FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Public Wi-Fi Hotspots in the Wild

Revealing vulnerabilities and restrictions for ordinary hotspot users

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Stefanie Plieschnegger, BSc
Matrikelnummer 0926102

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Mitwirkung: Univ.Lektor Dipl.-Ing. Dr.techn. Adrian Dabrowski, BSc

Wien, 10. Oktober 2018

Stefanie Plieschnegger	Edgar Weippl

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Public Wi-Fi Hotspots in the Wild

Revealing vulnerabilities and restrictions for ordinary hotspot users

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering & Internet Computing

by

Stefanie Plieschnegger, BSc
Registration Number 0926102

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Assistance: Univ.Lektor Dipl.-Ing. Dr.techn. Adrian Dabrowski, BSc

Vienna, 10th October, 2018

Stefanie Plieschnegger	Edgar Weippl

Erklärung zur Verfassung der Arbeit

Stefanie Plieschnegger, BSc
Artholdgasse 2/3/42, 1100 Vienna

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, 10. Oktober 2018

Stefanie Plieschnegger

Kurzfassung

Diese Arbeit evaluiert welchen Gefahren Nutzern von offenen Wi-Fi Hotspots in der Realität ausgesetzt sind. Es ist allgemein bekannt, dass unverschlüsselte Netzwerke abgehört und gesendete Daten in weiterer Folge missbraucht werden können. Im Zuge einer Feldstudie untersuchten wir offene Wi-Fi Hotspots aus dem echten Leben. Dabei wurden Zugangsdaten im Netzwerk gestreut, die als Köder dienen sollten.

Mithilfe eine Android App wurde die Studie umgesetzt. Diese konnte sich selbstständig zu offenen Wi-Fi Hotspots verbinden, gegebenenfalls mit Captive Portals interagieren, und Daten über das Netzwerk senden. Dabei wurden User Sessions über HTTP, FTP, SMTP und IMAP simuliert. Diese Protokolle sind vor Natur aus unverschlüsselt. Wir achteten darauf, dass für jeden Login andere und eindeutig indentifizierbare Zugangsdaten verwendet wurden. Außerdem waren wir unter Kontrolle des Servers, der diese Services verwaltete. Das ermöglichte uns Angreifer zu entdecken, da die Logindaten ein weiteres Mal verwendet worden wären.
Darüberhinaus, evaluierten wir generelle Einschränkungen, die in Verbindung mit dem jeweiligen Hotspot standen, wie z.B. Content Filter.
Während dem Zeitraum von über vier Monaten, testeten wir zahlreiche offene Wi-Fi Hotspots in sieben Ländern und initiierten dabei tausende Logindaten. Die Server mit den laufenden Services blieb noch einige Monate aktiv, aber wir konnten keine wiederverwendeten Login Daten identifizieren. Jedoch fanden wir heraus, dass einige Hotspots Daten über SMTP abfangen. Außerdem konnten wir feststellen, dass jeder vierte Captive Portal geschützte Hotspot, HTTPS Verbindungen manipulierte, solange der Nutzer sich noch nicht authentiziert hatte. Das führte folglich zu Zertifikatsfehlermeldungen. Außerdem konnten wir unterschiedlichen Content Filter Strategien aufdecken. Die am öftesten blockierten Inhalte betrafen Streamingseiten mit potenziellen Raubkopien, sowie pornographische Webseiten.

Wir konnten keine Netzwerkangreifer identifizieren, die unsere gestreuten Logindaten wiederverwendeten. Allerdings zeigt diese Feldstudie nur einen kleinen Ausschnitt der realen Welt. Nichtsdestotrotz konnten wir einige bedenkliche Abfangmechanismen feststellen. Darüberhinaus erhielten wir Einblick über aktuelle Wi-Fi Netzwerkkonfigurationen, vor allem in Hinsicht auf Content Filter Strategien.

Abstract

This thesis evaluates vulnerabilities that hotspot users face when using real-world public Wi-Fi hotspots. It is well-known that network traffic in unencrypted networks can be sniffed and intercepted. In a field study we assessed such Wi-Fi networks in the wild.

We designed an Android app that automatically connects to public Wi-Fi hotspots, interacts with captive portals, and sends bait credentials over the network. Those credentials are part of simulated user sessions for unencrypted protocols over HTTP, FTP, SMTP, and IMAP. We controlled the server offering those services and ensured that for every simulated session, unique credentials were used. Thus, in case someone sniffed the data, and tried to use this login credentials, we would have been able to identify this attack. Moreover, we evaluated some general configurations of the hotspots like content filtering.
Within over four month, we tested various public Wi-Fi hotspots in seven countries, sending thousands of bait credentials. The services on the bait server kept running for a few more months, but were not able to reveal any network sniffing attacks.
However, we revealed that a minority of public Wi-Fi hotspots intercepted SMTP traffic. We also found that every fourth captive-portal-protected Wi-Fi network tampered with HTTPS connections before a successful authentication and thereby triggered certificate errors. Moreover, we experienced various content filtering techniques and found that streaming services with potentially pirated content, and pornographic websites were blocked most frequently.

During our field study we could not detect any network sniffing attack, however, we only provided a snapshot of public Wi-Fi hotspots in the wild. We identified some alarming interception techniques. Moreover, we gained some insights about current network configuration decisions, especially regarding trends of blocking certain website content.

Contents

CHAPTER 1

Introduction

Public Wi-Fi hotspots have become an integral part of most cities, providing people free access to the internet. Nowadays, it is kind of an expected service in restaurants, bars, airports, hotels etc. The provided hotspots are usually open, meaning there is no encryption in place between client and access point. A recent study [117] (2017) reveals that access to public Wi-Fi is a strong deciding factor for people when choosing an accommodation (71%), or even when eating out (43%). Nearly half of the respondents name being able to navigate as one of the most important reasons to Wi-Fi access. Similar figures can be witnessed in a Kaspersky study [62], focusing on people traveling abroad either for leisure or work, which shows that three in four traveler connect to public Wi-Fi.
This high numbers of Wi-Fi users may not be surprising considering the digital age we are living in and the shift towards carrying a smartphone everywhere one goes. Despite the wide distribution and availability of mobile data, the high costs and limited data allowance explain the popularity of public Wi-Fi hotspots, especially when being abroad.

However, many users are not aware of the downside when accessing public Wi-Fi hotspots: unencrypted traffic can easily be eavesdropped and misused. Thus users are responsible to ensure their traffic is secure and cannot be sniffed. Users failing to do so will face various security and privacy risks, understandably depending on their individual habits of using the Wi-Fi network. Moreover, one may not consider traffic that is not initiated manually by the user but rather sent in the background automatically, potentially using unsecured protocols. This could be an email program synchronizing, or an app pushing or pulling data from a server and sending confidential data or login credentials in plain text.

While it is not a secret that traffic sniffing is possible in theory, people seem to feel it is unlikely to happen in real life. One good example may be the *Wall of Sheep* [91],

a project started at the security conference Defcon[1], with the goal of revealing any sensitive data that is sent in plain text through the public network. Although the hosted audience could be considered as being tech-savvy and security aware a lot of people ended up on that wall, exposing usernames and passwords.

Considering that exploitation of sniffed data may happen at a later point and could even be unnoticed, public Wi-Fi users may have a wrong feeling of security. The Norton Wi-Fi risk report [117] outlines that 60% of respondents claimed to feel at least "somewhat safe" when using public Wi-Fi. 25% stated that they use a Virtual Private Network (VPN) connection. However, the study does not provide any data regarding the use of HTTPS. Another study [73] conducted in Australia, revealed that about 27% of active Wi-Fi users perceive public Wi-Fi as secure and 12% claimed being unsure about the security. Nevertheless there still seems to be an educational need regarding security.

Wi-Fi users face various threats when connecting and using public hotspots. This thesis aims to evaluate vulnerabilities that could endanger genuine users in the real world when connecting to public Wi-Fi hotspots. We address active and passive attack scenarios. In a field study we use an Android app that connects to public Wi-Fi hotspots nearby and analyzes the networks. In order to identify network sniffers, honey traffic is created and sent over the network. Therefore, user sessions are simulated over FTP, IMAP, SMTP and HTTP. These commonly used protocols are unencrypted by default and thus an attacker could intercept the login credentials and reuse those by accessing the honey services on our server.
Besides the honey traffic distribution we also verify expected banners, a greeting text returned upon successful connection to FTP, IMAP and SMTP services. Other tested active attack scenarios include redirection, and modified website content.
In addition, we assess general restrictions dictated by the network. This includes denied access to certain website categories, and port restrictions. Moreover, we check the Wi-Fi hotspots for captive portals and test common authentication circumvention techniques.

Due to the requirement of testing real public Wi-Fi networks, which require physical presence, the field study is mainly focusing on hotspots around Vienna.
Naturally, the outcome of this study will only give a limited snapshot of the real world.

[1] https://www.defcon.org

CHAPTER 2

Motivation

Recent numbers suggest that web pages loaded over HTTPS are rising and make up over 60% [52], [40]. However, those figures only count websites and therefore just represents a subset of protocols used in daily online communication. Email clients, for example, usually synchronize automatically in the background and for most users it will not be that obvious if the used protocol uses encryption or not.
The main concerns of public Wi-Fi hotspots may be the trustworthiness of providers, and unencrypted traffic. A Wi-Fi hotspot provider controls the traffic that is going in and out of the network, and this consequently allows the provider to monitor, analyze, block, interfere, and eavesdrop the content that is going through the access point. Assuming you fully trust the hotspot provided by your favorite coffee shop in terms that your data will not be processed, analyzed, or misused. Even in that case, there are two scenarios to keep in mind: first, anyone who connects to this public Wi-Fi hotspot could sniff all packets that are sent and routed through the network, without leaving any traces. Second, you may be connected with a rogue hotspot: a Wi-Fi network with the sole purpose of intercepting, or eavesdropping data. In order to gain trust, they could assign the network a name you already trust (also known as *evil twin attack* [84]), or use a very common one like "Free guest wifi". Such Wi-Fi hotspots can be setup easily, with low costs but potentially high impact.

Another consideration point regarding public Wi-Fi hotspots are the terms and regulations that are forced onto users, by the provider. First of all, most public networks nowadays are using a captive portal, e.g., a landing page the user will be redirected to. Depending on the setup and requirements the user have to acknowledge conditions for use, or even provide detailed personal information [64].
Moreover, the provider may restrict what kind of services or websites the user can access. Some providers may decide to block SMTP traffic entirely to prevent any potential misuse by spammers. Others may block content that could be considered as

inappropriate or even illegal, for example file sharing websites, pornography, or certain blogs. In order to limit the consumed data volume even popular legal streaming services could be cut off.

To the best of our knowledge there have been no prior similar studies on public Wi-Fi networks in the wild that included testing vulnerabilities and restrictions for genuine users.

CHAPTER 3

Background

3.1 Wi-Fi Hotspots and Captive Portals

Public (or *open*) Wi-Fi hotspots are wireless access points which are not using any encryption and are accessible for everyone.
While pure connection to the hotspot does not require a password, most of these hotspots, however, present the user a **captive portal**. This is a common approach where the user is facing a website (the captive portal landing page) that the browser will be redirected to. The user will then have to actively interact with the captive portal, before internet access is granted [64]. We refer to this procedure as *authentication* or *login*.
Usually any outgoing network traffic will be blocked before successful authentication.
Figure 3.1 shows a simplified example of how this may work:

Step 1: The user has already connected to the public Wi-Fi hotspot, and tries to access the website `example.com` over a web browser.

Step 2: The hotspot answers with a redirect to the captive portal landing page. In this example, the redirect is going to `captive-portal-login.html`. As a result the user's browser will automatically follow to the redirected website.

Step 3: Now the user has to interact with the captive portal, e.g., by accepting terms, clicking a button, providing personal information or the like. Normally the user will then be notified if the authentication was successful.

Step 4: The user is authenticated and the Wi-Fi hotspot will allow outgoing connection and accessing the internet. Consequently the user is now able to request `example.com`.

Normally a user would have to confirm some terms and conditions for legal reasons. Some captive portals even require login via social media accounts, or ask for detailed

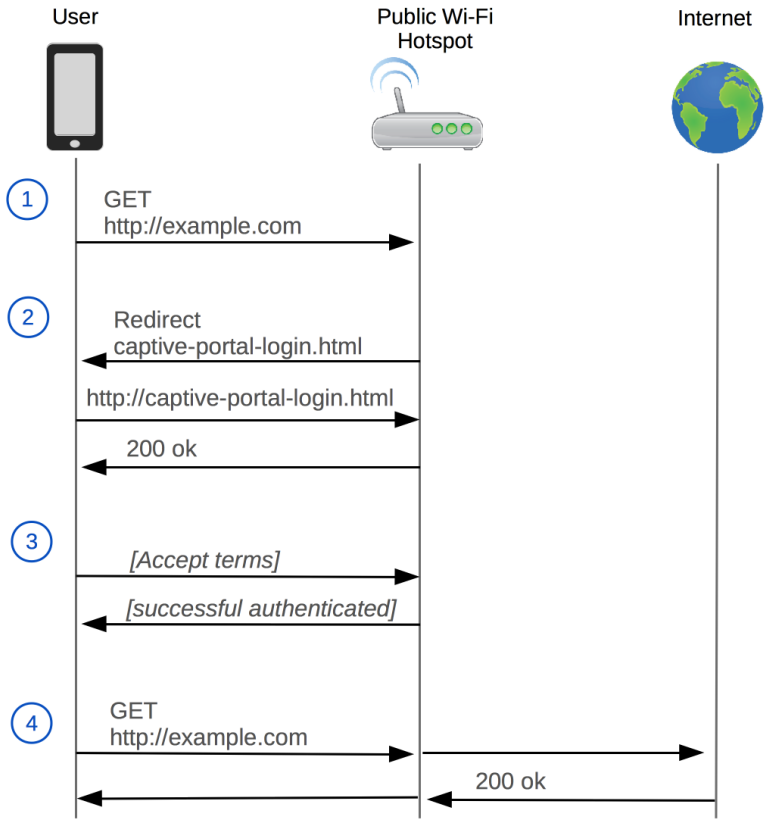


Figure 3.1: Captive portal authentication

information via a form [48]. Other captive portals may restrict the use to authorized users only, and ask for additional login credentials, or a token in order to log in. Providers could also request payment in turn for using the Wi-Fi.

Depending on the actual hotspot, the provided internet services may be restricted, even after successful authentication. Further restrictions could include a limited data or time usage, and blocked or filtered services or websites, etc.

Considering the process of interaction with the captive portal, we are distinguishing three stages:

- **Pre-Authentication** The user is connected to the Wi-Fi access point.
- **Captive Portal Login** It is tested whether a captive portal can be detected and if so, the authentication procedure is conducted. Depending on the captive portal this may involve various steps.
- **Post-Authentication** The authentication (if any) was successful and internet access has been granted. Note that if the authentication was unsuccessful, this

stage will not be reached.

3.1.1 Captive Portal Redirection Methods

In order to present the user with this captive portal screen when connecting to the Wi-Fi, the traffic of the user is intercepted and altered to force a redirection to the captive portal. This usually happens within the web browser. When redirecting HTTP requests, the captive portal could sends a HTTP response code 3xx indicating a redirect, that the browser will follow [48], [16]. However, the HTTP response may also be *511 Network Authentication Required*, which is a new proposed response code, that is recommended in order to identify a captive portal [87]. Other techniques for redirection include Domain Name System (DNS) [102] and Internet Control Message Protocol (ICMP) [96]. Devices that do not support web browsers are potentially cut off by these approaches by design [48].

This behavior of altering and modifying responses classifies as a Man-in-the-Middle (MitM) attack and therefore raises concerns. Especially handling of HTTPS requests (before the user is authenticated) are an important issue. Some captive portals may try to redirect those ones as well, resulting in certificate errors [104] and forcing users to ignore such warnings. This is not only bad practice, but also trains the user to dismiss any security warnings in order to make it work [10]. Suggested ways of handling HTTPS requests would simply be allowing or blocking all HTTPS traffic instead of redirecting [104].

3.1.2 Captive Portal Detection

To prevent those kind of errors, many clients have a built-in captive portal detection, which notifies the whenever a network requires a sign-in. The check is conducted by the Operating System (OS) directly or by a web browser. Usually the captive portal is detected by testing a reserved website (note that the request will be over HTTP) and comparing the expected response with the actual one. On Android for example, the OS will test automatically if unrestricted Wi-Fi is available once a connection to a Wi-Fi network has been established. For this check, Android calls a website which is expected to return an empty body and response code 204 [2]. If the result is different than expected, the user will be notified and prompted with the sign-in screen of the captive portal.
Microsoft Windows has a documented captive portal detection, which will automatically start a browser presenting the user the redirected website [76]. Apple provides an automatic detection on OSX and iOS as well [1]. In addition, some browsers have similar functionality like Firefox, which is also testing for a specific website, expecting *success* as response [3].

3.2 Honey Traffic

Honey traffic relates to the commonly known term *honeypot*. A honeypot is a mechanism deployed in a network (either physical or virtual) to attract attackers while being closely monitored and analyzed in order to identify any unusual behavior in the network [101]. In this thesis, honey traffic is used to describe unencrypted traffic that is spread over a public Wi-Fi network. This decoy content contains login credentials for different services about which we assume the attacker is potentially interested in gaining access to. As neither the network nor the traffic is encrypted, a person with malicious intent could simply eavesdrop the network traffic and extract the relevant login credentials.

3.3 Protocols

In the following some widespread protocols are explained. Although, secure extensions are available, it is not unusual to find those unencrypted protocols in practice. It is also possible that secure connection attempts are intercepted and stripped off. This attack that will be explained in more detail in section 3.4.

3.3.1 HTTP

Hypertext Text Transfer Protocol (HTTP) is the standard protocol for browsing the web, and is normally running over port 80 [44]. Sending sensitive data over HTTP (including forms using GET and POST requests) transmits the data in plain text.

Hypertext Transfer Protocol Secure (HTTPS) is the encrypted version with the default port 443 [44]. According recent statistics by Let's Encrypt [40] and Google [52], the encrypted web pages sum up to 60-80% of current traffic, which still leaves 20-40% unencrypted.
Encrypted web sites should prevent eavesdropping and manipulation of the traffic. However, also encrypted web sites may be vulnerable to attacks, depending on the actual encryption standard used [106]. The current standard and proposed protocol is Transfer Layer Security (TLS) 1.2, but older versions are still in use for backwards compatibility reasons with clients. Its predecessor Secure Socket Layer (SSL) has been deprecated in 2011 (version 2.0) [123] respectively in 2015 (version 3.0) [5] because of various security reasons. Nevertheless, even the unsecure SSL protocol is still in use, as the monthly scan of SSL- and TLS-enabled websites on `ssllabs.com` is revealing [63]. As in January 2018, there are still 3.5% using SSL 2.0 respectively 13% SSL 3.0.

3.3.2 FTP

File Transfer Protocol (FTP) is used for transferring files over the internet [99], [103]. The connection is established over Transmission Control Protocol (TCP) on standard port 21, which is reserved for the *control* port and as the name suggests, is used for receiving commands. There are two different modes that enable the actual file transfer:

active and passive. Normally port 20 is occupied for the active data transmission (the *data* port), where the client will issue a command telling the server which port it should connect to for transferring data. If the client is behind a Network Address Resolution (NAT) or a firewall, however, the active mode will not work. In such a case, the client will have a different internal IP address respectively the incoming connection may be blocked. Therefore a *passive mode* can be used to transfer data (passive data port). In the passive mode the server tells the client which port the client needs to connect to for starting the data transfer. Either way, the entire communication (control and data port) is in clear text.

A proposed extension is defined in RFC 4217 [45], commonly known as File Transfer Protocol over TLS (FTPS), using the TLS protocol for a secure communication. The client can use `AUTH TLS` to initiate a secure connection. Depending on the server's policy, only TLS sessions may be accepted.

3.3.3 IMAP

The Internet Message Access Protocol (IMAP) allows to retrieve and manipulate emails from a server to a client application [21]. It provides a variety of functionalities including creating, renaming and deleting of mailboxes, checking new messages, or searching for content. Mails that have been fetched by a client are preserved on the server. The default port for IMAP is 143.
In contrast, Post Office Protocol, Version 3 (POP3), another protocol for retrieving emails, has only limited functionality (list, fetch, delete emails) [83]. Also, POP3 normally removes mails from the server once they have been fetched by a client.

For IMAP two secure adaptions exist. One is the `STARTTLS` command, that can be issued by the client after connecting to the server in order to request a TLS secured connection [85], [74]. The second one is IMAP over TLS (IMAPS), which is already connecting via a secure channel (on standard port 993), but its use is discouraged as it may gives a wrong feeling of security [85]. For example the encryption used could be weak or it may lead to the assumption that the connection over the default port 143 is always unencrypted, which may not be the case.

Most email service provider are using the same credentials for IMAP and SMTP meaning that not only stored emails or email addresses may be of interest, but the attacker could try to abuse those login data for sending emails as well.

3.3.4 SMTP

Simple Mail Transfer Protocol (SMTP) provides capabilities to send emails. The client connects to the server usually on port 25 and sends commands [61]. The client has to authenticate before emails can be sent.
A newer standard, [49], proposes using port 25 only for relaying messages between Mail Transferring Agents (MTA), and relying on port 587 for email message submission. This

standard was introduced to split message relay and transfer from message submission, making it easier to divide specific security aspects and policies [49].

There is an extension for SMTP which enables encryption of the traffic, by sending the command `STARTTLS` after a connection has already been established [55], [74]. This command initiates a TLS session. However, this method is not encrypting the traffic from sender to recipient, but only from sender to the SMTP server.

3.4 SSL Stripping

SSL stripping is a MitM attack that prevents the upgrade to a HTTPS connection [69], [68], [106]. We assume that an attacker is able to intercept the traffic between the victim's client and the server. This is not an usual assumption and in fact Marlinspike explains how this could be done in a public Wi-Fi network by using the tool *arpspoof*[1] [67].
Most websites automatically redirect the user to HTTPS, e.g., by sending a response `301 Moved Permanently` if an HTTP connection was initiated. As a result, most users are requesting websites without specifying the protocol by explicitly typing the `https://` part, instead they are relying on the redirection or are not aware that the site should be encrypted in the first place.

In such a scenario, when the user is relying on the automatic redirection, the attacker can easily cut off the SSL part, e.g., by not telling the user's client (e.g., the web browser) about the redirection to HTTPS. While the attacker keeps communicating with the server over HTTPS, the user will be facing a plain text connection over HTTP. A simplified

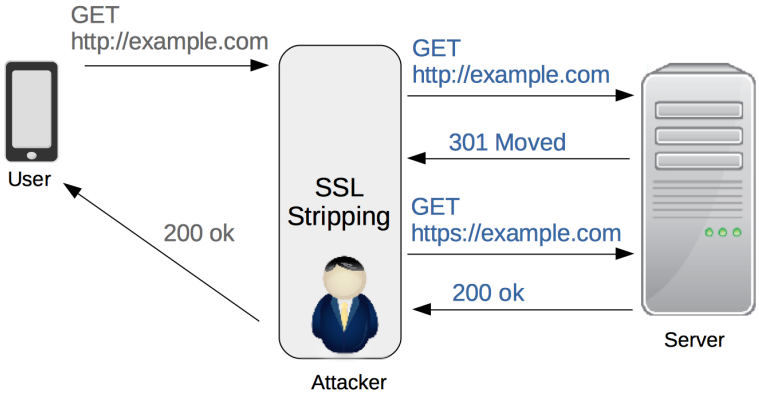


Figure 3.2: Example SSL stripping attack

attack is pictured in figure 3.2.

[1] `https://linux.die.net/man/8/arpspoof`

Step 1: The user is requesting the website `example.com`, which will automatically be sent over HTTP. As the attacker is in the middle of the connection, he or she will just forward it to the server.

Step 2: The server will answer with a redirect to `https://example.com`, which the attacker will follow to.

Step 3: Now the attacker has an encrypted communication channel with the server, and for the server everything looks fine.

Step 4: Note that only now the user will receive a `200 OK` by the attacker and therefore never know about the intercepted HTTPS upgrade.

Step 5: For any further request by the user, the attacker will keep modifying responses from the server, e.g., stripping HTTPS (including embedded links of the requested website etc.), so that the user is forced to continue the communication in clear text.

Marlinspike has presented the lightweight tool *sslstrip* [67] at BlackHat DC in 2009 [69]. The tool is written in Python and can be setup easily to run SSL stripping attacks on any traffic within the network [67], therefore providing an out-of-the-box functionality, which makes this kind of attack even more likely.

Similarly to this SSL stripping attack, the `STARTTLS` command could be attacked, e.g., by intercepting and preventing the upgrade to a secure connection [106]. A proof of concept implementation, *striptls* [121], capable of attacking various protocols including SMTP, POP3, IMAP, FTP and more has been proposed as well.
An empirical analysis about email delivery security [37], conducted in 2015, revealed that over 20% of mails sent over Gmail were affected by MitM attacks. The country with the most identified attacks was Tunisia, with over 96%, followed by Iraq with over 25%. This type of attack may be prevented by using strict policies on server and client side [55].

3.5 Port Testing

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are two protocols used for transmitting packets over a network. Both have distinct attributes, described in the following. As the packet transmission works in a different way, it is possible to bind listening sockets for both protocols to the same port.

3.5.1 TCP

TCP is known as a highly reliable internet protocol and it guarantees ordered, and error-checked transmission of packets [97]. To establish a connection between two end points, a handshake is used. During the connection certain status information is maintained that

is required to check the right order of received packets. Typical examples for protocols that are using TCP include HTTP [44], SMTP [61], FTP [99] and IMAP [21].

3.5.2 UDP

UDP is relying on the internet protocol as well, but in contrast to TCP the delivery and protection of duplicate packets is not guaranteed [94]. Instead, this protocol is transaction-oriented. The connection-less communication does not require a handshake and avoids overheads. However, this also means that packets may be lost without indication. Due to its lightweight characteristic, UDP is commonly used for the DNS protocol [78].

3.6 Tunneling Techniques

Using a tunnel is a preferred technique in order to circumvent a captive portal, e.g., when requiring passwords, tokens, or even payment. Depending on the setup of the captive portal, outgoing traffic may only be partially blocked. Usually HTTP requests are intercepted as already stated in 3.1.2. However, some captive portals may allow pinging external Internet Protocol (IP) addresses before authentication, indicating that ICMP is not blocked [50].
Furthermore, it may be possible to translate host names to IP addresses, revealing that DNS requests are allowed [64]. DNS cannot be blocked entirely, as somehow the redirection to the captive portal needs to be triggered by calling some arbitrary host name, that the DNS will have to respond to. The network could however spoof responses for DNS requests or run its own DNS server, resolving all host names to the captive portal's IP address before the user is authenticated [102].
Thus, testing if external ICMP and DNS requests are possible is rather easy. If those protocols can be accessed despite the captive portal, it may be possible to create a tunnel, where all traffic can be routed through, without the need for authentication and therefore completely bypassing the captive portal [50], [42].
However, besides captive portal or firewall circumvention, tunneling is also used by malware for remote Command and Control (C&C) channels, and is well known as a way for data exfiltration in general as it provides a stealthy way of communication [108], [92], [79], [35], [107].

3.6.1 ICMP Tunnel

ICMP is generally used for diagnostic reasons, or to indicate errors in IP operations (for example if a destination is unreachable) [96]. ICMP may be used by a gateway to redirect a host that could use a shorter route [11]. On most systems, the program *ping* is integrated as a standard tool for sending ICMP echo request and reply in order to probe the distance, and test availability of a host [82].

Using ICMP for a covert channel allows transferring data by sending echo request and reply packets [57], [108]. This works by encapsulating the actual content in the ICMP data payload. The approach may help to bypass firewall rules, or captive portals. Additionally the traffic could be encrypted as well. To make the tunnel work, a proxy is required that will receive the ICMP packets, and decode the requests. The proxy will process the actual request and send the response back to the client, again encapsulated in ICMP packets.
As this method is known for over twenty years [92], there are already various tools available, that could be used for tunneling the traffic like LOKI2 [93], ICMPTX [50], icmptunnel [57], or ptunnel [115].

Defense and Detection

Known defense strategies against ICMP tunnels, as outlined by Singh et al., include [108]:

- **Disabling all ICMP traffic.** This also blocks any traffic that may be necessary to check the network status and is therefore not a suitable option for most networks.
- **Partially disabling ICMP traffic.** This method may be error prone, and require thoughtful analysis as covert channels could trick those rules anyhow.
- **Limiting the packet size.** Naturally, the spoofed ICMP packets used in a tunneling scenario will have a larger size than genuine ICMP requests and replies. However, it will be hard to distinguish those packets from legitimate ones that are for example used to determine the capability of the network to transport large packets.
- **Preserving the packet's state.** With this approach, the firewall is modifying the outgoing packets, by changing the sequence number, time to live, payload and checksum. The reply is then checked and only forwarded to the client if it matches the expected preserved values. As this method requires high computing power, it may not be a suitable solution for every use case.

Singh et al. propose an ICMP filtering technique that involves a kernel modification and monitors all ICMP traffic. Data fields of suspicious packets are zeroed out before they are delivered to the outgoing interface. This method is recommended for end hosts by the authors [108].

3.6.2 DNS Tunnel

DNS is generally used to translate domain names to IP addresses (forward lookup) [77], [78]. DNS name servers store Resource Records (RR) for a domain and are responsible for answering queries respectively for forwarding the query to other name servers. The domain name system consists of a hierarchical tree structure, and domain names are

resolved (recursively or iteratively) beginning at the top level domain. The server that is able to resolve the query (e.g., has the corresponding RR) is called the authoritative server.

Some types of records include: *A* for IPv4 addresses (resp. *AAAA* for IPv6 [120]), *CNAME* representing the canonical name for an alias, and *TXT* for text strings that can hold a description. TXT is commonly used for Sender Policy Framework (SPF) records that contain information about authorized hosts that are permitted to be the sender for a specific domain name [60] and therefore used to identify spoofed emails and spam. The DNS protocol is usually transmitted over UDP on port 25, but could also be sent over TCP.

There are different ways of implementation and encoding techniques for DNS tunneling, one method explained by Farnham and Atlasis is to hide the data within the subdomain name, using BASE 32 encoding.
Similar to ICMP tunnels, the DNS protocol can be used to create a covert channel for a stealthy communication [42].

In figure 3.3 an example of a DNS tunnel is illustrated.

Step 1: The data that should be tunneled is BASE 32 encoded and part of the subdomain. The client is initiating the DNS request. Usually for each request, the local cache will be checked first, in order to speed up the process. However, caching will rather be undesired when using a DNS tunnel, and prevented by setting the time-to-live of records to a minimum.
Therefore, the DNS query will be forwarded to an internal DNS server that will most likely be present when being connected to a public Wi-Fi network. Even if this is not the case, the client will have a DNS server defined where the query will be sent to, thus the following steps would be the same.

Step 2: As the internal DNS server is not able to resolve the query, it will be forwarded to a root DNS server. This root DNS server in turn is resolving the last part `.com` of the query.

Step 3: Next, the top level domain DNS for `.com` receives the query and is able to resolve the next part `example.com`, and the query will be forwarded.

Step 4: The authoritative nameserver for `example.com` is capable of resolving the query. Note that this server is appropriately configured to handle the encoded DNS data. In this example, the encoded part `ONSWG4TFOQQG2ZLTONQWOZJO` translates to the text *secret message.* when decoding the BASE 32 string. The server will then send a response, here it is a CNAME response containing
`ORUGS4ZANFZSAYJAMRXHGIDUOVXG4ZLM.example.com` again with the data being hidden in the subdomain.

Step 5: Finally, the client receives the response to the issued query. The client will be able to decode the message *this is a dns tunnel*.

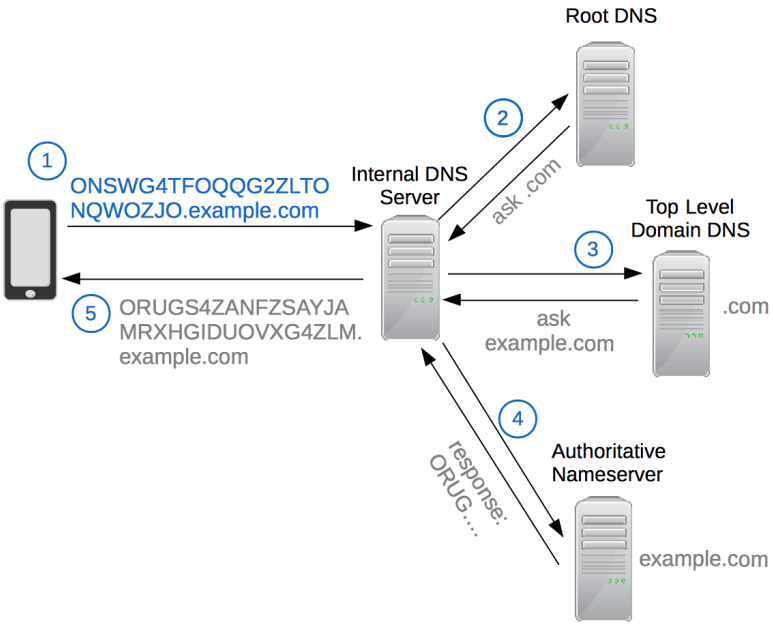


Figure 3.3: DNS tunnel example, adapted graphical version of [124]

This example shows how it would be possible to exchange arbitrary data between client and proxy server. Another possibility would be encoding the payload in a TXT response. As the server cannot contact the client directly (e.g., without responding to a request), the client may send queries periodically to poll the server [42].

One thing that needs to be kept in mind, when using DNS for tunneling is the limitation of the length: one request can have up to 253 characters with a maximum length of 63 for each label (subdomain). Only a subset of 63 characters is valid according to the RFC 1034 standard [77], namely the 52 alphabetic letters `a-z` and `A-Z`; digits `0-9`; and the hyphen symbol `-`. Although upper- and lowercase letters are accepted, DNS is case-insensitive [77], [78]. However, DNS may not only be used for host lookup, but could serve as a hierarchical database in general [39]. For this RFC 2181 [39] and RFC 4343 [38] clarify that any binary string may be used as label, and that there is no limitation to ASCII characters. This means, that in theory various encoding techniques could be used, however, in practice applications using DNS may have restrictions on acceptable input and could drop those packets.

There are many DNS tunneling tools available for end users that use different techniques to tunnel the traffic. The strategies vary regarding the used encoding techniques, and the type of records used for hiding the data [42].

Defense and Detection

DNS traffic may not be monitored in every environment. Depending on the technique or tool used, however, the generated traffic may be easily detectable [42]. Moreover, as tools are freely available, it is possible to analyze them and identify patterns to reveal suspicious requests. Some techniques of recognition include:

- **Restriction of resolving external names.** In order to prevent bypassing of a captive portal, a provider could block all outgoing DNS requests until the client is authenticated [64].
- **Analyzing DNS payload.** Suggested methods include filtering for uncommon records like TXT or attributes of the payload like length of domain or byte size. Another indicator could be the host name requested, as usually names are somehow meaningful and maybe found in dictionaries, while encoded names may have a higher entropy of the character set used [42].
- **Traffic analysis.** This approach tries to detect tunneling by examination of multiple requests and responses. Indicators for DNS tunnel may include amount of packets, number of host names for a distinct domain, geographic location of the DNS server, or the domain history (recently added domains may be more suspicious). Another method is identifying orphan requests, e.g., normally every DNS request would belong to some previous issued command like a HTTP request [42].

3.7 IPv4 vs. IPv6

The internet protocol is the standard protocol that all prior described protocols rely on and that is required to communicate between devices. The original version was proposed in 1981 [95]. As Internet Protocol Version 4 (IPv4) only provides about 4 billion (2^{32}) IP addresses, the new standard Internet Protocol Version 6 (IPv6) has been introduced in 1998 [23], which can provide 2^{128} addresses [24]. Officially, the IPv6 launch started on June 8, 2011 [126]. Measurements show a continuous deployment of the IPv6 protocol [112]. Google, for example, creates statistics about IPv6 adoption, by considering users that access Google over IPv6. They revealed that about 20% of traffic is using the new protocol (as in the beginning of 2018) [51].

CHAPTER 4

Related Work

The topic of this underlying thesis has been inspired by the work of Winter et al. who analyzed the Tor network using two exit relay scanners with the goal to identify malicious exit nodes [125]. While one scanner (*exitmap*) focused on detection of active MitM attacks, the other one, *HoneyConnector*, was sending bait credentials over FTP and IMAP in order to identify exit relays that are sniffing and actively exploiting login data. After monitoring the Tor exit nodes for several months, 40 malicious exit relays had been identified by *exitmap*, out of around 950 exit nodes that were active in the network at that time. Some of those 40 exit relays, however, could rather be classified as misconfigured: 2 of the relays were running anti virus software that tampered with IMAPS; 4 more exit nodes had a OpenDNS policy that would censor websites in the category pornography.
The remaining identified exit relays showed at least one form of MitM attacks including running *sslstrip* [67], or tampering with HTTPS, SSH or XMPP.
HoneyConnector was active for four months and imitated around 27,000 connections for each of the both protocols IMAP and FTP. All in all Winter et al. counted 255 login attempts using 128 different credentials, revealing that 27 exit nodes intercepted data. However, Winter et al. also point out that the attacker may not necessarily be the exit relay provider, although they assume it is likely. In fact, it may be possible that the connection is intercepted after leaving the exit relay, e.g., by the internet service provider, or the network backbone. Overall, 65 malicious or misconfigured exit relays were detected. Two relays were identified as malicious by both scanners.

A report on Wi-Fi security by F-Secure published in 2014, included two short experiments with Wi-Fi hotspots that had been placed prominently in public places around London [41]. In one part of the experiment the user had been presented terms and conditions that needed to be confirmed before granting access to free Wi-Fi. Those included an unusual term stating the user's first born child would be assigned to the provider in turn

for free Wi-Fi, which six people had agreed to during a time frame of about 1.5 hours. For the other experiment a public Wi-Fi hotspot was running for about 30 minutes. Within this time 250 contacts that had been counted. The devices already leaked information about previous connected hotspots. Furthermore 33 devices used the Wi-Fi actively for different services, even revealing sensitive login credentials.
Although these short trials show only a limited snapshot of the real world, it gives some ideas about the popularity of public hotspots and how the user's privacy and data could be exploited.
Sombatruang et al. had a similar research question for their study [114] in 2016, by asking "Why do people use unsecure public Wi-Fi?". To answer this question, they conducted a study, where they first set up a free Wi-Fi hotspot at 14 different public locations in London to monitor and collect traffic. Although no login credentials were revealed, some apps and websites were identified that leaked information and could have been exploited to extract private data. Next Sombatruang et al. interviewed 14 participants in person to discover the people's motivation of using public Wi-Fi. Even though 90% stated they had concerns about public Wi-Fi security, over 50% admitted using it anyhow, with the main reason that it is free. An online survey with 102 participants followed. Those were asked if they connected to public Wi-Fi in various scenarios, which included different locations, urgency, financial and non-financial transactions. For each situation the participants were asked if they chose to use public Wi-Fi when having 25%, 50%, 75% or 100% left on their mobile data plan or roaming. The survey revealed that the decision of connecting to public Wi-Fi is not consistent with the authors' calculated prediction.

Generally detection of eavesdroppers in Wi-Fi networks is challenging and in order to reveal attackers they need to leave some traces. The idea of Bowen et al. is to "confuse, deceive, and detect attackers by leveraging uncertainty" [9]. This is achieved by injecting believable decoy traffic into the network that will eventually trigger an action by the attacker as it is "trap-based". They propose an API that can automatically generate a large amount of bait traffic by analyzing an input. For this, they are using either TCP session samples of a template, or real recording of network traffic. By modifying attributes and content of the input, new and yet authentic data is created. The attacker will be identified if the decoy data is used, e.g., sniffed credentials are exploited. In order to verify that the generated bait traffic is indistinguishable from real traffic in a network Bowen et al. recruited experienced human judges. They were only able to classify about 50% of data correct [9].

Another way of accessing Wi-Fi security has been proposed by O'Connor and Sangster, who presented *honeyM*, a framework that allows to simulate mobile devices [88]. In contrast to the prior discussed methods, this approach is trying to identify malicious activities that are actively attacking mobile clients, exploiting known and even unknown vulnerabilities, and is not limited to Wi-Fi. The honey clients are imitating communication protocols and traffic, including Wi-Fi, Bluetooth, or Global Positioning

System (GPS). The system is monitoring and logging the wireless spectrum of relevant protocols in order to detect any suspicious activities.

In addition to attacks and sniffing within public Wi-Fi networks, privacy plays another important role. Cheng et al. analyzed how travelers may leak private information when using the free Wi-Fi on airports [19]. They collected data at airports, resulting in 20 datasets including over 150,000 sessions from four countries. From 2000 unique identified devices, it was possible to reveal 625 user names in total. Most of them were revealed due to the device sending multicast messages to resolve domain name queries. The names were leaked as the device name is sent along, and those is often directly related to the owner (e.g., *Alice's iphone*). 38 of the names were exposed because websites leaked information.
A recent research about browser history stealing [22] revealed that captive portal providers may expose websites that the user's client visited before. For HTTP this simply works by including image references within the landing page, referencing to the site of interest. If the browser has visited this website before, it will automatically attach the stored cookie, and it will be sent to the captive portal that in turn can evaluate this information. But also websites, that are using HTTP Strict Transport Security (HSTS) can be identified with this approach. Generally, HSTS should tell the browser to only connect over HTTPS. However, as the HSTS information will be cached after the first visit, the captive portal will be able to determine if the client has visited a certain website before.
Another privacy issue comes with mobile devices that are sending Wi-Fi probe requests and thereby leaking information about past visited access point identifiers. An experiment by Freudiger revealed that the probe requests could be exploited by attackers to gain knowledge about the whereabouts of mobile devices [47]. Freudiger also showed that the number of probe requests are depending on the underlying operating system and the amount of known networks .

CHAPTER 5

Analyzing Public Wi-Fi Hotspots

This chapter gives an overview about the most important concepts and structure of the study. First, we discuss how and by whom a user could be threatened when accessing a public Wi-Fi network. Next, we will outline the general idea of the field study and how different attack scenarios and general limitations could be measured.

5.1 Threats and Attackers

When we talk about security risks within a public Wi-Fi networks, we need to distinguish different type of threats respectively attackers. It is important to define those types and their (assumed) intention in order to understand the impact on public Wi-Fi users.

5.1.1 Hotspot Provider

By definition the operator of a Wi-Fi hotspot has access to all data that is sent over the network, for all in- and outgoing connections. It is the providers responsibility to make the service available, e.g., pay for the the infrastructure and the internet access. Thus we can assume it is in the providers interest to benefit from the service as well, in one way or another. One motivation could be to attract more customers, or to enhance their stay (and therefore their consumption) in a bar or restaurant that offers free Wi-Fi. Another interest may be to collect (and monetize) information about the user, e.g., personal data, or surfing habits for advertising or marketing reasons. In our perception, these kind of providers are legitimate, meaning that hotspots are offered permanently by a public authority or a company primarily to serve residents, tourists, or customers.

However, people with malicious intent could also set up hotspots with a genuine-looking name like "Free Wifi". Their intention include collecting and exploiting sensitive data,

executing MitM attacks and so forth.

There are different legal regulations regarding data protection, illegal activities of Wi-Fi users, and copyright infringement that hotspot providers have to comply with [43], [111], [46]. This may be one reason why many public hotspots are using captive portals. First of all, it asks the user to confirm some terms and conditions of use. Second, the captive portal often collects personal information like names, or email addresses. Sometimes even a social media account is required for login. This in turn reveals the user's identity and assigns the activities during the hotspot use to a person.
In 2016 the Court of Justice of the European Union has ruled that "The operator of a shop who offers a Wi-Fi network free of charge to the public is not liable for copyright infringements committed by users of that network" [89]. However, a provider is responsible to limit such activities for example by protecting the network with a password.
In Austria, the law states that a Wi-Fi operator is liable in case the illegal activity was known but no steps have been taken to stop it [4].
Consequently, providers may decide to block or monitor any traffic that could be considered harmful, and may use content filter to deny access to blacklisted websites.

5.1.2 Evil Twin

An *evil twin* describes a malicious access point that tricks a client into connecting to it, by behaving like a genuine and trusted hotspot nearby. To achieve this, the evil twin will use the same Service Set Identfier (SSID) than a well-known provider nearby. If the evil twin is physically closer to a client, the stronger signal is chosen. Therefore the client connects to the evil twin without realizing the attack [18].
It is important to note that this approach is somehow different than just setting up a rogue access point with a common and innocent-looking name as described above in 5.1.1. The key difference is that the client has already connected to the trusted network before, meaning that the SSID is saved. Consequently, the client will automatically connect to this SSID, without any user interaction. Furthermore, the connection will not look suspicious to the user, as he or she expects to be connected with a network having that SSID.
Once the attacker manages to lure users in, he or she will have the power to monitor and alter the network traffic.
Detection of evil twin attacks is challenging and out of scope of this field study. However, as the attacker's motivation is to collect data, it is likely that the login credentials used for the honey traffic will be used by the attacker.

5.1.3 Network Sniffer

Sniffing the traffic in a network is a passive form of attack. The network sniffer is connected with the same public Wi-Fi as the victim and can just monitor and collect all over-the-air traffic. While the pure collection of data is stealthy and hard to detect, the

attacker will usually try to exploit the data he or she gained, for example by using or selling login credentials. Besides the passive sniffing, the attacker could also threaten the user as a MitM and actively modifying content, e.g., by SSL stripping. Thus being able to reveal sensitive information about the user.

5.2 Overview of Procedure and Testing

The goal of this field study is to assess public Wi-Fi hotspots in the wild by means of data interception. Additionally, we test for restrictions of accessible services. A measurement server is setup and acts as a key component by providing two main functionalities: First, it will receive and store all test results. Second it is hosting the honey services, e.g., mock-up services for FTP, IMAP, SMTP and HTTP.
For the Wi-Fi assessment we need to be in physical vicinity of the hotspot. We have chosen Android smartphones as a medium to analyze the Wi-Fi networks. An app, installed on the smartphone device, is able to automatically detect and connect to open Wi-Fi networks nearby. It also identifies captive portals and tries to authenticate automatically. We will explain this procedure in more detail in section 6.2.1. Once the app has connected to an hotspot, various tests are executed. The results will then be sent to our measurement server where we collect all information. In the following we will give a short overview of the different testing scenarios used to assess public Wi-Fi networks and explain why these tests are relevant. Figure 5.1 gives a general overview about the procedure.

5.2.1 Testing Scenarios for Pre-Authentication

Right after the device has connected to a Wi-Fi hotspot we start with the first tests. At this stage it has not been verified yet, whether the internet is accessible. Rather this will be done after these tests have finished. Note that if the hotspot does not have a captive portal in place, all outgoing traffic should be allowed and thus the pre-authentication tests are expected to succeed.
Consequently, these testing scenarios will reveal how captive portals handle different use cases. We expect that a captive portal blocks any content before a successful authentication. However, it still may be configured to accept some harmless-looking traffic, like ICMP or DNS. Therefore, the pre-authentication tests include simple requests using those protocols. Tunneling traffic over ICMP or DNS is a common trick to circumvent captive portals. Consequently, the provider may choose to block those traffic. Furthermore, it is verified how captive portals are handling HTTPS connections before authentication. This may reveal improper redirection attempts of the captive portal, resulting in SSL certificate errors.

5.2.2 Testing Scenarios for Post-Authentication

After successful authentication with the captive portal, traffic should be routed to and from the internet.

One part of these tests includes the spreading of honey traffic. Furthermore, we try to reveal MitM attacks. The other part focuses on restrictions that are dictated by the network including blocked websites and services.

Revealing Attackers and Sniffers

Different user sessions are simulated over FTP, HTTP, IMAP and SMTP. Those services have been chosen as the traffic is sent unencrypted by default and we assume the login credentials are interesting for an attacker. For each session and each service a unique username and password is used. This way, it will be possible to identify attackers later on, once they are trying to reuse the intercepted credentials.
Every user session should look authentic, so after successful login some actions will be executed. For example: sending an email over SMTP or downloading a file using FTP, before the simulated session is terminated.
In addition, it is tested whether the greeting banners of the FTP, IMAP and SMTP services are as expected. Each of those ones should send a custom greeting banner once a connection has been established. In case the traffic is intercepted or routed through a proxy, this banner may change and indicate that we are not directly connected with the service.

Moreover, we identify SSL stripping attacks by requesting a website, that should automatically redirect to HTTPS. As we are aware of the expected redirect to HTTPS and the expected content of the website, we are able to reveal any irregularities.

Blocked Content

In order to determine whether public Wi-Fi hotspot filter specific content, we simply try to access certain websites. We selected representational sites for the following categories: gambling, pornography, news, blogs, social media, streaming, and file sharing.

Social network websites have been known to be banned in some countries like China, Iran or North Korea [86], [122]. Also Turkey blocked the access to Twitter in March 2014, but the constitutional court ruled that the ban was illegal a few days later [6]. Turkey is facing reoccurring reports of blocked or throttled access of social media websites, e.g., in 2016 [8]. Thus blocking social media sites including Twitter and Facebook is not unusual and could be political motivated. On the other hand it is also a common strategy to block those websites for productivity reasons at the workplace. Public Wi-Fi hotspots may have similar content filter in place or providers may have other reasons of blocking the use of social media.

File sharing is another hot topic as the content provided and shared may violate copyright laws. One very popular website, with high media coverage is The Pirate Bay. The European court ruled in 2017 that websites redirecting and linking to content

that is violating copyright may be held liable as well [110]. Regulations regarding downloading copyright protected material have become stricter, e.g., in the UK the maximum sentence has been raised from 2 to 10 years in prison [14]. It is not unusual for users to receive warnings, action for an injunction and penalties, and has for example become practice in Germany [36]. The UK started an educational program in 2017 by sending out warnings [109]. Consequently, websites that are known to violate copyright regulations are likely to get blocked by public Wi-Fi hotspots or even Internet Service Provider (ISP).
Another website used in this category is *mp3skulls*. It was a popular site for sharing music, but has been shut down in 2016 as it lost a legal battle against the music industry [72]. However, other versions of the website popped up, some of them claiming to use only legal content. We used `mp3skulls.top` for our tests and as this domain name ran out, replaced it with `mp3skulls.to`.

The *streaming services* category includes legal as well as potentially illegal websites. For the legal services we are focusing on popular ones including YouTube, Netflix, and Hulu. Those sites may be blocked by providers as they could use a large band width or result in a prolonged stay at the spot, which may not be in the interest of every public Wi-Fi provider (e.g., public service, bank etc.). The websites `kinox.tv` and `movie.to` have been chosen to represent streaming services that are known to infringe copyright laws. The question if streaming of copyright protected material was illegal is still a debated question in some countries. While Austrian's minister of justice claimed in 2016 that streaming was not illegal [12], the European court ruled that devices which are sold with build-in functionalities to stream copyright-protected content are illegal in 2017 [90]. This was also the first time that streaming was considered as copying, which could have implications for consumers of other streaming services as well [113]. Nevertheless, the Austrian supreme court obliged ISPs to block certain websites that are known to infringe copyright starting in 2014 [26] and keeps doing so since then [56], [25].

With the category *blogs* we are including websites that may have been considered as controversial or have been subject to censorship. For example, in Turkey in 2016, WikiLeaks has been blocked after government emails have been published on the platform [127]. Medium, a website hosting blogs and publications, has been banned by China [53] and Malaysia [20]. The other websites selected for this category may have witnessed similar situations or could in the future.

Additionally, websites that are reporting *news* have been blacklisted before, e.g., the New York Times was blocked in China in 2012 [13], and The Guardian was unreachable some time in 2014 [80]. Besides those two big news sites, we also included TorrentFreak for the tests, a website that is focusing on news about copyright, privacy and file sharing.

Online *gambling* have different regulations in each country. It may be legal in some

countries, or requires a license. Furthermore, gambling may be blocked for other reasons such as protection of minors.
Content filter may also be the main reason why *pornography* could be blocked from public Wi-Fi.

Besides websites, we are interested in blocked ports. In this scenario we check if TCP or UDP traffic can be routed to some pre-selected ports. Those include for example the standard ports for Session Initiation Protocol (SIP), or common VPN connections. We discuss the selected ports in detail in section 6.4.4.

Additionally, we test whether IPv6 is supported by the network. As the use of this IP protocol is rising it may be of interest to see whether free public Wi-Fi hotspot have the capability to resolve and use IPv6 addresses.

5.3 The Measurement Procedure in a Nutshell

In figure 5.1 the communication between the measurement server and the *Honey Client* app (running on an Android device) is pictured. It outlines when an unencrypted or encrypted channel is used. Furthermore, it shows the data flow between the components in a typical scenario.
Details about the server and client app will follow in the next section.

Step 1: The Honey Client app has established a connection to a public Wi-Fi hotspot. The *pre-authentication tests* (see also section 6.3) are performed. We do not know yet whether a captive portal is in place, and if it may block outgoing requests.

Step 2: It is tested whether a captive portal is blocking request and if so, the login procedure (see also section 6.2.2) is executed. Note: in this pictured scenario the authentication is successful and so we can continue with the testing procedure.

Step 3: We need to check whether login credentials are available. Those will be used for the honey services. Credentials are requested and sent over a secure channel and stored on the device locally.

Step 4: The *post-authentication tests* (see also section 6.4) are performed.

(a) The honey traffic is initiated by simulating user sessions. Note that the login credentials for all services are sent over an unencrypted channel.

(b) In order to test for SSL stripping attack, a website is requested over HTTP that should reply with a redirect to HTTPS.

(c) The app requests each of the pre-selected websites in order to identify blocked content.

(d) A simple message is sent to the server using the ports we want to test. If the app receives the expected reply, we can assume that the port is open.

(e) The app calls an IPv6 address. Only if the address can be resolved, this test is marked as successful.

Step 5: Finally, the results are reported to the server.

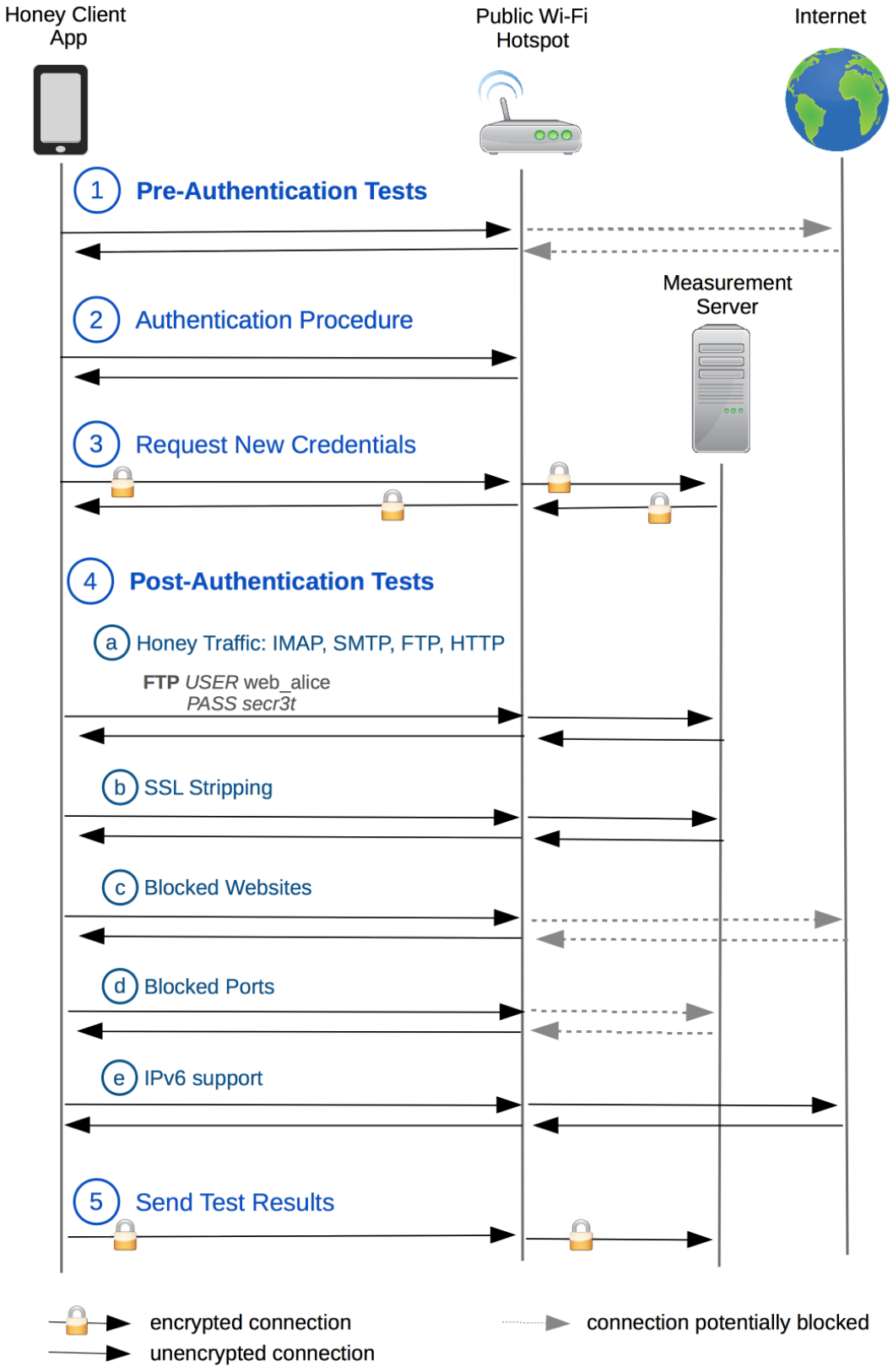


Figure 5.1: The measurement procedure

CHAPTER 6

Design and Implementation

In this chapter we discuss the overall design of the components. We start with describing the implementation and functions of the measurement server and the Android app (also referred to as *client*), and how those components interact with each other. We discuss details of the measurements and tests, and outline how we examined the capabilities, limitations and vulnerabilities of a network. Note that we use the term *test result* to describe the data that has been collected by one testing device during one test run (e.g., executing all testing scenarios described in the following).

6.1 Measurement Server

The measurement server component plays a the central role as it has some major responsibilities including:

- Creation and distribution of user credentials for the honey traffic.
- Providing services for IMAP, SMTP, FTP, HTTP.
- Logging access and login attempts.
- Communication with clients over a secure channel.
- Storing test results.

We use a virtual root server hosted on IPAX[1], running Ubuntu 16.04, that can be accessed over Secure Shell (SSH).

[1] `https://www.ipax.at`

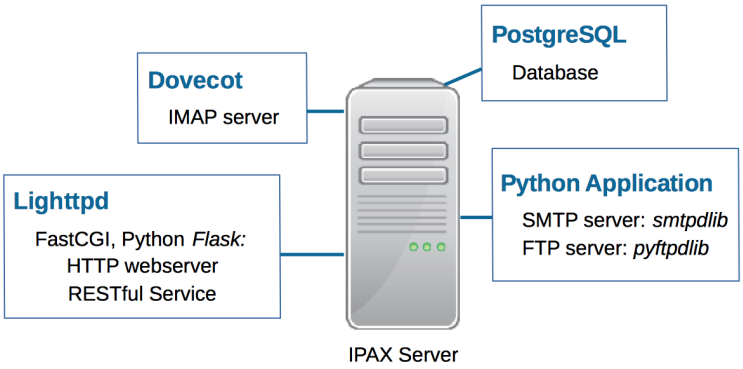


Figure 6.1: Main services running on the server

In figure 6.1 an overview of the server is given. For the webserver, we use *Lighttpd*, which runs two FastCGI applications: one for the webservice, that is required for the HTTP honey traffic; the other one is a RESTful service, which is used for the communication with the client, e.g., for the distribution of credentials, and receiving test reports. Note that the RESTful service is only accessed over HTTPS.
The IMAP service is provided by *Dovecot*, an open-source mail server. The other services required for the honey traffic (SMTP and FTP server), run in a standalone Python application.
All data, e.g., valid user credentials, login attempts, test results etc., is stored into a single *PostgreSQL* database. This database runs on the measurement server and is accessed by all services in order to verify login credentials, and store successful or failed login attempts.
In the following we will explain the main functionalities of the measurement server in more detail.

6.1.1 HTTPS configuration

We use *Let's encrypt*[2] as certificate authority, and tested the measurement server's configuration with *SSL Labs*[3] to ensure it meets the standard requirements. *Lighttpd* is configured to redirect some URLs automatically to HTTPS. This includes the RESTful service. Note however, that the client will not rely on the redirection for this service, but rather use HTTPS with a pinned certificate to ensure a secure communication.

[2] `https://letsencrypt.org`
[3] `https://www.ssllabs.com/`

Communication with the Client

The measurement server is responsible to create login credentials for the honey services. Thus, it is also a requirement to distribute those credentials in a secure way to the client applications. Therefore, a RESTful interface has been designed, that the clients will only access over HTTPS. It allows the clients to bulk-load new credentials for the honey services. Furthermore, it is used to upload test results and files. The RESTful service runs as a FastCGI application on *Lighttpd* with the Python microframework *Flask*[4].

SSL stripping

We use the capability of automatically redirecting for evaluating SSL stripping attacks. The measurement server will redirect any URL that is accessbile under `.../home/.*` to HTTPS. For the test, a simple website `index.html` is requested. This site contains an HTTPS link that could be stripped off by attackers.

6.1.2 Credentials and Accounts

The login credentials that are used to access the service are unique, e.g., no username or password exists more than once. This allows to track unauthorized accessed: we can assume that every successive login using the same credentials, could only be conducted by an attacker who has sniffed the data sent over Wi-Fi.
To ensure the login credentials for the provided services are indeed unique, one central authority needs to verify this condition. It is within the responsibility of the measurement server, as there are potentially countless Android devices in use for the field study, which actually spread the honey traffic using the app.

As the user credentials need to be unique and look authentic, we use real world names as input with random pre- or suffixes. For this, we extracted lists of names from a census in 1990 published by the United States Census Bureau [15]. So we end up with one list for surnames, and one combined list for male and female first names, derived by Meranda [75]. This sums up to lists with 5,163 distinct first names, respectively 88,799 surnames. Those lists serve as input for the username creation.

Each username has a *type*, e.g., is only valid for one of the four honey services.
The process of creating new credentials is the following: First, we select random names from the provided lists. Then, in order to expand the range of possible usernames, we are adding some characters to it. For FTP users we select a random prefix: *web_*, *ftp_*, *usr_*, *user_*, or the empty string (e.g., no prefix is prepended). This is a similar strategy to the one used by Winter et al. [125]. Then, for all types of users, we create a random suffix containing zero to four characters from lowercase letters and digits. For HTTP, SMTP and IMAP users we also select a separator (point `.`, underscore _, hyphen -, or

[4] `http://flask.pocoo.org`

empty string) that would be appended to the selected username before the chosen suffix. The password for the user is created from random characters, using ASCII letters and digits, with a length of 6 to 15 characters. In order to ensure the credentials are unique, we set a database constraints for the username and password, so only unique values are inserted and consequently distributed to the clients for use.
We pre-populated the database with credentials, so that the Android testing devices can receive data immediately after requesting those over the RESTful service. The Android client will receive a bulk of credentials for each request that are stored locally on the device, in order to speed up the process. Every time new data is requested by a testing device, we also trigger the process of creating new credentials. This way, we ensure that sufficient data should be available all the time.

Besides the creation of usernames, we generate the corresponding accounts and fill those with plausible data. Thus, for each FTP and IMAP user we pre-populate the accounts with random files respectively emails.

6.1.3 Honey Services

The honey services are not fully functional, and only serve for the honey traffic distribution and as a honey pot. Each service is configured to directly access the database in order to validate the provided user credentials. All successful and unsuccessful login attempts are logged and stored into the database. We only count unsuccessful logins that are using an username and a password, excluding any attempt of only connecting to the service without providing credentials.

IMAP

Dovecot is configured to serve as an IMAP email server, that only allows plain text login. For each account, we are populating a mailbox with a random amount of emails. We prepared a small subset of subjects, and senders that we randomly assign to the emails. Moreover, we set different flags, which may be: seen, draft, flagged, passed, replied, trashed.
The Dovecot server does only support retrieving of emails and typical related functions like searching, deleting, creation of subfolders, etc.
The measurement server logs any login attempt in a file. Therefore, we run a script every night that parses the log file and stores all successful and unsuccessful logins into the database.
Dovecot runs on the standard port 143.

SMTP

The SMTP server is implemented with Python, using the *smtpd* module from the standard library, and has only limited functionality. The server responds to `EHLO` requests, supports login with `PLAIN` and `LOGIN` command, and allows logged in users to send emails. Those

emails, however, are not delivered to recipients but just logged into a file.
The SMTP service is configured to accept credentials that are valid for IMAP as well. This way we can identify attackers that are assuming IMAP credentials could be valid for the SMTP server as well.
The SMTP server is bound to 2255, so that there are no root privileges required. However, a *iptable* rule is used to forward all requests from the standard port 25.

FTP

For the FTP server we are using the library *pyftpd*[5]. We also use a *iptable* rule to forward traffic from port 21 to 2121, where the FTP service listens for incoming requests. Each account can only access it's corresponding home directory, which is populated with random files. Those files should attract attackers by pretending to hold sensitive data like credit card information, login data, or contract details. We included different extensions and file types, e.g *TXT*, *DOC*, *PNG*, containing random data and having different file sizes. As some file names will be listed during the honey traffic test, we include some very promising names like *creditcard_company.txt*, *master_card.txt* or *gmail_login.txt*. Some users also have a directory `www` that may give the impression to hold the data for the webserver.
Each user has only read access to the home directory, but any attempt to write data will be logged as well.

HTTP

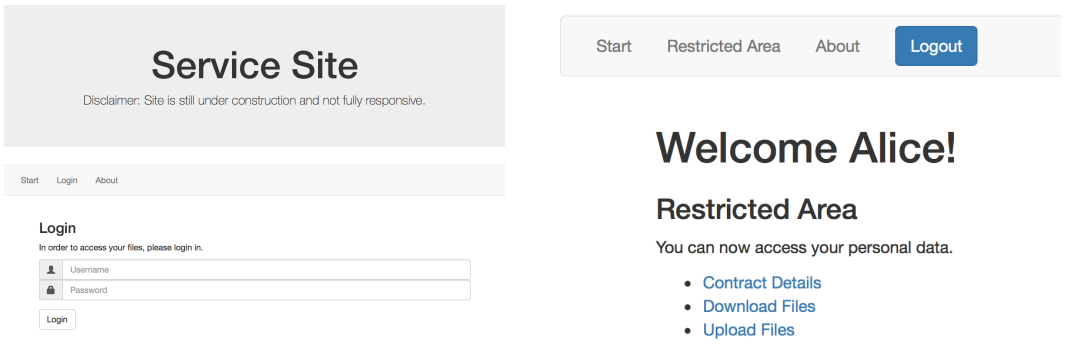
We constructed a simple website, using Python and *Flask*, that is deployed as a FastCGI application on the webserver *Lighttpd*. The site looks like it has been configured improperly, as it actually has an active HTTPS module, but does not redirect the user. So the service will be accessed over HTTP only.

The website imitates a service for employees and even outlines in the header *not being fully responsive and still under construction* (figure 6.2a). However, it has a login functionality and promises one can access personal files in the *restricted area*. After login, three additional functionalities are displayed, trying to attract attackers: *Contract Details*, *Download Files*, and *Upload Files* (figure 6.2b). We think that the part with uploading files may be the most attracting one. The linked sites do not exist though.

6.1.4 Responding to UDP and TCP Scans

Th measurement server also responds to the blocked port tests. Therefore, we provide services that send a simple string as response to any incoming requests on the ports of interest. Those ports are listed and discussed in detail in section 6.3.
For the TCP port testing, we use *xinetd*, a super-server daemon that is capable of

[5] `https://pypi.python.org/pypi/pyftpdlib/`



(a) Login

(b) Restricted area, displayed after login

Figure 6.2: Screenshots of website appearance

listening on ports and launching a service for this request. It can be simply configured to respond with a string, by using the `echo` command.
The UDP protocol requires a different approach as *xinetd* only supports TCP. Therefore, we construct a simple UDP server facilitating the standard `socket` library of Python. We use *iptable* rules to forward incoming requests to the same UDP server instance.

6.2 Android Client Application

The client application is responsible for executing the actual tests in public Wi-Fi networks and reporting the results to the measurement server. We have chosen an Android implementation as we already had experience with developing apps. The Android API provides the functionalities required, such as detection of and connection to hotspots. Devices running the Android OS are in general small and portable, which makes carrying it around easy and accessible. Moreover, we counted on having a set of devices available as we already were in possession of some Android devices.
We designed the *Honey Client* app, which is capable of automatically connecting to public Wi-Fi hotspots nearby, authenticating to captive portals, running tests, and uploading results. All of these steps require little to none user action, as the app runs in the background, while the device's display is turned off. Implementation details are discussed in the following sections.

6.2.1 The Honey Client App

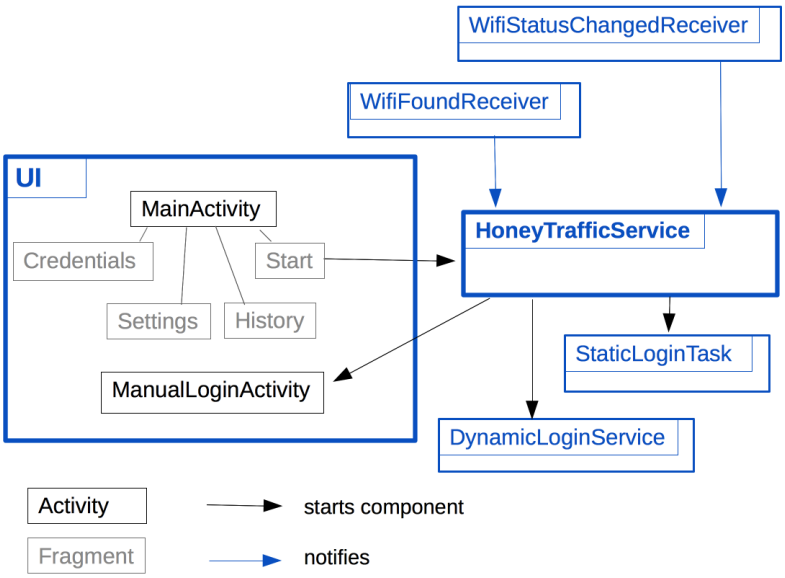


Figure 6.3: Main components of the Honey Client

Figure 6.3 gives an overview of the main components of the Honey Client app. Contrary to most apps, the User Interface (UI) of Honey Client does not contain the main functionality. The UI allows the user to access some features such as the history of tested Wi-Fi networks. However, the main tasks are executed in a *service*, an Android component that can run in the background. This *HoneyTrafficService* can be started and stopped by clicking a button on the app's start screen (see figure 6.4a).
We use *broadcasts* that are sent by the system to get notified about available Wi-Fi networks. The broadcast receiver is represented as *WifiFoundReceiver* in figure 6.3. The HoneyTrafficService chooses a Wi-Fi SSID and initiates a connection request to this network. Another broadcast receiver, *WifiStatusChangedReceiver*, notifies the HoneyTrafficService about network state changes, e.g., about any connection or disconnection from a Wi-Fi network.
Once the device is connected to the chosen network, the HoneyTrafficService performs the pre-authentication tests (see section 6.3). Next, it is evaluated whether a captive portal is in place. Therefore, the *StaticLoginTask* tests the connection and if it detects a captive portal, the authentication procedure is started. This process will be explained in more detail in section 6.2.2. Depending on the type and complexity of the captive portal, the *DynamicLoginService* and *ManualLoginActivity* may be started consequently.
Once we ensured that the network is not blocked by a captive portal anymore, the post-authentication tests (see section 6.4) are executed. The test results are uploaded and finally the network is removed.
Note that only the ManualLoginActivity will require user interaction.

Hotspot Selection

The WifiFoundReceiver will receive a list of all networks that have been detected nearby including encrypted ones. As we are only interested in public hotspots, we ignore any network that has an encryption capability listed. We also exclude any network that is likely to be an ad-hoc network by filtering for common names like *hp-print*, *setup* or *chromecast*. This way we can save precious time that would otherwise be wasted by trying to connect to an ad-hoc network, which would have failed eventually.
Furthermore, we only consider hotspots with a certain signal level, as we assume that the connection to a low signal network will be lost soon anyhow. The remaining list of available networks is sorted by the signal level, so that we choose networks with a stronger signal first. In addition, we only try to connect to networks, that have not been tested recently, e.g., within the last ten minutes. For this exclusion we are relying on the results stored in the database. In case a test device is located on the same place for some hours, the tests and consequently spreading of honey traffic will only run every few minutes on the same network.

Special Remarks to Testing Scenarios

In order to run tests in parallel and to speed up the process we make use of *RxJava 2*[6]. The library allows to zip tasks which can run in parallel and notifies once all tasks are finished.
The ICMP test is executed with the Android system tool `ping`, while for the DNS tunnel test we utilize the library *dnsjava*[7]. For communication with the services over IMAP, SMTP and FTP, the *Apache Commons Net*[8] library is used. For the FTP traffic we apply the passive mode to download files, which is required when the client is behind a NAT.

All communication with the measurement server involving HTTPS is initiated with a pinned certificate. For this, we use the public chain certificate for *Let's encrypt*, following the approach suggested by Android [28]. Pinning the certificate is necessary for two reasons: first, older devices may not have up-to-date trust chains; second it ensures that we are indeed talking to our measurement server.

For the honey traffic and the corresponding credentials, we use a *fail safe* approach: the selected username and password is marked as *used*, before the test begins. This way, we can be sure that no credentials are accidentally reused. The Wi-Fi connection can be lost any time and the tests may not have been fully executed.

[6] `https://github.com/ReactiveX/RxJava`
[7] `http://www.dnsjava.org`
[8] `https://commons.apache.org/proper/commons-net/`

UI Features

Although, the credentials for the honey services, are automatically requested if required and stored locally into a database, the UI provides an additional functionality to load credentials manually. However, a manual request is only necessary in case a network blocks any HTTPS connection, as this will also prevent requesting credentials over a secure channel. An overview of the amount of unused credentials is available within the app, as well as the amount of already used ones. Furthermore, the UI has a setting fragment, that allows the user to opt-out of categories for the blocked website tests (see section 6.4.3).
In general test results are uploaded automatically. In case the internet connection was lost or HTTPS is blocked, the results are sent to the measurement server during the next evaluation. Additionally, the upload can be triggered manually within the history view of the app.
The history also provides details for the blocked ports and blocked websites tests. Moreover, the history can be searched for SSIDs.

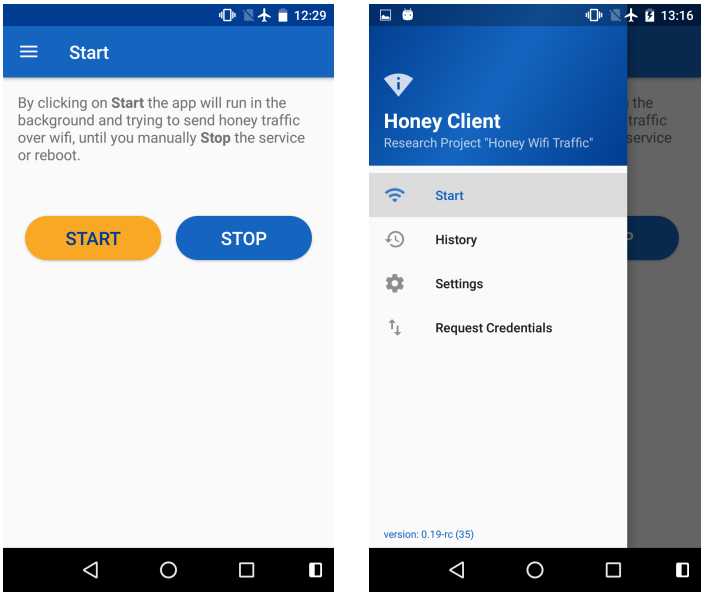
Compatibility

The app is backward compatible until Android API level 15 (Android 4.0.3 Ice Cream Sandwich) and has been extensively tested on different devices and versions up to Android API 23 (Android 7.0 Nougat). With this range of supported versions, we are targeting over 90% of devices running Android, as the latest statistics of platform versions revealed (February 2018) [33]. We also had a quick opportunity to test the app successfully on Android version API 27 (Android 8.1 Oreo).

6.2.2 Captive Portal Authentication

The requirement of automated interaction with captive portals is challenging. While some captive portals only require to click a button or check a box, others have more complex websites in place that may have been designed to prevent automating this process. Figure 6.5 shows some samples of captive portal landing pages that require the user to accept terms and conditions. The app should be able to automatically login to a fair amount of Wi-Fi hotspots. We collected samples from captive portals in the real world, analyzed those and tried building a solution that would be able to manage the sign-in process.

The automated login approach consists of three strategies that are run consecutively. In order to check that the access to the internet has been granted, we call a landing page on our measurement server (http://37.252.185.26) and test if the content returned is as expected. Once we can verify that the captive portal does not block requests anymore, the procedure is stopped. The automated approaches try to provide randomized dummy data for what we think is acceptable input, e.g., name and email. We will further try to naively fill forms with a static phone number and room number if asked, although these input fields will probably have constraints defined and could reject the input. However,



(a) Start screen (b) Menu

Figure 6.4: Screenshots of the Honey Client app

we do not target networks that require extensive information, e.g., passwords, token, or identity verification in order to authenticate. In figure 6.6 screenshots picture some of those captive portals.
In case we could not login automatically, we send the captive portal responses to the measurement server. This will help identifying problems and improving the algorithm in future.

Static Login

This method is part of every test run, as it checks whether there actually is a captive portal in place. In figure 6.3 it is represented as *StaticLoginTask* and runs as an *AsyncTask*[9]. It has been named *static login* as it is simply crawls the website that the captive portal is redirecting to, and stores it locally. The stored file is then statically examined and semantically interpreted, e.g., scanned for links, forms, and keywords. Next, we try to fill those identified forms, follow redirects or click on links. Each of those interactions will trigger another request and the response will again be stored locally in a file, and then processed. After each step we are checking whether the internet access is still blocked. The login procedure is stopped in case we managed to authenticate.

We are using the *OkHttp* library to crawl websites, as it allows intercepting network

[9] an Android specific class that executes in the background

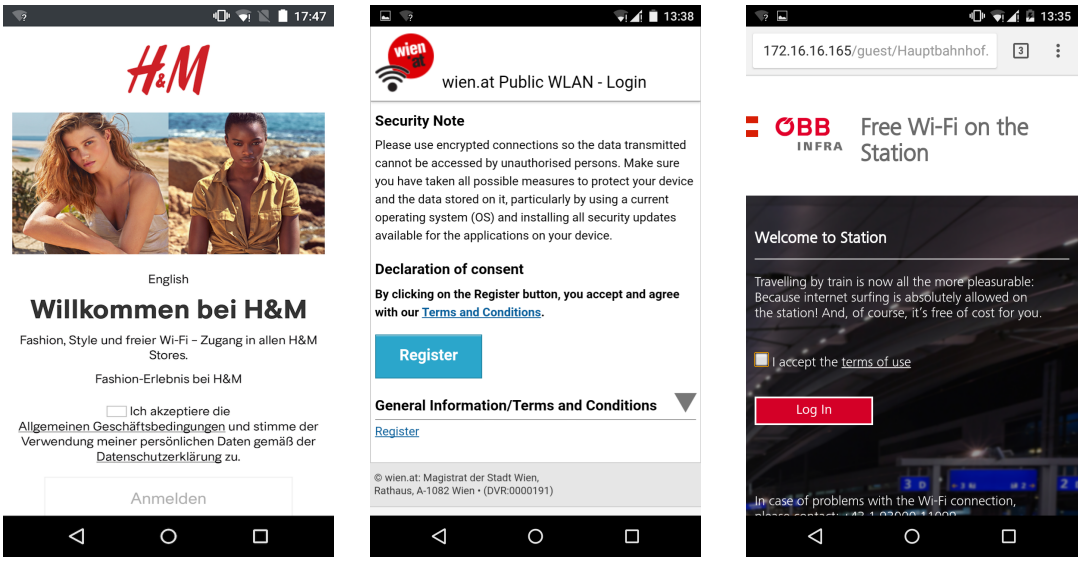


Figure 6.5: Samples of captive portals

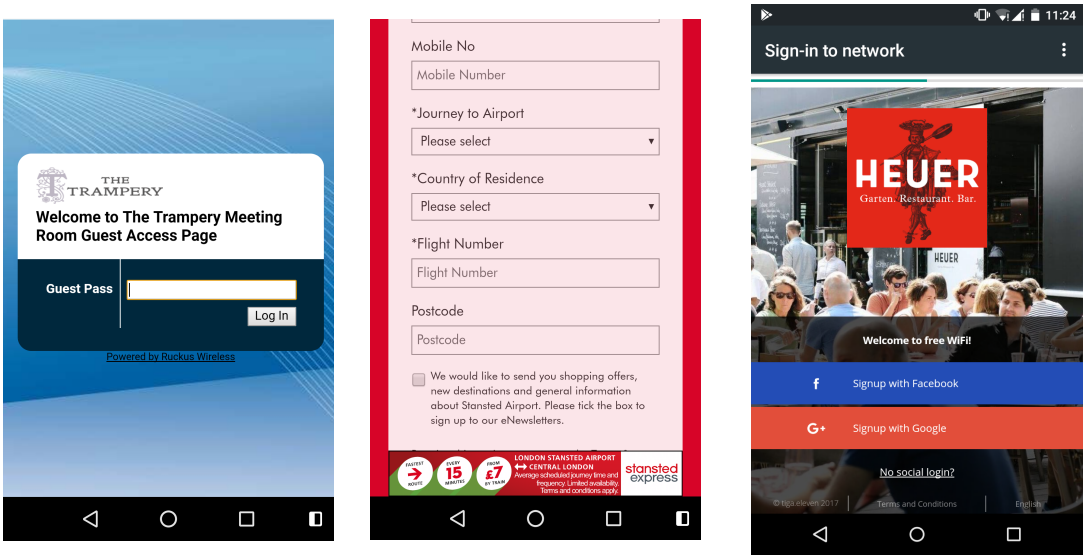


Figure 6.6: Captive portals requiring additional information

requests (important for debugging and logging reasons). Furthermore it is capable to follow redirects from HTTPS to HTTP and vice-versa automatically. This is bad practice in general but acceptable for the captive portal login and speeds up the login process. In addition, we check the response for any *meta refresh* tags and follow such redirects immediately. This tag is an HTML element telling the browser to reload, and is sometimes used to force a redirect.

The following steps are conducted recursively. E.g., for every request made that will load a new website, the response is stored locally and we continue examining this new response first. After 40 different websites have been loaded, we stop the process as we can assume that this approach will not be successful.

- **Find form on the website.** First, we check if the website contains any form. Forms are usually used to ask for confirmation or additional data to access the internet, for example a checkbox could be required to acknowledge terms and conditions, or it may be required to fill a name. If a form is identified, we fill it with dummy data. As the approach is static, we actually construct the `GET` or `POST` request that the form would trigger if a real user was interacting with the website. If the form requires a password input, the static login process is stopped and we carry on with the next procedure.
  Note, that we do not stop entirely. We have experienced some websites that fill out password fields running JavaScript. This may be an approach to prevent such static processing and automated handling of captive portals.

- **Extract promising links.** Next, we identify links that can be followed. Therefore URLs are extracted from the website, that are part of link tags (`<a>`). We have a list of keywords defined (see table 6.1), that is used for ranking and ordering the list of extracted hyperlinks. Those keywords are a mix of English and German words, and result from the collection of captive portal responses that we have analyzed. In addition we try to match patterns on the website that may contain hyperlinks as well, e.g., redirects that would be interpreted by a browser making use of the `location` attribute and append it to the list of identified links.

  Furthermore, we try to construct links that are used for *Cisco Meraki* [10] hosted solutions. We witnessed some captive portals making use of Meraki that built the link with a JavaScript function. An online documentation [118] revealed how links are constructed in general and we follow this approach. Although this will not work for all Meraki networks, it promises a faster login success for some captive portals.

[10] `https://meraki.cisco.com`

connect	verbinden	accept	akzeptieren	login	continue
log in	logon	einloggen	internet	surf	fortfahren
agree	access	zustimmen	register	registrieren	bestätige
anmelden	free	wifi	get online	signup	wlan

Table 6.1: Keywords used for ranking

The static approach is simple and straight forward as it does not require a browser and can run in the background. However, it is useless for captive portals that rely on JavaScript to display content, or to interact with the website.

Dynamic Login

In order to interpret JavaScript on a website, we came up with a solution that uses a `WebView`[11]. We use it within a service, *DynamicLoginService* (figure. 6.3), so that it is able to run in the background as well. In general, a WebView requires to be visible in order to load content (e.g., the screen of the device is turned on, displaying the WebView). However, we could use a system application overlay with zero height and width, making the WebView believe that it is the foreground while it is not.

The website is then loaded into the WebView which will automatically follow redirects, just like a browser would do. Once the page loading has loaded, we check whether the internet can be accessed. If so, we can stop the process. Else, we execute a JavaScript, that will extract the Document Object Model (DOM) of the website and store it locally into a file.
The following steps are essentially the same as with the static login attempt:

- **Find form or links within the DOM.** If we find a form, we build a script that can be executed on the WebView and will fill input fields, check boxes, perform clicks, etc.
- **Execute JavaScript.** The prepared script is then executed on the WebView. Normally, running a script will trigger the site to load new content, and we can then check the next DOM. However, it may happen that executing JavaScript does not have any side effects. In that case the same DOM will be analyzed again, searching for other possibilities, e.g., links to follow, and create a new script that will be executed.
- **Check previous site.** In case we run out of options (i.e., we already covered all possibilities to interact with the site), we will try to reload the previous site (if any) and continue with the process.

[11] a standard and browser-like view on Android, capable of displaying websites

Note that this form of interaction with the website is not reliable and in general hard to test and verify as using a WebView within a Service class is not an intended use case for Android.

Manual Login

As a last resort, when previous approaches failed, we ask the user for help. The user will be notified that a manual authentication is required. However, he or she is free to ignore this request. At the time the network is out of reach, it will be removed and the process will simply continue. In case the user is willing to help, the captive portal will be displayed in a common WebView, and the user can interact with it normally, like in a webbrowser. Once the app could verify that an internet connection has been established, the tests will start automatically.

6.2.3 Collecting Data

We also collect information about the Wi-Fi networks that Android provides. This includes for example Basic Service Set Identifier (BSSID), IP, gateway, DNS addresses. Furthermore, we extract details about the test device itself like OS version, brand, etc. This information may be helpful to identify problems with specific OS versions. We also create a Universally Unique Identifier (UUID) for each installation of the app. Note that this is not a unique identifier of the device. Rather this UUID will change if the app is reinstalled or data is deleted. This UUID is sent along with test results as well. It has been introduced so that we could match uploaded files, which will also include the UUID, with the test results. This may be helpful when analyzing failed automated login procedures, as results may vary depending on OS version and hardware capabilities.
In addition, Android's *LocationManager* [34] is used to store an estimated location along with the test result. It requests the last known location of the device. It relies on information from the network location provider for cell tower and Wi-Fi based estimation. Thus, all devices that do not have an active SIM, will only receive updates from Wi-Fi location. As a result, the position is not very accurate. It may happen that the location is out-of-date for example when the device was turned off and moved to another city.

6.2.4 Obstacles

Before actually building the app, we had to collect samples of real-world captive portals in order to find strategies to automate the process of authentication. Implementing the Android client required a lot of manual testing, as devices would show different results for the automatic captive portal login and especially the testing speed. This may be due to different hardware capabilities or OS versions. Moreover, mobile devices may lose the Wi-Fi connection at any time, and we had to make sure that the app could handle this without losing any data.

The app was constantly adapted as we collected feedback from users and tried to improve the testing performance and usability.

Android SDK Changes

Android publishes new SDK versions regularly, however, updates are generally not available for all devices [33]. Consequently, we faced some challenges, which we briefly discuss in the following.
Starting from Android 4.4 (API 19) the WebView relies on Chromium [31]. We witnessed different behavior using the *dynamic login* functionality on devices that are running lower API versions, e.g., scripts were not executed reliably or showed different results. However, this may also be related to the hardware components, as these were all low-budget devices.
With Android 6.0 (API 23) the behavior for Wi-Fi networks was changed [32]. Prior the change, any Wi-Fi connection was used as the main network, regardless whether the network actually had internet access. However, starting with API 23 the behavior differs: in case Android detects a Wi-Fi connection that is not allowing access to the internet (yet), it chooses other available networks over the Wi-Fi network. This means when the device is connected to a public Wi-Fi hotspot with a captive portal in place, it would still make use of cellular data if it is available. Thus we would not be able to detect the captive portal. In order to prevent this, the process must be explicitly bound to the Wi-Fi network. This works fine for almost all cases. However, we experienced problems when binding the network and using the `ping` command, which still seems to run over cellular data. For this reason, we exclude the ICMP tunnel test if we detect an active mobile data connection (see also section 6.3.1).
Furthermore, changes with the doze and standby mode were introduced with Android 6.0 [27]. These changes affect the app as it restricts background tasks and stops scanning for Wi-Fi networks. We used a few tricks to workaround the restrictions such as whitelist the app (i.e., so that it can run in the background), hold a partial wake lock that allows accessing the network while in stand-by, and used a *JobScheduler* to request a new scan for Wi-Fi every few minutes.

6.3 Pre-Authentication Tests

The following describes the tests that are conducted, once the Honey Client app managed to connect to a Wi-Fi hotspot. Those tests are executed in every case, e.g., regardless of captive portals and internet access.

6.3.1 ICMP Tunnel Test

For this test, we use the system tool *ping* on Android, and try to send and receive five packets by executing `/system/bin/ping -c 5 37.252.185.26`. The IP address is the one of our measurement server. If five packets could be received, the test is marked as successful (if at least one packet was received, we mark the test as partially successful).

DNS Query Request	Type	Response
Ctunnel21356sdfasd234123480-gsdfgnl324280 tgsdfgd012345678901.atrox.org	CNAME	C2tunnel324230870t353214ndfsv 082340123456789012.atrox.org
Ttunnel21356sdfasd234123480-gsdfgnl324280 tgsdfgd012345678901.atrox.org	TXT	"this is a dns tunnel test, 21356sdfasd234123480-gsdfgnl324280tgsdfgd"

Table 6.2: DNS tunnel test: resource records

Note that this only tests whether hosts can be reached that are not within the same network. There may be strategies in place to filter uncommon packets, as discussed in section 3.6.1.

6.3.2 DNS Tunnel Test

We send two DNS queries, that should be answered by a DNS server that we have configured. The first query, should be answered with a CNAME record, while the second one expects a TXT response. Only if both responses match the expected ones as shown in table 6.2, the test will be marked as successful. The *time to live* for each entry is set to 30 seconds, so that the cache will not keep the entry for long and we can rely on the results.

6.3.3 HTTPS Interception of Captive Portal

Captive portals may have different strategies for handling HTTPS requests before the user has authenticated and those should be revealed with this test. For the first tests, we used `https://google.com` (until mid of December), but continued testing with `https://www.microsoft.com` in a new version. The reason for changing the tested website was a suspected white-listing for popular websites within some captive portals, including the Google search. To evaluate the result of this test, we are collecting the following properties:

- Response code for the request
- URL of the final website loaded
- Protocol of the connection (which should be HTTPS)
- Title of the website we have been finally redirected to

If the requested site could not have been reached (e.g., an exception has been thrown) we report the details for this exception. This may include timeouts in case the connection to the Wi-Fi hotspot was lost, but also SSL related exceptions. These results need a deeper inspection after collection.

6.4 Post-Authentication Tests

Once we could verify that the access to the internet is granted, the post-authentication tests are executed.

6.4.1 Honey Traffic

In order to create authentic-looking honey traffic, we simulate user sessions. In the following it is described how the simulated session are constructed. Generally, these tests are considered as executed, once the login was successful, as this means that the login credentials have been sent over the network.
Differing banner, and any other unexpected event or response will be logged in a file as well and reported to the measurement server for debugging reasons and deeper inspection.

IMAP

Once the connection via IMAP has been established, the banner will be verified. It is expected to be:
** OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE AUTH=PLAIN] Welcome to the IMAP service. Ready.*

Then the app will simulate some typical tasks over IMAP:

- Login
- Select the inbox and try to fetch unread mail, if any
- Search for mails with subject *contract*
- Logout

SMTP

We only consider port 25 for this field study because of simplicity. The decoy data sent is not routed to any other MTA and the SMTP server does not deliver the decoy emails to a recipient. After the client has been able to connect via SMTP the expected banner is checked. The SMTP server is greeting with:
220 localhost Welcome to the SMTPServer. Have fun using this service!
Furthermore, it is also tested whether the `EHLO` response is as expected: `250-mail.asdf.com 250 AUTH LOGIN PLAIN`.
The app will:

- Send an EHLO command to the measurement server
- Login
- Send an email
- Logout

The email sent by the client will be logged by the SMTP server, but not delivered. For simplicity, we use the same recipient and the same message for every session. The sender will be adapted to the corresponding username.

FTP

Also for the FTP service, we compare the greeting banner first. The measurement server should greet with:
220-Welcome to our FTP Service. Be careful what you are doing and keep your credentials save. 220 (sic!)

Subsequently, the client application will:

- Login
- List files, and iterate directories
- Download the first file found
- Logout

HTTP

For the HTTP test, there is a simple web service configured.

The client will:

- Request `login.html` and compare if login site looks as expected
- Send login credentials
- Check if response is redirecting to the expected site `home.html`
- Follow the redirect and check if the username is part of the greeting on the website
- Logout

6.4.2 SSL Stripping

For the SSL stripping test, we test typical scenarios. First, the client requests a website, that should redirect to HTTPS. It is then verified that the content has not changed, e.g., links have been stripped.

The steps include:

- Request `home/index.html`
- Mark the HTTPS redirect as successful if it responded with a response 301 to `https://[...]/home/index.html`
- Follow the redirect (if any), and for HTTPS pin the chain certificate
- Compare the actual content with the expected one

Category	Websites
Social Network	http://facebook.com http://twitter.com
File Sharing	http://thepiratebay.org http://mp3skulls.top*
Streaming Service	http://youtube.com http://netflix.com http://hulu.com http://kinox.tv http://movie.to
Blogs	http://wikileaks.com http://medium.com http://wikipedia.com http://groups.google.com http://wordpress.com http://imgur.com
News	http://torrentfreak.com http://theguardian.com http://nytimes.com
Gambling	http://888holdingsplc.com http://bet-at-home.com
Pornography	http://pornhub.com

*replaced by http://mp3skulls.to as domain ran out

Table 6.3: Categories of tested websites

6.4.3 Blocked Websites

To test content that may be blocked by a network, we defined seven categories, and for each we test at least one representative website. These sites may be blocked for different reasons. While some may be marked as inappropriate (which is likely for pornography and gambling), others may be blocked as they are using too much bandwidth (for example streaming services). Some sites may be blocked for political or regulatory reasons, such as presumed illegal activities (e.g., file sharing).
If content filters are used to block the access, we can assume that websites with similar content will likely be blocked as well. The categories and websites are listed in table 6.3. Note that the *category* is a rough description for the general website content.

For the tests, each of the listed websites (table 6.3) is retrieved, and the following information is stored:

- Title of the website
- HTTP response code
- Website URL (or the URL we have been redirected to)

- IP address of the server the connection went to

These details are sent to the measurement server for a later inspection. We requested the websites using different countries of origin using VPN connections beforehand, and extracted a string or regular expression that would represent an expected title. For a first evaluation, we compare the actual title with the expected one and store a boolean indicating the website was reachable. In case of a positive match, we are sure the website was reachable. However, some titles of websites changed during the tests, or where localized. Thus, we need to further check all results that indicated that the website was not reachable or blocked, to eliminate false positives.

6.4.4 Blocking of Standard Ports

In order to identify ports that have been blocked by the network, we connect to selected ports over TCP and UDP. The complete listing of tested can be found in table 6.4. We only test a selected subset of ports. These ports are mostly standard ports for services that users may require to fulfill ordinary tasks. Some of those have already been explained in section 3.2. Furthermore, we also included the standard ports for the following protocols:

- Telnet [100] and remote user Telnet service [98]
- Simple File Transfer Protocol (SFTP) [66], a file transfer protocol that is simpler compared to FTP
- Internet Key Exchange (IKE) [58], provides functionalities to perform authentication and set up security associations
- SOCKS [65], a protocol used to exchange packets between a client and a server over a proxy server
- Internet Protocol Security (IPSec) [59], a protocol that provides security services for the IP layer
- Layer 2 Tunneling Protocol (L2TP) [71], a tunneling protocol
- Point-to-Point Tunneling Protocol (PPTP) [54], another tunneling protocol that is not defined as a standard, but has been developed by vendors
- Session Initiation Protocol (SIP) [105], commonly used for voice-over-IP

Additionally, we added ports that are traditionally used by BitTorrent for file sharing [7], and a list of ports that are suggested by the VPN provider *Mullvad* [81]. Furthermore, the test include one port that has been used by the remote access tool Cerberus, which is known as a hacking tool [70].

The test sends a message to the measurement server where a service replies with a predefined string: *Hello. Nice to meet you.*.
For each port and protocol tested, the result is marked as successful if we received the response as expected.
Additionally, we test if a TCP connection is accepted, for some ports that are already preoccupied on the measurement server. These ports include 21 (FTP), 22 (SSH), 25

(SMTP), and 143 (IMAP).

However, when interpreting the results we have to keep in mind that UDP is an unreliable protocol and packets may be lost. Therefore we can only be sure that the port is reachable, if we get a response. However, the opposite is not true, e.g., it is not possible to assume that the port is blocked, if we do not get a response due to the protocol's nature.

6.4.5 Support of IPv6

In order to test the capabilities and supported protocols of the networks, we try to access the URL `http://ipv6.google.com`. It is a website provided by Google that is only accessible over IPv6. If the website response is as expected, and we can verify that the address is indeed a IPv6 address, this test is marked as successful.

Port	Protocol used for Tests	Standard Service, Description
20	TCP	FTP
23	TCP	Telnet
25	UDP	SMTP
53	TCP, UDP	DNS
80	UDP	HTTP
107	TCP	Remote Telnet
110	TCP	POP3
115	TCP	SFTP
143	UDP	IMAP
220	TCP, UDP	IMAP version 3
443	UDP	HTTPS
465	TCP	SMTP over TLS
500	TCP, UDP	IKE
587	TCP	SMTP
989	TCP, UDP	FTPS
990	TCP, UDP	FTPS
993	TCP, UDP	POP3 over TLS
1080	TCP, UDP	SOCKS
1300	UDP	Suggested for VPN Mullvad
1301	UDP	Suggested for VPN Mullvad
1302	UDP	Suggested for VPN Mullvad
1194	TCP	OpenVPN
1195	UDP	Suggested for VPN Mullvad
1196	UDP	Suggested for VPN Mullvad
1197	UDP	Suggested for VPN Mullvad
1293	TCP	IPSec
1701	UDP	L2TP
1723	TCP, UDP	PPTP
5060	TCP, UDP	SIP
5061	TCP, UDP	SIP over TLS
5150	TCP	Cerberus RAT (malware)
6881	TCP, UDP	BitTorrent
6887	TCP, UDP	BitTorrent

Table 6.4: UDP and TCP ports tested

CHAPTER 7

Field Study

7.1 Preparation

Before we started the field study, we experimented and tested the app and services of the measurement server in small testing environments. For this we used other Android devices to create portable public Wi-Fi hotspots.
We also used such portal Wi-Fi hotspots to test whether the services of the measurement server are up and running as expected during the field study.

Furthermore, we tested automatic login functionality of the Honey Client app on real captive portals before we started the actual field study.

7.2 Timeline

On November 3, 2017 we started the first evaluation in the wild, in a supermarket. Not all tests had already been implemented. We continued introducing and adapting test cases along with the process of testing real world hotspots. Nevertheless, when we started with the first measurements we already spread honey traffic over IMAP and HTTP, tested blocked websites and ICMP tunneling capabilities.

On November 21, 2017 the server configuration was ready, and all honey services ran permanently. From this point on, we deployed the app onto several devices and started the measurements from public Wi-Fi networks in the real world.

For this thesis, we include testing results until March 24, 2018. E.g., we include all results that has been recorded by or sent to the measurement server until March 24. The services on the measurement server, however, kept running until mid of August 2018. This ensures to catch any attacker that has sniffed login credentials and tries to exploit those weeks or months later.

Name	SDK	Android OS	Brand	Model	Manufacturer
Sony Xperia tipo	15	4.0.3 (Ice Cream Sandwich MR1)	Sony	ST21i	Sony
LG P700 Optimus L7	15	4.0.3 (Ice Cream Sandwich MR1)	lge	LG-P700	LGE
Google Nexus 4	17	4.2 (Jelly Bean MR1)	Google	Nexus 4	LGE
Samsung Galaxy S4 Mini	19	4.4 (Kitkat)	Samsung	GT-I9195	Samsung
Google Nexus 4	21	5.0 (Lollipop)	Google	Nexus 4	LGE
Sony Xperia Z2	22	5.1 (Lollipop MR1)	Sony	D6503	Sony
Google Nexus 4	22	5.1 (Lollipop MR1)	Google	Nexus 4	LGE
OnePlus 2	23	6.0 (Marshmallow)	OnePlus	ONE A2003	OnePlus
Yotaphone 2	23	6.0 (Marshmallow)	YotaPhone	YD201	Yota Devices Limited
Samsung Galaxy S7 edge	24	7.0 (Nougat)	Samsung	SM-G935F	Samsung
Google Nexus 5X	25	7.1 (Nougat MR1)	Google	Nexus 5X	LGE
Google Nexus 6P	27	8.1 (Oreo MR1)	Google	Nexus 6P	Huawei

Table 7.1: Android test devices

7.3 Test Devices and Volunteers

Some volunteers participated in the field study and took devices on their daily routes or when traveling.
The users were only instructed to take the device along and to activate the app. It was up to the volunteers to decide whether they respond to manual login requests that may pop up.

We were in possession of several Android devices that we could use for the field study and we also provided those to the volunteers. Furthermore, some of the testers used their own devices for the measurements. We provided the app over the testing distribution tool *Beta by Crashlytics*[1]. This way we were able to push any updates of the app right away to the users.
Figure 7.1 contains a list devices that have been used for testing and measuring public Wi-Fi hotspots.

7.4 Locations

As the Wi-Fi hotspot testing requires physical presence, we mainly examined networks in and nearby Vienna. Thus, we visited shopping centers, train stations, airports, restaurants, etc.
However, we also had some volunteers that run the measurements during their trips to other cities and countries.

In the beginning of December we were able to run some tests on Wi-Fi networks around Florida, USA.

[1] http://try.crashlytics.com/beta/

At the end of December, three test devices measured hotspots during the 34th Chaos Communication Congress (34C3) [17] in Leipzig, Germany. The congress, organized by the Chaos Computer Club, is hosting a diverse audience including hackers and security specialists.
In January, we tested some Wi-Fi hotspots around London, mainly in various shops nearby Piccadilly Circus.
In addition, some devices collected data in other areas of Austria and other nearby European countries such as Germany, Slovakia, Czechia, Ukraine, and Italy.

7.5 Updates of Testing Scenarios and App Updates

The test for DNS tunneling was not properly implemented as we started the tests, so we only consider results starting from November 29, 2017.

Although, we constantly adapted and pushed new versions of the Android app to the users, other testing results will not be affected by those changes. The main improvements were concerned about usability and circumvention of the doze and stand-by mode.

7.6 Interruptions

Except for short-time maintenance windows, the server was up and running during the testing period. There were only two exceptions, when the database connection was interrupted.
Once on February 2, 2018, where we had to renew the server certificate. Restarting the services resulted in a lost database connection, that we only discovered some time later that day.
The second outage was on March 7, 2018. We are not aware of any special circumstances that caused this malfunctioning, we only know that the services could not access the database for about two hours. A restart of the services could fix the issue.
We could, however, recover the data that was transmitted to the server and were also able to relate most of the login attempts during these short interruptions, except for one. Details will be discussed in section 8.1.1.

CHAPTER 8

Results

The results discussed in the following include all data that have been received by the measurement server between November 3, 2017 (18:00 GMT+01:00) and March 24, 2018 (03:00 GMT+01:00). Note that testing devices may not have reported all data collected to the measurement server.
There are two conceivable scenarios: one, the internet connection was lost before uploading all results; and two, the app data on the device was deleted by the user.

When interpreting the results, we distinguish the tested networks by their SSID. This decision may be arguable as the network name can be freely selected and could also be used by people with malicious intend (e.g., for an evil twin attack). However, we think that it is a tangible approach and differences in collected data and measurements will reveal irregularities anyhow.
The internet connection for Wi-Fi networks in unreliable in general and may have been interrupted at any time. For the discussion of the results, we therefore only consider those Wi-Fi networks that were able to run the corresponding test.

The services on the measurement server kept running until mid of August, 2018.

8.1 Data Cleansing

For the evaluation we used a database dump from the original data on the measurement server that we imported into a local database for running queries. It contains all data that have been recorded until March 24, 2018 03:00 GMT+01:00.
As we used the measurement server and its services for testing various scenarios before the field study started, we only consider test results and access logs from November 3, 2017 18:00 GMT+01:00 onward.

In order to clean the data we filter SSID names that we have used to test whether the services of the measurement server are well functioning during the evaluation phase.

Furthermore, we insert missing data that was lost due to database interruptions but could be recovered from log files (see also section 8.1.1).

In addition, the GPS data requires some adaptions as the Wi-Fi based location tracking may deliver false or no results. We will discuss the related issues and cleaning approaches in more detail in section 8.1.2.

Moreover, we have to deal with test results that have been sent multiple times. The testing devices may send data more than once in case the connection had been lost before the data retrieval was acknowledged by the measurement server. However, each test result includes a time stamp of the test execution and a unique identifier for the device. As one device is only able to execute one test at a time, those attributes are sufficient to identify duplicates.

8.1.1 Database Interruptions

We had to deal with two database interruptions during the testing period. The RESTful service was not accepting data uploads for about 20 minutes on February 2, 2018. We could, however, reconstruct the two test results, which were sent to the measurement server during that time as we included verbose logging for any unexpected behavior. Moreover, there was a problem with the standalone Python app, running the FTP and SMTP service, which was not able to communicate with the database for about 7 hours. Consequently, any login attempt was denied. We identified seven login requests during that time, but lost the details about the used credentials the measurement server site. Nevertheless, as we logged details on the Android test devices, we were able match the missing pieces and could verify that all login attempts were initiated by testing devices.

The other outage on March 7, 2018, lasted for about two hours. During that time, only one test result was transported to the measurement server. Furthermore, we identified one failed login attempt over FTP at 8:51 GMT+01:00. The username used was *anonymous*. The entered password could not be recovered. However, this user does not exist and so we conclude that this relates to a scanning attack over the internet. There was only one login attempt we could not clarify. This attempt was over SMTP at 9:34 GMT+01:00 and we were not able to recover the used credentials.

As we could verify most of the missing data using the log files from the measurement server and from the Android testing devices, we created a script that inserts the lost data, e.g., the testing results and access attempts.

8.1.2 Location Mapping and Cleaning

As described in section 6.2.3 the Android testing devices rely on the *LocationManager* to request the last known location that is then stored along with the test result. The Wi-Fi based location estimation is not accurate, but delivers acceptable results in most cases.

However, we experienced two unexpected behaviors during the tests. Sometimes devices did not return any location and in other cases the reported GPS position was obviously wrong.

For all unavailable locations we assume that the LocationManager or the Google services were not working properly on the device. Furthermore, there are two explanations for the wrong GPS locations: One, the device was moved to a new location and has not received an update for the new position yet (i.e., still using the last known location). Second the BSSID of the network used had been mapped to another location before. This may happen when the physical location of a hotspot changes. For example think of a company that rents access points for large events to provide Wi-Fi hotspots. We experienced this behavior for the network *34c3-insecure*, which was the SSID of hotspots used for a conference in Germany. We assume that the access points used for this conference had been used in another area before and that Google mapped the BSSID of the devices to that other location. Also, the documentation of the Google Geolocation API describes that the Wi-Fi based location lookup requires only MAC addresses in order to determine an estimated position of a Wi-Fi network [29]. We can only assume that Android's LocationManager is using the same approach.

For test results with unknown locations, we checked if there was already an existing entry, reported by another device. We checked for results with the same BSSID and SSID combination and mapped the reported location. For the missing entries we used the Google Geolocation API to query for a location based on other Wi-Fi networks detected by the device. The API requires at least two MAC addresses of Wi-Fi nodes to estimate the location. As we collected data about any public Wi-Fi nearby, we were able to use the BSSID of other networks nearby the access point of interest. However, we only used the returned location if the estimated accuracy, returned by the API, was under 1,000 meters.

To interpret some of the test results we need to group the data by country. Therefore, we used Google's Reverse Geocoding API [30], which returns the country information for given GPS coordinates. Two locations did not return any result over the API. However, we manually searched for the GPS location over Google Maps to find the corresponding country. Those coordinates were 49.0825068,15.7586303 (Czechia), and 51.4716133,-0.4582375 (United Kingdom).

In order to eliminate known wrong locations, we sorted the test results by device (UUID) and time stamp tested, and then manually checked every entry that reported a different country before or after that test result. Considering the time stamp correlation, we identified 13 networks with a wrong location mapping and adapted the country information accordingly. Details about those corrected locations can be found in table 8.1.
The results may still contain wrong location information, but we were not able to reveal more, obvious mislabeled, networks.

For some other networks with multiple locations reported, we could not reveal a correct location either for various reasons. One example is the Wi-Fi network *FlixBus* that is provided on board by a bus travel company which operates between different countries. However, this example can be considered as special because the hotspot indeed physically moves between countries. Nevertheless, we did not alter any other locations.

8.2 General Data Collection

As already outlined in section 7.4, we mainly collected data within Austria and some European countries. The only location outside of Europe, was in Florida (USA).
Figure 8.1 depicts all European geographic locations that were reported by the devices. Note that this figure already contains corrected GPS data for the above mentioned networks.

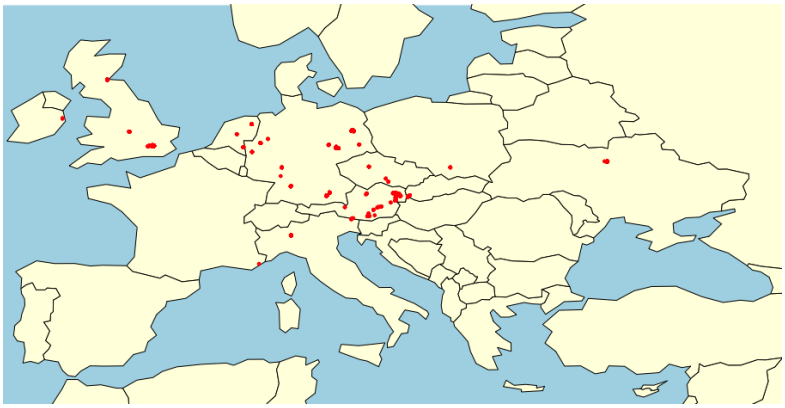


Figure 8.1: Locations of reported Wi-Fi tests

On the measurement server we received 7,574 scan results from 755 Wi-Fi networks (e.g., counting only distinct network names). Those figures suggest that we were able to connect to the Wi-Fi hotspots. However, the Wi-Fi connection is unstable in general.

Network	**Reported**	**Corrected**	**Explanation**
Air-VinziRast *e4:8d:8c:22:bf:98*	Austria, France	Austria	Timeline shows that only Austria can be correct
_Heathrow Wi-Fi *6c:f3:7f:61:a8:23*	Austria	United Kingdom	Airport, we assume that last known location was reported
*Donau Zentrum - Gratis WLAN *28:6f:7f:40:2d:5e*	Austria, United Kingdom	Austria	We assume that the last known location was reported
Lets Take a Walk *2c:5d:93:91:90:8e*	Austria, United Kingdom	United Kingdom	Probably wrong cached by LocationManager
WirelessViennaAirport *0c:85:25:c6:48:2f*	Germany	Austria	Airport, we assume that last known location was reported
Airport Free Wi-Fi *00:3a:9a:32:af:35*	Germany, Slovakia	Slovakia	Airport, we assume that last known location was reported
34c3-insecure, -Promenaden Hauptbahnhof, Leipziger, ShareBox - Share freely, Freifunk, Ratskeller-FreeWiFi, Pixel	United Kingdom, Poland, France, Netherlands, Ireland, Germany	Germany	LocationManager returned wrong location for *34c3-insecure*. Consequently, other networks had a wrong last known location. All devices that tested those networks, were only in Germany at that time.

Table 8.1: Corrected Locations

Consequently we counted 484 networks where at least the captive portal detection procedure started, which is the very first test case.

8.3 Captive Portals

We identified 390 distinct networks (i.e., unique SSID) that are using some form of captive portal. Furthermore, we counted 94 distinct networks that are accessible without the need for authentication. In other words they do not require a captive portal interaction and we can directly use the internet after connecting to the hotspot.
Out of the 390 distinct networks that are using a captive portal, we managed to authenticate to 126 using the automated or manual login methods. For another 140 networks out of the 390, the connection was lost before the login procedure could finish.
The remaining networks either asked for manual login or faced other issues during the login procedure (e.g., errors while loading web page, executing scripts, or the app was stopped by the user).

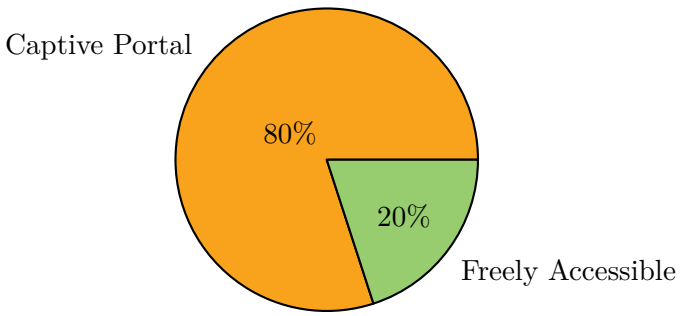


Figure 8.2: Wi-Fi hotspots and captive portals

When the field study started we also tried identifying captive portals that make use of password protection. In that case we intended to abort the testing procedure in an early stage. However, as we checked captive portals in the wild, we experienced some networks wrong classifications of *password protected*. Consequently, we stopped the login procedure because of the suspected password protection. To avoid those false positives and to expand the potential networks tested, we removed this constraint by the end of November.

8.3.1 Automated Captive Portal Login

The static login worked for 86 distinct networks, while the dynamic login succeeded for another 48 captive portals. Note that the SSID for static and dynamic login partially overlap as we revealed 14 networks that were at least once successful for the static and dynamic login procedure. This sums up to 120 successfully automated login interactions with distinct networks.
Although users were briefed regarding the use of the app, one tester reported that a login

was counted as *static login* as the tester accidentally used the built-in captive portal detection of Android to login to the captive portal *FreeWiFi@LeMeridien*. The above mentioned number have been corrected correspondingly. However, there may have been similar cases that have not been reported by the users.
Considering the 390 networks with detected captive portals, we managed to automatically connect to 31%. For another 36% we lost the connection to the Wi-Fi hotspot before the captive portal login attempt was completed.

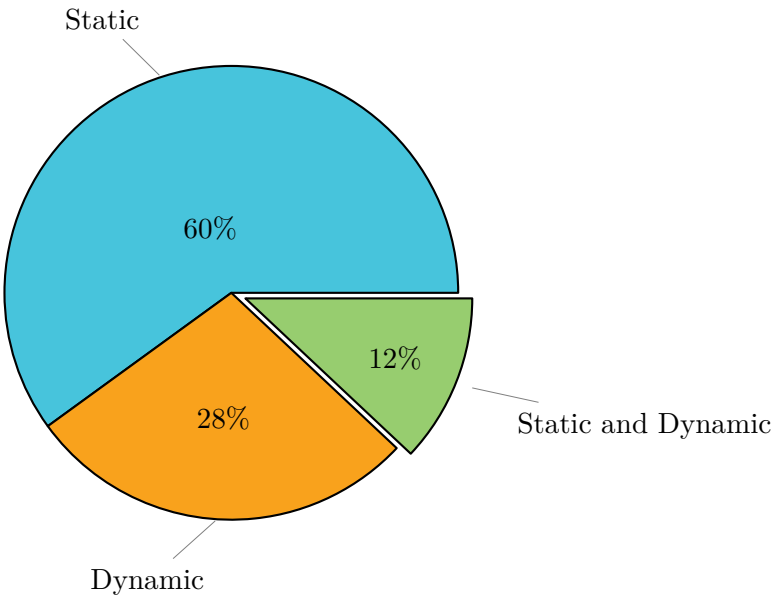


Figure 8.3: Automated and successful captive portal logins

The static login procedure makes up the majority of successful automated logins (figure 8.3). Furthermore, we managed to raise the number of successful authentication by interpreting JavaScript on captive portal landing pages, with the dynamic approach.

In addition, users successfully logged in to the captive portal manually for 11 distinct networks. Note that we discovered that 5 out of those have been also successfully connected with the static or dynamic approach at least once.

There may be some explanations why some networks worked with different login methods: For one, the connection could have been temporarily lost while trying one login method but reestablished when testing the next strategy. It is also possible that an updated app version caused this differences, or that the captive portal itself changed.

Device Identification

Moreover, we observed 79 networks (i.e., 63%) that remembered the devices at least for a short period of time. For those networks, we discovered a captive portal at least once. However, when testing the same hotspot subsequently, there was no captive portal interaction required. Note that we use a new instance of the *CookieManager* whenever we connect to Wi-Fi hotspot. Therefore, cookies will have no effect. Consequently, it seems like those captive portals store some information about the device (e.g., the MAC address) and keep them on a white list, at least for a predefined amount of time.

8.4 Pre-Authentication Test Results

8.4.1 ICMP Tunneling

We only considered test results that identified a captive portal, i.e., were not able to access the internet right away (390 networks). The results revealed that 95 distinct networks allowed pings to our measurement server before interaction with the captive portal. For 42 out of those we were not able to authenticate with the automated login methods. The manual login request was triggered for 10 of those.

The remaining 32 either lost the connection before the authentication procedure could finish, or experienced other errors.

The results show, that roughly 24% of the tested Wi-Fi networks with captive portals are likely to allow any ICMP traffic before authentication. Therefore, ICMP tunneling would be an option to circumvent the authentication, especially if the captive portal requires extensive personal information, tokens, passwords, or payment for login.
Note that we can only assume that a tunnel will work, as we did only check if an IP address outside of the network can be pinged. There may be additional checks in place to identify and block ICMP tunnels.

8.4.2 DNS Tunneling

We only consider DNS tunneling results starting from November 29, 2017 (see also section 7.5). Thus, we have 386 captive portal protected networks within this time frame. The tests revealed that 196 networks allowed DNS tunneling, meaning that the communication with our DNS server was initiated. For 111 out of these, we were not able to successfully authenticate to the captive portal during the login procedure.
52 out of the 111 triggered a manual login, but the request was either ignored or unsuccessful. For the remaining networks we experienced connection problems that eventually caused an interruption of the login procedure.

Consequently, we found that a higher number of captive portal protected networks is vulnerable for DNS tunneling techniques than for ICMP tunneling. In our field study, about 51% of the tested networks allowed communication to outside DNS servers before any interaction with the captive portal.

We further investigated the Dynamic Host Configuration Protocol (DHCP) info we collected from those 196 networks that were vulnerable for DNS tunneling. We found that for 74 networks at least one public DNS server was assigned. In addition, we identified 133 cases where the primary DNS server assigned had a private IP address.

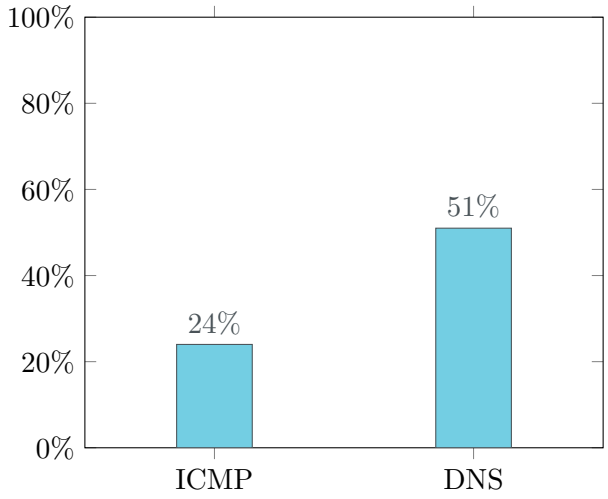


Figure 8.4: Recorded tunneling vulnerabilities for ICMP and DNS

8.4.3 HTTPS Handling

For the HTTPS redirection test before a successful authentication with a captive portal, we tested 384 networks.
We identified 31 captive portal protected networks that allowed the requested HTTPS connection before the login procedure. We changed the tested site starting from mid of December (see also section 6.3.3). Note that we did not have the opportunity to test all of the networks with both websites.
For those that we have though, we had some interesting findings: Two networks (*GUEST@VAPIANO* and *Matiki Free*) allowed the HTTPS connection to both of the tested websites, before the interaction with the captive portal. As we tested rather popular websites, we cannot assume that any HTTPS traffic is allowed. However, we know that at least some websites are white listed.
In addition, we identified three networks (*AKH-Hotspot*, *hhonors*, and *attwifi*) that accepted a secure connection to `google.com`, but refused connecting to `www.microsoft.com` over HTTPS: First two mentioned networks (*AKH-Hotspot* and *attwifi*) reported a `SSLHandshakeException`, complaining that the certificate

path could not be validated. This validation error indicates that the trust chain for the certificate cannot be build and that the keystore does not know anything about the certificate presented. Consequently, we assume that those networks used their own certificate and interfered the request probably by redirecting to the captive portal. The other network, *hhonors*, refused the connection due to a hostname verification problem. This means, that the certificate itself could be verified, but the hostname did not match the expected one (`www.microsoft.com`). Again, we assume that consequences from a redirection attempt to the captive portal.

48 more networks responded with a *hostname not verified* exception. Android devices running SKD version 22 also revealed more detailed information about the certificate. In table 8.3 some of those details are depicted. We selected certificate errors from networks we were able to connect to later on.
Moreover, there were 54 networks (excluding *AKH-Hotspot* and *attwifi*) identified that were causing an exception as the trust chain could not be verified (`CertPathValidatorException`). In addition, we even spotted three networks that caused an exception as the certificate presented was expired (see also table 8.2). This means that the network did not only interfere the connection, but also failed to renew their certificate.

Guest Wifi VE120	Certificate expired at Sun Dec 18 21:31:50 GMT+01:00 2016 (compared to Thu Feb 01 16:01:35 GMT+01:00 2018)
MPS - Cairoli	Certificate expired at Thu Dec 01 00:59:59 GMT+01:00 2016 (compared to Wed Jan 24 16:34:55 GMT+01:00 2018)
OEBB-GAST	Certificate expired at Fri Aug 11 06:40:59 GMT+02:00 2017 (compared to Tue Jan 23 11:35:58 GMT+01:00 2018)

Table 8.2: Samples of expired certificates

For the remaining *HTTPS before login* tests we were not able to successfully connect to the requested websites, but experienced various exceptions. Some of those may be fine, e.g. ,when the connection is refused. However, it is hard to distinguish which exceptions are caused by the captive portal (e.g., any HTTPS request is blocked before authentication), and which ones result from general connection errors. Consequently, we only discuss exceptions that are clearly related to certificate errors.

Figure 8.5 summarizes the witnessed behavior. Over a quarter of the tested networks interfered in some way with the HTTPS connection and consequently raised a certificate related exception. 8% of the tested networks allowed the HTTPS connection, so we conclude that those hotspots have at least some white listed websites configured. For the remaining 65% we were not able to establish a connection. However, we cannot distinguish connections that have been blocked by the captive portal from those ones that failed due to network problems.

BIKBOK_guest	sha1/tiKCXP4+woYxWKFB0yyGUuve964= DN: CN=*.netnordic.net,OU=PremiumSSL Wildcard,OU=IT,O=NETNORDIC BEDRIFTSKOMMUNIKASJON AS,2.5.4.18= #130430313032,STREET=Postboks 143 Sentrum,L=OSLO,ST=NO,2.5.4.17= #130430313032,C=NO subjectAltNames: [*.netnordic.net, netnordic.net]
BTOpenzone	sha1/YCYp9PgdoRNTskxXBbJOjKB8FUk= DN: CN=www.btwifi.com,OU=BT Wi-fi,O=British Telecommunications plc,L=London,ST=London,C=GB,2.5.4.5= #13083031383030303030,2.5.4.15= #131450726976617465204f7267616e697a6174696f6e,1.3.6.1.4.1.311.60.2.1.3= #13024742 subjectAltNames: [my.btopenzone.com, info.btopenzone.com, btopenzone.com, www.btopenzone.com, btwifi.co.uk, www.btwifi.co.uk, reg.btwifi.com, btwifi.com, info.btwifi.com, cdn.btwifi.com, my.btwifi.com, www.btwifi.com]
ESPRIT_free_WiFi	sha1/t4mmzVWCQdnnZ1qOjmTUaenfqeE= DN: CN=secured.esprit.com subjectAltNames: [secured.esprit.com]
FreeWiFi@LeMeridien	sha1/QvX+0o8Guxv/mYX1DWkpcKzbzTU= DN: CN=*.quadriga.com,OU=COMODO SSL Wildcard,OU=Hosted by Century web Design Ltd,OU=Domain Control Validated subjectAltNames: [*.quadriga.com, quadriga.com]
HSBC - Wifi	sha1/YCYp9PgdoRNTskxXBbJOjKB8FUk= DN: CN=www.btwifi.com,OU=BT Wi-fi,O=British Telecommunications plc,L=London,ST=London,C=GB,2.5.4.5= #13083031383030303030,2.5.4.15= #131450726976617465204f7267616e697a6174696f6e,1.3.6.1.4.1.311.60.2.1.3= #13024742 subjectAltNames: [my.btopenzone.com, info.btopenzone.com, btopenzone.com, www.btopenzone.com, btwifi.co.uk, www.btwifi.co.uk, reg.btwifi.com, btwifi.com, info.btwifi.com, cdn.btwifi.com, my.btwifi.com, www.btwifi.com]
MCO Internet	sha1/80y1XxWWNZG09quk4nKsnc5sp2I= DN: CN=*.goaa.org,O=Greater Orlando Aviation Authority,L=Orlando,ST=Florida,C=US subjectAltNames: [*.goaa.org, goaa.org]
TOWERFREEWIFI	sha1/MeCsgVxXpYALJTblKUND/FmO288= DN: CN=*.virginwifi.io subjectAltNames: [*.virginwifi.io, virginwifi.io]
VOR-Regio	sha1/44q1ycjcCfo0OnF7Hw8k7Df8n94= DN: CN=captive-portal.peplink.com,OU=PositiveSSL,OU=Domain Control Validated subjectAltNames: [captive-portal.peplink.com, www.captive-portal.peplink.com]
WIFIonICE	sha1/9unpOquGfLsoHnvvvcgL+a5nc/8= DN: CN=www.ombord.info,OU=Domain Control Validated subjectAltNames: [www.ombord.info]
_Free Airport WiFi	sha1/L+NSDDyfsJamUl8hSlFTNlRn9WA= DN: CN=*.berlin-airport.de,OU=Flughafen Berlin Brandenburg GmbH - Abt. FI,O=Flughafen Berlin Brandenburg GmbH,STREET=Flughafen Berlin Brandenburg,L=Berlin,ST=Berlin,2.5.4.17= #13053132353231,C=DE subjectAltNames: [*.berlin-airport.de, berlin-airport.de]
_Heathrow Wi-Fi	sha1/MeCsgVxXpYALJTblKUND/FmO288= DN: CN=*.virginwifi.io subjectAltNames: [*.virginwifi.io, virginwifi.io]
_InLinkUK Free Wi-Fi from BT	sha1/YCYp9PgdoRNTskxXBbJOjKB8FUk= DN: CN=www.btwifi.com,OU=BT Wi-fi,O=British Telecommunications plc,L=London,ST=London,C=GB,2.5.4.5= #13083031383030303030,2.5.4.15= #131450726976617465204f7267616e697a6174696f6e,1.3.6.1.4.1.311.60.2.1.3= #13024742 subjectAltNames: [my.btopenzone.com, info.btopenzone.com, btopenzone.com, www.btopenzone.com, btwifi.co.uk, www.btwifi.co.uk, reg.btwifi.com, btwifi.com, info.btwifi.com, cdn.btwifi.com, my.btwifi.com, www.btwifi.com]
stp-public	sha1/fQ8EQFocbZG2BgaWMY2zgI7IN3c= DN: CN=*.loop21.net subjectAltNames: [*.loop21.net, loop21.net]
viennasightseeing	sha1/fQ8EQFocbZG2BgaWMY2zgI7IN3c= DN: CN=*.loop21.net subjectAltNames: [*.loop21.net, loop21.net]

Table 8.3: Samples of HTTPS certificate errors

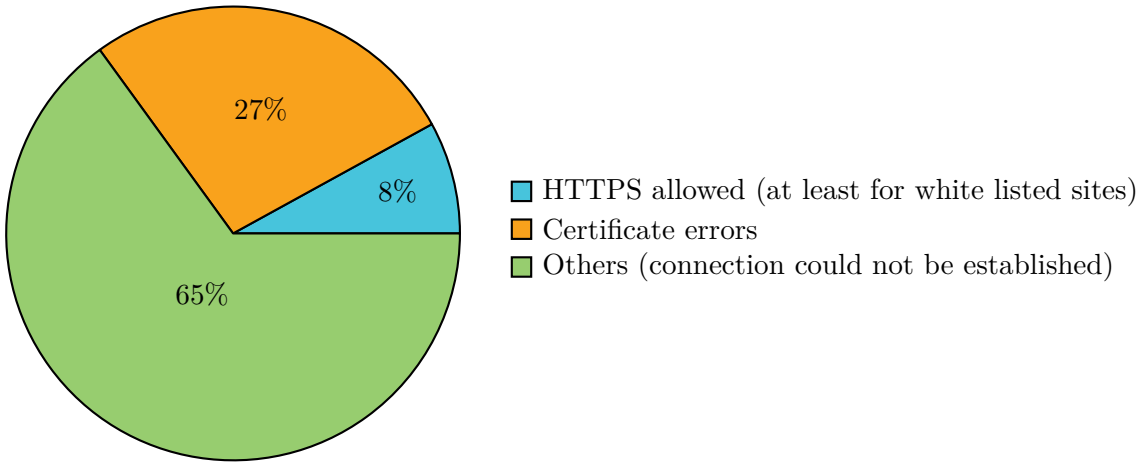


Figure 8.5: HTTPS handling of captive portals before successful authentication

8.5 Post-Authentication Test Results

8.5.1 Honey Traffic Spreading

To receive the amount of honey traffic spread, we counted the successful access logs on the measurement server side. This way we are able to include logins that have not been counted as successful by the Android device (e.g, if the response that the login was successful could not be received anymore as the connection was interrupted). There are even more situations, that explain why the data on the measurement server and the Android clients differ: Data collected within the app may not have been synchronized with the measurement server yet. Additionally, data may also be lost in case the app data had been deleted before all results were synchronized with the measurement server. Nevertheless, this credentials had been transmitted over the network.
We verified that all of these counted successful logins were indeed initiated by authorized devices. The access logs showed some login credentials that we could not match to any received test result. As we keep track of the UUID that requested login credentials, we were able to confirm that known test devices had been in possession of those login data. On the measurement server we counted 2,726 logins over FTP, 2,870 over IMAP, 2,673 over SMTP and 2,898 for the HTTP service. Note that the numbers are not equal for all services as some may have been blocked, and also the internet connection may have been lost at any time.

The logs did not indicate any repeated logins for the same login data. As the honey services kept running, we checked all records on the measurement server up to August 15, 2018. Therefore, we could not detect any evidence that an attacker has sniffed data as no user credentials were reused, even months after spreading the honey traffic.

8.5.2 Intercepted Services

We identified several Wi-Fi networks that reported a modified banner.
For the FTP traffic, we revealed two networks (*AirDigital* and *AirFree Unsecured*) that returned a modified FTP banner: `220 Zscaler/6.0: USER expected (Unix syntax).`
Both networks were tested on 2018-01-19 with only a few minutes in between. The login was unsuccessful in both cases. *Zscaler* is the name of an information security company which also provides a service for controlling FTP traffic[1]. Thus, it is likely that those networks are using this service.

Moreover, we discovered 9 distinct networks that intercepted the SMTP traffic, as the returned banner was different. The network *myhive Twin Towers Welcome*, *Telekom*, *BVG Wi-Fi*, *HSBC - Wifi*, *BTOpenzone*, and *WirelessHartlauer* all returned the same response: `220` followed by 65 star symbols `*`.
For all those networks, the SMTP test was at least once completed successfully. The most interesting networks of those ones may be *BVG Wi-Fi* as it is provided by the public transport service in Berlin[2].

Furthermore, we witnessed three networks that also returned a different response for the `EHLO` request. The login attempts for these network over SMTP were not successful, however. The tests that run within those networks were only a few minutes apart and located in London. Except for the timestamp and the IP address the following responses were identical for *FreeWiFi@LeMeridien*, *OmniAccess*, *GuestRoom@LeMeridien*:

```
[Banner]
220 hsia.quadriga.com ESMTP Exim 4.80.1 Thu, 01 Feb 2018 11:58:01 +0000


[EHLO]
250-hsia.quadriga.com Hello mail.asdf.com [192.168.54.205]
250-SIZE 52428800
250-8BITMIME
250-PIPELINING
250 HELP
```

These responses and the fact that the login was unsuccessful suggests that the SMTP traffic was redirected to an internal SMTP server within the connected network. We guess that these networks are managed by the service provider Quadriga[3].

8.5.3 SSL Stripping

The tests did not reveal any SSL stripping attempts.

[1] https://help.zscaler.com/zia/about-ftp-control
[2] https://www.bvg.de/de/Aktuell/BVG-Wi-Fi
[3] http://www.quadriga.com/solutions/iq-internet/

ADMIN	Access Blocked
Blocked URL	Check Point UserCheck
Content Blocked	Diese Seite wurde gesperrt!
Filtered	LIST OF COURT ORDERS
Message	Netzsperre - T-Mobile
Norton ConnectSafe	Requested Site Blocked
STOP!	Seite gesperrt
Seite kann nicht angezeigt werden	Sorry you can't access this page here
Sorry! Access Denied	UPC Kundeninfo
Web Page Blocked!	Website blocked
Website gesperrt	dm - Nutzungsbedingungen

Table 8.4: Titles used to indicate blocked content

8.5.4 Blocked Websites

For the categories of blocked websites, we experienced that Wi-Fi hotspots have different strategies for blocking unwanted content. We identified several networks that responded with a HTTP code `200 OK` and returned a static website explaining that the content is filtered. However, for the tests we only stored the returned title of the website.
Table 8.4 list such titles that we categorized as *blocked content* messages.

Consequently, we are able to interpret all responses that responded with a blocked content message, or with a response code `403 FORBIDDEN` as blocked. We counted 98 distinct networks that blocked at least one of the tested websites. In total numbers we experienced 3,141 denied requests.

We also registered various response codes within the 5xx range. In general those response codes indicate a server error. Table 8.5 lists the response codes for the tested websites that responded with such an error code. Some of those response codes are not standard, but are used by individual enterprises such as CloudFlare[4].

Response	**Message**	**Counted Networks**	**Individual Responses**
500	Internal Server Error	1	2
502	Bad Gateway	13	30
503	Service Unavailable	47	936
504	Gateway Timeout	4	12
520	Unknown Error*	7	91
521	Web Server is down*	1	1
522	Connection timed out*	8	27
523	Origin is unreachable*	1	1

*Cloudflare specific [116]

Table 8.5: 5xx response codes

[4] `https://cloudflare.com`

Especially the response code for `Service Unavailable` could may be used to indicate that the website is blocked. The amount of networks returning that code supports this suspicion. Therefore, we further investigate all those responses. To eliminate any obvious connection errors, we exclude all responses where we could match to a positive result. E.g., in case a network already tested this website successfully, meaning the website was reachable at least once, we ignored any 5xx response.
For the remaining responses we grouped the results by network, website and response code, and then counted the amount of times we faced the same response. We discovered one network that returned the response code `503` in 223 test scenarios for the website `pornhub.com`. Consequently, the likelihood that this website was blocked is high. Therefore, we introduce another category with *potentially blocked websites*. We also identified some responses, which responded with `200` and an empty title. In the end we only considered networks that returned the same response at least twice. This way we found another 45 potentially blocked website requests.
Note that we excluded two networks (*Vodafone Homespot*, *ShareBox - Share freely*) as the responses indicate that the internet was not accessible.
Table 8.6 gives an overview about the used HTTP response codes and the amount of networks that used those to indicate a blocked (or likely blocked) websites.

		Number of Networks	
Response	**Message**	**Blocked**	**Likely Blocked**
200	OK	81	6
403	Forbidden	19	-
502	Bad Gateway	-	1
503	Service Unavailable	-	16
504	Gateway Timeout	-	1
520	Unknown Error	-	4

Table 8.6: Number of networks and used response codes to indicate blocked content

Content Blocking Strategies

In addition, we searched for pattern regarding the blocking strategy. We stored the IP address of the server that responded to the website requests along with the result for the blocked website tests. Therefore, we can analyze those addresses. We used a reverse lookup API provided by ipinfo.io[5] to resolve the IP addresses to hostnames.
One content blocking strategy we discovered, is using a DNS service that denies access to blacklisted websites. We identified four networks that used OpenDNS[6] to block requests to `movie.to` and `pornhub.com`. Furthermore, another network used SafeDNS[7], which denied access to `888holdingsplc.com`, `bet-at-home.com`, `movie.to`, `pornhub.com`, `thepiratebay.org`, and `wikileaks.com`. Both DNS

[5] `https://ipinfo.io`
[6] `https://www.opendns.com`
[7] `https://www.safedns.com`

services responded with a response code `403` for each request.
Moreover, we experienced some ISP that actively denied access to some requested websites. Those include for example: A1 Telekom Austria, British Telecommunications, T-Systems Austria, or Tele2 Telecommunication. All of those responded with the code `200`.
We also found some networks that actively redirected to a static website announcing that the requested site was blocked, e.g., `unwirednetworks.net/content-blocked`. We identified some ISP which used this strategy as well, e.g., redirected to `www.t-mobile.at/netzsperre` and `www.upc.at/upc-kundeninfo/`.
In addition, one Wi-Fi hotspot used Norton ConnectSafe[8]. This allows the provider to configure different protection policies directly on the router. All requests that are handled by this router will be scanned and only be forwarded if the policy allows the request. We also found hints for a similar security gateway policy, provided by Check Point UserCheck [119].
Some networks seem to have various blocking strategies in place. This may indicate that the Wi-Fi hotspot provider uses a service that already blocks some content, but additionally denied access to some more websites. One example is the network *OEBB-station*, provided by the Austrian railway at some major stations in Austria. It blocked access to `pornhub.com` with a response code `403`, and denied access to `kinox.tv` and `movie.to` with a response code `200` and the message *Website gesperrt*.
For the remaining records we can only assume that service providers block sites by filtering specific content. We could experience some responses that contained the correct resolved IP address for the requested website, however, the content was not accessible anyhow.

Categories and filtered Content

The categories of websites were introduced to test whether networks filter for specific content. However, we experienced strong differences for single websites in some of the categories. This is likely due to our rather broad chosen classification. Table 8.7 provides more details about the specific websites and how many networks blocked the access to each one.

[8] `https://connectsafe.norton.com`

		Number of Networks		
Website	**Category**	**Blocked**	**Likely Blocked**	**Total**
movie.to	Streaming (other)	73	9	* **81**
kinox.tv	Streaming (other)	57	6	**63**
pornhub.com	Pornography	45	10	**55**
thepiratebay.org	File Sharing	49	5	**54**
bet-at-home.com	Gambling	16	7	**23**
888holdingsplc.com	Gambling	15	2	**17**
torrentfreak.com	News	8	2	**10**
netflix.com	Streaming (legal)	7	-	**7**
hulu.com	Streaming (legal)	6	-	**6**
mp3skulls.top	File Sharing	5	-	**5**
youtube.com	Streaming (legal)	4	1	**5**
imgur.com	Blogs	4	-	**4**
wikileaks.com	Blogs	1	3	**4**
facebook.com	Social Media	2	-	**2**
twitter.com	Social Media	2	-	**2**
medium.com	Blogs	1	-	**1**
wikipedia.com	Blogs	1	-	**1**
wordpress.com	Blogs	1	-	**1**
groups.google.com	Blogs	-	-	**0**
nytimes.com	News	-	-	**0**
theguardian.com	News	-	-	**0**

*One network (*Moebelix Free Wifi*) showed up for both categories

Table 8.7: Websites blocked by Wi-Fi hotspots

It can be observed, that two of the websites from the category *streaming* are leading the list of most blocked websites. As already discussed in section 5.2.2, those sites are known to include copyright protected material. Furthermore, we witness that perfectly legal streaming services are occasionally filtered as well. Consequently, we will introduce a new category to highlight the differences between those: For the following results, we use the category *legal streaming* to describe the services of YouTube, Netflix, and Hulu.
Furthermore, the table shows that only one website of category *news* has been blocked. However, we keep the other categories as defined in the beginning as we do not have enough data to identify other outliers.

Networks and Content Filter

For the following interpretations we take the total number into account, e.g., using the sum of blocked and likely blocked websites.
As the internet connection may have been lost during the tests at any time, the number of test results for each website differ. However, we counted 216 networks, that tested at least one website positively or negatively (e.g., we could verify that the website was filtered or allowed). Having a look at the positive results, we registered 215 networks

that successfully requested at least one of the tested websites. Out of those, 54 networks allowed access to all of the tested websites.

Figure 8.6 highlights the amount of networks that blocked or allowed categories of the tested websites. Note that the filtered and accessible networks may overlap, as we tested several websites for each category. We also experienced some networks that changed their filtering rules during our testing period, and we will discuss this in more detail later on. The figure indicate that the Wi-Fi hotspots tested only block a subset of websites in general. The categories with the most restrictions include streaming services, followed by pornography and file sharing.

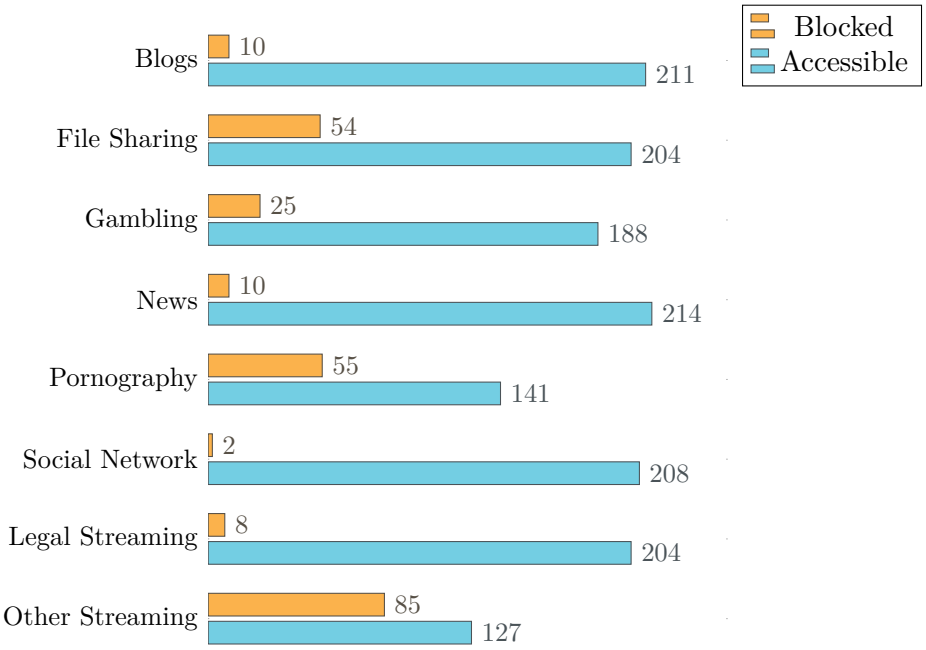


Figure 8.6: Absolute numbers of networks with content filter vs. accessible website categories

Differences between Countries

The blocked website tests were executed in seven different countries. Naturally, we tested the most individual networks (149) in Austria (AUT). Furthermore, we tested 30 networks in Great Britain (GBR), 24 in Germany (DEU), 13 within the United States of America (USA), and 9 in Ukraine (UKR). For Slovakia and Czechia we only have 3 respectively 2 networks tested and therefore exclude those from the comparison between countries. Note that we included the total amount of clearly and potentially blocked websites. Please keep in mind that although, we cleaned the data for known wrong locations beforehand (see section 8.1.2), this data may still contain some other wrong

location mappings.

Due to the small number of tested networks, the results in figure 8.7 should be interpreted with care. However, we can make some interesting observations. In Great Britain more than half of the tested networks blocked the access to pornographic websites. Moreover, the networks tested in the United States blocked legal streaming services more often than *Other Streaming Services*. However, we should keep the absolute numbers in mind (2 blocked legal streaming vs. 1 blocked other streaming). Nevertheless, the selected websites for the category *Other Streaming* may be more popular in Europe and therefore not part of standard content filter within the US. Interestingly, Wi-Fi hotspots in Germany allowed access to those streaming sites more often than other European countries.
For Ukraine we only found blocked websites in the categories *Gambling* and *Other Streaming*.

In addition, we identified some networks with the same SSID that operated in different countries. Three of those were identified Wi-Fi hotspots provided by public transport services that operate between different countries (*OEBB*, *OEBB-station*, and *FlixBus*). For another three SSIDs we recorded two countries of origin: Austria and Great Britain. Those three (*DESIGUAL_HAPPY_WIFI*, *H&M Free WiFi*, *WEEKDAY Free WiFi*) are provided by different fashion store chains. We only could detect differences within one of those networks: the hotspot provided by *Desigual* did not allow the connection to `movie.to` in Great Britain, while in Austria we were able to reach this website.

Deviations of Responses of Hotspots with the same SSID

We are interested whether the responses of networks regarding blocked or accessible networks changed over time. Table 8.8 gives an overview about collected responses from networks that revealed differences. We found some networks that denied access to websites that were ordered to be blocked by court [56], [26], [25]. Moreover, we witnessed that chain stores may have different filter strategies in place. In addition, we recognized that the website TorrentFreak was blocked by one network (*Wiener Linien Free WiFi*, provided by public transport service in Vienna), that later on allowed the access. We assume that the content filter for TorrentFreak was added accidentally, as there are no court orders or obvious reasons why this website would be blocked in Austria.
Furthermore, we experienced that *Freewave*[9], a hotspot service provider popular in Austria, returned different responses for the blocked website tests. Therefore, we assume that their customers are able to set individual content filters. We even witnessed different blocking strategies: some returned a response code `200 OK` with the message *Diese Seite wurde gesperrt!*, while we also found responses with `503 Service Unavailable`.

[9] `https://www.freewave.at`

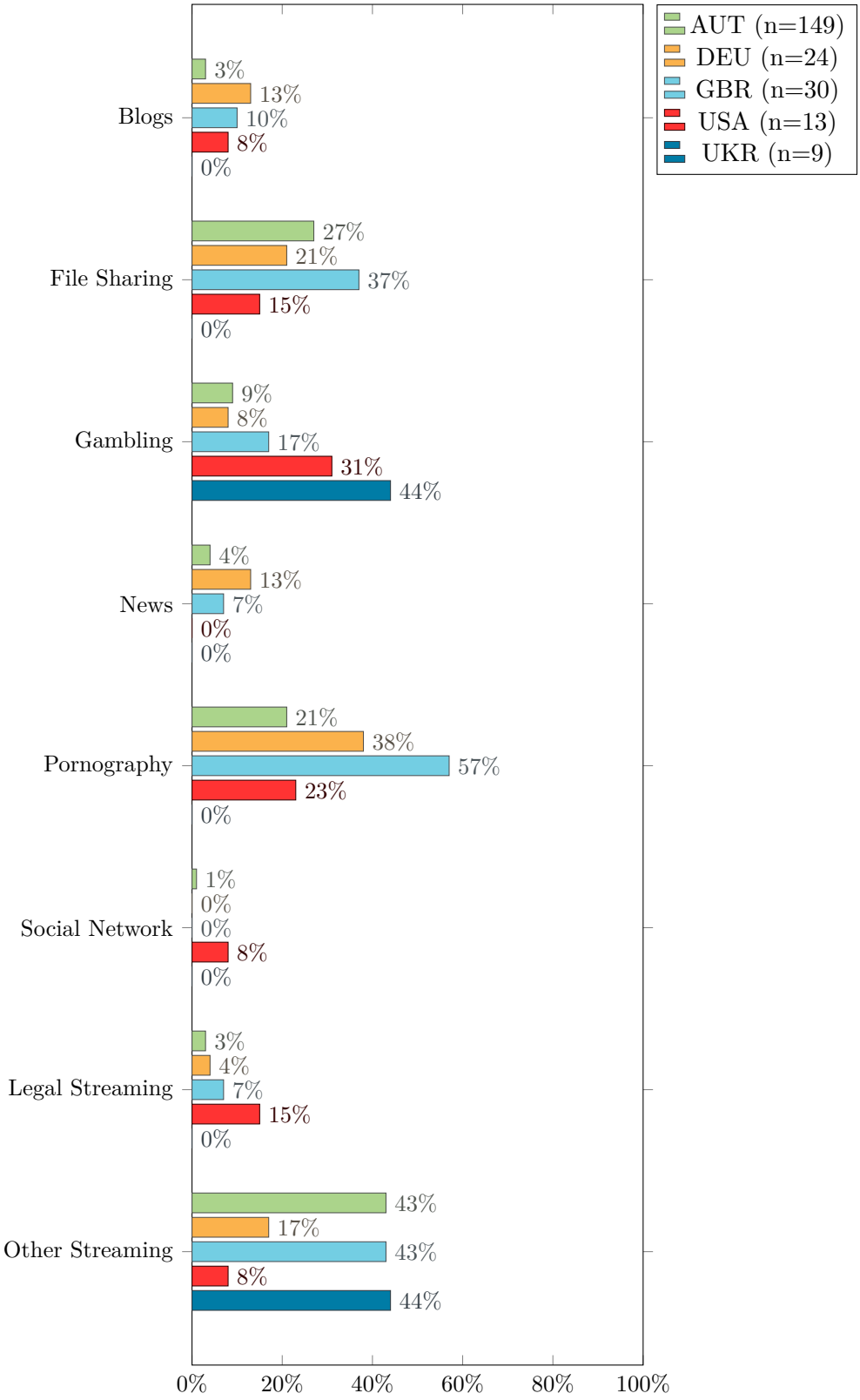


Figure 8.7: Networks that blocked categories of websites (in percent), grouped by countries

Another interesting finding are the tests recorded by the network *_Free Airport WiFi*, likely a hotspot name for airports. We could run tests on this network two times in different locations in Berlin, Germany. The first run was on January 1, 2018 and the tests showed that several websites were blocked. The second one, on February 8, 2018 allowed access to all of the prior blocked websites. Moreover, we detected that the DNS tunnel test was successful on the second date as well, when it was not for the first run. Although this behavior raises some suspicion, we do not have any evidence that we were connected to a rogue hotspot the second time. The location data for this second run suggests that the device was at the airport Berlin-Tegel.
Furthermore, we found that chain stores even within the same country may have different content filter in place. We found two more hotspots, one provided by a supermarket chain (*FreeWifi@Interspar*), the other one by a furniture store chain (*Moebelix Free Wifi*), that each showed deviations for some of the tested websites.

SSID	**Website**	**Accessible**	**Blocked**	**Remarks**
BIKBOK_guest	thepiratebay.org	2018-01-24	2018-02-02	
FreeWifi@Interspar	mp3skulls.top	-	-	Two different stores: one allowed access, one blocked
Freewave	movie.to kinox.tv	-	-	Freewave is a hotspot provider in Austria; results indicate that content filter can be set individually
LAN1	movie.to kinox.tv	2017-11-28	2017-12-12	
Moebelix Free Wifi	kinox.tv thepiratebay.org	-	-	Two different stores; one allowed access, one blocked
OEBB-station	thepiratebay.org	2018-01-18	2018-01-19	
Telekom	pornhub.com	-	-	Likely different operators: partially blocked
WLAN@Hofer	thepiratebay.org	2018-01-17	2018-01-22	
Wiener Linien Free WiFi	kinox.tv	2017-12-10	2017-12-12	
Wiener Linien Free WiFi	torrentfreak.com	2017-12-12	2017-12-10	TorrentFreak was blocked on this network for some time, but later on accessible.
_Free Airport WiFi	movie.to kinox.tv medium.com mp3skulls.top thepiratebay.org torrentfreak.com	2018-02-08	2018-01-01	Measurements on two days: first time websites were blocked; second time accessible
gateway	thepiratebay.org	2018-01-15	2018-02-27	

Table 8.8: Changes in received responses regarding blocked websites

8.5.5 Open and Blocked Ports

In order to determine the open respectively ports we group the networks by names (i.e., SSID). This way we count a successful connection only once per network. Although there may be some networks with the same name but different configurations, the potential

error rate should be low. In the previous section, we have already outlined the networks that had the same SSID but showed different configurations.
The port tests take the most time and devices lost the connection to some networks before all tests could finish. Therefore, we calculate the percentage for each port individually depending on the amount of responses we recorded. Note, that the device sometimes could take some time to realize that the connection was already lost, while the tests were still running. Thus, there may be false negatives in the results, e.g., the port was wrongly classified as being blocked.
As already outlined, the UDP results could additionally contain some false negatives due to the unreliable nature of the protocol.
Consequently, we focus on the successful connections we recorded for each port. Figure 8.8 gives an overview about the amount of networks that were able to connect to the tested ports over TCP respectively UDP. Note that not all ports have been tested for both protocols. There is one noticable port: port 53 (standard port for DNS) showed less response and could only be reached in 70% over TCP and 60% over UDP. This may be an attempt to block DNS tunneling (see also section 8.4.2). However, we were not able to find any correlation between the observations of potentially blocked DNS ports and the results found for DNS tunneling. Overall, there are no additional particular outliers. For the TCP protocol we only observe that the standard ports for SMTP (port 25), and SIP (port 5060) are slightly less available. SMTP traffic could be entirely blocked, assumingly to prevent spam misuse. Similar reasons could explain the blocked SIP port.

8.5.6 IPv6 Support

We only identified 5 networks (out of 256 that we successfully connected to) that support IPv6. This makes about 2% of the tested networks. This low number may be explained as we tested free Wi-Fi hotspots that likely want to keep the operating costs low. Therefore the support of IPv6 protocol along with IPv4 may not have a high priority.

8.6 Other Observations

8.6.1 False Positive SSL Stripping Records

As state in section 8.5.3 we did not reveal any SSL stripping attempt. However, in our database we found two networks that were marked as (false) positive. One of those *Vodafone Homespot* was sending a response code `302` when requesting the tested website. We did not follow the redirect but recorded the same response code when requesting the HTTP service, where no redirection to HTTPS was expected. Therefore, we conclude that the hotspot lost the information about the authenticated device and it was an attempt to redirect to the captive portal landing page.
The other one, *ShareBox - Share freely*, was a hotspot we tested during the Chaos Communication Congress in Leipzig, Germany. The hotspot was designed for "file-sharing and chatting", as stated on the landing page. For every request we sent, we received a response code `200`, containing a meta refresh redirect to the aforementioned landing

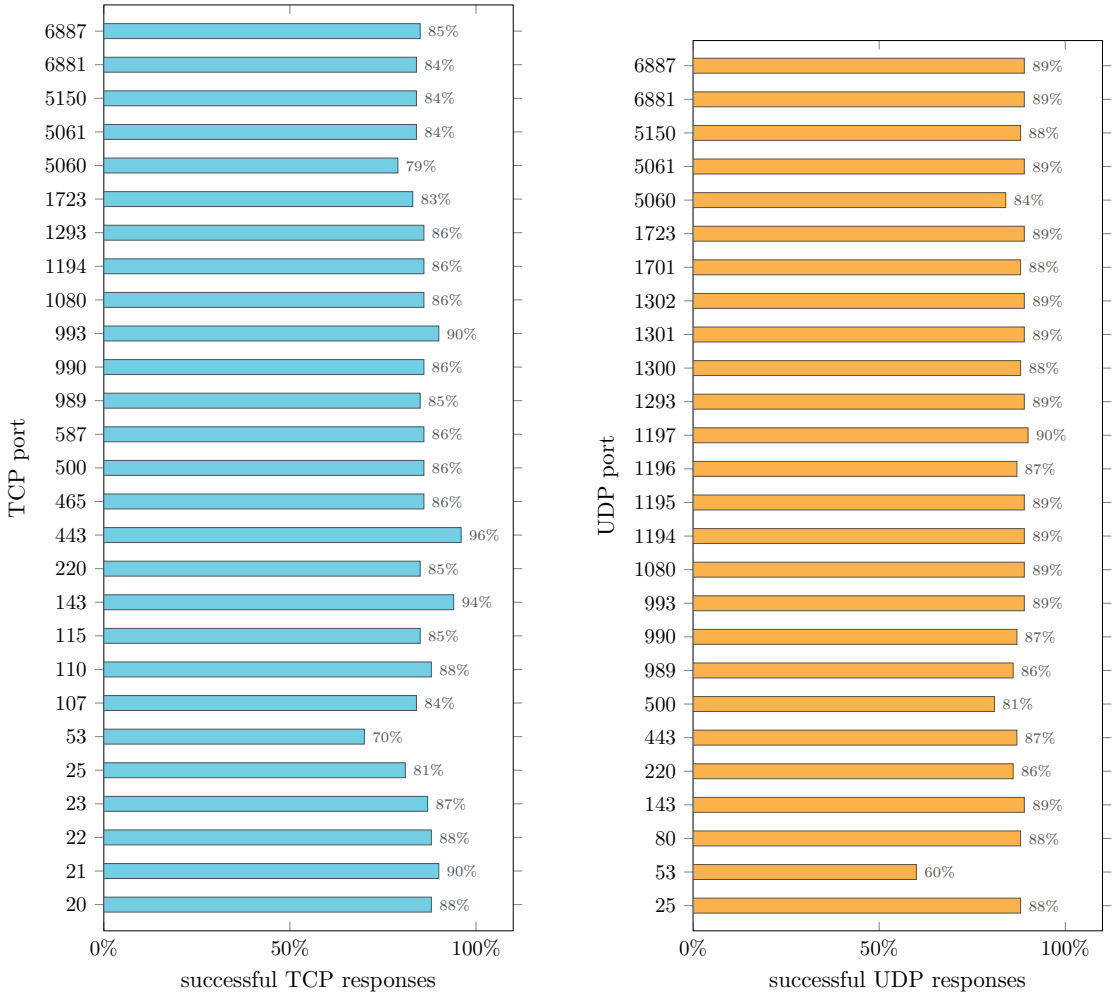


Figure 8.8: Amount of networks that successfully responded to tested ports

page. However, the hotspot also stated that it was not providing access to the internet. We are not sure whether this was true, as our captive portal detection had reported a successful connection to the internet.

8.6.2 Recorded Login Attempts

Naturally, the honey services deployed also attracted attackers that are generally scanning for open ports on any server that is reachable over the internet.
During the time frame of the active honey spreading we counted over 18,800 unsuccessful login attempts for the SMTP service. Moreover, we had over 330 attempts over FTP that failed as well. These figures increased sharply when we checked the unsuccessful logins later in August 2018 again. We counted over 256,900 failed logins for the SMTP service and 1,500 for FTP. There were even two unsuccessful login attempts for the

HTTP service. We assume those ones were conducted manually. Both times the same login credentials were used: username `admin`, password `admin`.
Note that the unsuccessful logins only consider attempts that provided credentials, and used supported methods. Our logs also showed various attempts of using unsupported commands, like `AUTH TLS`.
The UDP server logs also revealed that SIP commands were sent regularly to the measurement server.

CHAPTER 9

Discussion

In the field study we collected information about 755 Wi-Fi networks. However, due to connection issues not all tests ran on each network. We identified 484 networks that at least started the captive portal detection. Consequently, we group the results by test case and only consider networks that executed the particular scenario.

We found that 80% of tested networks (= 390 hotspots) redirected to a captive portal landing page and required some user input. The automated login procedure was successful for 31%. Moreover, the login was interrupted for 36% of the captive portal protected networks meaning the connection to the hotspot was lost.

For the pre-authentication tests we only consider captive portal protected networks. We revealed that 24% were vulnerable to ICMP tunneling. For 51% we successfully created a DNS tunnel. The number of networks that allowed tunneling is even higher than the networks we were able to authenticate to later on. We identified 52 networks that could not automatically login to the captive portal, but successfully sent data over a DNS tunnel. In addition, we had the same finding for 10 networks with ICMP tunnels.
Therefore, using a tunnel may also be another way to circumvent revealing personal information in order to login when a captive portal is in place. Note, that we did not further assess which of the vulnerable networks require a secret token, or even payment in order to access the internet.

Furthermore, the tests revealed that 27% of networks interfered the HTTPS connection before successfully authenticating to the captive portal. We even spotted three expired certificates. This MitM scenario is worrisome and results into browser errors. This in turn forces users to accept an untrusted certificate in order to use the Wi-Fi hotspot. We assume that those networks heavily rely on built-in captive portal detection, and

thus do not feel the necessity for appropriate HTTPS handling.
In addition we found that 8% of tested hotspots allowed a secure connection before login. The remaining networks did refuse the HTTPS connection, which is also an acceptable strategy.

The major part of the field study was concerned about data interception in public Wi-Fi networks. We simulated approximately 2,700 user sessions over each of the selected, unencrypted protocols (FTP, SMTP, IMAP, and HTTP). Even though, the measurement server and its services kept responsive for months after the field study, we were not able to detect any login attempts by sniffers. Therefore, we have no evidence of passive network attacks within the tested Wi-Fi hotspots.
However, we could reveal some hotspots, that intercepted services over FTP and SMTP. As we checked for banner modifications returned by the services, we found that 9 hotspots intercepted the SMTP traffic. Their intention may be scanning for spam that is sent over the network. Nevertheless, we can assume that the content of the emails is examined by those providers. Moreover, we identified two hotspots that intercepted FTP traffic, and another three for SMTP for which all login requests failed. Consequently, it seems like the connections were redirected to and handled by an internal network server, even though it is unclear why.
Those interceptions were, however, the only ones identified, and we did not encounter any SSL stripping attack.

Another research question of the field study was about access restriction to certain websites or services. All in all, we counted 98 networks that at least denied access to one of the tested websites. While some responded with a HTTP code `200 OK` and a short message stating the content is blocked, others used the response code `403 FORBIDDEN`, and some even made use of a `5xx` response.
Furthermore, we experienced different content filtering strategies. We revealed five networks that used OpenDNS or SafeDNS. In addition, some ISP redirected to a static website stating that the access to the requested website was denied.
The most frequent blocked category was *Streaming*, with the sites `movie.to` and `kinox.to`, followed by *Pornography* and *File Sharing*.
When assessing the blocked websites separately for each country, we found that Great Britain peaked in blocking pornographic websites: 57% of tested networks denied the access. We also witnessed that the top category of potentially illegal streaming services was rarely blocked in the USA (8%). This may be explained as the selected websites are particularly popular in Europe. Although, Germany filtered this category less (17%) than other European categories (~43%).
In addition, we collected different responses for some Wi-Fi hotspots with the same SSID in different test runs, e.g., requested websites were blocked and accessible. While some networks simply adapted the content filter over time, others showed different responses timeline independent. We witnessed this behavior for some store chains. This could

indicate that the stores independently manage the Wi-Fi hotspots. We also have reason to believe that the hotspot provider *Freewave* allows customers to set individual content filter. Moreover, we recorded measurements from two different airports in Berlin, sharing the same SSID. The first one revealed several blocked websites, but the second one allowed access to all prior blocked sites. Although, this behavior sounds suspicious, we could not find any evidence that the second Wi-Fi hotspot was spoofed.
The results for blocked TCP and UDP ports indicate that the standard port for DNS (port 53) is the most restricted one. Interestingly, we could not find any correlation between the blocked port and DNS tunnel vulnerabilities.

The results show only a snapshot of the real world, as we only tested a limited amount of networks. We have no evidence that any sniffer was monitoring one of the tested Wi-Fi hotspots. However, it is also possible that the simulated user sessions were not authentic enough. E.g., we used different user credentials every ten minutes when spreading honey traffic within the same network. In addition, all services were hosted by the same server, which may also raised suspicion to traffic sniffers.
Future research could focus on spreading more believable honey traffic, and inclusion of other protocols, e.g., SIP. Furthermore, spreading the services onto different servers would make the honey traffic services more stealthy. In addition, popular services could be added, that attract more attention like Gmail credentials. A longer and more widespread study could include countries worldwide.

CHAPTER 10

Conclusion

In our field study we assessed real-life public Wi-Fi hotspot vulnerabilities, such as traffic sniffing, interception, and tampering, utilizing an Android app.
We designed the app in a way that it would run tests on public hotspots without any user input required. Thus, when the app is active, it would automatically search for and connect to public Wi-Fi access points, spread honey traffic, run additional tests, and then send the results to the measurement server, before disconnecting from the Wi-Fi hotspot. The biggest challenge was the automated interaction with captive-portal-protected hotspots, as landing pages are not standardized. Therefore, we included two different automated methods: one that analyzed the landing page statically, e.g., sending forms with dummy data, or following redirects and links. The other one was a dynamic approach, basically with the same capabilities. However, it additionally interpreted and executed JavaScript to interact with the captive portal landing page. If none of those methods succeeded, the app would trigger a notification asking the user to manually perform the login, which was an optional choice.
We found that 80% of tested networks used a captive portal landing page and at least required some confirmation regarding the terms of Wi-Fi usage. The automatic login procedure succeeded for nearly every third captive-portal-protected Wi-Fi hotspot.

Our measurement server also provided the honey services for SMTP, IMAP, FTP, and HTTP. On the server we created unique user credentials for those services. Additionally, we populated the user accounts with random emails or files, in order to generate believable content for the simulated user sessions. The valid user credentials were distributed to the app over an encrypted connection so that the app could use those to login to the services, and to create the honey traffic.

During the time period of over four months, we run tests on public Wi-Fi hotspots within seven countries. We spread about 2,700 different credentials for each of the selected

services over the Wi-Fi networks. The honey services on the bait server were accessible for about another five months, but we were not able to reveal any network sniffer. This does not necessarily conclude that there are no passive attacks, as our test set may have been too small, or the generated honey traffic was not authentic enough and classifiable as bait. Regarding active attack scenarios, we identified nine public Wi-Fi hotspots that intercepted SMTP traffic. Those networks may actively scan for spam misuse. However, we have no information about how the data is processed, or whether it is even stored for additional analysis. Further five networks redirected FTP or SMTP traffic to an internal service, but the request was not forwarded to our bait server, which may indicate a misconfiguration. No other active attacks were observed during the field study.

Apart from attacks, we addressed specific network restrictions. Over a quarter of the captive-portal-protected networks triggered certificate errors when requesting a secure website connection, before a successful authentication to the captive portal. This interception of HTTPS may endanger genuine users, as it trains to accept unknown certificates. Especially, if there is no built-in captive portal detection on the user's OS or browser.
Additionally, we tested captive portal circumvention techniques and discovered that about a quarter of networks were vulnerable regarding ICMP tunneling, and about 50% for DNS tunneling. These methods may not only be used to bypass payments, but also to avoid providing detailed personal data.
Moreover, we revealed different content blocking strategies of public Wi-Fi hotspots. We found that the most filtered website categories include streaming services with potentially pirated content, and pornography. There were no consistent blocking strategies across countries, e.g., we did not reveal one country that consistently blocked the same website. Therefore, we assume that content filter are mainly depending on the Wi-Fi provider itself. Further studies could continue the assessment of filtered content within public Wi-Fi hotspots all around the world.

The field study provides a glimpse of the real world. Even though we did not reveal network sniffers, we could observe some worrisome behavior of network providers, including interception of HTTPS connections and SMTP traffic, and various content blocking strategies.

List of Figures

List of Tables

Acronyms

C&C Command and Control. 12

DNS Domain Name System. 7, 12, 13, 45

DOM Document Object Model. 37

FTP File Transfer Protocol. 2, 8, 11, 17, 23, 24, 45

FTPS File Transfer Protocol over TLS. 9, 45

GPS Global Positioning System. 19

HSTS HTTP Strict Transport Security. 19

HTTP Hypertext Text Transfer Protocol. 2, 7, 8, 10, 11, 16, 23, 24, 45

HTTPS Hypertext Transfer Protocol Secure. 2, 7, 8, 10, 17, 23, 45

ICMP Internet Control Message Protocol. 7, 12

IKE Internet Key Exchange. 44, 45

IMAP Internet Message Access Protocol. 2, 9, 11, 17, 23, 24, 45

IMAPS Internet Message Access Protocol over TLS. 9, 17

IP Internet Protocol. 12, 16

IPSec Internet Protocol Security. 44, 45

IPv4 Internet Protocol Version 4. 13, 16

IPv6 Internet Protocol Version 6. 13, 16, 24, 46

ISP Internet Service Provider. 42

L2TP Layer 2 Tunneling Protocol. 44, 45

MitM Man-in-the-Middle. 2, 7, 10, 11, 17, 22–24

MTA Mail Transferring Agents. 9

NAT Network Address Resolution. 9, 34

OS Operating System. 7, 32

POP3 Post Office Protocol, Version 3. 9, 11, 45

PPTP Point-to-Point Tunneling Protocol. 44, 45

RAT Remote Access Tool. 45

RR Resource Records. 13

SFTP Simple File Transfer Protocol. 44, 45

SIP Session Initiation Protocol. 24, 44, 45

SMTP Simple Mail Transfer Protocol. 2, 3, 9, 11, 23, 24, 45

SPF Sender Policy Framework. 14

SSH Secure Shell. 17, 24, 27, 44

SSID Service Set Identfier. 22

SSL Secure Socket Layer. 8

TCP Transmission Control Protocol. 8, 11, 12, 14, 18, 24

TLS Transfer Layer Security. 8–10

UDP User Datagram Protocol. 11, 12, 14, 24

UI User Interface. 33

VPN Virtual Private Network. 2, 24

XMPP Extensible Messaging and Presence Protocol. 17

Bibliography

[1] How to automatically login to captive portals on OS X?, 2012. URL https://apple.stackexchange.com/questions/45418/how-to-automatically-login-to-captive-portals-on-os-x. Accessed: 2018-01-03.

[2] How does WiFi in android detect if the device has to sign in or not?, 2015. URL https://android.stackexchange.com/questions/123129/how-does-wifi-in-android-detect-if-the-device-has-to-sign-in-or-not. Accessed: 2017-12-14.

[3] Turn off captive portal, 2017. URL https://support.mozilla.org/en-US/questions/1157121. Accessed: 2018-01-03.

[4] European Consumer Centre Austria. Liablity for unsecured Wi-Fi, 2016. URL http://europakonsument.at/en/page/liability-unsecured-wi-fi. Accessed: 2018-02-21.

[5] R. Barnes, M. Thomson, A. Pironti, and A. Langley. Deprecating Secure Sockets Layer Version 3.0. RFC 7568 (Proposed Standard), June 2015. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7568.txt.

[6] BBC. Turkey Twitter ban: Constitutional court rules illegal, 2014. URL http://www.bbc.com/news/world-europe-26849941. Accessed: 2018-02-27.

[7] BitTorrent.org. The BitTorrent Protocol Specification. URL http://www.bittorrent.org/beps/bep_0003.html. Accessed: 2018-03-03.

[8] Turkey Blocks. Facebook, Twitter, YouTube and WhatsApp shutdown in Turkey, 2016. URL https://turkeyblocks.org/2016/11/04/social-media-shutdown-turkey/. Accessed: 2018-02-27.

[9] Brian M Bowen, Vasileios P Kemerlis, Pratap Prabhu, Angelos D Keromytis, and Salvatore J Stolfo. Automating the Injection of Believable Decoys to Detect Snooping. In *Proceedings of the third ACM conference on Wireless network security*, pages 81–86. ACM, 2010.

[10] Danny Bradbury. Why do people ignore security warnings when browsing the web?, 2015. URL https://www.theguardian.com/technology/2015/feb/24/people-ignore-security-warnings-browsing-web. Accessed: 2017-12-14.

[11] R. Braden (Ed.). Requirements for Internet Hosts - Communication Layers. RFC 1122 (Internet Standard), October 1989. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc1122.txt. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864, 8029.

[12] Wolfgang Brandstetter. Anfragebeantwortung klärung hinsichtlich netzsperren, 2016. URL https://www.parlament.gv.at/PAKT/VHG/XXV/AB/AB_07061/imfname_502976.pdf. "7061/AB vom 04.02.2016 zu 7304/J (XXV.GP)", Accessed: 2018-03-03.

[13] Tania Branigan. New York Times blocked by China after report on wealth of Wen Jiabao's family, 2012. URL https://www.theguardian.com/world/2012/oct/26/new-york-times-china-wen-jiabao. Accessed: 2018-03-03.

[14] Aaron Brown. Pirate Bay proxy users caught downloading could get 10 YEARS in JAIL, 2017. URL https://www.express.co.uk/life-style/science-technology/801317/Pirate-Bay-Proxy-TorrentFreak-Torrent-Download-Jail. Accessed: 2018-03-02.

[15] United States Census Bureau. Frequently Occurring Surnames from Census 1990 – Names Files. URL https://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html. Accessed: 2018-03-06.

[16] Hanno Böck. Captive Portals, Ein Workaround, der bald nicht mehr funktionieren wird, 2016. URL https://www.golem.de/news/captive-portals-ein-workaround-der-bald-nicht-mehr-funktionieren-wird-1602-118963.html. Accessed: 2017-12-12.

[17] Chaos Computer Club CCC. 34. Chaos Communication Congress. URL https://events.ccc.de/congress/2017/wiki/Main_Page. Accessed: 2018-03-19.

[18] Shashwat Chaudhary. Evil Twin Tutorial, 2014. URL http://www.kalitutorials.net/2014/07/evil-twin-tutorial.html. Accessed: 2018-02-21.

[19] Ningning Cheng, Xinlei Oscar Wang, Wei Cheng, Prasant Mohapatra, and Aruna Seneviratne. Characterizing Privacy Leakage of Public WiFi Networks for Users on Travel. In *INFOCOM, 2013 Proceedings IEEE*, pages 2769–2777. IEEE, 2013.

[20] Jonathan Chew. Medium Has Been Blocked In Malaysia, 2016. URL http://fortune.com/2016/01/27/medium-malaysia-block/.

[21] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard), March 2003. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc3501.txt. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186, 6858, 7817, 8314.

[22] Adrian Dabrowski, Georg Merzdovnik, Nikolaus Kommenda, and Edgar Weippl. Browser History Stealing with Captive Wi-Fi Portals. In *Security and Privacy Workshops (SPW), 2016 IEEE*, pages 234–240. IEEE, 2016.

[23] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc2460.txt. Obsoleted by RFC 8200, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.

[24] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 8200 (Internet Standard), July 2017. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc8200.txt.

[25] RIS Rechtsinformationssystem des Bundes. Entscheidungstext, Bit Torrent, 2017. URL https://www.ris.bka.gv.at/Dokument.wxe?Abfrage=Justiz&Dokumentnummer=JJT_20171024_OGH0002_0040OB00121_17Y0000_000. Geschäftszahl RIS - 4Ob121/17y, Accessed: 2018-03-03.

[26] RIS Rechtsinformationssystem des Bundes. Entscheidungstext, UPC Telekabel II/kino.to, 2017. URL https://www.ris.bka.gv.at/Dokument.wxe?Abfrage=Justiz&Dokumentnummer=JJT_20140624_OGH0002_0040OB00071_14S0000_000. Geschäftszahl RIS - 4Ob71/14s, Accessed: 2018-03-03.

[27] Google Developers. Optimizing for Doze and App Standby, . URL https://developer.android.com/training/monitoring-device-state/doze-standby.html. Accessed: 2018-03-18.

[28] Google Developers. Security with HTTPS and SSL, . URL https://developer.android.com/training/articles/security-ssl.html#UnknownCa. Accessed: 2018-03-15.

[29] Google Developers. Geolocation API - WiFi access point objects, . URL https://developers.google.com/maps/documentation/geolocation/intro#wifi_access_point_object. Accessed: 2018-05-06.

[30] Google Developers. Reverse Geocoding (Address Lookup), . URL https://developers.google.com/maps/documentation/javascript/geocoding?hl=en#ReverseGeocoding. Accessed: 2018-05-06.

[31] Google Developers. Android 4.4 APIs, . URL https://developer.android.com/about/versions/android-4.4.html#Behaviors. Accessed: 2018-03-18.

[32] Google Developers. Android 6.0 Changes, . URL https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html#behavior-network. Accessed: 2018-03-18.

[33] Google Developers. Dashboards, . URL https://developer.android.com/about/dashboards/index.html. Accessed: 2018-03-09.

[34] Google Developers. Location Strategies, . URL https://developer.android.com/guide/topics/location/strategies.html#Updates. Accessed: 2018-03-18.

[35] Christian J Dietrich, Christian Rossow, Felix C Freiling, Herbert Bos, Maarten Van Steen, and Norbert Pohlmann. On Botnets that use DNS for Command and Control. In *Computer Network Defense (EC2ND), 2011 Seventh European Conference on*, pages 9–16. IEEE, 2011.

[36] Felix Disselhoff. Massenabmahnungen nach Porno-Streaming, 2013. URL http://meedia.de/2013/12/09/massenabmahnungen-nach-porno-streaming/. Accessed: 2018-03-03.

[37] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J Alex Halderman. Neither Snow Nor Rain Nor MITM... An Empirical Analysis of Email Delivery Security. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 27–39. ACM, 2015.

[38] D. Eastlake 3rd. Domain Name System (DNS) Case Insensitivity Clarification. RFC 4343 (Proposed Standard), January 2006. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc4343.txt.

[39] R. Elz and R. Bush. Clarifications to the DNS Specification. RFC 2181 (Proposed Standard), July 1997. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc2181.txt. Updated by RFCs 4035, 2535, 4343, 4033, 4034, 5452.

[40] Let's Encrypt. Let's Encrypt Stats. URL https://letsencrypt.org/stats/#percent-pageloads. Accessed: 2017-12-04.

[41] F-Secure. Tainted love: how WiFi betrays us, 2014. URL https://fsecureconsumer.files.wordpress.com/2014/09/wi-fi_report_2014_f-secure.pdf. Accessed: 2017-06-21.

[42] Greg Farnham and Antonios Atlasis. Detecting DNS Tunneling. *SANS Institute InfoSec Reading Room*, pages 1–32, 2013.

[43] Monica Ferrari. Top 3 liabilities for Wi-Fi hotspot providers offering free Wi-Fi, 2017. URL https://www.tanaza.com/blog/top-3-liabilities-wi-fi-hotspot-providers-offering-free-wi-fi/. Accessed: 2018-02-21.

[44] R. Fielding (Ed.) and J. Reschke (Ed.). Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard), June 2014. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7230.txt.

[45] P. Ford-Hutchinson. Securing FTP with TLS. RFC 4217 (Proposed Standard), October 2005. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc4217.txt.

[46] Electronic Frontier Foundation. Open Wi-Fi and Copyright: A Primer for Network Operators, 2014. URL https://www.eff.org/files/2014/06/03/open-wifi-copyright.pdf. Accessed: 2018-02-21.

[47] Julien Freudiger. How talkative is your mobile device?: an experimental study of Wi-Fi probe requests. In *WISEC*, 2015.

[48] Gennie Gebhart and Jacob Hoffman-Andrews. How Captive Portals Interfere With Wireless Security and Privacy, 2017. URL https://www.eff.org/de/deeplinks/2017/08/how-captive-portals-interfere-wireless-security-and-privacy. Accessed: 2017-12-12.

[49] R. Gellens and J. Klensin. Message Submission for Mail. RFC 6409 (Internet Standard), November 2011. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6409.txt. Updated by RFC 8314.

[50] Thomer M. Gil. ICMPTX (IP-over-ICMP) HOWTO. URL http://thomer.com/icmptx/. Accessed: 2018-01-09.

[51] Google. Google IPv6, Statistics. URL https://www.google.com/intl/en/ipv6/statistics.html. Accessed: 2018-01-19.

[52] Google. Transparency Report: Percentage of pages loaded over HTTPS, 2017. URL https://www.google.com/transparencyreport/https/metrics/?hl=en. Accessed: 2017-06-21.

[53] The Guardian. Publishing platform Medium may be blocked in China, reports say, 2016. URL https://www.theguardian.com/media/2016/apr/15/medium-blocked-china-internet-censorship. Accessed: 2018-03-03.

[54] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. Point-to-Point Tunneling Protocol (PPTP). RFC 2637 (Informational), July 1999. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc2637.txt.

[55] P. Hoffman. SMTP Service Extension for Secure SMTP over Transport Layer Security. RFC 3207 (Proposed Standard), February 2002. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc3207.txt. Updated by RFC 7817.

[56] Barbara Holzbauer. Diese Seite ist gesperrt. URL https://blog.t-mobile.at/2015/12/18/netzsperre/. Accessed: 2018-03-03.

[57] Dhaval Kapil. icmptunnel. URL https://dhavalkapil.com/icmptunnel/. Accessed: 2018-01-10.

[58] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (Internet Standard), October 2014. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7296.txt. Updated by RFCs 7427, 7670, 8247.

[59] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc4301.txt. Updated by RFCs 6040, 7619.

[60] S. Kitterman. Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1. RFC 7208 (Proposed Standard), April 2014. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7208.txt. Updated by RFC 7372.

[61] J. Klensin. Simple Mail Transfer Protocol. RFC 5321 (Draft Standard), October 2008. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc5321.txt. Updated by RFC 7504.

[62] Kaspersky Lab. Kaspersky Lab International Travel Report: the urge to connect at any cost is putting international travelers' data at risk, 2016. URL https://blog.kaspersky.com/kaspersky-lab-international-travel-report/12429/. Accessed: 2017-11-22.

[63] SSL Labs. SSL Pulse, Monthly Scan: January 03, 2018, 2018. URL https://www.ssllabs.com/ssl-pulse/. Accessed: 2018-01-08.

[64] Marc Laliberte. Lessons from DEFCON 2016 – Bypassing Captive Portals. URL https://www.secplicity.org/2016/08/26/lessons-defcon-2016-bypassing-captive-portals/. Accessed: 2018-01-09.

[65] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928 (Proposed Standard), March 1996. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc1928.txt.

[66] M. Lottor. Simple File Transfer Protocol. RFC 913 (Historic), September 1984. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc913.txt.

[67] Moxie Marlinspike. sslstrip. URL https://moxie.org/software/sslstrip/. Accessed: 2018-01-29.

[68] Moxie Marlinspike. More Tricks For Defeating SSL In Practice. Black Hat USA, 2009. URL http://www.blackhat.com/presentations/bh-usa-09/MARLINSPIKE/BHUSA09-Marlinspike-DefeatSSL-SLIDES.pdf. Accessed: 2018-04-30.

[69] Moxie Marlinspike. New Tricks For Defeating SSL In Practice. Black Hat DC, 2009. URL http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf. Accessed: 2018-04-30.

[70] Mohan The Mass. Hack Computer, hack Facebook / Twitter Password using Cerberus RAT, 2015. URL https://mohanthemass.blogspot.co.at/2015/04/hack-computer-hack-facebook-twitter.html. Accessed: 2018-03-03.

[71] N. McGill and C. Pignataro. Layer 2 Tunneling Protocol Version 3 (L2TPv3) Extended Circuit Status Values. RFC 5641 (Proposed Standard), August 2009. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc5641.txt.

[72] Hugh McIntyre. Illegal Download Site MP3Skull Is Closing For Good, 2016. URL https://www.forbes.com/sites/hughmcintyre/2016/02/26/illegal-download-site-mp3skull-owes-the-music-industry-22-million/#68db580c1c57. Accessed: 2018-03-03.

[73] Ian McShane, Mark A Gregory, and Christopher Wilson. Practicing Safe Public Wi-Fi: Assessing and Managing Data-Security Risks. Centre for Urban Research (CUR) RMIT University, 2016. ISBN 978-0-9941890-9-7.

[74] A. Melnikov. Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols. RFC 7817 (Proposed Standard), March 2016. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7817.txt.

[75] Deron Meranda. Deron's Data Pages, All first names. URL http://deron.meranda.us/data/census-derived-all-first.txt. Accessed: 2018-03-06.

[76] Microsoft. Captive portal, 2017. URL https://docs.microsoft.com/en-us/windows-hardware/drivers/mobilebroadband/captive-portals. Accessed: 2018-01-03.

[77] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Internet Standard), November 1987. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc1034.txt. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936, 8020.

[78] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Internet Standard), November 1987. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc1035.txt. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766.

[79] C. Mullaney. Symantec Official Blog: Morto worm sets a (DNS) record, 2011. URL https://www.symantec.com/connect/blogs/morto-worm-sets-dns-record. Accessed: 2018-01-17.

[80] Gerry Mullany. Guardian Website Blocked in China, Then Restored, 2014. URL https://sinosphere.blogs.nytimes.com/2014/01/08/guardian-website-blocked-in-china/. Accessed: 2018-03-03.

[81] Mullvad. Mullvad client - Advanced options. URL https://mullvad.net/en/guides/mullvad-client-advanced-options/. Accessed: 2018-03-18.

[82] Mike Muuss. The Story of the PING Program. Archived from http://ftp.arl.mil/ mike/ping.html on 2010-09-08. URL https://www.webcitation.org/5saCKBpgH. Accessed: 2018-01-17.

[83] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (Internet Standard), May 1996. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc1939.txt. Updated by RFCs 1957, 2449, 6186, 8314.

[84] Omar Nakhila and Cliff Zou. User-Side Wi-Fi Evil Twin Attack Detection Using Random Wireless Channel Monitoring. In *Military Communications Conference, MILCOM 2016-2016 IEEE*, pages 1243–1248. IEEE, 2016.

[85] C. Newman. Using TLS with IMAP, POP3 and ACAP. RFC 2595 (Proposed Standard), June 1999. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc2595.txt. Updated by RFCs 4616, 7817, 8314.

[86] Red Newswire. 10 Countries where Facebook, Twitter, and Youtube has been banned, 2016. URL https://www.rednewswire.com/10-countries-where-facebook-twitter-and-youtube-has-been-banned/. Accessed: 2018-02-27.

[87] M. Nottingham and R. Fielding. Additional HTTP Status Codes. RFC 6585 (Proposed Standard), April 2012. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6585.txt.

[88] TJ O'Connor and Ben Sangster. honeyM: A Framework for Implementing Virtual Honeyclients for Mobile Devices. In *Proceedings of the third ACM conference on Wireless network security*, pages 129–138. ACM, 2010.

[89] Court of Justice of the European Union. Judgment in Case C-484/14, Tobias Mc Fadden v Sony Music Entertainment Germany GmbH, 2016. URL https://curia.europa.eu/jcms/upload/docs/application/pdf/2016-09/cp160099en.pdf. Press Release No 99/16, Accessed: 2018-02-21.

[90] Court of Justice of the European Union. Judgment in Case C-527/15, Stichting Brein, 2017. URL https://curia.europa.eu/jcms/upload/docs/application/pdf/2017-04/cp170040en.pdf. Press Release No 40/17, Accessed: 2018-03-03.

[91] Wall of Sheep. WHAT IS THE WALL OF SHEEP? URL https://www.wallofsheep.com/pages/wall-of-sheep. Accessed: 2017-11-23.

[92] Phrack. Project Loki, 1996. URL http://phrack.org/issues/49/6.html. Volume Seven, Issue Forty-Nine File 06 of 16, Accessed: 2018-01-10.

[93] Phrack. LOKI2 (the implementation), 1997. URL http://phrack.org/issues/51/6.html. Volume Seven, Issue 51 September 01, 1997, article 06 of 17, Accessed: 2018-01-10.

[94] J. Postel. User Datagram Protocol. RFC 768 (Internet Standard), August 1980. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc768.txt.

[95] J. Postel. Internet Protocol. RFC 791 (Internet Standard), September 1981. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc791.txt. Updated by RFCs 1349, 2474, 6864.

[96] J. Postel. Internet Control Message Protocol. RFC 792 (Internet Standard), September 1981. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc792.txt. Updated by RFCs 950, 4884, 6633, 6918.

[97] J. Postel. Transmission Control Protocol. RFC 793 (Internet Standard), September 1981. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc793.txt. Updated by RFCs 1122, 3168, 6093, 6528.

[98] J. Postel. Remote User Telnet service. RFC 818 (Historic), November 1982. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc818.txt.

[99] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Internet Standard), October 1985. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc959.txt. Updated by RFCs 2228, 2640, 2773, 3659, 5797, 7151.

[100] J. Postel and J.K. Reynolds. Telnet Protocol Specification. RFC 854 (Internet Standard), May 1983. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc854.txt. Updated by RFC 5198.

[101] Niels Provos. A Virtual Honeypot Framework. In *USENIX Security Symposium*, volume 173, pages 1–14, 2004.

[102] DNS Redirector. Captive Portal. URL http://www.dnsredirector.com/portal/. Accessed: 2017-12-12.

[103] Jay Ribak. Active FTP vs. Passive FTP, a Definitive Explanation. URL http://slacksite.com/other/ftp.html. Accessed: 2017-12-11.

[104] Herman Robers. Captive Portal, why do I get those certificate warnings?, 2016. URL http://community.arubanetworks.com/t5/Technology-Blog/Captive-Portal-why-do-I-get-those-certificate-warnings/bap/268921. Accessed: 2017-12-14.

[105] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc3261.txt. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878, 7462, 7463, 8217.

[106] Y. Sheffer, R. Holz, and P. Saint-Andre. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). RFC 7457 (Informational), February 2015. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7457.txt.

[107] A. Shulmin and S. Yunakovsky. Use of DNS Tunneling for C&C Communications, 2017. URL https://securelist.com/use-of-dns-tunneling-for-cc-communications/78203/. Accessed: 2018-01-17.

[108] Abhishek Singh, Ola Nordström, Chenghuai Lu, and Andre dos Santos. Malicious ICMP Tunneling: Defense against the Vulnerability. In *Information Security and Privacy*, pages 217–217. Springer, 2003.

[109] David Snelling. Use The Pirate Bay? This warning could stop you visiting torrents sites again, 2017. URL https://www.express.co.uk/life-style/science-technology/754113/The-Pirate-Bay-KickAss-torrents-warning-government. Accessed: 2018-03-03.

[110] David Snelling. EU court issues illegal download warning as it rules Pirate Bay can be blocked, 2017. URL https://www.telegraph.co.uk/technology/2017/06/14/eu-court-rules-pirate-bay-can-banned-landmark-illegal-download/. Accessed: 2018-03-02.

[111] Wireless Social. Legal Compliance. https://www.wireless-social.com/how-it-works/legal-compliance/. Accessed: 2018-02-21.

[112] Internet Society. State of IPv6 Deployment 2017, 2017. URL https://cdn.prod.internetsociety.org/wp-content/uploads/2017/08/IPv6_report_2017-0606.pdf. Accessed: 2018-01-19.

[113] Christian Solmecke. EuGH – Streaming von illegal verbreiteten Kinofilmen ist eine Urheberrechtsverletzung – eine Abmahnwelle ist dennoch nicht zu erwarten, 2017. URL https://www.wbs-law.de/urheberrecht/eugh-zu-streaming-72808/. Accessed: 2018-03-03.

[114] Nissy Sombatruang, M. Angela Sasse, and Michelle Baddeley. Why do people use unsecure public Wi-Fi? An investigation of behaviour and factors driving decisions. In *STAST '16 Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust*, pages 61–72. ACM, 2016.

[115] D. Stødle. Ping Tunnel, 2011. URL http://www.cs.uit.no/~daniels/PingTunnel/. Accessed: 2018-01-10.

[116] CloudFlare Support. 5xx Server Errors. URL https://support.cloudflare.com/hc/en-us/articles/115003011431/. Accessed: 2018-06-24.

[117] Symantec. NORTON WI-FI RISK REPORT Report of Online Survey Results in 15 Global Markets, 2017. URL https://www.symantec.com/content/dam/symantec/docs/reports/2017-norton-wifi-risk-report-global-results-summary-en.pdf. Accessed: 2017-11-22.

[118] Cisco Systems. Configuring a Custom-Hosted Splash Page. URL https://documentation.meraki.com/MR/Splash_Page/Configuring_a_Custom-Hosted_Splash_Page. Accessed: 2018-03-09.

[119] Check Point Software Technologies. Configuring UserCheck. URL https://sc1.checkpoint.com/documents/R76/CP_R76_AppControl_WebAdmin/83287.htm. Accessed: 2018-09-16.

[120] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. DNS Extensions to Support IP Version 6. RFC 3596 (Internet Standard), October 2003. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc3596.txt.

[121] tintinweb. striptls 0.5, poc implementation of STARTTLS stripping attacks. URL https://pypi.python.org/pypi/striptls/0.5. Accessed: 2018-01-24.

[122] TorGuard. Six countries that block Social Networks, 2015. URL https://torguard.net/blog/six-countries-that-block-social-networks/. Accessed: 2018-02-27.

[123] S. Turner and T. Polk. Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176 (Proposed Standard), March 2011. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6176.txt.

[124] Cisco Umbrella. What Is the Difference between Authoritative and Recursive DNS Nameservers? URL https://umbrella.cisco.com/blog/2014/07/16/

difference-authoritative-recursive-dns-nameservers/. Accessed: 2018-02-16.

[125] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled Onions: Exposing Malicious Tor Exit Relays. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 304–331. Springer, 2014.

[126] Internet Society World IPv6 Launch. World IPv6 Launch. URL http://www.worldipv6launch.org. Accessed: 2018-01-19.

[127] Peter Yeung. Erdogan emails: Turkey blocks access to WikiLeaks after release of 300,000 secret government emails, 2016. URL http://www.independent.co.uk/news/world/europe/wikileaks-emails-release-government-turkey-erdogan-block-a7145671.html. Accessed: 2018-03-03.