find details, either.

With respect to the communication pattern that was presented at the begin of this section, side channels lack the intention to transmit data; covert channels lead an observer to believe in the non-existence of communication despite the latter might be able to access overt traffic; and obfuscation attempts to hide who is communicating with whom.

2.3 Potentials for Secret Communication

Distinct factors of cloud computing influence networking, and thus provide potential for the establishment of secret communication. These influence factors can be exploited to generate network traffic with certain characteristics like traffic amount, certain header

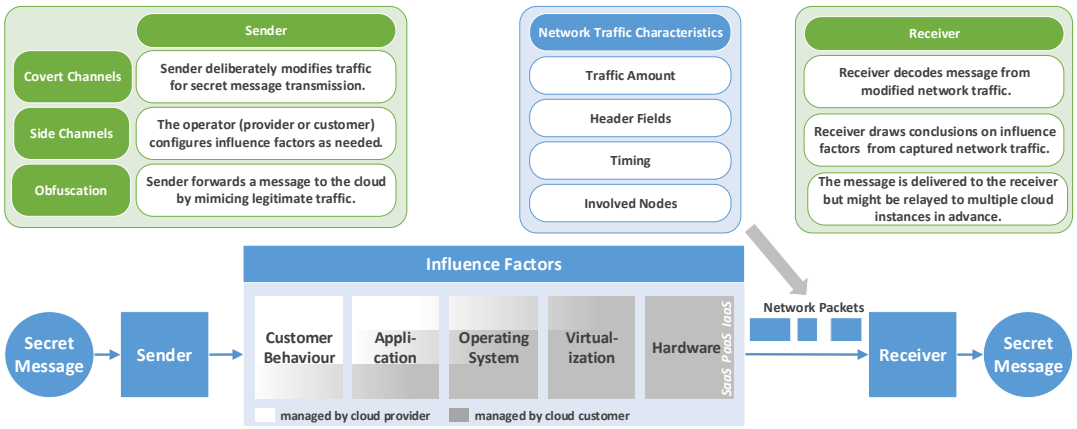


Figure 2.2: Potentials for Secret Communication

fields or packet timing as well as nodes involved in the communication. Exploiting its influence factors, a sender forms a traffic with certain characteristics in order to deliver the secret message; the receiver in turn interprets these characteristics, and infers this way the secret message. This process is illustrated in Figure 2.2.

Senders of covert channels aim to deliberately modify these influence factors, while senders of side channels configure them as needed in their environment. In both cases, receivers draw conclusion on the influence factors from the resulting network traffic's characteristics. An obfuscation's sender aims to mimic legitimate traffic and its characteristics.

In detail, we identified the following influence factors on networking. As they can impact traffic characteristics, they bear potential for secret communication in cloud computing.

- Cloud computing requires underlying physical *hardware* and infrastructure that is subject to physical limits. Consequently, cloud computing efforts are also limited.
- *Virtualization* has to partition hardware among virtual instances in a fair and efficient manner, but also has to protect instances from each other facing a natural trade-off.
- *Operating systems* are networking's pivot and support various protocols. Applications utilize them for delivering their messages[11].
- Cloud computing has led to plenty of novel *applications*. These applications provide new features, but also new weaknesses – both forming a substrate for secret communication.
- Cloud services attract large user populations; pervasive *customer behavior* has economic and societal impact bearing potential for secret communication.

[11] As an operating system's role does not differ in cloud computing from traditional use, we refrain from highlighting its impact on networking in this section.

Figure 2.2 depicts the influence factors as well as network traffic characteristics. Further, it highlights that the influence factors are managed either by the cloud provider or the cloud customers; distribution is however dependent on the service model. With management comes the power to influence networking, i. e., the service model defines secret communication's extent and the communication partners. We discuss in the following the influence factors' impact on networking, and sketch their potential exploitation for secret communication in cloud computing.

2.3.1 Underlying Physical Hardware and Infrastructure

Clouds ultimately rely on physical computing resources and are thus subject to their physical limitations; although cloud providers put effort in abstraction and concealment of their internal infrastructure, they cannot overcome physical principles. Hardware still has an impact on the service that is accessible by customers, and this becomes visible in manifold ways. For example, a network interface card cannot serve two or more virtual machines at the same time because it is able to send or receive just one packet at a time [33].

Neither are access times detached from their physical background. Responses from distant machines (in remote data centers) take longer than those from one close-by [34]. Round-trip times within a data center's segment or to a virtual machine on the same physical server are lower; and typically, there are also less intermediate hops [35]. Round-trip times might be artificially prolonged by intentionally delaying the response, but not speeded up. These general statements also hold for resources beyond networking. Drive redundancy reduces response times, and fetching information from a number of drives is faster than gaining the same from only one [36].

Clouds reuse known technologies. Providers aim to conceal their utilization or detailed configuration from customers. They frequently prohibit the Internet Control Message Protocol (ICMP) and thus standard tools like *ping* and *traceroute*. Networking however still works as expected and replacement of diagnosis tools by crafted, benign network probes is feasible. Development of such replacements is especially fruitful as knowledge and experience is existent from the pre-cloud era. This is highlighted at the example of the IPv4 *Time To Live* (TTL) field. The amount of intermediate hops can be extracted from the minimal TTL that is required without any need of ICMP messages [37].

In conclusion, layers of abstraction and concealment cannot hide all ground truths of the cloud and such limitations provide substrate for side channels that reveal internals of the cloud structure. However these aspects are less usable for covert channels as the underlying hardware cannot be easily modulated.

2.3.2 Virtualization: Utilization vs. Isolation

Virtualization means *"abstraction of physical hardware resources"* [38] and allows parallel execution of different operating systems on the same hardware. Virtualization uses hypervisors that control access to physical resources by intercepting requests of hosted

virtual machines and mediating them to the hardware [38]. It is a key enabler of multi-tenancy in clouds, and serves a twofold goal: First, it has to partition available hardware resources among tenants in an efficient manner to maximize hardware utilization. The better this goal is met, the higher the cloud provider's monetary revenue. Second, virtualization has to provide isolation for security [39, 40], and separate co-resident instances best possible from each other to prevent assaults. Due to resource costs, isolation is a natural antipode to maximum utilization and *"undermines the cloud's elasticity and business model"* [41].

Beyond, customers demand fair resource partitioning [42, 43]. This includes the provision of resources that customers are paying for; minor impact of neighbors' resource requests on one own's resources and the provision of minimum requirements. In contrast, cloud providers tend to resource over-subscription [44], and the sum of guaranteed resources exceeds the actually available hardware resources. This relies on the assumption that users do not request their full resource share at the same time, and is known from the power grid but may cause severe problem in case the previous assumption is intentionally falsified [45].

This leads to the following conclusion: Cloud providers have little motivation for better isolation as it would reduce monetary revenue. Quite the contrary, they over-subscribe resources to increase revenue and neighbors are likely to influence each other providing an ideal ecosystem for covert and side channels [44]. The potential is emphasized by the following two scenarios.

The hypervisor Xen [46] processes packets in a round-robbing manner in regular intervals; incoming packets are prioritized over outgoing. Buffers on the path are limited by size [33]. Packets might be delayed due to a heavy networking neighbor. Heavy networking of an instance causes fluctuations in neighbors' data transmission due to delayed packets [47], or packets might even be dropped due to a full buffer. A neighbor might intentionally exploit this coherence.

Increased demand for a certain resource, may not only influence neighbors with demand for the same resource. We highlight this by an example of networking and CPU usage: Instance A and B experience network load and share the NIC equally. If instance A experiences additional CPU load that requires the full time given for the CPU, B has increased chance of networking. The reason is twofold: First, A requires obviously less of the resource (networking). Second, the resource demand might shift in time and B is provided with the resource for a longer continuous period [48]. Thus, there are also inferences among different resources that might be exploited.

2.3.3 Novel Applications in Clouds

Cloud computing reduces start-up costs as no physical equipment anticipating future needs has to be bought. At the same time, flexibility allows easy scale-up and down with immediate needs [49]. Being of comparably low risk, the cloud is an excellent substrate for start-up companies providing all types of novel applications to the market;

enhanced by providers' market places where native as well as third-party applications are offered [50]. A number of now popular cloud services started in the data centers of large cloud providers, and are still there, e. g., *Spotify* and *Dropbox* in the *Amazon* cloud.

Cloud applications follow approaches that were unknown before or reuse known technologies in a different environment. In addition, they connect users that are unrelated to each other to a single central service albeit the application's purpose does not necessarily require this radial topology. Cloud services are accessible via the network and this traffic provides potentials for secret communication. For obfuscation, the cloud may serve as a reflection point. The traffic may alternatively serve as overt channel for the covert channel [1], or reveal internals and thus be a side channel. Examples of such novelties are provided in the following paragraph.

Data deduplication associates users that are unknown to each other due to storing the same file in their personal cloud-synchronized folder in order to optimize storage capacity through the elimination of redundancy [51, 52]. *Push Notification Services* release developers from reliably delivering updates for mobile applications, and allow forwarding of messages by means of an ID making the services a reflection point with a lot of users [53]. The market place bundles services related to a certain cloud on one place. The services are not only from the cloud provider, but also from third parties. While it might *"provide the customer with peace of mind by knowing that all purchases from the vendor's marketplace will integrate [...] smoothly"* [50], the provider does not check these third party offers in detail fostering all types of misuse and slackness [54].

2.3.4 Customer Behavior and Cloud Population

Clouds attract a high number of users. These users in turn maintain connections to cloud services causing massive network traffic and giving the providers significant market power. While this aspect is more societal than technological, it has particular impact on secret communication for three reasons.

- Maintained connections to these clouds are common and do not appear suspicious. A network administrator might not react to *Twitter* traffic as he might know that some users of his network are actively using this *SaaS* service. Even in case of additionally caused *Twitter* traffic, it might appear harmless and users might believe that this traffic origins from their own account.
- The pervasiveness of cloud traffic decreases the chance of being filtered or censored because users might recognize and condemn the intervention. For example, China refrained a long time from blocking large cloud providers for business reasons, and censored content could spread by moving it into the cloud [31, 32]
- Users maintain connections to the same cloud services albeit the service does not necessarily need to connect users. Thus, these services might be used as reflection point, and indirectly connect users that appear unrelated to each other in the first.

	Covert Channel	*Side Channel*	*Obfuscation*
Industrial Espionage	✓	✓	
Whistleblowing	✓		✓
Censorship Evasion			✓
Exchange of Illegal or Regime-Critical Content	✓		✓
Superficial Compliance to Cryptography Laws	✓		
Gain of Transmission Capacity	✓		
Compliance Checking		✓	
Reconnaissance		✓	
Data extraction from Compromized Hosts	✓		
Malware Communication	✓		✓

Table 2.2: Application Scenarios wrt. Types of Secret Communication

The high number of users is supported by the fact that basically everybody is able to join cloud services with almost no barriers. A lot of services are free of charge, others provide opening offers. However, an adversary is able to subscribe in the same way as an ordinary user does, and to investigate the cloud services in detail for potentials of secret communication. While providers might have a chance to check the users' identity, other users do not and have to trust the provider. Thus, there is no classical perimeter protection that separates the "trusted" inside from the "malicious" outside [39] anymore, and formerly insider attacks become outsider attacks [41].

2.4 Secret Communication Scenarios

The following list shows typical classes of information for which the communication partners have an incentive to hide from potential observers. Table 2.2 provides an overview on these scenarios with respect to applicable types of secret communication.

Industrial Espionage: Intellectual property is an important asset of companies and measures are taken to prevent its disclosure to the public or competitors. Due to measures

preventing such content from leaving the company, an inside spy has to use covert channels to evade this barrier; see, e. g., [55]. Conversely, competitors might also spy from outside and gain information from unintended sources of the victim via side channels.

Whistleblowing: Whistleblowers disclose content "*about non-trivial illegality [...] under the control of that organization, to an external entity having potential to rectify the wrongdoing*" [56]. Whistleblowers can use covert channels to secretly transmit information or use obfuscation techniques to conceal their identity.

The scenario is comparable to inside industrial espionage, if the whistleblower reports from inside the organization with protection against leakage. The whistleblower then might use a covert channel. If the whistleblower reports from outside the organization, it suffices to stay anonymous by means of obfuscation.

Censorship Evasion: Some countries apply Internet censorship and block access to certain content [57, 58]. However, obfuscation might redirect traffic over other nodes to evade censorship. Covert channels cannot be used because direct communication (and therefore overt traffic) between the two nodes is impossible; however covert channels can be piggybacked on obfuscated traffic.

Exchange of Illegal or Regime-Critical Content: The Internet serves as a channel for illegal content, e. g., trading of drugs [59] or child pornography [60], and communication partners aim to evade detection. They might apply two approaches that protect them in different ways: Obfuscation protects from the identification of individual perpetrators, although law enforcement, e. g., might order an inquiry against person or persons unknown after seeing the content. With covert channels, even the existence of the transmission is unknown. In non-democratic states, regime-critical content may be penalized and considered *illegal* in the context of local jurisdiction. Thus, we include regime criticism here.

Superficial Compliance to Cryptography Laws: Countries may restrict or prohibit the use of encryption [61], and the use of cryptographic protocols might lead to governmental punishment. If any kind of application of cryptography is penalized, encrypted messages have to be hidden from governmental observers in a covert channel to superficially fulfill the law.

Gain of Transmission Capacity: Covert channels use channels that are unintended for communication, and thus increase the total transmission capacity. More data is transmitted without paying for, and is of interest in case of high transmission costs, e. g., Internet taxes per gigabyte [62]. Obfuscation typically causes overhead in comparison to direct communication between the end nodes, and therefore cannot be used to increase transmission capacity.

Compliance Checking: Service-level agreements are negotiated between cloud providers and customers, but are frequently standardized due to market imbalance. The provider's economic power is typically far higher than the customer's, and checking compliance of the provider by the latter is difficult [63]. Customers may use side channels to check whether measured results are plausible considering the terms of contract.

Reconnaissance: An adversary might aim to discover and get information about a (not yet compromised) victim to tailor the succeeding attack, or place his own virtual machine on the same physical server for a cross-VM attack [64]. The more information gathered, the higher the chance of a successful attack. Therefore, an adversary may exploit side channels as they transmit information that the victim does not intend to disclose.

Data Extraction from Compromised Host: An adversary may aim to leak secret information of a compromised system, e. g., a secret key, without alarming the operators. She might use a covert channel as this is the most secure way of preventing an alarm and have ongoing access to the systems as the operator might otherwise change the key.

Malware Communication: Botnets are networks of nodes that are infected by malware and coordinated by *command and control* structures. These nodes "*contact a command and control (C&C) server to receive instructions or updates*" [65]. Botnet operators aim to evade discovery, and thus conceal their traffic and are moving their infrastructure (partly) into the cloud [66]. Depending on the extent of disguise, they choose obfuscation or covert channels. A botnet that aims to gain control over as many nodes as possible, e. g., for later denial-of-service attacks, might prefer obfuscation [67] to get best cost-benefit ratio and takes in exchange removal of malware from certain nodes into account. Alternatively, covert channels might be preferred in cases of higher demands on concealment, e. g., in targeted attacks or worms.

Part I

Secret Communication in Cloud Computing

In this part of the thesis, we focus on the specific problem of secret communication in cloud environments. Chapter 3 provides a survey on known approaches of secret communication in cloud computing; we classify the latter, and highlight that most side channels can be further developed into covert channels. Up to now, channels are assessed by their capacity; however, we conclude that the development of full attack paths including all steps from channel development over information extraction to exploitation appears more promising in order to assess (and understand) secret communication's impact on security.

In consequence, the remaining chapters develop such scenarios for cloud computing. Chapter 4 develops a test tool for firewalls, and investigates deployed firewalls at major IaaS cloud providers. Our results shed light on aspects that typically remain hidden from the average customers, and enables the latter to take (more) informed decisions with respect to security, but also performance or compliance. Chapter 5 shows how a side channel revealing all configuration parameters of Xen's rate limit functionality can be turned into a denial-of-service attack exploiting the same mechanisms. This attack causes up to 88.3 percent of packet loss or up to 13.8 seconds of delay in benign connections.

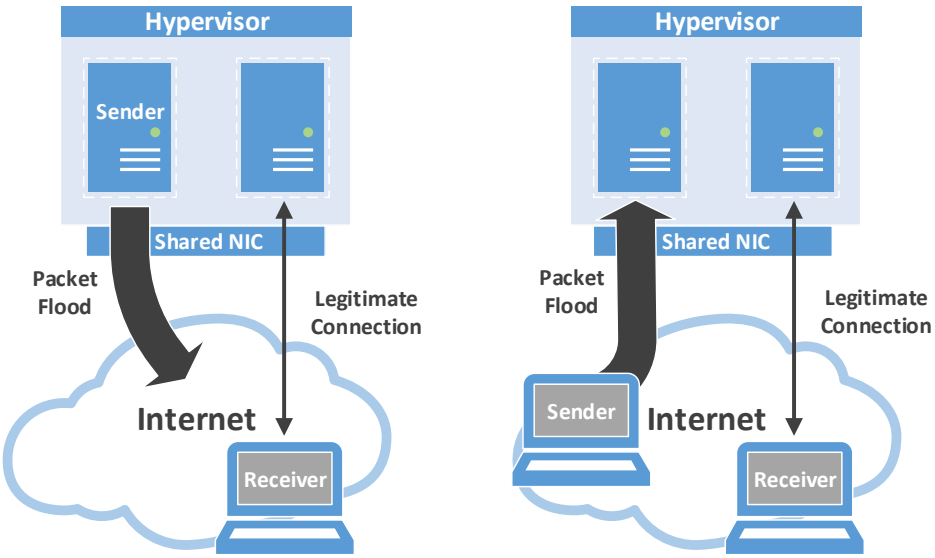
CHAPTER 3

A Survey on Secret Communication in Clouds

In this chapter, we describe and classify available approaches for network-based secret communication in cloud computing. We group known methods into (1) covert channels (Section 3.1), (2) side channels (Section 3.2) and (3) obfuscation (Section 3.3) according to the definitions in Section 2.2. As a number of secret communication channels rely on specific, widely unknown cloud-inherent mechanisms, we explain them directly in conjunction with the respective secret communication to ensure readability and compare different approaches with each other. Each type of secret communication is followed by a discussion. Finally, we added a unique identifier consisting of a letter – (c)overt channel, (s)ide channel and (o)bfuscation – in combination with a number to identify every means of secret communication in the following sections. Their identifiers are enclosed in brackets. Their classification is found in Section 3.4; our findings are discussed in Section 3.5.

3.1 Approaches for Covert Channels

In the following paragraphs, we describe covert channels in clouds. In a first step, we discuss covert channel that arise from cloud-immanent physical resource sharing, in particular sharing of NICs (*Resource Sharing Covert Channels*) before highlighting channels that exploit deduplication (*Deduplication Covert Channels*), a technology for storage optimization. Finally, we discuss channels that are specific to (novel) cloud applications.



(a) Cloud-based sender and external receiver (b) Cloud-external sender and receiver

Figure 3.1: Covert Channels using Packet Flooding

3.1.1 Resource Sharing Covert Channels

Resource sharing is cloud-immanent, and provides substrate for covert channels. This holds for central processing units (CPUs) [68], level 2 (L2) caches [69] and also for shared networking capabilities. Multiple virtual machines share various network interface cards (NICs) of a physical machine, and the number of NICs limits the maximum amount of sent/received packets. Virtual machines' network packets are scheduled and might have to wait before being forwarded to the network via one of the NICs due to high load resulting from given hardware limitations. As packet scheduling is a hypervisor task (or even outsourced to hardware assistance[1]), virtual machines are unaware of these latencies caused by other machines' traffic in general and would also not recognize extra-latencies deliberately caused by a neighbor machine. By deliberately causing high traffic a virtual machine might modulate packet timing of its neighbor in order to establish a covert channel.

Two distinct scenarios as depicted in Figure 3.1 are discussed in the literature [70, 71, 37]. They have in common that at least two virtual machines are located on the same physical machine, and share their networking resources[2]. An external node maintains a legitimate connection to one of these virtual machines, and triggers continuous data transmission,

[1] http://www.intel.com/content/www/us/en/network-adapters/virtualization.html

[2] We assume that a physical service has a single physical NIC for clarity of the explanations. Multiple shared NICs might reduce a virtual machine's impact on its neighbor and thus a channel's quality, but do not change a channel's basic principle.

e. g., by downloading again and again a file from a webserver. This external node is the covert channel receiver.

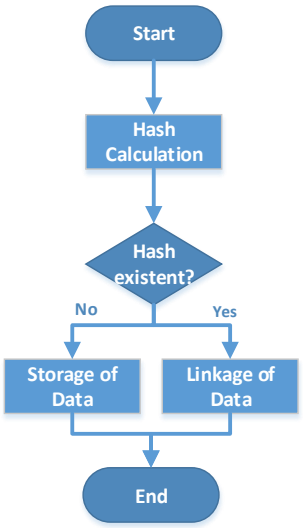
The first attack scenario, see Subfigure 3.1a, considers the co-resident virtual machine to be the covert channel's sender. This machine causes a packet flood and increases the latency of packets in the download process of the legitimate connection. The receiver measures the packet arrival rate looking for local extrema, which indicate the channel's symbols [70, 71] (c1). Recently, a stealthier version of this side channel has been proposed [72]; however, transmission capacity has decreased by 75 percent. In the alternative scenario of [37], an external sender influences the latencies of the legitimate connection by flooding the co-resident virtual machine, see Subfigure 3.1b (c2).

The sender's location has certain implications. (c1) requires the sender to rent a virtual machine, while in (c2) the sender might use an arbitrary co-resident neighbor that does not have to be in her control. Depending on its network bandwidth, the external sender might face difficulties in generating enough packets to successfully flood the cloud-based instance, and might have to synchronize with additional hosts for joint flooding [37]. The impact of packet floods on packet latency is dependent on the hypervisor's packet handling. For example, Xen prioritizes incoming packets over outgoing [33], and (c2) might thus outplay (c1).

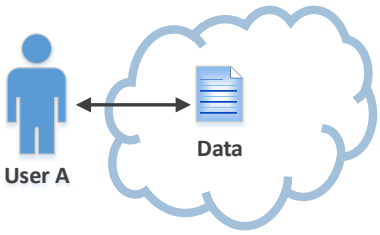
3.1.2 Deduplication Covert Channels

A variety of covert channels in cloud storage solutions arise from data deduplication – a technique to save storage capacity and in certain cases also networking. Instead of storing the same data multiple times, just one actual copy is maintained in the storage. Deduplication's working principle is depicted in Figure 3.2a: First, a hash is calculated over the data that should be stored, and by means of this hash the availability of this data in the storage is checked. Hash inequality implies its non-existance, and the data is added to the storage. In case of hash equality, just a link to the already existing file is generated.

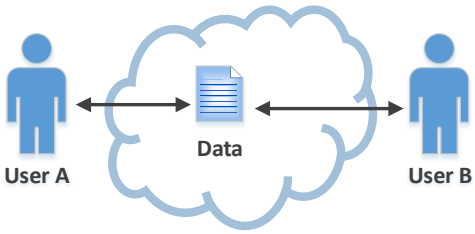
Different approaches of data deduplication are available. In the target-based approach data deduplication as described in Subfigure 3.2a is fully performed in the cloud storage leaving the external client unaware of any internal activities. In contrast, source-based deduplication splits the process among the stakeholders: The client calculates the hash and sends it to the cloud, actual data is supplied later if needed. This way networking can be reduced. Cross-user deduplication promises more resource savings by exploiting data redundancy within several user accounts. The resulting source-based cross-user deduplication then becomes a substrate for covert channels. The first user storing a certain data has to upload it leading to the situation depicted in Subfigure 3.2b. Subsequent users uploading the same data to their account are solely linked to the data without the need of any further data upload as depicted in Subfigure 3.2c. From the lack of this upload, the later users might infer the availability of this data in the storage. Data



(a) Hash equality indicates that certain data is already stored.



(b) Hash inequality: The data is added to the storage.



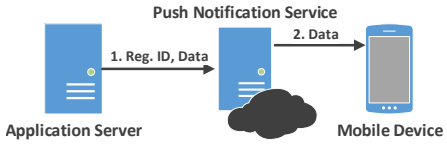
(c) Hash equality: Already available data is linked to a second account.

Figure 3.2: Data Deduplication

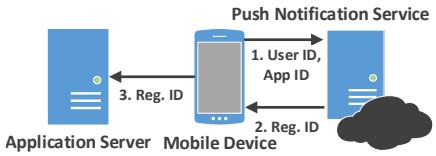
deduplication works on a file or block level. For better readability, we refer to files in the following paragraphs without loss of generality.

Based on these insights, [73] proposes two covert channels: In the first alternative, the sender and receiver have agreed on two files representing 0 and 1. The sender uploads one of these files to the storage, and the receiver is able to check which file is uploaded by attempting to upload both files. The sender proceeds with transmission through file deletion and upload of the respective file for the following bit after a certain time interval (c3). The other approach is based on a pre-defined template with a field for variable input. By brute-forcing all possibilities for the variable input using deduplication, the receiver is able to learn the file's content (c4). (c3) uses two distinct symbols, but the number of symbols might be easily increased by adding additional files. The number of symbols of (c4) is dependent on the potential values of the field. Both covert channels assume that no other customer uploads one of these files, and it is thus necessary to choose unusual files.

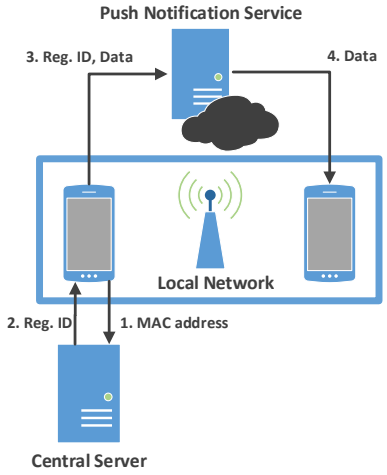
In [74], data deduplication is evaluated using the example of *Dropbox*. This specific storage solution did not delete files from the storage when users did so, and just removed the link between the user account and the actual file. Although this prevents the above covert channels, it enables other approaches: The sender uploads a file, deletes it from his own account and provides the file's hash value to the intended receiver. The receiver pretends uploading this file, and provides the received hash to the storage provider.



(a) Push Notification Service: Notification of Mobile Devices



(b) Push Notification Service: Registration of Mobile Device



(c) Covert Channel: Messaging between two Mobile Devices

Figure 3.3: Covert Channel via Push Notification Service

Then, his account is linked to the sender's uploaded data (c5). Beyond, *Dropbox* allowed uploading any file to an arbitrary account solely by using the receiver's host ID. This way, the sender is able to put information directly into the receiver's storage without ever claiming it as its own data (c6).

The advantage of (c6) is its increased anonymity in comparison to (c5) as the file is never associated with the sender's account. Even beyond, the sender does not necessarily have to be a regular customer with its own account at the *StaaS* provider. The difference between (c5) and (c6) is thus comparable to (c1) and (c2): (c5, c1) require the adversary to become a cloud customer, while (c6, c2) allow non-customers to be senders as well.

3.1.3 Novel Application Covert Channels

Spotique has been presented as a solution for communication with people in local proximity without publishing the current location to a broader audience, e. g., in social networks [53, 75]. It can also be considered as a covert channel as it introduces a communication channel that has not been intended this way. It is based on cloud push notification services like *Google Cloud Messaging*[3] or *Apple Push Notification Service*[4], whose intended functionality is the reliable notification of mobile applications in case there are news at the respective application server. In such a case, an application server sends registration

[3] developer.android.com/google/gcm
[4] developer.apple.com/notifications

IDs identifying the individual mobile devices and the message to the push notification service. The service itself guarantees delivery and sends the message immediately or later in case the addressed device is offline at the moment, see Subfigure 3.3a. At the mobile device, the operating system forwards the message to the appropriate application.

The registration IDs are generated in a registration process as depicted in Subfigure 3.3b. First, the mobile device reports its sender ID as well as the application ID to the push notification service, and receives a registration ID in response. In a second step, the mobile device forwards this registration ID to the application server.

Spotique modifies the registration insofar as mobile devices not only forward the registration ID, but also their MAC address to a central server that replaces the application server. The actual communication is depicted in Subfigure 3.3c: Sniffing on the local network a mobile device gains the MAC address of a node in proximity. To send a message, the sender asks the central server for the respective registration ID and sends it in combination with its message to the push notification service that reliably forwards the message to the intended receiver. Once the receiver's registration ID is known, the central server is not needed for succeeding messages and the channel can be even used to communicate with nodes that are not in local proximity anymore (c7).

Due to being a wide-spread peer-to-peer protocol, *Bittorrent* is increasingly used in *StaaS* and *SaaS* services for data synchronization among their internal nodes. The protocol uses a tracker that enables peers to locate others. Nodes announce their files available for download at this tracker by sending the torrent's hash file, a peer ID, IP address, port number, etc. The peer ID field is random and can therefore carry covert communication. The receiver accesses the content by requesting available seeders from the tracker. The latter sends all nodes providing the respective downloads including the IP address and the peer ID. By means of the address, the receiver identifies the covert channel's sender and infers the covert information from the ID field (c8). There exists also a compressed tracker response format, which omits a peer ID. In this case, the authors propose the use of another field – the alternative address. This field had originally been intended to enable connection passing through Network Address Translation (NAT) devices or proxys [76] (c9).

3.1.4 Discussion

Literature describes very different covert channel approaches. On the one hand, covert channels (c1, c2) exploit low layers of networking; on the other hand, approaches like (c5, c6) utilize the application layer. In any case, it boils down to (1) a sender that is able to modulate a certain resource (packet timing, stored files, etc.), and (2) a receiver that is able to infer this modulation. This point of view leads to the most obvious ways of covert channel mitigation – removal of (a) the shared resource in total, of (b) the sender's capability of modulation or of (c) the receiver's capability of observation. The latter two are commonly referred to as isolation (among tenants). Indeed, all alternatives

are prevalent nowadays, and have their distinct drawbacks for the provider and/or the customers.

The first strategy, i. e., rigid closure of potential covert channels, is common among *IaaS* providers. For higher payment, they provide *dedicated instances*[5] that are guaranteed to be alone on a physical server, or are solely co-resident to other instances of the same customer. This appears to be practicable in utterly sensitive scenarios, but not for the majority of cases as resource pooling and sharing is one of the cornerstones enabling *computing as a utility* for reasonable prices. Dedicated instances do not only have higher hourly fees[6], but a comparably high, hourly registration fee is charged in addition. This fee is independent of the number of running instances, and discriminates low-volume customers.

The more common approach however remains isolation, i. e., tenant behavior must not impact the other at all or only in a minimal way. It naturally opposes optimal resource utilization as isolation itself consumes resources. The cloud provider however might favor to lease these resources out to other customers in order to gain higher revenue instead of investing into better security[7]. Finally, both strategies – dedicated instances as well as isolation – protect against other tenants, but not against channels like (c6) that can be performed by practically everbody; the sender does not have to be a regular cloud customer or user.

In consequence, we believe that other directions of mitigation appear more promising, and should be considered in future research. Nowadays strategies focus on hindering covert channels as mentioned above, but development of novel covert channels – especially with the daily introduction of new cloud applications – might be as quick and straightforward as those for video gaming [77]. Like ancient Egypt lived with the annual Nile flooding, one might adapt mitigation strategy in a similar manner for cloud computing, i. e., living with the existence of covert channels, but protecting sensitive data against leakage via such channels. On the one hand, there are hardware-based approaches. For example, [78] protects security-relevant data like keys from unauthorized access by means of hardware-based control flow integrity, [79] proposes the implementation of additional hardware functionality that protects from a compromised hypervisor accessing its guests' memory page tables. On the other hand, there are software-based approaches; [80] partitions applications among more vulnerable public clouds running uncritical portions, and private clouds running more sensitive ones; [81] overlays public clouds with private clouds to meet higher security levels. Approaches like [78] and [80] apparently require a plan of action to distinguish critical from uncritical data, e. g., by means of risk analysis, while [79] and [81] protect all and might be more favorable for a cloud provider that is unaware of user data details.

[5] e. g., https://aws.amazon.com/ec2/purchasing-options/dedicated-instances/

[6] e. g., https://aws.amazon.com/ec2/pricing/

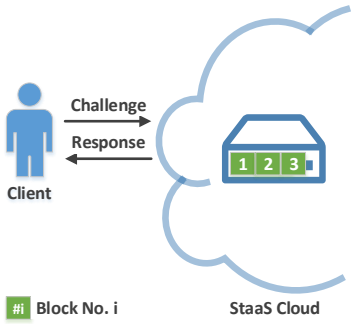
[7] Obviously, a provider is interested in a basic level of security to prevent everyday attacks, but might refrain from investment into more sophisticated solutions. Beyond, providers and customers naturally have contradicting interests. For example, customers wish instances that are resilient to malware; potentially increased resource use by malware however implies more revenue for the provider.

Cloud providers tend to camouflage their infrastructure for ostensible reasons of security; but providers consider their knowledge on infrastructure deployment also as their operational value that has to be kept secret. This behavior might increase an adversary's effort to develop covert channels, but also hinders customers from performing a detailed security analysis as well. This line of action conflicts with security's basic principle of disclosure [21]. Disclosure, however, offers substantial benefits for both customers and cloud providers. First, cloud security can benefit from analysis through external, e. g., academic, review as known from the field of cryptography. Second, disclosure might accelerate innovation in general (even beyond the field of security), and lead to the development of novel cloud applications that are in turn hosted at disclosing cloud providers (resulting in additional revenue). For example, car manufacturer *Tesla* declared its patents to be open source for exactly this reason [82, 83]. Especially cloud providers running their own hardware infrastructure do not have to fear new competitors as entry to this highly competitive market requires not only experience, but also immense upfront capital for physical infrastructure. Cloud computing requires a standardized way of disclosure that allows to keep cloud providers their worth of protection assets, and is trusted by customers at the same time.

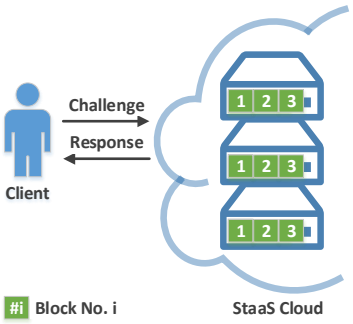
3.2 Approaches for Side Channels

In this section, we describe cloud-related side channels. For better readability, we call the entity that undeliberately reveals secret information victim, and the entity that accesses this information the adversary. We arrange side channels in five categories according to the secret information that they reveal and pre-requisites on placement of the adversary and the victim.

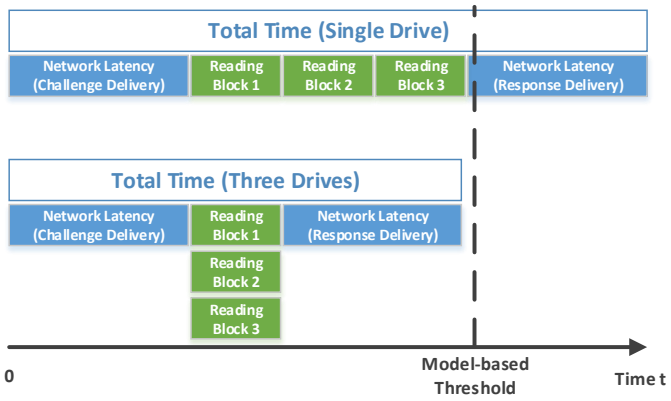
- *Internal side channels* are used by an adversary to reveal aspects about her own virtual machine or account. In this case, the cloud provider is the victim.
- By means of *placement side channels*, an adversary finds out about a victim's placement within the cloud.
- An adversary exploits *co-residency side channels* to gather information about neighbors, i. e., the victim and the adversary are co-resident on the same physical server.
- *Co-customer side channels* allow an adversary to reveal information about a victim that resides in the same cloud, but not necessarily on the same physical server, i. e., without the pre-requisite of co-residency.
- Finally, we discuss *en-route side channels* that are related to website fingerprinting. An adversary exploits network traffic characteristics to infer content that is accessed by the victim, and has therefore reside en-route, but not necessarily in the cloud, to analyze the traffic.



(a) Provider without Fault-Tolerance: All blocks are stored on the same drive.



(b) Provider with Fault-Tolerance: Blocks are stored on all drives.



(c) Distributed blocks are read simultaneously, and total challenge-response time is below threshold.

Figure 3.4: Remote Assessment of Fault Tolerance

3.2.1 Internal Side Channels

Internal side channels represent a novel type of side channels insofar as they provide customers information that are easily accessible in traditional IT landscapes but remain hidden in clouds. By evading cloud services' non-disclosure, they are means of gathering information about the provided infrastructure or providers' compliance with service level agreements.

Being used for data backup, cloud storage providers claim to use fault-tolerance in order to guarantee that data is not lost in case hardware components fail. General best practice in cloud computing as well as in traditional computing is data duplication across different hardware as well as physical data centers, i. e., locations. Thereby, the question how customers can verify a provider's guarantees arises.

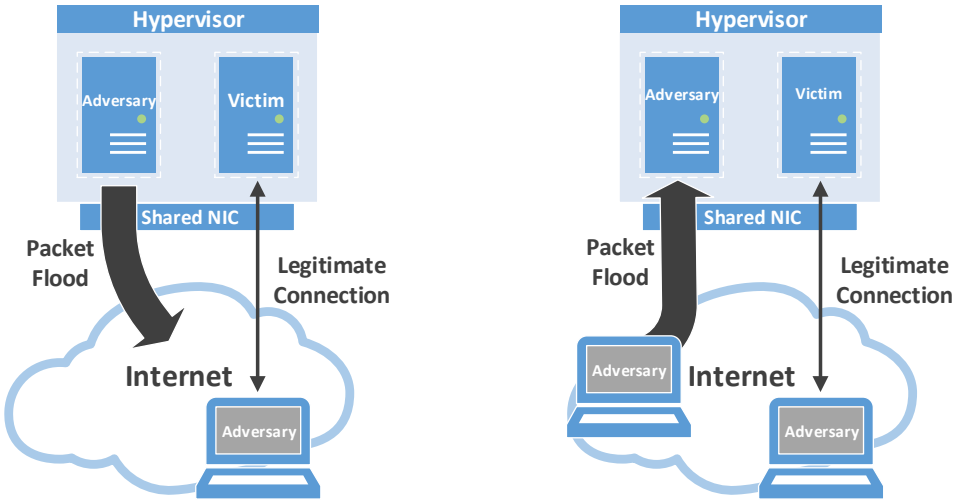
[36] presents remote assessment of fault tolerance by means of a challenge response protocol. First, the client requests the delivery of a number of data blocks from the storage solution. The latter retrieves these blocks. The time it takes to respond reveals the level of the provider's fault tolerance as it is assumed that a higher number of hard drives speeds up the response.

Figure 3.4 highlights this fact: The client requests three blocks in both cases. In the first scenario of Subfigure 3.4a, all blocks are stored on a single drive. Thus, the total time to deliver the response encompasses (1) the latency between client and server to deliver the challenge, (2) three times the period to read a block from the drive and (3) the latency to deliver the response back to the client, see also Subfigure 3.4c. As the total time measured is going to be higher than a threshold that is calculated by means of network and drive timing models, the client infers insufficient fault-tolerance. In the alternative scenario of Subfigure 3.4b, the blocks are available on all drives. In this case, the time to deliver the response consists of (1) the latency between client and server, and (3) the latency on the return path. As the three blocks can be fetched from the three drives simultaneously, the (2) period for reading a block just adds once to the total time, see Subfigure 3.4c. As the total time is below the threshold, one can deduce that sufficient fault-tolerance is provided (s1).

[34] complements this approach by assessing whether data is replicated over a number of geographically distributed data centers by measuring response time as well. The authors developed a model that allowed to infer the origin of cloud data in dependence of the response time and the client location (s2).

Both channels (s1, s2) measure the same physical quantity, i. e., response time, but infer from the results onto different internal states of the cloud. This raises the question whether both side channels can be applied to the same cloud, and still provide results with high accuracy. Control theory uses a related term *observability* to determine if and to which extent system internals can actually be inferred from available outputs [84]. Applying these concepts to (cloud) computing could help to assess potential impact of competing co-located side channels (s1) and (s2) onto their accuracy.

Cloud providers expand their data centers gradually and in accordance with their economic necessities. For example, a data center might initially consist of hardware A; is expanded by additional physical nodes of hardware B as the hardware market advanced and faces another expansion with nodes of hardware C. Such hardware upgrades transform data centers from homogeneous to heterogeneous. Variations in hardware performance propagate in virtual machines, which then may provide more or less hardware performance for the same money albeit being of the same instance type. [85, 86] use standard benchmarks, i. e., *UnixBench*, *RAMspeed*, *Bonnie++*, in combination with self-developed benchmarks to evaluate CPU, memory, disk and also network performance. With the gained knowledge, a customer is able to apply cost-saving approaches and choose a machine with better performance. We also refer to [87] for a more sophisticated strategy for performance optimization (s3).



(a) Cloud-based flooding with two adversary-controlled nodes

(b) Cloud-external flooding with three adversary-controlled nodes

Figure 3.5: Side Channels checking Co-Residency of Virtual Machines using Packet Flooding

3.2.2 Placement Side Channels

In terms of cloud computing, placement refers to the physical server a virtual machine resides at and we refer to the respective side channels as placement side channels. They are of special interest due to making it possible to reveal concrete information on a victim's location in advance of an attack. A unique aspect in clouds is co-residency detection, i. e., inferring whether a certain virtual machine is placed on the same physical server as oneself. This is of interest for adversaries as a number of hypervisor exploits require previous co-residency. Exploitation typically consists of two steps: First, the adversary instantiates a number of virtual machines and checks for co-residency of the adversary's machine with the victim. Second, she executes the exploit to harm her neighbor. However, co-residency checking is also of interest for benign customers to assess risks from neighbors. The literature describes three different approaches: (1) address vicinity, (2) measuring round-trip times and (3) the generation of deliberate delays in neighbor traffic.

Following the assumption that nearby nodes have close addresses, [88] developed a rapid test for non-co-residency in the *Amazon* cloud: If two node addresses are not within the same /24 IPv4 network prefix, they are not co-resident (s4). However, two addresses within the same prefix do not necessarily indicate co-residency. While this side channel appears inconspicuous due to the sole need of the victim's address, it requires knowledge on the respective cloud provider's addressing strategy as behavior differs.

While adversaries are looking for other virtual machines on the same physical server in *IaaS* clouds, they look for other platforms on the same virtual machine in *PaaS* clouds. Platforms of different customers share the operating system, and communicate with cloud-external nodes by means of the same IP address. Thus, a simple address comparison reveals co-residency [89] (s5). Similarly, co-resident virtual machines in *IaaS* clouds had the same gateway - the Xen hypervisor's privileged virtual machine running the respective devices drivers (*Dom0*) - in the past [35]. But the gateway is hidden now, at least at *Amazon* instances [88].

Round-trip times have been reported to be shorter among neighbors than among arbitrary cloud-based nodes [35] – a fact potentially caused due to short-circuiting of the hypervisor for performance reasons [90]. We denote such side channels as (s6).

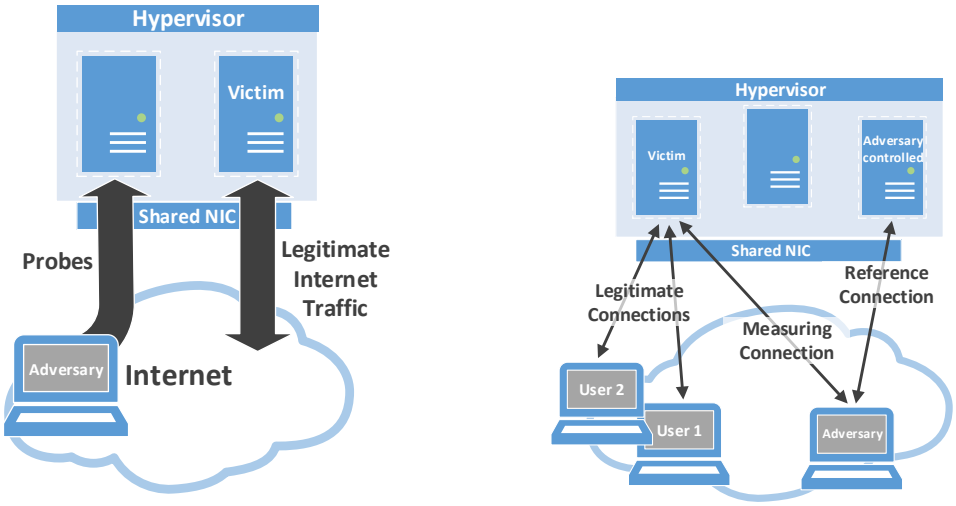
Assuming an adversary checking for co-residency with a victim based on flooding, two alternatives are available as depicted in Figure 3.5. First, the adversary maintains a legitimate connection from an external node to the victim. At the same time, she floods the network from her cloud-based virtual instance, see Subfigure 3.5a. In case this flood has a negative impact on the arrival packet rate, the virtual machines are co-resident [70, 71] (s7). Alternatively, the adversary can flood her own virtual machine from another external node, see Subfigure 3.5b. If extra latency is introduced into the legitimate connection, the virtual machines are again co-resident [37] (s8).

The basic principle of (s7) and (s8) is equivalent to their relative covert channels (c1) and (c2). The successful establishment of such covert channels between different nodes of the adversary implies co-residency of her virtual machine with the victim and forms a side channel. The reason lies in the fact that without co-residency these channels would not operate. Comparing the side channels, (s7) requires less nodes that are operated by the adversary in comparison to (s8). On the contrary, (s8) does not necessariliy require an adversary-controlled cloud instance and allows to check whether two virtual machines of strangers are co-resident. Then, an adversary might first attack a less secured neighbor before targeting the actual victim via the hypervisor.

3.2.3 Co-Residency Side Channels

Once co-residency is verified, an adversary is interested in further information about her neighbor. On the one hand, this additional information makes it easier to plan an attack; on the other hand, information about the neighbor is also helpful for benign customers, e. g., when deciding whether to stay, or terminate this virtual machine and instantiate a new one due to overlapping resource demands.

In [91, 92], timing side channels in shared event schedulers are examined. In the context of cloud computing, the appropriate scenario is depicted in Subfigure 3.6a. An adversary aims to infer a victim's (legitimate) networking behavior, e. g., to find usage patterns, peak loads or idle times. Therefore, she sends a low-bandwidth, high-frequency probe to the victim's neighbor and awaits the response. As both virtual machines share a packet scheduler the following holds: the more victim traffic, the longer the round-trip times of



(a) Side Channel using Low-Bandwidth, High-Frequency Probing

(b) Side Channel using Reference Connection

Figure 3.6: Side Channels measuring Neighbor's Traffic

the probes. We denote side channels of this type with (s9). The impact on the victim's privacy depends on the applied scheduling algorithm. First-come-first-serve provides high performance, but protects privacy least. The other extreme – protecting privacy at the cost of lower performance – is time-division-multiple-access [92].

The adversary could also directly probe the victim. However, there are scenarios where probing a neighbor appears more attractive: For example, a direct connection to the victim might be suspect. Further, the adversary needs a responding service of the victim. If such a service is not present, she might opt for a neighbor offering such a service. In comparison to (s7) and (s8), (s9) aims to reveal traffic volume, and not co-residency but requires previous co-residency. The exploited principle is similar to the one of covert channels (c1) and (c2). A virtual machine's traffic delays neighbors' traffic. The traffic causing the delay is deliberately caused by the adversary or sender in (c1), (c2), (s7) and (s8), but not in (s9). As in (s8), the adversary of (s9) does not have to control a cloud-based instance and does not have to register at the cloud provider.

In [70, 71], traffic is measured in a relative way by maintaining a measuring connection to the victim as well as a reference connection to a co-resident neighbor under the adversary's control, see Subfigure 3.6b. A changing ratio between the connections' throughput indicates a change in the victim's traffic. Any other causes for decreased throughput are filtered: network congestion or change in another co-resident load would impact the reference as well as the measuring connection in the same manner and not change the ratio. As its related covert channel (c1) and co-residence detection technology (s7), the packet arrival rate is measured (s10).

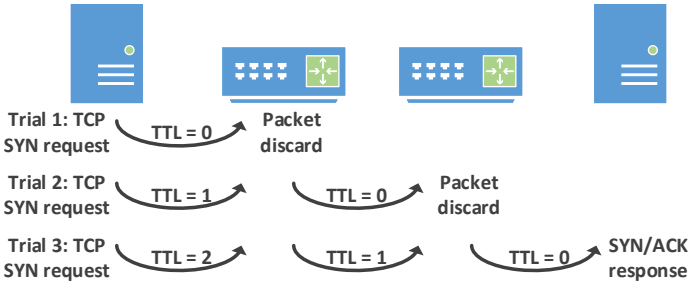


Figure 3.7: Side Channel counting Intermediate Hops

A more generic approach to infer neighbor resources is provided in [48]: Using two virtual machines – one is the adversary, the other the victim – that compete for network bandwidth on the same physical node, the authors show that causing CPU bottlenecks to the victim by requesting computing-intensive dynamic webpages enables the adversary to increase its network bandwidth. While this is presented as an attack mechanism called resource-freeing attack in [48], it might also serve as a side channel. If one experiences increased resource allocation for oneself while trying to trick a neighbor into such a bottleneck, it is possible to infer the neighbor's resource demand in normal operation (s11).

Considering (s10, s11), there are subtle differences in requirements. (s10) requires to connect to a victim via TCP and to regularly download some data whereas (s11) narrows this down to dynamic web pages. These differences are caused by the actual attack vector that is exploited by the respective side channel. (s10) utilizes the bottleneck of networking, (s11) the bottleneck of computing at the CPU. Finally, (s11) emphasizes mutual dependence of resources. However, this mutual dependence appears to be widely ignored with respect to cloud security and might become a starting point for the development of future attacks.

3.2.4 Co-Customer Side Channels

In contrast to co-residency side channels, we refer to side channels that reveal information about other cloud customers without requiring co-residency as co-customer side channels.

Data deduplication in *StaaS* does not only lead to covert channels, but also to related side channels: A customer can check whether a file has already been uploaded to the cloud and infer, e. g., whether somebody has placed confidential or illegal data in the cloud [73, 74]. Thereby, the algorithm that has already been used in the related covert channels (c3) and (c4), see Subfigure 3.2a, is followed. For example, [74] investigated the amount of copyright-protected material stored in *Dropbox* and found that it had been heavily used for storing *piratebay.org* torrents (s12).

IaaS assigns customers the highest level of customization, but also puts great effort into concealment of their internal infrastructure. Following the publication of [35], a number

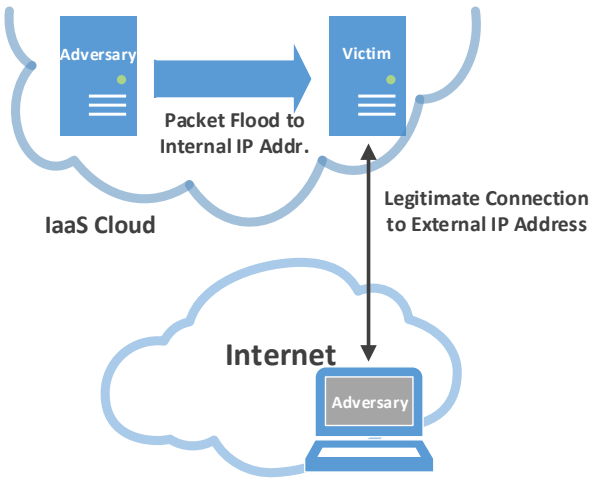


Figure 3.8: Side Channel that deanonymizes a cloud-based node's internal address

of classic network diagnosis tools were disabled, e. g., tracerouting. However, [37] shows that it is still possible to count intermediate hops between two virtual machines. Its principle is highlighted in Figure 3.7: The Internet Protocol's TTL field is generally decreased by one at every intermediate router, and discarded in case its value equals zero. Thus, the victim performs several trials of TCP connection attempts, and increases the TTL field from zero with every failed trial. The lowest TTL that causes a successful TCP SYN/ACK response is equivalent to the number of intermediate hops (s13). A single intermediate hop between virtual machines might indicate that they reside on the same server rack [88], and this in turn is of interest for power attacks: A simultaneous increase in power demand of machines on the same rack might in turn cause power outage by tripping the circuit breaker and results in denial-of-service [45].

Cloud-based nodes are typically reachable with an external address, but also have an internal address to be reachable for other cloud-based nodes. From an adversary's perspective, the internal address is of greater interest. First, firewalls might allow traffic from cloud-internal addresses while blocking external. Second, an adversary attacking a cloud-based victim from another cloud instance is able to utilize more bandwidth in comparison to attacking from outside as there is typically a high-speed cloud-internal network. For correlating an internal with an external address, [45] as well as [37] propose means of address deanonymization.

In [45], the authors highlight that the Domain Name System (DNS) differentiates between cloud-external and cloud-internal requests. While a request from outside the cloud provides an external address for a certain domain, queries from another cloud-based node lead to the respective internal address for the same domain due to performance

reasons. An adversary might simply correlate these addresses after two DNS requests (s14).

Alternatively, the adversary controls two nodes, at least one of them must be cloud-based to reach nodes by means of their internal address as depicted in Figure 3.8. The internal node sends a high amount of traffic to an internal address, while the other maintains a legitimate connection to the victim via the external address. If the latter experiences extra latencies, the two addresses belong to the same virtual machine, otherwise the flooder continues to the next internal address (s15).

The setup of (s15) is analog to (c2) and (s8); however, the legitimate connection is maintained to the host that is also probed. (c2) and (s8) split these two connections among two neighbors. Comparing (s14, s15), the adversary requires different prerequisites. (s15) needs an external address, and (educated) guesses on internal addresses or vice verse. (s14) requires a domain name for resolution, and thus no previous knowledge on addresses. Further, mitigation of the latter appears rather simple and a provider could decide to include external addresses in all DNS responses; though, this might cost resources as internal traffic is detoured. Mitigation of the first appears rather lavish as it requires changes of subtle networking details (at the hypervisor or the operating systems).

IaaS clouds frequently offer a way to publish machine images from third parties. These images contain pre-configured operating systems and can be instantiated by customers. However, the customer has to trust that the publisher has not left malicious code or back doors in them, as there is no control by the cloud provider. In [93], the authors discovered a number of virtual machines using the same Secure Shell (SSH) host key revealing their origin from the same machine image (s16). This has several drawbacks: First, it is possible to identify virtual machines that are inferred from the same machine image. Second, an adversary might instantiate the image herself, and prepare an attack specific to this image. Third, virtual machines are typically offered in different performance classes, and an adversary might infer a machine's class as certain images support only a limited number of different types.

Cloud providers are strongly interested in operating systems and the exact version installed on hosted virtual machines for the purposes of penetration testing, virtual machine management and digital forensics [94, 95]. While these papers rely on memory-based technologies, traditional network-based operating system fingerprinting is also applicable. Here, we can distinguish between active approaches with probing [96, 97] (s17) and passive ones with accessing logs or eavesdropping [98] (s18). However, as network-stack implementations are relatively robust over versions of the same operating system, they are not capable of providing the exact version.

(s16) seems to be more advantageous for malicious purposes than (s17) and (s18). Identifying the specific machine image instead of the operating system's version allows an adversary to tailor her attack more accurately to the victim. However, cloud providers could easily mitigate (s16) through checking machine images that are offered via their

market place, or by regularly checking the virtual machines for such striking features as equivalent host keys. In contrast, mitigation of (s17), (s18) appears unlikely as it would require to align all operating systems to behave in exactly the same manner for all details of networking, e. g., initial TTL values, timeouts, etc.

3.2.5 En-Route Side Channels

Beyond the above-mentioned categories, there are two side channels related to accessed websites. [99, 100] investigated side channels in *SaaS* providers and discovered that modern *SaaS* applications make it possible to infer private information about customers. The reasons are low entropy input, e. g., a limited number of alternatives for marital status, and stateful communication of a known program (despite encryption is used). This is exacerbated by the use of Web 2.0 technologies, e. g., AJAX, where little chunks of information are transmitted separately. By analyzing tax refund or online health services, the program logic and its internal states have been modelled to allow guessing the current position of a user within the model (s19).

[101] presents a technique for inferring the accessed website from frequency distribution of packet sizes despite using privacy enhancing technologies, e. g., Tor. This way an intermediate node, e. g., the cloud provider itself, is able to know which websites were accessed by hosted instances. A similar approach based on the measurement of volume, time and the direction of traffic is presented in [102] (s20).

3.2.6 Discussion

Clouds appear to be a black box for outsiders, but also for their customers. Thus, side channels have gained momentum with the success of cloud computing. Based on our literature research, we infer that interpretation of side channel results has to be performed with care. On the one hand, different side channels measure the same physical quantity in order to extract different information. For example, (s1) measures file access time to assess a file's existence on different drives. (s2) measures the same to infer geographic location, i. e., the data center, of a file that is stored in the cloud. On the other hand, interpretation is often dependent on the cloud provider and side channels cannot be re-used without adaption. For example, (s4) allows statements on co-residency and claims that instances that are not within the same /24 network prefix are definitely not co-resident. This was proven to be valid for *Amazon EC2*, but might not hold true for *Rackspace, Google Compute Engine, Microsoft Azure* or any other provider. Finally, cloud providers evolve their infrastructures without noticing customers. This means that, once revealed, side channels will not necessarily work in the future. For example, *Dropbox* appears to hinder (s12) now by uploading the files in any case [103].

Side channels can also be categorized according to their application scenarios. In Section 2.4, we identified three distinct application scenarios for side channels, namely industrial espionage, compliance checking and reconnaissance. Table 3.1 highlights suitability of presented side channels for these scenarios. Thereby, we separated industrial

		Ind. Espionage agst. Provider	Ind. Espionage agst. Customer	Compliance Checking	Reconnaissance
Internal	s1	✓		✓	
	s2	✓		✓	
	s3	✓		✓	
Placement	s4		✓	✓	✓
	s5		✓	✓	✓
	s6		✓	✓	✓
	s7		✓	✓	✓
	s8		✓	✓	✓
Co-Residency	s9		✓		✓
	s10		✓		✓
	s11		✓		✓
Co-Customer	s12	✓	✓		✓
	s13	✓	✓		✓
	s14		✓		✓
	s15		✓		✓
	s16		✓		✓
	s17		✓		✓
	s18		✓		✓
En-Route	s19		✓		✓
	s20		✓		✓

Table 3.1: Suitability of Side Channels wrt. Application Scenarios

espionage into two sub-categories. On the one hand, industrial espionage might be directed against the cloud provider itself to find details about its infrastructure. On the other hand, espionage might target somebody residing in the cloud, i. e., a cloud customer. The table indicates that the application scenarios are congruent with our five categories, e. g., all internal side channels serve industrial espionage against providers, and compliance checking; all placement channels serve espionage against other customers, compliance checking and reconnaissance. The odd ones appear to be (s12, s13) as they additionally serve industrial espionage against the provider in comparison to other co-customer side channels. The reason therefore lies in the distinct nature of these channels. (s12) allows not only to infer whether another customer has uploaded a specific file to the cloud storage, but also enables to find out whether the provider utilizes deduplication per se (or not). (s13) enables not only to infer the hop distance to a potential victim (espionage against customer), but might also be used to scout the whole cloud networking infrastructure by launching multiple instances and measuring the hops en-route (espionage against provider).

Further, we conclude from Table 3.1 that most side channels serve predominately malicious purposes, i. e., espionage or reconnaissance for later attacking, and should apparently be mitigated. Internal side channels (s1-s3) represent an exception as these kind of side channel reveal information like hardware and geographic spread or instance hardware types in a relative manner. This information supports the customer with regard to his own cloud participation, and further does not reveal any major trade secrets of cloud providers. Nevertheless, we believe that utilization of side channels for benign compliance checks is just a consequence of customer's lacking insight into the cloud infrastructure.

Many side channels have the potential to become a covert channel, or are accompanied by an associated covert channel as emphasized in Section 3.4. Thus, mitigation for these covert channels can potentially mitigate side channels, as well. At the moment dedicated instances and better isolation are predominant but protect only against side channels requiring co-residency. Disclosure of internal infrastructure involves external review, which increases the likelihood of discovery and mitigation of harmful side channels. However, protection of data from illegitimate memory access as for example described in [79] does not mitigate side channels, as information that is revealed via side channels is frequently not stored in memory.

Cloud computing seems to lack a methodical approach for security. At the moment, an arms race is taking place between research and cloud providers, and security is added in a retrospective manner. I. e., whenever a vulnerability like a side channel is detected it is patched. For example, [35] was the very first paper publishing cloud scouting by means of ordinary diagnosis tools like *ICMP Echo Requests*. In response, several cloud providers totally or partially filter *ICMP* [33]. The *Dropbox* client appears to have changed as well and is now uploading every file to the cloud in order to hinder (s12) [103]. Security however should be considered right from the beginning, i. e., in the design and development phase. The community is challenged to develop a planned approach to guarantee clouds that are secure by design. If security becomes part of the specification,

the potential for side channels (but also other kind of secret communication) is likely to decrease. Such an approach might be inspired by Privacy-by-Design [104] that even becomes mandatory according to soon-to-be European legislation [105].

3.3 Approaches for Obfuscation

Obfuscation techniques provide tools to achieve anonymity for people when using the Internet, e. g., Tor [30]. A major drawback of some of today's anonymity tools is the limited amount of proxy or relay nodes that allow packet rerouting. On the one hand, people want to stay anonymous; but on the other hand, they prefer not to route traffic from other users who want to stay also anonymous. As a solution, [106] proposes *Dust Clouds*, which consist of short-lived virtual machines running *Tor* that are launched at cloud providers. By providing specified machine images, adequate separation between a user's home node and the virtual machine is maintained with low effort. The issue remaining is billing for these virtual machines because billing data still allows linking back to the user and prepaid solutions are not widely available.

Based on the same idea, [107] presents *Cloud-based Onion Routing (COR)* an ecosystem using *Tor* that introduces another layer of indirection between the cloud provider and the communication partners. These intermediate anonymity service providers rent virtual machines from cloud providers, and run the relay nodes. Clients with the wish to anonymously communicate may create their own relay circuit spanning multiple cloud and anonymity service providers. This way communication spans multiple administrative boundaries to overcome trusting a single provider. Payment relies on encrypted tokens (o1). The authors further claim that *Tor* nodes are easily blocked due to having publicly announced, typically static addresses, but are also actively found by the Great Firewall of China [108]. Moving relays to the cloud enables frequent address change, but censors also tend to refrain from blocking cloud providers due to societal and economic reasons.

Countries applying censorship sometimes even refrain from blocking encrypted services in case they are (economically or socially) important. Circumvention approaches over such services are ignored as they would force censors in computationally expensive traffic analysing techniques, and false positives might hamper innocuous people. *CloudTransport* as presented in [109] is such a hide-within system, and exploits public cloud storage providers as tolerated encrypted services.

The architecture, depicted in Figure 3.9, connects a user in a censor's jurisdiction with a bridge by means of a shared storage account as a rendezvous point. The bridge is operated outside of the censored area, e. g., by volunteers. The user client wraps its network packets into a file, and uploads the latter to the account. The bridge waits for and reads the file, forwards the packets and deletes the file afterwards. The response is delivered back the same way: The bridge writes a file into the account for the user client.

The advantages for the respective communication partners are manifold: First, *Cloud-Transport* tunnels over available cloud storage protocols and promises to be indistinguish-

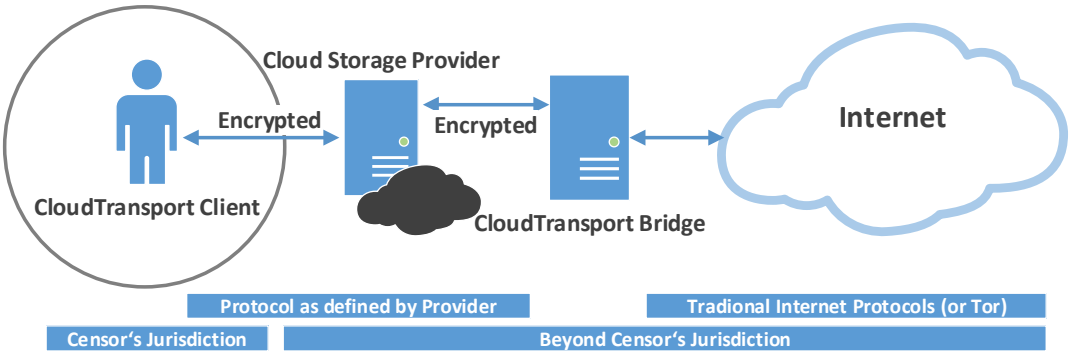


Figure 3.9: CloudTransport: Architecture

able from ordinary storage use. Approaches that imitate certain protocols are typically more prone to detection. Second, albeit discovering bridges is easy for observers it cannot impede *CloudTransport* by filtering traffic as the clients are connected to the cloud storage and the censor does not want to stop the latter service. Third, bootstraping is simple by means of an encrypted ticket that is written into a bridge-owned dead drop, and clients do not have to be informed of a bridge's address change (o2). A *Tor* client in contrast requires notification over a changed bridge address as well as attacking a bridge/filtering traffic towards a bridge negatively impacts data transmission.

The other aspect of obfuscation concerns botnets aiming at concealment of their *command and control* communication; otherwise, they risk detection. It has become a recent trend to install communication on social networks with the motivation that this traffic is lost amidst all the legitimate traffic users produce. [110] characterizes a *Twitter*-based communication structure (o3): The bot makes a request to a certain twitter account, and receives an RSS feed containing Base64-encoded text. After decoding, one or more URLs of the URL shortener *bit.ly* are gained. Being redirected to the original address, *zip* files are downloaded, decoded and executed. Gathered information is returned to a botnet's server. [111] even proposes a way to create plausible cover messages for the tweets to stay under the radar.

[112] however use *Pastebin* (o4), a clipboard-like website for sharing text-based content without the need for registration. Users might upload data anonymously, and can share their data with others by means of an URL containing a random ID. The latest posts are also accessible via the time line. A botmaster might access the data by means of the URL that it received from an infected device, or alternatively fetch the timeline at a regular interval. While the providers might remove suspect messages, the adversary itself cannot be identified. The server cannot solely be used as a dropzone for stolen data, but also to disseminate commands from the master to the bots.

SaaS based *command and control* structures' biggest advantages is their undetectability. It seems impossible to easily and accurately distinguish a botnet's behavior from legitimate one when using certain applications - especially as they have millions of users. Comparing (o3) with (o4) also reveals a difference on an application-level approach of mitigation: While *Twitter* could disable the account that had been hard-coded in (o3), the anonymous way of *Pastebin* hinders such an action.

For mobile botnets, a more sophisticated solution using push notification services is proposed in [113]. Thereby, the botmaster's commands are sent from a *command and control* server, that replaces the application server in Subfigures 3.3a and 3.3b, to infected mobile devices (bots) via the push notification service. High stealthiness in general is gained: First, the bot communicates only once directly with the *command and control* server – during registration[8]. Second, neither heartbeat traffic nor command dissemination cause high overhead or suspicious patterns in the traffic in comparison to apps, like mail clients and messengers, that are typically installed on mobile devices. The *command and control* traffic appears as a benign application's one (o5).

The question, however, arises why the use of push notification services as *command and control* infrastructure is considered as obfuscation, while *Spotique* (c7) is considered a covert channel in Section 3.1. We classified (o6) as obfuscation because hiding among a large set of nodes and users is the focus, but the technology itself is used as intended – even though in bad faith. In contrast, (c7) creates a communication channel for messaging between mobile devices. The channel is used in an unintended way.

Discussion

Our literature research reveals five approaches of obfuscation that utilize cloud computing. The motivation for going into the cloud however varies; in total, we identified three distinct reasons therefore. First, anonymity tools like *Tor* lack proxies and relays; there are not enough volunteers running such nodes. However, this shortage should not be considered as a shortage of computing power per se; people rather refrain from provision due to potential consequences, e. g., cyber-attacks against their own servers or legal prosecution. *Dust Cloud* and *Cloud-based Onion Routing* aim to overcome this issue by means of cloud computing. Cloud computing introduces an additional layer of indistinctness between volunteers (operating and paying for the relay) and the relay itself.

The second type of obfuscation wraps (censored or illegitimate) communication into traffic that appears benign like *CloudTransport* – a well-known strategy to overcome censorship [114]. Wrapping traffic in cloud applications exploits the economic and societal power of cloud computing as popular cloud applications are not blocked – despite being known to be used for overcoming censorship. If a censor totally blocked this cloud service,

[8] In an enhanced architecture, the return path has also been detoured over the push notification service and there is no direct contact. Thereby, the server generates a second push notification account, subscribes itself and delivers the necessary credentials in the malicious application.

	o1	o2	o3	o4	o5
Whistleblowing	✓	✓			
Censorship Evasion	✓	✓			
Exchange of Illegal or Regime-Critical Content	✓	✓			
Malware Communication			✓	✓	✓

Table 3.2: Suitability of Obfuscation wrt. Application Scenarios

it would risk severe societal or economic consequences. If it blocked certain connections, the risk of overblocking, i. e., blocking benign connections, would remain.

Third, numerous cloud applications are social apps like *Twitter*, and enable people to communicate in an easy way. Entry barriers are kept low to motivate people to join, but might negatively impact users' privacy and security [115]. Such low entry barriers have at least two consequences. On the one hand, a high number of people join the service, and cause lots of application-related traffic. In consequence, appearance of such traffic does not raise suspicion. On the other hand, entry barriers are low for botmasters and bots as well. This third kind of obfuscation exploits cloud computing due to its freedom of suspicion. Summarizing, motivation for cloud-based obfuscation is threefold: (1) Cloud computing provides an additional layer of anonymity; (2) cloud applications have a high societal or economic value and thus remain uncensored; (3) cloud applications have a high number of users, and thus related traffic appears unsuspicious.

Beyond, application scenarios appear to be a distinctive feature among obfuscation approaches. In Section 2.4, we identified four applications scenarios for obfuscation, namely whistleblowing, censorship evasion, the exchange of illegal or regime-critical content as well as malware communication. Table 3.2 shows the suitability of the five obfuscation approaches with respect to these four application scenarios. Approaches (o1, o2) are appropriate for the scenarios of whistleblowing, censorship evasion and the exchange of illegal or regime-critical content. The remaining approaches (o3 - o5) appear to serve malware communication, i. e., *command-and-control* communication, only. The first group appears to serve predominantly benign purposes and its implementations are of rather high technical finesse in comparison to the latter. Thus, mitigation against obfuscation in general remains a double-edged issue. Following the principles of our Western democracies, (o1 - o2) should rather be supported. Misusing social networks for bots should undoubtedly be mitigated. The development of mitigation against "benign approaches" might even play into the hands of censors.

Accordingly, we identified a number of future research directions with respect to obfuscation. These directions reflect the conflicts between mitigation and support, but also the multidisciplinarity of related application scenarios. Censorship evasion, whistleblowing

and regime critics are not solely technical challenges. Thus, evaluation of obfuscation's quality should not only rely on technical analysis; but also on expertise from sociology to answer the impact on society and how users actually use certain solutions, economics to investigate the monetary impact of certain actions and political science to investigate the impact on international relations.

Cloud applications might increase their entry barriers in order to prevent bots from joining their network, e. g., by using Completely Automated Public Turing Tests to Tell Computers and Humans Apart (Captachas) [116]. This, however, increases the effort for ordinary customers as well, and they might decide to use less demanding apps as an alternative. For example, less secured mobile messengers tend to be more popular than more secure alternatives [117, 118]. On the one hand, usable security might be able to develop enhanced security mechanisms for cloud applications that are handier for humans than today's, but protect against the participation of botnets. On the other hand, machine-to-machine communication [119] as caused by botnets is likely to have different characteristics than human communication. Thus, traffic anomaly detection is worth to be considered in future research for this distinct scenario. Similarly, a censor could aim to detect traffic anomalies in a cloud application's traffic, and abort suspect connection attempts [114]. The *CloudTransport* approach (o2) wraps packets of HTTP requests into files that are stored at an *StaaS* service. File updates caused by such secret communication might differ from ordinary file updates, and allow to infer improper use of the respective storage service.

3.4 Classification

In this section, we classify covert channels, side channels and obfuscation according to their characteristics. A taxonomy is depicted in Figure 3.10, and provides an overview on means of secret communication; it distinguishes covert channels in resource sharing, deduplication and novel application covert channels; side channels in internal, placement, co-residency, co-customer and en-route side channels. Details on these sub-categories of covert and side channels have been provided in Sections 3.1 and 3.2.

3.4.1 Classification of Covert Channels

We classify covert channels described in Section 3.1 according to the following attributes:

Exploited Channel: This attribute names the concrete technology which is exploited in the respective covert channel.

Symbol Encoding: Symbols are used to transmit the information content in telecommunications and define how a secret message is encoded.

Sender/Receiver: The attribute identifies the respective source and sink of the secret channel, which are not necessarily the same as for the overt communication. We distin-

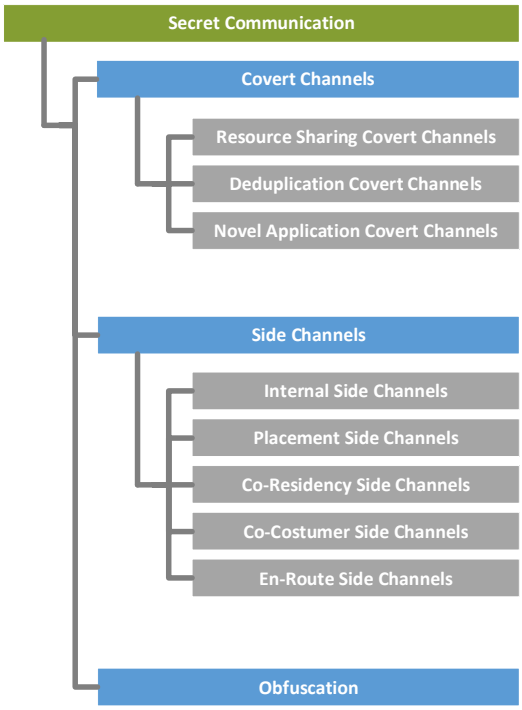


Figure 3.10: Taxonomy of Secret Communication in Cloud Computing

guish between *cloud providers*, *cloud-based nodes*, *(cloud-)external nodes* and *neighbors* as defined in Subsection 2.1.3. Neighbors are cloud-based nodes but have to fulfil additional constraints with respect to co-residency. Thus, we use the more specific term of neighbor if applicable. Further, we add *nodes en-route*, i. e., nodes between the sender and receiver of an overt channel.

Service Model: We distinguish between *SaaS*, *PaaS*, *IaaS* and *StaaS* as defined in Subsection 2.1.1.

Transmission Type: Transmission type states whether data is transmitted to a single receiver (*unicast*), a group of receivers (*multicast*) or a all receivers simultaneously (*broadcast*) – a distinctive feature also mentioned in [1]. Covert channels do not contain addresses as in classic networking protocols. Thus, we refer to channels that are accessible by everybody as broadcast. Those having a higher barrier than just measuring the respective quantity, e. g., by having some kind of token or a certain placement, are considered as unicast or multicast in dependence of the number of receivers typically fulfilling this requirement.

ID	Ref.	Exploited Channel	Symbol Encoding	Sender	Receiver	Service	Tr. Type	Bidir.
Resource Sharing Covert Channels								
c1[1]	[70, 71]	NIC sharing	packet arrival rate	nei.	ext.	IaaS	b	n
c2[1]	[37]	NIC sharing	latency of file download	ext.	ext.	IaaS	b	y
Deduplication Covert Channels								
c3	[73]	data deduplication	availability of certain files	ext.	ext.	StaaS	u	y
c4	[73]	data deduplication	inserted values in pre-def. template	ext.	ext.	StaaS	u	y
c5	[74]	shared storage	file content	ext.	ext.	StaaS	u	y
c6	[74]	data deduplication	file content	ext.	ext.	StaaS	u	y
Novel Applications Covert Channels								
c7	[53, 75]	push notification service	notification message content	ext.	ext.	SaaS	u	y
c8	[76]	P2P tracker mechanism	Peer ID field	int.	int.	SaaS[2]	u	y
c9	[76]	P2P tracker mechanism	alternative address field	int.	int.	SaaS[2]	u	y

Reference (Ref.)
Sender/Receiver: cloud provider (prov.), neighbor (nei.), cloud-based node (int.), external node (ext.), node en-route (en-rou.)
Transmission Type (Tr. Type): unicast (u), multicast (m), broadcast (b)
Bidirectionality (Bidir.): yes (y), no (n)

[1] Channel is based on co-residency.
[2] P2P protocols are used at various *SaaS* for updating internal servers in datacenters.

Table 3.3: Covert Channels in Cloud Computing

Bidirectionality: This column gives information on whether a channel is capable of information transfer in both directions, i. e., not only from the sender to the receiver, but also from the receiver to the sender.

The results of classifying the presented covert channels are shown in Table 3.3.

3.4.2 Classification of Side Channels

We categorize side channels described in Section 3.2 according to the following attributes:

Extracted Information: This column identifies the information that is gained by the application of the respective side channel.

Measured Quantity: This attribute names the physical quantity that is measured by the side channel's operator to infer the extracted information.

ID	Ref.	Extracted Information	Measured Quantity	Service
Internal Side Channels				
s1	[36]	hardware spread of file	file access time	StaaS
s2	[34]	geographic location of file	file access time	StaaS
s3	[87, 85, 86]	instance hardware type	network bandwidth	IaaS
Placement Side Channels				
s4	[35, 88]	non-co-residency	IP network prefix	IaaS
s5	[89]	co-residency	IP addresses	PaaS
s6	[35]	co-residency	round-trip time	IaaS
s7	[70, 71]	co-residency	packet arrival rate	IaaS
s8	[37]	co-residency	latency in file download	IaaS
Co-Residency Side Channels				
s9	[91, 92]	neighbor's traffic amount	round-trip time	IaaS
s10	[70, 71]	neighbor's traffic amount	TCP throughput ratio (measurement/reference connection)	IaaS
s11	[48]	neighbor's resource use	network bandwidth	IaaS
Co-Customer Side Channels				
s12	[73, 74]	file availability	file upload	StaaS
s13	[37]	no. of intermediate hops	min. TTL of successful connection	IaaS
s14	[45]	private address	internal DNS resolving	IaaS
s15	[37]	private address	latency in file download	IaaS
s16	[93]	Amazon Machine Image (AMI) type	SSH host key	IaaS
s17	[96, 97] icw[1] [94, 95]	operating system/version	protocol response behavior (active)	IaaS
s18	[98] icw[1] [94, 95]	operating system/version	protocol behavior (passive)	IaaS
En-Route Side Channels				
s19	[99, 100]	inserted value in web application	response data size	SaaS
s20	[101, 102]	accessed website	frequency distribution of packets	IaaS

[1] in conjunction with

Table 3.4: Side Channels in Cloud Computing

Service Model: see Subection 3.4.1

The results of side channel classification are shown in Table 3.4.

3.4.3 Potential of Side Channels Becoming Covert Channels

ID	Symbol Encoding	Sender	Receiver	Feas.	Tr. Type	Bi-dir.	ID
Internal Side Channels							
s1	number of distinct physical disks	prov.	ext.	n	-	-	-
s2	geographic locations	prov.	ext.	y	u	n	f1
s3	physical server of virtual machine	prov.	int.	y	u	n	f2
Placement Side Channels							
s4	network prefix or physical server of virtual machine[1]	prov.	int.	y	b	n	f3[2]
s5	network address or virtual machine of platform[1]	prov.	ext.	y	b	n	f4[2]
s6	round-trip time or physical server of a virtual machine[1]	prov.	nei.	y	m	n	f5[2]
s7	physical server of a virtual machine	prov.	ext.	y	b	n	f6[2]
s7	packet arrival rate	int.	ext.	y	b	n	c1[2]
s8	physical server of a virtual machine	prov.	ext.	y	b	n	f7[2]
s8	latency in file download	ext.	ext.	y	b	y	c2[2]
Co-Residency Side Channels							
s9	traffic amount	nei. or ext.	ext.	y	b	y	f8[2]
s9	physical server of a virtual machine	prov.	ext.	y	b	n	f9[2]
s10	traffic amount	nei. or ext.	ext.	y	m	n	f10[2]
s10	physical server of a virtual machine	prov.	ext.	y	m	n	f11[2]
s11	levels of CPU use	ext.	nei.	y	m	n	f12[2]
s11	physical server of a virtual machine	prov.	nei.	y	m	n	f13[2]

Reference (Ref.)
Sender/Receiver: cloud provider (prov.), neighbor (nei.), cloud-based node (int.), external node (ext.), node en-route (en-rou.)
Feasibility: yes (y), no (n)
Transmission Type (Tr. Type): unicast (u), multicast (m), broadcast (b)
Bidirectionality (Bidir.): yes (y), no (n)

[1] The provider is able to modify the one or the other; however, the receiver cannot distinguish.
[2] Channel is based on co-residency.

Table 3.5: Side Channels with Potential of Becoming a Covert Channel (Part I)

Side channels in general have the potential to become covert channels. Necessary conditions for covert channel existence were proposed in [120, 121]. The defined *Constraint of Communication* condition requires the existence of confidential information at respective nodes. Then, a *Potential for Communication* is identified in case communication from

ID	Symbol Encoding	Sender	Receiver	Feas.	Tr. Type	Bi-dir.	ID
Co-Customer Side Channels							
s12	file availability	ext.	ext.	y	u	y	c3,c5,c6
s13	no. of intermediate nodes	prov.	int.	n	-	-	-
s14	internal addresses	prov.	int.	y	b	n	f14
s15	internal addresses	prov.	int.	y	b	n	f15
s16	host keys	int.	int. or ext.	y	b	y	f16
s17	protocol-specific response behavior	int.	int. or ext.	y	b	y	f17
s18	protocol-specific behavior	int.	en-rou.	y	m	n	f18
En-Route Side Channels							
s19	web-service's input fields	ext.	en-rou.	y	m	n	f19
s20	access of certain websites	int.	en-rou.	y	m	n	f20

Reference (Ref.)
Sender/Receiver: cloud provider (prov.), neighbor (nei.), cloud-based node (int.), external node (ext.), node en-route (en-rou.)
Feasibility: yes (y), no (n)
Transmission Type (Tr. Type): unicast (u), multicast (m), broadcast (b)
Bidirectionality (Bidir.): yes (y), no (n)

[1] The provider is able to modify the one or the other; however, the receiver cannot distinguish.
[2] Channel is based on co-residency.

Table 3.6: Side Channels with Potential of Becoming a Covert Channel (Part II)

this source to a sink exists. However, the assessment regarding confidential information must be done for every system individually. In the following, we identify the necessities of a side channel to become a potential covert channel:

- At least two distinct **symbols** are necessary to represent 0 and 1.
- A **sender** is somebody who is able to change the symbols.
- The **receiver** is able to access these symbols after transmission to infer the secret message.

Feasibility: Feasibility refers to whether there is potential for a covert channel. Beyond the identification of at least two disjunct symbols, sender and receiver, the change in states has to be of adequate effort. We define adequate effort as follows: (1) The change from one symbol to another one is possible within seconds. (2) The change does not require any physical modification.

ID	Ref.	Channel	Sender	Receiver	Service	Tr. Type	Bi-dir.
o1	[106, 107]	*Tor* traffic	ext.	ext.	IaaS	u	y
o2	[109]	cloud storage	ext.	ext.	StaaS	u	y
o3	[110]	*Twitter* communication	ext. (bot master)	ext. (bot)	SaaS	b	n[1]
o4	[112]	*Pastebin* communication	ext. (bot)	ext. (bot master)	SaaS	b	y
o5	[113]	push notification service	ext. (bot master)	ext. (bot)	SaaS	u	y

Reference (Ref.)
Sender/Receiver: cloud provider (prov.), neighbor (neigh.), cloud-based node (int.), external node (ext.), node en-route (en-rou.)
Transmission Type (Tr. Type): unicast (u), multicast (m), broadcast (b)
Bidirectionality (Bidir.): yes (y), no (n)

[1] The respective references describe a unidirectional approach, but bidirectionality would be feasible.

Table 3.7: Obfuscation in Cloud Computing

Transmission Type, Bidirectionality: see Subsection 3.4.1
These fields are filled solely in case of feasibility.

The results of potential covert channels based on the presented side channels are provided in Tables 3.5 and 3.6. We identify 18 out of 20 side channels that have the potential to become a covert channel, five thereof even allow the development of two covert channels each. For every of these future covert channels, we introduce an additional identifier with the initial letter f and a number for unique identification in the remainder of this section. Alternatively, it is referred to the respective covert channel with the initial letter c if this channel has been already described in the literature.

3.4.4 Classification of Obfuscation

We classify obfuscation techniques according to the following characteristics:

Channel: This column identifies the respective communication channel that is used for obfuscated traffic. In comparison to covert channels, the channel is used as intended.

Sender, Receiver Service Model, Transmission Type and Bidirectional: see Subsection 3.4.1

The classification results are provided in Table 3.7.

3.5 Findings and Discussion

In this section, we summarize and discuss our findings. We group our results into three classes: general findings, applicability to communication scenarios, and potential damage.

3.5.1 General Findings

Cloud as Obfuscation Infrastructure: The connection of external nodes via covert channels further means that the cloud serves as an intermediate, which is rather a typical characteristic of obfuscation, see Table 2.1. Still, we consider (c2-c7) as well as (f8) and (f10) covert channels as they use a resource unintended for communication or in an unintended way for information transmission and do not hide in the mass. Nevertheless, it must be stated that these covert channels include an aspect of obfuscation, which might enable even better hiding.

Interpretation of Side Channel Information: Side channels deliver various kinds of information. However, measured quantities seem ambiguous for some approaches: Table 3.4 reveals that the access time is an indicator of the file's hardware spread (s1), but also of its geographical spread (s2). In a similar way, latency during a download is introduced in the case of probing for co-residency (s8), the neighbor's traffic amount (s9) as well as in the case of address deanonymization (s15). Thus, one has to use care at the interpretation of measurements.

Secret Communication Approaches and Cloud Delivery Models: We found approaches for secret communication in *SaaS*, *PaaS*, *IaaS* and *StaaS* clouds. This indicates that there is potential for secret communication in all types of clouds; however, the distribution among delivery models varies. Covert channels arise primarily from *StaaS*; side channels from *IaaS* and obfuscation from *SaaS*. Approaches in *PaaS* represent a minority.

Potential for Covert Channels from Side Channels: The classification of covert channels from the literature in Table 3.3 has shown that almost all of them still connect external nodes with each other although a number of additional stakeholders are available in cloud environments, e. g., the provider or a neighbor. At the same time, considering potential future covert channels based on today's side channels revealed a wide variety of sender/receiver combinations, see Tables 3.5 and 3.6. We identified potential covert channels between cloud providers and cloud-external nodes or cloud-based nodes. This leads us to the conclusion that these novel combinations have not been considered in depth by now.

Potential of Side Channels Checking for or Demanding Co-Residency: The analysis of side channels becoming covert channels revealed a distinct group of channels. These side channels have two characteristics: First, they check for co-residency or require

previous co-residency in their setup. Second, a third party (actively) performs a certain action that consequently enables to infer information from the measured quantity. These actions encompass flooding from a victim's neighbor (s7), or flooding the neighbor (s8), generating general traffic (s9, s10) or modulate resource use by means of certain network requests (s11).

These channels bear two degrees of freedom (influence factors) each that can be exploited as symbols in a potential covert channel: The cloud provider might move the virtual machine to another physical server and void co-residency indicating a symbol, and moving the machine back indicating another symbol. This leads to a series of covert channels having the provider as a sender, and placement as symbols (f6, f7, f9, f11, f13). Their difference lies in the measured quantity. Alternatively, the third party might act as a sender by performing its action or not (or at another intensity) leading to side channels among internal and external nodes in various constellations (c1, c2, f8, f10, f12). Every of the above mentioned side channels results in two covert channels each. In comparison, (s4-s6) each lead only to a single covert channel (f3-f5) because no third party is required. The provider might change a machine's placement or its network address, but the channel remains anyway the same from the receiver's point of view.

3.5.2 Applicability to Communication Scenarios

In the following paragraphs, we aim to discuss the approaches of secret communication with their relevance to the secret communication scenarios defined in Section 2.4.

Secret Communication from Cloud Providers: With the analysis considering side channels becoming potential covert channels, we identified additional stakeholders in communication. One might ask for the adequate application scenario of covert channels with the cloud provider as sender (f1-f7, f11, f13-f15). In this context, we want to highlight their potential in insider industrial espionage or whistleblowing.

Side Channels to get Insight about Cloud Providers: In Section 2.4, we identified three application scenarios for side channels, among them outside compliance checking. In compliance checking, side channels might be used to gain knowledge that is not directly accessible by means of the operated machine or account, and verify a providers' conformance to the service level agreement. Channels (s1) and (s2) are of interest to prove whether the provider has spread data for fault tolerance over various hardware and geographical locations, (s3) allows to choose a certain type of offered hardware. (s12) makes it possible to check whether a *StaaS* provider applies deduplication across multiple customers. (s13) allows to infer a provider's internal network structure. Beyond, side channels might be applied for protection: (s4-s8) allow to ensure that a cloud instance is alone on a physical server, or only co-resident to benign ones, e. g., of the same organization. (s12) makes it possible to check whether a confidential file has been moved into a *StaaS* solution.

Side Channels to Search for Victims: The majority of side channels seem to enable the search for victims to plan a subsequent attack: (s5, s6, s7, s8) enable checking for co-residency or at least for vicinity (s4, s13); (s9-s11) allow to infer a neighbor's traffic load; (s16-s18) reveal a node's operating system or image type; (s14, s15) its internal address. (s12) allows to identify the presence of a victim by the availability of a specific file. (s19, s20) enable to spy on a victim's communication. Industrial espionage against cloud customers might use the same channels, but without the intention of attacking.

Obfuscation Objectives: All types of obfuscation connect external groups, see Table 3.7. However, approaches (o3-o5) are used for *command and control* between the botmaster and its bots. (o1, o2) have been proposed *with the good in mind*, i. e., censorship evasion and the transmission of regime-critical content; both support unicast and bidirectional communication. In comparison, obfuscation for bot nets appears less sophisticated with respect to their method of concealment as well as the lack of encryption and tend to be rather broadcast (o3,o4). An exemption is the unicast and more sophisticated approach (o5). It might thus be worthwhile to think about it as a means of censorship evasion or transmission of regime-critical content. Obfuscation for bot nets is based without exception on *SaaS* clouds. It seems that the diversity and seemingly endless supply of *SaaS* is a good substrate for C&C infrastructures of botnets, and (partly) a successor of the formerly used IRC channels.

3.5.3 Potential Damage

The potential damage that can be caused by secret communication depends on several factors.

Covert Channel Capacity: Covert channels' impact is considered to depend on their capacity. The literature review revealed only the capacity of the following two approaches: 4 bits per second (c1), 20 bytes (c8) or 4 byte (c9) per announcement to the tracker. In some scenarios, a single bit of transmitted data suffices, while a high-capacity channel is useless in others due to an overall lack of sensitive data. Hence, we believe that it is more promising to ask which information is gained by the adversary, and which advantages does the latter take from this information.

Value of Side Channel Information: In the case of side channels, one has to ask oneself likewise whether it is acceptable that the information gained from them is known. While this might be the case for internal side channels that reveal aspects about one's own instance, it is definitely not the case if other customers are measured, as there are privacy concerns.

Dependance on Obfuscation Objectives: For obfuscation, the answer seems divided. Approaches such as cloud-based *Tor* rather deserves support, while hindering botnets from using benign infrastructure seems worthwhile. However, mitigation remains

a double-edged sword and the development of mitigation might play in the hand of censors.

CHAPTER 4

Investigating Firewalls in IaaS Cloud Computing

In traditional networks, perimeter firewalls protect internal nodes from attacks from the outside; the firewall as well as the internal nodes are managed by the same entity and considered to be benign. Cloud computing blurs this concept of benign internal and potentially malicious external; basically everybody is able to rent cloud instances residing within the perimeter and run their applications there, while the provider remains unaware, i. e., considering used protocols, ports, etc., in terms of networking. Nevertheless, firewalls remain an important appliance within cloud computing, and a number of cloud providers offer firewalls to their customers. At the same time, providers aim to conceal their infrastructure for superficial reasons of security, but also in order to protect their competitive edge. In consequence, the cloud remains a black box for their customers, and so do the offered firewalls. Details on their type, location within the architecture, functionality, etc. are rare; documentation is rather unsatisfying due to its recipe style; and in turn, security considerations are shots into the dark.

In this chapter, we aim to overcome this gap in knowledge, and investigate the role and security of firewalls at major Infrastructure-as-a-Service (IaaS) cloud providers. Therefore, we utilize firewall tests as side channels in order to gain more details on the deployed firewalls, and investigate the latter's evolution over time between 2014 and begin of 2016. Our analysis shows that cloud providers are constantly introducing new functionality or improving usability. Thus, we provide our extensible test tool on an open-source base to the public. Summarizing, our results shed light on firewall aspects which are not accessible by average cloud tenants. We address our results' implications on securing cloud instances, and consider improvements for the provider as well as the machine owners.

The remainder of this chapter is structured as follows: Section 4.1 provides background

Provider	Service Model
Amazon Web Services	IaaS
IBM (Softlayer)	IaaS / PaaS
Microsoft Azure	IaaS / PaaS
Google	IaaS / PaaS
Rackspace	IaaS

Table 4.1: A Comparison of Cloud Computing Providers

on major IaaS providers, and different types of firewalls. As the providers typically refrain from providing in-depth information on their infrastructure, we develop an extensible tool for firewall probing in Section 4.2. In Section 4.3, we analyse firewall evolution over time. Section 4.4 introduces additional test cases focusing on the aspects of TCP filtering, fragmentation and Path MTU discovery. Section 4.5 discusses the results and infers practical advice for gaining secure virtual machines. The chapter is concluded in Section 4.6.

4.1 Background

In this section, we provide an overview of today's cloud computing providers and a typology of firewalls.

4.1.1 Major IaaS Cloud Computing Providers

The current cloud computing marketplace is a young market and still relatively diverse [11], but a number of larger providers hold considerable market share. This paper looked at a number of different cloud computing providers who all enjoy a considerable level of market share in the cloud computing space, summarized in Table 4.1. Consistent with the origins and high initial setup costs of a cloud computing platform, they are all large, well recognized names in the technology sphere. Providers are ranked in approximately decreasing size of market share, although the way in which cloud computing operates makes it extremely difficult to make accurate comparisons. Companies are often under no obligation to report the extent of their revenue which comes from cloud computing, and the use of different metrics makes direct comparison difficult. However it is universally acknowledged that Amazon is the largest player in the cloud computing space, with as much cloud revenue and capacity as the majority of the competition combined [122]. Many providers offer a number of service types or are difficult to classify. Google's Compute Engine uses an IaaS model much the same as Amazon's Elastic Compute Cloud (EC2), which shares little with Google App Engine, the company's other offering firmly

in the PaaS area. Parts of the Azure cloud suite are also PaaS offerings, but their virtual machines are a standard IaaS product.

4.1.2 Firewall Typology

A firewall is a component residing at the border of two networks which inspects traffic going from one to the other. For best security results, no traffic should bypass the device to guarantee inspection of the whole communication process. Reflecting the historic development of these security tools, firewalls are grouped into three types [123].

The most simple are *packet filters*. They hold a number of rules which define allowed or disallowed traffic; while the first is forwarded to the other network, the latter is dropped. Rules include source and destination addresses as well as ports identifying the service. The decision is based solely on the currently inspected packet and no connection context is maintained. The advantages of this general approach allow the firewall to handle a high level of service without any respective in-depth knowledge; this also allows the integration of newly developed services the firewall is not already aware of. Beyond, the simple architecture has only a limited need for resources in comparison to other types. On the other hand, they do not have full insight into the communication as a number of protocols are stateful, i. e., packets may be in general valid, but not at this certain point in communication.

Stateful filters go beyond this approach and maintain a context for connections. This allows the use of previously seen packets as part of the decision regarding the current packet [124]. This way it is possible to guarantee that certain packet types are protocol-compatible; not only considering the message format, but also their point in the communication process, although at the cost of requiring more memory. Typically, real-world stateful filters also use simple packet filtering for the best effect.

The third step in firewall evolution are *application layer filters* which have special code for every application. While this allows even deeper inspections and is thus also considered to be more secure, the drawback is the specialized code for every single application. This generally leads to the situations that only the best known services are available, without any extension for niche or novel protocols, and is even more costly in terms of computing. Currently, *deep packet inspection* is all the rage, which also scans the packet payloads, e. g. for malware.

For all kind of firewalls, the permissive and the restrictive approach can be applied. Describing the default behavior, a permissive firewalls allows all traffic to pass unless specified otherwise; restrictive configurations drop everything unless specified. Assuming in general everything as evil, the latter is far more common.

4.2 Firewall Testing Tool

In this section, we present our firewall test tool. First, architectural aspects are considered; then, we describe our implementation.

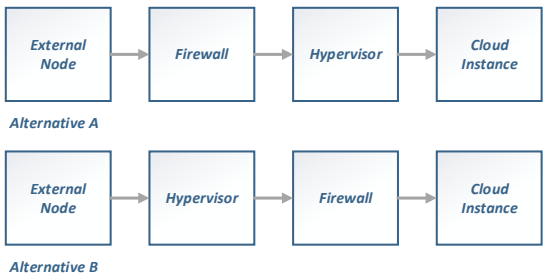


Figure 4.1: Alternatives for Hypervisor-Based Firewalls

Figure 4.2: Assumption for Black Box Firewall Testing in Cloud Environments

4.2.1 Architectural Consideration

Although firewalls are offered by multiple cloud providers, none of them allows deep insights into the architecture. The available documentation is typically of limited use due to being more like a handbook. With regard to firewalls, the following is known from a customer's perspective: (1) We are able to configure ports and addresses, mostly from a web interface for a group or a single machine. (2) Every customer is able to make a configuration for his own needs. However, the providers refrain from stating whether the customer can configure partially a perimeter firewall this way or a hypervisor-based firewall. While Amazon's approach of security groups lets one think of a more de-centralized approach, Microsoft Azure naming of *end points* lets one rather believe in a configurable perimeter. In the case of a hypervisor-based solution, we are further not able to infer whether the firewall is placed in front of the hypervisor, or afterwards directly before the guest operating system, see Figure 4.1 for both alternatives.

While this seems minor at first sight, the order may have serious impact: Hypervisors are not obliged to deliver packets to the guests without any alteration. For example, they can decide to reassemble fragments before forwarding for reasons of performance. One has to be aware, that the connection between hypervisor and guest is not constricted to typical network requirements like a MTU (Maximum Transmission Unit). Thus, in the first case of Figure 4.1 the packets investigated by the firewall are how they traveled the network, while they might have been altered by the hypervisor in the second case. This has impact on the way firewall rules have to be defined as well as the complexity of the firewall mechanism itself, e. g., a hypervisor reassembling the fragments releases the firewall from doing so.

As a consequence of these uncertainties, we have chosen to establish a black box model

including the firewall independent of its location in combination with the hypervisor functionality, see Figure 4.2. We are able to access the prober as well as the virtual machine for measurements and therefore define the following general test approach:

1. Start of the capturing tool on the receiver, i. e., the virtual machine,
2. Establishment of pre-conditions, e. g., TCP handshake,
3. Sending of test packet(s) from the prober to the virtual machine,
4. and observe whether packet(s) has/have been captured by the sniffing tool.

We prefer this over invoking a response from the virtual machine as this would add more chances for failure. By means of this set-up and specific test scenarios, we aim to answer the following question:

Which aspects (e.g. protocols and respective fields) are filtered? This aspects encompasses the ISO/OSI layer the firewall is inspecting and which protocol header fields are included into inspection. Conceivably, we will heavily work with the network layer protocol IP and transport layer's TCP and UDP. This also include layer-dependent mechanisms, e. g., fragmentation.

Do cloud firewalls reveal stateful or stateless behavior? Knowledge on firewall behavior enables customers to estimate the extent of protection and the residual measures that are necessary to gain the required level of protection, e. g., by additional host-based firewalls. As today's state-of-the-art are stateful firewalls, it would be unusual to find plain packet filters. Thus, we will investigate the "extent of statefulness" of the tested implementations.

Are application layer filters implemented? Application layer filters would imply a large intervention into a customer's traffic and might restrict their free choice of ports. On the other hand, it would provide more control on what is going into the cloud. We limit our research however on application layer filters for HTTP as we believe that this would be the first choice beneath SSH and FTP for providers for implementation, due to the vast number of web servers in the cloud.

4.2.2 Implementation

As a precondition we presume that at least one port is reachable by the testing client, which is used later as a feedback and communication channel between the test client and the server. Our approach consists of two components: A testing client that executes test cases against a cloud instance, and a server component running on the cloud instance and recording the traffic during test execution. A test case implements a concrete scenario outlined in Section 4.3.2. The test case can evaluate the network traffic that reached the

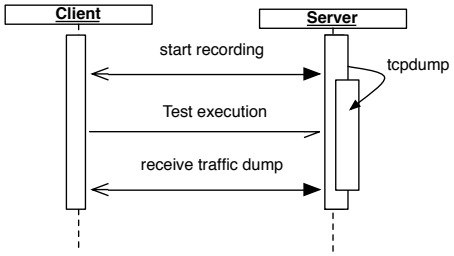


Figure 4.3: Communications between Test Client and Server Component

server instance in order to decide if the test case was blocked by the cloud firewall, i. e., it represents a side channels revealing a firewall's range of functionality. In best case the test case hides a randomized token, e.g., by adding it to the payload, within the test connections and seeks afterwards in the traffic dump for this token. If the token is present, the test case fails because the request has passed the cloud firewall.

Figure 4.3 gives a detailed view of the communication between the test client and the server component running on the cloud instance. In detail, the client connects to the server component running on a cloud instance using a plain TCP connection. This channel is used for coordination and callbacks. A single test case is stored as a separate file in a directory. The client loads a test case from this directory and creates a test case instance. Afterwards the client tells the server component over the communication channel to start recording the network traffic using tcpdump. Subsequently the client executes the test case and requests the tcpdump of this session from the server component. The test case instance now evaluates the success of this test iteration using the dump received from the server. The client repeats this steps until all test cases are executed. We released the implementation including all test cases[1].

4.3 Results: Firewall Evolution over Time

Public cloud providers constantly improve their offer, and might also alter firewall capabilities in this processes. Thus, we ran our experiments twice, and compare today's results (2016) with past ones from the year of 2014. Thereby, we identify various modifications. In particular, Subsection 4.3.1 focuses on cloud-based firewalls' default configuration, Subsection 4.3.2 explains the utilized test cases for probing using our firewall test tool, and finally Subsection 4.3.3 analyzes the results.

[1] `https://gitlab.sba-research.org/johanna/cloud-firewall-monitoring-tool`

4.3.1 Default Setup at Major Providers

In this step, we investigate the availability of firewalls and their scope of functionality. Out of the providers in Table 4.1, Rackspace and IBM (Softlayer) could not be tested as they do not offer a firewall, the subject of this paper. We looked at Amazon's Elastic Compute Cloud (EC2), Microsoft's Azure Cloud Platform and Google's Compute Engine. It has to be noted though that Azure offers now two deployment models – *Azure Classic* and *Azure Resource Manager*. New virtual machines should be launched into the more recent *Resource Manager* mode, while the old machines remain in *Classic* mode if not migrated.

The latter – EC2, Azure and Compute Engine – provide firewalls to protect machines, groups of machines or virtual machines. However, only Compute Engine names them *firewall*, EC2 goes for *security groups* and Azure Classic for *endpoints*. Azure Resource Manager additionally provides *network security groups*. Thus, we include them separately into our considerations. All of them are now configurable via the web interface that also allows to instantiate virtual machines; but vastness of the interface makes finding configuration sometimes difficult. Adequate documentation is provided, mostly in recipe style and by means of practical examples.

By default, all offer good security by vastly blocking inbound traffic, preventing unsecured machines being exposed to the Internet unintentionally. Nevertheless, there are subtle differences in the inbound configuration as cloud instances have to be remotely accessed anyway: Amazon EC2 and Microsoft Azure solely opens port 22 for SSH in case of Linux instances; alternatively, port 3389 for Remote Desktop Protocol (RDP) in case of Windows instances. Google Compute Engine opens both ports independently of the respective instances. In all cases, rules can be added to allow certain transport layer protocols, ports and protocols. Although it varies whether single values or (IP/port) ranges are allowed.

Amazon as well as Azure's network security groups provide separate configurations on inbound and outbound traffic, while others provide inbound configuration only and in general all kind of outbound traffic is allowed to pass. Amazon provided only inbound configuration in the older EC2-classic configuration as well, so it might also be included in the future at the other providers. Although outbound traffic is configurable, the default configuration of Amazon and Azure Resource Manager allows any kind of outgoing traffic. Beyond, Amazon is the only provider giving the advice to restrict SSH connections to your own IP address when instantiating a new virtual machine, and provides a mechanism for automatic detection therefore.

Azure now not only provides endpoints as a means of protection, but also network security groups that contain rules for the protection of a virtual network. While they were only controllable in a programmatic way at the end of 2015, they are now also accessible via the web interface. By default no network security group is present, and protection is thus solely provided by endpoints. An overall overview can be seen in Table 4.2.

Provider	Firewall available	Web interface	Documentation available	Inbound/outbound separated	Configuration parameters	Default inbound config	Default outbound config
Amazon EC2	✓[a]	✓	✓	✓	transport layer protocol[b], port (single or range), source (anywhere or single IP)	SSH (TCP, 22) from anywhere[c])	* to anywhere
Google Compute Engine	✓	✓	✓	✗	transport layer protocol, port (single or range), source (anywhere, single IP or range)	ICMP SSH (TCP, 22) from anywhere RDP (TCP, 3389) from anywhere UDP and TCP (all ports) from same /16	* to anywhere (not configurable)
IBM (Softlayer)	✗						
Microsoft Azure (Classic)	✓[d]	✓	✓	✗	transport layer protocol port (single) source (range)	SSH (TCP, 22) from anywhere	* to anywhere (not configurable)
Microsoft Azure (Resource Manager)	✓[e]	✓	✓	✓	priority (of a rule) transport layer protocol port (single or range) source and destination (anywhere or range) action (allow or deny)	SSH (TCP, 22) from anywhere	* to anywhere
Rackspace	✗						

[a] called *security groups*
[b] Availability of pre-configured rules for popular protocols, i. e., DNS, SMTP, HTTP, etc.
[c] Amazon suggests to change the address to your own IP after machine launching. A mechanism for detecting this IP is provided.
[d] called *endpoints*
[e] called *network security groups*

Table 4.2: Default Settings of Firewalls from Major Cloud Service Providers

This chapter concentrates on *provider provided software firewalls*. Some providers, such as IBM Softlayer, offer the option of a dedicated hardware firewall, which is beyond the scope of this paper and has more in common with traditional enterprise networks than the cloud environment. Similarly, we are not looking at local, instance or operating systemlevel firewalls, an area covered by traditional firewall products such as iptables and other commercial offerings. We are primarily interested in the new challenges that the cloud brings to security, and how provided perimeter firewalls can work towards mitigating these threats.

Finally, let us conclude the changes since our first measurements in 2014:

- Amazon EC2 provides functionality to automatically determine one's own IP address – a valuable support for firewall configuration for less experienced users. Amazon's intention is to restrict SSH/RDP access for remote maintenance.
- Google Compute Engine's firewall is now also configurable by means of the web interface. Previously, it was necessary to use Google Cloud SDK via the terminal. This change makes control easier for users without or limited command line experience.
- In its new deployment model, Microsoft Azure provides network security groups in addition to endpoints. They allow separate inbound/outbound configuration. While they had to be configured by means of the Azure command line in the past, they are now accessible via the web interface.

4.3.2 Test Cases

For our test tool, see Section 4.2, we implement 26 test cases according to Table 4.3. With these test cases, we determine the responses of firewalls to specific inputs, in order to classify them and to examine how they would respond to certain well known security threats. The test cases are separated in groups A to E depending on the functionality. Some test scenarios, e. g., 1 or 7, are benign scenarios to test for functionality and connection. For a detailed explanation, we refer to the test cases provided with our testing tool. We chose to examine the firewalls' responses in the following areas:

A: Internet Protocol (IP): Test cases 1 to 6 cover basic aspects of IP and a variety of illegitimate field combinations. These scenarios test the extent of packet investigation on the network layer. At present, we limit the test cases to IPv4 as IPv6 is not widely supported in public clouds. By now, IBM Softlayer and Rackspace – both not offering firewalls – are the only providers supporting IPv6 natively at their virtual machines.

B: Fragmentation: The firewall responses to fragmented packets, both normally generated and "malicious" overlapping packet fragments, were measured in test cases 7 to 11. The same operating system was used for all tests, and kernel level measuring tools were employed to ensure that the firewall, not the operating system, was responsible for

the observed behavior. We used both, fragmented ICMP and UDP packets, except for Azure where only UDP was used, as Azure does not permit ICMP traffic at all.

C: Basic TCP and UDP: Group C scenarios (test cases 12 to 17) cover basics aspects of the transport-layer protocols TCP and UDP, i. e., invalid source and destination ports as well as invalid checksums.

D: TCP Flagging: Group D (test cases 18 to 23) contains a number of packets with TCP flag combinations, tested without a previously established connection. From these results, we infer whether there are checks on absurd combinations in absence of an established connection.

E: Stateful Behavior: Stateful behavior, i. e., in combination with an established connection, is tested with test cases of group E (test cases 24 and 25). In general, all stateless test cases should be repeated within a connection. In our first measurements, we saw at a very early stage that stateless behavior is not prevalent, and thus limited the number of developed test cases.

F: Application Layer: The last group F targets the application layers firewalls. We target HTTP as it seems to be one of the most heavily used protocols in clouds.

These areas have been well addressed by traditional firewall products.The absence of any form of firewall logging means that the testing took on a black box approach, with limited information available even with full access to the firewall configuration menus.

4.3.3 Firewall Responses over Time

We ran the test cases twice, once at the end of 2014 and now (mid 2016); results of both runs are presented in Table 4.3. We used the firewall default configurations with slight adaptions: First, we allowed TCP traffic for our testing tool's synchronization, and second we opened another port, both for TCP and UDP traffic, for probing. We chose port 100 (or alternatively port 3333) for the first, and port 6666 for the latter. Where possible, we allowed ICMP traffic to pass. In the following paragraphs, we describe our results, and especially highlight differences to our results from the first measurements, as they indicate further development of cloud firewalls.

A: Internet Protocol (IP): Firewalls filter all these packets as expected; however, with a single exception. Azure generally disallows ICMP (see test case 1) without any possibility to opt-in. The other providers allow ICMP traffic; Google by default and Amazon offers respective configuration possibilities. The results of these test cases imply that packet investigation on the network layer is not only present, but also successfully filters malformed packets. We did not find any differences between our results from 2014 and now. All these test cases were originally written using ICMP; however, Azure's

ID	Name	Amazon		Google		Azure Classic		Azure Res. Mgr.
A	**IP**	**14**	**16**	**14**	**16**	**14**	**16**	**16**
1	Valid ICMP Request	✗	✗	✗	✗	✓	✓	✓
2	Checksum invalid	✓	✓	✓	✓	✓	✓	✓
3	Invalid packet length	✓	✓	✓	✓	✓	✓	✓
4	Invalid header length	✓	✓	✓	✓	✓	✓	✓
5	Reserved flag unequal zero	✓	✓	✓	✓	✓	✓	✓
6	IP protocol number unequal ICMP, TCP or UDP	✓	✓	✓	✓	✓	✓	✓
B	**Fragmentation**							
7	Benign Fragmentation	✗	✗	✗	✓	✗	✓	✗
8	Overlapping Fragmentation	✗	✗	✗	✓	✓	✓	✓
9	Overlapping Fragmentation without Terminating Fragment	✓	✓	✓	✓	✓	✓	✓
10	Overlapping Fragmentation in Reverse Order	✗	✗	✗	✓	✓	✓	✓
11	Tiny Fragments	✗	✓	✗	✓	✗	✓	✓
C	**Transport Layer Protocols**							
12	Invalid Source Port (TCP)	✗	✗	✗	✗	✗	✗	✗
13	Invalid Source Port (UDP)	✗	✗	✗	✗	✗	✗	✗
14	Invalid Destination Port (TCP)	✓	✓	✓	✓	✓	✓	✓
15	Invalid Destination Port (UDP)	✓	✓	✓	✓	✓	✓	✓
16	Invalid Checksum (TCP)	✗	✗	✗	✓	✗	✗	✗
17	Invalid Checksum (UDP)	✗	✗	✗	✓	✗	✗	✗
D	**TCP Flags**							
18	Null Packet (no flags)	✗	✗	✗	✗	✗	✗	✗
19	SYN, FIN	✗	✗	✗	✗	✓	✓	✓
20	SYN, FIN, PSH	✗	✗	✗	✗	✓	✓	✓
21	SYN, FIN, RST	✗	✗	✗	✗	✓	✓	✓
22	SYN, FIN, RST, PSH	✗	✗	✗	✗	✓	✓	✓
23	FIN	✗	✗	✗	✗	✗	✗	✗
E	**Stateful Behaviour**							
24	SYN in Established Connection	✗	✗	✗	✗	✗	✗	✗
25	ACK without ACK Number	✗	✗	✗	✗	✗	✗	✗
F	**Application Layer**							
26	Improper HTTP Request	✗	✓	✗	✓	✗	✓	✓

Table 4.3: Test Cases and Results per Cloud Provider (✓ filtered, ✗ unfiltered)

policy forced us to rewrite them (as well as those for fragmentation, see next paragraph) for UDP as a transport layer protocol; we provide both to our readers.

B: Fragmentation: While benign fragments (test case 7) passed the firewall at all providers in 2014, they do not pass in Google and Azure at the moment; unfortunately, we could not identify the exact reasons from the network traces but consider scapy's fragmentation functionality as root. Overlapping fragmentation are mostly filtered; however, the responses differ with overlapping fragments which are terminated (test case 8). While Amazon still lets them pass, Google and Azure filter them. Their actions are consistent when the fragments are received in reverse order (test case 10). Looking at the packet captures from the receiver, we saw that fragmented packets are reassembled before reaching the virtual machine, i. e., by the firewall or the hypervisor. We believe that this is done for performance reasons. Changes are prevalent for tiny fragments, i. e., packets that are reassembled from a single fragment; they are now filtered at all providers.

C: Basic TCP and UDP: Test cases 12 to 17 indicate checks of transport layer protocol fields. Invalid source ports, i. e., zero, is allowed to pass at all providers, invalid destination ports are filtered. The latter behavior appears obvious when following a port-based filtering approach as there is no rule for this port. While invalid checksums were allowed to pass at our first measurements; Google seems to check them now. While transport layer checks were absent in 2014; at least Google seems willing to introduce such checks now.

D: TCP Flagging: Test cases 18 to 22 test absurd flag combinations, provider reactions differ. While Amazon and Google let them pass, Azure Classic filters all expect the null packet (test case 18). All providers let FIN packets (test case 23) pass albeit there is no established connection; such FINs appear however meaningless without the latter. In conclusion, Amazon and Google do not appear to check flags at all, Azure to a certain extent. Beyond, we did not identify any changes in comparison to our first measurements.

E: Stateful Behavior: All firewalls allow sending SYN flags in an established connection as well as packets containing the ACK flag without an acknowledgement number, i. e., there are no changes in comparison to our previous measurement. In addition, it appears as there is no stateful behavior at any of the providers.

F: Application Layer: In our previous measurements, no signs of an application layer firewall were found. Now, it seems that all check for invalid HTTP requests. This might be an indicator that application layer firewalls have been implemented meanwhile. However, we saw that Amazon still accepts the header when sent within an established TCP connection; thus, the results might also be caused by some sort of statefulness.

Additional Findings: Each platform disallows the use of certain protocols: Google's Compute Engine does not allow SMTP (port 25) or SMTP over SSL (465 and 587);

Azure does not permit ICMP packets to be sent or received. In a similar way, EC2 also blocks sending of SMTP mail by default, though this can be enabled by submitting a support request and using a set of Amazon APIs (SES). None of the services responded to either crafted UDP packets or to a UDP scan using the Nmap port scanning tool. The result were the same no matter if the ports were opened or closed in the firewall. We see from this that the firewall does not send an ICMP "Port Unreachable" notification when the port is closed. This is expected for Azure, which completely disallows the use of ICMP, but perhaps less so for Amazon EC2 and Google Compute Engine.

4.4 Results: An In-Depth Look on Responses

On the one hand, comparison of firewall functionality over time has shown certain improvements; on the other hand, certain functionality has remained unchanged over the last years. Thus, we decided to extend our work and investigate the following issues in more details. First, we focus on overlapping fragmentation. While the firewalls appear to work well at the very first glimpse, there might be more subtle differences and shortcomings, see Subsection 4.4.1. Then, we investigate TCP flag-based filtering in more detail, see Subsection 4.4.2. Finally, we take a look on unknown TCP options, and Path MTU Discovery, see Subsection 4.4.3.

4.4.1 Fragmentation

According to the results of our first measurements, see Subsection 4.3.3, overlapping fragmentation appears to be filtered and thus securely handled. In consequence, we aim to take a more detailed look answering the following questions: Are all fragments filtered? Are fragments already assembled when reaching the virtual machine, and how are they assembled? Therefore, we implemented 22 additional test cases according to the ones proposed in [125]; however, translated to IPv4.

In principle, every test case consists of three fragments:

- The first fragment consists of 24 bytes; thereof, the first eight bytes form the ICMP header, or alternatively UDP if ICMP is prohibited. The remainder is a fixed string. Its *More Fragments* (MF) flag is set as there are further fragments underway.

- The third packet has a fragmentation offset of 24, i. e., it is non-overlapping with the first fragment, and could form a legitimate packet with the first fragment. Its MF flag is never set as it is the last fragment.

- The second fragment is sent in-between; and overlaps with the first and/or the third fragment. Its fragmentation offset as well as its length varies; and so does its MF flag. Every test case is first executed with the second fragment's flag unset; and then with the flag being set.

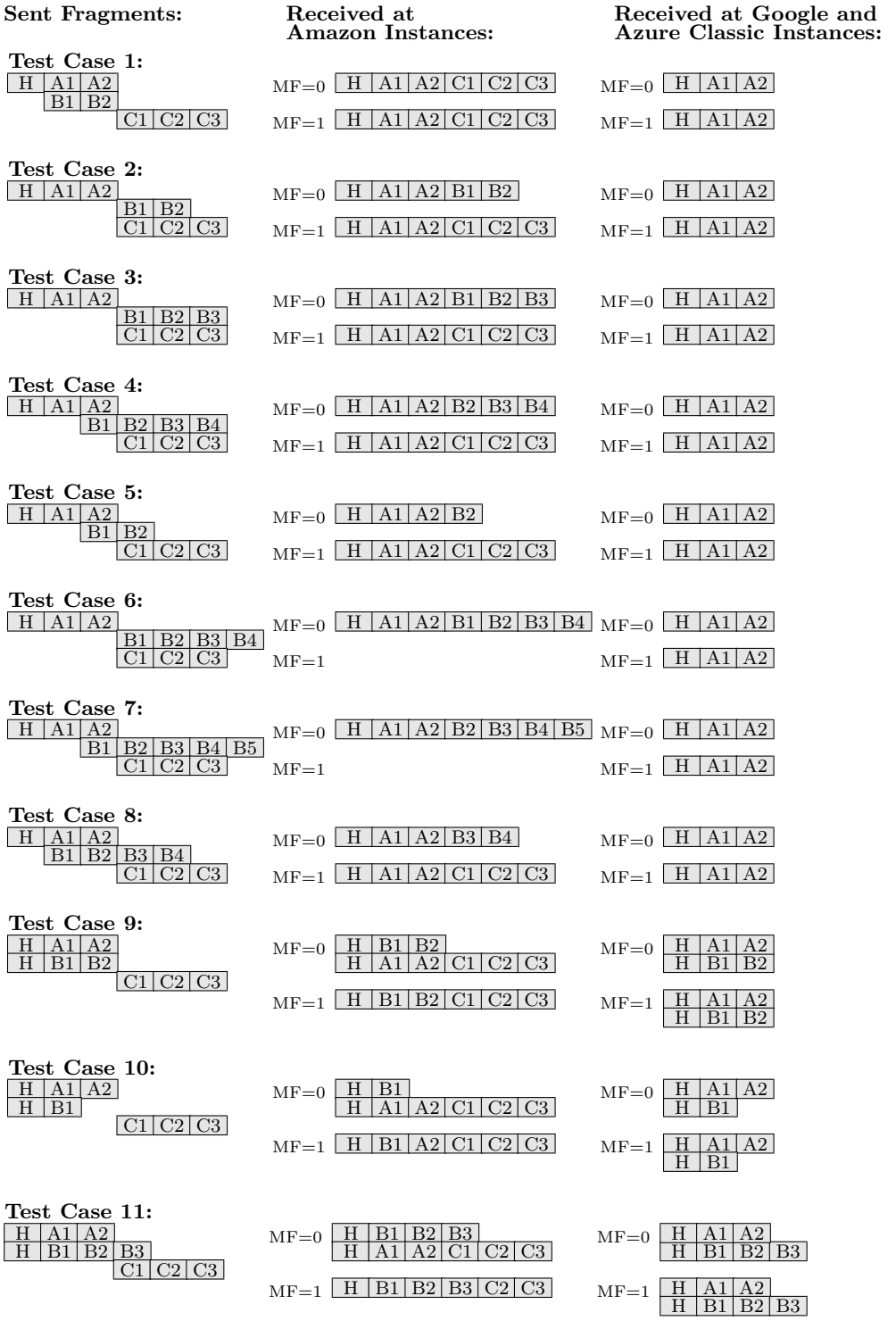


Figure 4.4: Fragmentation in Amazon, Google and Microsoft Clouds

The test cases as well as the results are shown in Figure 4.4. Every block represents eight bytes; H an ICMP or UDP header, the remainder letters indicate payload. The columns show the captured packets at Amazon, Google and Azure Classic instances.

Google and Azure Classic instances show the same behavior: It appears that they drop fragments when detecting overlapping ones; an exception are solely those with their own header having a fragmentation offset of zero. In addition, fragments following such overlapping fragments are also filtered; the results never show a part of the third fragment. There is further no difference between test cases with a set/unset MF flag. Amazon instances show different behavior. First, there are two test cases where all fragments are filtered (test case 6 and 7, each with MF is set). Second, the first fragments appear to be preferred over later ones; these with an unset MF flags are preferred over those with a set flag.

With respect to fragmentation, Azure Resource Manager shows different characteristics than its classic counterpart (not present in the figure). First, it is the only virtual machine where we see (not assembled) fragments in our network captures, i. e., neither the hypervisor nor the firewall seem to reassemble them and deliver them as-is to the virtual machine. Second, it always delivers the first and the second fragment; the third is delivered in all use cases except 1 to 8, each with MF unset. In consequence, we assume that all fragments before the first with MF is zero are allowed to pass.

4.4.2 TCP Filtering

Our previous results imply the absence of TCP flag-based filtering at Amazon and Google; only Azure seemed to check to a certain extent. Nonetheless, we aim to expand our knowledge on the exact details of filtering, and created 128 additional test cases. In a first step, we probe all flag combinations, i. e., 64 test cases as a consequence of six flags, without a previously established TCP connection. In the ideal case, only packets with a set SYN flags should be allowed to pass in order to establish such a connection. In a second step, we repeat these tests, but establish the connection before by means of a TCP handshake.

The results are shown in Table 4.4. As predicted, Amazon and Google do not appear to check the flags at all. However, Azure shows not only filtering in the absence of a connection, but also stateful filtering behavior – contradicting our forecasts based on our first measurements. In the first case, it filters all except the following four flag combinations (1) S, (2) PS, (3) US, and (4) UPS. This can be seen as a secure configuration. In the latter case, it filters 24 combinations while letting the remainder 40 pass. These filtered cases are combinations of SYN and FIN, and/or SYN and RST. These are frequently used for port scanning attempts, and are thus of utter importance to be filtered. Some remaining combinations do not make sense in a TCP connection, e. g., the combination of FIN and RST; however, they appear less threatening in comparison to the combinations mentioned above. The provider might thus have chosen to let them pass.

	Amazon and Google		Azure (both)	
Connection est.	no	yes	no	yes
(null)	✗	✗	✓	✗
F	✗	✗	✓	✗
S	✗	✗	✗	✗
SF	✗	✗	✓	✓
R	✗	✗	✓	✗
RF	✗	✗	✓	✗
RS	✗	✗	✓	✓
RSF	✗	✗	✓	✓
P	✗	✗	✓	✗
PF	✗	✗	✓	✗
PS	✗	✗	✗	✗
PSF	✗	✗	✓	✓
PR	✗	✗	✓	✗
PRF	✗	✗	✓	✗
PRS	✗	✗	✓	✓
PRSF	✗	✗	✓	✓
A	✗	✗	✓	✗
AF	✗	✗	✓	✗
AS	✗	✗	✓	✗
ASF	✗	✗	✓	✓
AR	✗	✗	✓	✗
ARF	✗	✗	✓	✗
ARS	✗	✗	✓	✓
ARSF	✗	✗	✓	✓
AP	✗	✗	✓	✗
APF	✗	✗	✓	✗
APS	✗	✗	✓	✗
APSF	✗	✗	✓	✓
APR	✗	✗	✓	✗
APRF	✗	✗	✓	✗
APRS	✗	✗	✓	✓
APRSF	✗	✗	✓	✓
U	✗	✗	✓	✗
UF	✗	✗	✓	✗
US	✗	✗	✗	✗
USF	✗	✗	✓	✓
UR	✗	✗	✓	✗
URF	✗	✗	✓	✗
URS	✗	✗	✓	✓
URSF	✗	✗	✓	✓
UP	✗	✗	✓	✗
UPF	✗	✗	✓	✗
UPS	✗	✗	✗	✗
UPSF	✗	✗	✓	✓
UPR	✗	✗	✓	✗
UPRF	✗	✗	✓	✗
UPRS	✗	✗	✓	✓
UPRSF	✗	✗	✓	✓
UA	✗	✗	✓	✗
UAF	✗	✗	✓	✗
UAS	✗	✗	✓	✗
UASF	✗	✗	✓	✓
UAR	✗	✗	✓	✗
UARF	✗	✗	✓	✗
UARS	✗	✗	✓	✓
UARSF	✗	✗	✓	✓
UAP	✗	✗	✓	✗
UAPF	✗	✗	✓	✗
UAPS	✗	✗	✓	✗
UAPSF	✗	✗	✓	✓
UAPR	✗	✗	✓	✗
UAPRF	✗	✗	✓	✗
UAPRS	✗	✗	✓	✓
UAPRSF	✗	✗	✓	✓

Table 4.4: TCP Filtering Behavior With and Without An Established TCP Connection

In consequence, we summarize our findings with respect to TCP flags:

- Amazon and Google neither check TCP flags in case of an established TCP connection, nor in the absence of such a connection. This means that each and every TCP flag combination is allowed to pass the firewall.

- Azure shows a more sophisticated behavior in both of its deployment models. Without an established connection, it filters all flag combination except S, PS, US, and UPS.

- In comparison to the results of Subsection 4.3.3, we could further discover stateful behavior at Azure. From all (64) flag combination, only 40 are allowed to pass. The filtered appear to prevent various kinds of port scanning.

4.4.3 Further Findings

In the following paragraphs, we discuss three additional findings considering (1) unknown TCP options, (2) Path MTU discovery and (3) ICMP Echo Requests. First, we aim to send a TCP header with an unknown TCP option. The test case included the option type 111 (which is undefined), and ten bytes of payload. Our results show that these option could pass at all tested providers.

Second, Path MTU discovery searches for the Maximum Transmission Unit that is able to pass a network, and is known for causing "black holes" [126] in case of misbehaving firewalls. If the transmitted bytes exceed a network's MTU, a router en-route fragments the latter. However, packets with the DF (Don't Fragment) bit set are not allowed to be fragmented en-route; and thus an ICMP Destination Unreachable (ICMP code 4)/Fragmentation Needed (ICMP type=4) is returned. If a firewall now filters exactly these ICMP message, it breaks Path MTU discovery and creates a black hole for packets. In consequence, firewalls that allow outgoing IP packets with a set DF bit must also allow this certain kind of ICMP message as a response. We performed manual checks leading to the following results: Amazon allows outgoing packets with DF is one as well as incoming ICMP Destination Unreachable messages; Google filters all kind of ICMP messages with type is three (but letting others pass!), and both deployment models of Azure appear to block all kind of ICMP messages, see also Section 4.3).

During this checks, we found another issue of interest. While Azure Resource Managers prohibits Echo Requests from the virtual machine to other targets; Azure Classics seems to let them pass as well as their replies. This implies that there is some statefulness insofar as Echo Replies to outgoing Requests are allowed to pass. In consequence, we re-tried our checks at Amazon Classic for Path MTU discovery answering to previously outgoing UDP packets; however, still without success. In consequence, only Amazon shows behavior that is accordant with RFC 2979 [126].

4.5 Discussion

The cloud providers Amazon Elastic Cloud Compute, Google Compute Engine and Microsoft Azure have implemented firewalls for protection of their customers' virtual machines. Currently, Azure provides two alternative deployment models Classic and Resource Manager; the latter not only provides endpoints, but also (more accurately configurable) network security groups. Current measurements show that quality has increased since our first measurements at the end of 2014; improvements are found with regard to filtering quality and/or usability of firewall configuration. Differences are especially visible with respect to ICMP filtering, fragmentation and TCP filtering.

With regard to filtering, Azure Classic as well as Google have changed their fragmentation rules; and tiny fragments are now filtered at all providers. Further, HTTP requests without a previously established TCP connection are not feasible anymore. Beyond, Google now checks TCP and UDP checksums and filters packets with invalid ones. At the first glance, the more recent Azure Resource Manager seems to behave the same way as its older counterpart Azure Classic; however, fragmentation appears to be handled differently as our more detailed measurements show.

Considering usability, Google Compute Engine caught up and its firewall is now also configurable via the web interface. Previously, the firewall was solely configurable by means of command line instructions and made configuration difficult for less experienced users. We saw a similar change with Azure's network security groups; at the end of 2015, they were just configurable in a programmatic way. Now, they are also easily configurable via Azure web interface.

In a second step, we extended our measurements with respect to three aspects. First, we investigated TCP filtering – both in the absence and presence of an established TCP connection. The results are discouraging as solely Azure provides adequate filtering behavior. In the absence of a connection, just SYN packets (and three of its variants) are allowed to pass. In the other case, it filters all combination of SYN and RST as well as SYN and FIN as they do not make sense in a legitimate connection but are heavily used in port scanning approaches. Amazon and Google neither show some sort of stateful behavior, nor any attempts of filtering absurd flag combinations. We assume that cloud providers deliberately allow a wide range of packets since they do not know in advance what rented machines are used for, and thus refrain from the implementation of stateful behavior. Despite being though from a security perspective, cloud providers might prioritize easy handling over better security. As a consequence, we advise customers to additionally configure further means of protection, e. g., a host-based firewall. This firewall could not only perform statefully, but also include application layer firewalls and deep packet inspection; however, there is the drawback of additional resources use at the customer's expense.

Fragmentation handling shows a more pleasing picture; but again, strategies vary among different providers. Overlapping fragments are filtered at Google and Azure Classic (and these providers might be more vulnerable to denial-of-service attacks in case an

attacker spoofs overlapping fragments); Amazon aims to make meaningful packets from the received packets (and thus might be more vulnerable to fragments overwriting each other). With Amazon, Google and Azure Classic, it seems that some intermediary, i. e., the firewall or the hypervisor, is reassembling before forwarding traffic to the virtual machine as we could not identify any fragments in our network captures. Our measurements at Azure Resource Manager show distinct (not reassembled) fragments; and lets us conclude that there is no reassembling intermediary. The latter means that final reassembly is dependent on the operating system. Path MTU discovery appears to be broken at all providers except Amazon, and appears to be a consequence of ICMP blocking. It appears that this protocol is still considered harmful; especially as Azure aims to block it (almost) totally.

We still believe that logging is one of the larger features missing in today's firewall implementations. As we have not observed any modification in more than a year, we draw the conclusion, that a logging feature is not right at the top of the providers' list of priorities. Nevertheless, we claim that logging would be an significant improvement for overall security.

4.6 Conclusion

This chapter examines firewall implementations in public clouds focusing on the major providers Amazon Elastic Cloud Compute, Google Compute Engine and both deployment models of Microsoft Azure (Classic and Resource Manager). We develop a tool that utilizes firewall tests as side channels in order to gain details of firewall implementations. We took a more detailed look on TCP filtering, fragmentation and Path MTU discovery. TCP filtering is solely present at a single provider, Azure even shows stateful behavior, while the remainder do not filter absurd flag combination at all. Fragmentation is handled by filtering overlapping fragments (Google and Azure), or reassembling "most meaningful" packets (Amazon). Finally, it seems that an intermediary, i. e., the firewall or the hypervisor, are already reassembling the packets before forwarding them to the final virtual machine in most cases. We conclude that the offered firewalls provide a solid base of protection, and advise customers to configure firewalls according to their needs when running cloud instances and additionally deploy host-based firewalls. We believe that there is still room for further gains: all cloud providers could integrate stateful behavior and logging capabilities.

CHAPTER 5

Exploiting Network Rate Limits of the Xen Hypervisor

A key technology in cloud computing is virtualization as provided by the *Xen* hypervisor [46] that enables multiple virtual instances to share a physical server [7]; but at the same time, resource sharing provides opportunity for adversarial virtual machines to launch attacks against its neighbors. For example, side channels exploiting shared hard disks [35] or network capabilities [37] allow to check for co-residency of two virtual machines; data might be leaked from one virtual instance to another via covert channels exploiting CPU load [68] or cache misses [69]; an instance might free up resources for itself when tricking the neighbor into another resource's limit [48]; and shared network interfaces allow to infer a neighbor's networking behavior [70, 71]. Mitigation follows two principal directions: On the one hand, dedicated hardware eliminates mutual dependencies and thus the threat of co-residency, but contradicts cloud computing's premise of resource sharing. On the other hand, isolation reduces the impact of a virtual machine's behavior on its neighbors despite resource sharing. With respect to networking, rate limits are introduced as means of isolation in order to throttle a virtual machine's maximum amount of traffic per time interval. This approach is considered to guarantee fair distribution of bandwidth among virtual instances and mitigates denial-of-service of neighbors in case a single instance (accidentally or maliciously) requests all bandwidth. The *Xen* hypervisor provides such a rate limiting functionality [127].

The introduction of a countermeasure should raise the question whether it does not form a new attack vector itself. Throttling network traffic however seems to be such a universal approach that its implementation into the *Xen* hypervisor is barely scrutinized. Solely, [47] investigates rate limiting's quality of isolation; [128] analyzes rate limiting with respect to bandwidth utilization. This chapter overcomes this gap and examines the impact of *Xen*'s rate limiting functionality on security. Our analysis reveals that rate limits might protect from co-residency threats, but allow (yet unknown) attacks that

are directed against the rate limited virtual machine itself. In particular, we propose a side channel and a denial-of-service attack. The side channel reveals *Xen*'s configuration parameters that are related to the rate limiting functionality, while the denial-of-service attack causes up to 88.3 percent of packet loss or up to 13.8 seconds of delay in benign connections. Our results emphasize that *Xen*'s rate limiting snaps back, and revision should be considered.

The remainder of this chapter is structured as follows: Section 5.1 provides details on *Xen*'s networking in general and its rate limiting functionality in particular, whereas Section 5.2 analyzes this mechanism with respect to security. Section 5.3 presents our side channel revealing configuration parameters and respective measurement results; Section 5.4 presents three flavors of our denial-of-service attacks and discusses them with respect to their impact on benign connections. Overall results are discussed in Section 5.5. Section 5.6 concludes.

5.1 Background

This section first provides a general overview on Xen's networking architecture. Its rate limiting functionality however throttles only a virtual machine's outbound traffic; thus, we describe a virtual machine's outbound traffic path in a second step. Finally, we focus on the credit-based algorithm eventually throttling a machine's traffic.

General Networking Architecture: The Xen hypervisor follows the approach of paravirtualization; it provides device abstractions to its virtual machines – in terms of Xen virtual machines are called *domains* – so that all sensitive instructions like those for device I/O are redirected over the hypervisor. Paravirtualizing hypervisors do not need specific hardware capabilities; but require modifications of the operating systems running in the virtual machines [46]. With respect to Xen, the hypervisor in the narrower sense is responsible for CPU scheduling, memory management and interrupt forwarding. The remainder tasks are delegated to *domain0* – a privileged virtual machine with the right to access physical I/O devices and to interact with other (non-privileged) domains. Abstract networking devices consist of two distinct parts: (1) *netfront* devices are provided to non-privileged domains replacing classic networking interfaces; (2) its counterpart *netback* resides in *domain0*, multiplexes packets from multiple *netfront* devices and forwards them to the physical network interface card as in standard Linux operating systems [47, 129].

Outbound Traffic Path: Packets originating from non-privileged virtual machines (*domainN*) have to pass *domain0* on their way to the physical network; the respective handover path is depicted in Figure 5.1. Therefore, Xen provides descriptor rings, i. e., ring buffers, as central points of communication. The ring does not directly contain data; this data is rather stored in buffers that are indirectly referenced via the ring descriptors. Packets pass this path in the following manner. First, packets are enqueued in the virtual machine's network interface TX queue. Then, netfront forwards these packets from the

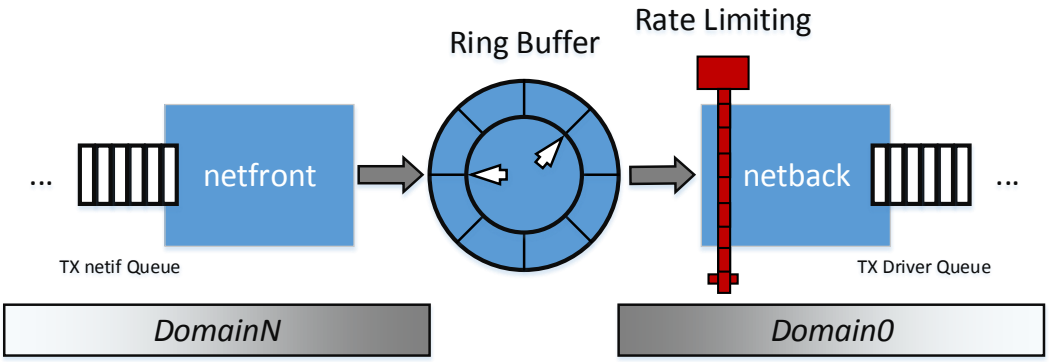


Figure 5.1: Xen's Outbound Traffic Path

TX queue to the ring buffer, and notifies netback. Netback – being within *domain0* and thus having access to physical drivers – hands them over to the physical network interface card's driver queue and removes them from the ring buffer. Beyond, netback is the place of rate limiting. If a respective virtual machine exceeds its assigned bandwidth quota, netback refrains from taking further packets from the ring buffer and discontinues forwarding for some time. As items are not removed from the ring anymore, the buffer becomes full. As soon as a virtual machine's netfront detects this, it signalizes this fact to the upper networking layers by means of a flag. Packets pending in the ring buffer have to wait for further processing until the next bandwidth quota is received.

Rate Limiting: Rate limiting throttles a virtual machine's bandwidth – however, it confines outbound traffic only – and is configured by means of two parameters [127]. The parameter *rate* defines the respective bandwidth limit in MB/s, while *time window* defines the replenish interval of the rate limiting algorithm. Its default value is 50 ms. Looking behind the scenes, the algorithm is credit-based[1]. With every packet forwarded from the ring buffer, the respective packet size is subtracted from the remaining credit. In case of lacking credits, two alternatives remain: (1) immediate replenishment of credits and continuation of transmission, or (2) discontinuation and waiting for replenishment of credits at a later point in time. Immediate replenishment is only possible if the last replenishment happened at least the time defined by the parameter *time window* ago. In the alternative case, a timer is set to the time of next replenishment, and packet transmission is rescheduled as soon as credits are regained. According to the parameters *rate* r and *time window* t, the credit bytes per interval c calculates to $c = r \cdot t$, and the total amount of available credit is limited to this number. This implies that accumulating unused credits for later transmission is impossible. There is a single exception if c remains

[1]Kernel 3.16.0, /net/xen-netback/netback.c

below 128 kB, i. e., rates of less than 2.5 MB/s, as then jumbo packets might seize up the interface. In such a case, credit accumulation up to 128 kB is allowed.

5.2 Security Analysis

In this section, we perform a manual security analysis of Xen's rate limiting functionality. This analysis reveals distinct characteristics that may serve as attack surface; we describe these characteristics, highlight their implication on security and discuss them with respect to cloud computing. Finally, we provide a high-level overview of our attacks that exploit the found characteristics.

(1) Unidirectional Bandwidth Limits: Xen allows to restrict a virtual machine's outbound bandwidth, but inbound remains unlimited without any chance for change. In consequence, the transmission paths are asymmetric. In principle, asymmetry in bandwidth is a known phenomena, e. g., *Asymmetric digital subscriber line* (ADSL), but Xen's asymmetry appears to contradict its application in cloud services as highlighted by the following analogy. ADSL's asymmetry is concordant with its application in consumer broad-band connections. Consumers typically request more downstream than upstream bandwidth, and thus favoring the first direction (at the expense of the latter) is reasonable. Cloud instances predominantly require higher outbound than inbound bandwidth, e. g., when used as application, web or streaming servers. Xen however performs precisely the opposite and limit's the more utilized outbound direction[2].

Bandwidth is not only unequally distributed, but also differs by magnitudes as in consequence inbound traffic is only limited by the underlying hardware. Outbound bandwidth in public clouds starts from 12.5 MB/s for small cloud instances; assuming a 10-Gigabit physical network in the data center, maximum inbound outperforms maximum outbound bandwidth by a factor up to 100.

(2) Susceptibility to Burst Transmissions: Xen's algorithm is prone to burst transmissions. A virtual machine transmitting high amounts of traffic shoots its wad at the begin of a time slot, and has to wait for new credits then. At the time of replenishment, further packets might already wait for transmission and cause another burst consuming all credits. In consequence, packets experience latencies when pausing for the next slot; however, these latencies are only experienced by outbound traffic due to the unidirectional bandwidth limitation. In case the outbound traffic exceeds the configured bandwidth for a longer period of time, packets might even be dropped: Packets remain in the ring buffer as a result of credit shortage. As a consequence, netfront cannot forward packets to the ring descriptor anymore and causes a growing backlog in the virtual machines TX queue. If the number of packets becomes larger than this queue's size, packets are dropped. By default, time window is set to 50 ms; according to the documentation *"a good balance*

[2]Cloud providers like Rackspace (see https://www.rackspace.com/cloud/servers) or Amazon EC2 (see https://aws.amazon.com/en/ec2/pricing/) typically do not even charge inbound traffic.

between latency and throughput and in most cases will not require changing" [127]. This implies that the credit-based algorithm is rather coarse-grained as time slots in the virtual machine's traffic are of the same order of magnitude as round trip times, and the bursts are externally observable.

Attacks Exploiting Rate Limiting: We found two attacks exploiting these characteristics – a side channel revealing Xen configuration parameters that are related to its rate limiting functionality, and a denial-of-service attack causing significant delays and packets drops in benign connections to third parties. We provide a high-level overview on these attacks, before addressing them in more detail in Sections 5.3 and 5.4.

1. **Side Channel:** Pushing a virtual machine into its outbound traffic limits leads to burst transmissions that can be observed. By measuring time between to bursts, it is possible to infer the parameter *time window* t; by summing up the bytes of a burst, an adversary is able to infer the amount of credits c per interval and subsequently calculate the *rate* r.

2. **Denial-of-Service Attack:** An adversary might force a virtual machine to spend all its credits; in consequence, a virtual machine has not enough credits left in order to serve benign requests. Respective responses are significantly delayed as they have to wait for credit replenishment, or dropped due to full buffers. This denial-of-service attack is insofar remarkably as it exhausts outbound bandwidth in comparison to ordinary bandwidth exhaustion attacks exhausting inbound bandwidth.

5.3 Side Channel

If a virtual machine requires more bandwidth than assigned, its traffic becomes bursty due to Xen's credit-based rate limiting algorithm. An adversary might exploit this behavior to determine a virtual machine's configuration parameters *time window* t and *rate* r by means of the following side channel. The adversary sends a high number of legitimate requests to the virtual machine. The latter replies according to the chosen protocol; however, the sum of all replies exceeds the assigned bandwidth and outbound traffic becomes bursty as depicted in Figure 5.2. The time interval between two bursts is equivalent to the configured *time window* t, as the virtual machine receives credits for further transmission immediately after the timer expires. Summing up the size of all packets within a burst allows to determine the victim's *credit rate* c. Finally, the adversary is able to calculate the victim's assigned bandwidth (parameter *rate*) $r = c/t$. The side channel is advantageously protocol independent. The only stringent objective is that the virtual machine reliably replies; thus, a wide variety of protocols are worth considering, e. g., ICMP, DNS, etc. The more outbound traffic, the better; the larger the amplification between outbound and inbound traffic, the better; both facilitate to reach the assigned rate limit for outbound traffic.

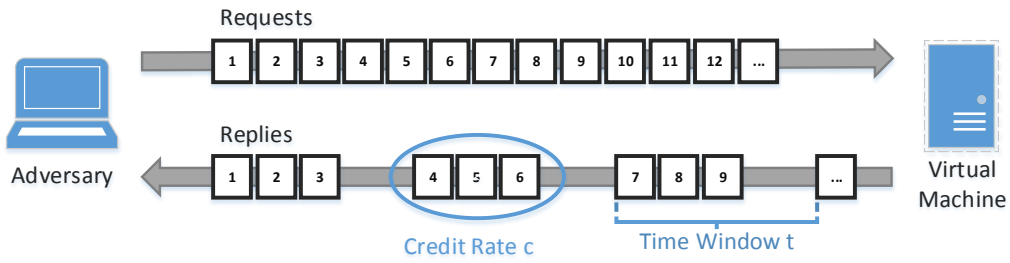


Figure 5.2: Side Channel Attack Scenario

We evaluated this side channel in our experiments. For our experiments, we use Xen version 4.4.1 (on Debian 8.2) on an Intel i5-750. On the hypervisor, two virtual machines run Debian 7.9; each guest is pinned to a separate CPU, *domain0* runs on the remaining two CPUs. The two virtual instances were rate limited and bridged via the hypervisor. The adversary ran Debian 8.2 on an Lenovo X200 laptop. The hypervisor and the adversary's laptop were connected via a 1 Gbit/s network switch. The virtual machine was limited to 5 MB/s at the default time window of 50 ms. Checking the configuration with *iperf*[3], we measured 4.7 MB/s from the virtual machine to the adversary (throttled outbound traffic), and 117.3 MB/s in the other direction (unthrottled inbound traffic). Attacking the virtual machine, the adversary sent 16 ICMP Echo Request of length 1458 bytes, waiting for a millisecond before sending the next 16 ICMP Echo Requests causing up to 22.2 MB/s of inbound traffic for the virtual machine. In total, the attack runs for 1000 of such cycles sending in total 16000 Echo Requests. Repeating this attack ten times, we inferred the configuration parameter from the measurements according to the following approaches:

- **Time Window:** The begin of a time window is indicated by a packet following a (larger than usual) pause. Thus, we extracted all packets following a pause of at least 5 ms, and measured the time window between these first packets of subsequent bursts. Rounding off to whole milliseconds, we took the most frequent candidate of all test runs.

- **Credit Rate:** In the previous step, the first packets of bursts have already been determined; the credit rate is now calculated by summing up the size of all packets from this first packet of the burst to the last one. The last packet of the burst is the one right before the first packet of the next burst. Again, the most frequent candidate is taken from all candidates.

[3] https://iperf.fr/

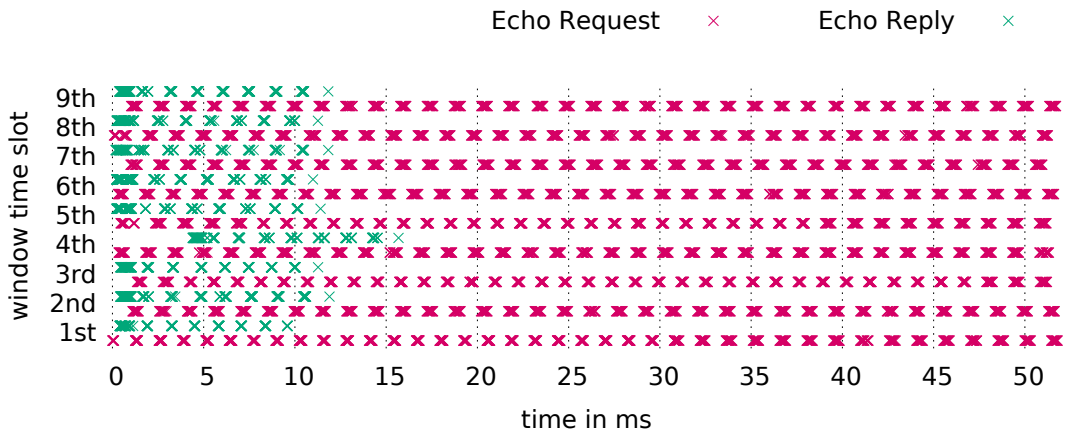


Figure 5.3: Side Channel Measurement Results using ICMP

Xen Configuration		Attack Parameters		Side Channel		
Configured Bandwidth	iperf (Outbound)	Requests per Cycle	Inbound Bandwidth	Credit Rate c (Measured)	Time Window t (Measured)	Rate r (Calculated)
MB/s	MB/s		MB/s	B	ms	MB/s
5	4.7	16	23.3	249318	52	4.8
10	9.4	32	46.7	498636	52	9.6
20	18.9	32	46.7	998730	52	19.2
30	27.8	48	70.0	1498824	56	26.8
40	37.0	60	87.5	1998918	52	38.4

Table 5.1: Bandwidth Measurements with Fixed Time Window of 50 ms

This way, we inferred a *time window* t of 52 ms, and a *credit rate* c of 249,318 bytes; the resulting bandwidth r is thus 4.8 MB/s. Figure 5.3 depicts a network trace of the side channel from the adversary's point of view; for reasons of simplicity, the graph is already slotted in time intervals of 52 ms. While the adversary sends requests in regular intervals, the virtual machines replies predominantly at the begin of a time slot. Afterwards, it remains silent due to lacking further credits. One can also see in the figure that the number of sent replies is high at the begin of a time slot; this is an indicator that all waiting replies are sent at once immediately after credit replenishment. The side channel was measured with different configurations of the virtual machine. First, we altered the bandwidth keeping the time window at the default configuration of 50 ms; results are provided in Table 5.1. Then, we modified the time window at a fixed bandwidth of 5 MB/s; results are provided in Table 5.2. The first line of Table 5.1, and the second line of Table 5.2 represents the results of the measurement that has been described above.

Xen Configuration		Attack Parameters		Side Channel		
Configured Time Window	iperf (Outbound)	Requests per Cycle	Inbound Bandwidth	Credit Rate c (Measured)	Time Window t (Measured)	Rate r (Calculated)
ms	MB/s		MB/s	B	ms	MB/s
70	4.8	16	23.3	349920	72	4.8
50	4.7	16	23.3	249318	52	4.8
30	4.6	16	23.3	148716	32	4.6
20	4.9	16	23.3	100602	24	4.2
10	4.1	16	23.3	49572	8	6.2

Table 5.2: Time Window Measurements with Fixed Bandwidth of 5 MB/s

Our results show that the measured time window is slightly longer than the configured time window. Taking a look into Xen's source code, the time window is strictly speaking the time period for the timer; this additional time of mostly 2 ms might be caused by credit replenishment, packet forwarding, etc. that is necessary after the timer expires. Actual bandwidth appears to be below the configuration parameter; however, our side channel appears to reflect *iperf* measurements well. Measurements for 30 MB/s at 50 ms of Table 5.1 shows an increased time window; however, evaluation shows two almost equally frequent candidates – 56 ms and 48 ms – both equally distant from the expected 52 ms. Similarly, measurements for 5MB/s at 20 ms (peaks at 16 ms and 24 ms) as well as 5MB/s at 10 ms (peaks at 8 ms and 16 ms) of Table 5.2 show two such peaks. For the latter however the lower peaks has slightly more candidates. The reason for less quality of the latter two results might be the rather small *time window* t. Pauses before first packets of a burst become shorter with decreasing time windows; thus, our algorithm looking for 5 ms pauses might struggle to detect beginning packets at such low time windows accurately. This might be overcome by looking for shorter pauses.

5.4 Denial-Of-Service

Traffic exceeding the rate limit has to wait for a free time slot in the future; beyond, if the backlog of waiting packets becomes too much, buffers become full and packets are dropped. Deliberately filling the buffers, an adversary might exploit this behavior in order to perform a denial-of-service attack causing significant packet delays or even drops of benign traffic.

For evaluation, we extended the measurement setup by an additional host representing the victim as depicted in Figure 5.4. The victim ran Ubuntu 14.4 LTS on a Lenovo X60 laptop. The virtual machines were rate limited to 5 MB/s at the default window time of 50 ms. The victim had a benign connection to the virtual machine; we decided to probe the virtual machine with ICMP Echo Requests at an interval of 10 ms. In total, these

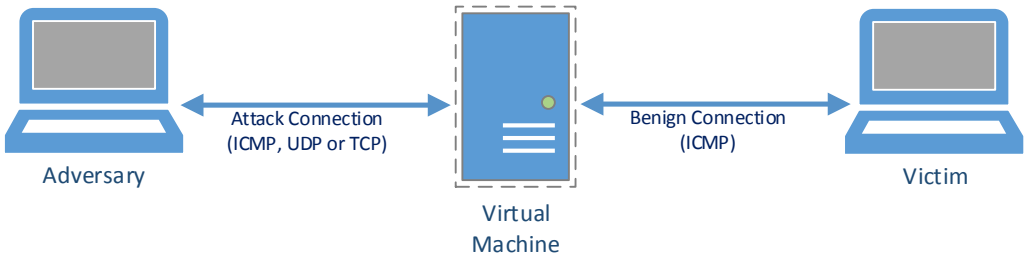


Figure 5.4: Denial-of-Service Attack Scenario

requests (and potentially received replies) require a maximum bandwidth of 19.6 kB/s which is negligible in comparison to the attack traffic. For the adversary, we tested three alternatives for causing the backlog – by means of ICMP Echo Requests, UDP-based traffic amplification and TCP acceleration as described in the following paragraphs.

ICMP Echo Requests: As with the side channel, the adversary sends multiple Echo Requests and pauses afterwards for 1 ms repeating both actions in a loop. Echo Requests however bear the drawback of being non-amplifying, i. e., the virtual machine's (maximum) outbound traffic is of the same amount as the inbound traffic from the adversary.

UDP-Based Traffic Amplification: A virtual machine might host a service that answers with replies that exceed the requests in size and thus amplifies inbound traffic. An adversary sending numerous such requests is able to trigger more outbound traffic than with ICMP. Susceptible protocols are predominantly UDP-based, e. g., NTP, DNS, SSDP or BitTorrent, and bandwidth amplification factors reach up to 4670.0 [130]. In that paper, the authors investigated amplifying protocols with respect to reflective denial-of-service. Such attacks require source address spoofing in order to redirect replies to the victim – a prerequisite that is not necessary for our denial-of-service attack. This implies that (1) there are even more protocols than described in these papers that are susceptible to our attack and (2) ingress filtering does not prevent our attack. For our evaluation, we scripted a simple UDP server that responded with a bandwidth amplification factor of 100. The server ran on the virtual machine; our adversary sent respective UDP requests in the same manner as the ICMP Requests – sending a certain number of UDP requests before pausing for 1 ms repeating both actions in a loop.

TCP Acceleration: TCP connections, e. g., when serving a HTTP request, are frequently asymmetric with respect to transmitted payload; a server is sending amounts of data while the client almost exclusively acknowledges receipt with a couple of bytes. TCP is a reliable protocol, adjust its speed according to given networking capabilities and

	Adversary			Victim		
ID	Requests per Cycle	Inbound Bandwidth	Potential Outbound Bandwidth	Average Delay	Maximum Delay	Dropped Replies
		MB/s	MB/s	ms	ms	%
noattack				0.221	0.491	0
icmp16	16	23.3	23.3	14.3	65.7	67.5
icmp32	32	46.7	46.7	23.6	49.3	85.6
icmp48	48	70.0	70.0	21.8	51.5	83.6
icmp60	60	87.5	87.5	16.7	52.8	88.3

Table 5.3: Denial-of-Service Attack with ICMP Echo Requests

	Adversary			Victim		
ID	Requests per Cycle	Inbound Bandwidth	Potential Outbound Bandwidth	Average Delay	Maximum Delay	Dropped Replies
		MB/s	MB/s	ms	ms	%
noattack				0.221	0.491	0
udp4	4	0.2	23.2	23.0	53.3	0
udp8	8	0.5	46.4	22.9	53.5	0
udp12	12	0.7	69.6	22.7	53.4	0
udp15	15	0.9	87.0	23.6	53.7	0

Table 5.4: Denial-of-Service Attack with UDP-Based Traffic Amplification

thus does not automatically lead into a denial-of-service attack; but an adversary might intentionally accelerate a TCP connection by means of optimistic acknowledgments [131]. Such optimistic acknowledgments are sent prior the receipt of the respective segment, lead the server to believe in higher available bandwidth and make the server send at a higher speed than normally. For our evaluation, we installed an Apache[4] server on the virtual machine providing a 100 MB file for download. For the adversary, we re-implemented this attack with respect to current TCP implementations as congestion control has significantly changed over the last decade and ran the attack when downloading the previously mentioned file.

Results for ICMP and UDP-Based Attacks: Results for the ICMP-based attacks are found in Table 5.3; results for UDP-based attack with traffic amplification in Table 5.4.

[4] https://httpd.apache.org/

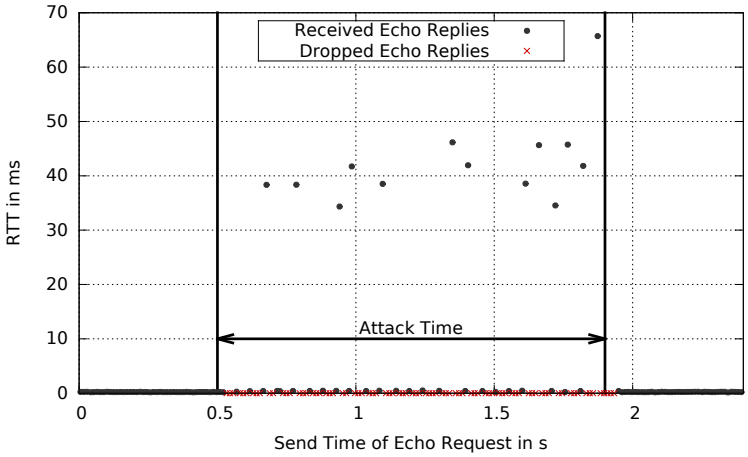


Figure 5.5: Impact on the Victim (icmp16)

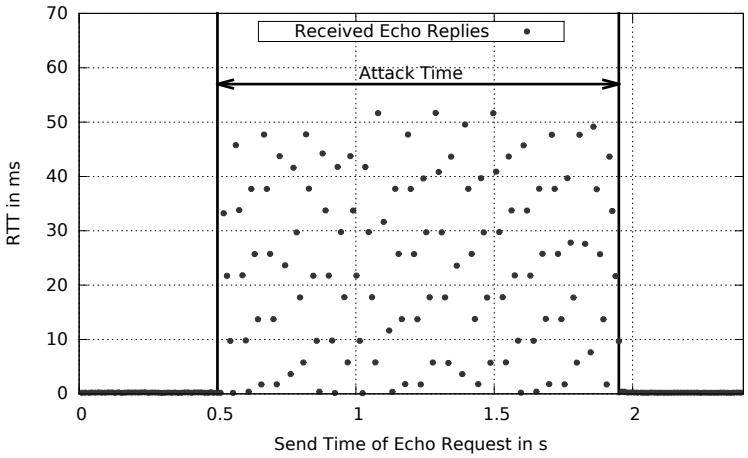


Figure 5.6: Impact on the Victim (udp4)

As in the side channel, the results are based on ten test runs each. Inbound bandwidth refers to the traffic that is sent from the adversary to the virtual machine, while the potential outbound bandwidth refers to the bandwidth that would be caused in the reverse direction in the absence of rate limits. The remaining three columns show the average delay of replies to the victim's Echo Requests, the observed maximum delay as well as the relative amount of dropped replies. In both tables, the first line represents the latter values in the absence of an attack for reasons of comparison.

The results highlight the following: (1) All attacks significantly increase the delays by two orders of magnitudes. (2) The higher the ICMP bandwidth, the more packet drops. The average delay however appears to decrease at higher attack bandwidths; and might be an artifact of increased drop rates as less replies were received by the victim and were taken

	Adversary			Victim		
ID	Requests per Cycle	Inbound Bandwidth	Potential Outbound Bandwidth	Average Delay	Maximum Delay	Dropped Replies
		MB/s	MB/s	ms	ms	%
noattack				0.221	0.491	0
tcp		0.01	65.8	1625.8	13791.6	33.2

Table 5.5: Denial-of-Service Attack with TCP

into account for average delay calculation. The maximum delay of *icmp16* might be higher than the remainder for the same reason. (3) UDP-based attacks do not cause any packets drops; average and maximum delay are higher than in the ICMP-based attacks, and appear to be independent of the attack bandwidth. The reason might be that the virtual machine favors ICMP traffic over UDP, and thus drops attack traffic rather than the victim's. However, amplification allows the adversary to reduce the amount of sent traffic in order to gain the same potential outbound bandwidth at the virtual machine; for example, attack *icmp16* leads to the same potential outbound bandwidth as *udp4*. Figure 5.5 depicts a test run of the ICMP-based attack, Figure 5.6 of the UDP-based attack. Both figures show the increased round-trip times of the victim; the first figure further shows packet drops.

Results for TCP-Based Attack: The result of our TCP attack are found in Table 5.5. In comparison to ICMP- or UDP-based attacks, delays are much higher. The average delay is 1625.8 ms, the maximum delay even 13791 ms, i. e., almost 14 seconds. Packet drops are however below the ICMP-based attack: 33.2 percent.

Figure 5.7 shows the sequence numbers of sent TCP acknowledgments and received TCP payload from the adversary's perspective. While the first increases exponentially to maximize the virtual machine's congestion window, the latter increases only in a linear manner. This linear increase is caused by the rate limit of 5 MB/s, and provides a first evidence that the virtual machine operates at its networking limits. Moreover, enlarged sections of this figure clearly depict the bursty transmission and the underlying roughly 50 ms intervals, see Figure 5.8. Figure 5.9 shows the attack's impact on the victim's round-trip times: Right at the start, round-trip times are as expected less than a millisecond; then, round-trip times start to increase. The maximum recorded delay is 13,791.6 ms. In a third phase, the buffers are full and Echo Replies are dropped at a large-scale. As numerous packets are dropped, the buffer is released and round-trip times decrease back to normal.

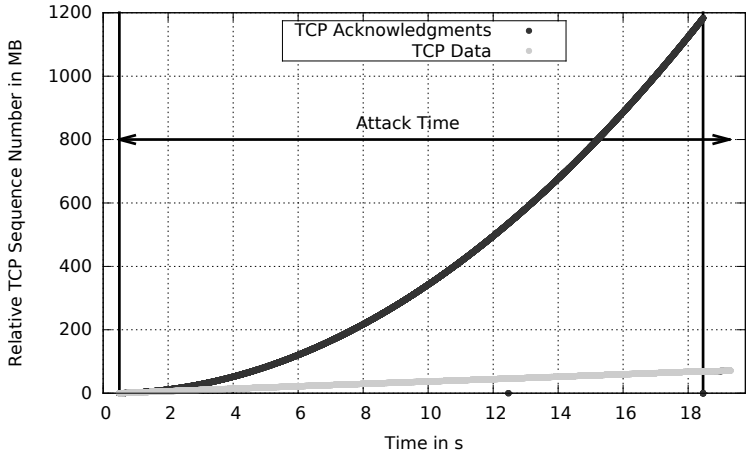


Figure 5.7: Relative Sequence Numbers (tcp)

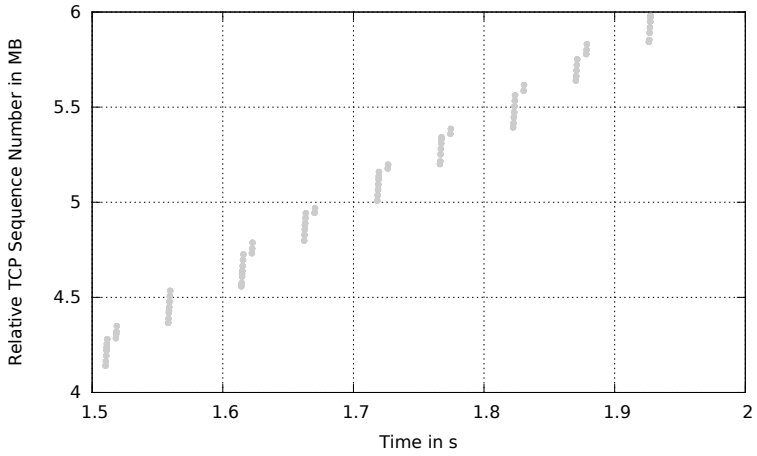


Figure 5.8: Relative Sequence Numbers (tcp) (Enlarged Section)

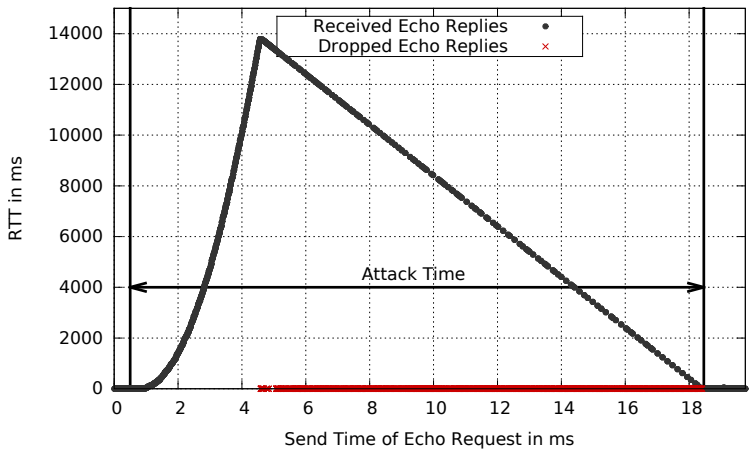


Figure 5.9: Impact on the Victim (tcp)

5.5 Discussion

Throttling network bandwidth hinders a virtual machine from claiming all available resources and cutting off supply to neighbor machines. Such rate limits have always been considered as means of security against denial-of-service attacks, but not as an attack vector themselves. Notwithstanding, our work conveys by the example of the popular Xen hypervisor that (1) configuration parameters of rate limiting are easily gained through a side channel and (2) serve as a base for novel denial-of-service attacks exploiting (allegedly protective) rate limits. In comparison to traditional bandwidth consumption attacks, our denial-of-service attack shows a peculiarity with respect to the point of consumption. A traditional denial-of-service attack jams the virtual machine's *inbound* link, exploiting rate limiting functionality as shown in this chapter causes jam on the *outbound* link. In the first case, a virtual machine would not receive any further traffic and might suspect irregularities on the network. In the latter case, it would still obtain requests and would be (mostly) unaware that responses are stuck in the hypervisor. Digging its own grave, it would even answer incoming requests strengthening the attack. Xen's unilateral bandwidth limits (not throttling inbound traffic) is an additional blessing as requests from the adversary are reliably forwarded to the victim. This means that there is in principle no need for traffic amplification; but admittedly, the attack is more likely to succeed with some sort of traffic amplification, e. g., when striking over the Internet with much lower bandwidth.

Our side channel allows to infer all configuration parameters of Xen's rate limiting – the rate and the window time. On the one hand, this enables an adversary to plan an attack, e. g., our denial-of-service-attack, more accurately. Further, an adversary once knowing these parameters of a virtual machine would be able to glean the latter's networking behavior; but also benign customers might use the side channel to check compliance of the configuration with their service contract. On the other hand, the side channel may also serve as a way to identify the underlying hypervisor of a virtual machine as Xen. Beyond, the side channel has potential to be developed further into a covert channel. A limitation is however given by network jitter as an adversary depends on clear distinction between subsequent time slots. This limitation however is only valid for the side channel, not for the denial-of-service attack.

Our denial-of-service attack causes latencies of up to almost 14 seconds, and packet drops of up to 88.3 percent. Service degradation is generally undesired, for example, it decreases interactivity [132]; but there are also scenarios beyond the obvious that we would like to highlight. First, virtual machines are remotely synchronized by means of a synchronization protocol like Network Time Protocol (NTP) for purposes of time measurements [133, 134]. However, the synchronization algorithm easily looses its stability in case of variable path delays [135], and these delays are heavily increased by our attack for a certain period before going back to normal; further, synchronization is prone to path asymmetry [135], and this asymmetry is also exacerbated by our attack. Synchronization errors are already in the milliseconds in presence of moderate CPU load [134], and will become significantly worse in presence of our attack making accurate time measurements

in the cloud a nightmare. Second, temporal lensing was lately introduced as a way of attacking [136]; thereby an adversary performs a reflective denial-of-service attack that concentrates into a single (short, but high-bandwidth) pulse striking the victim. This is achieved by using reflectors with different attack path latencies, i. e., requesting reflectors with long paths before those with shorter ones, with the goal that all replies reach the victim simultaneously. The more reflectors with higher latencies are found, the more the adversary is able to funnel. The longest path latency found by the authors was 800 ms. In case such a reflector resided on a virtual machine, its responses could be delayed up to almost 14 seconds by hitting this virtual machine with our TCP attack. This approach would significantly increase temporal lensing's power of impact by providing seamlessly controllable reflectors.

Finally, our results provide a further explanation to cloud phenomena: [137] measured TCP and UDP performance in the *Amazon EC2* cloud that is known to use the Xen hypervisor, and identified regular bandwidth drops[5]. They seem to occur roughly every 50 ms, and might be a consequence of rate limiting. This observation might further be an indicator that rate limiting was (and possibly still is) deployed at this major cloud provider; but Rackspace – another public cloud provider also using Xen – might also throttle virtual machines this way as they claim that only outbound traffic is limited [138]. Parenthetically, public cloud providers charge only outbound traffic while inbound remains free. This implies that our denial-of-service attack does not only impact a virtual machine's availability, but also costs the owner actual money and could be used to economically harm somebody.

In consequence, mitigation is of utter importance; however, none of the following suggestions fully prevents our attacks. (1) Throttling inbound traffic as well would only prevent non-amplifying attacks, but might negatively impact a host's availability. However, providers could choose to apply such limits only in the presence of an attack – provided that adequate detection mechanisms are prevalent. (2) A modification of the credit-based scheduler enabling short spikes (by spending previously saved credits) would increase the effort to overwhelm rate limits and buffers for the adversary. (3) Decreasing the *time window t* makes our side channel more prone to jitter (and thus prevent it) as the time slots cannot be clearly distinguished anymore, but would have a negative impact on performance. Alternatively, the algorithm might be modified in order to be less deterministic, e. g., by randomizing the time window.

5.6 Conclusion

Rate limits are known to guarantee fair bandwidth distribution and to prevent denial-of-service attacks among virtual machines on the same *Xen* hypervisor; but our work shows that rate limits themselves become a vector for externally-launched attacks. The underlying reasons are *Xen*'s unidirectional rate limits throttling outbound traffic only, and its susceptibility to burst transmissions. In this chapter, we propose two distinct

[5] See Figure 5 in [137]

attacks exploiting rate limits. Our side channel reveals all configuration parameters that are related to rate limiting functionality and form a substrate for a denial-of-service attack; the latter causes up to 13.8 seconds of packet delay or up to 88.3 percent packet drops. Beyond ordinary service degradation, these latencies may heavily destabilize time synchronization in clouds due to increased path asymmetry and path variability; but may also strengthen temporal lensing attacks due to providing reflectors with controllable path latency. There is indication that popular cloud providers like *Amazon EC2* or *Rackspace* are using *Xen*'s rate limiting; thus, a large number of hosts remains conceivably vulnerable.

Part II

Secret Communication with IPv6

In this part of the thesis, we focus on secret communication related to the Internet Protocol version 6 (IPv6). As in cloud computing, the ability to establish such secret communication provides a powerful instrument to adversaries and can be used for later attacks. Chapter 6 provides a survey on security and privacy vulnerabilities in IPv6. In comparison to cloud computing, secret communication with IPv6 appears to play a minor role; there exists only a handful of approaches. Among these channels, IPv6 addresses seem to be of most interest.

In consequence, the remainder chapters develop scenarios exploiting IPv6 addresses as side channels. Chapter 7 develops a pattern-based approach for reconnaissance that generates potential host addresses for active probing (scanning) as exhaustive search became infeasible with IPv6. Our results show that automatically mining implicit patterns from address data sets outperforms other approaches of address generation. Chapter 8 proves that the IPv6 Privacy Extension bears implicit patterns and is vulnerable to a side channel – contradicting the Privacy Extension's intended goal of privacy protection. An adversary is able to predict future temporary addresses once the internal state is known, and is further able to synchronize to this internal state by exploiting the victim's previous addresses as a side channel.

CHAPTER 6

A Survey on IPv6 Security and Privacy

This chapter summarizes and systematizes IPv6 vulnerabilities as well as associated countermeasures in a nutshell. In addition to scientific papers, we included *Requests for Comments* (RFCs) as well as non-scientific contributions like videos or blogs in order to complete our systematization. Finally, we infer three major challenges for IPv6: secret communication, reconnaissance and addressing. The remainder of this chapter is structured as follows: Section 6.1 introduces IPv6 and related technologies. Section 6.2 summarizes currently known security vulnerabilities, while Section 6.3 considers privacy in relation to IPv6. Section 6.4 presents our systematization of IPv6 vulnerabilities and countermeasures. Finally, Section 6.5 discusses major challenges with respect to IPv6.

6.1 Background on IPv6

In comparison to IPv4, its successor IPv6 encompasses four major modifications: (1) The address length has been quadrupled to 128 bit, providing $3.4 \cdot 10^{38}$ unique addresses. These contain a subnet prefix and an interface identifier, and are represented by 8 quadruples of hexadecimal values separated by colons [139]. (2) Regarding the amount of receivers, three types of addresses are distinguished: *unicast*, *anycast* and *multicast* addresses. There are no *broadcast* addresses in IPv6. (3) The header format has been simplified and is now fixed to 40 byte, as shown in Table 6.1. Fragmentation and other optional functionality has been shifted to *extension headers*, which are inserted between the IP and the upper-layer protocol header. (4) Fragmentation has further been limited to end nodes with the objective of router offloading. (5) Formerly mandatory IPsec [140, 141, 142] is seen as its fifth major modification before being released as optional [143].

With IP being the Internet's main protocol, many constitutive Internet technologies are heavily tied to it and the change to version 6 resulted in updates of related protocols.

Size in Bits	Field Name	Comment
4	Version	set to *6*
8	Traffic Class	replaces *Type of Services*
20	Flow Label	for packet flow marking
16	Payload Length	incl. *IPv6 Extension Headers*
8	Next Header	
8	Hop Limit	replaces *Time to Live*
128	Source Address	
128	Destination Address	

Table 6.1: IPv6 Header Format [144]

One of them is the *Internet Control Message Protocol* (ICMPv6) [145]. In spite of a reduced number of message types, its scope has increased beyond error and diagnostic messages. Performing now also address resolution by means of the *Neighbor Discovery Protocol* (NDP) [146], it is also the successor of the *Address Resolution Protocol* (ARP) and responsible for router discovery.

IPv6 addresses are either configured manually, statefully (such as by *Dynamic Host Configuration Protocol* (DHCPv6) [147][1]), or by the newly introduced *Stateless Autoconfiguration* (SLAAC) [149, 150], providing plug-and-play connectivity. With SLAAC, the host first creates a link-local address on its own. After receiving a *router advertisement*, the node generates global addresses with the announced network prefixes. Recommended network prefix sizes for end sites are between /48 and /64 [151, 152].

Due to the increasing number of mobile nodes, mobility support [153] has gained importance. It allows nodes to remain transparently reachable via the same address while wandering through the network. In case the mobile node is in a foreign network, it provides its actual address to its router by means of a binding update. This provides two possibilities for correspondent nodes to communicate with the mobile node: The communication can be passed on to the home agent, which tunnels the traffic on to the mobile node. Alternatively, route optimization allows direct communication without the home agent by using a certain routing header.

The transition from version 4 to 6 takes time and is accompanied by a phase of coexistence. Some nodes are capable of both protocols, while others are limited to one or the other. Therefore, transition technologies that bridge this gap have been developed; they can be divided into two main types: (1) Tunneling delivers a packet as another packet's payload. [154] provides a general description on tunneling IPv6 over IPv4, while

[1]The stateless DHCP approach is technically speaking not a means of address assignment because it does not maintain a client state [148].

[155] is a specification for tunneling other protocols over IPv6. Currently, there are a high number of different technologies tunneling IPv6 over IPv4: *6to4* [156, 157], *IPv6 rapid deployment* [158, 159], *6over4* [160, 161], *ISATAP* [162] and Teredo [163, 164]. (2) Alternatively, protocol translation, i. e., the translation of IPv4 into IPv6 headers and vice versa, can be used. Due to being tightly connected, IP translation also includes ICMP translation. The first specification *Network Address Translation - Protocol Translation (NAT-PT)* has been criticized by [165, 166] for numerous reasons, e.g. lacking support of DNSSEC. Its successor is standardized in [167, 168, 169, 170, 171]. However, tunneling is currently preferred.

6.2 Security Vulnerabilities

In the course of the development of the new Internet Protocol version, changes in and supplements to functionality were made. These enhancements, however, yield different behavior and therefore often result in novel security vulnerabilities. In this section, we summarize fundamental security vulnerabilities in IPv6 and present feasible countermeasures. We organize them by intended functionality, starting with extension headers, fragmentation and other native header fields. Subsequently, *Neighbor* and *Multicast Listener Discovery* are discussed, followed by tunneling and mobility support.

6.2.1 Extension Headers

Extension headers provide optional functionality and are inserted before the next-layer protocol header. Two of them are of further interest for security: (1) The *routing header type 0* holds a list of addresses that have to be visited en route to the receiver. By alternating the two addresses, the packet cycles between two nodes, causing traffic amplification on a remote path and possibly resulting in denial of service [172]. This extension header was more harmful than beneficial and was finally deprecated [172][2].

Offloading routers was a major focus during development. *IPv6 extension headers* are, therefore, only allowed to be processed at end nodes. The only exception is the *Hop-by-Hop header* and its *Router Alert option*, which may be used for updating in the future. However, this option may also cause a decrease in router performance when many such packets are sent [174].

Initially, extension headers and options did not have to follow a certain format, therefore, middleboxes are not necessarily able to process new extension headers. Later, a uniform format for extension headers was standardized [175].

6.2.2 Fragmentation

IPv6 did not explicitly prohibit the reassembly of overlapping fragments initially despite this being a well-known security threat that can be used, e. g., to evade firewalls [176].

[2] *Routing header type 0* differs from the benign *type 2* [173] used for mobile applications.

The best-known way of doing so is overwriting the TCP SYN flag. The countermeasure in IPv4 was dropping fragments with an offset of one byte [177]. But this is no appropriate mitigation for IPv6 because an arbitrary number of *extension headers* can be inserted prior to the next-layer protocol header and cause any offset.

Such insertions are also able to shift flags or port numbers to succeeding fragments. Common firewalls collect incoming packet fragments and reassemble them in any case, but reassembly implementations differ, making IPv6 vulnerable to the same attack scenarios as IPv4 [177, 178]. These differences in reassembly can also be used to fingerprint operating systems [125].

As a consequence, overlapping fragments are now explicitly forbidden because benign nodes do not have any need of sending overlaps [179]. Further, deep packet inspection should treat initial fragments without flags or port numbers with suspicion as there is a guaranteed MTU (Maximum Transmission Unit) in IPv6. Finally, fragmentation is still a stateful process within a stateless protocol with the risk of memory overflow.

Specific to IPv6 are *atomic fragments*. These packets consist of only one fragment and are used in protocol translation to deliver an identifier for fragmentation in IPv4 [180]. Unfortunately, these fragments can cause dropping of benign fragments that have the same identifier. Thus, the two types of fragments should be handled in isolation from each other.

6.2.3 Mandatory IPv6 Header Fields

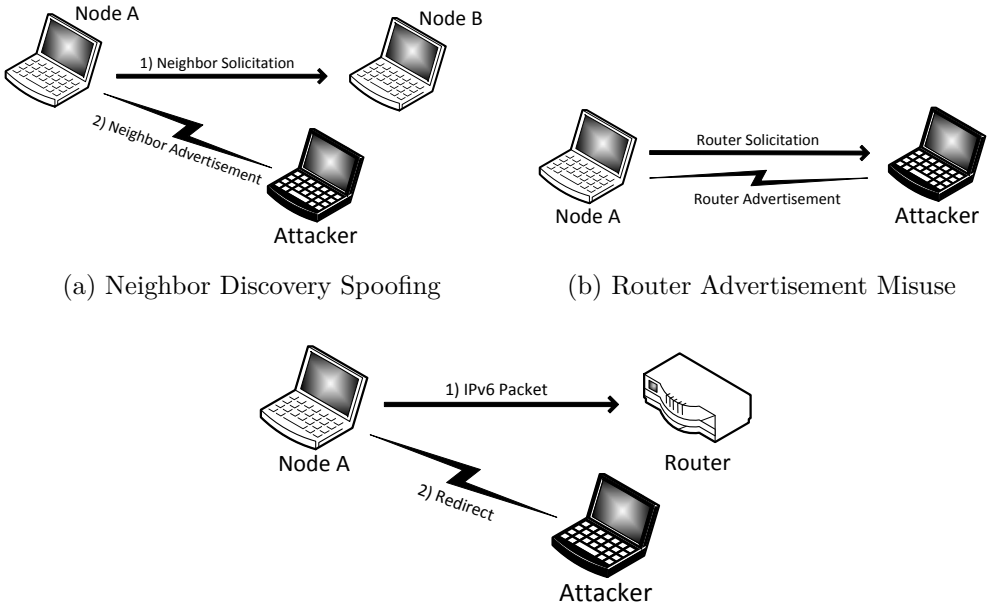
Similar to the *Router Alert option*, a high number of different *flow labels* is able to decrease router performance because the latter has to store a state for every label value. An adversary can also gain access to someone else's quality of service by using the same *flow label* [181].

6.2.4 Neighbor Discovery

Neighbor discovery has many security implications due to its philosophy of trusting everybody on the local network. Assuming an adversary has managed to reach the local network, they can perform a variety of malicious actions.

Address Resolution: Spoofing attacks that provide wrong link-layer addresses are still possible (Figure 6.1a). Adversaries are further able to prevent victims from address assignment by answering to *duplicate neighbor detection*. One applied countermeasure is *Optimistic Duplicate Address Detection*. Here, the node assumes that its address is unique in any case [182].

Router Advertisement Spoofing: Any node on the local network is able to announce itself as a router (see Figure 6.1b), or spoof a router's announcement. A number of variations of this attack are known: (1) Setting the router's lifetime to zero kicks the



(a) Neighbor Discovery Spoofing

(b) Router Advertisement Misuse

(c) Redirects

Figure 6.1: Basic Attacks with Neighbor Discovery

remainder from the client's configuration. (2) Announcing an arbitrary prefix lets the clients assume this prefix is local [183, 184].

(3) Flooding the network with *router advertisements* with various prefixes causes clients to configure one address per announcement and may lead to denial of service. These problems are not fully solved by using DHCP, as the adversary can force the node to abandon DHCP. As a countermeasure, the router advertisement guard – a middlebox filtering illegitimate announcements – is proposed [185, 186].

Advertisements may also be sent unintentionally due to misconfiguration. Preferences of benign announcements should therefore be high to guarantee service even in such a case [187].

Redirects: An adversary may redirect traffic by sending *redirects* and change the sender's configuration this way.

Smurf Attacks: An adversary sends a request to a multicast address, spoofing the victim's source address. Responses are returned to the victim, causing a denial of service. Adequate request types are *echo requests* or IP packets with an unknown extension header option of type 10. *echo requests* to multicast addresses must not be answered, but some implementations do. In contrast, the alternative containing an unknown option has to be answered [188]. Considering the latter, non-answering has been proposed [189], but even

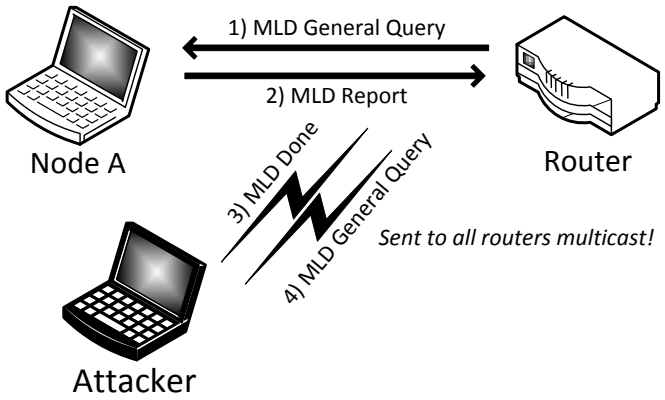


Figure 6.2: Multicast Listener

in case of becoming a standard, an exception remains for *Packet too Big* messages for path MTU discovery.

General security mechanisms tackling all vulnerabilities together have been targeted. With IPsec being initially mandatory, neighbor discovery seemed adequately secure, but it suffered from bootstrapping problems. Securing it would require manual key exchange, and therefore, unacceptable effort. As a consequence, *Secure Neighbor Discovery* (SeND) was introduced [190]. With this technology, cryptographically generated addresses enable the association of addresses to a public key [191], and signing messages with the private key prevents spoofing. However, RSA is calculation intensive and the overhead makes the systems more prone to denial-of-service attacks. Even more limiting is the low support. For example, there is only one proof-of-concept implementation for Microsoft operating systems [192]. Therefore, the only option remains to prevent adversaries from joining the local network through physical protection or link-layer access control.

6.2.5 Multicast Listener Discovery

Multicast Listener Discovery (MLD) is a protocol maintaining information on nodes listening to multicast addresses. This allows the forwarding of packets destined for these addresses. A query router in charge of maintaining this information regularly sends general query messages asking for listening nodes. The latter answer with report messages. A malicious node can abort this forwarding of multicast-destined packets by sending a spoofed done message. The effect, however, would last only until the next general query message that is answered by the victim, initializing forwarding again.

Thus, the adversary has to attempt to become the query router itself. The query router is determined by having the lowest address. Although routers are frequently assigned ascending addresses, the lowest IPv6 interface identifier `::` (all zeros) is typically unused and addressing starts with `::1` [193] – possibly an IPv4 legacy.

After becoming the query router, it stops sending query requests, causing an MLD denial

of service. However, the old query router will start querying again if it does not see MLD requests. However, if the adversary sends such queries only to the all-router multicast address, the other routers are satisfied while the nodes face deteriorated service (see Figure 6.2). Assigning the lowest address `::` to the legitimate router is an adequate countermeasure, as explained above.

6.2.6 Tunneling

At the beginning of IPv6 deployment, tunneling illegitimate content over IPv6 was easy because many firewalls let any IPv6 traffic pass. While this has changed drastically, special threats arise from transition technologies due to the combination of the two IP versions.

Routing loops are an issue of automatic tunneling mechanisms, e. g. *Teredo* or *ISATAP* [194, 195]. Starting with a native IPv6 packet with a spoofed source address, this packet is forwarded to a tunnel ingress point. There it is encapsulated into an IPv4 packet and forwarded. At the egress point, the packet is decapsulated and equals the first, which is forwarded again to the ingress point. This causes traffic amplification because the *hop count* is only reduced on native IPv6 routers. Mitigation methods may include the general avoidance of multiple tunnels and border routers, a list of other tunnel routers' addresses to drop their packets, and checking IPv4 and IPv6 addresses for consistency [196, 194].

Special attacks are known for *Teredo*: (1) Cycling is possible between an end node and a cone NAT supporting hair-pin routing. (2) Even endless looping is possible with a bubble request. Originally intended to open another NAT via the server, the request to open the server address causes the server to send bubbles endlessly.

Nested encapsulation means the encapsulation of tunnel packets in packets of another tunnel, causing additional overhead through another packet header or even fragmentation. To counter this, a *Tunnel Encapsulation Limit option* limiting the number of nested tunnels has been introduced [197].

6.2.7 Address Space Size

The massive expansion of address space returns vulnerabilities known from the Internet's early days. Simplistic implementations of neighbor discovery may hold too many still unanswered neighbor address requests caused by network scanning. To mitigate this denial of service, filtering unused address space and minimal subnet sizing is proposed [198]. There is even the discussion of minimizing subnets down to e. g. a /124, but then it is likely that implementations fail due to assuming minimum subnetworks of /64.

Point-to-point links encounter the threat of ping-pong packets in case a router forwards a packet back over the incoming interface and causes packet cycling. As above, taking smaller subnets, e. g., /127 would mitigate the risk [199]. Alternatively, the latest ICMPv6

specification [145] mitigates this by returning an ICMPv6 *Destination Unreachable* message.

6.2.8 Mobile IPv6

Binding updates inform the home agent of a mobile node's current address and enable it to stay reachable via its home address. Spoofing binding updates may inform the agent of a wrong address and can be used for man-in-the-middle, hijacking, passive wire tapping or denial-of-service attacks. In order to prevent these attacks in mobile IPv6 networks, the use of IPsec is recommended [200].

6.3 Privacy Issues

Internet-based technologies are becoming increasingly pervasive and exhibit a tendency to neglect users' privacy, addressing privacy violations is of utmost importance. In this section, we highlight privacy-related challenges along with state-of-the-art countermeasures.

6.3.1 Addressing

As stated above, an 128-bit IPv6 address consists of a network prefix and an interface identifier. While the first is given by the network on which the host resides, the interface identifier is independently generated by the host. Initially, the modified EUI-format containing the MAC address was proposed for generation of the interface identifier [149]. Since using a hardware address results in unique identifiers even across different subnets, it is easy to track a node's movement through the network. A draft now even proposes their deprecation [201].

Numerous address formats have been proposed as an alternative: (1) The *Privacy Extension* generates an MD5 hash at a regular time interval – typically 24 hours – and uses this as the identifier [202]. While this impedes long-term tracking, short-term tracking is still possible as the identifier does not change simultaneously with the prefix. (2) Another alternative frequently proposed is DHCPv6. However, it relies on the static *DHCP Unique Identifier* (DUID). By sniffing DUIDs locally or requesting the respective DHCP servers directly, an adversary is still able to correlate a node with its current address [203].

With *Mobile IPv6*, there is a trade-off between keeping track of all sessions during network switching and the privacy breach allowing to be traceable across different networks. By including the home address and the temporary care-of address in one packet, a potential adversary is able to eavesdrop on the communication channel and infer the device's location. This may be prevented by encryption, e. g., IPsec. However, nodes communicating with the mobile device can still track the latter. To prevent such privacy breaches, the care-of address and the home address must also be changed simultaneously [153].

6.3.2 Reconnaissance

The discovery of unknown nodes is typically the first step in an attack or penetration test, but the sheer size of the address range makes brute-forcing impossible. Thus, more sophisticated methods are necessary: (1) In 2007, an analysis of IPv6 addresses in the wild showed frequent address structures for the first time [204]. While servers and routers tend to follow the modified EUI-Format and "low" addresses, clients have a significant portion of addresses generated by the privacy extension. Further analyses are feasible by *address6* [201]. Results of such analyses have resulted in *scan6* of the same toolkit. This tool searches for low-byte, IPv4-based, port-based or modified EUI addresses.

(2) Another source for addresses is DNS, which will be becoming more popular with IPv6 due to the address length. First, it is possible to query known domains. Second, reverse entries can be exploited at BIND or NDS implementations [205]. As the response for an empty non-terminal differs from other error messages, it is possible to infer whether addresses starting with this prefix are known to this server. (3) Beyond DNS, all other sources of addresses are of interest as well, e. g., *Node Information Queries* [206], *Inverse Discovery* [207] or *whois.net* [208].

(4) A modified version of the smurf attack is also capable of reconnaissance. Instead of spoofing the source address, the adversary inserts its own address and receives responses with previously unknown source addresses. However, one has to be aware that a high number of responses may cause a denial of service to oneself [193]. To prevent revealing individual addresses, servers listening to anycast addresses should also use this anycast address as a source address in the response [176].

But inherent features of IPv6 also make reconnaissance easier: (1) The assignment of more than one address to an interface is legitimate, but for reconnaissance it is sufficient to discover one. (2) Addresses expire after a preferred lifetime, but are still used for an existing connection for some time [150]. (3) Clients using the privacy extension further own a stable address that can be assigned randomly or following the modified EUI format [201]. (4) ICMP must not be totally filtered with IPv6. Even further, filtering *echo requests* and *responses* is said to be less important due to the alleged possible risk from scans [209]. An overview on this topic is also given by [208].

6.3.3 Covert Channels

Covert channels are communication channels violating system policies. In total, 22 possible covert channels have been found in the IPv6 header and its extensions [210], thereof six alone in the plain IPv6 header: (1) *flow label*, (2) *traffic class*, (3) *payload length*, (4) *next header*, (5) *hop limit* and (6) the *source address*. The most known covert channels are the *flow label* with 20 bit [181] and the *traffic class* with 8 bit, as their use is still vaguely defined. While the latter is allowed to be changed en route, the modification of the *flow label* was previously prohibited [211]. This, however, has changed: resetting is allowed in case a covert channel imposes a serious risk [181]. The *payload length* channel increases this value, and includes the covert messages at the end

of the payload; the *next header* inserts an additional extension header and hides the message there; channels exploiting *hop limit* are well known as *time to live* channels from IPv4 and insofar nothing new [212]. A novel covert channel of 64 bit is however provided by the interface identifiers, i. e., the remainder 64 bit of the address. As the privacy extension causes frequently changing random addresses, it is highly unlikely that these secret messages are detected [213].

6.4 Systematization of Knowledge

Systematization means arranging something so as to present the content more clearly. Section 6.2 and Section 6.3 explained security and privacy vulnerabilities as well as countermeasures for IPv6 verbally. This section presents them so that they can be taken at a glance and serve as a checklist for researchers and practitioners alike. With the more in-depth verbal description in the previous sections and this systematic overview, this chapter presents the subject in multiple ways, allowing it to be used as a reference guide.

The methodology has to fulfil two goals: (1) a clear arrangement and (2) a brief description of the attacks. In Section 6.4.1, an appropriate approach is presented. Section 6.4.2 contains the systematization for vulnerabilities, Section 6.4.3 for countermeasures and Section 6.4.4 shows the adequacy of countermeasures to vulnerabilities.

6.4.1 Methodology

[214] developed an extendible common language for describing computer security incidents. According to this work, *"an attack is a series of steps taken by an adversary to achieve an unauthorized result"*. It consists of a tool for exploitation, a vulnerability describing a system weakness, an event – a directed action intended to change the state of a system – and an unauthorized result. The event consists of an action performed by the adversary on a certain target. We adapted this common language for the purpose of describing IPv6 security and privacy vulnerabilities and the respective countermeasures. The original common language did not offer a description for countermeasures, but we believe describing them as a sequence of steps as well is adequate.

6.4.2 Systematization of Vulnerabilities

The vulnerabilities have been systematized by means of six attributes: (1) action, (2) object, (3) target, (4) unauthorized result, (5) origin, and (6) type.

The action describes the activity of the adversary and is further specified by the object and the target. The object describes the entity the action is performed on. The target defines the victim node. If the latter attribute is left free, all types of nodes are likely to be attacked. While object and target are not enumerated, a limited number of values exist for action. The following list defines them in accordance with place holders for object and target in brackets:

- *assign:* set the address for [target] to [object]
- *flood:* emit a high number of [object] to [target]
- *insert:* include [object] into [target]
- *listen:* eavesdrop on the traffic for [object]
- *scan:* iterate through the addresses of [target]
- *send:* emit a packet including [object] to [target]
- *spoof:* emit [object] to [target] pretending to be another node

The unauthorized result describes the aftermath of the malicious action. Further, the origin of a vulnerability and a threat type is defined. The attribute vulnerability indicates whether the vulnerability results from a design, implementation or configuration flaw according to the following definitions by [214]:

- *configuration: "a vulnerability resulting from an error in the configuration of a system"*
- *design: "a vulnerability inherent in the design or specification of hardware or software whereby even a perfect implementation will result in a vulnerability"*
- *implementation: "a vulnerability resulting from an error made in the software or hardware implementation of a satisfactory design"*

The threat type is also limited to three values following the definitions by [215]:

- *interception: "some unauthorized party has gained access to an asset"*
- *interruption: "an asset of the system becomes lost, unavailable, or unusable"*
- *modification: "an unauthorized party not only accesses but tampers with an asset"*

The resulting systematization for the above described vulnerabilities is found in Tables 6.2 and 6.3 (security vulnerabilities) and Table 6.4 (privacy vulnerabilities).

ID	Vulnerability	Action	Object	Target	Unauthorized Result	Origin	Type
v01	Fragmentation Header I	send	overlapping fragments		modified header fields	design	modification
v02	Fragmentation Header II	send	port number in second fragment		middlebox evasion	design	interception
v03	Fragmentation Header III	flood	fragments		memory shortage	design	interruption
v04	Fragmentation Header IV	flood	atomic fragments		packet loss	design	interruption
v05	Routing Header Type 0 I	send	routing header		traffic amplification	design	interruption
v06	Routing Header Type 0 II	send	routing header		middlebox evasion	design	interception
v07	Extension Header Options I	send	router alert option		increased workload	design	interruption
v08	Extension Header Options II	spoof	invalid 10xxxx option	multicast address	multiple responses	design	interruption
v09	Hop-by-Hop Header	send	hop-by-hop header		increased workload	design	interruption
v10	New Extension Header	send	unknown extension header		middlebox evasion	design	interception
v11	New Extension Header	send	unknown extension header		increased workload	design	interruption
v12	Flow Label I	send	different flow labels		memory shortage	design	interruption
v13	Flow Label II	send	existing flow label		quality-of-service theft	design	interruption
v14	Neighbor Advertisement I	spoof	neighbor advertisement		wrongly resolved address	design	interruption
v15	Neighbor Advertisement II	spoof	neighbor advertisement		traffic redirection	design	modification
v16	Neighbor Advertisement III	spoof	neighbor advertisement		address assignment prevention	design	interruption
v17	Router Advertisement I	spoof	router advertisement		new default router	design	modification
v18	Router Advertisement II	spoof	router advertisement		removed default router	design	modification
v19	Router Advertisement III	spoof	router advertisement		wrong locally-announced prefix	design	modification
v20	Router Advertisement IV	flood	router advertisement		multiple address assignment	impl.	interruption
v21	Router Advertisement V	spoof	router advertisement		prevention of DHCP assignment	design	interruption
v22	Router Advertisement VI	send	router advertisement		IPv6 activation	impl.	modification
v23	Redirect I	spoof	redirect		redirected traffic	design	modification
v24	Redirect II	spoof	redirect		wrong locally-announced node	design	modification
v25	Echo Request I	spoof	echo request	multicast address	multiple responses	impl.	interruption
v26	SeND	send	authenticated messages		increased workload	design	interruption

Table 6.2: Classification of Security Vulnerabilities (Part I)

ID	Vulnerability	Action	Object	Target	Unauthorized Result	Origin	Type
v27	Tunneling I	send	IPv6 packet as IPv4 payload		middlebox evasion	impl.	interception
v28	Tunneling II	send	tunnel packet	relay router	cycling packet	impl.	interruption
v29	Tunneling III	send	tunnel packet		cycling packet	conf.	interruption
v30	Teredo	send	Teredo bubble	server	cycling packet	design	interruption
v31	Nesting	insert	packet into packet		packet overhead	conf.	interruption
v32	Fragmentation Header V	send	packet too big		inclusion of atomic fragments	design	interception
v33	Neighbor Discovery	scan		subnetwork	memory shortage	impl.	interruption
v34	Forwarding	send	returning packet		traffic amplification	design	interruption
v35	Mobile IPv6 I	spoof	binding update	home agent	traffic redirection	design	modification
v36	Multicast Listener	assign	lowest address	itself	new MDL query router	design	modification

Table 6.3: Classification of Security Vulnerabilities (Part II)

ID	Vulnerability	Action	Object	Target	Unauthorized Result	Origin	Type
c01	Fragmentation Header VI	send	overlapping fragments		identification	impl.	interception
c02	Modified EUI Format	scan	interface identifier	networks	tracking	design	interception
c03	Echo Request II	send	echo request	invalid multicast address	identification of sniffing nodes	impl.	interception
c04	Mobile IPv6 II	listen	binding update		tracking	design	interception
c05	DHCP I	listen	DHCP traffic		tracking	design	interception
c06	DHCP II	send	DHCP information request	DHCP server	tracking	design	interception
c07	DNS	send	DNS request	DNS server	reconnaissance	design	interception
c08	Reverse DNS	send	Reverse DNS query		reconnaissance	impl.	interception
c09	Echo Request III	send	echo request	multicast address	multiple responses	impl.	interception
c10	Extension Header Options III	send	packet with invalid option	multicast address	multiple responses	design	interception
c11	Anycast	send		anycast address	response with unicast address	impl.	interception
c12	Traffic Class	insert	secret information	traffic class field	leaked information	design	interception
c13	Flow Label	insert	secret information	flow label field	leaked information	design	interception
c14	Privacy Extension I	insert	secret information	interface identifier	leaked information	design	interception

Table 6.4: Classification of Privacy Vulnerabilities

6.4.3 Systematization of Countermeasures

Countermeasures are described by the two attributes action and object, which have the same purpose as for vulnerabilities. However, the list of actions changes to the following:

- *assign:* set [object]
- *disable:* deactivate [object]
- *encrypt:* encode [object] to be secured against reading and/or tampering
- *filter*[3]*:* remove [object] when passing
- *isolate:* process [object] separately
- *limit:* define maximal value for [object]
- *log:* write message about [object]
- *minimize:* reduce number of [object] as much as possible
- *prohibit:* ban [object]
- *respond:* return with [object]

Object is not enumerated. The countermeasures are further classified into three groups of activity levels: (1) *detective* countermeasures discover a present attack, (2) *preventative* countermeasures are taken before an attack takes place, and (3) *reactive* countermeasures are triggered by the attack. The resulting systematization is found in Table 6.5 (detective and preventative countermeasures) and Table 6.6 (reactive countermeasures).

[3]Discarding has been included in filtering as it can also be understood as removing messages.

ID	Countermeasure	Action	Object
	Detective		
c01	NDP Mon	log	inconsistent NDP msg.
	Preventative		
c02	Use Anycast Address	respond	with anycast as source address
c03	DHCP	assign	addresses statefully
c04	No Forwarding	prohibit	forwarding over same interface
c05	Fragment Isolation	isolate	atomic from other fragments
c06	IPsec	encrypt	packets
c07	IPsec with Manual Keys	encrypt	packets
c08	No IPv6 Support	disable	IPv6
c09	Format Deprecation	prohibit	modified EUI format
c10	Multicast Listener Address	assign	lowest address to router
c11	No Multiple Edge Routers	disable	other edge routers
c12	No Multiple Tunnels	disable	other tunnels
c13	No Multicast Responses	prohibit	answers to multicast addresses
c14	No Overlapping Fragments	prohibit	overlapping fragments
c15	Packet Rate	limit	packet rate
c16	Physical Protection	prohibit	physical access to network
c17	Privacy Extension	assign	temporary random address
c18	RA Throttler	limit	router advertisements
c19	No RAs	disable	router advertisements
c20	No Routing Header Type 0	prohibit	routing header type 0
c21	Router Preference	assign	highest preference
c22	Segmentation	segment	network
c23	SeND	encrypt	NDP messages
c24	Subnet Size	minimize	subnet size
c25	Temporary DUID	assign	temporary DUID
c26	No Tunneling	disable	all tunnels
c27	Uniform Format	limit	number of ext. header formats

Table 6.5: Systematization of Countermeasures (Part I)

ID	Countermeasure	Action	Object
	Reactive		
c28	Address Change	assign	new addresses simultaneously
c29	Address Checks	filter	inconsistent addresses
c30	Change Field en route	assign	default value
c31	Echo Requests	filter	echo requests
c32	Hop-by-Hop Options	filter	hop-by-hop extension header
c33	Routing Header	filter	routing headers
c34	Fragmented Packets	filter	packets with port not in 1st frag.
c35	Invalid Options	filter	options of type '10xxxx'
c36	Link Layer Access Control	filter	unauthorized clients
c37	Message Checks	filter	invalid ICMP msg.
c38	NDP Inspection	filter	inconsistent msg.
c39	RA Guard	filter	invalid router advertisements
c40	RA Filtering	filter	router alert options
c41	Router Listing	filter	msg. from other tunnel routers
c42	Tunnel Enc. Limit	limit	number of nested packets
c43	Tunnel Ingress and Exit	filter	at tunnel end points
c44	Unused Addresses	filter	unused addresses

Table 6.6: Systematization of Countermeasures (Part II)

6.4.4 Vulnerabilities and Appropriate Countermeasures

Table 6.7 and Table 6.8 show the adequacy of countermeasures to vulnerabilities. We created a matrix where each row represents a vulnerability and each column a countermeasure. A checkmark indicates that a countermeasure is adequate. There is no distinction between various levels of mitigation, e. g., total mitigation vs. some improvement of status quo.

The introduction of a certain countermeasure may lead to new vulnerabilities. For example, the use of SeND to prevent *router advertisement* attacks creates a vulnerability to denial-of-service attacks due to increased calculation efforts. Likewise, the use of the privacy extension prohibits tracking, but makes it possible for the interface identifier to be used as a covert channel. Thus, a method may be a vulnerability and a solution to another vulnerability at the same time. Further, there are vulnerabilities that cannot be mitigated easily by means of the mechanisms presented here, e. g., memory shortage due to fragment flooding.

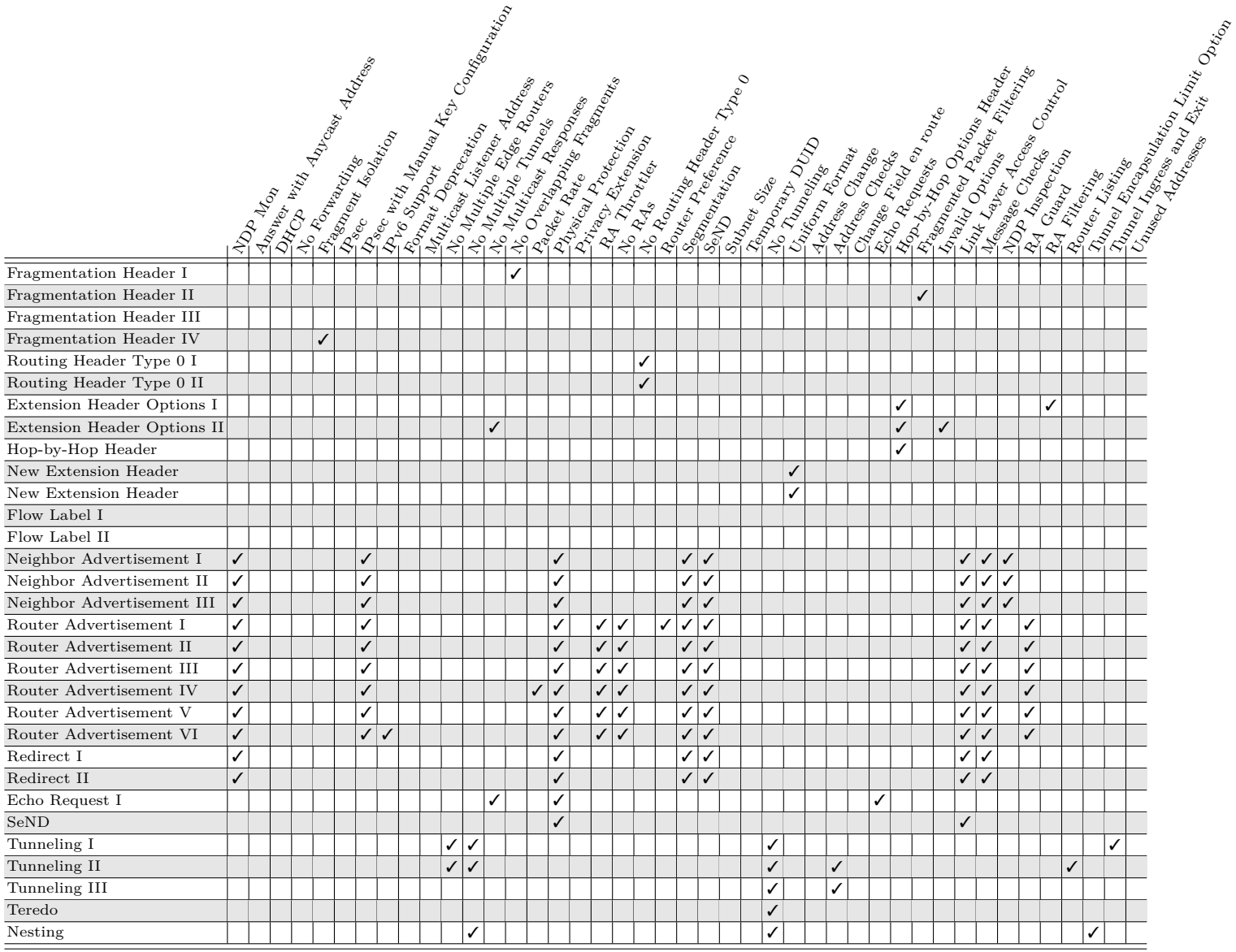


Table 6.7: Evaluation of Countermeasures (Security Vulnerabilities)

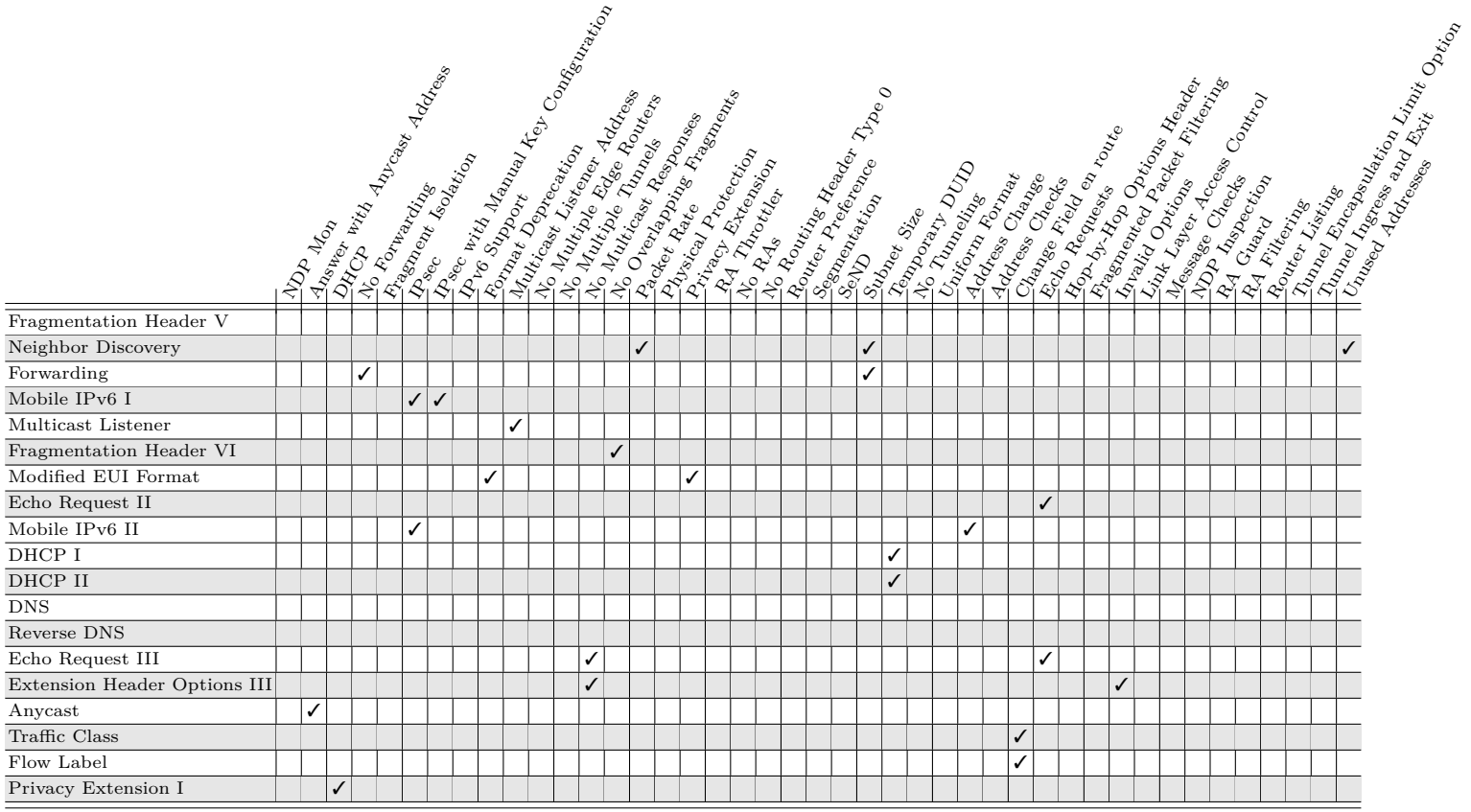


Table 6.8: Evaluation of Countermeasures (Privacy Vulnerabilities)

6.5 Findings and Discussion

Large-scale IPv6 deployment is unquestionably a practitioners' task. However, in this case, practice and research live in mutual symbiosis. The practical experience gained from large-scale deployments typically reveals previously unknown security issues that are not easily solved. As such, they are bounced back to research, where in-depth investigation takes place. In this chapter, we described IPv6's status quo with the objective of identifying such back-bouncing topics. While many vulnerabilities have already been considered in practice, the results from our systematization suggest that there is a variety of research challenges to be investigated. In this section, we infer these main challenges regarding IPv6 and propose possible approaches for mitigation.

6.5.1 Secret Communication

Our survey has a broad focus on IPv6 security and privacy. In comparison to cloud computing, secret communication appears to have lower importance with IPv6; the amount of research in the respective field is considerably lower.

IPv6 provides numerous covert channels; but the most outstanding appears to be the IPv6 addresses of 128 bit length. On the one hand, it appears obvious that 128 bit allows more expressiveness than 32 bit of legacy IPv4 addresses. On the other hand, the potential for covert channels does not solely arise from this increased size. The reasons are rather the following: Node density is considerably lower in IPv6 networks; typically a few hosts reside in a /64 network and are provided with more addresses than the whole IPv4 Internet. Thus, addresses can be chosen almost freely. Further, SLAAC introduces a (partly) decentralized way of address assignment and a host generates a part of its address itself without any further restrictions. Finally, fast changing addresses as introduced by the IPv6 privacy extension as well as multiple addresses per network interface are becoming the norm allowing secret data transmission at a regular interval while maintaining normal networking behavior.

With respect to IPv6, literature solely describes covert channels, i. e., a sender with the intention to reveal (sensitive) information. We believe that these channels could not only serve as a covert channel, but are rather of the opinion that side channels are a more likely attack scenario as the sender does not have to actively participate in communication. By the example of IPv6 addresses, addresses could unintentionally reveal patterns that provide insights into an organization's network architecture or the administrator's habits on address assignment. Both could lay the foundation for later attacks.

Finally, the IPv6 privacy extension might be considered as a means of obfuscation as it aims to conceal the client address in order to protect privacy. In comparison to the definition of Section 2.2.4, no intermediate node is required as the loads of different addresses dupe multiple nodes.

6.5.2 Reconnaissance

Even though reconnaissance in IPv6 has been considered impossible, various techniques have proven the opposite. Nevertheless, they have some drawbacks: (1) DNS querying reveals mainly servers that are intended to be found anyway. (2) Messages to multicast addresses invoking responses may result in a denial of service and their deprecation is foreseeable. (3) Eavesdropping, i. e., passive listening to network traffic, does not work for outside adversaries as it is unlikely that packets originating within the victim's prefix will run into the adversary in an arbitrary location on the Internet.

Considering this, scanning is still the most promising reconnaissance type due to (1) invoking active responses from the victim, (2) revealing the stable address instead of a temporary one, (3) its local as well as global applicability, (4) its independence from certain protocols and (5) the difficulty of mitigating it due to using the inherent functionality of protocols. What seems to be legacy is brute-force scanning, i. e., iterating through all possible addresses – the method of choice in IPv4. In conclusion, research has to find new address selection algorithms for active probing to replace brute-forcing and manage the large amount of IPv6 addresses in this way. We believe that the exploitation of address structures is promising. Reconnaissance is however also dependent on the developments in addressing, see the next subsection.

6.5.3 Addressing

Every proposed addressing solution has a serious drawback: (1) The modified EUI-format is easily traceable by benign administrators as well as adversaries using out-of-the-box tools like ping. (2) The usage of DHCP does not mitigate this issue because of the unique and stable DUID, and (3) the privacy extension is highly volatile. Therefore, especially administrators fear its negative impact on logging. (4) Manual address assignment is possible for servers and routers, but not for a large amount of clients, e. g., in the Internet-of-Things (IoT). These drawbacks highlight the lack of an adequate address assignment structure for the clients' side in IPv6.

This implies that IPv6 has not been well understood, and that the most suitable address format for distinct application scenarios remain unclear. For example, better protection of privacy might be more important for mobile nodes than for stationary ones, more important for private users than for corporate users. Requirements for client addressing have to be defined prior the development of another approach, but therefore we need to make an inventory of current approaches, understand their advantages as well as their drawbacks before defining another – now privacy protecting – address format.

CHAPTER 7

A Pattern-Based Reconnaissance Approach

Internet-wide scanning experienced a tremendous boom in recent years; and today, scanning the entire Internet takes not more than an hour to complete [216, 217]. This evolution has provided a number of insights into the Internet ecosystem: *Heninger et al.* investigated the cryptographic protocol TLS and SSH, and found hosts using the same keys as others [218]. *Durumeric et al.* studied the HTTPS certificate system and showed that the majority of trusted certificates were controlled by three organizations [219]. *Rossow* scanned for nodes which were vulnerable for reflection attacks and found millions of them [130]. Even further, scanning plays a vital role in vulnerability mitigation: By handing scanning results over to CERTs, *Kührer et al.* measured a drop of 92 % of hosts vulnerable to NTP reflection attacks [220]. *Durumeric et al.* measured a 47 % decrease of servers vulnerable to Heartbleed after reporting [221]. Apart from Internet-wide scanning, scanning a certain subnet has always been a part of penetration testing to discover potential victims.

The presented approaches have in common that they probe every address within a certain range. However, this method collides with the introduction of IPv6 [144]. The new version of the Internet Protocol has an increased address range making it impossible to scan even the smallest subnet in the common way. As a consequence, practitioners and researchers looked out for alternative ways of reconnaissance[1], and the resulting development can be best described in two steps that followed different premisses:

(1) If one cannot probe all addresses, one has to use other sources to gain valid addresses. In a first step, researchers and practitioners accessed systems that stored addresses for

[1] In this chapter, we use the term *reconnaissance* for the discovery of unknown hosts in a network. The term *(network) scanning* is used for the discovery of unknown host through sending requests in await for responses. According to this definitions, *scanning* is a means of reconnaissance.

their intended application. Thereupon are public archives, centralized application servers or various ways of leveraging the Domain Name Systems [208, 222]. The drawback of this approach is that nodes that are not participating or not listed are never discovered.

(2) If one cannot probe all addresses, one has to include more information to synthesize promising addresses. In a second step, researchers and practitioners targeted to reduce the address space through address patterns. Thereby, all addresses containing a certain pattern are probed. Known patterns arise from standardization (e. g. *Modified EUI format*) or striking structures in addresses (e. g. low-byte addresses) [208, 222]. This approach also has its drawbacks. The benefit of patterns that are inferred from standards is unknown because these standards are not without alternatives and other IPv6 addressing schemes exist. Striking patterns are typically crafted manually, based on educated guesses and might not be as wide-spread as they might seem to humans. To the best of our knowledge, no evaluation of pattern-based approaches is available.

In this chapter, we overcome this gap and assess pattern-based scanning in IPv6 in an experimental set-up. We evaluate not only currently known patterns with respect to their applicability to reconnaissance, but also develop a pattern-based scanning algorithm. This algorithm automatically extracts patterns from a small training set of addresses in a first step, i. e., the patterns are extracted by means of a side channel, and generates addresses based on these findings for later scanning. Our results imply that pattern-based scanning is a feasible approach for the discovery of IPv6 hosts, but known patterns are of limited benefit. They are outperformed by our novel algorithm using a side channel. Our algorithm's results vary with respect to certain parameters, however, we are able to pre-estimate the quality of our results.

The remainder of the chapter is structured as follows: Section 7.1 discusses reconnaissance in both protocol versions IPv4 and IPv6. Section 7.2 presents the considered scenario for scanning, explains our novel pattern-based algorithm for scanning and current, manually crafted patterns. In Section 7.3, we cover experiments to evaluate pattern-based scanning. The results are discussed in Section 7.4, Section 7.5 concludes this chapter.

7.1 Internet Reconnaissance

Our research is based on two foundations: First, we discuss network scanning with the predecessor version IPv4 and highlight why approaches of more sophisticated address generation serve a different goal than increasing the number of discovered hosts. Second, we highlight ways of reconnaissance with IPv6, show the advantages of scanning with version 6 and further highlight current scanning approaches.

7.1.1 Scanning with IPv4

The de-facto standard IPv4 scanner is the open-source tool *nmap* [223]. This and similar tools are crafted for scanning small address ranges. Due to maintaining a connection-wise state they are not capable of sending high numbers of packets. Internet-wide scanning

with a tool like *nmap* requires a high number of nodes, lots of time and/or money [218]. Their counterparts are specialized scanning tools that are optimized for high traffic rates: *IRL scanner* was the first in 2010 [224] and covered the whole Internet in 24 hours using a single machine. As this tool had never been released to the community, the issue was brought up again in 2013 and two new tools were presented: *ZMap* [217] and *masscan* [216]. *ZMap* is a modular, open-source tool that predominates in academia and enabled a variety of insights into the global Internet ecosystem, e. g., [218, 219]. *masscan* is also open-source and claims to be faster. Being released to the public, both tools were found to be used in the wild [225].

With respect to this chapter, the tools' address generation is of interest: *nmap* iterates through all addresses of a range in ascending order and starts with the lowest. This method imposes the drawback of possibly overloading a destination because close addresses are likely to be also topologically close. Internet-wide scanners thus aim to balance the traffic by scrambling the address order. *IRL scanner* uses a reversed linear congruential generator permutation, *ZMap* iterates over a multiplicative group of integers modulo a prime slightly larger than 2^{32}, and *masscan* encrypts an incrementally increased index by means of a hash function. Although the latters' address generation is more sophisticated, they still target to probe every single address in a range.

7.1.2 Reconnaissance with IPv6

Initially, the focus drifted to ways of reconnaissance beyond scanning [208] due to the myth of IPv6's unscannability. On the one hand, sources that store addresses could be used: (1) Querying the DNS for known domains reveals addresses, and unhandy IPv6 addresses might be more likely listed than their IPv4 counterparts. (2) Different answers of certain DNS server implementations allowed the reduction of the address space because the server's response differs for empty non-terminal from other errors. (3) A variety of other services might also be used, e. g., Node Information Queries, log files or centralized application servers. On the other hand, (4) some IPv6 implementations responded to requests to multicast addresses with their unicast address and allowed reconnaissance for local adversaries. Summarizing however, no approach seemed more promising than scanning: The attacker actively invokes a response and is thus independent of the victim's networking customs. Scanning is locally as well as globally applicable and can base on a variety of protocols. Scanning exploits the protocols' intended functionality that cannot be fully prevented without an impact on regular networking. In return, one has to deal with the fact that not all addresses can be probed.

An early analysis of IPv6 addresses provided the insight that they include extra expressiveness due to their increased length [204]. *Reversing* this expressiveness is an approach to create actually used IPv6 addresses. *Gont et Chown* [222] searched through addressing standards for exploitable patterns for address reduction and also proposed patterns for manually crafted addresses. *Gont* implemented these patterns in the scanning tool *scan6* [226]. The idea behind pattern-based scanning is the reduction of search space as a consequence of the fact that not all addresses in IPv6 can be probed. Thus, one aims to

probe more likely addresses prior less likely ones as opposed to the IPv4 approaches that use a changed address order solely to prevent destination overloading. These patterns are manually crafted based on educated guesses, and have never been evaluated with respect to their applicability. Further, manual crafting needs manual updates in case the underlying addressing schemes change, and do not go beyond obvious patterns.

This chapter aims to overcome these issues: It assesses whether pattern-based approaches are feasible in general, evaluates the current approaches in detail and compares them to our novel pattern-based algorithm. This algorithm automatically discovers addresses, and generates new addresses for scanning based on these findings.

7.2 Scanning Design

In this section, we present our considered scenario for scanning, explain the recursive design of our novel pattern-based algorithm and finally describe the manually crafted patterns from the literature in detail.

7.2.1 Considered Attack Scenario

We consider hosts that reside in the same IPv6 network prefix, and an adversary that resides at an arbitrary location on the Internet without local access to the targeted network. The adversary aims to discover as many hosts as possible. He/She is aware of manually crafted address patterns and further has a representative sample of addresses in this prefix[2]. This scenario is typical for penetration tests or adversaries targeting a certain organization unit. Internet-wide scanning consists of a multitude of such scenarios with different prefixes. The assumptions are realistic insofar as manually crafted patterns are publicly available ([226, 222]). The address sample might be gained from the organization unit itself, e. g., insider information, but might also be derived from a similarly organized network.

7.2.2 Recursive Algorithm

This dual-purpose algorithm automatically discovers patterns in a training set of addresses, and generates addresses based on these patterns for scanning. The algorithm for pattern discovery is recursive and refines a given pattern through the determination of an additional bit per recursion. This additional bit is chosen in a way that the refined pattern covers the highest number of addresses among all pattern candidates. With every recursion the number of determined bits increases by one, thus decreasing the number of undetermined bits by one. If the number of undetermined bits falls below a given threshold, address generation is started. Then, all addresses that contain the

[2] An IPv6 address consists of a 64-bit network prefix, and a 64-bit interface identifier. Technically speaking, the adversary aims to discover interface identifiers as the prefix is known, but we stick to the term addresses for comprehensibility.

current pattern are generated in ascending order. In addition to the following textual representation, our algorithm is depicted in Algorithms 1 and 2.

Algorithm 1 `doRecursionWith` determines a further bit in every recursion.

Require: *pattern* is a bit pattern with determined and undetermined bits

```
1: if count(undetermined bits) < threshold then
2:    iterateAddresses(pattern)
3:    return false
4: end if

5: rule = findBestRule(pattern)
6: pattern = apply(pattern,rule)
7: doRecursionWith(pattern)

8: alternativeRule = inverse(rule)
9: alternativePattern = apply(pattern,alternativeRule)
10: doRecursionWith(alternativePattern)
```

Algorithm 2 `findBestRule` finds the rule for the highest number of addresses.

Require: *pattern* is a bit pattern with determined and undetermined bits

```
1: addresses = getAddressesWith(pattern)

2: for each undetermined bit in pattern do
3:    calculateSupportForRule(addresses, undet. bit = 0)
4:    calculateSupportForRule(addresses, undet. bit = 1)
5: end for each

6: return rule with highest support
```

Refined Pattern: The refined pattern covers the highest number of addresses among all candidate patterns. To find this pattern, rules are created and their key performance indicator *support* is calculated. In detail, for every undetermined bit b_u two rules are generated: *Rule 1: The address is appropriate to the current pattern.* $\Rightarrow$ *The undetermined bit* b_u *is zero.* and *Rule 2: The address is appropriate to the current pattern.* $\Rightarrow$ *The undetermined bit* b_u *is one.* The support of a rule is the ratio of the number of addresses fulfilling the rule to the number of addresses fulfilling the current pattern. Applying the rule with the highest support to the current pattern provides the refined pattern that is provided to the next recursion.

Inverse Rule and Pattern: The recursion is not only recalled with the refined pattern, but also with its inverse pattern. In case the best rule of a recursion is *The address*

is appropriate to the current pattern ⇒ The undetermined bit b_v is zero. Its inverse rule is *The address is appropriate to the current pattern. ⇒ The undetermined bit b_v is one.* Applying the inverse rule to the current pattern leads to the inverse pattern, and another recursion is called with this inverse pattern. This guarantees that all patterns are included in the manner of a binary search tree and more likely addresses are probed prior less likely.

Initialization: A pattern with at least one determined bit is necessary for initialization. The algorithm has to be started $2^{number\ of\ start\ bits}$ times to fully cover the search space.

Stop Condition: If the number of undetermined bits falls below a certain threshold, the recursive pattern generation is stopped. Based on the current pattern, all appropriate addresses are iterated in ascending order for scanning. The number of generated addresses is $2^{threshold-1}$.

The start bits and the threshold are parameters of our pattern-based algorithm. Start bits should be chosen in a way that does not impede pattern finding. The threshold of the stop condition defines the transition from pattern discovery to address generation, and thus defines the degree of exploitation of known combinations in addresses versus the flexibility to find slightly different addresses.

7.2.3 Manual Patterns

In the following paragraphs, we discuss manually crafted patterns as defined in [226]. The major difference between this work and our approach is that in [226] the list of scanned patterns bases on experience instead of sample analysis and is fixed. This means that an upgrade has to be done manually by releasing a new version of the scanner. With the approach outlined in this work, trends in the selection of address deployment will be incorporated into the reconnaissance. The manually crafted patterns are as follows.

Low-byte: It was detected that many administrators simply select low numbers for the two low bytes of addresses, so-called *low-byte* addresses. The scanned address range is 2001:db8::0-100:0-1500, i. e., 1.381.889 addresses in total.

Ports: Several different ports are defined as standard ports for services. Administrators use simple schemes to map these service ports into the last bytes of an IPv6 address. The port pattern uses 23 different port numbers of popular services to create four address ranges per port. By the example of FTP (port 21), these ranges are: 2001:db8::0-5:21, 2001:db8::21:0-5, 2001:db8::0-5:15 and 2001:db8::15:0-5[3].

OUIs: IPv6 Interface Identifiers in *Modified EUI format* contain the three byte Organizationally Unique Identifier (OUI), a fixed pattern of two bytes and another three free bytes. This pattern iterates through all 2^{24} addresses of a certain OUI, e. g., 2001:db8::1234:56ff:fe(00-ff):0-ffff with the showcase OUI 1234:56. [226] further mentions a *vendor* pattern consisting of all OUIs of a certain vendor, and a *virtual machines*

[3] 21 in decimal is 15 in hexadecimal.

pattern taking OUIs usually used by virtualization software like *VMWare* and *vbox*. We consider them as a particular case of the *OUI* pattern.

7.3 Experiments

This section describes our experimental set-up and included data sets. Further, our gained results are provided and it is shown that our algorithm's performance can be predicted to a certain degree.

7.3.1 Experimental Setup

Our set-up consists of two *Python* scripts. One implements our recursive pattern-based algorithm and generates an address list. This address list represents the addresses that are scanned. A data set represents the addresses that are used by hosts in this network. The second script compares the generated list with this data set to evaluate the number of actually discovered hosts. We decided for this approach instead of scanning real-world networks in order to gain non-ambiguous results: No response in real-world scanning might have various reasons, like no host listening to this address or that the requested service is not supported. Our set-up allows to isolate IPv6 address generation from non-IPv6 factors.

For every run, two data sets are required: Our pattern-based algorithm requires a training data set to find patterns, while the evaluation algorithms requires a test data set to evaluate the number of successfully discovered hosts. Both data sets are created from our entire data (see below) by a 10-folds cross validation. Typically, the entire data set is split in ten portions of equal size, nine portions form the training data set, one portion the test data set [227]. We believe that using the smaller data set to find patterns, and discover addresses in a larger data set represents the process of reconnaissance more adequately. Thus, we took the smaller set for training, the large for reconnaissance. For the creation of all subsets, we used *WEKA*'s *stratified remove folds algorithm* [228] and the addresses' LSB as classifier. The manually crafted patterns were evaluated by the same evaluation algorithm and on the same test data sets for comparability. The list of addresses were extracted from *scan6* by means of *tcpdump*, and another *Python* script extracting the addresses. In total, we ran ten runs per data set.

For our experiments, we were able to access three real-world data sets that represent a different node type each. *Client* addresses were gained from logs of RIPE's IPv6-enabled homepage *www.ripe.net*. In total we had 167 347 addresses. Requesting AAAA records of the *Alexa Top Million* revealed 16 644 *Server* addresses. Tracerouting the path to these servers revealed 12 982 distinct *Router* addresses. Both data sets were collected from a *Rackspace* cloud instance in the *Dallas* region. These data sets include addresses from a high number of organisations and are thus assumed to be representative for address assignment habits of administrators.

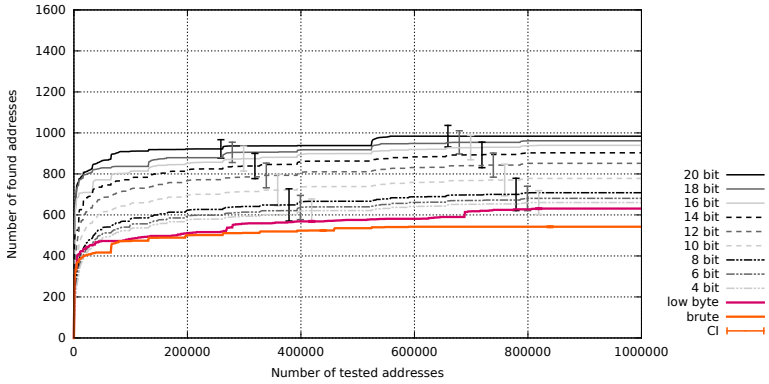


Figure 7.1: Routers: Comparison of Pattern-Based Algorithm, Low-Byte Pattern and Brute-Forcing

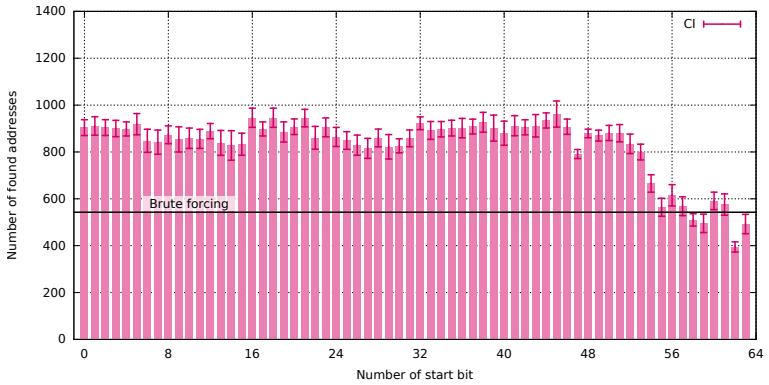


Figure 7.2: Routers: Found Addresses in Dependence of Start Bit

7.3.2 Results

Figures 7.1 to 7.6 provide the results for the three host types routers, servers and clients. The first figure respectively shows the number of found addresses in dependence of the number of probed addresses for different thresholds. The results of our recursive algorithm are contrasted with the results of the low-byte pattern and brute-forcing[4]. The second figures show the number of addresses found by the recursive algorithm in dependence of the start bit with a threshold of 18. The graphs average ten runs, and also provide the confidence interval (CI).

Routers: Our recursive algorithm outperforms brute-forcing and the low-byte pattern (see Figure 7.1). The higher the threshold, the higher the number of found addresses.

[4]In this chapter, *brute-forcing* is considered as probing addresses in ascending order and starting with the lowest.

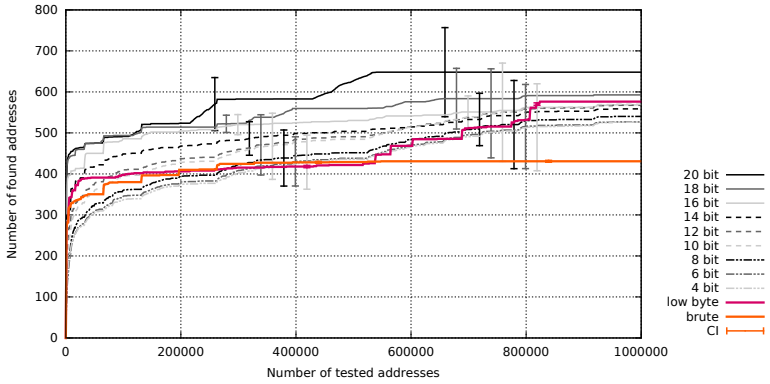


Figure 7.3: Servers: Comparison of Pattern-Based Algorithm, Low-Byte Pattern and Brute-Forcing

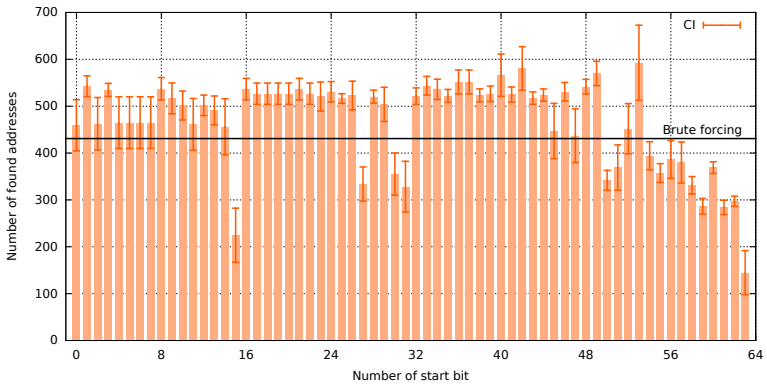


Figure 7.4: Servers: Found Addresses in Dependence of Start Bit

With a threshold of 20, the algorithm reveals plus 442 addresses in comparison to brute-forcing, and plus 353 in comparison to the low-byte pattern. A threshold of 4 still reveals plus 118 respectively 29 addresses. The low-byte pattern discovers 88 addresses more than brute-forcing. Considering initialization, bit 0 to 46 and bit 48 to 53 result in more than 800 discovered addresses (see Figure 7.2). The maximum is 962 addresses (bit 45). Starting with bit 56 to 63 results in less than 620 addresses, the minimum is 395 addresses (bit 62).

Servers: Higher thresholds perform better, and our recursive algorithm outperforms brute-forcing (see Figure 7.3). However, low thresholds (4, 6 and 8 bits) are below brute-forcing in the beginning, but are outstripping brute-forcing within the first third of probes. With a threshold of 20, the algorithm discovers plus 217 addresses in comparison to brute-forcing, and plus 72 in comparison to the low-byte pattern. The low-byte pattern reveals 145 more addresses than brute-forcing, and experiences a steep increase not

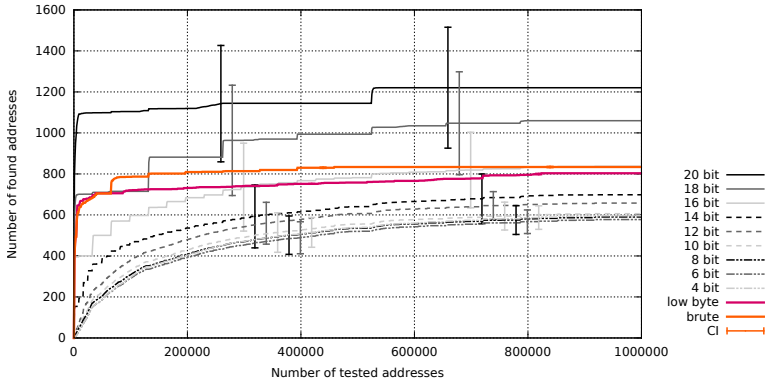


Figure 7.5: Clients: Comparison of Pattern-Based Algorithm, Low-Bytes Pattern and Brute-Forcing

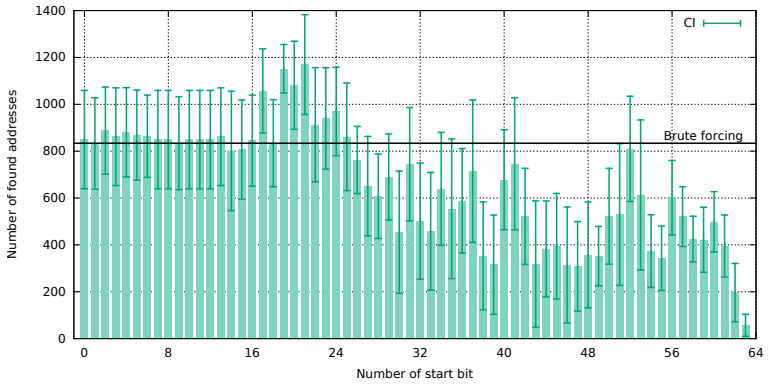


Figure 7.6: Clients: Found addresses in Dependence of Start Bit

only in the beginning, but also at about 500.000 probes. Considering initialization, the maximum is 593 addresses starting with bit 53 (see Figure 7.4). Its neighbor bits 52 and 54 however result in only 452 and 294 addresses, but bit 16 to 26 all result in more than 500 addresses. Results for bit 54 to 63 are below brute-forcing.

Clients: The recursive algorithm with thresholds above 16 reveals more addresses than brute-forcing and the low-byte pattern (see Figure 7.5). With a threshold of 20, the recursive algorithm reveals plus 387 addresses in comparison to brute-forcing, and plus 417 addresses in comparison to the low-byte pattern. Noticeably, low-byte performs slightly worse than brute-forcing and reveals 31 addresses less. Considering initialization, only bit 17 and 19 to 21 provide more than 1000 addresses (see Figure 7.6). Starting with bit 1, 14, 15 and 26 to 63 performs worse than brute-forcing.

Manual Patterns: The port pattern includes only 552 probes. Table 7.1 shows that

	Routers	Servers	Clients
Port Pattern	52.6 (CI 1.0)	35.5 (CI 0.9)	43.8 (CI 2.0)
Low-Byte Pattern	204.1 (CI 0.8)	192.8 (CI 1.0)	209.6 (CI 0.8)
Brute-Forcing	203.1 (CI 0.8)	191.8 (CI 1.0)	208.6 (CI 0.8)
Our Algorithm	407.3 (CI 1.7)	274.3 (CI 1.7)	386.1 (CI 81)

Table 7.1: Port Pattern in Comparison to Alternatives

	Routers	Servers	Clients
Top 1	8.1 (CI 0.5)	163.8 (CI 2.7)	707.4 (CI 6.0)
Top 2	6.3 (CI 0.5)	41.4 (CI 0.9)	311.0 (CI 3.8)
Top 3	4.7 (CI 0.3)	18.0 (CI 0.7)	147.4 (CI 2.5)
Top 4	4.5 (CI 0.4)	13.5 (CI 0.8)	140.9 (CI 2.4)

Table 7.2: OUI Pattern: Best Results

these probes reveal 44 client, 53 router and 36 server addresses. The table compares these results to the number of discovered addresses in the first 552 probes of the low-byte pattern, brute-forcing and our recursive algorithm with a threshold of 18. All of them reveal roughly four-times the addresses of scanning with the port pattern, or even more.

Table 7.2 shows the four most successful scanning attempts of the OUI pattern per host group. Every attempt requires 2^{24} probes. For routers and servers a low number of addresses is discovered: The most-frequent OUI reveals only 8 router addresses and 164 server addresses. The most-frequent OUI in clients reveals 707 addresses, but remains a singular result. The second-frequent reveals 311 address, and the remaining below 150 addresses. These are all rather low results considering the roughly 16 million probes per OUI.

7.3.3 Parameter Prognosis

An adversary using our pattern-based algorithm for host discovery likes to know in advance whether a certain start bit is a good choice because the algorithm's results are heavily dependent on initialization. We claim that the results are dependent on the bit ratio as shown in Figure 7.7. This ratio indicates the part of addresses with this certain bit set to one. Low ratios provide better results, than ratios close to 50 %. Starting with

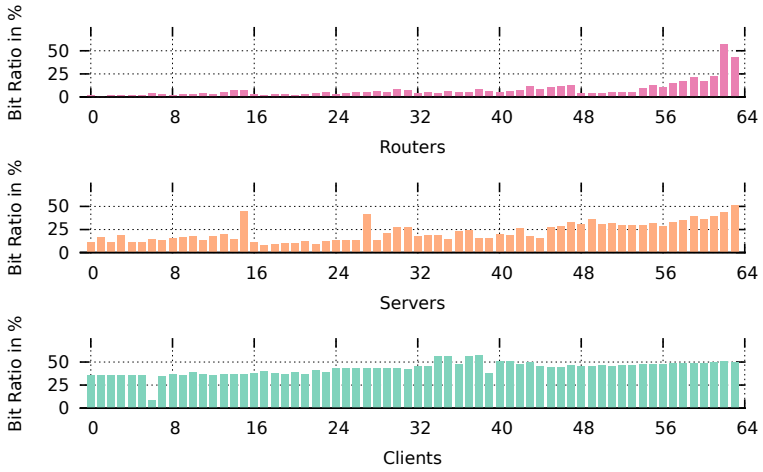


Figure 7.7: Bit Ratio: Ratio of Addresses with a Certain Bit Set to One.

	Routers	Servers	Clients
linear	2519	2148	9424
quadratic	2389	1974	9424
exponential	3641	2738	10109

Table 7.3: Residual Variances of Regression Analysis

such ratios of about 50 % postpones a pattern with a rather high number of appropriate addresses. This impedes our algorithm's intention of prioritizing frequent patterns.

We performed regression analysis to investigate this. Quadratic regression fitted best among linear, quadratic and exponential approaches evaluated by the residual variance, as shown in Table 7.3. The coefficient of determination *R2* is 0.58 for routers and 0.48 for servers. This means that roughly half of the scanning results' variance is determined by the starting bit's bit ratio. Residual variances for client nodes are higher in comparison and regression fitting is of less quality. *R2* is only 0.13 for quadratic regression. The influence of the initial bit on the overall result is minor. Figure 7.8 shows the scatter diagrams for routers, and Figure 7.9 servers respectively. Every crossing indicates a bit ratio and its related scanning result. The resulting quadratic regression curves are added to the scatter diagrams.

7.4 Discussion

Our results on pattern-based scanning approaches are twofold: On the one hand, the manually crafted patterns that are known from the literature turned out to be of limited benefit. On the other hand, we proposed a pattern-based algorithm that automatically

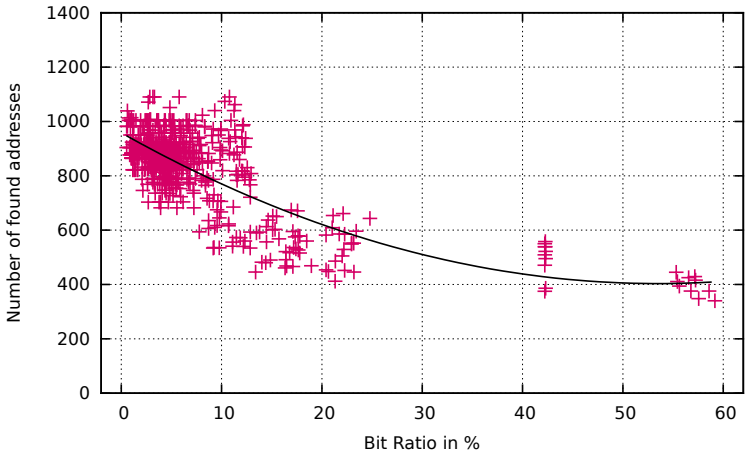


Figure 7.8: Routers: Scatter Diagram and Regression Curve

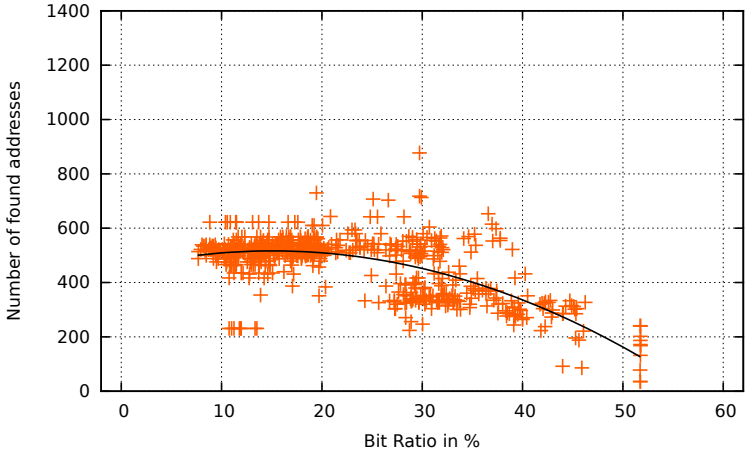


Figure 7.9: Servers: Scatter Diagram and Regression Curve

discovers patterns in a sample and generates addresses based on the found patterns. This novel algorithm is able to outperform brute-forcing and manually-crafted patterns. Focusing on manually-crafted patterns, the *low-byte* pattern solely performs well with servers, and is even worse than brute-forcing for clients. The *port* pattern finds a fourth or less nodes than all other approaches within their first 552 probes, and the *OUI* pattern typically finds less than half of the nodes of our algorithm but requires 16 times the number of probes. Return to our pattern-based algorithm, it outperforms the other evaluated approaches, but its overall performance is dependent on the start bit that is set during initialization. Taking a neighbor bit might already impact the performance negatively. However, we have shown that the algorithm's performance in dependence of the start bit can be pre-estimated by means of simple bit-wise statistics for servers and routers to a certain degree. Pre-estimation with respect to clients is of less quality, and

might be a consequence of the high amount of random addresses generated by the Privacy Extension in the sample data sets. A removal of these temporary addresses from the data set is likely to improve the results on pre-estimation, but also on overall scanning results as their pseudo-randomness might negatively impact the pattern discovery. [204] proposes an algorithm to identify such addresses. Hosts using a temporary address are anyway also reachable via a stable address that might be even easier to guess.

The success of our pattern-based algorithm also highlights that address scanning is feasible with a comparable low effort due to address-inherent patterns, and defense-in-depth in the context of IPv6 reconnaissance seems incomplete; notwithstanding, that already early RFCs as of 2008 advise the assignment of addresses *"that are not obvious to guess"* [229] as low node density alone does not guarantee to be undiscoverable. This points the way towards mitigation of pattern-based scanning approaches. Random or pseudo-random addresses, e. g., [230], repel the threat of finding implicit patterns. Mitigation against our algorithm nevertheless requires more effort than against manually-crafted patterns. To mitigate the threat of manual patterns, choosing an address beyond the patterns is enough. For example, 2001:db8::5:21 is probed by the port pattern, but 2001:db8::6:21 is not.

7.5 Conclusion

In this chapter, we proposed a new methodology for enabling scanning for active IPv6 addresses based on rule mining. Our algorithm automatically extracts patterns from a data set of addresses; the approach is considered a side channel unintentionally revealing implicit patterns. The fundamental idea behind this approach lies in the observation that administrators do not select random addresses when migrating their services to the IPv6 world, but rather rely on patterns. While, opposed to IPv4, probing every single address is not possible for IPv6 due to the sheer amount of existing addresses, this approach uses prediction of patterns based on a sample set of addresses in order to rearrange the scanning order to enable faster retrieval of addresses. While this does not allow scanning the whole IPv6 range, this technique opens up great chances for fast retrieval of a large amount of used addresses. Contrary to approaches based on experience, this method allows the regular recalculation of the most likely patterns in order to detect changes in the typical selection of addresses. Future work includes the implementation of this techniques into a scanning tool, more results are needed with respect to the performance of the algorithm, as well as further research on speeding up searches.

CHAPTER 8

An Attack against the IPv6 Privacy Extension

The power of correlation lies in its capability of making sense from large amounts of data that seem unrelated to each other by combing countless pieces of information [231]. This means that a person's different activities on the Internet can be correlated to each other, and this condensed information typically exceeds what people believe can be found out about their lives. Addresses play a sensitive role in this: On the one hand, an address has to accurately identify the receiver so that traffic reaches its intended destination. On the other hand, address-based correlation enables the attribution of different transactions to the same origin and allows to gain insights into others' Internet behavior. General protection strategies against correlation like an attribute's removal or its encryption seem inadequate for addresses as intermediate nodes require access for appropriate data delivery.

Addressing, in turn, is heavily dependent on the protocol, and IPv6 introduced new aspects in the matter of address-based correlation. Initially, all addresses of an interface were defined to include a globally unique identifier and thus allowed simplest address correlation over an interface's full lifetime [139]. In response, temporary addresses that change by default every 24 hours were introduced. This mechanism is known as the Privacy Extension [202], and is considered as state-of-the-art privacy protection in IPv6. It is implemented in major desktop and mobile operating systems.

In this chapter, we scrutinize the IPv6 Privacy Extension's capability of protecting against address-based correlation, and therefore focus on the algorithm for temporary address generation. We find that once the algorithm's state is known by an adversary, she is able to accurately predict a victim's future addresses. Beyond that, we develop a way that allows an adversary to synchronize to the victim's state by exploiting observed temporary addresses as a side channel, and appraise the attacker's effort to perform our

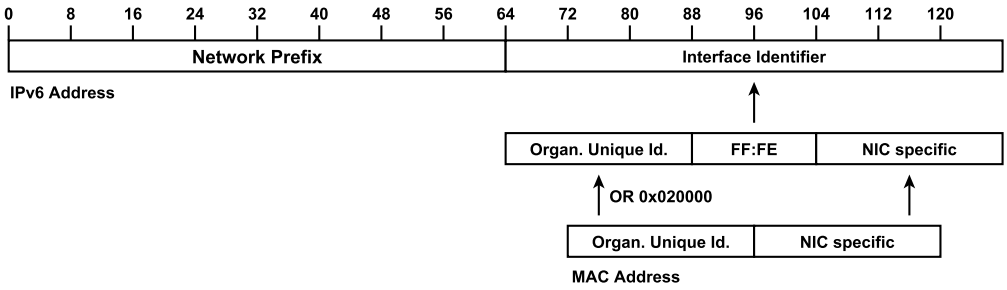


Figure 8.1: IPv6 Addresses using Interface Identifiers in Modified EUI-64 Format

attack with currently available technology. Our results yield 3.3 years of hashing but advances in technology are going to decrease this time period. We highlight mitigation strategies; however, our most important contribution may be the impetus for a revision of the extension's specification.

The remainder of this chapter is structured as follows: Section 8.1 provides details on addressing in IPv6 and the Privacy Extension. Section 8.2 summarizes privacy implications of competing IPv6 addressing standards as well as known vulnerabilities of the Privacy Extension. Section 8.3 describes the assumed attack scenario and is followed by a security analysis of the extension's address generation algorithm that identifies four weaknesses in Section 8.4. Based on these insights, the development of our attack is described in Section 8.5. Its feasibility is discussed in Section 8.6, which is followed by an investigation of current operating systems' vulnerability in Section 8.7. Strategies for mitigation are presented in Section 8.8, and Section 8.9 concludes this chapter.

8.1 Background

This section provides background on IPv6 addressing in general: the address structure, address assignment and their implications for address-based correlation. In a second step, we focus on the IPv6 Privacy Extension and describe its principal idea as well as its algorithm for temporary interface identifier generation.

IPv6 Addressing: IPv6 addresses have a length of 128 bit and are portioned into two distinct parts of equal size as depicted in Figure 8.1. The first 64 bits form the network prefix, and are dependent on a host's location in the network. The remaining 64 bits form the interface identifier (IID) that enables a subscriber's identification on the link. Address configuration for clients is done via Stateless Address Autoconfiguration [150] and does not require human intervention: Routers advertise the network prefix on the network, and hosts form their global IPv6 addresses by combining the announced prefix with a self-generated interface identifier.

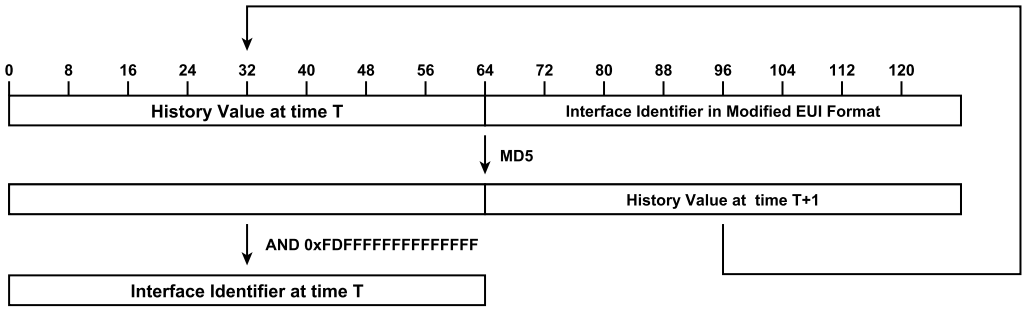


Figure 8.2: Interface Identifier Generation According to the Privacy Extension

The interface identifier was initially intended to follow the modified EUI-64 format [139] that infers an interface identifier from the 48 bit media access control (MAC) address, see also Figure 8.1. The MAC address consists of a 24 bit organizationally unique identifier, and a network interface card (NIC)-specific part of equal size. A fixed pattern of two bytes is inserted between these parts and a universal/local bit is set to one in order to form the identifier.

The MAC address is globally unique and typically remains stable over a host's lifetime[1]. Consequently, the interface identifier that is included in every IPv6 address is globally unique and stable as well. All addresses of a certain host have the same second half, while their network prefix changes according to the visited location. An adversary is thus able to attribute various transactions to the same origin based on the identifier and trace a host's Internet behavior even beyond a certain sub-network. The adversary is further able to retrace a host's movement in the network as the included network prefixes allow localization.

The IPv6 Privacy Extension: The Privacy Extension is presented as a solution that impedes correlation *"when different addresses used in different transactions actually correspond to the same node"* [202]. Its basic principle are interface identifiers that change at a regular interval of typically 24 hours. Hosts form temporary IPv6 addresses from the announced prefix in combination with the current interface identifier, and change the IPv6 address with every newly generated identifier. An expired address is considered deprecated and not used for new connections, but still serves already active transactions.

A host's successive interface identifiers have to be chosen in a way that appears random to outsiders and hinders them in attributing different identifiers to the same origin. Thus the IPv6 Privacy Extension defines an algorithm for a pseudo-random generation of these temporary identifiers as described in the following and depicted in Figure 8.2:

[1]Technically speaking the MAC remains stable over the NIC's lifetime, but we suppose that personal computers, laptops, tablets and mobiles keep their NIC over their whole lifetime.

1. A 64 bit history value is concatenated with the interface identifier in the modified EUI-64 format.

2. An MD5 digest is calculated over the concatenation of the previous step to gain a digest of 128 bit length.

3. The digest's leftmost 64 bits are extracted and bit 6 is set to zero in order to form the temporary interface identifier.

4. The digest's rightmost 64 bits form the next iteration's history value and are stored.

5. In case the generated interface identifier is found to be used by other local devices or reserved, the process is restarted to gain another identifier.

The very first history value is initialized with a random value the first time a system boots. This algorithm is defined for systems with present stable storage, which is necessary to keep the history value across system restarts. Devices like stationary PCs, laptops, tablets and smart phones are typically considered to have such storage. However, in its absence, it is allowed to randomly re-initialize the history value after every system restart.

Temporary IPv6 addresses are assigned in addition to stable addresses in modified EUI-64 format, and do not replace them in order to prevent negative impacts on addressing. Temporary addresses are used in outgoing connections to stay private, while stable addresses make it possible to stay reachable for incoming requests.

8.2 Address Formats and Known Vulnerabilities

Our research has a two-pronged foundation: First, we discuss various IPv6 address structures with respect to privacy, and highlight the IPv6 Privacy Extension's outstanding positions due to its capability to protect against geographical as well as temporal address-based correlation. This further emphasizes why the extension's secure standardization and implementation is an important aspect of IPv6 privacy. Second, we summarize previously discovered vulnerabilities of the Privacy Extension, and illustrate their minor importance in comparison to the new attack that we present in this chapter.

8.2.1 IPv6 Address Formats and Address Correlation

There are ways to form IPv6 interface identifiers for Stateless Address Autoconfiguration beyond the modified EUI-64 format and the Privacy Extension: (1) manually configured stable identifiers, (2) Semantically Opaque Identifiers [230] and (3) Cryptographically Generated Addresses (CGAs) [191]. CGAs, however, require authenticated messages as defined by Secure Neighbor Discovery (SeND) [190] instead of plain Neighbor Discovery [146].

We discus these alternatives with respect to an adversary's capability for address correlation, and consider two distinct aspects of address correlation:

- Temporal correlation refers to address-based correlation over multiple sessions of a stationary host.
- Geographical correlation refers to address-based correlation over multiple sessions of a mobile node.

The difference is the network prefix: A stationary host stays in the same sub-network and includes the same network prefix in all its addresses. A mobile node wanders and changes the network prefix when moving.

Addresses using the Modified EUI-64 format include the globally unique MAC address, and all of a host's addresses are equivalent in their second part. This fact allows the correlation of multiple sessions of a stationary or mobile node, i. e., this type of address is vulnerable to both forms of address correlation and, beyond that, also for active host tracking [232, 233]. Apart from global uniqueness, the same is valid for (manually configured) interface identifiers that remain static.

Semantically Opaque Interface Identifiers are generated by hashing the network prefix and a secret key among other parameters. As the hash calculation includes the address prefix, the interface identifier changes from subnet to subnet and prevents geographical correlation. The identifier, however, remains stable in a certain network, even when returning from another network, and allows temporal correlation over long periods of time. Due to their recent standardization their availability in current operating systems is limited.

Cryptographically Generated Addresses are generated by hashing the public key and other parameters and are bound to certain hosts. Ownership is verified by signing messages that originate from this address with the corresponding private key. The network prefix is included as a parameter into hashing, and a node's CGA changes from network to network, preventing geographical correlation of traffic. However, their generation comes at high computational costs, and prevents address changes as a means of protection against temporal correlation in practise [234]. An approach to overcome the limitation with respect to frequent address change has been proposed [235]. However, CGAs and SeND lack acceptance and are neither widely implemented nor deployed.

The discussion is summarized in Table 8.1, and is accompanied by the capabilities' native availability in the current client operating systems Mac OS X Yosemite, Ubuntu 14.10 (Utopic Unicorn) and Windows 8.1, see Table 8.2. The results emphasizes the unique position of the Privacy Extension: First, it is the only mechanism using Stateless Address Autoconfiguration that is currently deployed at a larger scale and that is intended to protect against traffic correlation. Second, it is the only mechanism that considers protection against temporal as well as geographical address correlation.

	Modified EUI-64	*Stable (Manual)*	*Sem. Opaque Id.*	*CGA*	*Privacy Extension*
Temporal Correlation	-	-	-	-	✓
Geographical Correlation	-	-	✓	✓	✓

Table 8.1: IPv6 Address Formats wrt Protection against Address Correlation

Mac OS X Yosemite	✓	✓	-	-	✓
Ubuntu 14.10	✓	✓	-	-	✓
Windows 8.1	✓	✓	-	-	✓

Table 8.2: IPv6 Address Formats wrt. Availability in Client Operating Systems

In this chapter, we develop an attack that overcomes the belief that the Privacy Extension provides adequate protection against address correlation. The attacks leaves a gap that cannot be filled by another address mechanism, and highlights the importance of revisiting the extension's current definition.

8.2.2 Known Vulnerabilities of the Privacy Extension

Drawbacks of the IPv6 Privacy Extension were discussed before, and follow two principal directions. First, its design does not impede active tracking, e. g., by using ping. Temporary addresses are assigned in addition to stable ones, and an adversary can still actively probe multiple subnets for a certain interface identifier in order to trace a host's movement. The respective specification, however, explicitly states its intention to protect solely against passive eavesdroppers, and not against active adversaries [202]. Beyond, the Privacy Extension further introduces a covert channel as regular changing addresses become normal.

Second, shortcomings in the extension's protection against address correlation are known. A node does not have to change its interface identifier when moving to a new network prefix. Thus, tracking a host's movement remains feasible within an identifier's lifetime

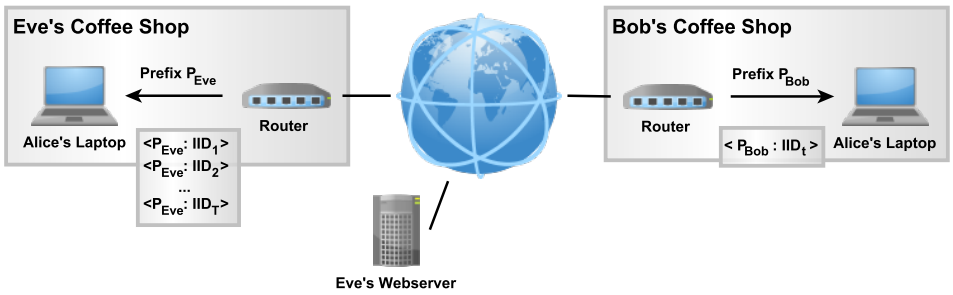


Figure 8.3: Attack Scenario

of typically 24 hours [236, 232]. For mitigation, the inclusion of the network prefix into the interface identifier calculation was proposed [236]. The respective specification also allows the change of an identifier in such a situation [202]. Our attack supports the second direction, and highlights that adversaries are able to perform address correlation even when the Privacy Extension is used. In comparison to known attacks, our attack cannot be fully mitigated within the specification's limitations.

8.3 Attack Scenario

Our attack scenario is depicted in Figure 8.3 and assumes full IPv6 deployment. We assume three stakeholders named Alice, Bob and Eve. Alice loves coffee, and regularly visits coffee shops. Then, she brings her laptop with her, and uses the offered Internet access to read mails or to chat. Bob and Eve each run a coffee shop, and provide Internet access to their guests. They deployed Stateless Address Autoconfiguration, and their routers advertise the respective IPv6 network prefix so that customers are able to configure their global IPv6 addresses by connecting the prefix with their self-generated interface identifiers. Bob's router advertises the prefix P_{Bob}, Eve's router advertises P_{Eve}. Eve further runs a webserver to advertise current offers. She records her coffee shop's local traffic, and logs visits to her webserver.

Alice visits Eve's coffee shop for T successive days[2], and connects her laptop to the coffee shop's local network. Eve's router advertises P_{Eve}, and Alice's laptop configures a stable IPv6 address from this prefix and the stable interface identifier. Alice has enabled the IPv6 Privacy Extension, and thus temporary addresses are created in addition to the stable address by combining the prefix with the interface identifier of the day. Alice's

[2]Although the T days do not necessarily have to be successive, we claim so here for better readability. In case days are missing, e. g., due to weekends, one simply has to consider these gaps when calculating the current state.

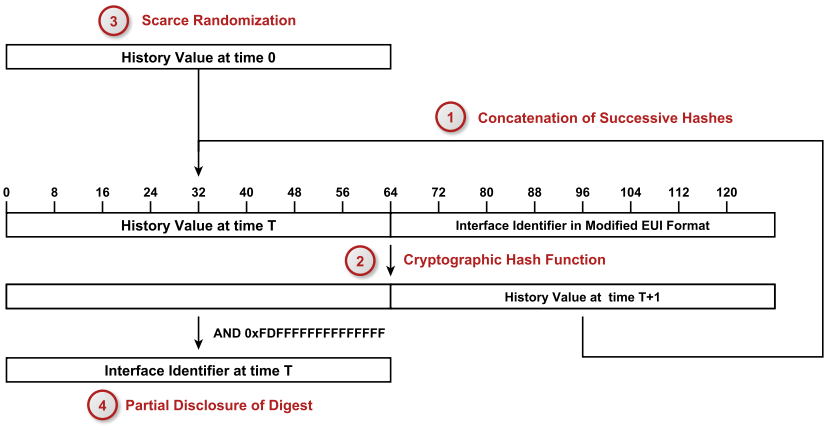


Figure 8.4: The Privacy Extension's Characteristics Impacting its Quality of Protection

temporary addresses are $< P_{Eve} : IID_1 >, < P_{Eve} : IID_2 >, ..., < P_{Eve} : IID_T >$ for day $1, 2, ..., T$.

After T days, Alice stops going to Eve's coffee shop. On an arbitrary day t ($t > T$), Alice visits Eve's competitor Bob. She connects her laptop to Bob's local network. Bob's router announces the prefix P_{Bob}, and Alice's laptop forms a stable identifier from this prefix. In addition, the Privacy Extension generates a temporary address $< P_{Bob} : IID_t >$. On this day, Alice visits Eve's website to check current offers and causes a respective log entry.

Eve is interested tracing her customers' activities, and wants to find out whether (1) Alice is still interested in her offers and visits the webserver, and whether (2) Alice is drinking coffee at a competitor.

We refer to this scenario in the remainder of the chapter for illustration of our attack. This scenario was developed due to its representativeness for day-to-day life, but we are sure that there are plenty of alternative scenarios. The preconditions for an adversary are moderate: She has to gain a victim's MAC address and T successive interface identifiers that have been generated by the Privacy Extension. The MAC address is gained from local traffic as in the presented scenario, or inferred from the stable IPv6 address in case the latter is in Modified EUI-64 format. Interface identifiers are included in the temporary addresses, and are inferred from there.

8.4 Security Analysis

In this section, we perform a manual security analysis of the Privacy Extension's algorithm for temporary interface identifier generation as defined in [202] and presented in Section 8.1. Our analysis reveals four striking characteristics that facilitate the prediction of

future interface identifiers. While some of them might seem minor in isolation, their combination forms a reasonable attack vector as described in Section 8.5. In this section, we consider each characteristic separately: First, we describe the characteristic and highlight the specification's argumentation in its favor. Next, we infer implications on security. Figure 8.4 contrasts the algorithm for temporary address generation with the discussed characteristics; the depicted numbers are consistent with the following paragraphs.

(1) Concatenation of Successive Hashes: Interface identifiers are based on MD5 digests that are chained with each other because an iteration's result is partly included into the next hash calculation. The RFC states that *"In theory, generating successive randomized interface identifiers using a history scheme [...] has no advantages over generating them at random,"* [202] but claims an advantage in case two hosts have the same flawed random number generators. Performing Duplicate Address Detection would make both hosts recognize their identical identifiers and trigger the generation of new identifiers. However, the flawed random number generators would again provide identical numbers, leading to identical identifiers. The presented algorithm is said to avoid this as the inclusion of the (globally unique) interface identifier in Modified EUI-64 format leads to different temporary interface identifiers in the next iteration.

It remains unclear why the inclusion of a globally unique identifier, e. g., in Modified EUI-64 format, requires working with a history scheme, i. e., the concatenation of successive hashes. We believe that inclusion of a globally unique interface identifier and a random value into MD5 digest calculation is sufficient. It seems unlikely that sequences of equivalent random numbers result in successive collision in case a globally unique identifier is included into calculation.

The concatenation does not only appear dispensable with respect to the discussed aspect, but also negatively impacts the algorithm's quality of protection. Successive interface identifiers are dependent on each other, and today's state influences future identifiers. An adversary might exploit this to predict a victim's future identifiers.

(2) Cryptographic Hash Function: The Privacy Extension aims to create random-appearing interface identifiers, but states that pseudo-randomness suffices *"so long as the specific sequence cannot be determined by an outsider examining information that is readily available or easily determinable"* [202]. For the algorithm, MD5 with its adequate properties with respect to randomization has been *"chosen for convenience"* [202].

MD5 is considered broken, but a general dissolution would be an overshooting reaction: MD5 turned out to be prone to collisions that can be found within seconds on commodity hardware [237]. Pre-image attacks are still of high complexity and remain practically infeasible. The Privacy Extension uses MD5 for randomization, and neither relies on collision resistance nor pre-image resistance. Taking these considerations into account, the extension's choice of MD5 is justifiable.

MD5 is, however, a comparably fast hash function and the more hashes per second, the more feasible brute-force search becomes. This especially holds in combination with a limited input range. In 2012, a cluster of four servers hosting 25 off-the-shelf graphics processing units (GPU) achieved 180 Gigahashes per second [238], and time is usually in favor of the adversary as technology moves forward.

(3) Scarce Randomization: The RFC claims that *"To make it difficult to make educated guesses as to whether two different interface identifiers belong to the same node, the algorithm for generating alternate identifiers must include input that has an unpredictable component from the perspective of the outside entities that are collecting information"* [202].

Our analysis, however, identifies only scarce unpredictability in the algorithm for temporary address generation. Every iteration includes 128 bits into MD5 digest calculation:

- 64 bit of the former iteration's result, i. e., the remainder of the MD5 hash that was not used for the temporary interface identifier, and
- the 64 bit interface identifier in Modified EUI-64 format. This identifier is not kept secret. An adversary might infer it from the stable IPv6 address that is assigned in addition to temporary addresses or from the MAC address. 17 bit of this identifier are fixed and thus the same for all nodes anyway.

In conclusion, there is no entropy added per iteration and this fact makes prediction of future identifiers easier as there are less possibilities. The only unpredictable component of the presented algorithm is the very first history value of 64 bit that should *"be generated using techniques that help ensure the initial value is hard to guess"* [202].

(4) Partial Disclosure of Digest: A temporary interface identifier is generated by taking *"the leftmost 64-bits of the MD5 digest and set bit 6 [...] to zero"* [202]. The gained interface identifier forms a temporary IPv6 address when combined with the current network prefix. The address is present in packets' address fields and accessible by others.

As a consequence, an eavesdropper gains 63 bit (one bit is overwritten with zero as mentioned above) of the calculated MD5 digest. This eavesdropped part does not present the algorithm's internal state, i. e., the history value, but both are part of the same MD5 digest. In conclusion, 63 bit of every iteration's MD5 digest is readily available to outsiders without any further processing effort and form a side channel of the algorithm's internal state. The algorithm leaks information but does not add entropy in an iteration.

8.5 Attack Design

We will now explain the steps of our attack in detail. We will include the characteristics that have been found in the security analysis of Section 8.4. In a first step, we will analyse

the predictability of future addresses if the current state (history value) is known. As this turns out to be promising, we investigate methods to gain the current state. Finally, we summarize our full attack.

Predictability of Future Identifiers: For rather unambiguous prediction of future temporary identifiers, two requirements have to be met. First, future identifiers have to be dependent on the current state of the algorithm. Second, the calculation of the next identifier should include little randomness. The less random input, the better predictability.

We know from the previous section that both conditions apply to the IPv6 Privacy Extension: Interface identifiers are based on concatenated hashes. A part of the digest is used for the identifier, the other is called the history value and used as an input for the next calculation. An iteration's input is twofold – the mentioned history value and the interface identifier in Modified EUI-64 format that is inferred from the MAC address. This means that there are no unpredictable components that are included. In conclusion, an adversary that is aware of the victim's history value and its MAC address is able to calculate the next temporary interface identifier according to the following recipe:

1. Infer the interface identifier in Modified EUI-64 format from the victim's MAC address. This requires the insertion of a fixed pattern of two byte, and setting bit 6 as described in Section 8.1.
2. Concatenate the victim's current history value with the interface identifier in Modified EUI-64 format generated in step 1.
3. Calculate the MD5 digest of the concatenation of step 2.
4. Extract the first 64 bits from the calculated digest and unset bit 6 to form the next temporary interface identifier.
5. Extract the remaining 64 bits from the digest and form the next history value.

This way an adversary is not only able to compute the next interface identifier, but all future identifiers by repeating the described steps. As a consequence, it seems worth developing methods to gain the algorithm's internal state.

Synchronization to the Current State: The internal state could be leaked, e. g., by means of malware, but this approach would imply an active adversary that does not simply eavesdrop. In the following paragraphs, we show that eavesdropping over a number of consecutive days is sufficient to gain the internal state: As described in Section 8.4, a temporary interface identifier that is included into an IPv6 address inherently discloses 63 bit of an iteration's MD5 digest. While the disclosed part is not the internal state, it is nevertheless related to the latter as both are clips of the same MD5 digest. The disclosed interface identifier can be considered a side channel of the internal state.

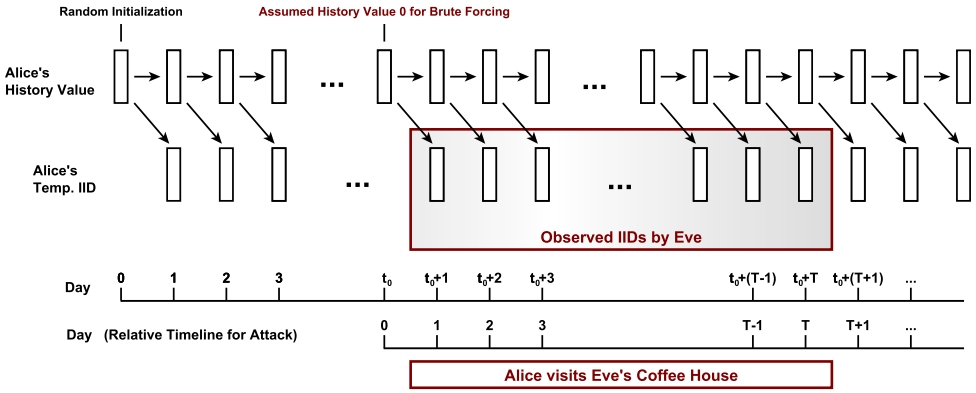


Figure 8.5: Synchronization to Current State

Figure 8.5 depicts a situation like our attack scenario from Section 8.3. The victim's very first history value is randomly initialized at day 0 and determines the history value and the temporary interface identifier of day 1; the history value of day 1 in turn determines history value and temporary interface identifier of day 2 and so on. The randomly assigned history value at day 0 determines one of 2^{64} deterministic sequences that the victim's interface identifiers follow.

An adversary might probe all possible values for this history value at day 0, and compare the first half of the MD5 digest with the interface identifier of day 1. If they are equal this value might represent an appropriate candidate. As it is only possible to compare 63 bit of the MD5 digest, it is likely that numerous candidates remain. The adversary thus extracts the second half of the digest as a candidate for the history value at day 1, includes it in another iteration containing an MD5 calculation, compares the result with the interface identifier at day 2 and further shrinks the candidate set until a single candidate remains. Then, the adversary has identified the internal state.

It is, however, unlikely that an adversary observes the very first temporary addresses that a victim generates after its installation; an adversary rather observes an arbitrary sequence of T successive addresses starting at day $t_0 + 1$ as indicated in Figure 8.5. Due to the algorithm's structure, the adversary then assumes the history value at day t_0 to be randomly initialized without loss of generality. The adversary does not have to know the number of temporary addresses that the victim has generated before being recorded. For this reason, we added a relative time line for the attack in the figure for readability.

Composite Attack: Based on the attack scenario of Section 8.3, the gained insights of the previous paragraphs and Figure 8.5, we summarize Eve's steps towards predicting Alice's future identifiers.

On Alice's first visit at Eve's coffee shop on day 1, Eve has to perform the following steps:

- *Data Extraction from Traffic Records:*
  Eve records the local traffic in her coffee shop, and is thus able to read Alice's MAC address from Ethernet headers as well as her temporary IPv6 address. From this temporary IPv6 address, Alice extracts the last 64 bits that are the interface identifier for the first day IID_1.

- *Generation of Modified EUI-64 Interface Identifier:*
  Eve infers Alice's interface identifier in Modified EUI-64 Format from the MAC address by inserting a fixed pattern of two bytes and setting bit 6 as described in Section 8.1. Alternatively, she might read the identifier in Modified EUI-64 format directly from Alice's stable IPv6 address.

- *Reduction of Candidate Set:*
  Eve probes all possible values for the assumed initial history value at day 0, concatenates the value with the stable identifier in Modified EUI-64 format, and calculates the MD5 digest. If the first part of the MD5 digest equals Alice's current temporary address[3], the remainder of the digest forms a candidate for the next iteration's history value and is added to the candidate set of the first day C_1. In this step, Eve reduces the initial candidate set C_0 of 2^{64} alternative sequences to a smaller set C_1 that is stored for the next day.

On every further visit of Alice at Eve's on subsequent days t with $1 < t \leq T$, Eve performs:

- *Data Extraction from Traffic Records:*
  Eve extracts today's temporary interface identifier IID_t from Alice by reading the traffic records.

- *Further Reduction of Candidate Set:*
  Eve probes all values for the history value that are present in yesterday's candidate set C_{t-1}, concatenates the values with the stable identifier in Modified EUI-64 format, and calculates the MD5 digest. If the first part of the MD5 digest equals Alice's current temporary address IID_t, the remainder of the digest forms a candidate for the next iteration's history value and is added to the candidate set C_t. In this step, Eve further reduces the number of alternative sequences to a smaller set that is again stored for the next day.

[3]The comparison is done on 63 different bits (0-5 and 7-63); bit 6 is always set to zero in temporary addresses, see Section 8.1.

This is performed whenever a new temporary address is available until a single candidate remains. This single candidate represents the algorithm's internal state, the history value, and allows to predict future addresses from now on.

On every further day t with $t > T$, Eve is able to anticipate Alice's temporary interface identifier for this day:

- *Anticipation of Current Temporary Address:*
  Eve concatenates the history value of day T with the stable identifier in Modified EUI-64 format and calculates the MD5 digest. She extracts the history value, and repeats the calculation with the new history value. In total, $(t - T)$ MD5 digest calculation are performed.
- *Assemblage of the Interface Identifier:*
  Eve forms Alice's interface identifier IID_t from the first part of the last MD5 digest by setting bit 6 to zero.

With this knowledge, Eve is able to search her web server's logs for the calculated temporary identifier and attributes certain visits to Alice. At the same time, the prefix that the temporary identifier is concatenated with to form an IPv6 address provides information on the sub-network that Alice resided at the time of the page visit. If this is equivalent to Bob's assigned prefix, Eve is able to infer that Alice drank coffee at Bob's coffee shop.

8.6 Feasibility

In the previous sections, we identified weaknesses of the IPv6 Privacy Extension and developed an attack exploiting these characteristics. The question on the attack's practicability with respect to today's technology remains, and is discussed in this section. Three aspects have to be considered: (1) the minimum number of observed interface identifiers, i. e., the number of days that Alice has to visit Eve's coffee shop, (2) the expenditure of time for brute-forcing, and (3) the storage capacity to save the candidate set for the next day. Finally, a modified version of our attack for limited storage capabilities is presented.

Number of Address Observations: Alice has to visit Eve's coffee shop so often that Eve gains enough temporary identifiers for synchronization to the internal state. We assume that Alice generates one temporary address per day as recommended by the RFC [202], and an iteration of the attack corresponds to a day.

On the first day, Eve probes 2^{64} potential values for the history value and compares their MD5 digest to the observed interface identifier of Alice. The unequal ones are excluded, and the appropriate ones form the candidate set C_1 of potential values for the next day. The size of the candidate set is dependent on the ratio of candidates that Eve is able

to reject per day. With p being this ratio, the size of the candidate set C_t for day t is calculated as follows

$$|C_t| = 2^{64} \cdot (1-p)^t \tag{8.1}$$

Eve has to repeat the explained step until a single candidate remains, i. e., $|C_t| = 1$, and the minimum number of days T_{min} is calculated as follows

$$T_{min} = ceil\ \frac{log(2^{64})}{log(p-1)} \tag{8.2}$$

The more candidates can be excluded per iteration, the less successive interface identifiers have to be known by Eve. If Eve is able to reduce the candidate set by only 50 % every day, the minimum number of days is 64. A reduction by 99 %, 99.99 %, 99.9999 % shortens this to 10, 5, 4 days.

Time Expenditure for Brute-forcing: Every iteration requires brute-forcing the current candidate set C_t, and means an MD5 digest calculation for every candidate. Assuming a hash rate r indicating the number of calculated hashes per second, the total time T_{Brute} for brute-forcing is calculated as follows

$$T_{Brute} = \frac{1}{r} \sum_{i=0}^{T_{min}} |C_i| = \frac{2^{64}}{r} \sum_{i=0}^{T_{min}} (1-p)^i \tag{8.3}$$

Assuming $1 - p < 1$[4], the equation is bounded as follows and allows an estimation of the total time expenditure for MD5 brute-forcing

$$T_{Brute} < \frac{2^{64}}{r} \sum_{i=0}^{\infty} (1-p)^i = \frac{2^{64}}{r} \cdot \frac{1}{p} \tag{8.4}$$

A hash rate of 180 G/s with MD5 is feasible [238]. The more candidates can be excluded, the less time is required. If Eve is able to reduce the candidate set on average by only 50 % every day, the time for brute-forcing remains 6.5 years, a reduction by 99 % shortens this to 3.3 years. Time expenditure appears high at the first sight, but time plays for the adversary, and advances in technology are likely to decrease this effort. It is likely that faster hashing is already feasible today as the given hash rate was measured at a cluster of 25 consumer GPUs back in the year 2012 and GPUs have recently experienced extensive advancement.

[4] p is the portion of candidates that can be excluded per iteration.

Storage of Candidate Set: Appropriate candidates for the history value have to be stored for the next iteration. The history value size is 8 byte, and the storage demand S_t is dependent on the size of the candidate set.

$$S_t = |C_t| \cdot 8\,byte = 2^{64} \cdot (1-p)^t \cdot 8\,byte \tag{8.5}$$

The following calculation considers the first iteration due to its worst case character[5]: If Eve is able to reduce the candidate set on average by only 50 % every day, the storage demand for the first iteration is 74 Exabyte, a reduction of 99 %, 99.99 %, 99.9999 % reduces the storage demand to 1.5 Exabyte, 15 Petabyte, 148 Terabyte.

This storage demand, however, can be circumvented by a modification of the attack. In our initial design of Section 8.5, Eve synchronized to Alice's state simultaneously to her coffee shop visits, but Eve might alternatively perform the attack retroactively. Therefore, she stores Alice's successive interface identifiers for T_{min} days before starting the attack. Instead of storing an appropriate candidate after the first iteration, she performs the second, third, etc. iteration with this candidate as long as it appears appropriate. Otherwise, it is rejected. This way the storage demand is reduced to a few bytes for execution of the algorithm for temporary interface identifier generation.

8.7 Implementation in Operating Systems

In this section, we assess current operating systems that support the IPv6 Privacy Extension with respect to their individual vulnerability. We tested Mac OS X Yosemite, Ubuntu 14.10 (Utopic Unicorn) and Windows 8.1 Enterprise as representatives of the three major ecosystems on clients. In doing so, we faced the challenge that we cannot access the respective sources of all operating systems, and had to rely on the externally observable pattern of successively generated interface identifiers. A machine running an operating systems that implemented the Privacy Extension as described in the respective RFC has to generate the same sequence of successive interface identifiers whenever originating from a defined initial state. The sequence appears unchanged when faced with some external factors, while changing in dependence of other factors. The specific influencing factors are discussed later in this section.

For checking the stated premise, we created a setup of two virtual machines running in VMWare Workstation 11 and Fusion Pro 7. The machines were virtually connected for networking. One ran the tested operating system; we refer to this machine as the testee. To save time, we decreased the preferred lifetime on all operating systems and forced the generation of a new temporary address at an interval of twelve minutes. We finally created a snapshot of the testee that made it possible to return it to the initial state after every test. The testee generated temporary addresses after a router's announcement of a network prefix. The second virtual machine thus ran Ubuntu 14.10 simulating this

[5]The candidate set C_0 does not have to be stored as it contains all 2^{64} possible values.

	Deterministic Sequence	*Time-Invariance*	*Prefix-Invariance*	*Restart-Invariance*	*MAC-Variance*
Windows 8	✓	✓	✓	✗	✓
Ubuntu 14.10	✗				
Mac OS 10.10	✗				

Table 8.3: Temporary Address Characteristics wrt to Client Operating Systems

router; to send ICMPv6 Router Advertisements the tool *fake_router6* from the *thc-ipv6* toolkit [193] was used. We recorded the temporary addresses of the testee by means of local scripts.

Using the above premise, we tested the operating systems for five criteria. First, repeating the test without any changes multiple times has to result in the same sequence of successive interface identifiers due to the algorithm's determinism. If this holds, the sequence is checked for their dependence on various influencing factors. The algorithm has to be invariant to time, the announced prefix as well as system restarts and provide the same sequence of identifiers, while it has to be variant to a change of the MAC address. These conditions are inferred from the algorithm's definition in the respective RFC: Neither the point in time of address generation is included into the calculation nor the identifier's lifetime. Thus, a later repetition of the experiment or a change in the interval may not have an impact on the identifiers. The same holds for the announced network prefix. The algorithm has to be invariant to system restarts as the current state has to be stored in stable storage; all the tested operating systems require the availability of such a storage. In contrast, the MAC address is included into the calculation, and its change should result in different identifiers. These are necessary criteria, and are not sufficient criteria. The results of our tests are shown in Table 8.3.

Ubuntu 14.10 does not generate deterministic sequences, and its temporary interface identifiers appear to be assigned by a random number generator without following the defined algorithm. A review of the source code[6] supports this. Mac OS X Yosemite showed the same behavior.

Windows 8.1 provides the same sequence whenever originating from the same state, and

[6]Kernel 3.16.0, /net/ipv6/addrconf.c, line 1898

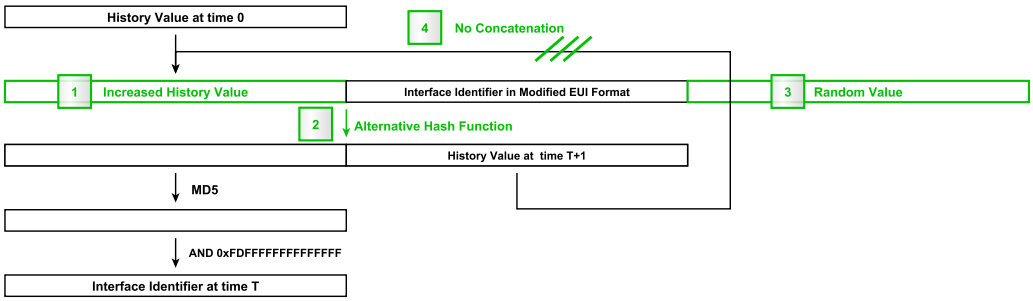


Figure 8.6: Mitigation Strategies for Generation of Temporary IIDs

further fulfills the conditions of time and prefix invariance as well as MAC variance. Restarting the machine or the interface, however, influences the sequence. Thus, we assume that Windows 8.1 implements the Privacy Extension's version for systems without presence of stable storage. In such a case, the first history value after a restart is randomly assigned. This assumption coincides with the fact that we could not find any appropriate history value in the Windows Registry analysing Registry diffs. Further signs supporting our assumption are the collaboration of Microsoft in the definition of the RFC, as well as the algorithm's description in older TechNet Library articles [239].

The gained insights lead to the following conclusion: While Ubuntu 14.10 and Mac OS X Yosemite seem to be immune to our attack, Windows 8.1 appears to be vulnerable – admittedly to a decreased extent as reinitialization of the history value is performed with every restart. However, systems that are continuously running for longer periods or using sleep mode remain vulnerable; and sleep mode is widely used for laptops. For interest, the operating systems' protection to our attack is gained by disobeying the Privacy Extension's standard. Ubuntu and Mac OS seem to totally ignore the proposed generation algorithm, while Windows 8.1 appears to implement the alternative for systems without stable storage albeit it assumes such storage according to its system requirements.

8.8 Mitigation

In this section, we recommend changes to the address generation mechanism for mitigation of our attack. We propose two kinds of strategy: The first aims at impeding synchronization to the algorithm's current state, while the other removes the predictability of future identifiers in general.

Restraint of Synchronization: Our attack is based on the fact that an adversary is able to learn a victim's state by observating them over multiple days, and one might hamper an adversary's synchronization to the algorithm's internal state for mitigation. These strategies do not offer protection in case the state is leaked. The following

explanations are supported by Figure 8.6; the numbers in the figure match those provided in the following paragraphs.

(1) An increased history value would imply improved randomization and increase the size of the initial candidate set C_0, see Equation 8.1. As a consequence, the adversary has to observe more successive identifiers according to Equation 8.2, and time expenditure for brute-forcing increases, see Equations 8.3 and 8.4. The algorithm's current design, however, does not allow an isolated increase of the history value. The MD5 digest's first half forms the temporary interface identifier and its second the current history value. Beyond, there are no bits available that could serve as additional bits for an increased history value. Thus, this strategy would require the replacement of MD5 by another hash function.

(2) MD5 is considered insecure, and its replacement by a state-of-the-art hash function seems tempting. MD5 is vulnerable to collision attacks, and insecure for applications that rely on collision resistance, e. g., as necessary for certificates [240]. The IPv6 Privacy Extension, however, exploits a hash function's randomization, and replacing MD5 with the currently used SHA-265 would only modestly increase brute-force effort [241].

Removal of Identifiers' Predictability: Another precondition of our attack is the dependency of future identifiers on the current state and predictable inputs only. The following mitigation approaches tackle this issue by removing the predictability of future identifiers in different ways.

(3) Including a random value in every iteration makes the digest dependent on more inputs, and adds unpredictability with every new interface identifier. This is the major difference to an increased history value as mentioned above that solely increases randomization at the algorithm's initialization. Even if the current state is leaked, it is impossible to accurately predict future interface identifiers. Moreover, this measure does not require a dissolution of MD5.

(4) A removal of the concatenation would result in successive addresses that are not related to each other; instead, the history value could be randomly initialized for every new address. A similar but more limited approach is defined by the Privacy Extension's standard, but only for devices without stable storage [202]. As such systems are not able to store the history value across system restarts, they are allowed to randomly initialize the first history value after a reboot. Their vulnerability is thus dependent on their typical restart intervals in comparison to the temporary addresses' lifetime. Nevertheless, it seems curious that an alternative algorithm for specific devices is more secure than the standard algorithm.

Alternatively, temporary interface identifiers could be randomly assigned without such a complex algorithm. A host's vulnerability to address correlation is then dependent on the quality of its random number generator. We see advantages in this approach because high-quality random number generators are necessary in modern operating systems on personal computers, laptops and mobiles anyway. The Privacy Extension would benefit

from this quality and further be updated automatically with every improvement of the number generator. For systems without an appropriate random number generator, an alternative would have to be available. This practice is opposed to today's standard that defines a rather complex algorithm *"to avoid the particular scenario where two nodes generate the same randomized interface identifier, both detect the situation via DAD, but then proceed to generate identical randomized interface identifiers via the same (flawed) random number generation algorithm"* [202] and lowers security for all systems that implement the Privacy Extension.

Finally, we considered the question which mitigation strategies are in accordance with the current specification, and have drawn the following conclusions: (1) It is allowed to use another hash function instead of MD5. The brute-force effort would, however, increase only modestly, and a replacement brings only limited protection. (2) The history value is allowed to be randomly re-initialized after every system restart, but this behavior is restricted to systems without stable storage. However, a variety of systems that implement the Privacy Extension like personal computers, laptops, tablets or mobiles do not lack stable storage, and have to follow the standard variety of the algorithm. (3) The Privacy Extension is considered more secure the shorter the temporary addresses' lifetime. This inherent belief has to be revised with respect to the presented attack because more addresses are provided to the adversary within the same time interval, making synchronization to the current state easier.

8.9 Conclusion

The IPv6 Privacy Extension aims to protect privacy by regularly changing the address, and defines an algorithm for the generation of interface identifiers that are combined with the advertised network prefix to form temporary IPv6 addresses. In this chapter, we presented an attack that questions the extension's capability of protection: An adversary is able to predict future temporary interface identifiers once the internal state is known, and is further able to synchronize to this internal state by exploiting the victim's previous interface identifiers as a side channel. In consequence, an adversary knows interface identifiers belonging to the same host; in turn, she is able to perform address-based correlation of different transactions and infer (private) details about people's Internet behavior. Moreover, an adversary might even retrace a host's movement in the network based on the network prefixes that are included in the respective addresses.

The presented attack is worthwhile as it does not solely identify a privacy vulnerability but questions a whole measure for privacy protection. The Privacy Extension was developed with the intention to impede address-based correlation, and our attack shows that it does not meet its goal. Nevertheless, we believe that the general idea of temporary addresses is valuable, and recommend a revision of the algorithm for interface identifier generation. We want to highlight the fact that merely replacing MD5 does not solve the problem, as the vulnerability arises from the concatenation of successive interface identifiers, scarce randomization and information leakage via a side channel. MD5 just

makes the attack easier due to its fast nature. Proper mitigation within the current definition appears impractical, and we want to stress the importance of strategies beyond today's specification.

Operating systems appeared less vulnerable than originally assumed. This does not, however, oppose a revision, as their robustness is gained by silently disobeying the standard and should not be held as a virtue. The standard in its current form can tempt developers to implement a version of the Privacy Extension that is vulnerable to side channels, and should be adapted soon. This utmost concern is further emphasized by the fact that the Privacy Extension is the only widely deployed IPv6 mechanism using Stateless Address Autoconfiguration that is intended to protect against temporal as well as geographical address correlation.

Part III

Epilog

CHAPTER 9

Conclusion and Future Work

Secret communication characterizes clandestine approaches of communication and encompasses *covert channels*, *side channels* as well as approaches of *obfuscation*. Secret communication serves manifold goals like industrial espionage, compliance checking, reconnaissance or malware communication; and a full attack typically consists of three successive steps: (1) *channel development*, (2) *information extraction* and (3) *information exploitation*. While previous work focuses on the aspect of channel development, the thesis at hand includes all three aspects asking the following question: *Which information is gained by an adversary exploiting secret communication, and which advantages does the adversary take from this information?* This line of action further implies that secret communication is dependent on the application scenario; thus, we investigate side channels in *cloud computing* on the one hand, and with respect to the *Internet Protocol version 6 (IPv6)* on the other hand. While the first represents a new operational model with additional functionality through the reuse of existing technologies, the latter is a new technology replacing its predecessor with almost the same functionality to overcome address scarcity on the Internet.

We contribute four full attacks using side channels, two for each application scenario; Table 9.1 highlights them with respect to the three attack phases. In Chapter 4, we examined the functionality scope and quality of firewalls that are provided by public IaaS cloud providers. The gained information on filtering behavior might be exploited in two ways. In good faith, a customer might decide for additional protection, e. g., a host-based firewall, as a consequence of discovered limitations of the default firewalls, e. g., their lacking statefulness. In bad faith, an adversary might evade the firewall in order to strike the virtual machine behind, e. g., using fragmentation attacks. In Chapter 5, we measured traffic patterns of virtual instances residing at Xen hypervisors to infer all configuration parameters that are related to bandwidth throttling. We have shown that an adversary is able to adjust a denial-of-service attack based on the same mechanisms to its victim

Chapter	Development Phase Channel	Extraction Phase Information	Exploitation Phase Attack
4	Firewall Filtering	Firewall Behavior	Compliance Checking Firewall Evasion
5	Network Throttling	Configuration Parameters	Compliance Checking Denial-of-Service
7	IPv6 Addresses	Implicit Address Patterns	Reconnaissance
8	IPv6 Addresses	Internal State	Address Correlation

Table 9.1: Contributed Side Channels wrt. Attack Phases

with this information; in contrary, a benign customer could check compliance with the service contract, i. e., whether the providers assigns as much bandwidth as promised.

With respect to IPv6, we developed two side channels, see Chapter 7 and 8, both use IPv6 addresses as their channel. The first discovers implicit address patterns that are exploitable for reconnaissance, i. e., the discovery of previously unknown victims. By means of the gained patterns, we generate potential addresses making active probing (scanning) with IPv6 possible in the first place as scanning even the smallest IPv6 subnetwork exhaustively would take longer than the sun's lifespan. Addresses generated by the IPv6 Privacy Extension had been considered to be insusceptible against such pattern exploitation. We disprove this belief; first, successive addresses are related to each other; second, future address are predictable once an internal state is known; and finally, this internal state can be inferred using former addresses as a side channel. This makes users of the Privacy Extension vulnerable to address correlation, i. e., the attribution of different Internet transactions to a single user using solely IP addresses.

The thesis at hand shows that side channels become important whenever information is not accessible in a regular manner; this fact is especially apparent when using the side channel in good faith. If a customer does not receive enough technical information from the cloud provider, the first is tempted to use side channels to infer details on firewall functionality or network bandwidth configuration. If exhaustive Internet-wide scanning, as in IPv4, becomes infeasible with IPv6, researchers snatch for more information in order to generate likely address candidates.

It is further noteworthy that cloud computing forms a substrate for a variety of channels exploiting very diverse technologies, and this work solely discussed network-based secret communication in cloud computing not mentioning all the other ways of secret communication using CPU load or cache hits; whereas IPv6 offers only a limited number of approaches and these are all classic storage channels. Beyond, it appears that the

IPv6 address is outstanding among these channels as general protection strategies like an attribute's removal or its encryption fail; both our channels exploit addresses.

For future work, we plan to continue our efforts on secret communication in both scenarios, cloud computing as well as IPv6, in a threefold way. First, academia seems to cultivate side channels in cloud computing; on the contrary, business appears to be unaware despite such channels' immense potential to check cloud providers for compliance. Thus, we plan to combine known side channels into a single tool in order to provide research outcomes in a usable way to the public. For professional use, some challenges however remain. One the one hand, side channels are dependent on the underlying infrastructure and thus on the provider; on the other hand, cloud infrastructures change over time, and the tool would require regular adaptions.

With regard to IPv6 addressing, we aim to achieve a revision of the IPv6 Privacy Extension and its fast re-implementation in vulnerable operating systems as time in favor of the adversary. By now, however, our attack is still considered to be practically infeasible due to its brute-force effort taking multiple years; thus, we aim to develop an improved and practically feasible version supporting our initial statements in order to promote a revision.

Finally, we see potential to utilize IPv6 addresses as Moving Target Defense (MTD). The address could change at a regular interval in a deterministic way, but appear random to observers. Hosts having a key are able to determine the current address, and connect to a server; the others are excluded. If an adversary sniffs a legitimate address, she will automatically loose connectivity with the next address. Such Moving Target Defense might be fruitful for Internet-of-Thing scenarios, and home servers serving only a limited group of users, e. g., the residents of a single household. With regard to the type of secret communication, this means a change in comparison to our approaches. We exploit IPv6 addresses as side channels; this implies that communication is unintended by the address owners. In the described scenario, all partners would actively communicate and patterns are hidden in the address on purpose, i. e., the address would be exploited as a covert channel.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] S. Zander, G. Armitage, and P. Branch, "A Survey of Covert Channels and Countermeasures in Computer Network Protocols," IEEE Communications Surveys & Tutorials, vol. 9, no. 3, pp. 44–57, 2007.

[2] D. C. Latham, Trusted Computer System Evaluation Criteria, 1986.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging {IT} Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," Future Generation Computer Systems, vol. 25, no. 6, pp. 599 – 616, 2009.

[4] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud Computing: State-of-the-Art and Research Challenges," Journal of Internet Services and Applications, vol. 1, no. 1, pp. 7–18, 2010.

[5] J. Geelan, "Twenty-One Experts Define Cloud Computing," http://cloudcomputing.sys-con.com/node/612375/, accessed: 2015-02-10.

[6] P. Mell and T. Grance, The NIST Definition of Cloud Computing, 2011.

[7] T. Mather, S. Kumaraswamy, and S. Latif, Cloud Security and Privacy: an Enterprise Perspective on Risks and Compliance. O'Reilly Media, 2009.

[8] L. Columbus, "Roundup Of Cloud Computing Forecasts And Market Estimates, 2015," http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/, accessed: 2015-08-07.

[9] J. D'Onfro, "Amazon's Cloud Revenue is bigger than its Four Closest Competitors combined," http://uk.businessinsider.com/aws-revenue-is-bigger-than-its-four-closest-competitors-combined-2015-4, accessed: 2015-07-02.

[10] L. Columbus, "Cloud Computing Adoption Continues Accelerating In The Enterprise," http://www.forbes.com/sites/louiscolumbus/2014/11/22/

cloud-computing-adoption-continues-accelerating-in-the-enterprise/, accessed: 2015-08-07.

[11] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud Computing – The Business Perspective," Decision Support Systems, vol. 51, no. 1, pp. 176 – 189, 2011.

[12] G. J. Popek and R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures," Communications of the ACM, vol. 17, no. 7, pp. 412–421, 1974.

[13] G. Malkin, "Internet Users' Glossary," RFC 1983, Internet Engineering Task Force, 1996.

[14] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50–55, 2008.

[15] S. Leimeister, M. Böhm, C. Riedl, and H. Krcmar, "The Business Perspective of Cloud Computing: Actors, Roles and Value Networks," in European Conference on Information Systems (ECIS), 2010.

[16] S. Cabuk, C. E. Brodley, and C. Shields, "IP Covert Channel Detection," ACM Transactions on Information and System Security (TISSEC), vol. 12, no. 4, pp. 22:1–22:29, 2009.

[17] I. Moskowitz and M. Kang, "Covert Channels-Here to Stay?" in Annual Conference on Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security, 1994.

[18] S. Cabuk, C. E. Brodley, and C. Shields, "IP Covert Timing Channels: Design and Detection," in ACM Conference on Computer and Communications Security (CCS), 2004.

[19] W. Mazurczyk, S. Wendzel, S. Zander, A. Houmansadr, and K. Szczypiorski, Information Hiding in Communication Networks: Fundamentals, Mechanisms, and Applications. Wiley-IEEE Press, 2016.

[20] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-Based Survey and Categorization of Network Covert Channel Techniques," ACM Computing Surveys, vol. 47, no. 3, pp. 50:1–50:26, 2015.

[21] A. Kerckhoffs, "La Cryptographie Militaire," Journal des Sciences Militaires, vol. 9, pp. 5–38, 1883.

[22] B. Schneier, Applied Cryptography: Protocols, Algorithms and Source Code in C. John Wiley & Sons, 1996.

[23] W. Diffie and M. Hellman, “New Directions in Cryptography,” IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644–654, 1976.

[24] R. M. Needham and M. D. Schroeder, “Using Encryption for Authentication in Large Networks of Computers,” Communications of the ACM, vol. 21, no. 12, pp. 993–999, 1978.

[25] T. Caddy, “Side-Channel Attacks,” in Encyclopedia of Cryptography and Security. Springer US, 2011.

[26] D. Genkin, A. Shamir, and E. Tromer, “RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis,” in CRYPTO 2014, 2014.

[27] J. Postel, “Internet Protocol,” RFC 791, Internet Engineering Task Force, 1981.

[28] “TCP/IP Fingerprinting Methods Supported by Nmap,” http://nmap.org/book/osdetect-methods.html, accessed: 2015-06-25.

[29] Y. Zhang, A. Juels, A. Oprea, and M. Reiter, “HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis,” in IEEE Symposium on Security and Privacy, 2011.

[30] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The Second-generation Onion Router,” in USENIX Security Symposium, 2004.

[31] J. South, “Punching a Hole in the Great Firewall,” http://www.chinafile.com/Punching-Hole-Great-Firewall, accessed: 2015-06-26.

[32] E. Dou and A. Barr, “U.S. Cloud Providers Face Backlash From China’s Censors,” http://www.wsj.com/articles/u-s-cloud-providers-face-backlash-from-chinas-censors-1426541126, accessed: 2015-06-26.

[33] J. Ullrich and E. Weippl, “Protection through isolation: Virtues and pitfalls,” in The Cloud Security Ecosystem - Technical, Legal, Business and Management Issues. Elsevier/Syngress, 2015.

[34] K. Benson, R. Dowsley, and H. Shacham, “Do You Know Where Your Cloud Files Are?” in ACM Workshop on Cloud Computing Security, 2011.

[35] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds,” in ACM Conference on Computer and Communications Security (CCS), 2009.

[36] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest, “How to Tell if Your Cloud Files Are Vulnerable to Drive Crashes,” in ACM Conference on Computer and Communications Security (CCS), 2011.

[37] A. Herzberg, H. Shulman, J. Ullrich, and E. Weippl, "Cloudoscopy: Services Discovery and Topology Mapping," in ACM Workshop on Cloud Computing Security, 2013.

[38] G. Pék, L. Buttyán, and B. Bencsáth, "A Survey of Security Issues in Hardware Virtualization," ACM Computing Surveys, vol. 45, no. 3, pp. 40:1–40:34, 2013.

[39] L. M. Kaufman, "Can Public-Cloud Security Meet Its Unique Challenges?" IEEE Security & Privacy, vol. 8, no. 4, pp. 55–57, 2010.

[40] A. Roy, S. Sarkar, R. Ganesan, and G. Goel, "Secure the Cloud: From the Perspective of a Service-Oriented Organization," ACM Computing Surveys, vol. 47, no. 3, pp. 41:1–41:30, 2015.

[41] A. Aviram, S. Hu, B. Ford, and R. Gummadi, "Determinating Timing Channels in Compute Clouds," in ACM Workshop on Cloud Computing Security, 2010.

[42] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011.

[43] D. Shue, M. J. Freedman, and A. Shaikh, "Performance Isolation and Fairness for Multi-Tenant Cloud Storage," in USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2012.

[44] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," in ACM Conference on Computer and Communications Security (CCS), 2011.

[45] Z. Xu, H. Wang, Z. Xu, and X. Wang, "Power Attack: An Increasing Threat to Data Centers," in Network and Distributed System Security Symposium (NDSS), 2014.

[46] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," SIGOPS Symposium on Operating Systems Principles, 2003.

[47] B. Adamczyk and A. Chydzinski, "On the Performance Isolation across Virtual Network Adapters in Xen," in International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING), 2011.

[48] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, "Resource-freeing Attacks: Improve Your Cloud Performance (at Your Neighbor's Expense)," in ACM Conference on Computer and Communications Security (CCS), 2012.

[49] B. Iyer, "Why Cloud Technology Is the Smart Move Right From Start Up," http://www.entrepreneur.com/article/241914, accessed: 2015-07-02.

[50] M. Rouse, “Cloud Marketplace,” http://searchcloudprovider.techtarget.com/definition/cloud-marketplace, accessed: 2015-07-02.

[51] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, “Demystifying Data Deduplication,” in ACM/IFIP/USENIX Middleware ’08, 2008.

[52] J. Paulo and J. Pereira, “A Survey and Classification of Storage Deduplication Systems,” ACM Computing Surveys, vol. 47, no. 1, pp. 11:1–11:30, 2014.

[53] D. Namiot and M. Sneps-Sneppe, “Local Messages for Smartphones,” in Conference on Future Internet Communications (CFIC), 2013.

[54] M. Huber, M. Mulazzani, S. Schrittwieser, and E. Weippl, “Appinspect: Large-scale Evaluation of Social Networking Apps,” in ACM Conference on Online Social Networks, 2013.

[55] Reuters, “Snowden says NSA engages in Industrial Espionage,” http://www.reuters.com/article/2014/01/26/us-security-snowden-germany-idUSBREA0P0DE20140126, accessed: 2015-02-11.

[56] P. Jubb, “Whistleblowing: A Restrictive Definition and Interpretation,” Journal of Business Ethics, vol. 21, no. 1, pp. 77–94, 1999.

[57] S. Aryan, H. Aryan, and J. A. Halderman, “Internet Censorship in Iran: A First Look,” in USENIX Workshop on Free and Open Communications on the Internet (FOCI), 2013.

[58] A. Chaabane, T. Chen, M. Cunche, E. De Cristofaro, A. Friedman, and M. A. Kaafar, “Censorship in the Wild: Analyzing Internet Filtering in Syria,” in Internet Measurement Conference (IMC), 2014.

[59] N. Christin, “Traveling the Silk Road: A Measurement Analysis of a Large Anonymous Online Marketplace,” in International Conference on World Wide Web (WWW), 2013.

[60] BBC, “Child Abuse Sites on Tor compromised by Malware,” http://www.bbc.com/news/technology-23573048, accessed: 2015-02-11.

[61] B.-J. Koops, “Crypto Law Survey,” http://www.cryptolaw.org/, accessed: 2015-02-11.

[62] BBC, “Hungary Internet Tax Cancelled After Mass Protests,” http://www.bbc.com/news/world-europe-29846285, accessed: 2015-02-11.

[63] A. Sunyaev and S. Schneider, “Cloud Services Certification,” Communications of the ACM, vol. 56, no. 2, pp. 33–36, 2013.

[64] Z. Xiao and Y. Xiao, "Security and Privacy in Cloud Computing," Communications Surveys & Tutorials, IEEE, vol. 15, no. 2, pp. 843–859, 2013.

[65] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles, "Botnets: A survey," Computer Networks, vol. 57, no. 2, pp. 378 – 403, 2013.

[66] X. Han, N. Kheir, and D. Balzarotti, "The Role of Cloud Services in Malicious Software: Trends and Insights," in Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2015.

[67] D. Brown, "Resilient Botnet Command and Control with Tor," http://conference.hitb.org/hitbsecconf2010kul/materials/D2T1%20-%20Dennis%20Brown%20-%20Botnet%20Command%20and%20Control%20with%20Tor.pdf, accessed: 2015-10-29.

[68] K. Okamura and Y. Oyama, "Load-based Covert Channels Between Xen Virtual Machines," in ACM Symposium on Applied Computing (SAC), 2010.

[69] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting, "An Exploration of L2 Cache Covert Channels in Virtualized Environments," in ACM Workshop on Cloud Computing Security, 2011.

[70] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler, "Detecting Co-residency with Active Traffic Analysis Techniques," in ACM Workshop on Cloud Computing Security, 2012.

[71] ——, "On Detecting Co-Resident Cloud Instances Using Network Flow Watermarking Techniques," International Journal of Information Security, vol. 13, no. 2, pp. 171–189, 2014.

[72] K. Block and G. Noubir, "Return of the Covert Channel, Data Center Style," in ACM Workshop on Cloud Computing Security, 2015.

[73] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side Channels in Cloud Services: Deduplication in Cloud Storage," IEEE Security & Privacy, vol. 8, no. 6, pp. 40–47, 2010.

[74] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, "Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space," in USENIX Security Symposium, 2011.

[75] M. Sneps-Sneppe and D. Namiot, "Spotique: A New Approach to Local Messaging," in Wired/Wireless Internet Communication, 2013, vol. 7889.

[76] J. Desimone, D. Johnson, B. Yuan, and P. Lutz, "Covert Channel in the BitTorrent Tracker Protocol," in International Conference on Security and Management (SAM), 2012.

[77] B. Hahn, R. Nithyanand, P. Gill, and R. Johnson, "Games Without Frontiers: Investigating Video Games as a Covert Channel," in IEEE European Symposium on Security and Privacy, 2016.

[78] J. S. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, "SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment," in Network and Distributed System Security Symposium (NDSS), 2015.

[79] S. Jin, J. Ahn, S. Cha, and J. Huh, "Architectural Support for Secure Virtualization Under a Vulnerable Hypervisor," in IEEE/ACM International Symposium on Microarchitecture, 2011.

[80] M. Smit, M. Shtern, B. Simmons, and M. Litoiu, "Partitioning Applications for Hybrid and Federated Clouds," in Conference of the Center for Advanced Studies on Collaborative Research, 2012.

[81] M. Shtern, B. Simmons, M. Smit, and M. Litoiu, "An Architecture for Overlaying Private Clouds on Public Providers," in International Conference on Network and Service Management, 2013.

[82] E. Musk, "All Our Patent Are Belong To You," https://www.teslamotors.com/blog/all-our-patent-are-belong-you, accessed: 2016-01-21.

[83] Knowledge@Wharton, "What's Driving Tesla's Open Source Gambit?" http://knowledge.wharton.upenn.edu/article/whats-driving-teslas-open-source-gambit/, accessed: 2016-01-21.

[84] R. Kalman, "On the General Theory of Control Systems," IRE Transactions on Automatic Control, vol. 4, no. 3, pp. 110–110, 1959.

[85] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, "Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2," in USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2012.

[86] Z. Ou, H. Zhuang, A. Lukyanenko, J. Nurminen, P. Hui, V. Mazalov, and A. Yla-Jaaski, "Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds," IEEE Transactions on Cloud Computing, vol. 1, no. 2, pp. 201–214, 2013.

[87] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for Your Money: Exploiting Performance Heterogeneity in Public Clouds," in ACM Symposium on Cloud Computing, 2012.

[88] Z. Xu, H. Wang, and Z. Wu, "A Measurement Study on Co-residence Threat inside the Cloud," in USENIX Security Symposium, 2015.

[89] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-Tenant Side-Channel Attacks in PaaS Clouds," in ACM Conference on Computer and Communications Security (CCS), 2014.

[90] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift, "A Placement Vulnerability Study in Multi-Tenant Public Clouds," in USENIX Security Symposium, 2015.

[91] S. Kadloor, X. Gong, N. Kiyavash, T. Tezcan, and N. Borisov, "Low-Cost Side Channel Remote Traffic Analysis Attack in Packet Networks," in IEEE International Conference on Communications (ICC), 2010.

[92] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam, "Mitigating Timing Based Information Leakage in Shared Schedulers," in IEEE International Conference on Computer Communications (INFOCOM), 2012.

[93] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider, "AmazonIA: When Elasticity Snaps Back," in ACM Conference on Computer and Communications Security (CCS), 2011.

[94] Y. Gu, Y. Fu, A. Prakash, Z. Lin, and H. Yin, "OS-Sommelier: Memory-only Operating System Fingerprinting in the Cloud," in ACM Symposium on Cloud Computing, 2012.

[95] ——, "Multi-Aspect, Robust, and Memory Exclusive Guest OS Fingerprinting," IEEE Transactions on Cloud Computing, 2014.

[96] D. E. Comer and J. C. Lin, "Probing TCP Implementations," in USENIX Summer Technical Conference, 1994.

[97] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," in Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 1997.

[98] R. Beverly, "A Robust Classifier for Passive TCP/IP Fingerprinting," in Passive and Active Network Measurement Conference (PAM), 2004.

[99] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow," in IEEE Symposium on Security and Privacy, 2010.

[100] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen, "Sidebuster: Automated Detection and Quantification of Side-channel Leaks in Web Application Development," in ACM Conference on Computer and Communications Security (CCS), 2010.

[101] D. Herrmann, R. Wendolsky, and H. Federrath, "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-Bayes Classifier," in ACM Workshop on Cloud Computing Security, 2009.

[102] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website Fingerprinting in Onion Routing Based Anonymization Networks," in ACM Workshop on Privacy in the Electronic Society, 2011.

[103] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z.-L. Zhang, "Towards Network-level Efficiency for Cloud Storage Services," in Internet Measurement Conference (IMC), 2014.

[104] European Union Agency for Network and Information Security, "Privacy and Data Protection by Design - from Policy to Engineering," 2014.

[105] Presidency of the Council of the European Union, "General Data Protection Regulation - Analysis of the final compromise text with a view to agreement," 2015.

[106] R. Mortier, A. Madhavapeddy, T. Hong, D. Murray, and M. Schwarzkopf, "Using Dust Clouds to Enhance Anonymous Communication," in International Workshop on Security Protocols, 2010.

[107] N. Jones, M. Arye, J. Cesareo, and M. J. Freedman, "Hiding Amongst the Clouds: A Proposal for Cloud-based Onion Routing," in USENIX Workshop on Free and Open Communications on the Internet (FOCI), 2011.

[108] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, "Examining How the Great Firewall Discovers Hidden Circumvention Servers," in Internet Measurement Conference (IMC), 2015.

[109] C. Brubaker, A. Houmansadr, and V. Shmatikov, "CloudTransport: Using Cloud Storage for Censorship-Resistant Networking," in Privacy Enhancing Technologies Symposium (PETS), 2014.

[110] E. Kartaltepe, J. Morales, S. Xu, and R. Sandhu, "Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures," in Applied Cryptography and Network Security (ACNS), 2010.

[111] N. Pantic and M. I. Husain, "Covert Botnet Command and Control Using Twitter," in Annual Computer Security Applications Conference (ACSAC), 2015.

[112] G. Kontaxis, I. Polakis, and S. Ioannidis, "Outsourcing Malicious Infrastructure to the Cloud," in SysSec Workshop, 2011.

[113] S. Zhao, P. P. C. Lee, J. C. S. Lui, X. Guan, X. Ma, and J. Tao, "Cloud-based Push-styled Mobile Botnets: A Case Study of Exploiting the Cloud to Device Messaging Service," in Annual Computer Security Applications Conference (ACSAC), 2012.

[114] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, "Blocking-Resistant Communication through Domain Fronting," Privacy Enhancing Technologies Symposium (PETS), vol. 2015, no. 2, pp. 46–64, 2015.

[115] M. Fire, R. Goldschmidt, and Y. Elovici, "Online Social Networks: Threats and Solutions," IEEE Communications Surveys & Tutorials, vol. 16, no. 4, pp. 2019–2036, 2014.

[116] L. von Ahn, M. Blum, and J. Langford, "Telling Humans and Computers Apart Automatically," Communications of the ACM, vol. 47, no. 2, pp. 56–60, 2004.

[117] S. Schrittwieser, P. Frühwirt, P. Kieseberg, M. Leithner, M. Mulanzzani, M. Huber, and E. Weippl, "Guess Who's Texting You? Evaluationg the Security of Smartphone Messaging Applications," in Network and Distributed System Security Symposium (NDSS), 2012.

[118] R. Mueller, S. Schrittwieser, P. Fruehwirt, P. Kieseberg, and E. Weippl, "What's New with WhatsApp & Co.? Revisiting the Security of Smartphone Messaging Applications," in International Conference on Information Integration and Web-based Applications & Services (iiWAS), 2014.

[119] M. Weyrich, J. P. Schmidt, and C. Ebert, "Machine-to-Machine Communication," IEEE Software, vol. 31, no. 4, pp. 19–23, 2014.

[120] J. Jaskolka and R. Khedri, "Exploring Covert Channels," in Hawaii International Conference on System Sciences (HICSS), 2011.

[121] J. Jaskolka, R. Khedri, and Q. Zhang, "On the necessary conditions for covert channel existence: A state-of-the-art survey," Procedia Computer Science, vol. 10, pp. 458 – 465, 2012.

[122] J. Bort, "Amazon is crushing IBM, Microsoft and Google ..." www.businessinsider.com/amazon-cloud-beats-ibm-microsoft-google-2013-11, Accessed: 2014-09-04.

[123] S. Bellovin and W. Cheswick, "Network Firewalls," Communications Magazine, IEEE, vol. 32, no. 9, pp. 50–57, Sept 1994.

[124] M. Gouda and A. Liu, "A Model of Stateful Firewalls and its Properties," in International Conference on Dependable Systems and Networks (DSN), June 2005.

[125] A. Atlasis, "Attacking IPv6 implementation using fragmentation," Black Hat Europe, 2012.

[126] N. Freed, "Behavior of and Requirements for Internet Firewalls," RFC 2979, Internet Engineering Task Force, 2000.

[127] redhat, "33.10.Limit network bandwidth for a Xen guest," https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtualization/sect-Virtualization-Tips_and_tricks-Limit_network_bandwidth_for_a_Xen_guest.html, accessed: 2016-04-26.

[128] L. Mei and X. Lv, "Optimization of network bandwidth allocation in Xen," in IEEE International Conference on High Performance Computing and Communications (HPCC), IEEE International Symposium on Cyberspace Safety and Security (CSS), IEEE International Conference on Embedded Software and Systems (ICESS), 2015.

[129] C. Li, S. Xi, C. Lu, C. D. Gill, and R. Guerin, "Prioritizing soft real-time network traffic in virtualized hosts based on Xen," in IEEE Real-Time and Embedded Technology and Applications Symposium, 2015.

[130] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in Network and Distributed System Security (NDSS), 2014.

[131] R. Sherwood, B. Bhattacharjee, and R. Braud, "Misbehaving TCP Receivers Can Cause Internet-wide Congestion Collapse," in ACM Conference on Computer and Communications Security (CCS), 2005.

[132] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges," IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 369–392, 2014.

[133] U. Lampe, M. Kieselmann, A. Miede, S. Zöller, and R. Steinmetz, "A Tale of Millis and Nanos: Time Measurements in Virtual and Physical Machines," in European Conference on Service-Oriented and Cloud Computing (ESOCC), 2013.

[134] T. Broomhead, L. Cremean, J. Ridoux, and D. Veitch, "Virtualize Everything but Time," in USENIX Symposium on Operating Systitems Design and Implementation (OSDI), 2010.

[135] M. Ullmann and M. Vögeler, "Delay attacks: Implication on NTP and PTP time synchronization," in International Symposium on Precision Clock Synchronization for Measurement, Control and Communic 2009.

[136] R. Rasti, M. Murthy, N. Weaver, and V. Paxson, "Temporal Lensing and Its Application in Pulsing Denial-of-Service Attacks," in IEEE Symposium on Security and Privacy, 2015.

[137] G. Wang and T. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in IEEE International Conference on Computer Communications (INFOCOM), 2010.

[138] Rackspace, "Pricing," https://www.rackspace.com/cloud/servers/pricing#num4, 2016, accessed: 2016-04-26.

[139] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291, Internet Engineering Task Force, 2006.

[140] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301, Internet Engineering Task Force, 2005.

[141] S. Kent, "IP Authentication Header," RFC 4302, Internet Engineering Task Force, 2005.

[142] ——, "IP Encapsulating Security Payload (ESP)," RFC 4303, Internet Engineering Task Force, 2005.

[143] E. Jankiewicz, J. Loughney, and T. Narten, "IPv6 Node Requirements," RFC 6434, Internet Engineering Task Force, 2011.

[144] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, Internet Engineering Task Force, 1998.

[145] A. Conta, S. Deering, and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443, Internet Engineering Task Force, 2006.

[146] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, Internet Engineering Task Force, 2007.

[147] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 3315, Internet Engineering Task Force, 2003.

[148] R. Droms, "Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6," RFC 3736, Internet Engineering Task Force, 2004.

[149] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC 2462, Internet Engineering Task Force, 1998.

[150] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," RFC 4862, Internet Engineering Task Force, 2007.

[151] IAB and IESG, "IAB/IESG Recommendations on IPv6 Address Allocations to Sites," RFC 3177, Internet Engineering Task Force, 2001.

[152] T. Narten, G. Huston, and L. Roberts, "IPv6 Address Assignment to End Sites," RFC 6177, Internet Engineering Task Force, 2011.

[153] C. Perkins, D. Johnson, and J. Arkko, "Mobility Support in IPv6," RFC 6275, Internet Engineering Task Force, 2011.

[154] E. Nordmark and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers," RFC 4213, Internet Engineering Task Force, 2005.

[155] J. Haas and S. Hares, "Definitions of Managed Objects for BGP-4," RFC 4273, Internet Engineering Task Force, 2006.

[156] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds," RFC 3056, Internet Engineering Task Force, 2001.

[157] C. Huitema, "An Anycast Prefix for 6to4 Relay Routers," RFC 3068, Internet Engineering Task Force, 2001.

[158] R. Despres, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)," RFC 5569, Internet Engineering Task Force, 2010.

[159] W. Townsley and O. Troan, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification," RFC 5969, Internet Engineering Task Force, 2010.

[160] B. Carpenter and C. Jung, "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels," RFC 2529, Internet Engineering Task Force, 1999.

[161] J. Wu, Y. Cui, X. Li, M. Xu, and C. Metz, "4over6 Transit Solution Using IP Encapsulation and MP-BGP Extensions," RFC 5747, Internet Engineering Task Force, 2010.

[162] F. Templin, T. Gleeson, and D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)," RFC 5214, Internet Engineering Task Force, 2008.

[163] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380, Internet Engineering Task Force, 2006.

[164] D. Thaler, "Teredo Extensions," RFC 6081, Internet Engineering Task Force, 2011.

[165] G. Tsirtsis and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)," RFC 2766, Internet Engineering Task Force, 2000.

[166] C. Aoun and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status," RFC 4966, Internet Engineering Task Force, 2007.

[167] F. Baker, X. Li, C. Bao, and K. Yin, "Framework for IPv4/IPv6 Translation," RFC 6144, Internet Engineering Task Force, 2011.

[168] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators," RFC 6052, Internet Engineering Task Force, 2010.

[169] X. Li, C. Bao, and F. Baker, "IP/ICMP Translation Algorithm," RFC 6145, Internet Engineering Task Force, 2011.

[170] M. Bagnulo, P. Matthews, and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers," RFC 6146, Internet Engineering Task Force, 2011.

[171] M. Bagnulo, A. Sullivan, P. Matthews, and I. van Beijnum, “DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers,” RFC 6147, Internet Engineering Task Force, 2011.

[172] J. Abley, P. Savola, and G. Neville-Neil, “Deprecation of Type 0 Routing Headers in IPv6,” RFC 5095, Internet Engineering Task Force, 2007.

[173] D. Johnson, C. Perkins, and J. Arkko, “Mobility Support in IPv6,” RFC 3775, Internet Engineering Task Force, 2004.

[174] C. Partridge and A. Jackson, “IPv6 Router Alert Option,” RFC 2711, Internet Engineering Task Force, 1999.

[175] S. Krishnan, J. Woodyatt, E. Kline, J. Hoagland, and M. Bhatia, “A Uniform Format for IPv6 Extension Headers,” RFC 6564, Internet Engineering Task Force, 2012.

[176] E. Davies, S. Krishnan, and P. Savola, “IPv6 Transition/Co-existence Security Considerations,” RFC 4942, Internet Engineering Task Force, 2007.

[177] G. Ziemba, D. Reed, and P. Traina, “Security Considerations for IP Fragment Filtering,” RFC 1858, Internet Engineering Task Force, 1995.

[178] I. Miller, “Protection Against a Variant of the Tiny Fragment Attack (RFC 1858),” RFC 3128, Internet Engineering Task Force, 2001.

[179] S. Krishnan, “Handling of Overlapping IPv6 Fragments,” RFC 5722, Internet Engineering Task Force, 2009.

[180] F. Gont, “Processing of IPv6 "Atomic" Fragments,” RFC 6946, Internet Engineering Task Force, 2013.

[181] S. Amante, B. Carpenter, S. Jiang, and J. Rajahalme, “IPv6 Flow Label Specification,” RFC 6437, Internet Engineering Task Force, 2011.

[182] N. Moore, “Optimistic Duplicate Address Detection (DAD) for IPv6,” RFC 4429, Internet Engineering Task Force, 2006.

[183] X. Yang, T. Ma, and Y. Shi, “Typical DoS/DDoS threats under IPv6,” in International Multi-Conference on Computing in the Global Information Technology (ICCGI), 2007.

[184] P. Nikander, J. Kempf, and E. Nordmark, “IPv6 Neighbor Discovery (ND) Trust Models and Threats,” RFC 3756, Internet Engineering Task Force, 2004.

[185] T. Chown and S. Venaas, “Rogue IPv6 Router Advertisement Problem Statement,” RFC 6104, Internet Engineering Task Force, 2011.

[186] E. Levy-Abegnoli, G. V. de Velde, C. Popoviciu, and J. Mohacsi, "IPv6 Router Advertisement Guard," RFC 6105, Internet Engineering Task Force, 2011.

[187] R. Draves and D. Thaler, "Default Router Preferences and More-Specific Routes," RFC 4191, Internet Engineering Task Force, 2005.

[188] F. Gont, "Security Assessment of the Internet Protocol Version 4," RFC 6274, Internet Engineering Task Force, 2011.

[189] F. Gont and W. Liue, "Security Implications of IPv6 Options of Type 10xxxxxx," Internet Draft, Internet Engineering Task Force, 2013.

[190] J. Arkko, J. Kempf, B. Zill, and P. Nikander, "SEcure Neighbor Discovery (SEND)," RFC 3971, Internet Engineering Task Force, 2005.

[191] T. Aura, "Cryptographically Generated Addresses (CGA)," RFC 3972, Internet Engineering Task Force, 2005.

[192] H. Rafiee, A. Alsa'deh, and C. Meinel, "WinSEND: Windows SEcure Neighbor Discovery," in International Conference on Security of Information and Networks (SIN), 2011.

[193] M. Heuse, "thc-ipv6 toolkit v2.7," https://www.thc.org/thc-ipv6/, accessed: 2015-04-12.

[194] G. Nakibly and F. Templin, "Routing Loop Attack Using IPv6 Automatic Tunnels: Problem Statement and Proposed Mitigations," RFC 6324, Internet Engineering Task Force, 2011.

[195] G. Nakibly and M. Arov, "Routing loop attacks using IPv6 tunnels," in USENIX Workshop on Offensive Technologies (WOOT), 2009.

[196] P. Savola and C. Patel, "Security Considerations for 6to4," RFC 3964, Internet Engineering Task Force, 2004.

[197] A. Conta and S. Deering, "Generic Packet Tunneling in IPv6 Specification," RFC 2473, Internet Engineering Task Force, 1998.

[198] I. Gashinsky, J. Jaeggli, and W. Kumari, "Operational Neighbor Discovery Problems," RFC 6583, Internet Engineering Task Force, 2012.

[199] M. Kohno, B. Nitzan, R. Bush, Y. Matsuzaki, L. Colitti, and T. Narten, "Using 127-Bit IPv6 Prefixes on Inter-Router Links," RFC 6164, Internet Engineering Task Force, 2011.

[200] J. Arkko, V. Devarapalli, and F. Dupont, "Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents," RFC 3776, Internet Engineering Task Force, 2004.

[201] F. Gont, D. Thaler, and W. Liue, "Deprecating EUI-64 Based IPv6 Addresses," Internet Draft, Internet Engineering Task Force, 2013.

[202] T. Narten, R. Draves, and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 4941, Internet Engineering Task Force, 2007.

[203] S. Groat, M. Dunlop, R. Marchany, and J. Tront, "What DHCPv6 Says About You," in World Congress on Internet Security (WorldCIS), 2011.

[204] D. Malone, "Observations of IPv6 addresses," in Passive and Active Network Measurement Conference (PAM), 2008.

[205] P. van Dijk, "Finding v6 Hosts by Efficiently Mapping ip6.arpa," http://7bits.nl/blog/posts/finding-v6-hosts-by-efficiently-mapping-ip6-arpa, accessed: 2013-11-14.

[206] M. Crawford and B. Haberman, "IPv6 Node Information Queries," RFC 4620, Internet Engineering Task Force, 2006.

[207] A. Conta, "Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification," RFC 3122, Internet Engineering Task Force, 2001.

[208] T. Chown, "IPv6 Implications for Network Scanning," RFC 5157, Internet Engineering Task Force, 2008.

[209] E. Davies and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls," RFC 4890, Internet Engineering Task Force, 2007.

[210] N. B. Lucena, G. Lewandowski, and S. J. Chapin, "Covert Channels in IPv6," in Privacy Enhancing Technologies Symposium (PETS), 2006.

[211] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering, "IPv6 Flow Label Specification," RFC 3697, Internet Engineering Task Force, 2004.

[212] S. Zander, G. Armitage, and P. Branch, "Covert Channels in the IP Time To Live field," in Australian Telecommunication Networks and Applications Conference (ATNAC), 2006.

[213] J. Lindqvist, "IPv6 Stateless Address Autoconfiguration considered harmful," in IEEE Military Communications Conference (MILCOM), 2006.

[214] J. D. Howard and T. A. Longstaff, "A common language for computer security incidents," Sandia Report: SAND98-8667, Sandia National Laboratories, 1998.

[215] C. P. Pfleeger and S. L. Pfleeger, Security in computing. Prentice Hall Professional, 2003.

[216] R. Graham, "masscan," https://github.com/robertdavidgraham/masscan, Accessed: 2015-03-04.

[217] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-Wide Scanning and its Security Applications," in USENIX Security Symposium, 2013.

[218] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices," in USENIX Security Symposium, 2012.

[219] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS Certificate Ecosystem," in Internet Measurement Conference (IMC), 2013.

[220] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks," in USENIX Security Symposium, 2014.

[221] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer et al., "The matter of Heartbleed," in Internet Measurement Conference (IMC), 2014.

[222] F. Gont and T. Chown, "Network Reconnaissance in IPv6 Networks," Internet Draft, Internet Engineering Task Force, 2015.

[223] G. Lyon, "nmap," http://nmap.org, Accessed: 2015-03-04.

[224] D. Leonard and D. Loguinov, "Demystifying Service Discovery: Implementing an Internet-Wide Scanner," in Internet Measurement Conference (IMC), 2010.

[225] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-Wide View of Internet-Wide Scanning," in USENIX Security Symposium, 2014.

[226] F. Gont, "IPv6 Toolkit," https://github.com/fgont/ipv6toolkit, Accessed: 2015-03-04.

[227] I. H. Witten, E. Frank, and M. A. Hall, Data Mining, Pratical Machine Learning Tools and Techniques. Morgan Kaufmann, 2011.

[228] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," ACM SIGKDD Explorations Newsletter, vol. 11, no. 1, pp. 10–18, 2009.

[229] S. Turner, "CMS Symmetric Key Management and Distribution," RFC 5275, Internet Engineering Task Force, 2008.

[230] F. Gont, "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)," RFC 7217, Internet Engineering Task Force, 2014.

[231] A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith, "Privacy Considerations for Internet Protocols," RFC 6973, Internet Engineering Task Force, 2013.

[232] M. Dunlop, S. Groat, R. Marchany, and J. Tront, "IPv6: Now You See Me, Now You Don't," in International Conference on Networks (ICN), 2011.

[233] S. Groat, M. Dunlop, R. Marchany, and J. Tront, "IPv6: Nowhere to Run, Nowhere to Hide," in Hawaii International Conference on System Sciences (HICSS), 2011.

[234] A. Alsadeh, H. Rafiee, and C. Meinel, "Cryptographically Generated Addresses (CGAs): Possible Attacks and Proposed Mitigation Approaches," in IEEE International Conference on Computer and Information Technology (CIT), 2012.

[235] A. AlSa'deh, H. Rafiee, and C. Meinel, "IPv6 Stateless Address Autoconfiguration: Balancing between Security, Privacy and Usability," in Foundations and Practice of Security, 2013.

[236] D. Barrera, G. Wurster, and P. C. Van Oorschot, "Back to the Future: Revisiting IPv6 Privacy Extensions," LOGIN: The USENIX Magazine, vol. 36, no. 1, pp. 16–26, 2011.

[237] S. Turner and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms," RFC 6151, Internet Engineering Task Force, 2011.

[238] J. M. Gosney, "Password Cracking HPC," in Passwords Security Conference, 2012.

[239] TechNet, "IPv6 Addressing (Tech Ref)," https://technet.microsoft.com/en-us/library/dd392266%28v=ws.10%29.aspx, accessed: 2015-04-12.

[240] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger, "Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate," in CRYPTO 2009, 2009.

[241] "eBASH (ECRYPT Benchmarking of All Submitted Hashes," http://bench.cr.yp.to/results-hash.html, accessed: 2015-04-12.