FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Security and Privacy in Mobile Environments

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Georg Merzdovnik, BSc
Matrikelnummer 0525449

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dipl.-Ing. Mag.rer.soc.oec. Dr. techn. Edgar R. Weippl

Diese Dissertation haben begutachtet:

Davide Balzarotti	Stefan Mangard

Wien, 24. April 2017

Georg Merzdovnik

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Security and Privacy in Mobile Environments

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Georg Merzdovnik, BSc
Registration Number 0525449

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dipl.-Ing. Mag.rer.soc.oec. Dr. techn. Edgar R. Weippl

The dissertation has been reviewed by:

Davide Balzarotti	Stefan Mangard

Vienna, 24th April, 2017

Georg Merzdovnik

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Georg Merzdovnik, BSc
Oberndorf 80
3820 Raabs an der Thaya
Austria

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, 24. April 2017

Georg Merzdovnik

Acknowledgements

I would like to thank my supervisor, Edgar Weippl, for providing me the environment to pursue my ideas and work on this thesis. Furthermore, I would like to thank my great colleagues at SBA Research for all the interesting talks and discussions during coffee breaks.

Especially our *A-Team*, Artemios Voyiatzis, Aljosha Judmayer, Johanna Ullrich, and Nicholas Stifter, whom I want to thank for still working/sharing a room with me, although I am surely not always easy to deal with. But then, a Bitcoin miner really makes a lot of noise that needs to be complained about. Without Artemios, this thesis would not be at the point where it is now. Thank you for all your patience, discussions, and help to put this work together. Thanks to our "Bitcoin-Boys", Aljosha and Nicholas, I learned more about Bitcoins, blockchains, and smart contracts than I ever intended to. While we shared a lot of jokes about it, it's nonetheless always inspiring to work with people who bring so much passion for their topic that you can't help but learn while listening to their discussions.

Much appreciation also goes to my colleagues in teaching. For me it was a great experience to be able to pass on my knowledge. I also thank my comrades from *We_0wn_Y0u*, our TU Wien based CTF team. Although keeping the team running and playing CTF's cost me a lot of weekends, it was always fun playing with you guys.

Foremost, I am most grateful for my beloved wife Karoline. Thank you for being by my side through all these years. You always supported me in my love for computers, even if this meant that I spent long nights and weekends in front of a screen. I am also deeply grateful for my two children, Isabella and Alexander. It were often the simple things, like playing with them in the sand, that helped me to care less about setbacks and motivated me to keep going.

Kurzfassung

Die Anzahl und Verbreitung von Smartphones unterliegt einem ständigen Wachstum, und Smartphones wurden zu einem zentralen Teil im Leben ihrer Benutzer. Sie erlauben einen einfachen Zugriff auf vielfältige Informationen und ermöglichen überdies unterschiedlichste Arten der Kommunikation. Eine wichtige Rolle in ihrem Erfolg spielt das große Angebot an Applikationen. Zu Jahresbeginn 2017 enthielt alleine der Google Play Store mehr als 2.6 Millionen Applikationen. Diese reichen von einfachen Spielen oder Wetter-Apps, über Messaging Services bis hin zu Office-Anwendungen. Neben ihrer Nützlichkeit können sie allerdings auch ein Risiko für Anwender darstellen.

Die immense Anzahl an verfügbaren Anwendungen bringt auch qualitativ minderwertige Software mit sich. Während solche Anwendungen möglicherweise kein direktes Risiko für Geräte darstellen, kommunizieren sie oft mit Backend Servern im Internet. Wetter Applikationen benötigen Daten-Updates, Spiele erlauben In-Game Käufe oder speichern andere Informationen auf Servern, und Office-Anwendungen sind oft mit Cloud-Speichern verknüpft. Selbst wenn eine App selbst keine Netzwerkverbindung benötigt, inkludieren Gratis-Anwendungen oft Werbung, welche wiederum aus dem Internet nachgeladen werden muss. Das eigentliche Problem entsteht, wenn diese Kommunikationswege nicht ausreichend gesichert sind. Einerseits wird oft nicht einmal Transport Layer Security (TLS) verwendet, andererseits könnte die Umsetzung von TLS fehlerhaft sein. Eine verbreitete Praxis unter Android Apps ist *Certificate Pinning*, um ein Server Zertifikat zu fixieren (to pin), und keine anderen Zertifikate, auch keine legitimen, zu akzeptieren. Bei korrekter Implementierung bietet Certificate Pinning Schutz gegen Man-in-the-Middle (MitM) Angriffe. Viele Applikation verwenden allerdings eine eigene, oft inkorrekte Implementierung. Oftmals brechen diese die im System integrierte Verifikation von TLS-Zertifikaten und bieten dadurch weniger Sicherheit als eine vergleichbare Lösung ohne Certificate Pinning.

Darüber hinaus zieht ein Ökosystem dieser Größe auch bösartige Anwendungen an. Obwohl Google automatisierte Analyse von Applikationen einsetzt, um diese aus dem Play Store fern zu halten, kommt es trotzdem zur Verbreitung von Malware. Und einige andere App-Stores führen keine initiale Sicherheitsanalyse von eingestellten Applikationen durch. Für Benutzer selbst ist es oft unmöglich zwischen böswilligen und nicht-bösartigen

Anwendungen zu unterscheiden. Auch deshalb, weil Malicious Software (Malware) unter Android häufig in gutartige Programme eingeschleust und anschließend weiterverbreitet wird. Diese beiden Versionen sind für normale Benutzer in ihrem Verhalten und Aussehen nicht unterscheidbar. Daher wurden bereits unterschiedlichste Analyse-Plattformen zur Malware Identifikation vorgestellt. Diese Systeme widmen sich in der Regel allerdings unterschiedlichen Problemen und sind oft nicht in der Lage, alle Gefahren korrekt zu identifizieren.

In dieser Arbeit fokussieren wir uns daher darauf, die Sicherheit und Privatsphäre von Anwendern in mobilen Umgebungen zu verbessern. Einerseits interessieren wir uns für die Sicherheit der Netzwerkkommunikation. Wir analysieren das bestehende Ökosystem von Third-Party-Tracking in Web- und Mobilanwendungen und bewerten Präventionsmaßnahmen nach ihrer Wirksamkeit bei der Blockierung von Tracking-Bemühungen. Wir zeigen, dass eine Vielzahl an Informationen noch im Klartext übertragen wird ohne Transport Layer Security zu verwenden. Darüber hinaus, auch wenn TLS verwendet wird, können diese Tracking-Informationen noch von Angreifern für bestimmte Attacken verwendet werden. Auf der Grundlage unserer Erkenntnisse schlagen wir Ansätze zum Schutz der Nutzer vor.

Jüngste Fälle haben gezeigt, dass auch *valide* Zertifikate, welche im On-Device Trust Store eingetragen sind, nicht immer als vertrauenswürdig eingestuft werden können. Wir analysieren alternative Methoden zur Zertifikatsvalidierung, speziell Notar-basierte Validierungssysteme. In einer Langzeitstudie über die Dauer eines Jahres präsentieren wir tägliche Messungen der Zertifikatsvalidierungs-Fähigkeiten verschiedener Notar-Systeme. Während die Verwendung dieser Systeme für Web-basierte Anwendungen abnimmt, argumentieren wir, dass ihre Verwendung in bestimmten Szenarien, wie *Internet of Things* (IoT) oder mobilen Anwendungen, nützlich sein kann.

Mobile Apps verwenden bereits Certificate Pinning, um Interception-Angriffe zu verhindern. Ein offenes Problem ist die Notwendigkeit von Anwendungsaktualisierungen bei sich ändernden Zertifikaten. Wir stellen eine On-Device Certificate Pinning Lösung basierend auf Notar-Services zur Verfügung, welche gepinnte Zertifikate ohne Anwendungs-Updates automatisch aktualisiert. Dies funktioniert für die Anwendung transparent und braucht keine Unterstützung durch deren Entwickler. Wir argumentieren, dass dieses Szenario auch nützlich sein könnte, um die Sicherheit der Netzwerkschicht für IoT-Geräte zu erhöhen, wo Software-Updates möglicherweise nicht so leicht einspielbar sind.

Schließlich bewerten wir vorhandene Android Malware Analyseplattformen und geben Auskunft über die Effektivität und Abhängigkeiten dieser Dienste untereinander. Dies ermöglicht es Analysten, die am besten passenden Systeme zur Erfüllung ihre Analyseaufgaben auszuwählen. Zusätzlich präsentieren wir Lösungsvorschläge um die Unterschiede in diesen Systemen auszugleichen, um effektivere Malware-Analyse zu ermöglichen.

Abstract

The number of smartphones is constantly increasing and they have become a central part of user's life. They allow for easy access to all kind of information and furthermore enable means for communication. A big role for their success is the huge amount of available applications. As of the beginning of 2017, the Google Play Store alone holds more than 2.6 million applications. These applications range from games or weather apps, through messengers to office suites, they can also pose a risk for their users.

With the huge amount of available applications, it is inevitable that they also include bad quality software. While these applications may not pose a direct risk to the device itself, many of them are communicating to some kind of back-end server on the internet. Weather applications need to update their data; games allow for in-game purchases or just store game information on the server, and office applications are often tied to some kind of cloud storage. Moreover, even if the application itself has no direct need for communication, often "free" applications include some kind of advertisement, which in turn again needs to communicate with some server. The problem arises, if these communication paths are insufficiently secured. On the one hand, they might not even use Transport Layer Security (TLS) for communication. On the other hand, their use of TLS may be broken. A common practice among Android applications is to use certificate pinning, to *pin* a server's certificate and do not accept any other, not even legitimate, certificates. If implemented correctly, pinning provides protection against man-in-the-middle attacks. However, many applications have a broken custom implementation of certificate pinning. Sometimes this even breaks ordinary TLS certificate verification, thereby rendering the application less secure compared to not implementing certificate pinning at all.

Another problem that such a large ecosystem and user base attracts are malicious applications. While Google employs automated analysis of applications to keep them out of the Google Play store itself, even there sometimes malware slips through. Other available app-markets may not even employ an initial security analysis of applications. For users, it is often hard to differentiate between malicious and non-malicious applications, because harmful apps are often re-packaged versions of benign software, which adds a malicious piece of code. Since these two versions are indistinguishable during normal

operation, several analysis platforms to identify malicious applications have been proposed already. However, these systems usually attack different problems and therefore still lack certain functionality to identify all threats.

In this thesis, we therefore focus on means to enhance users' security and privacy in mobile environments. We deal with different aspects of security. On the highest layer, we are interested in issues with network communications. We analyze the existing ecosystem of third party tracking in web and mobile applications and evaluate defenses according to their effectiveness in blocking tracking efforts. We show that there is still a lot of information transmitted in clear text, without the use of Transport Layer Security. In addition, even when TLS is used, this tracking information can still be used by attackers for certain kind of attacks. Based on our findings we propose different approaches to protect users.

Recent cases have shown that not even *valid* certificates, trusted by the on-device Trust Store can be trusted in all cases. We analyze alternative means for certificate validation. Specifically, we explore notary-based validation schemes. We provide a longitudinal study through the course of one year and present daily measurements of certificate validation capabilities of various notary services. While the use of these schemes seems to be diminishing for web-based applications, we argue that their use can be useful in certain scenarios, like Internet of Things (IoT) or mobile applications.

Mobile apps already employ certificate pinning to prevent interception attacks. A problem that remains is that the application still needs to be updated when the corresponding certificate changes. We therefore provide an on-device certificate pinning solution, which utilizes notary services to update pinned certificates automatically, without the need for application updates. It works transparent to the application and does not need any assistance by the developer. We argue that this scenario could also be viable to increase network layer security for IoT devices, where software updates might not be so easily possible.

Finally, we evaluate existing Android malware analysis platforms and provide information on the effectiveness and interdependencies of these services. This allows analysts to select the best fitting system or subset of systems to accomplish their analysis task.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Smartphones gained an important role in users' lives throughout the past years. Apart from their apparent application as mobile phones, they gained further significance as a centralized point of communication and organization. Messenger applications, as Facebook and WhatsApp are used to connect with your friends, banking applications and mobile wallets allow us to directly control our funds and calendar while mail applications let us keep track of our private and work affairs. A large software ecosystem for these applications is evolving at unprecedented speeds. Furthermore, the Internet of Things (IoT) continuously introduces additional devices into this ecosystem. Devices, such as smartwatches, smart thermostats, and internet connected lighting systems are built with size, design and ease of use in mind. Therefore, the central hub to supervise, control, monitor, and configure these connected appliances nowadays is usually a user's smartphone with an accompanying application.

However, while there is a continuous growth of available systems and applications, at the same time their users are exposed to numerous threats violating their security and privacy:

- Bad quality of software increases the attack surface and possibility of data leakage. Today's ecosystem of available code sources makes it easy for careless developers to copy functionality without verifying their correct functionality.
- Nefarious practices by service providers expose private information of their users. Personalized information about users can be a valuable good for companies. Therefore a bad practice that is often employed is the collection of as much information

as possible about users, even if this information may be unrelated to the provided service itself.

- Malevolent actors spread malicious software (malware) to gain advantage over their victims. In the modern IoT world, sophisticated malware incorporate multiple layers of defense and advanced attack strategies, often coordinated from remote command and control centers.

While all of these problems are already well known, defense techniques still lack behind. For IoT devices also often security and privacy are not a first-hand concern for the product developers since they increase the overhead for communication and data processing, which is often a concern in low-power devices.

Online services and modern communication forms enforce this problem even further. On the one hand, sensitive user data is constantly transmitted to servers distributed around the world. Online service providers, like advertisement (ad)-network providers, are just beginning to shift from plain text communication to secured and encrypted data transfers to protect the users information from eavesdroppers. Furthermore, the users have to trust the service provider to not misuse their data, to protect it properly against leakage, and to ensure that the information collected about them cannot be accessed by unauthorized sources. Often, there is no way for a user to either verify or extend protections to increase app and device protection.

1.2 Problem Statement

This thesis explores and evaluates threats against the privacy and security of unaware users on otherwise secure environments. Hereby we focus on different layers in the environment. On the one hand, Network-level attackers might be able to access sensitive information if data transfers lack sufficient protection. On the other hand, malicious applications can compromise the whole device and thereby affect the user directly. Therefore, a user has to trust the network communication, third party libraries embedded in the application (like advertisement libraries), and the application itself. In this thesis, we therefore look at the following problems:

User Privacy: For modern web and mobile applications, it has become a common practice to rely on third party services, like advertisements, analytics, social integration widgets and more. While they provide benefits for the site's owner, their widespread use has serious implications for the users. Data aggregation by these third parties can affect privacy. Furthermore, unsecured network transfers may also compromise a users security, by allowing malicious actors to embed malicious code into delivered content. A prominent case which gained widespread attention

is the piggybacking of third-party tracking cookies by the National Security Agency (NSA) to de-anonymize Tor users [1]. Many tech savvy users rely on blocking tools to mitigate the risk posed by such services. Previous research focused on measuring the prevalence of tracking on common websites [2]–[4] and identifying means of fingerprinting browser [5]–[8].

However, there is still no evidence on how well different blocking mechanisms can protect users from this kind of threat. Therefore the first research topic of this thesis focuses on the state of third-party tracking and if and how well existing tools are able to protect against the different threats posed by this technique. Based on this analysis we propose improvements to existing techniques to further enhance blocking capabilities.

Network Level Protections: While third party libraries can pose a risk for users, often the problems already start at a lower level. Transport Layer Security (TLS) is currently the most widely-used protocol on the Internet to facilitate secure communications, in particular secure web browsing. TLS relies on X.509 certificates as a major building block to establish a secure communication channel. The security provided to Internet applications by the TLS protocol relies on the trust we put on Certificate Authorities (CAs) issuing valid X.509 identity certificates. Certificate Authorities are trusted third parties that validate the TLS certificates and establish trust relationships between communication entities. However, recent incidents have shown that the subversion of the chain of trust is viable. Examples include the infamously hacked certificate authorities DigiNotar and Comodo, during which their private keys were stolen [9]. Incidents such as the case of Superfish[1] and the Dell eDellroot certificate[2] demonstrate that sometimes even system vendors, like Lenovo or Dell, accidentally introduce vulnerabilities.

To counter prevalent attack vectors - like compromised CAs issuing fraudulent certificates and active man-in-the-middle (MitM) attacks - *TLS notary services* were proposed as a solution to verify the legitimacy of certificates using alternative communication channels[3]. While these solutions seem to be promising, they usually depend on a well-functioning ecosystem of servers to work well and reach consensus to validate certificates. To better understand these systems and consider them for further use cases, the second part of this thesis therefore seeks to give a long-term study on the continuous use and operation of different notary service solutions. We analyse their effectiveness in validating certificates and evaluate the additional security they provide for end users.

[1] https://support.lenovo.com/at/de/product_security/superfish

[2] http://en.community.dell.com/dell-blogs/direct2dell/b/direct2dell/archive/2015/11/23/response-to-concerns-regarding-edellroot-certificate

[3] https://github.com/moxie0/Convergence

Application Improvements: More and more developers turn to the development of applications for mobile platforms and a continuously growing number of applications is released per day, with a total of more than 2.6 million apps[4] in the Google Play store, as of January 2017. Since more and more information gets transferred from devices to back-end services, special care to secure these transmissions has to be taken. *TLS certificate pinning* is an approach often used to improve security by restricting communication to trusted sources only. This defense allows mitigating man-in-the-middle (MitM) attacks that employ valid but fraudulent certificates. Yet, the implementation of certificate pinning for mobile applications, and especially for Google Android apps, is cumbersome and error-prone, resulting in inappropriate connection handling and privacy leaks of user information [10], [11]. Furthermore, certificate pinning requires constant application updates, as often as the servers' certificate or chain of trust changes. This could potentially break older applications, which are still in use but not updated anymore.

Therefore, the next part of this thesis proposes application improvements that work independent of the app's developer. The main goal is to improve the use of TLS in applications while at the same time limiting the amount of user interaction needed for these improvements.

Malicious Applications: Android has become the most popular operating system, with an estimated market share of 86.8% at the end of 2016 [12]. Expecting a shipment of 1 billion Android devices in 2017 and with over 50 billion total app downloads since the release of the first Android phone in 2008, cyber criminals naturally expanded their vicious activities towards Google's mobile platform. With an estimated number of 700 new Android applications released every day, keeping control over malware is an increasingly challenging task. Additionally to ordinary problems in application analysis, developers of malicious applications constantly improve their methods to obfuscate applications in order to hinder analysis efforts.

In recent years, a vast number of static and dynamic code analysis platforms for analyzing Android applications and making decision regarding their maliciousness were introduced in academia and in the commercial world. These platforms differ heavily in terms of feature support and application properties that are analyzed. The last part of this thesis therefore concludes with an evaluation of available analysis platforms for Android applications. This gives us the ability to detect dependencies between different systems and propose improvements to enhance analysis results.

[4] `http://www.appbrain.com/stats/number-of-android-apps`

1.3 Main Results

The first part of this thesis focuses on privacy attacks against unaware users on otherwise secure environments.

Software vendors offer various security- and privacy-enhancing tools. Advertisement blocker software aims to protect user privacy by filtering content and metadata for any personal identifiable information that could be used by companies for tracking and targeted advertisements based on user profiles. We perform a large-scale study over a period of one year and show that the vast majority of "ad blockers" perform rather poor [A1]. We also take into account ad-blocking solutions for mobile applications, and show that they do not keep up with their browser-based competitors. The prevalence of user tracking also has further implications.

A common misconception is that the combination of HTTP and TLS are sufficient to protect data in transit, e.g., when involved in activities such as browsing websites. We show that service providers offering Internet access through public Wi-Fi access points can actually steal private information from unsuspected users connecting to them [A7]. By leveraging so called *captive Wi-Fi* portals an attacker is able to steal the browsing history of unsuspecting victims based on transmitted cookies or previously sent HTTP Strict Transport Security (HSTS) headers. Many incidents showcased that otherwise-trusted Certificate Authorities issue fraudulent TLS certificates. If appropriately used, such certificates can be utilized to bypass security and privacy protection. We show that TLS Notary Services can provide an additional layer of protection and study their long-term performance [A2]. Our findings are summarized for a general audience in [A5].

The second part of the thesis is concerned with application level problems of TLS usage. Proper certificate handling and validation is a very challenging task. We study more than 10000 of the most popular general-purpose Google Android apps available on the Google Play store. We show that over the years (2012–2016) the situation worsens: more and more of the most popular applications are implementing certificate handling incorrectly, thereby creating a large surface for man-in-the-middle attacks against user privacy and security [A10]. We therefore implemented automated and transparent handling of certificate pinning for Android applications. It enables correct pinning of certificates, even if an application either has an incorrect implementation or would not support it at all in the first place [A9].

A problem with our proposal for certificate pinning is that it still needs user interaction, every time the certificate of the back-end server changes. To minimize this problem and reduce the amount of user involvement in system-level decisions, we extend the proposed pinning system and introduce certificate validation through the ICSI Notary service [A3]. Based on this we can verify through the notary service servers if the certificate change is legitimate or if we are currently victim of a man-in-the-middle attack. We only need

to inform users in the second case and warn them about potentially malicious behavior. Given the hardness of handling security appropriately, third-party libraries are often provided as a means to simplify development. We extensively study the impact of application middleware on a systems overall security [A8]. Furthermore, we evaluate the usefulness of such libraries for the task of improving user privacy. We also cover possibilities for obfuscating a smartphone's location through system extensions [A11].

The third part of the thesis focuses on analysis of malicious applications. As defenses against malware improve, malware tries to hide its existence and function from analysis and detection tools. Obfuscation is often used to hide malicious functionality within otherwise benign looking applications. We evaluate the state-of-the-art obfuscation methods and evaluated their functionality with respect to existing analysis tools [A13]. Apart from software only obfuscation we also looked into ways of hardware-assisted obfuscation, making analysis even more difficult [A12]. The huge amount of Android malware also triggered the development of various different analysis tools and environments. To better understand the available tools and evaluate their performance, we assess the state-of-the-art dynamic code analysis platforms for Android and evaluate their effectiveness with samples from known malware corpora as well as known Android bugs [A14]. Our analysis indicates a low level of diversity in analysis platforms resulting from code reuse that leaves the evaluated systems vulnerable to evasion.

1.3.1 List of Publications

[A1] **Merzdovnik, G.**, Huber, M., Buhov, D., Nikiforakis, N., Neuner, S., Schmiedecker, M., Weippl, E., "Block Me If You Can: A Large-Scale Study of Tracker-Blocking Tools", in *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, IEEE, 2017,
Lead contribution to the idea with 1 co-author. Contribution of the implementation of the distributed AdBlocker-analysis system for browsers; Contribution to the Android application analysis with one co-author. Evaluation of results for web and mobile applications; Lead contribution to paper draft. Paper revisions with co-authors.

[A2] **Merzdovnik, G.**, Falb, K., Schmiedecker, M., Voyiatzis, A., Weippl, E., "Whom You Gonna Trust? A Longitudinal Study on TLS Notary Services", in *Data and Applications Security and Privacy XXX - 30th Annual IFIP WG 11.3 Conference (DBSec 2016)*, Springer, 2016,
Contribution to the collection system implementation with one co-author; Significant contribution to the data collection; Data evaluation and presentation of results; Significant contribution to the paper draft and paper revisions with co-authors.

[A3] **Merzdovnik, G.**, Buhov, D., Voyiatzis, A. G., Weippl, E. R., "Notary-Assisted Certificate Pinning for Improved Security of Android Apps", in *Availability, Reli-*

ability and Security (ARES), 2016 11th International Conference on, IEEE, 2016, pp. 365–371,
Significant contribution to the basic idea. Contribution to the underlying system implementation and evaluation with 1 co-author; Writing the paper draft; Paper revisions with co-authors.

[A4] **Merzdovnik, G.**, Judmayer, A., Voyiatzis, A. G., Weippl, E., "A performance assessment of network address shuffling in IoT systems (Extended Abstract)", in *Sixteenth International Conference on Computer Aided Systems Theory (EUROCAST 2017). Las Palmas de Gran Canaria, Spain, February 19-24, 2017*, 2017,
Significant contribution to the idea; Implementation of the testbed; Evaluation of results; Paper draft and contribution to proof-reading and revisions with co-authors.

[A5] **Merzdovnik, G.**, Buhov, D., Voyiatzis, A. G., Weippl, E., "u'smile - Secure Mobile Environments", *ERCIM News*, no. 109, pp. 53–54, Apr. 2017,
Contribution of idea and paper draft; Contribution to proof-reading and revisions with co-authors.

[A6] Judmayer, A., **Merzdovnik, G.**, Ullrich, J., Voyiatzis, A. G., Weippl, E., "A performance assessment of network address shuffling in IoT systems", in *Computer Aided Systems Theory – EUROCAST 2017*, ser. LNCS, Springer, 2017, to appear,
Significant contribution to the idea; Implementation of the testbed; Evaluation of results; Paper draft and contribution to proof-reading and revisions with co-authors.

[A7] Dabrowski, A., **Merzdovnik, G.**, Kommenda, N., Weippl, E., "Browser History Stealing with Captive Wi-Fi Portals", in *Proceedings of Workshops at IEEE Security & Privacy 2016, Mobile Security Technologies (MoST)*, 2016,
Significant contribution to the ideas for history stealing possibilities; Collection of validation data; Evaluation of effectiveness of the system; Contribution to proof-reading and paper revision with co-authors.

[A8] Aufner, P., **Merzdovnik, G.**, Huber, M., Weippl, E., "Plugin in the Middle-Minimising Security Risks in Mobile Middleware Implementations", in *Proceedings of the 7th International Conference on Security of Information and Networks*, ACM, 2014, p. 434,
Contribution to the idea and implementation with co-authors. Lead contribution of the paper draft. Contribution to proof-reading and paper revisions with co-authors.

[A9] Buhov, D., Huber, M., **Merzdovnik, G.**, Weippl, E., "Pin It! Improving Android network security at runtime", in *Proceedings of the 15th IFIP Networking Conference, Networking 2016*, IEEE, 2016, pp. 297–305,

Contribution to the underlying idea and to the paper draft with two co-authors; Contribution to proof-reading and paper revisions.

[A10] Buhov, D., Huber, M., **Merzdovnik, G.**, Weippl, E., Dimitrova, V., "Network Security Challenges in Android Applications", in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security (ARES 2015)*, IEEE Computer Society, 2015, pp. 327–332,
Contribution to the ideas; Contribution to the paper draft and paper revisions with co-authors.

[A11] Hochreiner, C., Huber, M., **Merzdovnik, G.**, Weippl, E., "Towards Practical Methods to Protect the Privacy of Location Information with Mobile Devices", in *Proceedings of the 7th International Conference on Security of Information and Networks*, ACM, 2014, p. 17,
Contribution to the idea with co-authors; Contribution to the paper draft, proof-reading and paper revisions with co-authors.

[A12] Schrittwieser, S., Katzenbeisser, S., **Merzdovnik, G.**, Kieseberg, P., Weippl, E., "AES-SEC: Improving software obfuscation through hardware-assistance", in *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, IEEE, 2014, pp. 184–191,
Significant contribution to the underlying ideas of the proposed system with co-authors; Prototype implementation of the system in LLVM; Performance analysis and evaluation. Contribution to the paper draft and revisions with co-authors.

[A13] Schrittwieser, S., Katzenbeisser, S., Kinder, J., **Merzdovnik, G.**, Weippl, E., "Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?", *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, p. 4, 2016,
Significant contribution to static and dynamic application analysis methods. Contribution to proof-reading and revisions with co-authors.

[A14] Neuner, S., Veen, V. V., Lindorfer, M., Huber, M., **Merzdovnik, G.**, Mulazzani, M., Weippl, E. R., "Enter Sandbox: Android Sandbox Comparison", *Proceedings of the IEEE Mobile Security Technologies workshop (MoST)*, May 2014,
Contribution to the sandbox evaluation with co-authors; Contribution to writing, proof-reading and paper revisions with co-authors.

1.4 Structure of the Work

The further parts of this thesis are structured as follows. Chapter 2 looks at existing literature and concepts related to the remainder of the thesis. This includes aspects from TLS and third party tracking as well as dynamic instrumentation, obfuscation and

analysis of Android applications.

Chapter 3 provides a large-scale study of tracker blocking tools for browsers and Android devices and insights into the prevalence of third party trackers. This is followed by a longitudinal study on the use of TLS notary services in Chapter 4.

Chapter 5 applies the collected knowledge and describes the concept of notary-assisted certificate pinning to protect applications against man-in-the-middle attacks.

Chapter 6 provides an evaluation of Android malware analysis platforms (sandboxes) and provides insights into their functionality and interdependencies.

Finally, Chapter 7 concludes on the different aspects of the work and provides an outlook on future work.

CHAPTER 2

Literature Review and Background Information

The distribution of smartphones is constantly increasing with an estimated number of 1 billion Android devices being shipped in 2017. Together with this increase in distribution also the availability of new applications is constantly rising, with more than 2.6 million applications available in the Google Play Store as of the beginning of 2017. While these developments give more and more people access to a plethora of services, they also put forth new challenges for security and privacy of users. Cyber criminals have naturally extended their vicious activities towards Google's mobile operating system. With an estimated number of 700 new Android applications released every day, keeping control over malware is an increasingly challenging task. Apart from malicious developers and applications, there are other threats that need to be taken into consideration as well. Often applications are poorly designed and implemented and are vulnerable to certain kind of attacks, like man-in-the-middle (MitM) attacks on the network layer. One reason for this is erroneous implementations of TLS certificate validation. Additionally, a vast number of applications available for "free" relies on advertisements to generate income, which might also pose threats to users privacy and security. In recent years, a vast number of static and dynamic code analysis platforms for analyzing Android applications and making decision regarding their maliciousness have been introduced in academia and in the commercial world. These platforms differ heavily in terms of feature support and application properties being analyzed.

In this chapter, we provide a review of the literature on privacy and security implications in Android applications and their corresponding back-end services. First we focus on work related to malicious Android applications, Android application analysis in general and also provide some background on possibilities for obfuscating Android applications.

We then discuss work related to network level analysis and communication protection methods. We specifically focus on the use of TLS as well as possibilities to improve the certificate system to better protect application communication on the network level. Finally, we review application-level problems dealing with certificate validation as well as existing methods to detect these threats.

2.1 Internet Communications

2.1.1 Transport Layer Security (TLS)

The goal of the Transport Layer Security (TLS) protocol is to provide data confidentiality and integrity between two communicating computer applications. An often-used example use of TLS is for securing communication between a web browser (client) and a web site (server). The protocol is defined in various proposed standards by the IETF, including among others RFCs 2246, 3546, 4346, 4366, 4680, 4492, 5246, 5288, 5746, 6176, and 6655. The origins of the TLS protocol date back to 1993, when SSL v1.0 was defined. The current version is 1.2 and the next major protocol revision, TLS v1.3, is expected soon [13].

2.1.2 TLS and X.509 certificate validation

TLS can, optionally, authenticate the identity of the two communicating parties using public-key cryptography. This is widely used for at least authenticating the server side, i.e., for proving that a (web) client indeed connects with the intended (web) server. The server authenticity is based on the Internet X.509 Public Key Infrastructure Certificate, as defined in RFCs 5280 and 6818. Connection establishment between client and server is depicted in Figure 2.1.

When a client connects to a server over TLS, the server presents its certificate for proving its identity. The server certificate should be signed by a certificate authority (CA) that the client trusts, either explicitly (e.g., by having the user click on a warning message) or implicitly (e.g., by consulting its "trust store", i.e., a set of pre-distributed "root" certificates through which a chain of trust is built).

All root certificates are considered equally-trusted. Hence, any of the CAs can issue an equally-valid certificate for a given server. If any of the CAs is compromised, then it can be tricked to issue a fake but valid certificate for a server.

To cope with this inefficiency, various approaches have been proposed. A Certificate Revocation List (CRL) can be periodically distributed stating which of the issued certificates by a CA are not valid anymore. This can still leave a window of opportunity for an attacker, until a client updates its CRL. Online Certificate Status Protocol (OCSP)

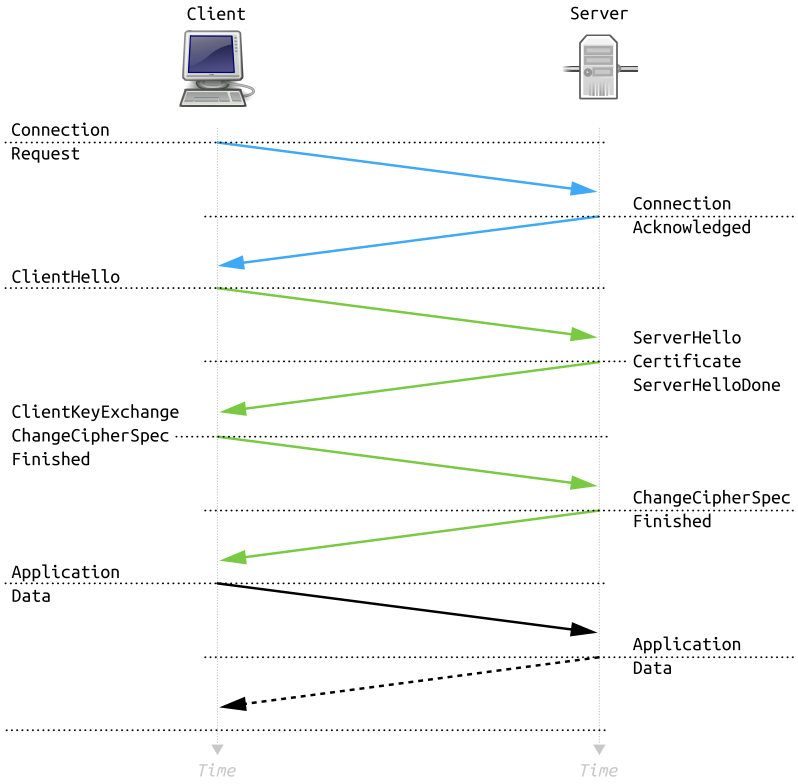


Figure 2.1: Full TLS1.2 handshake (Source: Wikipedia

was proposed to solve this problem by allowing a client to contact online (at the time of a TLS connection setup) the CA and verify the validity of the presented certificate. Yet, this extra connection with a third-party server can introduce significant latency in page loads, especially in environments with mobile clients (e.g., smartphones) connecting over wireless or cellular links. OCSP stapling (formally, the TLS Certificate Status Request extension, defined in RFC 6066) removes some of this burden by allowing the server to append (staple) a time-stamped OCSP response signed by the CA during the initial TLS handshake.

HTTP Public Key Pinning (HPKP) is defined in RFC 7469 [14]. It allows a server to "pin" the hashes (fingerprints) of the valid certificates during a connection. On subsequent connections the client can check the hashes of the presented certificate. If they do not match the known ones, then the client can assume that a man-in-the-middle (MitM) attack is taking place.

HPKP cannot defend against impersonation attacks mounted when a client visits a previously unknown server. In this case, even if HPKP is employed by the server side, the client has to inherently trust the unknown hashes presented by the MitM attacker.

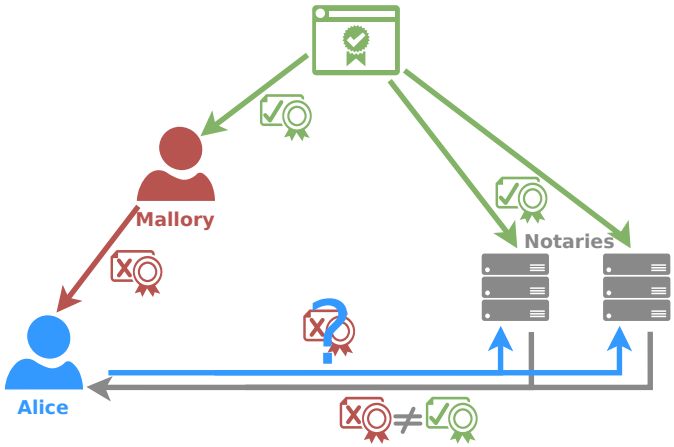


Figure 2.2: Detecting impersonation attacks using TLS Notary Services

Other protection mechanisms have been implemented in the form of browser extensions. Soghoian et al. [15] implemented *Certlock*, an extension that is based on the trust-on-first-use policy to bind the CA to the CommonName of a websites certificate. This method is similar to pinning every certificate on first encounter. Winter et al. [16] provide a system that uses an independent Tor circuit for certificates that issued a browser warning. However, this check is only issued for certificates triggereing a warning, therefore it does not protect against valid but yet malicious certificates. Syverson and Boyce also employ Tor for page verification, but they do not rely on probing the same server on the same domain; instead, they host the site again on a `.onion` address and use this mirror to compare the keys [17]. Holz et al. [18] implemented CrossBear, a system which employs hunter nodes to track down TLS MitM attacks.

2.1.3 TLS Notary Services

TLS certificate notary services can be used to verify a certificate through multiple paths. Wendlandt et al. [19] proposed *Perspectives*, which is based on multiple servers to observe the state of TLS certificates. *Convergence* [20] builds on the same principles as *Perspectives* and provides further methods for trust management.

TLS Notary Services (or simply, "notaries") are a defense against impersonation attacks utilizing crowd knowledge, as collected by notary servers. The key observation is that an attacker is not able to intercept all possible communication links with a server and mount MitM attacks. Thus, notaries can collect certificates from different points of observation (i.e., perform multi-path probing), which cannot be intercepted concurrently or altogether. As illustrated in Figure 2.2, when a client is presented with a (possibly

[5] `http://www.perspectives-project.org/`
[6] `http://www.convergence.io/`

impersonated) server certificate, it consults the publicly-available notaries and compares the received results in order to detect the attack.

Perspectives[5] and Convergence[6] are two example implementations of *active* notaries. The ICSI Certificate Notary[7] is an example implementation of a *passive* notary. It builds its certificate database by passively monitoring traffic at multiple independent Internet sites. The database can be queried through a public DNS interface.

The scalability issues of Convergence are studied in [21]. Laribus [22] is an attempt to build a peer-to-peer (P2P) notary service exploiting social-connectivity graphs so as to remove the need for centralized notaries. A longitudinal study on the availability and functionality of publicly-available TLS Notary Services is provided in [23].

2.1.4 Large-Scale TLS Protocol Studies

The problems with TLS and trusted certificate authorities have been studied for years. Several large-scale studies that focused on the TLS ecosystem have been conducted lately. One of the first large-scale studies targeting TLS certificates is the Electronic Frontier Foundation's SSL Observatory [24]. Its dataset includes publicly visible SSL certificates available through IPv4.

Holz and Durumeric [25], [26] focused on the IPv4-wide analysis of TLS in the context of HTTPS. Mayer et al. [27] and Holz et al. [28] focused on TLS in other application domains, like the e-mail ecosystem. In particular, the recently proposed improvements to port-scanning as well as the open-source release of tools like *zmap* [29] and *masscan* [30] made it easy to collect IPv4-wide information on specific questions. Durumeric et al. also set up a special search engine, Censys [31], which is backed by these Internet-wide scans and allows for deeper analysis. While these studies provide an interesting and valuable view on the TLS ecosystem, they are not designed to provide further information on fraudulently issued certificates.

2.1.5 Certificate validation in Google Android

The Google Android operating system supports inherently the TLS protocol. Android devices come with a bundled trust store containing more than 150 certificates of root certificate authorities [32]. This list is initially populated by Google but may be further customized by third parties, such as the manufacturer of the device and the cellular network operators. The size of the list steadily increases over the years, raising the concern of the research community regarding the trust model of Internet-deployed TLS [33]. Android developers can integrate TLS functionality and certificate validation in their apps for secure communication with servers. This functionality is offered through an API of

[7] `http://notary.icsi.berkeley.edu/`

the operating system. There are currently three alternatives for realizing this:

- TLS and certificate handling by the operating system (default handling).
- Custom validation by the application developer.
- Utilization of the "Network Security Configuration" functionality in Android versions starting from 7 ("Nougat").

Default handling

The default certificate handling is done automatically by the operating system and frees the developer from all housekeeping operations. At the same time, it provides the least of the control on how exactly the validation is performed. The developer must procure and install a server certificate signed by one of the (many) trusted CAs that come with the operating system and take appropriate action in case the certificate is expired or the root CA is removed, e.g., due to a root CA compromise.

Certificate pinning is currently not supported in this mode. Thus, the security-cautious developers must implement by themselves custom validation to realize this helpful functionality of certificate handling.

Custom validation

A second option for the developers is to handle the certificate validation by themselves. This offers the greatest flexibility and allows support for self-signed certificates as well. At the same time, the application developer is solely responsible for the flawless implementation of the admittedly complex procedure of chained certificate validation. Until now, custom validation is the only means to implement certificate pinning functionality.

Android Nougat

Android N (Nougat) is the codename for the latest release of the Android operating system (version 7), announced in March 2016. In Android N, apps can customize the behavior of their secure (HTTPS, TLS) connections safely, without any code modification, by using the declarative *Network Security Config*[8] instead of using the conventional error-prone programmatic APIs (e.g., the X509TrustManager).

Android N is the first version supporting certificate pinning at the application level as a means to defend against MitM attacks[9]. The pinning information will be bundled within the `network_security_config.xml` resource file of each application.

[8] `https://developer.android.com/preview/api-overview.html#network_security_config`
[9] `https://developer.android.com/preview/features/security-`

2.1.6 Issues with certificate pinning

Certificate Pinning has already found widespread distribution in mobile applications [10], [11]. However, research findings suggest that the developers cannot cope sufficiently with certificate validation in general and certificate pinning in particular: A study performed in 2012 revealed that more than 1,000 applications out of a sample of 13,500 included a completely wrong implementation of the validation procedures [10]. A year-long study between 2013 and 2014 revealed that the situation actually does not improve but rather it is getting worse over the time [34].

Interviews with developers of applications with broken validation indicate that the developers are not fully aware of the security implications of such erroneous implementations [35]. Quite often, the developers do not consider MitM attacks as a threat altogether. Rather, their aim is solely to implement self-signed certificates because it is more convenient to them, despite the wide availability of free server certificates.

"Pin It!" is a novel approach to offer certificate pinning functionality transparently to the applications [34]. This is achieved by intercepting system calls related to certificate handling and enforcing the pinning with the assistance of the user. If a new certificate is detected, the user is asked to confirm the new certificate hash for future reference. This approach assumes that the first connection with the (web) server is not tampered (the so-called "trusted-on-first-use" or "TOFU" principle) and that the user can make informed decisions about the presented certificates. User surveys suggest that the latter is a strong assumption as the users fail en masse even in the simpler task to judge whether their browser session is protected by TLS or not [35]. It should be noted however that the "Pin It!" approach is the only feasible one at the moment and does not require an application update by the developer. It offers adequate protection for the security-conscious users against a careless developer who implements incorrectly certificate validation and exposes the private data of the users.

The Android N approach is a step towards the right direction as it simplifies the integration of certification pinning. It is less invasive in nature and does not require a rooted device, since the *Network Security Config* is an integral art of the forthcoming operating system. Still, it requires the prompt action of the developer, in order to take advantage of the new functionality.

It remains to be seen if this approach will gain popularity among the developers. Also, if it will be possible for all Android devices to upgrade to the new operating system and how fast; if not and in the meantime, the developers will have to opt for a dual implementation of their application: one that supports certificate pinning through Android N and one through other means (or, even worse, not at all, creating an illusion of security and

`config.html#CertificatePinning`

confusion to the users regarding the offered security level).

We also note that in the case of Android N, certificate pinning will occur on a per-application basis. This will be realized using a bundled resource file. Hence, if the pinning information must be updated (e.g., due to a security incident), the developers must go through the whole, time-consuming process of delivering an application update to their users through the app marketplace /e.g., Google Play), introducing further delays and extending the window of exposure to MitM attacks.

2.1.7 Third Party Tracking

Measurement studies. To the best of our knowledge there has been no study on blocking third-party trackers. Mayer and Mitchell [36] provide a survey describing how companies track users online, and discuss current protection strategies. In addition, they introduced *FourthParty* [37], a web measurement tool to analyse third-party tracking. The authors used FourthParty to analyse the effectiveness of different blocking tools based on the Alexa Top 500. They found *AdblockPlus* with *EasyList* and *EasyPrivacy* to be the most effective browser extension. Balebako et al. [38] proposed a research methodology to analyze behavioral advertising based on web history and textual Google ads. They analyzed the built-in browser functionality, *Ghostery*, *Abine*, as well as the *Do not Track* (DNT) header, which can be enabled in the browsers settings. They found that blocking tools were effective against behavioral profiling by textual Google text ads because this tools completely removed the JavaScript code that generated them. Roesner et al. [2] analyzed the Top 500 websites, Non-Top 500, and the AOL testdata set in 2012 on existing trackers using a custom Firefox extension. Furthermore, they proposed a classification of different trackers according to their relationship with first-party websites and users, and analyzed different browser-protection mechanisms. Reisman et al. [39] showed that the current state of third-party tracking enables surveillance, and analyzed different blocking strategies. They found that Ghostery was the most effective tool for blocking trackers on the Alexa top 500 websites. All mentioned research had one key finding in common: dedicated tracker blockers significantly outperform other protection methods, such as the disabling of third-party cookies, the usage of the Do-Not-Track header, and the setting of opt-out cookies.

Stateless tracking. Motivated by the initial findings of Eckersley [40], a number of researchers further investigated stateless tracking and its implications. Yen et al. [41] performed a fingerprinting study, similar to Eckersley's, by analyzing logs of Bing and Hotmail. Interestingly, the authors found that a client's IP in combination with her user-agent string, provided enough entropy to uniquely identify over 80% of users. Nikiforakis et al. [5] described how fingerprinting works by analyzing the code of three browser-fingerprinting providers. Their work also showed that the spoofing of user-

agents is an insufficient protection method that can cause more harm than good. Acar et al. [7] developed the *FPDetective* framework to detect web-based fingerprinters in the wild. They found that fingerprinting was used by over 400 domains in the Alexa Top 1 Million dataset. In a later study, the authors also investigated the usage of canvas-fingerprinting [42] in the wild as one more vector for uniquely identifying users across multiple websites [6]. The most extensive measurement on stateless tracking has been performed by Englehardt and Narayanan [8], [43], including novel findings on the use of AudioContext fingerprinting.

Tracking defense strategies. A number of defense strategies have been proposed in the past. Guha et al. [44] proposed Privad which acts as a privacy-preserving dealer for advertising. The approach by Guha et al. tries to find a balance between privacy and still showing relevant ads to users. To the best of our knowledge no systems, comparable to Privad, are used by advertisement providers. Recent proposals like the TrackingFree browser by Pan et al. [45], rely on separated identities (browser principals) for different websites in order to hinder tracking by third parties. A similar approach has been proposed by Torres et al. [46], whereas a custom browser extension enforces separate web identities per website [46]. Privaricator [47] uses a modified stock web browser to fake browser fingerprints. These proposals offer promising methods to hinder stateful and stateless tracking but all rely on browser vendors adopting their technologies. Tracker-blocking browser extensions thus offer the best protection strategy against online tracking and these tools also have positive side-effects, such as protecting users from malicious advertisements (malvertising) and active URL hijacking attacks [48]. Cranor [49] showed that these tools are plagued by usability issues and proposed improvements. The usability survey by Leon et al. [50] furthermore highlighted that only one of out five participants was able to enable the optional blocking feature of Ghostery.

Finally, given the effectiveness of tracker blocking tools, the research community focuses on improving of the underlying tracker-blocking rules. Previous research leverages machine learning for complementing existing rulesets. Bau et al. [51] proposed to use supervised machine learning to detect tracking third-party domains. Bhagavatula et al. [52] evaluated different machine learning algorithms and found that the k-nearest neighbor algorithm outperformed the accuracy of other classifiers. They used EasyList as a baseline and analyzed if the could correctly predict URLs included in this popular ad-blocking list. Gugelman et al [53] used a naïve Bayes classifier to detect privacy-intrusive services based on statistical HTTP traffic features. It is interesting to note that the research community bases their experiments on the EasyList and EasyPrivacy rulesets, for training their detection classifiers.

2.2 Android Applications

2.2.1 Improving Security and Privacy with Privileged Applications

The demand to customize tools is part of human nature; from the variety of models, colors, and equipment to choose from when buying a car to the trivial process of choosing a new desktop wallpaper, humans like to give property a touch that makes it *their own*. It is only natural, then, that given a device that can (in theory) perform *any* computational task, users want it to perform them in the way *they* desire.

Jailbreaking [54] has become a very popular modification for iPhones, allowing users to install alternate versions of the App Store and circumvent many other restrictions of the traditionally very locked-down iPhone ecosystem. However, because Apple's business model strongly depends on platform control, vulnerabilities exploited for jailbreaks are commonly patched within weeks and the security of the iOS kernel and bootloader has improved rapidly in recent years. The resulting obstacles for deployment make iOS devices an unattractive target for the development of security-relevant benign root apps.

Android is based on the Linux kernel and thus vulnerable to kernel flaws such as Cheddar Bay (CVE-2009-1897[10]). Additionally, flaws have been found and exploited in Android components such as `vold`, the volume manager daemon (CVE-2011-1823[11], the implementations of which became known as *Gingerbreak* and *zergRush*), or `adb`, the Android Debug Bridge (rageagainstthecage[12]). Lastly, some modifications performed by vendors themselves introduce new vulnerabilities. The most notable case is probably the *Exynos*[13] vulnerability present on some Samsung devices, which is essentially a world-writeable `/dev/kmem` device that allows direct physical memory read/write access to every user or app on the system.

But the possibility to exploit such flaws also opens up entirely new avenues for security and usability enhancements. For instance, Android is the only mobile operating system (OS) where a flexible user-configurable firewall exists - although it is not installed by default and requires root privileges to work.

2.2.2 Network security

Nearly all smartphones are connected to the Internet, and many capabilities that make a mobile device useful - such as the ability to receive and send mails, synchronise calendars or look up public transport connections - require a network connection to work. However,

[10] `http://www.cvedetails.com/cve/CVE-2009-1897`
[11] `http://www.cvedetails.com/cve/CVE-2011-1823`
[12] `https://intrepidusgroup.com/insight/2010/09/android-root-source-code-looking-at-the-c-skills/`
[13] `http://forum.xda-developers.com/showthread.php?t=2048511`

the majority of all Android apps request permission to access the Internet, many of them just to support in-app advertisements. By default, there is no possibility for the user to restrict Internet access to specific hosts or even monitor which apps connect where.

Firewalls

Being able to shield local applications from potentially malicious inbound traffic and blocking specific apps from accessing the Internet is a standard feature in any modern desktop operating system. However, these capabilities do not exist on smartphones today - even though the issues address also exist in mobile environments.

On Linux, firewalling is performed by using the `netfilter`[14] kernel component; it allows for filter decisions based on layer 2-4 characteristics (such as TCP flags and connection status, source and destination port or IP address), device name and even user/group ID of the application that is sending outbound traffic. The latter enables outbound filtering on a per-app basis in Android, as each application has an own `uid` and `gid`.

Android firewalling is relatively straightforward: First, the kernel module `netfilter` must be compiled for the respective kernel (sources of which are mandatory to publish under the GPL, although vendors often delay the release). To install and load the module, root privileges are required. The same goes for the actual configuration using the `iptables` frontend. No further work is necessary.

The simplicity and utility of firewalling on Android has spawned numerous apps (commercial and free) that offer this capability[15]. Common features include application black- or whitelisting based on the connectivity status (such as *Connected via WiFi*, *3G* or *Roaming*), logging, and password protection.

Flow monitoring

Short of actually blocking an application's access to the Internet, concerned users might be interested in knowing which host the apps connect to and when. This may aid users in deciding whether applications behave in a acceptable manner, which remote hosts should be made unreachable for applications and - to a certain degree - which protocols are actually used by an app to communicate with its servers.

This feature can actually be implemented without root access by polling `/proc/net/tcp` and the `fd` entries of all running processes. However, the latter step is very inelegant and should thus be avoided. It is also generally considered a bad idea to use `inotify` on `/proc`. One possible more elegant solution that does require root is the use of

[14] `http://www.netfilter.org`
[15] `http://joyofandroid.com/best-firewall-apps-for-android/`

Xposed Framework[16], a modification that allows pre- and postprocessing hooks for any function called via Dalvik, Android's process virtual machine in pre 5.0 Versions. It works by overwriting `/usr/bin/app_process`, which is the executable responsible for loading classes and initialising applications. One could hook known API methods in the `java.net` package to collect details of connection attempts. Native applications as well as functions included via Java Native Interface (using the Android Native Development Toolkit [NDK]) would bypass this method.

Monitoring DNS resolution

In many cases, users are not interested in the network layer details of connections but would still like to have a quick overview of the servers contacted by applications. Monitoring the resolution of hostnames to IP addresses would allow users to gain just that while working around one of the pitfalls of most flow monitoring implementations: They often rely on reverse DNS records (which are often different from the domain names used by applications, usually because default rDNS settings by the data center have not been modified) to display human-readable information about current connections.

The implementation of this approach is not trivial. DNS resolution is performed by methods defined in `resolv.h`[17] of the `libc` (bionic, in the case of Android), and while the traditional way to *overwrite* functions consists in using the environment variable `LD_PRELOAD` to load a library that includes the specific function, this is completely ignored by `zygote`, the operating system's virtual machine process. Even `LD_LIBRARY_PATH` is only considered once, namely when `zygote` is executed at boot time by `init`. After that, `zygote` simply forks for each application that is started.

It is possible to capture DNS resolution of all apps apart from native processes by modifying the underlying classes provided by Android or using the approach chosen by Xposed[18], which allows the interception of calls into Android libraries.

2.2.3 Host security

While network monitoring and filtering may aid in detecting and suppressing threats through their interactions with other hosts, most malware found on phones today does not rely on this method. Antivirus and OS hardening are the main topics explored in this chapter and present an area where smartphones have a decisive disadvantage compared to desktop environments. For instance, most users would be less than pleased if their laptop simply stopped receiving security updates after a year - however, this is the situation for most smartphones today.

[16] `http://repo.xposed.info`
[17] `http://www.kickflop.net/blog/2011/01/02/tracing-linux-hostname-resolution`
[18] `https://github.com/rovo89/XposedInstaller`

Antivirus

Antivirus suites on traditional desktop environments have long made it a requirement to be granted elevated privileges. Many tasks that are understood to be basic functionality - such as scanning executables as soon as they are downloaded or executed, performing modifications on system files or even terminating malware processes - can not be performed by just any app, and thankfully so.

However, only two mobile antivirus solutions make use of superuser privileges (if enabled to do so by the user). At least one of them does not utilise its privileges during traditional malware detection or removal tasks, instead using it to offer a firewall (as discussed above), app backups and protection from unauthorised removal. There is a lot more potential for these applications, especially in combination with functionality as provided by the Xposed Framework (or similar offerings). Android malware is often used to send premium SMS' or calls [55], events that can and should trigger heuristic detection mechanisms. Furthermore, while it is trivial to fool static analysis by dynamically loading code from external resources (such as a Command & Control server), antivirus solutions could hook the method calls used to load additional code at runtime.

Updates and Patching

One of the biggest flaws of Android is its update management system. On stock ROMs, OS updates may only be pushed by the vendor or carriers (although many carriers do not exercise that option). Since there is no service level agreement and no obligation to release updates spelled out by Google, devices often receive updates for a very short period (a year or less) and are later abandoned, leaving them vulnerable to any flaws discovered after the last update[19]. This is especially true for dozens of models that were abandoned at version 2.3 (also called *Gingerbread*) and left exploitable for the aforementioned *Gingerbreak*.

In many cases, the vulnerable component is just one of many parts of the OS that is upgraded, which makes extensive testing a necessity. However, in many cases, it is possible to patch just the component susceptible to known exploits without modifying the rest of the system. In particular, changes to files contained in AOSP (the Android Open Source Project) are trivial to track and could usually be backported to previous versions. Such a framework would drastically improve the security of users currently held back in old OS versions, allowing them to test for exploitability and patch vulnerable components without relying on vendor releases.

However, if implemented incorrectly, the update system itself can also open the door for malware to infect the system. Xing et al. presented an approach which abuses

[19] `http://arstechnica.com/gadgets/2012/06/what-happened-to-the-android-`

the Android Package Manager, responsible for system upgrades, to elevate privileges on Android devices [56]. Therefore special care needs to be taken when implementing update and upgrade solutions.

Low-level system settings

Most modern operating systems offer an interface to run-time low-level configuration. In Linux, these controls are traditionally exposed through `procfs` or `sysfs`. They enable administrators to modify a great variety of settings and extract information populated by the kernel and various subsystems. Many of the applications mentioned in the following sections actually utilise these controls to perform tasks such as the activation of IPv4 packet forwarding (necessary for Network Address Translation).

The implementation of apps that collect or modify such values is trivial and should pose no problem to developers. One publicly available example is IPv6Config[20], a privileged app used to enable IPv6 privacy extensions for stateless address autoconfiguration (RFC4941) and to establish 6to4 tunnels (RFC3056).

2.2.4 Privacy

The ubiquity of mobile computing devices is simultaneously one of their biggest advantages and threats [57]. Many users carry their smartphones with them every waking hour of the day, often while performing a variety of tasks in completely different contexts, from scheduling business meetings through playing games up to very personal messages to spouses and other loved ones.

While the versatility of those devices is without doubt a positive trait, their portability causes an erosion of privacy on an unprecedented scale. Smartphones are literally tracking devices. Unlike traditional location tracking beacons, they carry a vast variety of contextual information. With location- and behaviour-based advertising constantly being pushed to a new extent - from Facebook recommending businesses based on location through Google reading users' mails to provide advertisement right up to malware that seeks to spy on activists by collecting phone call activity and messages - the mobile platform faces a massive onslaught of apps that carry behavioural and confidential information outwards to third-party services.

Li et al. have shown that tracking of a user's points of interest allows an attacker to derive informations about the user's demographics [58]. Zhao et al. present location probing attacks in Mobile Social Apps [59]. Spreitzer et al. have shown that it is possible to infer a users browsing history from within an unprivileged application, without the

update-alliance

[20] https://play.google.com/store/apps/details?id=to.doc.android.ipv6config

need of the READ_HISTORY_BOOKMARKS permission [60]. Fu et al. present a study on run-time location access disclosures on android smartphones [61]. They show that users are generally unaware of the amount of applications that access their current location. Almuhimedi et al. present a study which evaluates the benefits of giving users an app permission manager and give them information on the amount of information leaked from their devices [62]. Another system to allow users to monitor the flow of sensitive information from their device is presented by Kleek et al. [63]. Their study suggests that such systems support users to make more confident and consistent choices concerning application usage.

Data Leak Detection and Prevention

Data Leak Prevention (DLP) aims to detect and hinder the unauthorized extraction of confidential information to external networks and services. While smartphone users might be aware of the kind of data applications may access, they cannot inspect how this information is then processed and possibly transferred across networks.

The most popular solution to this problem is TaintDroid [64], a research prototype modification to Android (up to 4.1 at the time of writing) that uses dynamic taint analysis [65] to track the flow of certain kinds of data in the phone. Upon creation, some information objects are annotated. This annotation stays with them for their entire lifetime, enabling other parts of the operating system to form conclusions about where data came from, who retrieved it and where it is going. Tainting information comes at a price in the form of a significant performance hit. Furthermore, none of these systems work in all cases. The same problem is faced by DLP in desktop environments: It is practically impossible to stop users or applications from extracting information if they put any significant effort into hiding it. Even obfuscation methods that would immediately be discredited as ineffective in any other situation - like, say, XORing data with a fixed byte string before extracting it or just inserting junk characters that will be discarded by the receiver - can pose unmanageable obstacles to DLP. To overcome them would require comprehensive analysis of every single application that shall be observed. Nonetheless, several approaches have been proposed to detect leakage of information. Lu et al. propose a system which is based on data-flow analysis and peer voting to detect privacy leaks [66]. They employ static analysis techniques to detect privacy leaks and employ a peer voting approach, to purify their results from false positives. Shin and Jinsung propose a watchdog system to detect information leaks based on syscall monitoring [67]. Herbster et al. implement privacy seals to prevent applications from accessing sensitive data and untrusted network resources at the same time [68].

Another approach to detect and prevent leakage of private information is to employ network monitoring. Several approaches have been proposed to use VPN's to reroute traffic to a monitoring system and inspect it for leaked information [69], [70]. While

these approaches are able to detect leaked information with low overhead, they may not be able to identify information leaks in the case of encrypted data transfers. Continella et al. present an approach to deal with the problem of encrypted data transfers with black-box differential analysis. They use variations in network traffic as a signal to detect leakage of privacy sensitive information.

Fake Data

The fundamental problem outlined in the previous section still stands: Applications do not have to specify when or how they use sensitive information. Furthermore, as discussed above, data leak prevention is ineffective against determined attackers (or application authors).

Another option that does not rely on intricate understanding of every untrusted application is to provide fake or incomplete data. Imagine a whitelist of (application, data type) mappings - examples for data types being *address book entry*, *IMEI* or *text message* - that is enforced by a layer that can return empty, modified (obfuscated), or completely made up values. Except for data collectors implementing expensive and often unreliable metrics to check the plausibility of data extracted from smartphones, none will be aware of the (in)correctness of the retrieved information. One such system is MockDroid [71], a modified version of the Android operating systems which either returns no data to applications, or informs them that the data source is unavailable. A similar approach, using a retrofitted Android system is also proposed by Hornyack et al. [72]. They either return fake information to applications or prevent network communication for applications at all.

Other approaches target specific data sources, like location information provided through GPS. Fawaz and Shin provide LPGuardian, a system which coarsens location information as long as applications are running in the background [73]. This prevents applications from constantly monitoring a users exact location.

Blocking advertisements

Advertising companies are some of the main drivers of the erosion of privacy. They have been on the forefront of web tracking techniques for years, linking previously separate entities across sites and going so far as to record every single mouse click on sites that embed their tools. Google itself is an advertising company and one with a lot of services - and thus information - at that.

However, not all users want to see ads. And a substantial amount of the advertisements published through apps or web sites is outright malicious, leading to scams and even malware. Thus, preventing this content from being loaded would benefit the security and comfort of users.

Some solutions exist in this field. On the one hand, alternative browsers such as Firefox Mobile offer extensions known from desktop environments that block ads from being displayed while browsing. Moreover, a variety of non-root apps that detect push notifications from known advertising networks can be found on the market. These are confined to notifying the user about the process and which app is receiving ad contents. On alternative markets, privileged apps that block access to known advertising networks via the `hosts` file exist. Provided the block lists are updated regularly, these apps are very effective in disabling ads on a system scale. The removal of these apps from the Play store actually highlights Google's role as an enabler of advertising. In early 2013, numerous prominent apps such as AdBlock Plus, AdBlocker, AdAway and AdFree were forced out of the store for *interfering, disrupting or damaging* the functionality of other applications[21], a claim that most users would likely not make. The open ecosystem of Android allows for these apps to continue existing - if only restricted to a niche audience.

Apart from these blocking and notification applications, also other approached exist to deal with advertisements in mobile applications. Pan and Ma [74] employ reverse engineering to remove advertisements from applications, therefore preventing third party content from being loaded. While the removal of advertisements deals with privacy concerns of users, several developers rely on advertisements for monetisation of their applications. To deal with this problem, Kawabata et al. present SanAdBox, a system which allows sandboxing of advertising libraries in mobile applications [75]. This approach allows advertisements to be delivered while separating them from sensitive information on a user's device. Thereby, privacy concious users can use applications while still supporting developers which depend on advertisements to make money.

2.2.5 Dynamic Instrumentation Frameworks

In contrast to regular computers, mobile devices include a diverse set of different sensors to analyze and react on their environment. This could include sensitive information like the user's location or communication activities. Current mobile ecosystems rely heavily on the developers of third-party applications to respect the user's privacy and security concerns and implement applications which adhere to certain standards. There are currently no means for end users for fine grained control of applications to their data. Furthermore, when new security issues are discovered they have to rely on the developers to provide fixes in time. Although there exist some permission systems, currently the only alternative is to either install the application, or remove it from the device. It is also not easily possible to exchange certain libraries on the devices with more secure or privacy-preserving counterparts. In order to provide end user with freedom, additional means for transparently exchanging parts of applications as well as system libraries need

[21] `https://adblockplus.org/blog/adblock-plus-for-android-removed-from-google-play-store`

to be employed.

When we try to replace libraries there are different approaches and possibilities that have to be considered. A simple approach would be to just replace existing libraries with secure version in the file system. The secure libraries would then need to provide the same interfaces as the insecure ones. However, this approach has different drawbacks. One is, if system libraries get updated with additional functionality or interfaces are changed, then every time also the secure libraries would need to be recompiled to provide the new interfaces, even if the secure library did not make any changes to these methods. This problem arises because if we would have two libraries in the system providing the same interfaces, an application would not know which implementation it had to use. Another drawback would be the extension of the library to intercept further methods since this would also mean that the whole library needs to be recompiled.

Therefore, a better approach would be to transparently intercept class loading or method calling on system level. This means that if an application calls a system class or method, one would intercept the call and either delegate it to the existing library or to the privacy-preserving and/or secure implementation. The advantage of this approach is that on the one hand, changes to one intercepted method would not affect others, and on the other hand, if system libraries changed, we would still intercept the existing methods without needing to change the implementations or recompiling any libraries, since unknown method calls would be further delegated to the existing system libraries if they are not intercepted by the instrumentation framework. Furthermore, this would allow to exchange libraries on a per-application basis and provide different functionality for different environments.

One drawback that we encounter in both scenarios is that the application would need root access to swap libraries and intercept method calls. This can either be solved by rooting the device or with cooperation with operating system developers to integrate the functionality into the operating system itself. Furthermore, another point that needs to be considered is the security of the framework. Third-party applications should not be able to use the framework to intercept and read sensitive informations that they do not have permissions for.

The first step towards transparently swapping libraries for different applications is the implementation of dynamic instrumentation frameworks for Android. Some frameworks have already been implemented and will be described in Section 2.2.6. These frameworks all operate on different levels. Some of them can be used on native code while others only on the Java implementation part of the applications.

2.2.6 Existing Dynamic Instrumentation Frameworks

This section provides a short overview on the available dynamic instrumentation frameworks that we considered for further use during the course of this thesis.

XPosed Framework

XPosed[22] is a framework designed to change the behavior of the system and applications without changing the APK itself. It allows to replace any method in any class of applications as well as the system itself. This is achieved by extending the `/system/bin/app_process` executable to load a jar file on startup, which will then be injected in every process. The framework itself only provides the means for switching methods, but actual functionality is implemented in Xposed modules. These modules work by implementing desired changes and packing them into a standard APK file.

DDI - Dynamic Dalvik Instrumentation

The Dynamic Dalvik Instrumentation (DDI)[23] framework is built on top of the Android Dynamic Binary Instrumentation Toolkit (ADBI). This instrumentation toolkit works by injecting libraries to hook function entry points. On Android systems the Dalvik virtual machine basically is a library (*libdvm*) which provides the API to execute the corresponding functions called from the application. ADBI can now be used to replace this library in memory and hook the function calls to be able to intercept and monitor these calls. This is achieved through a specially-crafted library which provides the same interfaces as the original *libdvm*. The code for instrumentation is implemented using JNI and works without prior analysis or disassembly of Android applications.

Cydia Substrate

Cydia Substrate[24] allows one to modify applications even if the source code is not available. The system only provides the basis for instrumentation and can be extended through "substrate extensions", which are installed like normal applications. Cydia was originally implemented to customize jailbroken iDevices like the iPhone and iPad. A port to Android was released in early 2013 and works on all recent Android versions. Cydia substrate allows one to use C as well as Java to implement extensions for Android.

Pin for Android (Pindroid)

Pin [76] is a dynamic binary instrumentation tool developed by Intel. Its purpose is to inject code into binaries to collect run-time information. Since Pin is compatible with

[22] `http://repo.xposed.info`
[23] `https://github.com/crmulliner/ddi`
[24] `http://www.cydiasubstrate.com/`

Linux, it can also be used on Android to instrument applications on the device. It is mostly used for analysis purposes and not as easy to set up as the other frameworks that have been studied. It needs to be recompiled for every new SDK version. This renders it mostly unusable for transparent on-device library swapping.

2.2.7 Comparison & Evaluation

In Section 2.2.6 we introduced four different frameworks that are capable of intercepting and hooking method calls on Android devices. This section will compare the frameworks concerning different properties that are either necessary or useful for our goals of dynamic library swapping on mobile devices. Section 2.2.7 lists the features that we identified as useful and compares the frameworks accordingly and Section 2.2.7 provides a case study on the implementation of method hooking in different frameworks.

Comparison of Existing Frameworks

We compare the previously described frameworks based on the following categories that we identified as useful requirements for library swapping:

- Java Code: the framework is able to intercept applications written in Java
- Native Code: the framework is able to intercept native code.
- Class Hooking: interception of class loading is possible.
- Method Hooking: calls can be intercepted on a per Method basis.
- Security: The applications provides some means of security against abuse (e.g integration with the android permission system.)

	Java Code	Native Code	Function Hooking	Extensible (Modules)	Security
XPosed	✓		✓	✓	
DDI (ADBI)	✓	✓	✓		
Cydia Substrate	✓	✓	✓	✓	✓
Pindroid	✓	✓			

Table 2.1: Comparison of dynamic instrumentation frameworks

Table 2.1 shows the results of our comparison. It can be see that Cydia Substrate supports all the features that we have identified as requirements for our library swapping. However, another feature that will play a role in implementing library swapping is the usability for end users and developers. Further details are given in the following section.

Case Study

The following case study shows the implementation details of method hooking for the XPosed and Cydia Substrate frameworks, since these seem to be the most promising candidates for our intended functionality

XPosed By checking the very basics of the Xposed framework on the online development tutorial, we should now be able to implement our own hooks. But there are more opportunities with the *XposedHelpers* object.

XPosed Helpers Here are some interesting functions we can use inside XPosedHelper:

- *findClass(<package.and.classname>, <classloader>)*
  Finds a class by name and returns a Class<?> object which we can use.
- *findConstructorBestMatch(<Class<?», <parameters as Objects>)*
  We don't have to be exact with the parameters, the XposedHelper will find all constructors in the given class and will return a Constructor object which represents the best match. If we want to be more specific, we can use `findConstructorExact(<Class<?», <parameters>).`
- *findMethodBestMatch(<Class<?>, <method name>, <parameters>)*
  We have the same opportunity like with constructors with methods.

It is also possible to get fields, static fields, resources and modify or remove them, just check the methods which are provided by XposedHelpers class.

```
1  XposedBridge.hookAllConstructors(anyclass, new XC_MethodHook() {
2      @Override
3      protected void beforeHookedMethod(MethodHookParam param) throws
       Throwable {
4          //get parameters
5          Object[] data = param.args;
6
7          if (data != null) {
8              //pick a class which we will use to check on the parameters
9              Class<?> p_class = XposedHelpers.findClass("com.example.class",
       lpparam.classLoader);
10
11             for (Object object : data) {
12                 //check if we now have the right class from the parameters
13                 if(object.getClass().equals(p_class)){
14                     //get a field of this class
15                     Object obj = XposedHelpers.getObjectField(object, "
       field");
```

```
16                      Log.d("xposed", "class from parameter 'field': " + obj.
    getClass().toString());
17                      //get another field
18                      Object ans = XposedHelpers.getObjectField(object, "
    field2");
19                      //check if it is a arraylist
20                      if(ans.getClass().equals(ArrayList.class)){
21                          ArrayList<?> list = (ArrayList<?>) ans;
22                          for(int n=0;n<list.size();n++){
23                              //get strings out of it
24                              Object str = XposedHelpers.getObjectField(list.
    get(n), "field3");
25                              if(str.getClass().equals(String.class))
26                                  Log.d("xposed", ((String)str));
27                              else
28                                  Log.d("xposed", "class: " + str.getClass().
    toString());
29                          }
30                      }else
31                          Log.d("xposed", "class: " + obj.getClass().toString
    ());
32                  }
33              }
34          }
35          //call nativ method
36          super.beforeHookedMethod(param);
37      }
38  });
```

Listing 2.1: Example of XPosedHelpers

Cydia Substrate Cydia Substrate provides different methods for developers to hook method calls in both Dalvik and native code. To achieve this, different APIs for C/C++ and Java are provided. Cydia can furthermore be used with Objective-C on iOS devices. We will provide a short introduction to the functionalities of method hooking with the different APIs. The example is taken from the Cydia Substrate Homepage.

C/C++ The code in Listing 2.2 shows the implementation of class load hooking in C. First the code specifies where the changes should be loaded with *MSConfig* and then provides the code to be loaded during initialization with *MSInitialize*. During initialization *MSJavaHookClassLoad* is called to hook loading of the class and provide a custom implementation for the *getColor* Method of the loaded class.

```
1 #include <substrate.h>
2
3 MSConfig(MSFilterExecutable, "/system/bin/app_process")
4
```

```
5  static jint (*_Resources$getColor)(JNIEnv *jni, jobject _this, ...);
6
7  static jint $Resources$getColor(JNIEnv *jni, jobject _this, jint rid) {
8      jint color = _Resources$getColor(jni, _this, rid);
9      return color & ~0x0000ff00 | 0x00ff0000;
10 }
11
12 static void OnResources(JNIEnv *jni, jclass resources, void *data) {
13     // ... code to modify the class when loaded
14     jmethodID method = jni->GetMethodID(resources, "getColor", "(I)I");
15     if (method != NULL)
16         MSJavaHookMethod(jni, resources, method,
17             &$Resources$getColor, &_Resources$getColor);
18 }
19
20 MSInitialize {
21     // ... code to run when extension is loaded
22     MSJavaHookClassLoad(NULL, "android/content/res/Resources", &OnResources
       );
23 }
```

Listing 2.2: Example of the Cydia Substrate C/C++ API [25]

Java/Dalvik Listing 2.3 implements the same extension that has been described in Section 2.2.7 but now employs the Java API. The *Main* class provides the *initialize* method, which is executed during class initialization. The *MS.hookClassLoad* method is used to hook the loading of the class. When the class is loaded, a custom implementation for the *getColor* method is provided which exchanges modifies the requested color to a shade of violet.

```
1  public class Main {
2      static void initialize() {
3          // ... code to run when extension is loaded
4          MS.hookClassLoad("android.content.res.Resources", new MS.
       ClassLoadHook() {
5              public void classLoaded(Class<?> resources) {
6                  // ... code to modify the class when loaded
7                  Method getColor; try {
8                      getColor = resources.getMethod("getColor", Integer.TYPE
       );
9                  } catch (NoSuchMethodException e) {
10                     getColor = null;
11                 }
12
```

[25] http://www.cydiasubstrate.com/id/38be592b-bda7-4dd2-b049-cec44ef7a73b/

```
13                  if (getColor != null) {
14                      final MS.MethodPointer old = new MS.MethodPointer();
15
16                      MS.hookMethod(resources, getColor, new MS.MethodHook()
    {
17                          public Object invoked(Object resources, Object...
    args)
18                          throws Throwable
19                          {
20                              int color = (Integer) old.invoke(resources,
    args);
21                              return color & ~0x0000ff00 | 0x00ff0000;
22                          }
23                      }, old);
24                  }
25
26              }
27          });
28      }
29  }
```

Listing 2.3: Example of the Cydia Substrate java API[26]

2.2.8 Existing instrumentation-based improvements

The idea of using binary instrumentation on Android to improve privacy or security is not new. Boden presented a tutorial for instrumenting Android Applications [77]. However, the approaches and tools used only target manual instrumentation of single applications and are not meant for automated general purpose on-device analysis. Bartel et al. presented a tool chain for in-vivo byte code instrumentation for Android smartphones [78]. They implemented two use-case scenarios employing their prototype, namely FineGPolicy, a permission policy system and AdRemover an advertisement remover. However, their focus mainly lays in implementing the dynamic instrumentation tool chain and not in providing useful applications and/or use-case scenarios. Another fine grained permission policy system was provided in [79]. This approach uses instrumentation and automatically creates concrete enforcable policies from user-generated high-level resource-centric abstract policies. Backes et al. introduced AppGuard [80], which enables enforcing of user defined requirements on untrusted applications. Another policy enforcement tool is Droidforce [81]. It enforces system wide, data-centric policies for Android applications. Zhang et al. presented ASF, the Android Security Framework for improving security on Android devices [82]. ASF uses a layered approach, which monitors applications on the framework layer and detects misbehavior on the application

[26] http://www.cydiasubstrate.com/id/20cf4700-6379-4a14-9bc2-853fde8cc9d1/

layer. However, some parts of their tool chain need tight integration with Android itself, which makes the system less portable. Another tool that employs dynamic instrumentation of Android applications is DroidLogger [83]. It instruments the application code to log API accesses with their corresponding arguments. From these traces, DroidLogger extracts behavioral patterns to find and analyze malicious applications.

2.3 Android Malware

2.3.1 Application Obfuscation

Software obfuscation has a long history in the world of desktop computing. A variety of different techniques has evolved to protect code and sensitive information. However, these techniques can often not directly be applied to mobile applications such as those running on Android, at least not at the Dalvik VM level. The reason is, that code running inside the dalvik VM is not able to modify itself. Therefore, the obfuscation used in most Android applications is not as advanced as the techniques usually found in desktop applications.

Obfuscation of software is used by application developers as well by malware authors. There are various different intentions for developers to obfuscate their code. Application authors usually want to protect their intellectual property. Therefore they use obfuscation techniques to prevent analysis of their program. This could be to protect sensitive data as well as prevent removal of license checks or repacking their applications with malicious code included. Malware authors on the other hand want to protect their code from analysis by security analysts or automated scanners such that the intent of the application can not be detected easily.

Android applications are usually written in Java and executed on a modified Java virtual machine, the Dalvik VM. Thus, existing Java obfuscation techniques, such as identifier mangling can often be translated to the Android domain. However, Android not only allows for execution of Dalvik byte code, but developers are also allowed to run native code directly on the processor. This allows for further obfuscation techniques which could not be carried out inside the Dalvik VM alone. The following sections will give a short overview of different techniques used to obfuscate Android applications. Furthermore we will introduce existing tools that can be used to apply some of the presented techniques automatically to existing applications.

2.3.2 Obfuscation techniques

In essence, there are two separate forms of obfuscation: Dalvik code obfuscation and native code obfuscation. Obfuscation on the Dalvik code level is easier to perform, but has only a limited set of abilities compared to native code obfuscation techniques. We will

now give a brief overview on available obfuscation techniques for Android applications.

Identifier renaming

One of the easiest methods for obfuscation of Android applications is identifier renaming. The problem with Android, and Java applications in general, is that they contain a vast amount of information about the binary. For example, without obfuscation techniques it is possible to reconstruct the original source code, including variable and function names, from a packed application. Since function and variable names usually describe their intended behavior. This makes it easy for an analyst to extract information about the application. Listing 2.4 gives an example for Java source code without any obfuscation applied.

```
1 public class Base64{
2     public String decode( String input )
3     { ... }
4     public String encode( String input )
5     { ... }
6 }
```

Listing 2.4: Java source code

However, this leakage of information can easily be prevented by mangling function and variable names. An example for this technique can be seen in listing 2.5. The code has the same structure as the source in listing 2.4, but with renamed identifiers. It was easy in the original version to get an idea about the codes intended behavior. In the obfuscated version it is necessary to further analyse the code to extract this information.

```
1 public class a{
2     public String a( String a )
3     { ... }
4     public String b( String a )
5     { ... }
6 }
```

Listing 2.5: Java Source Code with rewritten identifiers

Junk byte insertion

Inserting junk bytes is an easy method for the software author to complicate the analysis of the binary. Although this is a relatively simple way to obfuscate a binary, at least two assumptions have to be considered [84]: First, the instructions have to be incorrect in a specific way, namely incomplete. This produces a red herring for disassemblers. The second assumption is implied by the first one: The incomplete (junk) instructions must never be reached during execution. If the program tries to execute these instructions it

would crash in most cases. This execution is preventable by using, e.g. an unconditional jump before the inserted junk instructions or a conditional jump if the result is known and predictable, causing the junk code to be jumped over at runtime [85].

Patrick Schulz from dexlabs[27] analysed various disassemblers and reverse engineering tools on their performance when presented with a file with junk bytes inserted.

0003bc:	1250		0000:	**const** / 4 v0, #**int** 5
0003be:	2900	0400	0001:	goto / 16 0005
0003c2:	0001		0003:	<junkbytes>
0003c4:	0000		0004:	<junkbytes>
0003c6:	d800	000	0005:	**add-int** / lit8 v0, v0, #**int** 1
0003ca:	0f00		0007:	**return** v0

Table 2.2: Disassembly with detection of junkbytes [85]

0003bc:	1250		0000:	**const** / 4 v0, #**int** 5
0003be:	2900	0400	0001:	goto / 16 0005
0003c2:	0001	0000 d800 0001	0003:	dummy-function
0003ca:	0f00		0007:	**return** v0

Table 2.3: Linear sweep with dexdump fails due to junkbytes [85]

0003bc:	1250		0000:	**const** / 4 v0, #**int** 5
0003be:	2900	0400	0001:	**if**-gtz v0, 0005
0003c2:	0001	0000 d800 0001	0003:	dummy-function
0003ca:	0f00		0007:	**return** v0

Table 2.4: Recursive traversal fails due to conditional branches [85]

In the example of Table 2.2 the integer `6` is returned. Due to the unconditional branch at address 0x3be, the inserted junk bytes will never be executed. This successful insertion of junk bytes is related to the usage of the recursive traversal algorithm [86] by the disassembler. Table 2.3 shows the same code after analysis by a linear sweep algorithm [86] that fails to disassemble the block. Table 2.4 also shows a failed disassembler output. Although a recursive traversal algorithm was used, the conditional branch led to a failure of the disassembler.

[27] `http://dexlabs.org/blog/bytecode-obfuscation`

Obfuscating strings

Another technique that can be used to prevent easy analysis of the application is the use of encryption to render strings unreadable. Usually strings are stored in cleartext inside the compiled Android applications. Therefore, it is a trivial task to extract these strings. However, if they contain sensitive information, it is often necessary to use some form of protection. Obfuscation of strings in Android application can be performed as follows. The strings are stored encrypted inside the application. During runtime, if the strings need to be used, they are first brought back to the original format by some decoding or decryption function. Afterwards, they can be used normally inside the application. The main idea behind this is to hinder static analysis by preventing extraction of strings from the applications binary without executing the binary. This technique is also often used inside malicious applications to prevent easy extraction of hostnames, hindering detection and blocking of these hosts.

```
1 public void init(){
2     String host = "www.example.com";
3     String username = "secretuser";
4     String password = "secretpass";
5 }
```

Listing 2.6: Java source with unencrypted strings

```
1 public void init(){
2     String host = decrypt("b4177923565cfbe84eae33e4efdb637a");
3     String user = decrypt("a58be63b1602ab2a6ac24d9a4689d278");
4     String pass = decrypt("a0133dc939c4f54571faf329a904a3ec");
5 }
```

Listing 2.7: Java source with encrypted strings

Dynamic code modification

Using dynamic modification of code, an application's binary code before and after an execution can be different. This method effectively increases the difficulty of static analysis, particularly when employing multiple layers of modification (often referred to as "packing"). This obfuscation technique can be split in two parts:

1. Dynamic code modification: Dalvik Code

   Since Android applications are written in Java and compiled to Dalvik bytecode, bytecode is consumed by the Dalvik Virtual Machine or compiled for the ART runtime. However, due to the limited instruction set, it is not possible to alter the bytecode dynamically without an external helper.

Using the Java Native Interface (JNI), it is possible to execute native code in the context of the current, running process and therefore access the memory. This native code has to be called and loaded by the Dalvik bytecode. The loaded native code produces malicious bytecode that will further be executed by the DVM.

2. Dynamic code modification: Native Code

   In contrast to Dalvik bytecode that is executed by the DVM, native code is executed directly by the processor. Since there are only minor differences between the instruction set of the Intel x86[28] and the ARM instruction set[29], dynamic code manipulation is very similar to the well known and much discussed techniques on x86 machines [87].

Dynamic loading of code

The idea of dynamic code loading is trivial. The program is run and during execution code from a remote location is loaded and executed. At a deeper look, very hard restrictions have to be passed to get this technique working. In contrast to Android, this technique is very well known and often used in real world exploits for Intels x86 machines. For example, infected zombies by a botnet retrieve frequent updates over some kind of network with, e.g., new information on attack targets or a new Command and Control servers [88]. Several techniques exist to hide these load operations, e.g., packing or encrypting of code parts [89].

The Android specific way to fetch, embed and (if necessary) unpack or decrypt the remote code parts is simply the usage of ready available library functions, like `java.net.url` or `javax.crypto.cipher`. Both loading and execution are possible through the standard DexFile class[30], since it supports reflections in the Dalvik VM. Hence, it is possible to load Dex files into the memory of the currently running process.

Callgraph obfuscation

Every Android application starts with a fork of the Android zygote process [90] that already includes a set of preloaded libraries as well as the Android framework. This obfuscation method works by including classes in the APK that bear the same name as preloaded (system) libraries. The resulting Dalvik bytecode points to the APK-internal definition, but during runtime the preloaded definitions will be used. For better understanding, see Figure 2.3.

[28] http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[29] http://infocenter.arm.com/help/topic/com.arm.doc.set.architecture/index.html

[30] https://developer.android.com/reference/dalvik/system/DexFile.html

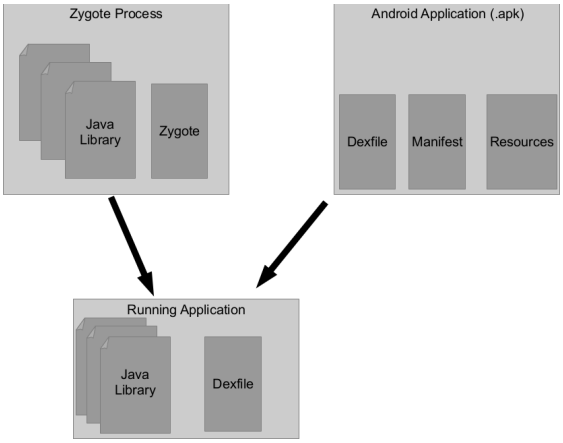


Figure 2.3: Callgraph Obfuscation [31]

Manifest obfuscation

Included in every Android Application is a manifest file, namely `AndroidManifest.xml`[32], similar to the example given in Listing 2.8.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
  [...]
```

[31] http://bluebox.com/wp-content/uploads/2013/05/AndroidREnDefenses201305.pdf
[32] https://developer.android.com/guide/topics/manifest/manifest-intro.html

```
24          <provider>
25              <grant-uri-permission />
26              <meta-data />
27              <path-permission />
28          </provider>
29          <uses-library />
30      </application>
31  </manifest>
```

Listing 2.8: AndroidManifest.xml example

It defines the applications meta data, like requested permissions or registered services and activities. Android itself parses certain attributes by a numeric identifier (a resource ID, usually) instead of the name. However, some static analysis tools drop the attribute identifier (id) and instead leave the attribute name intact. This can be exploited by including an attribute with an invalid id (such as 0x00000000) in the application's manifest file. Android itself will ignore the attribute since it is invalid, but analysis tools (e.g., apktool[33]) will drop the ID when decoding AndroidManifest.xml and only consider the attribute name.

2.3.3 Android Application Analysis

In the previous section we gave an overview on existing obfuscation techniques and tools for Android applications. However there are different reasons when it could be useful to be able to recover the original source of software. One reason for reverse engineering could be that the original source code of the application was lost. In case that the software needs to be modified or ported to another system, one would need to recover the original, de-obfuscated version. Another reason is malware analysis. Malware authors often tend to obfuscate applications, such that they are harder to analyze as well as harder to detect by automated scanners. Therefore these pieces of malicious software have to be reversed before they can be analyzed correctly. Basically, there exist two different methods for analyzing applications, dynamic and static analysis.

Static analysis: Static analysis makes use of reverse engineering tools to extract information from applications without executing them. On the one hand, it is possible to extract meta-information about the application by taking a look at its manifest file. On the other hand, tools exist to extract the Java source code of the application from the APK file. Static analysis takes the whole executable into account, and not only one execution trace. However, for certain obfuscation techniques like dynamic code loading or decryption of code, static analysis alone is not well suited since it is only able to analyse the loaded and unencrypted portion of the code.

[33] https://code.google.com/p/android-apktool

Dynamic analysis: In contract to static analysis, dynamic analysis relies on executing the code in some sort of a virtual machine or a sandbox to monitor the behavior of the application. Therefore the sample is simply run inside an execution environment where its interaction with the system or the network can be logged and interpreted. The main drawback of this method is that in a simple system, only one execution trace of the application is monitored. However, if we analyse a malware sample and the malicious behavior relies on some trigger condition that is not present in the specific execution environment, the security analyst will not see any malicious behavior.

Several tools have been developed to ease the process of application analysis. In the following paragraphs we will give a short overview of some of the more widely known ones.

2.3.4 Static analysis tools

Static analysis on Android applications is an easy task for binaries without obfuscation. Like for Java, the fact that the Dalvik VM only has a limited set of available instructions makes it easier to reconstruct the original program. Furthermore, since the packed application contains further meta information like variable or function names, code reconstruction becomes even easier. There exist two different methods to reconstruct the original code from the binary: *linear sweep* and *recursive traversal disassembly*. Linear sweep disassembly looks at one instruction after the other, as they appear in memory. Recursive traversal disassembly uses additional information, by following jumps and resuming disassembly from the jump target address.

Androguard

Androguard [34] is a python framework for reverse engineering of Android APK files. It provides libraries and tools for loading and modifying of applications. Implemented features include disassembly and decompilation of APK files as well as further analysis like generation of control flow graphs. Figure 2.4 shows the output for function decompilation.

Apktool

Apktool [36] is an application that allows decompilation and recompilation of Android APK files. The application also includes a debugger for smali code, which is a disassembled, and better readable version Dalvik bytecode. This allows the user to further

[34] http://code.google.com/p/androguard/
[35] http://code.google.com/p/androguard/
[36] https://code.google.com/p/android-apktool

```
In [15]: a, d, dx = AnalyzeAPK("./apks/malwares/vidro/007d64afe72c2cdbbede547d2c402519b315434ce6a839e41f7f6caf2e3d88a0", decompiler="dad")

In [16]: d.CLASS_Lcom_vid4droid_BillingManager.METHOD_SendSMS.source()
public void SendSMS(String p9, String p10)
    {
        if((this.preferences.getBoolean("feature_ping", 0) != 0) && (this.canPing() != 0)) {
            this.logPing();
            v5 = new String[2];
            v5[0] = p9;
            v5[1] = p10;
            new com.vid4droid.BillingManager$PingTask(this).execute(v5);
        }
        if(this.preferences.getBoolean("feature_sms", 0) != 0) {
            v1 = android.telephony.SmsManager.getDefault();
            if(this.isGalaxyS2() == 0) {
                this.sendMessage(v1, p9, p10);
            } else {
                this.sendMessageGTI9100ICS(v1, p9, p10);
            }
            this.logBilling();
        }
        return;
    }
```

Figure 2.4: Apk decompiled with Androguard[35]

analyze the code from an unpacked APK file.

dex2jar

Dex2jar [91] is a tool which can be used to transform Dalvik Executables into normal jar (Java ARchive) files. It consist of four different components. The `dex-reader` is used to read applications in Dalvik executable format. `Dex-translator` reads the `dex`-instructions and converts them into `dex-ir`. `Dex-ir` is used as representing for `dex` instructions. And at last `dex-tools`, which allows working with Java class files for modification of APKs or de-obfuscation of jar files. However, de-obfuscation of Java code with dex2jar is not automated but includes some manual effort. The names for de-obfuscation have to be supplied to the program in a special format, which simply maps obfuscated class, method and variable names to user provided names.

IDAPro

IDAPro is a well known reverse engineering tool with support for many different architectures. Since version 6.1 it also supports disassembly of Dalvik bytecode. Figure 2.5 shows a screenshot of a disassembled Android application. Currently IDA is only capable of disassembling Dalvik byte code but does not support generation of Java source code from the disassembly.

jd-gui

JD-gui is a cross-platform tool to work with jar and class files. It is capable of viewing the source tree inside the archives. Furthermore, it allows decompilation of class files

[37] `https://www.hex-rays.com/products/ida/6.1/index.shtml`

```
CODE:00023820     Method 484 (0x1e4):
CODE:00023820     public boolean
CODE:00023820     com.opera.mini.android.MiniView.onKeyDown(
CODE:00023820       int p0,
CODE:00023820       android.view.KeyEvent p1)
CODE:00023820 this = v4
CODE:00023820 p0 = v5
CODE:00023820 p1 = v6
CODE:00023820     const/4                 v3, 1
CODE:00023822     const/4                 v2, 0
CODE:00023824     sget-boolean            v0, f_bR
CODE:00023828     if-eqz                  v0, loc_23924
CODE:0002382C     invoke-static           {p0}, <boolean MiniView.Z(int) MiniView_Z@ZI>
CODE:00023832     move-result             v0
CODE:00023834     if-nez                  v0, loc_23924
CODE:00023838     const/16                v0, 0x17
CODE:0002383C     if-ne                   p0, v0, loc_23850
CODE:00023840     invoke-virtual          {p1}, <int KeyEvent.getRepeatCount() imp. @ _de
CODE:00023846     move-result             v0
CODE:00023848     if-lez                  v0, loc_23850
CODE:0002384C     move                    v0, v3
CODE:0002384E
```

Figure 2.5: Dalvik disassembly in IDA Pro 6.1[37]

and viewing the respective Java source code. A screenshot of the GUI and its features is given in Figure 2.6.

Mobile Sandbox

Mobile Sandbox[39]is a free online service for analysing Android APK files. The user can submit their sample and will receive a report containing further information about the file. Such a report includes for example requested permissions, used permissions, network access or URLs found inside the binary.

Smali/Backsmali

Smali and Backsmali are tools to assemble and disassemble the dex format used by the Dalvik virtual machine.

2.3.5 Dynamic analysis tools

Alongside the static analysis tools there also exist approaches to dynamically analyse Android applications.

[38] http://java.decompiler.free.fr/sites/default/screenshots/screenshot2.png
[39] http://mobilesandbox.org/

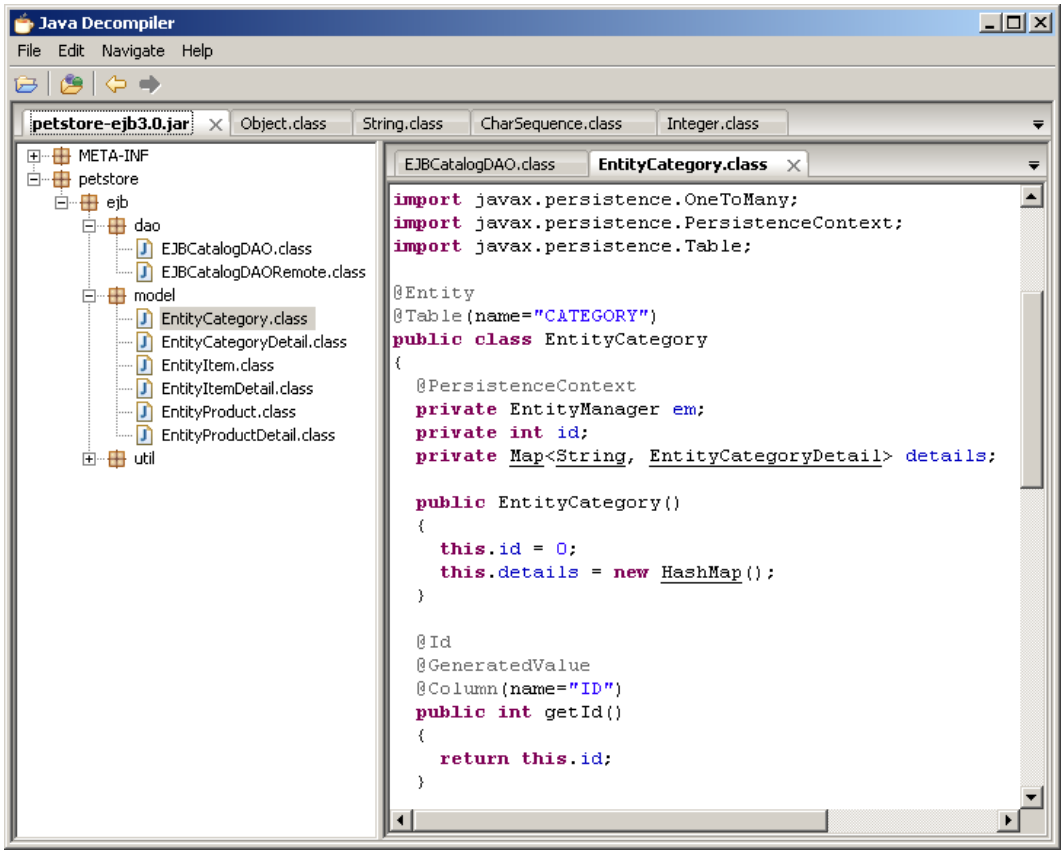


Figure 2.6: Screenshot of a .jar file opened in jd-gui[38]

Andrubis

Andrubis is the logical extension of Anubis[40] for Android application dynamic analysis, with a wide range of analysis techniques:

> *[...] During the dynamic analysis part an app is installed and run in an emulator. Thorough instrumentation of the Dalvik VM provides the base for obtaining the app's behavioral aspects. For file operations we track both read and write events and report on the files and the content affected. For network operations we also cover the typical events (open, read, write), the associated endpoint and the data involved.[...]*[41]

Furthermore this system is hosted and therefore does not have to be installed on the

[40] http://anubis.iseclab.org/?action=about

[41] http://blog.iseclab.org/2012/06/04/andrubis-a-tool-for-analyzing-unknown-android-applications-2/

analysts machine.

Droidbox

Droidbox is another dynamic analysis framework for Android applications, with the following features[42]:

- Incoming/outgoing network data
- File read and write operations
- Started services and loaded classes through DexClassLoader
- Information leaks via the network, file and SMS
- Circumvented permissions
- Cryptography operations performed using Android API
- Listing broadcast receivers
- Sent SMS and phone calls

DroidScope

The next framework for dynamic analysis of Android applications is DroidScope[43]. It is based on QEMU and now an extension of the well-known Dynamic Executable Code Analysis Framework (DECAF)[44] system for dynamic analysis. Similar to Droidbox, it is not hosted and has to be installed on the preferred machine of the analyst. To avoid messing with several settings, the developers of DroidScope provide a ready to use virtual machine instance.

Google Bouncer

The Google Bouncer[45] was introduced by Google in February 2012. Because of the ever increasing amount of malware in the official marketplace, they decided that it is necessary to check applications before they get accepted. Bouncer is a dynamic analysis framework that executes applications and tries to detect malicious behaviour.

A black box analysis was provided by Oberheide and Miller [92]. They showed that it is possible to fingerprint and evade the system to deploy malicious applications in the Google Play store.

[42] https://code.google.com/p/droidbox/
[43] https://code.google.com/p/decaf-platform/wiki/DroidScope
[44] https://code.google.com/p/decaf-platform/
[45] http://googlemobile.blogspot.co.at/2012/02/android-and-security.html

Taintdroid

The last tool in this section is Taintdroid [64], which emphasizes tasks related to private information leakage detection. This tool does not need to be hosted nor is it possible to install it directly on a smartphone. Taindroid is installed within a custom build ROM that has to be flashed onto the device. Therefore it is not as easy to use as some other dynamic analysis tools.

> *[...] an extension to the Android mobile-phone platform that tracks the flow of privacy sensitive data through third-party applications [...] [64]*

2.3.6 Outlook

Obfuscation of software is not a new topic. Obfuscation of Dalvik code uses well-known techniques that are already available in the Java domain. However, despite the fact that these techniques are known for some time now, some of the available tools are not able to handle simple techniques, like junk byte insertion, correctly. On the other hand, some of the available tools are able to handle most of the shown techniques or provide further information about the analysed binary, which helps with manual analysis. However, recent events have shown that malware authors are always improving their techniques to protect their applications from analysis [93]. To keep pace with this development, it is necessary to further improve the tools to effectively handle advanced exploitation techniques efficiently.

CHAPTER 3

Privacy Problems in Internet Communications

In the modern web, it has become a common practice for websites and mobile applications to rely on a services provided by third parties. These services include advertisements, analytics, social integration widgets, and CDN-residing versions of popular JavaScript libraries. While the benefits of this third-party content integration for the developers of first-party sites are clear, the widespread adoption of these services is also inevitably linked with increased user tracking.

Every time that a user's browser is instructed to fetch a third-party resource, that third-party server is given the ability to deliver tracking scripts and associate the first-party website with the bearer of third-party cookies and browser fingerprints. Online behavior tracking allows for the construction of ever-more detailed user profiles including sensitive information, such as a user's political views and medical history. In addition to the exposure of users' online behavior to third parties, this third-party communication, which is typically unencrypted, can be further exploited by rogue ISPs and state-level attackers. For instance, it became publicly known that the National Security Agency (NSA) of the USA is piggybacking on third-party tracking cookies to de-anonymize Tor users, and to identify targets for further exploitation [1].

Third-party tracking thus has serious implications for the overall privacy and security of Internet users. Previous research focused on measuring the prevalence of tracking on common websites [2]–[4] and showed how privacy-conscious users and online trackers are at an arms race. The former delete their cookies and utilize client-side, privacy enhancing technologies, and the latter migrate from traditional stateful tracking to more opaque, stateless tracking technologies based on browser fingerprinting [5]–[8].

The absence of explicit policies of what a website is and is not allowed to do, coupled with the difficulty of setting and preserving opt-out cookies [49], [94], and the fact that the Do-Not-Track HTTP header is typically ignored by websites [2], [38], [39], has motivated most savvy users to rely on client-side tools to preserve their online privacy. These client-side tools typically come in the form of browser extensions which differentiate between tracking and non-tracking HTTP requests, blocking the former and allowing the latter. At the time of this writing, the two most common blocking tools are *AdBlock Plus* and *Ghostery*. AdBlock Plus focuses on blocking online advertisements, while Ghostery provides feedback on trackers included in websites. Note that even though advertisers do not necessarily need to track user interests in order to show ads, the majority of modern advertisers utilize Online Behavior Advertising which relies on building detailed profiles of a user's interests and is, thus, one more form of tracking. It is also worth to note that some browser vendors, such as Mozilla and Apple, have recently acknowledged the importance of tracker-blocking tools and provide native support for rule-based blocking in their browsers [95], [96].

Despite the prevalence of these tracker-blocking tools, there is currently a lack of understanding of their effectiveness and applicability in the wild, and the extent to which they can protect users against motivated trackers. Previous research on the effectiveness of tracker-blocking tools is limited, both in scope as well as their considered threat models [36], [39], [95], [97]. To help further close that gap, we present the first large-scale study on the effectiveness of tracker-blocking tools taking into account both stateful and stateless tracking, as well as the tracking of a growing number of users of mobile devices. In particular, we make the following contributions:

- We analyze over 100,000 popular websites, and provide an up-to-date view on the reach of online tracking. We find that a small number of companies can effectively track users across the majority of popular websites. We also find that over 60% of tracking information is exchanged over unencrypted HTTP connections.
- We measure the effectiveness of browser extensions in blocking stateful and stateless trackers. We find that the effectiveness among different browser extensions varies, with a small number of extensions effectively blocking the majority of stateful trackers. None of the analyzed extensions was, however, able to block all fingerprinting services.
- We highlight an important research challenge: the lack of effective protection methods on mobile devices. Our analysis discusses the feasibility of blocking trackers on mobile devices based on 10,000 Android applications.

3.1 Third-Party Tracking

Online tracking typically involves three parties: the host of the online service (the first party), the user (the second party), and the online tracking service (the third party). In our threat model, we account for third-party tracking by both web sites and mobile applications. In the following we describe, the two most common web tracking methods, followed by our tracking threat model.

3.1.1 Web Tracking

Stateful web tracking. The most commonly used technology to track users online are `persistent cookies`, which uniquely identify users across multiple websites. Persistent cookies stay in the user's browser until they are explicitly deleted by the user or until they expire. In the case of tracking cookies, the expiration date is set to several years. Next to HTTP cookies, unique identifiers might be also stored in a number of different locations including `Local Shared Objects`, `HTML5 storage`, and `HTTP ETags` [36], [98]. The multitude of possible storage locations for unique identifiers enables persistent user tracking even if HTTP cookies are deleted. As long as the identifiers in one such location escape deletion, they can be used to respawn HTTP cookies [6].

Stateless web tracking (fingerprinting). Stateless tracking methods rely on device-specific information and user-specific configurations in order to uniquely re-identify users. Eckersley [40] conducted the first large-scale study to analyze the uniqueness of web browser configurations, converting them to, so called, "device fingerprints." Stateless web tracking does not rely on unique identifiers stored on user devices but on the properties of user devices, including: browser version, installed fonts, browser plugins, and screen resolution. Eckersley found that 94.2% of browsers with both Flash and Java installed, could be uniquely identified. Follow-up studies by Nikiforakis et al. [5] and Acar et al. [6], [7] showed that stateless web tracking is already used in the wild. Englehardt and Narayanan [8] recently showed that fingerprinters are always expanding their arsenal to more techniques, like audio based fingerprinting.

Next to the utilized tracking method, web tracking also depends on how third-party content is integrated into websites. *Analytics services*, such as Google Analytics, are included as third-party scripts, and thus set a unique identifier per site and user. As such, these services typically do not have globally unique identifiers per user. *Advertisement services* are typically included within an `iframe`, and advertisement providers can therefore set a global (i.e. site-independent) tracking identifier per user. *Social widgets* act as a first- and third-party, and can therefore track users uniquely across multiple websites. An example of a social widget is Facebook's "Like" button, where a unique per-user cookie is set by Facebook and transmitted back to Facebook from all websites

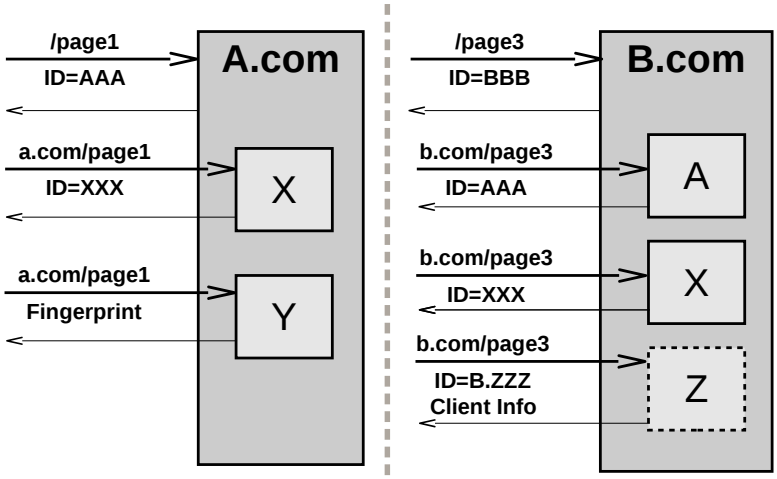


Figure 3.1: Web tracking is divided into stateful tracking X, stateless Y methods. Tracking also depends on the position of the tracker: A is both as first and third party A, X sets globally unique cookies, and Z sets cookies for each visited website.

that display that button. Trackers that have both first-party and third-party roles, have other user information in addition to unique identifiers, such as the users' full names and e-mail addresses. First parties might also unintentionally leak personal information to third parties via various coding mistakes [39], [99].

Figure 3.1 summarizes the various web tracking methods and the differences between third party services. X, Y correspond to advertisement services, A to a social widget service, and Z to an analytics service. All three services transmit the currently visited website back to the third party service. Provider X relies on stateful methods to transmit a unique identifier with each request, whereas provider Y transmits a device fingerprint (stateless tracking). Thus, if a user wiped all persistent tracking storage locations, provider Y would still be able to track the user.

3.1.2 Mobile Tracking

The tracking threats involved in browsing websites apply equally well to desktop browsers, as well as browsing done via mobile devices. Mobile devices, however, provide an additional source of potential information leakages: third-party services bundled with mobile applications [100]–[102]. In-app, third-party services, transmit unique device identifiers (`UDID`) to track the behavior of users. In older versions of Android and iOS, third parties were allowed to access these immutable UDIDs. Since 2013 both mobile operating systems, replaced immutable UDIDs with advertisement IDs, which can be reset by users. Mobile third-party trackers can still, however, collect additional unique device identifiers, such as the device's `IMEI`, or `MAC` address of the WiFi interface, in order to

reclaim reset-resistant tracking.

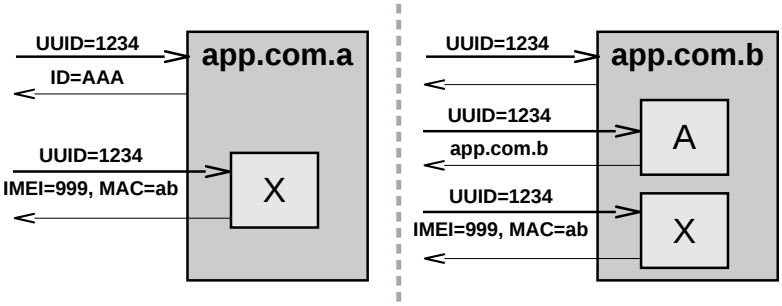


Figure 3.2: Mobile application tracking relies on unique user identifiers (UUIDs). Tracker (X) in addition uses the phone's IMEI and MAC address of the WiFi hardware. Tracker (A) is both in a first- and third-party position.

Figure 3.2 summarizes tracking via mobile applications. Service A is in both a first- and third-party position, whereas X corresponds to a typical advertisement service. Mobile in-app tracking enables third parties to create user profiles based on which applications certain people use, and can collect additional information, such as the users' location. Every time a mobile application is started, the name of the application together with unique device identifiers are transferred to third party services.

3.1.3 Threat Model

We exclude first-party tracking from our model, under the assumption that users visit those sites intentionally and the sites may legitimately need to track users to provide, for instance, the notion of a session. Our threat model accounts for the following threats posed by third-party trackers:

- **Stateful and stateless tracking by third parties**
  Both stateful and stateless tracking methods, ultimately rely on transferring either a unique user identifier, or a device fingerprint, to the tracking third party.

- **Passive collection of transmitted identifiers**
  The passive collection of transmitted identifiers is enabled by additional third-party trackers who rely on unencrypted communication protocols.

We investigate the interactions of the trackers described in our aforementioned threat models with tracker-blocking software, because it is currently the only protection method which prevents communication with third-party tracking services. Furthermore, we use a more stringent classification of tracking services than that of prior work, e.g., compared with the work of Roesner et al. [2]. Specifically, we do not treat analytics services

separately. We argue that these services can hypothetically match per-site identifiers to a unique user profile, based on various system properties (fingerprints) they collect. Our threat model does not account, however, for Internet Service Providers (as is the case in [103], [104]) or other entities that are able to actively manipulate en route web traffic.

3.2 Tracker Blocking

In this section we describe the various tracker-blocking methods that users have at their disposal, including the ones that can access and block all network traffic, as well as the ones specifically situated in users' browsers.

3.2.1 Network-based blocking

Network-based blocking methods were in use long before web browsers supported the notion of plugins and extensions. In the following, we discuss the most common network-based filtering methods and their drawbacks.

DNS blocking. DNS blocking uses address-based blacklists in order to block access to certain domains. In the context of blocking trackers (including ads) DNS blacklists are commonly distributed in the form of a `hosts` file. These hosts files are intended as replacements or extensions to the stock hosts files of operating systems. A number of projects maintain hosts files with popular advertising and tracking domains, which redirect requests to these domains to `localhost`. Examples of DNS blacklists focused on blocking advertisement and trackers include the longstanding *MVPS hosts*[46] and *Peter Lowe's list*[47]. DNS blacklists exist since the late 1990s and this network-based method is now experiencing a renaissance for blocking in-app advertisements on rooted mobile devices[48]. This tracker-blocking method works independently of the used application, but is also the most coarse-grained form of blocking. DNS filtering can be used to block entire (sub)domains but not individual `URIs`. That is, one cannot block *newyorktimes.com/tracker.js* while maintaining access to *newyorktimes.com*.

Interception Proxies. Interception proxies forward and modify web traffic. A popular privacy-enhancing interception proxy is *Privoxy*[49]. Privoxy has URI-based filtering capabilities, and can also modify the content and headers of web requests. As such, interception proxies can be used for more fine-grained blocking of third-party tracking by removing individual cookies, blocking certain URIs, and removing tracking code from

[46] `http://winhelp2002.mvps.org/hosts.htm`
[47] `http://pgl.yoyo.org/as/`
[48] `https://www.adaway.org`
[49] `http://www.privoxy.org/`

Table 3.1: Common browser extensions to block online trackers, installations (Aug. 2016), and underlying filter rules.

Browser Extension	Filter-rules	Firefox users	Chrome users
AdBlock Plus (ABP)	ABP	18,689,656	10,000,000+
AdBlock	ABP	NA	10,000,000+
Ghostery	custom (proprietary)	1,337,831	2.348.209
uBlock (Origin)	ABP	1,243,409	3,852,990
AdBlock Edge	ABP	408,410	NA
Disconnect	custom (GPL)	265,773	797,097
Blur	custom (proprietary)	176,027	329,446
Privacy Badger	algorithmic	80,291	324,062

web pages. Interception proxies cannot, however, intercept or modify encrypted TLS (HTTPS) traffic. In theory, interception proxies could use their own custom TLS Certification Authority to modify HTTPS traffic. In practice, an active `man-in-the-middle` attack that blocks trackers puts users at serious security risks [105]. Furthermore, even with a custom Certification Authority, interception proxies cannot handle websites that utilize certificate pinning [106], [107].

All network-based blocking methods have the advantage of working independently of the underlying application or browser. These methods have, however, two important shortcomings. First, as mentioned earlier, they cannot perform fine-grained blocking on encrypted web traffic (proxy), but only block entire domains (DNS-based). Second, third-party content cannot always be reliably detected at the network level. Specifically, at every third-party request, the network-level blocking tool must be able to reliably differentiate intentional third-party requests (user clicked on a third-party link and is expecting to navigate to a different website) from unintentional, tracking-related, third-party requests.

3.2.2 Browser Extensions

Browser extensions can reliably detect third party content, and can modify any content loaded by web browsers including encrypted web traffic. Table 3.1 summarizes the most popular tracker-blocking browser extensions available for users of Mozilla Firefox and Google Chrome. In the following, we describe the different browser extensions in more detail.

Ad Blockers. The apparent need for blocking ads has led to some of today's most popular browser extensions. This trend also becomes apparent when comparing the install counts of different tracker-blocking tools across extension markets. *AdBlock Plus*

(ABP) is the most popular of these extensions. At the time of writing ABP also has by far the most users of all Firefox extensions. ABP limits user tracking by blocking ads from being loaded. A number of other extensions build upon the filter rules by ABP (shown in Table 3.1). ABP filters are written using a custom pattern syntax and then internally translated to regular expressions. There are two basic types of ABP filter rules: general blocking filters and CSS filters. CSS filters are used to hide previously blocked ad elements on websites. In addition to filter rules, exceptions for these filters can be defined. The use of regular expressions in the filter rules is discouraged because of the potential performance impact. By default, ABP subscribes to *EasyList* filter rules, which include general adblocking rules. A number of additional subscriptions exists to improve regional blocking of ads, as well to block additional third-party trackers (*EasyPrivacy*).

Tracker Blockers. Tracker-blocking extensions focus on blocking trackers. The most popular extension in this category is *Ghostery*. It is important to note that Ghostery does not block trackers by default. It merely provides feedback on which third-party trackers are included in each visited website. Similar extensions are *Disconnect*, Abine's *Blur*, and EFF's *Privacy Badger*. Tracker blocking rules include: third-party domains, specific URIs, and "surrogates." Surrogates offer click-to-play functionality for social widgets similar to the ones proposed by Roesner et al. [2].

3.2.3 Types of Rulesets

The effectiveness of all tracker-blocking methods discussed so far depends on their underlying blocking ruleset. Rulesets can be divided into three broad categories: *community-driven*, *centralized*, and *algorithmic*. The most popular community-driven rulesets for blocking ads and trackers origin from the development of the AdBlock Plus browser extension. At the time of this writing, the main AdBlock Plus ruleset (`EasyList`) consists of over 17,000 URI patterns and more than 25,000 CSS tags to be blocked. `EasyPrivacy` is a ruleset for AdBlock Plus with more than 9,000 community-maintained rules targeted at blocking trackers. The subscriptions offered by the AdBlock Plus community are used in a number of other browser extensions, including: AdBlock, AdBlock Edge, and uBlock. Every change to the AdBlock Plus rulesets is tracked via a public Mercurial source control management repository[50]. Eyeo, the company behind the AdBlock Plus browser extension, at the end of 2011, started an "acceptable ads program"[51]. By 2015, Eyeo's acceptable ads program allowed for the whitelisting of over 300 businesses [108]. The acceptable ads program is enabled by default for the AdBlock Plus browser extension.

Ghostery, Disconnect, and Blur rely on a centralized approach to create blocking rules.

[50] `https://hg.adblockplus.org/`

[51] `https://adblockplus.org/development-builds/allowing-acceptable-ads-in-adblock-plus`

This means that the companies behind these three tracker-blocking tools maintain and curate blocking rules. These centralized, top-down, filter rules are, in general, more compact than community-driven approaches. For example, Disconnect consists of a list of 2,200 third-party domains, compared with over 9,000 rules in the community-driven EasyPrivacy ruleset.

The third category are algorithmic approaches for blocking rules. These blocking tools do not rely on regularly updated blacklists, but instead use heuristics to automatically detect third-party trackers. The most popular example for the use of algorithmic rulesets is EFF's Privacy Badger which labels third parties as trackers by observing the requests between first-party and third-party websites and searching for the same high-entropy strings exchanged between multiple first-party websites and individual third-party ones.

3.3 Study Design and Methodology

In this section, we describe the methodology of our large-scale tracker analysis. The section is divided into two main parts: our web tracking evaluation and our mobile tracking evaluation.

3.3.1 Web Tracking

We evaluate the effectiveness of the most popular rule-based advertisement and tracker blocking browser extensions. Specifically, we use the following browser extensions:

- AdBlock Plus 2.7.3 (Default settings)
- Disconnect 3.15.3 (Default settings)
- Ghostery 6.2.0 **(Blocking activated)**
- EFF Privacy Badger 0.2.6 (trained with Alexa Top 1,000)
- uBlock Origin 1.7.0 (Default settings)

Overall, whenever possible, we use browser extensions with their default settings, to simulate the experience of users who install an extension but do not further configure them. We include AdBlock Plus because it is by far the most popular browser extension to block advertisement, which is one form of tracking due to the trend of Online Behavior Advertising (OBA). AdBlock Plus by default relies on the EasyList ruleset but whitelists some "acceptable" advertisement networks. Ghostery is the most popular browser extension focused on online tracking but, by default, Ghostery only displays detected trackers but does not block them. For our measurements we thus activated blocking for all of

Ghostery's third-party categories. Disconnect is an alternative for Ghostery and Disconnect's ruleset is also used for Firefox's tracking protection [95]. uBlock Origin markets itself as a lightweight alternative for AdBlock Plus and as a "wide-spectrum blocker". We use uBlock with its default settings, which include EasyList and EasyPrivacy rulesets, as well as other community-driven rulesets to block: ads, trackers, and malware. Finally, we trained EFF's Privacy Badger with the Alexa Top 1,000 websites to evaluate the effectiveness of this novel algorithmic blocking approach.

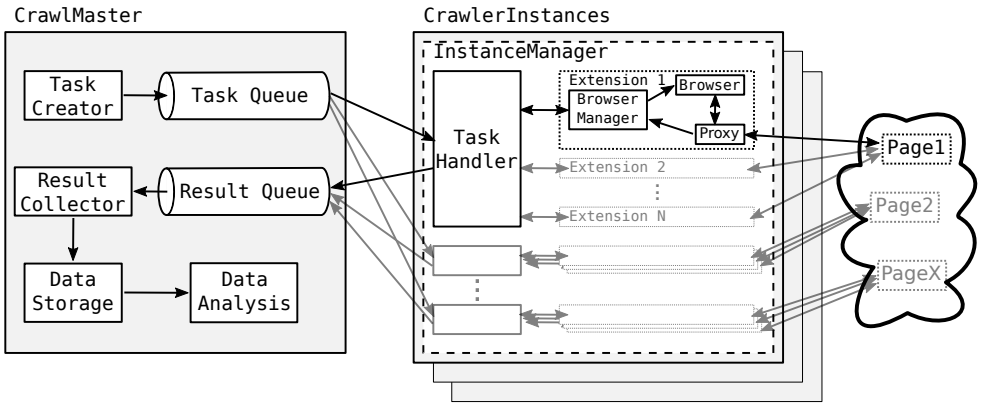


Figure 3.3: System overview of our modular web measurement framework CRAWLIUM. The CrawlMaster is responsible for task distribution and data handling. The CrawlerInstances each manage several task handlers which process the tasks by running them against the different browser extensions in parallel. The results are sent back to the CrawlMaster for collection and analysis.

Analysis Framework. In order to analyze the browser extensions, we developed a distributed, modular web crawler framework called CRAWLIUM. We developed the CRAWLIUM measurement framework because none of the existing measurement frameworks, such as OpenWPM [109], were able to run multiple browser configurations in parallel and support high scalability at the same time.

A high level system overview of our framework is outlined in Figure 3.3. Our analysis framework consists of two instance types: `CrawlMaster`, and `CrawlerInstances`.

The `CrawlMaster` instance is responsible for producing task and aggregating results, while the `CrawlerInstances` are responsible for executing the actual measurement tasks. The `CrawlMaster` needs to run all the time for uninterrupted operation. However, `CrawlerInstances` can be terminated at any point in time. Each crawler instance hosts several `TaskHandlers`. These are responsible for running the tasks on the browsers and sending the results back to the manager instance. Each `TaskHandler` is responsible for a set of `BrowserManagers`, one for each browser extension and another one without modifications, as a baseline for the analysis ("plain" profile). BrowserManagers handle command execution on browsers and collect results from the proxy. They

also take care of restarting browsers and proxies and retry execution of tasks in the case of an error.

By running the browser with the different profiles in parallel, we account for potential temporal effects during crawls. These temporal effects are, for example, changes to website between requests from different profiles. Since our setup is very modular, it can be easily adapted to other user profiles. The actual framework is based on Firefox, Selenium and `mitmproxy` [110] to load web pages and collect request and responses.

To thoroughly evaluate the extensions concerning their functionality, we also collect statistics on resource consumption. Therefore, our framework collects information on CPU usage and memory consumption for each browser based on a task granularity. Every time before a new task is sent to the browser, the BrowserManager collects the current memory consumption and resets the count for CPU usage. After the task is handled by the browser, we store the percentage of used computation resources and collect a second memory snapshot. Finally, for each finished task, we reset the browser by closing all windows and pages that were opened during crawling.

Web Sample. The sample of our evaluation is seeded from the global *Alexa Top Sites*. The detection requires the analysis of the actual (sub)pages from the Alexa Top Sites which contain trackers. On certain websites, such as news sites, tracking code (e.g., social widgets) is embedded in news pages, and not on the landing page of domains. We therefore use a twofold crawling strategy in order to account for trackers on nested web pages. First, we use `PhantomJS` to determine the landing page of the top 200,000 domains in the global Alexa Top Website dataset. This phase of the measurement does not collect final results but serves as a first stage to determine the sample set of pages for the second stage of the measurement. We chose PhantomJS to account for `AJAX` requests, and thus have access to the content of current web sites while still maintaining a low resource profile during sample set selection. Second, we enumerate all nested links on a given domain by analyzing the gathered web site. After collecting the nested links, we select two random subpages for each website. If no nested links are found, or the domain did not respond, the given domain is excluded. Finally, each valid website sample consists of three webpages: the landing page and the two random subpages.

Domain Aggregation. To extract the main domains from URIs, we rely on the *python tld package*, which in turn uses Mozilla's public suffix list[52]. We removed user-provided TLD information from Mozilla's list, to accurately group our results based on providers instead of individual services.

Detection of Fingerprinters. In addition to traditional stateful third-party tracking,

[52] `https://publicsuffix.org/`

our large-scale evaluation accounts for tracking based on fingerprinting. Our analysis is based on the findings provided by Acar et al. [7] in FPDetective and Englehardt et al. [8] based on OpenWPM. Acar et al. provide several regular expressions[53] to detect fingerprinters based on their URIs while Englehart et al. provide specific URIs to identify fingerprinters. We used these regular expressions and URIs to detect if a page includes a fingerprinting script based on the collected results. This analysis provides a baseline on the effectiveness of existing browser extensions to prohibit the execution of well-known and recently identified fingerprinting scripts.

3.3.2 Mobile Tracking

To better understand the difference between desktop and mobile systems we analyzed a sample set of 10,000 popular Android applications concerning the inclusion of third party trackers. We obtained the sample from Viennot et al. [111]. We expect that the browser extensions ported to mobile platforms yield comparable results to our web tracking experiments. This analysis complements our web tracking experiments in the sense that we include in-app third party services which cannot be easily blocked without rooting the underlying operating system. We conducted pilot experiments by statically analyzing our collected sample with `androguard`[54] to extract activities found in the applications' manifest file. These pilot experiments, however, showed a clear drawback of static analysis: although the extracted activities point to specific ad provider, we were unable to determine the final ad-network tracking mobile users. The main reason behind the limit of static analysis regarding mobile in-app tracking, is due to the fact that nowadays most mobile ads are delivered through mediation networks, such as, Google AdMob[55]. This implies that even if an application contains an activity pointing to the AdMob advertisements, it does not reveal whether the advertisement which is going to be served will originate from AdMob itself, or from some different ad provider which is part of the mediated networks.

To overcome the limitation of static analysis, we set up a dynamic analysis environment. Our framework executes in the Genymotion emulator[56] and exercises apps with the help of Monkeyrunner. We furthermore use an MITM proxy to intercept all outgoing and incoming traffic for each of applications in our sample. Finally, we evaluate two common blocking approaches for mobile in-app advertisement: DNS-based blocking and *Adblock Plus for Android*. To analyze DNS-based blocking we used the rulesets of two different

[53] `https://github.com/fpdetective/fpdetective/blob/master/src/crawler/fp_regex.py`
[54] `https://github.com/androguard/androguard`
[55] `https://developers.google.com/admob/android/mediation-networks`
[56] `https://www.genymotion.com`

applications, *AdAway*[57] and *MoaAB (Mother of all AD-BLOCKING)*[58]. AdAway and MoaAB are Android applications for rooted Android phones which rely on a network-based approach to block mobile in-app advertisement by replacing Android's stock host file. The Adblock Plus for Android application does not require root access but creates a proxy which filters web traffic based on the same filter rules as the Adblock Plus browser extension. The Adblock Plus Proxy only intercepts HTTP traffic and thus does not block advertisement on HTTPS requests[59]. In order to evaluate the effectiveness of these three popular mobile blocking tools, we matched their underlying rulesets against the collected requests from our dynamic Android application analysis. For AdAway and MoaAB, we used their hosts files. For Adblock Plus we matched requests against their default ruleset.

3.4 Results

In this section we present the results of our evaluation of more than 100,000 popular websites and 10,000 Android applications.

3.4.1 Collected Data

We seed our crawling framework with the top 200,000 Alexa web sites. The actual crawling was performed on Amazon EC2 via their Oregon datacenter, therefore all requests originated from the same region, in order to limit potential location bias. We carried out multiple measurements of the Top 200,000 Alexa web sites, to ensure our framework produces comparable and valid samples. The data discussed in this section is based on data obtained in May 2016. Out of 200,000 websites from the Alexa dataset, we consider 61.93% of them as having been properly crawled. This lower number can be attributed to two reasons. First, during the PhantomJS analysis a number of websites did not respond, which reduced the set of tasks available for the second stage of the crawl to 191,492 websites (4.25% failed). Second, we only consider web site samples where none of the browser extensions caused timeouts when loading. The numbers for successful results for the different extensions is shown in Table 3.2. Privacy Badger caused the highest number of failed samples, whereas only 71% of all websites loaded without timeouts with this extension installed. Except from Privacy Badger, all extensions improved our sample collection success rate as compared to the plain profile, whereas less than 10% of all websites produced timeouts. Our filtering process finally resulted in a total set of 123,876 websites which were successfully analyzed with all browser extensions. These websites are uniformly spread in the Alexa top 200K ranks. Therefore, we argue that

[57] https://www.adaway.org
[58] http://forum.xda-developers.com/showthread.php?t=1916098
[59] https://adblockplus.org/releases/adblock-plus-10-for-android-released

our results are generalizable and characteristic of the entire range. In terms of requests, our crawlers were able to collect over 137 million HTTP(S) requests.

Table 3.2: Successfully crawled web pages per extension. We consider a request per extension as failed if they either did not return any results at all after three tries or had at least one embedded request time out after 90 seconds.

Plugin	# Sites	# Success	Failed %
plain	191,492	164,815	13.93%
adblockplus	191,492	170,636	10.89%
disconnect	191,492	176,659	7.75%
ghostery	191,492	179,068	6.49%
privacybadger	191,492	136,796	28.56%
ublock-origin	191,492	178,233	6.92%

The analysis of Android applications was performed at our local lab. We used our dynamic analysis framework to collect the network requests of the 10,000 most popular Android applications. We excluded 939 applications from our set that caused runtime errors while being analyzed.

3.4.2 Identified Third-Party Services

We extracted the set of domains to which the different browser instances issued requests. The actual top level domains were identified based on Mozilla's public suffix list (see Section 3.3.1). The information was then aggregated to determine how often a specific domain (TLD+1) occurs in different popularity ranks of first-party websites. Figure 3.4 outlines the distribution of the most popular third-party domains in our crawled set of popular websites.

We found that the great majority of third-party services belong to relative small number of large Internet players. Table 3.3 shows an aggregated view on the third-party services we observed, based on the on the meta-information provided by Falahrastegar et al. [112]. Google provides by far the most popular third-party web services. Overall *Google services are included by 97% of websites in our sample.* The reach of Google can be attributed mainly to three service categories: analytics, advertisements, and CDN services (e.g., googleapis.com). Social widgets by Facebook and Twitter are included by 47%, and 24% respectively, by all websites in our sample. Amazon is the third biggest third-party service due to their CDN and cloud computing services.

Mobile Third Parties. Table 3.3 also shows the reach of Internet companies in the context of Android applications. Google's reach of 74% of their third-party services for Android applications follows intuitively as Google develops Android. In comparison

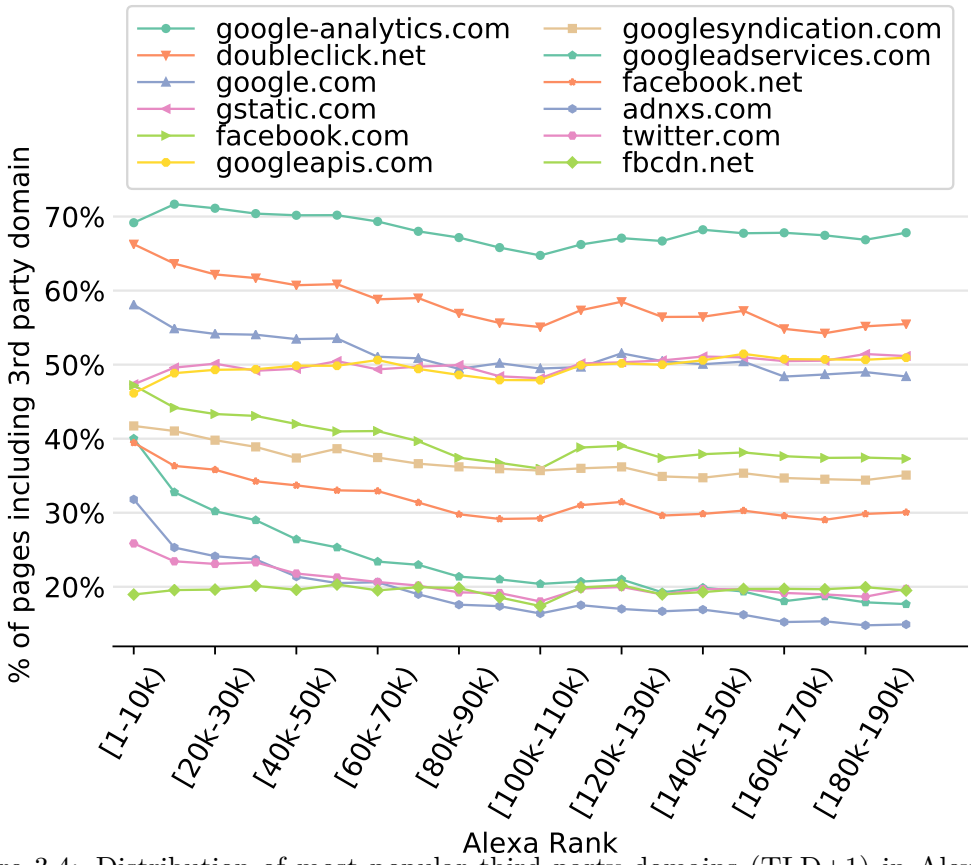


Figure 3.4: Distribution of most popular third party domains (TLD+1) in Alexa Top 200,000 websites in 10,000 intervals.

with our web sample, however, it appears that Facebook, Twitter, and Amazon have considerable less reach on Android applications in comparison to websites.

Insecure Content Delivery. Our results indicate that the majority of observed third-party services use unencrypted (insecure) protocols to deliver content and to exchange tracking information. Figure 3.5 outlines the inclusion of third-party content through `HTTP` and `HTTPS` for our complete set of analyzed websites. Our results show that *more than 60% of websites* still uses HTTP for third-party, content delivery. There are also several web pages that access the same third-party domain through HTTP as well as HTTPS. This behavior is likely due to initial requests performed through HTTP and upgraded to HTTPS by a third-party content provider. We found that, for our Android sample, more than 75% of all requests were performed over HTTP. Furthermore, the mobile blocking tools we analyzed had an overall small impact on requests to third party services, with the best DNS-based blocking list (MoaAB) reducing requests to third parties by 25%. As such, our results effectively show that browser extensions reduce insecurely loaded third-party content on websites. Ghostery, for example, limits

Table 3.3: Percentage of websites and Android applications reached by the Top 15 companies that provide third-party services. The results show the total reach (plain) as well as the reach after the application of each blocking solution. For the web dataset these are AdBlock Plus [abp], Disconnect [dc], Ghostery [gh], PrivacyBadger [pb], ublock Origin [ubo], and all applications combined [c]. For Android these are EasyList [e], AdAway [a] and MoaAB [m]

	Desktop							*Mobile*			
	plain	abp	dc	gh	pb	ubo	c	plain	e	a	m
Google	97	93	80	66	93	69	60	74	74	57	54
Facebook	47	44	5	2	4	39	0	6	6	6	6
Amazon	25	21	21	13	20	13	10	8	8	8	7
Twitter	24	21	6	1	19	19	1	1	1	1	1
Yahoo	18	6	4	2	3	2	1	14	14	14	0
AddThis	15	14	8	0	0	0	0	0	0	0	0
ComScor	14	10	1	0	1	0	0	2	2	0	0
AOL	11	0	1	0	1	0	0	0	0	0	0
Adobe	10	5	0	0	0	0	0	0	0	0	0
Quantcast	9	5	1	0	0	0	0	0	0	0	0
Conversant(ValueClick)	8	1	0	0	1	0	0	0	0	0	0
RadiumOne	6	1	0	0	0	0	0	0	0	0	0
Baidu	6	6	6	2	0	1	0	2	2	2	0
AudienceScience	5	0	0	0	0	0	0	0	0	0	0
Sizmek	5	0	0	0	1	0	0	0	0	0	0

insecurely loaded third-party content to about 20%. This, however, means that attackers could, in the worst case, still target users on every fifth website through passive and active attacks on third-party services.

3.4.3 Blocking Behavior and Shortcomings

We categorized the 30 most popular third-party tracker domains identified in our experiment to compare the effectiveness of our analyzed browser extensions with respect to blocking specific third-party service categories. Figure 3.6 shows the effectiveness of browser extensions in blocking the most common categories of third-party services. This figure outlines the blocking behavior of the popular browser extensions we analyzed. Table 3.9 in the Appendix provides a detailed view on the 30 most-popular third parties identified in our measurements, and the impact of tracker-blocking browser extensions. For example, the first row in Table 3.9 shows that Google Analytics was the most popular third party service in our sample, with 53.6% of requests to *www.google-analytics.com* performed over HTTP, and 13.2% performed over HTTPS (plain column). The columns following "plain" show the impact of our evaluated browser extensions,

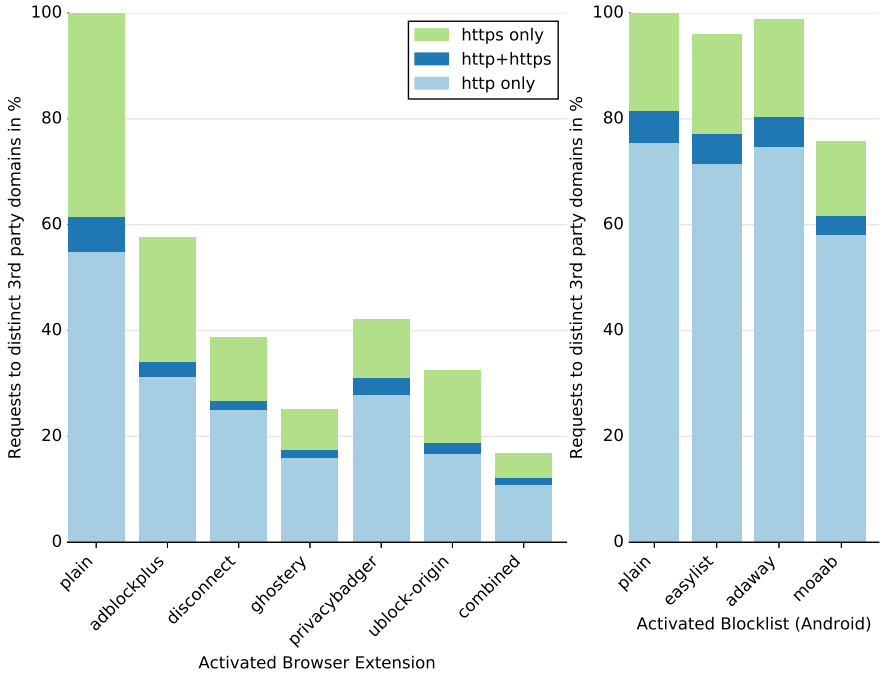


Figure 3.5: Protocols used for requests to distinct third party domains.

where, for example, uBlock effectively blocks all HTTP and HTTPS requests to Google Analytics.

AdBlock Plus is by far the most popular ad blocker. Our findings show that AdBlock Plus blocked **the least amount** of advertising-related, third-party requests of all tracker blocking extensions in our measurements. The blocking behavior of AdBlock Plus can be attributed to its acceptable ads program (discussed in Section 3.2), which resulted in an overall decrease by 4% of blocked advertisements in our measurements. Table 3.9 details our finding: the majority of browser extensions, e.g., completely block *googleads.g.doubleclick.net*, while AdblockPlus still allows it to be included in about 1.5% of pages through HTTP and 13.7% of pages through HTTPS.

Furthermore, our measurements highlight an important issue: **a number of browser extensions fail to effectively block social widgets** (e.g., Facebook's Like button or Twitter's share/retweet button) from tracking users. Disconnect fails to block requests originating from Twitter's social widgets in our measurements. uBlock Origin, with the community-driven EasyPrivacy rules, fails to significantly impact tracking by major social networks, such as Facebook and Twitter. PrivacyBadger is the only extension to completely block third-party requests to Facebook (*https://www.facebook.com*, see Table 3.9) but does not block all requests to Twitter.

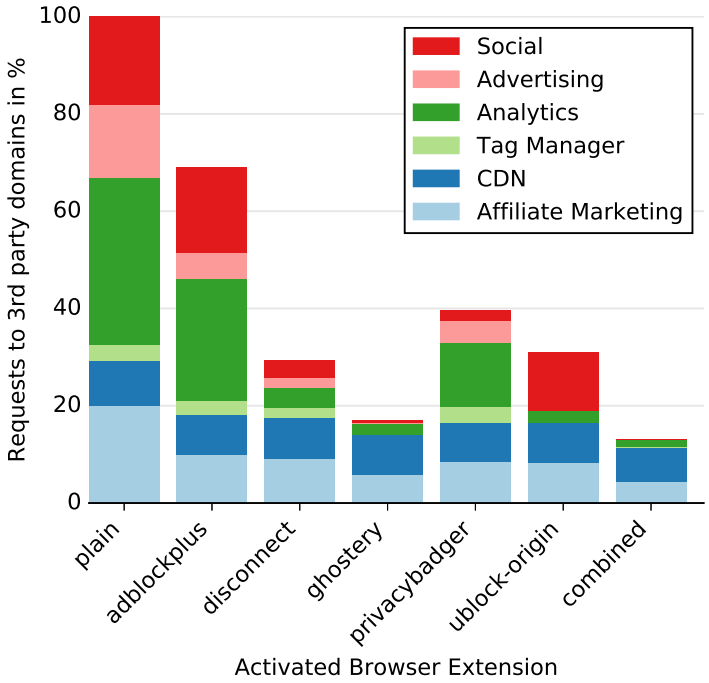


Figure 3.6: Categories blocked by different extensions, and all extensions combined. The data shows the categorized and aggregated numbers of the 30 most popular third-party services in our sample of 123,876 websites.

Overall, our results suggest that top-down approaches for rulesets (Disconnect, Ghostery) outperform community-based rulesets (AdblockPlus, uBlock origin) in terms of their overall effectiveness in blocking the most popular third-party trackers. *PrivacyBadger* takes a different approach at blocking third party trackers and does not come with a preloaded blocking list, with its blocking capabilities depending largely on the pages that have been previously visited. Furthermore, certain major third parties, like *Google Analytics*, are not considered trackers, because they do not share state between pages, and are therefore not blocked by *PrivacyBadger* at all[60]. Overall, PrivacyBadger showed promising effectiveness as compared with traditional rule-based blocking extensions but, as previously discussed, also led to a large number of timeouts and therefore to a potentially large number of malfunctioning webpages.

Mobile Blocking. Table 3.4 outlines third party services detected in our Android application sample and the impact of our evaluated blocking tools in detail. The three blocking tools we evaluated offered limited protection against third-party tracking. AdAway [a] blocked the four most popular domains for mobile ad delivery (DoubleClick, Ads by Flurry, googlesyndication, and admob) but did not include rules for *Chartboost*

[60] https://github.com/EFForg/privacybadgerfirefox/issues/298

and other common mobile analytics providers. Adblock Plus for Android with their default EasyList [e] ruleset detected common advertisement providers but this proxy-based solution can not block HTTPS requests. Table 3.4 highlights this limitation of Adblock Plus for Android. For example, the first row of Table 3.4 shows that 41.89% of apps that make requests to `googleads.g.doubleclick.net` are not blocked because they used HTTPS. The default rulesets of Adblock Plus also lack mobile-specific rules for other popular third parties, such as AdMob. The DNS-based "Mother of All AD-BLOCKING" [m] blocklist had the biggest impact on third-party tracking in Android applications. There are specific third party services that **none of these tools can easily block**. Facebook, for example, uses HTTPS for all requests, meaning that proxy-based blocking does not work, and DNS-blocking of, *graph.facebook.com*, would break the functionality of applications.

3.4.4 Blind spots of different rule sets

Apart from the ability of browser extensions to block the third-party domains with the largest footprint in terms of their web presence, we are also interested in each extension's ability to block smaller, less popular, third-party trackers. Furthermore, we analyze if there exists a trend of blocked third parties from the more popular to the less popular websites. To this end, we first extracted the number of distinct third-party domains included in the plain requests (no blocking extension installed) for each 10,000 Alexa rank interval. This dataset was split into 3 distinct sets containing third parties that were included on (2-20)/(20-200)/200-10000) distinct first-party pages. Therefore, the first set includes third parties with the smallest footprint (web presence) whereas the last set third parties with the largest one. For each browser extension we then analyzed the number of the third-party domains that were not blocked. We consider a site as being, at least partially, blocked, if the inclusion count drops to half of the lower bound (e.g. less than 1/10/100 inclusions left respectively). Therefore, if a browser extension blocks more third party domains in each respective set, it will consequently have a lower rate of third party domains still included.

The results of this analysis are outlined in Figure 3.7. One conclusion we can draw from the results is that third-party domains with a larger footprint seem to be blocked more effectively by all extensions. Furthermore, we see that *Ghostery* has the best performance in blocking third parties with more than 20 inclusions. However, as seen in first plot, *uBlock* has a better performance on third parties included in less than 20 pages per Alexa rank interval. Our results indicate that smaller tracking companies are able to avoid attention from blocking tools and, thus, persist regardless of the presence of tracker-blocking extensions. An interesting side-effect is that in the very competitive and crowded sector of third-party tracking, tracker-blocking tools with incomplete coverage are indirectly "helping" smaller players by blocking their larger competitors.

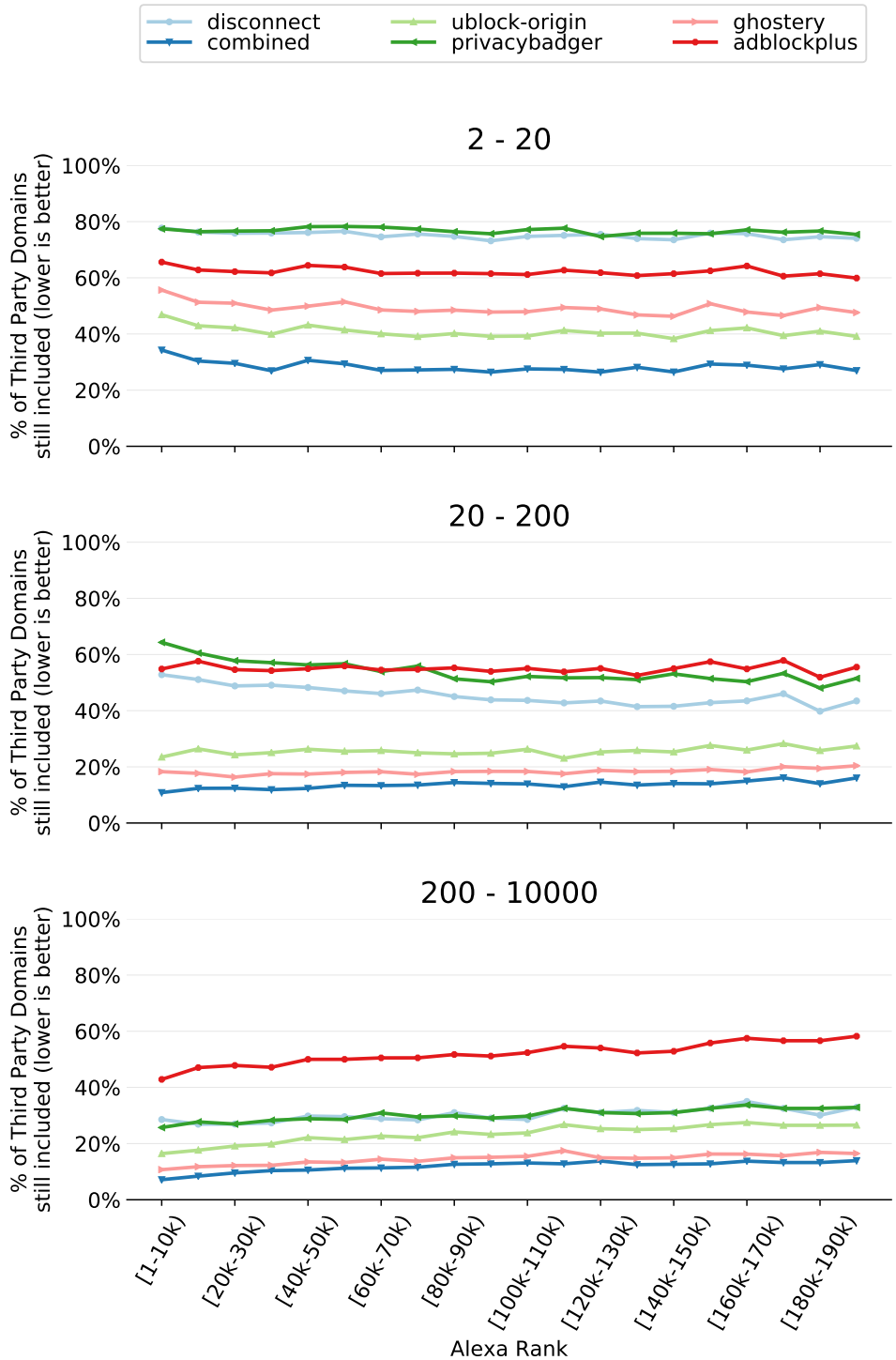


Figure 3.7: Sum of included third-party domains, with 2-20/20-200/200-10000 inclusions, which are not blocked by a specific browser extension in relation to the plain profile. In all graphs, the lower an extension is at the y-axis, the better (less third-parties remaining).

Table 3.4: Per app distribution of the top 15 third-party services in the Android sample in % of total apps and the impact of DNS-/proxy- based blocking.

		plain *http/https*	easylist *http/https*	adaway *http/https*	moaab *http/https*
doubleclick.net	googleads.g	17.22/41.89	/41.89	/	/
	stats.g	0.08/0.71	0.08/0.71	/	/
	pubads.g	0.26/0.43	/0.43	/	/
google.com	android.clients	0.08/28.48	0.08/28.48	0.08/28.48	0.08/28.48
	www	2.37/4.97	2.24/4.97	2.37/4.97	2.37/4.97
	accounts	/0.61	/0.61	/0.61	/0.61
googlesyndication.com	pagead2	12.56/15.52	/15.52	/	/
	tpc	1.38/11.60	0.03/11.60	/	/
	video-ad-stats	0.01/	0.01/	/	/
googleapis.com	fonts	7.12/11.09	7.12/11.09	7.12/11.09	7.12/11.09
	www	/4.44	/4.44	/4.44	/4.44
	play	/4.35	/4.35	/4.35	/4.35
gstatic.com	fonts	6.72/10.74	6.72/10.74	6.72/10.74	6.72/10.74
	csi	2.22/3.62	2.22/3.62	2.22/3.62	/
	www	3.07/0.98	2.91/0.98	3.07/0.98	3.07/0.98
admob.com	media	16.52/0.22	16.52/0.22	/	/
	e	0.02/	/	0.02/	/
google-analytics.com	ssl	/11.44	/11.44	/	/
	www	3.92/0.63	3.92/0.63	/	/
googleusercontent.com	lh3	6.38/8.38	6.38/8.38	6.38/8.38	6.38/8.38
	lh5	0.03/0.28	0.03/0.28	0.03/0.28	0.03/0.28
	lh4	0.06/0.15	0.06/0.15	0.06/0.15	0.06/0.15
flurry.com	data	9.30/3.94	9.30/3.94	9.30/3.94	/
	ads	0.26/0.73	/0.73	/	/
	cdn	/0.66	/0.66	/0.66	/
adobe.com	mobiledl	8.77/	8.77/	8.77/	8.77/
	airdownload2	6.91/	6.91/	6.91/	6.91/
	sp.auth	/0.09	/0.09	/0.09	/0.09
chartboost.com	live	/4.34	/4.34	/4.34	/
	a	/3.16	/3.16	/3.16	/
	www	/3.09	/3.09	/3.09	/
unity3d.com	stats	7.01/	7.01/	7.01/	/
	config.uca.cloud	/0.01	/0.01	/0.01	/0.01
	api.uca.cloud	0.01/	0.01/	0.01/	0.01/
facebook.com	graph	0.11/3.97	0.11/3.97	0.11/3.97	0.11/3.97
	m	0.04/2.42	0.04/2.42	0.04/2.42	0.04/2.42
	www	0.36/1.20	0.36/1.20	0.36/1.20	0.36/1.20
amazonaws.com	s3	0.19/3.20	0.19/3.20	0.19/3.20	0.19/3.20
	prod-static-images.s3	0.21/	0.21/	0.21/	0.21/
	s3-us-west-1	0.08/0.15	0.08/0.15	0.08/0.15	0.08/0.15
tapjoyads.com	ws	0.01/4.46	0.01/4.46	0.01/4.46	/

Fingerprinters. In addition to measuring the effectiveness of browser extensions in blocking well known third parties, we also investigated their ability to block stateless fingerprinting services. These services are able to identify users based on different attributes that are exposed through their browser, like available fonts or installed extensions. To quantify each extension's ability to block fingerprinting, we leveraged the previously detected fingerprinters found by Acar et al. [7] and the newly identified fingerprinters by Englehardt et al. [8]. Specifically, we utilized the regular expressions provided by the authors of FPDetective on Github[61] and the URI's provided in [8]. The results we collected by applying the rules to our dataset are shown in Table 3.5, Table 3.6, Table 3.7 and Table 3.8. Between the two studies, it seems that recently more third parties rely on fingerprinting to identify users. A number of the fingerprinting services we detected were not blocked by any of our evaluated web browser extensions, such as *MERCADOLIBRE*, *SiteBlackBox*, and *CDN.net*. Even though some of these services were identified by both studies as the providers of fingerprinting scripts, it is unfortunate to see that they are not completely blocked by all of browser extensions. For example, *CDN.net* was identified by FPDetective (3 years ago) and again by Englehardt et al., and yet none of the extensions include it in their rule sets. Furthermore we noticed, that the numbers between OpenWPM and our crawl are distributed differently. As an example we inspect the first three scripts in canvas fingerprinting. While Englehart et al. identified *doubleverify.com/dvtp_src_internal23.js* and *doubleverify.com/dvtp_src_internal24.js* as the fingerprinters with most inclusions both of them were not included in our dataset at all. However, as we checked for the regular expression *doubleverify.com/dvtp_src_internal.*.js* we were able to identify a similar number of inclusions, albeit with slightly different names. We argue that fingerprinters change the names of their scripts to evade overly strict rule sets. We also noticed that for some instances we observed more invocations of fingerprinting scripts with activated browser extensions as compared to our vanilla (plain) browser instance. This divergence is likely a result of additional measures by websites to combat clickfraud in the absence of other third-party identifiers [5]. Finally, our analysis of popular Android applications showed that *ThreatMetrix* was included in 149 applications, i.e., 1.64%, of our sample. We found that only the extensive DNS-based block list "Mother of all AD-BLOCKING"[62], effectively blocked this fingerprinting service.

3.4.5 Overhead of Tracker Blocking

Blocking trackers with browser extensions comes at the cost of additional memory and CPU overhead for matching requests against their blocklists. This section discusses the overhead of the browser extensions we measured. These findings are especially

[61] https://github.com/fpdetective/fpdetective/blob/master/src/crawler/fp_regex.py

[62] http://forum.xda-developers.com/showthread.php?t=1916098

Table 3.5: Number of pages with detected fingerprinting services, listed without any extensions (plain) and per blocking extension for **FP-Detective**. (See Table 3.3 for a description of the table header).

	Web Dataset							Android			
	plain	abp	dc	gh	pb	ubo	c	plain	e	a	m
BlueCava	27	5	0	0	25	0	0	0	0	0	0
Myfreecams	0	0	0	0	0	0	0	0	0	0	0
Mindshare Tech.	1	1	1	1	1	0	0	0	0	0	0
AFK Media	2	2	2	2	1	2	1	0	0	0	0
CDN.net	15	14	17	17	14	16	11	0	0	0	0
ANALYTICSPROS	0	0	0	0	0	0	0	0	0	0	0
Anonymizer	0	0	0	0	0	0	0	0	0	0	0
AAMI	0	0	0	0	0	0	0	0	0	0	0
VIRWOX	1	1	1	1	1	1	1	0	0	0	0
ISINGLES	0	0	0	0	0	0	0	0	0	0	0
BBelements	0	0	0	0	0	0	0	0	0	0	0
Inside graph	16	17	16	0	15	0	0	0	0	0	0
PIANOMEDIA	0	0	0	0	0	0	0	0	0	0	0
ALIBABA	0	0	0	0	0	0	0	0	0	0	0
MERCADOLIBRE	4	5	5	5	4	5	3	0	0	0	0
LIGATUS	0	0	0	0	0	0	0	0	0	0	0
ThreatMetrix	39	39	39	1	37	0	0	149	149	149	0
IOVATION	98	97	97	1	97	4	1	0	0	0	0
MaxMind	14	13	13	14	12	1	1	1	1	1	0
Analytics-engine	0	0	0	0	0	0	0	0	0	0	0
Coinbase	0	0	0	0	0	0	0	0	0	0	0
SiteBlackBox	11	11	12	12	11	11	11	0	0	0	0
Perferencement	0	0	0	0	0	0	0	0	0	0	0

Table 3.6: Number of pages with detected fingerprinting services, listed without any extensions (plain) and per blocking extension for **OpenWPM: Canvas Font Fingerprinting**. (See Table 3.3 for a description of the table header).

	Web Dataset							Android			
	plain	abp	dc	gh	pb	ubo	c	plain	e	a	m
mathid.mathtag.com/device/id\.js	121	16	1	1	0	1	0	0	0	0	0
mathid.mathtag.com/d/i\.js	437	374	2	1	2	3	0	0	0	0	0
admicro1.vcmedia.vn/core/fipmin\.js	39	1	0	3	41	1	0	0	0	0	0
.*.online-metrix.net	39	39	39	1	37	0	0	149	149	149	0
pixel.infernotions.com/pixel/	6	6	6	1	7	1	1	0	0	0	0
api.twisto.cz/v2/proxy/test	0	0	0	0	0	0	0	0	0	0	0
go.lynxbroker.de/eat_session\.js	0	0	0	0	0	0	0	0	0	0	0

Table 3.7: Number of pages with detected fingerprinting services, listed without any extensions (plain) and per blocking extension for **OpenWPM: Canvas Fingerprinting**. (See Table 3.3 for a description of the table header).

	Web Dataset							Android			
	plain	abp	dc	gh	pb	ubo	c	plain	e	a	m
doubleverify.com/dvtp_src_internal.*\.js	4118	78	8	6	37	9	0	0	0	0	0
doubleverify.com/dvtp_src_internal24\.js	0	0	0	0	0	0	0	0	0	0	0
doubleverify.com/dvtp_src_internal23\.js	0	0	0	0	0	0	0	0	0	0	0
ap.lijit.com/sync	799	41	1	3	364	1	0	1	1	1	0
cdn.doubleverify.com/dvbs_src\.js	1781	26	4	2	51	4	0	0	0	0	0
rtbcdn.doubleverify.com/bsredirect5\.js	290	8	2	0	2	0	0	0	0	0	0
g.alicdn.com/alilog/mlog/aplus_v2\.js	116	121	129	131	48	0	0	0	0	0	0
static.audienceinsights.net/t\.js	39	17	27	25	30	0	0	0	0	0	0
static.boo-box.com/javascripts/embed\.js	21	0	21	0	21	0	0	0	0	0	0
admicro1.vcmedia.vn/core/fipmin\.js	39	1	0	3	41	1	0	0	0	0	0
c.imedia.cz/js/script\.js	45	35	45	0	43	38	0	0	0	0	0
ap.lijit.com/www/delivery/fp	826	27	296	3	349	1	0	1	0	1	0
www.lijit.com/delivery/fp	20	1	8	0	8	0	0	0	0	0	0
*amazonaws.com/af-bdaz/bquery\.js	40	0	35	0	32	0	0	0	0	0	0
.cloudfront.net/./platform.min\.js	23	24	23	3	25	15	2	0	0	0	0
voken.eyereturn.com/	41	0	1	0	13	0	0	0	0	0	0
*.hwcdn.net/fp/Scripts/PixelBundle\.js	2	1	1	1	1	0	0	0	0	0	0
static.fraudmetrix.cn/fm\.js	11	11	11	11	14	11	9	0	0	0	0
e.e701.net/cpc/js/common\.js	11	11	12	9	10	10	5	0	0	0	0
tags.bkrtx.com/js/bk-coretag\.js	631	391	134	0	449	6	0	10	10	10	0
dtt617kogtcso.cloudfront.net/sauce.min\.js	1	1	1	1	1	1	1	0	0	0	0

important to mobile devices where mobile browsers are slowly opening up to extensions, but computing resources and the impact on battery life are still a limited commodity.

Overall, our evaluated browser extensions did not cause a significant CPU overhead compared to the plain profile. Figure 3.8 shows the absolute CPU overhead of the plain profile (no browser extensions) and all tested extensions in our sample. Our measurements contain a number of outliers but also show a clear trend for all analyzed websites. While it seems that the use of privacybadger has a slight CPU overhead, two of the tested extensions even led to a reduction of the overall CPU usage of the tested web browser (*Disconnect* and *uBlock-origin*). Figure 3.9 shows that browser extensions have a considerable impact at overall memory consumption. For this analysis, we first excluded the results from browser instances where we did not collect continuous samples of 30 accessed webpages, because browser instances were restarted since one of the tasks (three webpages of one website) failed. This resulted in different sample subsets for the different browser extensions. We found that all browser extensions resulted in a higher initial memory consumption. After 30 webpages have been accessed, the memory footprint varies depending on the used browser extension. This is because the initial memory consumption due to the presence of each extension is slowly amortized by the resource

Table 3.8: Number of pages with detected fingerprinting services, listed without any extensions (plain) and per blocking extension for **OpenWPM: WebRTC Local IP discovery** (See Table 3.3 for a description of the table header).

	Web Dataset							Android			
	plain	abp	dc	gh	pb	ubo	c	plain	e	a	m
cdn.augur.io/augur.min\.js	111	31	4	43	57	21	2	0	0	0	0
click.sabavision.com/*/jsEngine\.js	78	54	81	84	77	56	46	0	0	0	0
static.fraudmetrix.cn/fm\.js	11	11	11	11	14	11	9	0	0	0	0
.hwcdn.net/fp/Scripts/PixelBundle\.js	2	1	1	1	1	0	0	0	0	0	0
www.cdn-net.com/cc\.js	15	14	17	17	14	16	11	0	0	0	0
scripts.poll-maker.com/3012/scpolls\.js	3	3	3	3	4	3	3	0	0	0	0
static-hw.xvideos.com/vote/displayFlash\.js	6	7	9	9	7	10	6	0	0	0	0
g.alicdn.com/security/umscript/3.0.11/um\.js	0	0	0	0	0	0	0	0	0	0	0
load.instinctiveads.com/s/js/afp\.js	0	0	0	0	0	0	0	0	0	0	0
cdn4.forter.com/script\.js	4	4	4	2	5	0	0	0	0	0	0
socauth.privatbank.ua/cp/handler\.html	2	3	2	1	0	1	0	0	0	0	0
retailautomata.com/ralib/magento/raa\.js	0	0	0	0	0	0	0	0	0	0	0
live.activeconversion.com/ac\.js	0	0	0	0	0	0	0	0	0	0	0
*.ml.com/publish/ClientLoginUI/HTML/cc\.js	3	2	1	3	2	2	1	0	0	0	0
cdn.geocomply.com/101/gc-html5\.js	0	0	0	0	0	0	0	0	0	0	0
retailautomata.com/ralib/shopifynew/raa\.js	0	0	0	0	0	0	0	0	0	0	0
2nyan.org/animal/	2	2	2	2	2	2	2	0	0	0	0
pixel.infernotions.com/pixel/	6	6	6	1	7	1	1	0	0	0	0
167.88.10.122/ralib/magento/raa\.js	0	0	0	0	0	0	0	0	0	0	0

savings of blocking third-party trackers. AdblockPlus was responsible for a significant memory overhead, while none of the extensions resulted in less memory being used.

3.5 Discussion

3.5.1 Limitations

We tackle the challenge of evaluating the effectiveness of tracker-blocking tools on a large scale. However, like any other large scale study, our work has some limitations.

First, our results on third-party tracking are lower bounds because our analysis does not account for content behind registration walls, since we cannot realistically obtain accounts (paid and free) for thousands of websites. Second, we used the default settings of all browser extensions, with the exception of Ghostery, where we manually activated blocking of third-party trackers (default mode of Ghostery is to only report the presence of trackers but not to block them). Therefore, readers must not misinterpret our experiments and reach the conclusion that merely installing Ghostery currently offers the best protection against trackers. Finally, the PrivacyBadger extension was limited to blocking trackers based on our training on the Alexa top 1,000 websites. As such, in more realistic workloads or usage that spans many days, PrivacyBadger could perform

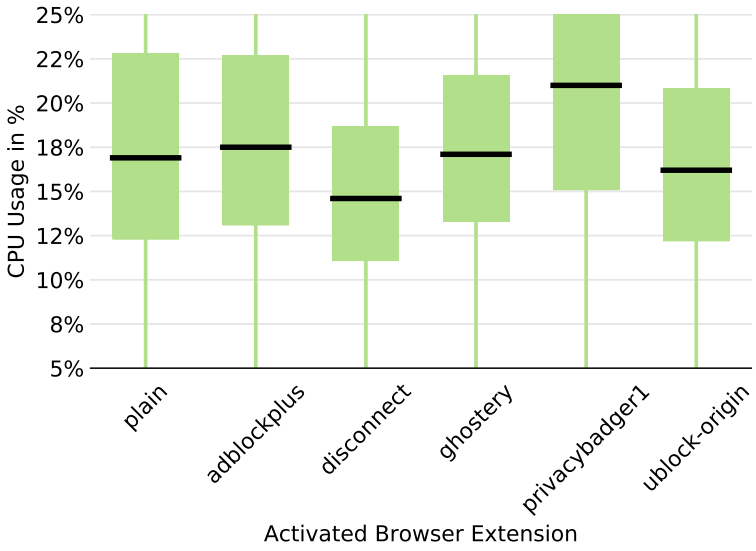


Figure 3.8: Absolute CPU usage for different browser instances for each of the extensions.

better (or worse) than it did in our measurements.

3.5.2 Future Tracking Defenses

Based on our findings we identify the following major challenges for future tracker-blocking browser extensions:

Social widgets are important. Despite the reach and impact of social widgets on the tracking of users, a number of existing browser extensions failed to effectively block Facebook's and Twitter's widgets. Future tracking defenses should focus on the creation of effective surrogates for common social widgets.

Creation of filter rules. We identify the need for research to automate, or at least assist the laborious process of creating tracker-blocking filter rules. Previous research relied on the community-driven EasyList and EasyPrivacy filter rules [51]–[53]. Our findings suggest that the centralized rule sets by Ghostery or Disconnect might provide a better baseline for future research.

Closing blindspots. In addition to the varying effectiveness of different browser extensions regarding different stateful online trackers, all evaluated browser extensions failed to completely block well-known stateless fingerprinting services. We argue that this is because of the opaque nature of fingerprinting which makes it harder for users to spot and hence report. Ideally, novel research into detecting stateless fingerprinters would automatically create blocking rules, since for some of the identified fingerprinters even after three years no filter rules exist. Finally, our results suggest that the proprietary filter-

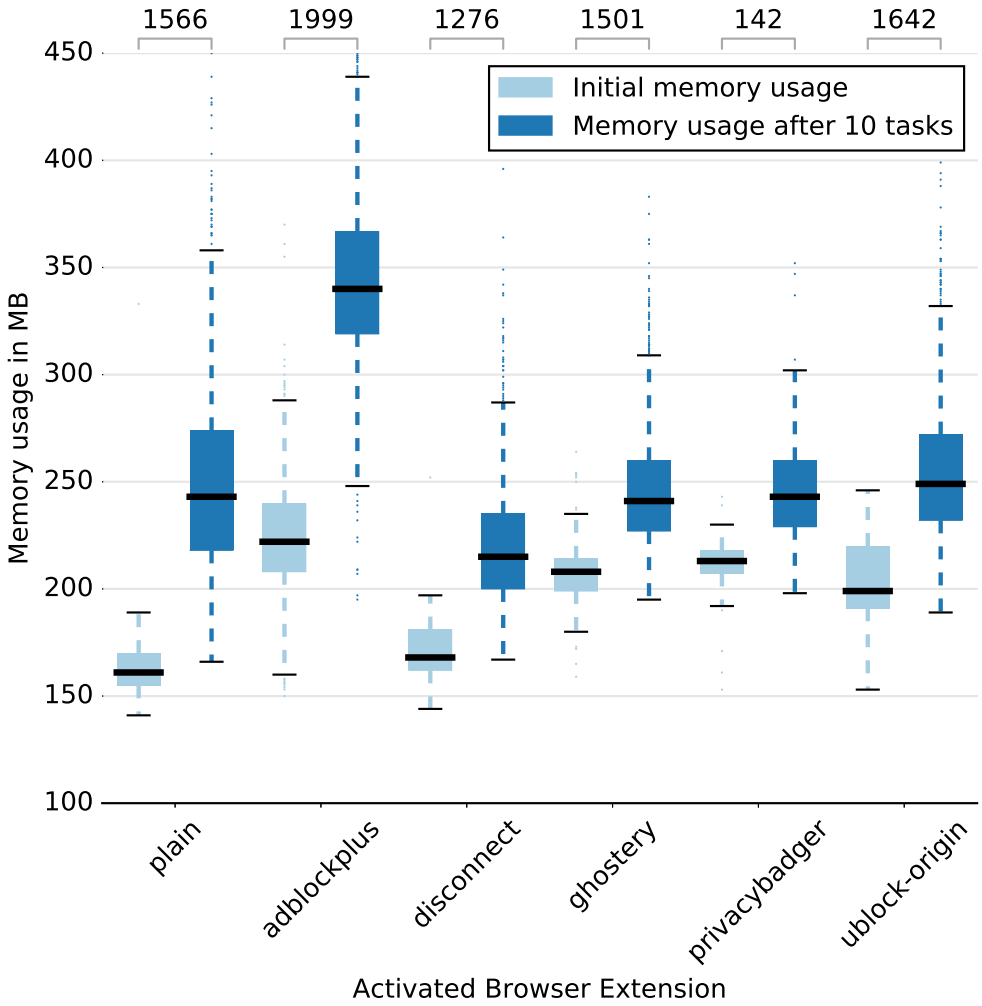


Figure 3.9: Initial memory usage and memory usage after requesting 30 web pages per browser extensions in MB. The numbers on the top show the amount of samples for each of the extensions.

rules of Ghostery should be complemented with community-based ones from uBlock, to account for less popular third-party trackers.

Methodology for detecting broken websites. Our findings highlight an important research challenge for the automated creation of blocking rules: the risk of breaking websites. The heuristic creation of blocking rules with EFF's Privacy badger showed promising effectiveness but also led to the highest number of unresponsive websites in our sample. Future research should therefore focus on methods to automatically detect whether a certain rule breaks the functionality of websites.

Provide for content distribution networks. This ideal combination of filter-rules would ultimately reduce third-party services to content distributions networks, which cannot be blocked without breaking the functionality of websites. The growing usage of CDNs could ultimately thwart all existing tracker-blocking tools since CDN providers could deploy probabilistic stateless tracking based on the IP address and user agent of their users. A possible countermeasure for future tracker blocking solutions, would be the inclusion of popular JavaScript libraries/fonts to make requests to CDNs unnecessary. There is currently one browser extension for Mozilla Firefox which complements tracker blocking browser extensions by providing popular CDN content locally[63].

Mobile in-app tracking. Mobile devices are often neglected in the discussion of third-party tracking protection, despite their growing usage. Currently, the most common browser extensions for blocking web trackers are available for Android (through Firefox Mobile) and iOS [96]. Unfortunately, tracking by mobile applications is harder to block. The rooting of mobile phones is beyond the reach of average users, and therefore blocking can only be performed at a network level. Our results showed that AdAway/-MoaAB (DNS-based blocking) and Adblock Plus for Android (proxy-based blocking) fail to significantly impact tracking by third parties on mobile applications.

3.5.3 Online Tracking and Security

In our large scale analysis, blocking extensions did not result in noticeable CPU overhead, and in the case of Disconnect, it even led to reduced CPU usage. The majority of analyzed browser extensions, however, led to an increased memory footprint. Adblock-Plus resulted in the biggest overhead which can be attributed to their use of cosmetic CSS-based filters which hide advertisements and the space they used to occupy, in addition to blocking them. Despite the memory overhead we measured, tracker blocking has additional benefits for the security of users.

Third-party tracking and third-party content in general has been exploited as an attack vector in the past. NSA used tracking identifiers to identify targets for further exploitation by passively analyzing unencrypted traffic en route to third-party tracking services. Our measurements show that over 60% of third-party services did not use TLS to protect third-party requests and responses. Next to passive attacks abusing unprotected requests and responses, a web-wide over-reliance on specific third-party trackers can also be abused by active adversaries. For example, in a recent nation-state attack later dubbed "Great Cannon" [48], the attackers replaced advertisement and analytics code loaded from *baidu.com* with malicious code which performed Denial-of-Service (DoS) attacks against specific targets. This example shows that popular third-party services can make very attractive targets for attackers as a step to reach as many end-users as pos-

[63] https://decentraleyes.org

sible. For example, if attackers would be able, through whatever means, to successfully attack *google-analytics.com*, they would be able to push malicious code to approximately 70% of the top web pages.

Table 3.9: This table outlines common third parties we detected in our sample of 123,876 websites out of the Alexa Top 200,000. The numbers account to the percentage of inclusion in different websites with respect to the total sample (http/https).

		plain *http/https*	adblockplus *http/https*	disconnect *http/https*	ghostery *http/https*	privacybadger *http/https*	ublock-origin *http/https*	combined *http/https*
google-analytics.com	www	13.2/53.6	19.2/47.4	1.4/0.4	0.9/	54.6/13.7	/	/
	ssl	/6.9	/6.8	/	/	/5.7	/	/
		/	/	/	/	/	/	/
doubleclick.net	stats.g	/35.4	0.6/34.6	/	/	/	/	/
	googleads.g	3.3/31.9	1.5/13.7	/	/	/1.4	/	/
	cm.g	13.3/25.9	8.0/3.8	1.7/0.8	/	0.9/	/	/
google.com	www	13.8/41.1	7.7/24.3	6.9/14.6	4.9/13.1	6.8/14.0	5.6/14.0	5.0/10.2
	apis	/14.4	0.5/14.3	/8.5	/1.3	/	/14.1	/
	accounts	/10.1	/9.9	/3.8	/1.3	/	/9.8	/
gstatic.com	fonts	21.2/24.3	20.8/21.9	21.7/19.0	21.0/17.5	20.8/15.5	21.4/22.5	18.2/15.5
	www	4.0/17.3	1.1/4.0	1.4/4.0	0.6/2.9	1.4/3.7	0.6/3.9	0.6/2.1
	ssl	0.5/10.4	0.5/10.2	0.5/4.0	/1.5	/0.5	0.5/10.1	/0.5
googleapis.com	fonts	23.8/15.9	23.6/12.9	24.2/13.5	23.5/13.3	23.6/12.6	23.9/13.2	20.9/12.3
	ajax	16.2/10.3	15.5/9.6	16.2/9.9	15.4/9.6	16.0/9.4	15.5/9.6	13.3/8.3
	maps	1.8/2.4	1.8/2.4	1.8/2.6	1.9/2.6	1.7/2.3	1.8/2.5	1.3/1.8
facebook.com	www	1.4/37.9	1.7/35.7	/0.7	/1.0	/	1.0/22.8	/
	staticxx	2.8/22.7	4.3/22.5	/	/	0.8/0.8	2.9/22.7	/
	graph	1.9/2.0	1.9/1.8	/0.5	/	0.9/1.1	1.0/1.7	/
googlesyndication.com	pagead2	27.5/29.4	0.7/0.7	14.6/1.1	/	16.7/16.1	/	/
	tpc	4.5/20.1	/0.5	/	/	/	/	/
	video-ad-stats	/	/	/	/	/	/	/
facebook.net	connect	8.0/24.1	10.8/20.5	1.8/0.6	/	/	9.5/18.6	/
	www.connect	/	/	/	/	/	/	/
googleadservices.com	www	10.2/6.2	9.9/5.5	5.7/2.4	/	9.2/4.9	/	/
	partner	9.4/0.9	8.1/0.8	0.6/	/	8.8/0.8	/	/
	pagead2	/	/	/	/	/	/	/
twitter.com	platform	13.8/13.6	13.8/13.5	3.5/1.0	/	12.5/11.8	12.6/12.8	/
	syndication	/11.7	/11.5	/	/	/	/5.0	/
	analytics	/5.9	/2.6	/	/	/1.7	/	/
fbcdn.net	static.xx	/19.0	/18.8	/	/0.4	/	/18.5	/
	scontent.xx	/8.9	/8.7	/0.5	/	/0.5	/9.0	/
	external.xx	/1.0	/1.0	/	/	/	/1.1	/
adnxs.com	ib	14.8/4.1	7.2/1.1	1.2/0.4	/	5.7/0.8	/	/
	secure	0.4/4.7	/2.2	/	/	/2.5	/	/
	acdn	1.6/	/	/	/	0.6/	/	/
cloudfront.net	d5nxst8fruw4z	/2.8	/2.7	/2.8	/	/2.6	/	/
	d31qbv1cthcecs	/2.7	/2.6	/2.7	/	/2.5	/	/
	dnn506yrbagrg	1.5/0.4	1.5/0.4	1.5/0.5	/	1.5/0.4	/	/
yahoo.com	ads	7.5/3.2	/	/	/	/	/	/
	pr-bh.ybp	3.2/1.9	/	/	/	/	/	/
	cms.analytics	2.4/	1.7/	/	/	/	/	/
googletagmanager.com	www	9.9/4.5	9.8/4.5	9.9/4.6	/	9.7/4.3	/	/
		/	/	/	/	/	/	/
addthis.com	m	9.8/1.3	9.7/1.3	/	/	/	/	/
	s7	6.3/1.0	6.3/1.1	5.6/1.0	/	/	/	/
	su	5.0/0.4	3.8/	/	/	/	/	/
amazonaws.com	s3	2.2/2.1	2.0/2.0	2.1/2.0	1.7/1.5	2.1/1.8	1.6/1.6	1.3/1.1
	load.s3	2.7/1.0	1.3/	/	/	/	/	/
	cloudfront-labs	2.4/	2.3/	2.4/	/	2.3/	/	/
scorecardresearch.com	b	9.7/	7.3/	0.5/	/	/	/	/
	sb	/3.3	/1.2	/	/	/	/	/
	sa	/0.9	/	/	/	/	/	/
mathtag.com	sync	7.3/2.0	3.2/0.7	/	/	/	/	/
	pixel	4.9/0.7	2.5/	/	/	/	/	/
	tags	1.2/2.4	/	/	/	/	/	/
rlcdn.com	idsync	9.9/2.4	5.3/0.7	0.6/	/	/	/	/
	rc	1.6/	1.3/	/	/	/	/	/
	ei	/	/	/	/	/	/	/
2mdn.net	s0	1.9/9.2	/	/	/	/	/	/
	s1	/4.4	/	/	/	/	/	/
	s0qa	/0.9	/	/	/	/	/	/
adsrvr.org	match	9.0/2.2	/	/	/	/	/	/
	insight	0.7/0.7	/	/	/	/	/	/
	usw-lax	0.6/	/	/	/	/	/	/
openx.net	us-u	7.4/4.0	/	/	/	/	/	/
	us-ads	1.1/	/	/	/	/	/	/
	u	0.9/	/	/	/	/	/	/
bluekai.com	tags	9.2/3.3	4.6/	/	/	0.5/	/	/
	stags	/1.3	/0.5	/	/	/	/	/
	analytics	/	/	/	/	/	/	/
rubiconproject.com	pixel	6.6/4.7	/	/0.4	/	/	/	/
	optimized-by	1.8/	/	/	/	/	/	/
	ads	1.7/	/	/	/	/	/	/
googletagservices.com	www	9.0/2.4	7.6/1.0	0.5/	/	8.4/1.1	/	/
cloudflare.com	cdnjs	3.3/4.3	3.1/2.3	3.3/2.4	3.2/2.4	3.2/2.2	3.2/2.3	2.8/2.0
	ajax	2.3/	2.2/	2.3/0.4	2.3/	2.3/	2.2/	2.0/
	www	/	/	/	/	/	/	/
advertising.com	sync.adaptv	4.2/1.4	/	/	/	/	/	/
	pixel	2.8/0.5	/	/	/	/	/	/
	cas.pxl.ace	1.4/	/	/	/	/	/	/
bidswitch.net	x	7.3/3.2	1.3/1.0	0.8/0.6	/	/	/	/
	useast-aws2	/	/	/	/	/	/	/
	us-east	/	/	/	/	/	/	/

CHAPTER 4

TLS Certificate Validation Extensions

Secure communication is a key part of today's Internet applications. The majority of online applications, ranging from e-mail to VPN and browsing the web, rely on SSL and TLS[64] to provide secure communication mechansims such as authenticity, confidentiality, and integrity. TLS 1.2 is, at the time of writing, the most recent version [113], with TLS 1.3 currently in the making. Trust in the TLS ecosystem is distributed over software vendors and an underlying public key infrastructure (PKI) composed of various certificate authorities (CAs). To establish a secure connection, a client verifies the signature of a server's certificate. If the server's certificate is signed by a trusted certificate authority, the certificate is accepted, otherwise it is rejected. To determine if a CA is to be trusted, the client relies on a so called "trust store", i.e., a list of certificate authorities that it can trust. These trust stores are usually shipped with the application or are included in the operating system. If an attacker gets her hands on one of the private keys of one of these certificate authorities, she is able to issue valid (trusted) certificates for arbitrary-named servers, since the signatures can only be validated against the local trust store. This allows for effective Man-in-the-Middle (MitM) attacks against any kind of targets.

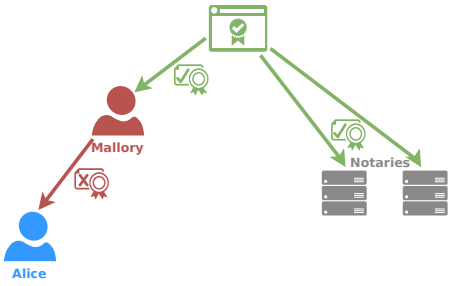
Recent incidents have shown that the subversion of the chain of trust is a viable scenario. Examples include the infamously hacked certificate authorities DigiNotar and Comodo [9], during which their private keys were stolen. Incidents such as the case of Superfish[65] and the Dell eDellroot certificate[66] demonstrate that sometimes even system

[64] Hereafter, we use the term "TLS" to refer to all incarnations of SSL and TLS, if not specified otherwise.

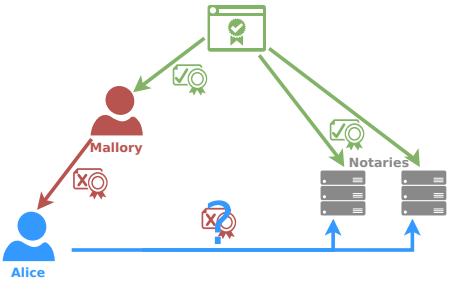
[65] `https://support.lenovo.com/at/de/product_security/superfish`

[66] `http://en.community.dell.com/dell-blogs/direct2dell/b/direct2dell/archive/`

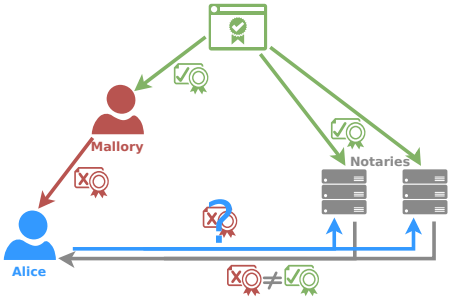
vendors accidentally introduce vulnerabilities. In these cases, trusted certificate authorities were included in the local trust store of the operating system, which also included the private keys to provide extended functionality, allowing everyone to extract the CA private key and launch unnoticed MitM attacks. For affected users, there is nearly no possibility to distinguish between valid server certificates and those signed by fraudulent CAs, since there are no visible distinction marks and the client's software marks them as trusted.



(a) Alice requests a webpage. Mallory intercepts the request and presents a forged certificate.



(b) Alice asks the notaries if they know the certificate.



(c) The notaries' responses tell Alice that the certificate is different from what they have seen.

Figure 4.1: The usual flow of a request for certificate notary services.

To solve the problem of multiple valid and trusted certificate chains, several solutions have been proposed recently. These solutions include DANE [114], public key or certificate pinning using HPKP [14], and TLS notary services. The latter are based on the principle of multi-path probing. Figure 4.1 depicts the usual workflow of such notary services. The idea is to query different "notary" servers if they are presented with the same certificate for a certain communication entity as the client. Therefore, to launch

2015/11/23/response-to-concerns-regarding-edellroot-certificate

an undetected MitM attack, an attacker would need to intercept as well all the connections to the entity that originate from all the queried notary servers. Since these notary servers are usually spread in different networks around the globe, the risk of an effective, unnoticed MitM attack is highly reduced, even if the certificate is trusted by the local trust store. On the other hand, such a system could reduce the dependability on certificate authorities, since the validation does not have to depend on trusted certificate authorities, but could rely solely on the quorum of a set of notary servers.

DANE is far from being usable in practice as it relies on DNSSEC which is still not widely deployed. Certificate and public key pinning are still supported only by selected applications (e.g., Chrome, Firefox, and some mobile apps [10], [11]). On the other hand, TLS notaries are already implemented as browser extensions, thus being usable in practice. However, there is still no complete study on the long-term usage of notary services and how they react to changes in a real-world setting. We therefore implement a modular system to evaluate notary services in the long term and on a daily basis, independently of the used browsers.

The contributions are as follows:

- We present a longitudinal study on the effectiveness of three well-known notary services over a one-year period.
- We describe a concept of mapping multiple TLS notaries for transparent end-user protection and an implementation of it as a proxy service.
- We identify problems of combining these services, including lack of widespread adoption and the problem of view inconsistencies.

The rest of this chapter is organized as follows: In Section 4.1, we describe our concept of a proxy notary. In Section 4.1.3, we describe our methodology for evaluating the proxy as well as the three TLS notary services independently, whereas our results are described in Section 4.2. We discuss these results in Section 4.3.

4.1 Methodology and Measurement Setup

To monitor the effectiveness and behavior of different notary services, we set up an automated crawling environment. Figure 4.2 provides an overview of the overall design.

We implemented the proxy in `mitmproxy` [110], which allowed us to validate certificates through several extension modules. These extension modules implement interfaces to various notary services, which are described in Section 4.1.2. We used this system to collect daily statistics of these implemented extension modules over a one-year period.

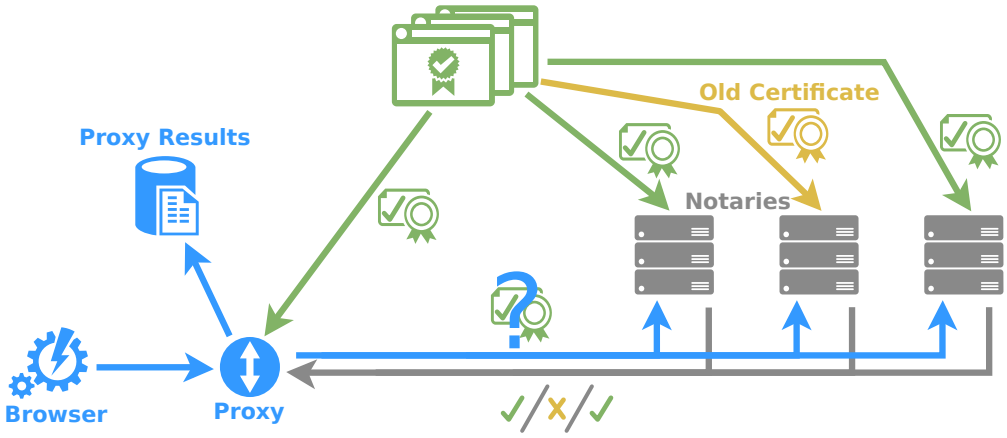


Figure 4.2: Overview of our measurement setup.

4.1.1 Data Collection System

The data collection system was implemented in such a way that it is extensible, reusable and can furthermore be used by the end users to evaluate their own browsing session. Therefore the overall data collection consists of three components: (1) a web browser, (2) an intercepting proxy to monitor HTTPS sessions, and (3) proxy plugins to query various notary services.

Browser

To query the different webpages, we utilized `wget` with the proxy settings pointing to our intercepting proxy. While we used a lightweight, GUI-less browser for our periodic scans, any other browser could be used as well. This makes the validation proxy described in the next paragraph easier to deploy in combination with other systems. End users can use the proxy to secure or evaluate TLS certificates against various notary services without the need to install separate plugins in their browsers.

Intercepting Proxy

To conduct the certificate validation, we implemented an HTTP/HTTPS proxy server in Python 2 using the `mitmproxy` [110] library as a basis. The proxy server acts as an intermediary between the client and the web server. For each encountered HTTPS certificate, the proxy server conducts the certificate validation using the configured notary services.

Proxy Plugins

To make the system extensible, the communication with the notary services is implemented as plugins. This makes it easy to extend our system so as to evaluate additional notary services. The proxy in general supports two modes of operation: synchronous and asynchronous. In synchronous mode, the proxy waits for all the responses from the notary services before the original page is passed to the requesting browser. In case of a validation error, this allows to terminate the page load before the page content is rendered to the user.

The second proxy mode asynchronously collects validation information from the notary services and logs them in the file system for later inspection and analysis. In this mode, the page load cannot be interrupted or terminated, since the page is served to the user without waiting for validation responses. For the evaluation, we only look at the results from the asynchronous mode.

4.1.2 Notary Services

We identified three services that were in use and also had an active and open infrastructure, namely Perspectives, Convergence, and ICSI. We give a short introduction to the inner workings of these systems in the next paragraphs.

Perspectives

Wendlandt et al. [115] pioneered the *multi-path probing* approach: The system employs multiple independent servers, called *notaries*, which observe publicly-visible web servers and store data about their certificates. When a client contacts a server using TLS, it queries a number of notaries. The notaries reply with information about which certificate the server in question was using in which time period. Using this information, the client can make a more informed trust decision: Do the notaries see the same certificate as the client?

Convergence

Moxie Marlinspike developed Convergence[67], which builds on the same design principles as Perspectives, but it incorporates other ideas and principles as well. Its central idea is "trust agility", i.e., the users themselves can choose whom to trust and may also revoke their trust. Similarly to Perspectives, Convergence relies on notaries to decide if a certificate is trustworthy or not. However, the decision process is somewhat different. Using a REST web service API, the client sends a request containing the host, port number, and certificate hash to each notary it wishes to query. The server sends one

[67] `https://github.com/moxie0/Convergence`

out of five different types of responses, which can be distinguished by the HTTP status code. The possible responses are:

- The notary could verify the certificate.
- The notary could not verify the certificate.
- The notary cannot decide whether to accept or reject the certificate; the client should ignore this notary in its trust decision.
- The client sent a malformed request.
- The server could not perform the request due to an internal error.

This approach makes the implementation of a client rather simple, because the client just has to count the votes collected from the notaries. The protocol is described in more detail in [116]. The user can decide whether decisions are based on majority voting or if an unanimous vote is mandatory in order to accept a certificate.

ICSI Certificate Notary

ICSI Notary [117] is a service from the University of Berkeley that monitors certificates. In contrast to the two aforementioned services, the ICSI Certificate Notary passively monitors traffic from multiple Internet sites and builds a database of certificates seen in this traffic.

The database can be queried by clients by issuing a DNS query containing the hash of the certificate. The service responds to the client whether it has observed that certificate in the past, and if it could trace this certificate to a valid root certificate through one of the following responses:

1. ICSI has seen this certificate:
   a) ICSI can establish a chain of trust to a certificate from the Mozilla root store → ICSI replies 127.0.0.2 to the request.
   b) ICSI cannot establish a chain of trust → ICSI replies 127.0.0.1
2. ICSI has not seen this certificate or an error (such as a time-out) has occurred → no reply

Note that it is not possible to distinguish between the cases *"a query timed out"* and *"ICSI has not seen this certificate"*, therefore our proxy plugin rejects the certificate in both cases.

4.1.3 Data Collection

Our data collection involves periodic TLS certificate validation requests to the set of analyzed notary instances for 1,000 web pages. The scans were conducted daily, and each scan involved queries to the three different notary services for each of the encountered certificates. We conducted the evaluation of the validation proxy in two steps: First, we collected a sample set of pages served through HTTPS. Secondly, we conducted daily scans to validate the corresponding TLS certificates against different notary services and analyzed their responses.

Sample Selection

To select samples, we initially obtained the list of Alexa Top 1,000,000 sites [118] on November 29, 2013. From this list, we then selected the top 1,000 sites that responded to an HTTPS query within 30 seconds. This selection represents the websites that attract most of the visits by users, including pages such as Facebook, Twitter, and Google. Many of the selected websites did respond to HTTPS queries, but with an immediate redirection to a (non-secure) HTTP connection. This means that while they do support HTTPS, many users will probably not use it. However, we still included these sites in the evaluation under the assumption that HTTPS is likely to be used in some parts of the website, like the login pages.

Periodic Scan

Between January 31, 2014 and January 29, 2015, for a period of one year, the collection was conducted daily. For each run of the scan, the proxy server was started and the previously selected URLs were queried, with the different notary plugins enabled. The data returned by the proxy plugins as well as the collected certificates were stored for further analysis. To get a baseline for comparison, we also queried the URLs without using a proxy server. Thus, in one evaluation run, each site from our data set was queried for a total of four times.

For each pair of URL and validation method, the following measurements were taken:

Verdict: Whether the validation method *accepted* or *rejected* the site's X.509 certificate.

Reason: The reason why a certificate was rejected, if it had been rejected. This metric is specific to each validation method.

Validation Time: The entire time the validation process of a certificate took, including querying the notary server(s) and waiting for a response.

4.2 Results

In the following, we describe our results and findings from the collected dataset. For each notary service, we analyzed how long it took to answer a validation request and also how long it took to react to certificate changes. Furthermore, we studied the availability of these services over the course of one year.

4.2.1 Certificate Changes

To analyse the functionality of notary services, it is important to observe actual certificate changes. Figure 4.3 depicts the number of different certificates per website we encountered during the course of our study. In 80% of the cases, the websites changed at least once their certificate; some 10% of them changed more than 9 times their certificate within the one year that our study was active.

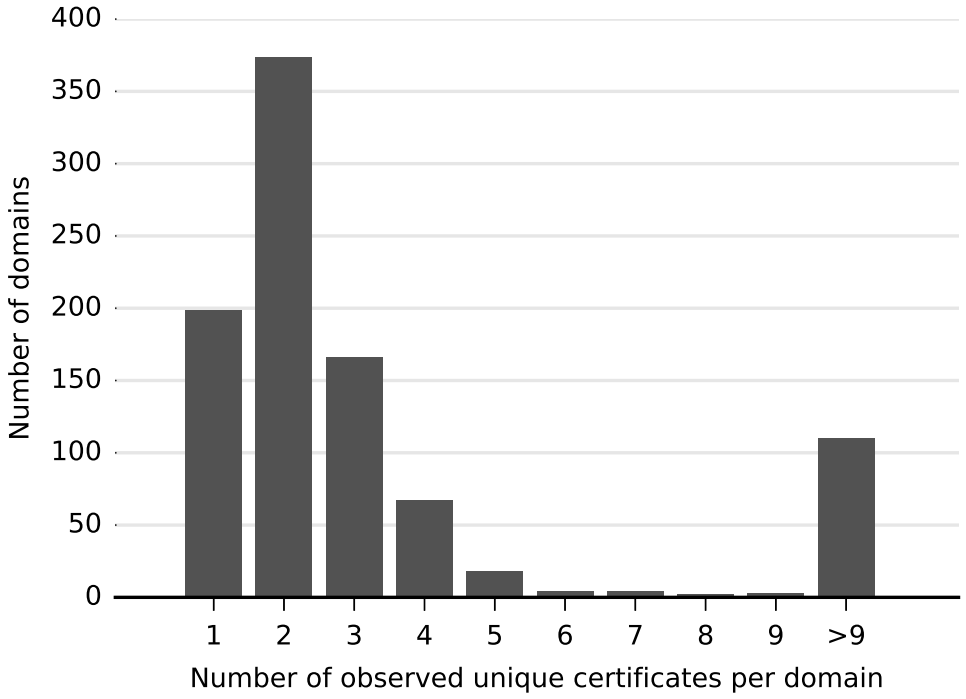


Figure 4.3: Number of different certificates observed for each tracked domain

4.2.2 Validation Time

An important factor concerning notary services from a usability point of view is their response time to validation requests. Therefore, we conducted an analysis of the response time of the various services. With the 1,000 webpages crawled daily for one year, we

collected in total more than 350,000 response timing samples per analyzed notary service. Figure 4.4 summarizes the timing information for the three notary services.

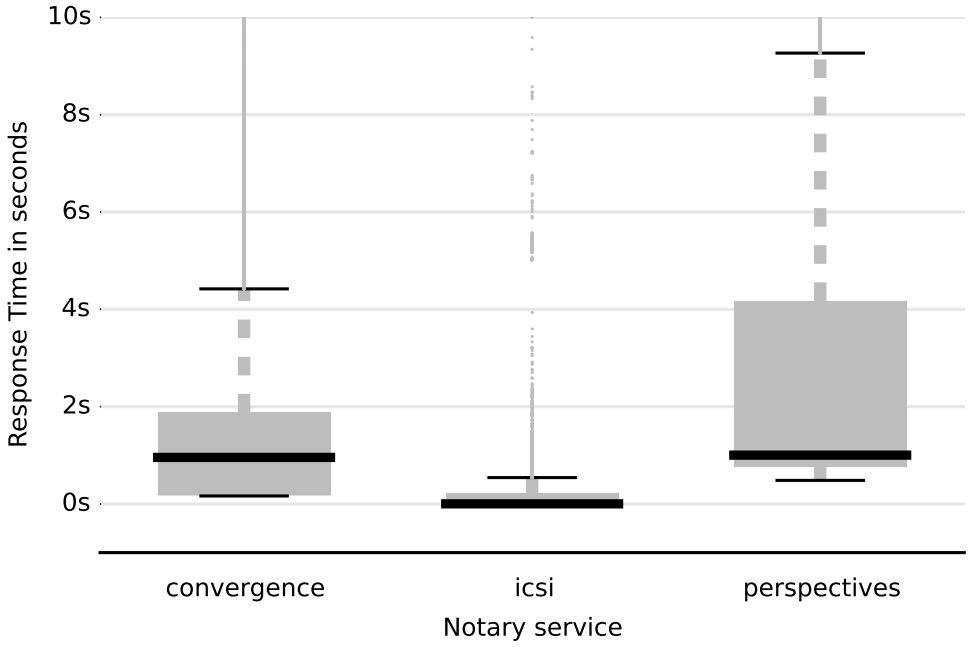


Figure 4.4: Response times of notary services to validation requests. Outliers are cut off at 10 seconds.

The DNS-based approach of ICSI yields the fastest responses to queries, with the majority (95%) of answers received in under one second. While about half of the responses for Convergence and Perspectives are also below this mark, response times for these two services have a far higher fluctuation. This can be an issue in the case where the notary services are used to validate certificates before a page is loaded, as it could introduce noticeable page load delays for the users. We note that Convergence usually employs a client cache for fingerprints, in an effort to improve the loading times. We did not implement this caching in our proxy so as to get a comparison of the notary service based on newly-encountered pages.

4.2.3 Certificate Acceptance Duration

While the response time is certainly important for the general usability in day-to-day browsing, another temporal factor to take into consideration is the time a notary needs to mark new or changed certificates as valid once they are introduced or updated. Figure 4.5 depicts the time it took the different services to mark a new certificate as valid after it was changed on the server. Since we conducted daily crawls at a fixed time, the

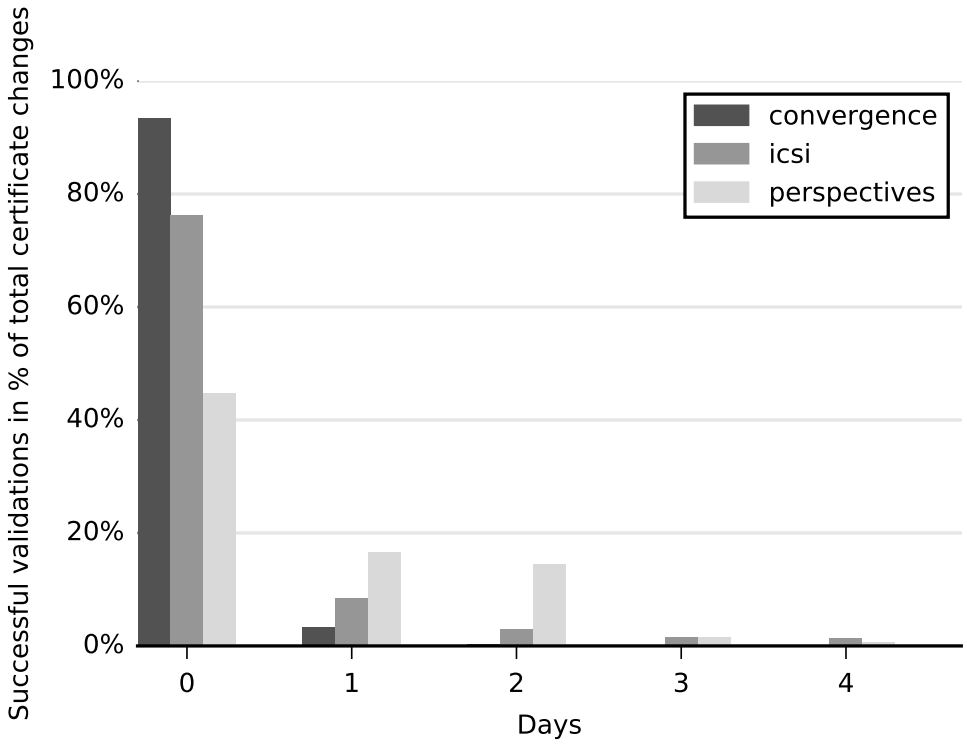


Figure 4.5: Time until a newly-seen certificate is marked as validated in percent of the total of observed certificate changes within the period of one year.

resolution of our scan is also on a daily basis. Therefore, a value of zero days means that the certificate was changed by a server as well as validated by a service within this 24-hour time frame. The validations are set in relation to the total amount of certificate changes that we observed during our scanning period. In the case of Convergence we only considered the server we setup and did not include the official server results, since the latter only responded in error for the majority of our scans.

It takes less time to Convergence so as to adopt to newly-changed certificates, with the majority of certificates seen as valid within the first 24-hour time frame, as depicted in Figure 4.5. ICSI is only able to validate about 75% of changed certificates within the same time frame. This fact could be due to the nature of ICSI, which relies on passive information collection, whereas Convergence actively probes servers itself. The relatively low validation rate of Perspectives (45%) can most likely be accredited to the fact that more and more of the servers failed; in the end, it was not possible to reach a quorum on the validity of a certain certificate. Therefore, some of the changed certificates could not be validated successfully anymore. However, even with these limitations in mind, we can still see the general trend that it takes a longer time for Perspectives to successfully

validate certificates compared to the other two services. It takes one day for Convergence and at most three for ICSI to fully synchronize.

4.2.4 Service Availability

To use notary services effectively for certificate validation, it is necessary that a sufficiently large set of servers is reachable. Otherwise the decision if a certificate should be accepted or not is either based on a small set of servers, which makes interception easier, or no consensus on the state of the certificate can be reached at all.

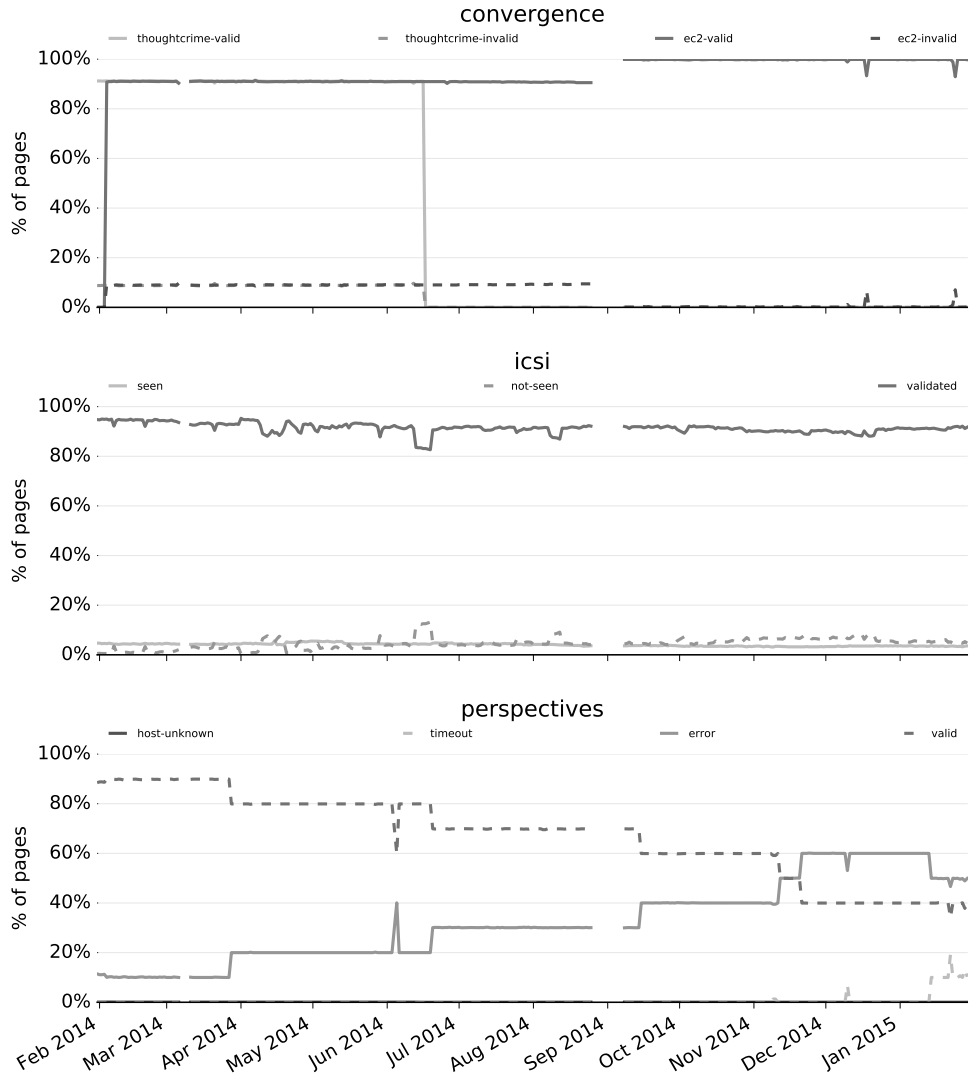


Figure 4.6: Timeline of the responses collected from the different services over the course of one year.

Figure 4.6 provides an overview on the status of certificate validation of the three notary services during the course of one year. The return state for each of the services is given as a percentage for all the collected service responses. It shows the daily average of responses to the 1,000 page request made by the respective crawler.

ICSI was constantly up and running during our scan. We experienced several problems with both Convergence and Perspectives. The analysis of Convergence was based on two servers. The first one was the official server available at *notary.thoughtcrime.org* and the second one was a server we hosted on an Amazon EC2 instance. The official server became unresponsive in the middle of June 2014.

We encountered a similar problem with Perspectives, where the initially-available servers one after the other shut down or responded in error. As described earlier, the Perspectives validation of certificates operates with a quorum-based approach, in which at least a certain amount of servers must provide a valid response. Due to the fact that more servers answered with an error state, this requirement was no longer met and therefore, from a certain point in time, all certificates were rejected by the system, even if some of the servers still provided a valid response.

4.3 Effectiveness of Notary Services

Analyzing notary services on a longitudinal scale reveals several problems and shortcomings that limit the usability of these services. In the following we discuss the observed limitations and possible future directions for the deployment of notary services.

4.3.1 Response and Validation Times

One problem with notary services is the delay that these services introduce in page requests. As we described in Section 4.1.1, there are two approaches to verify a certificate through a notary: synchronous and asynchronous. Both approaches have positive and negative sides. Since the synchronous method waits for all notary responses before actually requesting the page, it can introduce a significant delay (as shown in Section 4.2.2) in page loading, especially if a notary server times out. On the other hand, the asynchronous method loads the page before it receives all notary responses, therefore leaving a window of exposure before notifying the user that something went wrong.

Another problem is the reaction to legitimate certificate changes, namely how long it takes until a service marks a newly seen certificate as valid. Our study shows that it can take up to several days until a certificate is considered as valid. Until the new certificate is validated and has been seen by all the notary services, it will appear as an MitM attack.

4.3.2 Adoption and Continuous Operation

For a notary service (or multi-path probing in general) to be useful for actually validating certificates, there are two important factors that need to be met: (i) Services need to be adopted by the users. This means that users have to run their own servers which others can query. For example, if there is only one official server you can query, this defeats the whole concept. (ii) It implies that one has to fully trust this service, which introduces a single point of failure. A single server could just provide wrong answers to the client's queries without a possibility to check these claims, which would be similar to a device-hosted trust store. On the other hand, even if users set up their own servers, the question is how long they can keep them up and running for other clients to use. Therefore, an important factor to consider is that the amount of available servers could fluctuate. The clients need to be informed of failing servers, since this influences the weight of still-running services in the case of majority voting.

Currently it seems that the adoption of these services by users is low. At the end of the study, the Firefox Add-on for Perspectives has 5,334 users[68] and the plugin for Convergence only 77 users[69]. During our study some of the official servers seem to be discontinued, which does not help to increase the trust in this system. What our insights show is that either the incentive for the users to host their own notary services has to be increased or the system itself has to be adapted. One possible adaption is presented by *tofu*[70], proposing a P2P-based system in which every client is automatically also a host. While this system may be able to solve the problem of service availability, it could still impose further risks that need to be analyzed in the future.

4.3.3 Privacy

Beside the technical aspects, some other must be considered as well. One is the possibility of privacy implications. By using a third-party service to validate certificates, it is easy for its server(s) to collect information about the pages a client visited. Therefore it is possible for the server(s) to build a browsing profile of the specific user. One solution to this problem is for the users to host their own servers. However, this is not always an option and future research should focus on the possibilities to validate certificates without giving away too much information about the client.

While we do not have concrete solutions to these problems (yet), we believe that notaries are a viable alternative to increase the overall security of TLS. Thus, they should be studied further so as to overcome the current limitations.

[68] https://addons.mozilla.org/en-US/firefox/addon/perspectives/
[69] https://addons.mozilla.org/en-US/firefox/addon/convergence-extra
[70] https://gun.io/blog/tofu-web-security/

CHAPTER 5

Privileged Applications to Improve Network Communication Security

5.1 Introduction

TLS certificate pinning is a proposed approach to defend against man-in-the-middle (MitM) attacks for HTTPS communications. The implementation of certificate pinning for mobile applications, and especially for Google Android apps, is cumbersome and error-prone. This results in inappropriate connection handling and privacy leaks of user information.

We propose a design to realize TLS notary-assisted certificate pinning as a means to transparently defend against MitM attacks on behalf of all installed applications in a device. The collective knowledge provided by trusted notary services can increase both the security and the usability of the Android devices.

In this chapter we therefore present the following contributions:

- We describe a new design for implementing certificate pinning at the Android Runtime layer defending against a revised threat model with stronger adversaries.
- We enrich the certificate pinning decision with TLS notary-assisted information.
- We evaluate the proposed design and show that *both* security and usability are increased *without* introducing noticeable overhead.

- We describe a proof-of-concept implementation of the design for the Google Android platform that demonstrates its applicability and feasibility.

5.2 Design and System Architecture

In this Section, we describe the design and the architecture of an enhanced system supporting certificate pinning for Google Android mobile applications.

Our design aims to address the shortcomings of previous approaches mentioned in Section 2.1.6. More specifically, we aim to offer: (i) increased security, by relying on the collective knowledge of TLS notaries; (ii) transparent protection for all installed applications; and (iii) increased usability, by reducing the user burden and involvement in the bare minimum.

5.2.1 Threat model

We assume a threat model where an attacker is able to launch MitM attacks on the connection between an application installed on a Google Android device and its respective web server. The attacker can intercept the TLS connection phase and inject a fraudulent certificate towards establishing a fake connection. Further, we assume that the attacker is located closely to its victim (network-wise), e.g., in a fake wireless access point but they cannot intercept communications in other parts of the Internet (e.g., between the web server and the TLS notary service nodes).

5.2.2 System design

The design of our system follows closely the system architecture of Google Android. The latter comprises four layers, as depicted in Figure 5.1. Following a bottom-up approach, the first layer is the *Linux Kernel*, containing all the necessary drivers that power all of the functionalities presented by the Android applications. The next layer contains the essential *Libraries* and the *Android Runtime*. The latter consists of the *Core Libraries* and the *Dalvik Virtual Machine* (DVM). Every Android application executes in its own DVM. The *Application Framework* layer provides functionalities such as views, activity manager, window manager, telephony, and location services.

Our design introduces a "Notary-assisted Pinning" component in the Android Runtime layer. The component can interface directly with the low-level functionality offered by the Linux kernel, without application intervention. Furthermore, at this level, it can hook and protect transparently all installed applications, without requiring special application logic implemented in the latter.

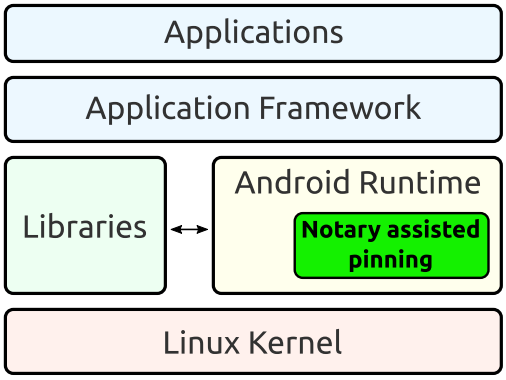


Figure 5.1: Notary-assisted pinning in Android architecture

5.2.3 Component functionality

When an application initiates a TLS connection, the related Android library call is intercepted and the control is passed to the component. The component workflow is depicted in Figure 5.2.

The component will first check if the presented certificate refers to an already pinned web server certificate. If yes, then the certificate hash is compared with the pinned one. In case a match is found, the connection is allowed and the control passes back to the calling application. In case a match is not found, the component initiates a transaction with one or more TLS notaries and checks if the hash is known to them. Once a positive conclusion is made, the control passes back to the calling application. If a negative conclusion is made, the connection is terminated and the control passes back to the calling application. If a conclusion cannot be reached (e.g., the notaries are not accessible or there is no consensus in their replies), the component can either (i) terminate the attempted and possibly untrusted connection or (ii) as a last resort, generate a warning message and ask for action confirmation by the user.

If an entry for the specific web server is not already present in the pinning database, i.e., the web site was not visited until now, then the component will again resort to the collective knowledge of the notaries. If a positive decision can be made, the pinning database is updated accordingly.

5.3 Evaluation

The evaluation of our design is based on three dimensions, namely functionality, security, and usability. We analyze each of them in the next paragraphs.

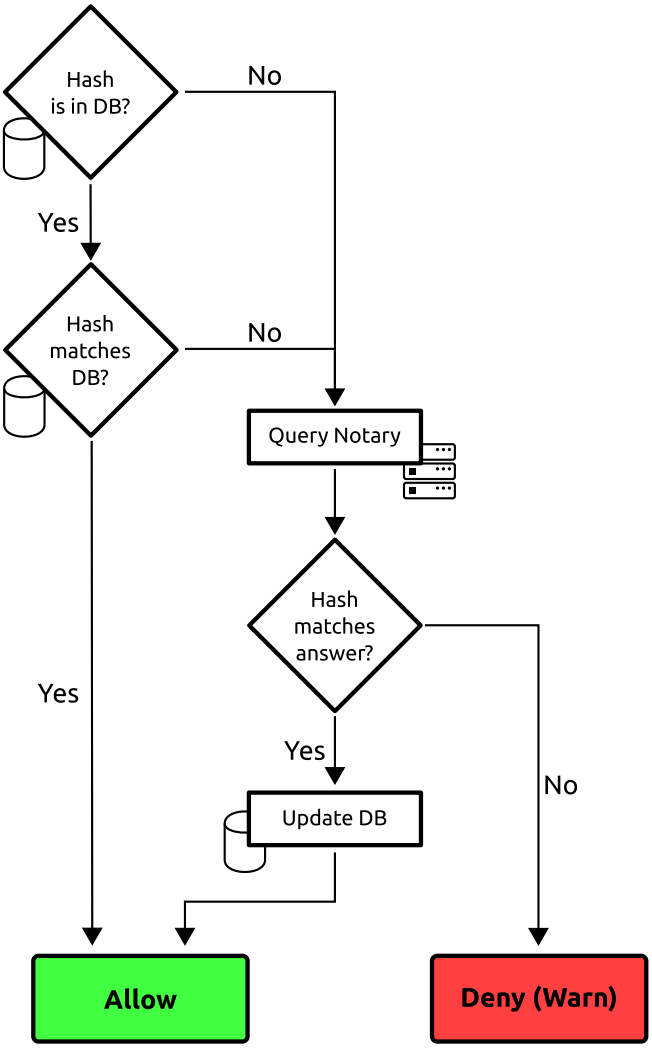


Figure 5.2: Certificate Validation Workflow

5.3.1 Functionality

The proposed design does not require modifications to the functionality of the installed applications or some additional effort for the developers, as it operates transparently at a lower layer of abstraction (namely, Android Runtime). Furthermore, the system interactions involve only actions that are common to TLS processing and certificate validation (e.g., cryptographic computations and network exchanges). Some response latency can be expected and considered acceptable, as it is similar to periodically consulting a CA using OCSP.

Our design does not interfere with the recently-announced Android N certificate pinning

functionality, as the latter is implemented in the application layer. Thus, we consider that our design will actually *enhance* the provided functionality and also provide an additional layer of defense, in the meantime of distributing a new version of the application through the app stores. Finally, the proposed design neither depends on a specific Android device model nor a specific application.

5.3.2 Security

We argue that our proposed design offers increased security for the device owners. We are the first to offer certificate pinning functionality for all Android applications. This provides a first layer of defense against MitM attacks that present a different albeit valid certificate for a known web server. Even if such a certificate is presented, our design relies on the collective knowledge of trusted TLS notary services for evaluating the new information. This is a significant improvement over the "Pin It!" alternative, which relies on the user comprehending the TLS warning messages and making an informed decision [34].

The utilization of TLS notaries provides an additional layer of defense against TOFU-based attacks. While it might be the first time for a device to be presented with a specific certificate, it is highly improbable that the notaries have not seen it already. Hence, the local lack of knowledge is accommodated through the collective knowledge of the notaries.

We expect that the developers themselves are among the first to install their application, once it becomes available in the app stores. Hence, they will feed the notaries with trusted information at a very early stage, even before the general availability of the application. This can happen, for example, during the period that they bring online their web server and perform the necessary pre-deployment quality assurance tests.

TLS Notary Services operate under the multi-path probing principle [23]. Hence, for launching a successful attack, one must be able to interfere with all paths to the notaries and inject fake certificates in their databases. This attack is outside the threat model that is described in Section 5.2.1. Yet, we note that even if this attack is successful in first place, the window of opportunity for an attacker would be rather small. We expect that the notary operators will sooner than later detect the poisoning and remove the fraudulent certificates.

The above analysis leaves one path for an adversary to exploit. This is the communication path between the Android device and the notaries. At this stage, the adversary must present an unseen certificate to the device, so as to force the communication with the notaries. If the device cannot communicate with the notaries, then our component will (preferably) drop the TLS connection as well. We consider this as a better alternative to issuing a warning to the user for further action. Still it is a configurable option for the

user to choose. If the connection is dropped, then no information will be transmitted and leaked to the intercepting web server that is under the control of the adversary. Hence, the security is maintained. It should be noted that this behavior leads to a denial-of-service (DoS) attack. Protection against DoS attacks is outside the scope of our design and of certificate pinning in general; the aim is to secure the transmitted information from eavesdroppers. In this case, some other countermeasures should be employed (e.g., switching to an alternative network, for example a cellular connection, or delaying transmission until the device moves away the network reach of the adversary).

An adversary could also try to impersonate as a notary or alter the responses of the legitimate notaries, instead of denying the connection to the notaries altogether. By and large, this depends on the implementation of the notary query interface. If a secure connection is realized for this (e.g., over TLS), then the adversary will not be able to intercept or impersonate the notary. This leads to a TOFU-based attack scenario again. However, the certificates of the notaries can be embedded in our component and thus, defend against this threat. We expect that the notaries will be well-defended, using state-of-the-art technology. Thus, we consider as minimal, if existent at all, the risk of revocing their certificates due to a security incident. Even in that unfortunate case, it will take just an application update to restore the correct functionality. Furthermore, since the notaries do not operate on a per-application basis, this would not be a targeted attack against a specific application, device, or user but rather a generalized attack against the notary infrastructure itself.

If a clear text connection is realized for the communication with the notaries (e.g., the DNS-based query interface of ICSI Certificate Notary), then there is always the possibility for an adversary to manipulate the responses. Hence, it is necessary to realize an underlying secure channel, so as at least be able to detect fake responses (e.g., require that all responses are signed with a trusted key).

5.3.3 Usability

We consider that our design improves the usability of certificate pinning for the device user, on top of the increased security. The “Pin It!” approach involves the device user in the trust decision for each and every certificate that does not match the stored one and for each and every newly-visited web site, where no information could have been stored already. This is far from optimal, especially if this involvement results in breaking their mental model for their primary task at hand, so as to cope with a secondary one [119].

Our approach avoids the involvement of the user as much as possible and relies on the collective knowledge of the TLS notaries instead. As depicted in Figure 5.2, the design relies on user involvement as an optional step (bottom right). This happens *only* when the certificate hash presented by the visited web site does not match (i) the already-

pinned one and (ii) the one that notaries are aware of or (iii) have not ever seen a certificate for this web site. Even in this very rare case, it is a configurable option either to deny the connection automatically (preferable) or ask the user to confirm and continue their visit at their own risk. So, it takes just a few users to populate the notary servers and then all users utilize the collective knowledge, a clear improvement for both usability and security.

It should be also noted that our approach works automatically for all installed applications, without user intervention or action, further reducing the burden for dealing with secondary security tasks. At the same time, the user is relieved from the risk of accepting a forged certificate and the certificate pinning functionality is performed automatically for them.

5.4 Proof-of-Concept Implementation

We implemented an Android component as a proof-of-concept (PoC) of our design to study its behavior in a realistic environment. There are many frameworks available that allow the on-device dynamic instrumentation and ease the development [120]. These frameworks allow to target, intercept, and modify specific library calls.

We opted for the Cydia Substrate framework[71], based on the analysis of [120]. Cydia Substrate is a dynamic instrumentation framework that enables interception and/or modification of system and application calls. Just by itself, Substrate does not provide any specific functionality. It acts as a platform (base) for developing particular modules, known as "extensions". The framework itself modifies the core of the Android system by injecting specific `jar` files. This gives the opportunity for the developers to intercept and manipulate the application and system calls. This is the only reason for its essential requirement which is the root privilege. Non-rooted devices keep this part of the system protected from performing any changes. Currently, the Google Android security model does not allow modification of the Android Runtime, hence, our component must be installed on a rooted device.

We base our PoC on the codebase of "Pin It!" that is readily available as open-source software[72] and realizes the basic certificate functionality already [34]. We enhanced the implementation to include the application logic to assist the certificate pinning decision based on information provided by notaries, as depicted in Figure 5.3. For the PoC, we integrate the query interface provided by the ICSI Certificate Notary service, which is publicly-available. The ICSI Certificate Notary service is consulted on the first encounter of a certificate and in case a certificate change is detected for an already pinned certifi-

[71] `http://www.cydiasubstrate.com/`
[72] `https://github.com/dbuhov/pinningTrustManager`

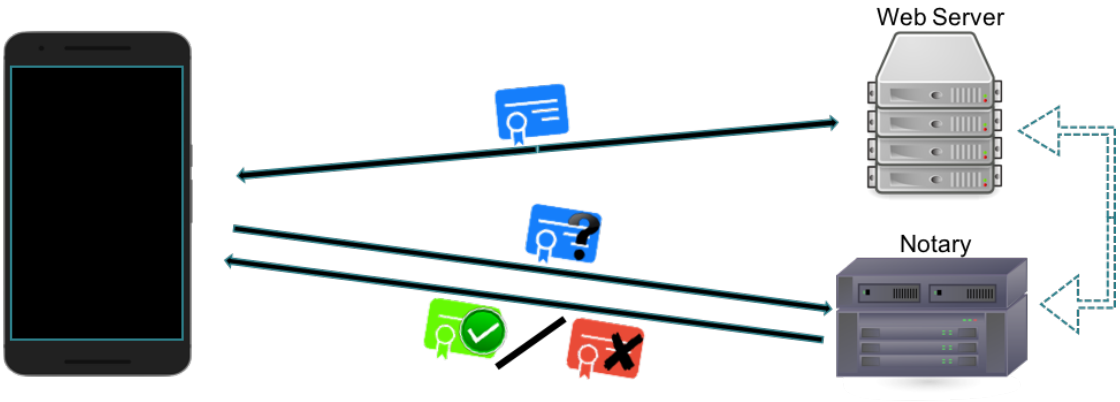


Figure 5.3: Notary-assisted certificate pinning

cate. In all other cases, the received certificates are checked against the pinned ones, as previously. If a mismatch is detected, the connection is terminated and an appropriate notification is issued for the user (cf. Figure 5.4).

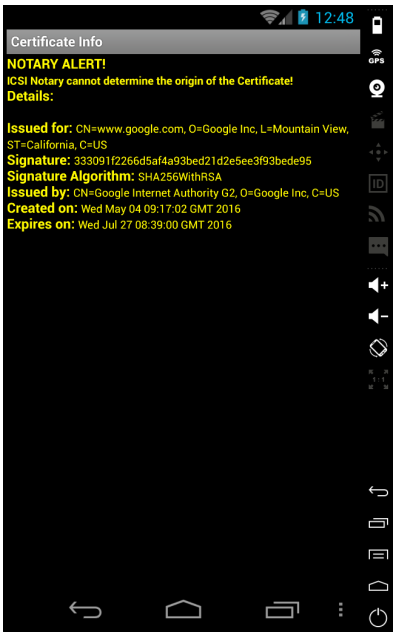


Figure 5.4: User notification from inside the PoC implementation

We experimented with our PoC implementation using valid and expired certificates. We did not notice any application problems (e.g., crashes or freezes) or noticeable delays. The query interface of the ICSI Notary Service was quite stable and the latency introduced by the additional DNS query was indistinguishable from normal network opera-

tions and the heavy cryptographic operations involved in the TLS connection setup. We note the communication with the notary is very sporadic in nature anyway: only on first encounter and when the certificate changes. The empirical evidence for the TLS notary operation that we presented in Chapter 4 shows that certificate changes occur every few months at the most frequent [23]. Overall, the PoC implementation confirms that our design is sound and feasible to implement in the Google Android environment.

CHAPTER 6

Analysis of Malicious Android Applications

With an estimated market share of about 86%, Android has become the most popular operating system for smartphones and tablets [12].

An additional incentive for mobile malware authors to target Android instead of other mobile platforms is Android's open design that allows users to install applications from a variety of sources. However, the diversity of third-party app stores and volume of apps published poses a considerable challenge to security researchers and app store administrators when trying to identify malicious applications.

With over 1 million apps available for download via the official Google Play Store[73] alone, and possibly another million spread among third-party app stores, it is possible to estimate that over 20,000 new applications are being released every month[74]. More than 70.000 new applications were released in Google Play in March 2017[75] This requires scalable solutions for quickly analyzing new apps in order to isolate malicious and other possibly unwanted apps. Google reacted to the growing interest of miscreants in Android by introducing *Bouncer* in February 2012, a dynamic analysis sandbox that automatically checks apps submitted to the Google Play Store for malware[76]. However, research has shown that Bouncer's detection rate is still fairly low and that it can be easily bypassed [121], [122].

[73] http://mashable.com/2013/07/24/google-play-1-million

[74] http://www.appbrain.com/stats/number-of-android-apps

[75] Appbrain (https://www.appbrain.com/stats/number-of-android-apps), Accessed: 2017-04-01

[76] http://googlemobile.blogspot.com/2012/02/android-and-security.html

More than a year before Google's deployment of *Bouncer*, Bläsing et al. [123] were the first to present a dynamic analysis platform for Android applications called *AASandbox* (Android Application Sandbox). Since then, an ever increasing number of dynamic sandboxes for analyzing potentially malicious Android applications have been introduced in academia and in the industry. Similar to dynamic malware analysis platforms for Windows binaries such as CWSandbox [124] or Anubis [125], dynamic sandboxes for Android run targeted applications in a controlled environment to obtain a behavioral footprint. Results are normally presented to the user in the form of a report, possibly including a classification that indicates whether the app is benign or malicious. Most of these systems use some form of hybrid analysis, i.e., leveraging additional static analysis during a preprocessing phase to enhance the dynamic analysis results.

Egele et al. [126] systemized the existing knowledge on dynamic analysis of traditional malware. However, no such survey on dynamic malware analysis in a mobile context exists. In this chapter, we systematically analyze the state-of-the-art for the dynamic analysis approaches for the Android platform and compare existing approaches in terms of provided features and analysis effectiveness.

In particular, we provide the following contributions:

- We survey current state-of-the-art Android malware detection techniques.
- We discuss methods to detect and fingerprint dynamic analysis sandboxes.
- We compare 16 dynamic analysis platforms for Android regarding their features, level of introspection, functionality and interdependencies.
- We evaluate the effectiveness of ten of these dynamic sandboxes using a selected set of malware samples from publicly available malware corpora. Furthermore, we analyze the susceptibility of these dynamic sandboxes to the so-called Master Key vulnerabilities.

The remainder of this chapter is organized as follows: Section 6.1 gives an overview of current Android malware behavior and distribution techniques. Section 6.2 discusses Android analysis techniques and numerous existing malware analysis frameworks. We then discuss our evaluation criteria and sandbox interdependencies in Section 6.3. Finally, we compare the sandboxes and evaluate their effectiveness and limitations in Section 6.4.

6.1 Study Design

In order to gain a better understanding of the requirements for Android malware detection techniques to successfully analyze mobile malware samples, we give a brief overview

on the current mobile malware threat landscape. In the following we discuss the motivation of mobile malware authors, different distribution methods as well as available malware data sets.

6.1.1 Motivation

In August 2010, the first malicious Android application named *AndroidOS.FakePlayer* was discovered in the wild[77]. FakePlayer monetized infected devices by secretly sending text messages to premium numbers. Since then, both the sophistication as well as the amount of observed malware samples increased steadily. Reports focusing on mobile malware trends estimate that the number of malicious Android apps now ranges from 120,000 to 718,000 [127], [128].

The main motivation for mobile malware authors is *financial gain*. One way to monetize infected devices, leveraged by FakePlayer and countless other malware families since then, is by sending text messages to premium numbers, registered to malware authors. In addition to these so called *toll fraud* schemes, malware authors leverage their apps to spy on users and collect personal information. Mobile spyware has capabilities to forward private data to a remote server under the control of malware authors. In a more complex form, the malware could also receive commands from the server to start specific activities and become part of a *botnet*. Furthermore, mobile versions of the banking Trojan ZeuS (ZeuS-in-the-Mobile, or ZitMo) are a way to circumvent the two-factor authentication of online banking systems by stealing mobile TAN codes[78]. Broadcast receivers are an Android-specific feature of particular interest to malware authors as they can be used to launch a background service as soon as the device is started and secretly intercept and forward incoming text messages to a remote server. This capability was used in the aforementioned Eurograbber attack [129] in order to authorize financial transactions.

6.1.2 Distribution

To lure victims into installing malicious apps, a common strategy employed by malware authors is to *repackage* popular applications with malicious payloads. The target applications often include paid applications, which are then offered for "free". Attackers commonly use third-party marketplaces to distribute their repackaged applications, as these marketplaces fail to verify submitted apps. Juniper Networks, for instance, found that malicious applications often originate from third-party marketplaces, with China and Russia being the world's leading suppliers [130]. Zhou et al. [131] analyzed repackaged apps in six alternative Android marketplaces and found that in addition to repackaging apps with malicious payloads, repackaged apps furthermore modify embed-

[77] https://www.securelist.com/en/blog/2254/First_SMS_Trojan_for_Android
[78] http://www.securelist.com/en/analysis/204792194

ded advertising libraries to steal the ad-revenue of application developers.

Attackers can also leverage *drive-by downloads* adapted to the mobile context. Methods to trick users into installing malicious apps include in-app advertising[79], specially crafted websites, or QR codes [132]. Drive-by downloads might, however, also exploit *platform-level vulnerabilities* to install malware in a stealth fashion. Due to Android's loose management to device software, Android versions have become fragmented, with only 1.8% of all devices running the latest Android version 4.4 (codename KitKat)[80], as of February 2014. This fragmentation makes new security features, as well as bugfixes for core components preventing against arbitrary code execution exploits, only available to a small group of users. Android versions prior to 2.3.7 are especially vulnerable to root exploits (examples include *RageAgainstTheCage*[81], *Exploid*[82] and *zergRush*[83]). While these exploits were originally developed to overcome limitations that carriers and hardware manufactures put on some devices, they have also been used by malware authors to obtain a higher privilege level without a user's consent. This approach allows malware to request only a few permissions during app installation, but still gaining root access to the entire system once the app is executed.

6.1.3 Malware Data Sets

Access to known Android malware samples is mainly provided via the Android Malware Genome Project [133], Contagio Mobile[84], and VirusShare[85]. The Android Malware Genome Project contains over 1,200 Android malware samples from 49 families. Contagio Mobile offers an upload dropbox to share mobile malware samples among security researchers and currently hosts 164 archives, where some archives contain more than 16,000 samples. VirusShare also hosts a repository of malware samples with over 11,000 Android samples available to researchers. Furthermore, the multi-engine anti-virus scanning services VirusTotal[86] and AndroTotal [134] provide researchers with access to submitted samples.

[79] http://www.androidauthority.com/ggtracker-malware-hides-as-android-market-17281/
[80] http://developer.android.com/about/dashboards/index.html
[81] http://thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage/
[82] http://thesnkchrmr.wordpress.com/2011/03/27/udev-exploit-exploid
[83] http://github.com/revolutionary/zergRush
[84] http://contagiominidump.blogspot.com
[85] http://www.virusshare.com
[86] http://www.virustotal.com

6.2 Android Malware Analysis Frameworks

Ever since the first Android phones were released in 2008, researchers have proposed dozens of frameworks for a variety of purposes. In this section, we outline our efforts in systematizing these proposals by enumerating and analyzing this existing knowledge.

Analysis frameworks may use a number of techniques to produce a report about an app's functionality or perform a classification whether an app is benign or malicious. *Static analysis* techniques extract features from the Android application package (APK) and the Dalvik bytecode. *Dynamic analysis* techniques monitor an app's behavior during runtime in a controlled environment. Results from static analysis can also enhance dynamic analysis, e.g., to efficiently stimulate a targeted application and trigger additional behavior, resulting in *hybrid analysis* approaches.

In the following paragraphs we briefly explain different analysis methods for both static and dynamic analysis and present available tools and frameworks for both approaches. Our evaluation mainly focuses on dynamic analysis frameworks, however, we also present various static analysis tools that might assist dynamic analysis and therefore be integrated in the presented dynamic analysis frameworks. For each dynamic analysis framework, we distinguish a number of characteristics and provide a brief summary. Based on their main purpose and approach, we classify the different research efforts into distinct categories.

6.2.1 Static Analysis Tools

Static analysis tools fall in one of the following categories:

- **Extraction of meta information:** Tools that extract information from an application's manifest and provide information about requested permissions, activities, services and registered broadcast receivers. Meta information is often used during later dynamic analysis in order to trigger an application's functionality.
- **Weaving:** Tools that rewrite bytecode of existing applications using a bytecode weaving technique. Using this technique, analysis frameworks can, for instance, insert tracing functionality into an existing application.
- **Decompiler:** Tools that implement a Dalvik bytecode decompiler or disassembler.

One of the most popular comprehensive static analysis tool for Android applications is *Androguard* [135]. It can disassemble and decompile Dalvik bytecode back to Java source code. Given two APK files, it can also compute a similarity value to detect repackaged apps or known malware. It also has modules that can parse and retrieve information

from the app's manifest. Due to its flexibility, it is used by some other (dynamic) analysis frameworks that need to perform some form of static analysis. *APKinspector*[87] is a static analysis platform for Android application analysts and reverse engineers to visualize compiled Android packages and their corresponding DEX code. *Dexter*[88] is a web application designed for static analysis of Android applications. Its features are comparable with those of Androguard and APKinspector, however, it has some additional collaboration functionality to ease knowledge sharing among multiple researchers.

APKtool[89] is a tool for reverse engineering Android applications. It can decode resources to nearly original form and rebuild them into a new Android package after they have been modified. APKtool can be used to add additional features or extra support to existing applications without contacting the original author and thus assist bytecode weaving approaches. *Joe Sandbox Mobile* uses static instrumentation, which is equivalent to bytecode weaving in our classification. *Radare2*[90] is an open source reverse engineering framework which provides a set of tools to disassemble, debug, analyze, and manipulate binary Android files. Other disassembler tools for DEX files include *Dedexer*[91] and *smali/baksmali*[92]. Both read DEX files and convert them into an "assembly-like" format which is largely influenced by the *Jasmin* syntax[93].

Other tools aim at enabling static analysis on Android applications by retargeting them to traditional `.class` files, which then can be processed by existing Java tools. One example is *ded* [136], [137], which later evolved into *Dare*[94]. Similarly, *dex2jar*[95] can convert an APK file directly to a `.jar` file and vice versa. *JEB*[96] is a commercial flexible interactive Android decompiler. It claims to be able to directly decompile Dalvik bytecode to Java source code, as well as disassemble an APK's contents so that users can view the decompressed manifest, resources, certificates, etc.

6.2.2 Dynamic/Hybrid Analysis Frameworks

Dynamic analysis frameworks monitor the behavior of unknown applications at runtime by executing the targeted application in a controlled environment to generate a behavioral footprint. Dynamic analysis can monitor an app's behavior utilizing one (or more) of the following techniques:

[87] `https://github.com/honeynet/apkinspector/`
[88] `http://dexter.dexlabs.org/`
[89] `https://code.google.com/p/android-apktool/`
[90] `http://radare.org/y/`
[91] `http://dedexer.sourceforge.net`
[92] `https://code.google.com/p/smali/`
[93] Jasmin is an assembler for the Java VM: `http://jasmin.sourceforge.net`
[94] `http://siis.cse.psu.edu/dare/`
[95] `http://code.google.com/p/dex2jar/`
[96] `http://www.android-decompiler.com/`

- **Taint tracking:** Taint tracking tools are often used in dynamic analysis frameworks to implement system-wide dynamic taint propagation in order to detect potential misuse of users' private information.
- **Virtual machine introspection (VMI):** VMI-based frameworks [138] intercept events that occur within the emulated environment. *Dalvik VMI* based systems monitor the execution of Android APIs through modifications in the Dalvik VM. *Qemu VMI* based systems are implemented on the emulator level to enable the analysis of native code. However, emulators are prone to emulator evasion [139].
- **System call monitoring:** Frameworks can collect an overview of executed system calls, by using, for instance, VMI, `strace` or a kernel module. This enables (partial) tracing of native code.
- **Method tracing:** Frameworks can trace Java method invocations of an app in the Dalvik VM.

The first dynamic Android analysis framework developed was AASandbox [123]. It implements a system call monitoring approach using a loadable kernel module. Furthermore, it uses the resulting system call footprint to discover possibly malicious applications.

A popular taint tracking framework is TaintDroid [64]. TaintDroid is implemented on top of the Dalvik VM and monitors applications for the leakage of sensitive information. However, ScrubDroid [140] presented a number of attacks to circumvent dynamic taint analysis. VetDroid [141] is a dynamic framework that measures actual permission use behavior by dynamically building a permission use graph. Their data tainting method is built upon TaintDroid, but improves it by identifying implicit and explicit permission use points.

DroidBox [142] uses TaintDroid to detect privacy leaks, but also comes with a Dalvik VM patch to monitor the Android API and report file system and network activity, the use of cryptographic operations and cell phone usage such as sending SMS and making phone calls. However, it can be easily bypassed if applications include their own libraries, in particular by using it for cryptographic operations. In the latest version, DroidBox utilizes bytecode weaving to insert monitoring code into the app under analysis[97]. ANANAS [143] is a recently published framework focusing on a modularized architecture with analysis modules for logging e.g. file system activity, network activity or system calls. It also employs API monitoring similar to the most recent version of DroidBox and uses a loadable kernel module for system call monitoring.

DroidScope [144] uses VMI to reconstruct Dalvik and native code instruction traces. The authors also implemented their own data tainting method, named TaintTracker. One of

[97] `http://code.google.com/p/droidbox/wiki/APIMonitor`

the huge benefits is that it does not require any changes to the Android sources, however JIT has to be selectively disabled as it blurs the Dalvik instruction boundaries. Andrubis [145] leverages various techniques, is VMI-based and monitors events in the Dalvik VM as well as native code for system-calls through QEMU VMI. It further employs TaintDroid for data tainting. TraceDroid [146] has the benefit of generating a complete method trace output and was also integrated into Andrubis. It further captures network traffic and refrains from analyzing native code, as `ltrace` and `strace` are deemed sufficient for this purpose. This is also the approach used by Mobile Sandbox[98], which runs the app in the emulator and uses `ltrace` to track native code. Finally, CopperDroid [147] uses a VMI-based approach as well, but compared to the other frameworks it is also capable of analyzing IPC and RPC-based communication between applications.

Andrubis, TraceDroid, Mobile Sandbox and CopperDroid all perform input stimulation, most notably the monkey exerciser and the emulation of common events like GPS lock, SMS received or boot-completed. To further enhance code coverage, several frameworks for automatically extracting and exercising UI based triggers have been proposed. The SmartDroid framework [148] statically extracts the function call graph and activity call graph, and then dynamically traverses these graphs to find elements that trigger sensitive behavior. In contrast to other frameworks, however, it does not focus on possibly malicious activities like sending SMS or accessing files. AppsPlayground [149] is a framework that automates analysis of Android applications and monitors taint propagation (using TaintDroid), specific API calls and system calls. Its main contribution is a heuristic-based intelligent black-box execution approach to explore the apps GUI. The goal of AppIntent [150] is to distinguish user-intended data transmission from unintended ones, and is as such related to SmartDroid. It uses an event-space constraint guided symbolic execution technique to construct event sequences, which effectively reduces the event search space in symbolic execution for Android apps.

A number of additional dynamic analysis platforms have been implemented and made available to the public via web applications: SandDroid[99], VisualThreat[100], ForeSafe[101] and the Joe Sandbox Mobile APK Analyzer[102]. These frameworks, however, come often with very little public documentation on how they function, which makes it hard to make statements on any new approaches used by these implementations. It is likely that most of these platforms use (modified versions of) existing tools like DroidBox, TaintDroid and Androguard to complement their dynamic analysis engine. This is confirmed for example on SandDroid's webpage, which states that it is powered by both DroidBox and Androguard. While we could not find a public reference which Android version

[98] `http://mobilesandbox.org/`
[99] `http://sanddroid.xjtu.edu.cn`
[100] `http://www.visualthreat.com`
[101] `http://www.foresafe.com`
[102] `http://www.apk-analyzer.net`

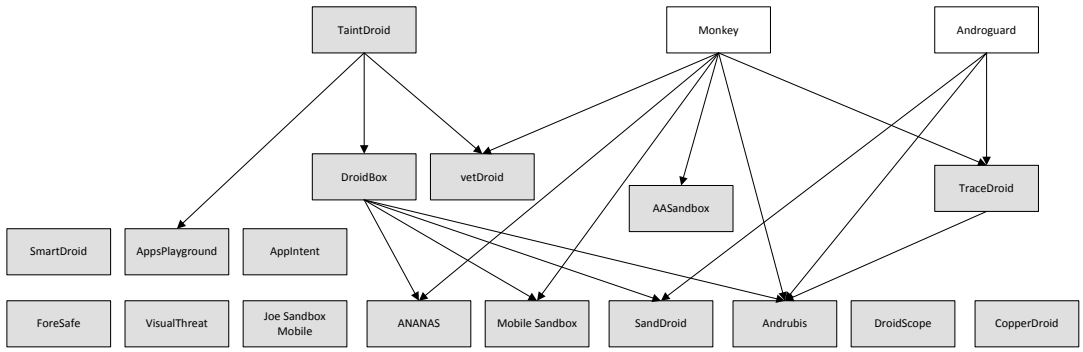


Figure 6.1: Interdependency graph illustrating sandboxes relying on other tools or services.

is used by Joe Sandbox Mobile, it was possible to create APK files with varying API level requirements and fingerprint the Android version to API level 15. This either is Android version 4.0.3 or 4.0.4, which is an upper bound of supported Android versions. This fingerprinting technique was unsuccessful with ForeSafe and VisualThreat.

6.2.3 Other Frameworks

For the sake of completeness it is also worth mentioning various other frameworks that offer online services for the analysis of Android apps based on their static features or meta information. Badger[103] provides a web service for extracting required permissions as well as included advertising applications from uploaded APK files. Undroid[104] performs slightly more comprehensive static analysis and extracts requested permissions as well as API calls that use these permissions. The Android Observatory [151] also extracts required permissions from the manifest and further tries to match the app's certificate against those of known malware.

VirusTotal[105] and AndroTotal [134] are two multi-engine anti-virus scanning services. VirusTotal uses static AV-based scanner engines to assess the maliciousness of a submitted APK file. AndroTotal uses dynamic sandboxing to test samples against mobile malware detectors, similar to VirusTotal. However, it uses the apps themselves on physical devices as well as in the emulator instead of command line versions with the same underlying signature database. It also provides links to analysis results from other frameworks: VirusTotal, CopperDroid, ForSafe, SandDroid, and Andrubis.

[103] http://davidson-www.cs.wisc.edu/baa
[104] http://www.av-comparatives.org/avc-analyzer/
[105] http://www.virustotal.com

Table 6.1: Overview of Android analysis frameworks and their availability - either as source code (src) or through a public web interface (www).

Framework	src	www	Framework	src	www
AASandbox [123]			ForeSafe		•
AppIntent [150]			Joe Sandbox Mobile		•
ANANAS [143]			Mobile Sandbox [106]		•
AndroTotal [134]		•	SandDroid		•
Andrubis [145]		•	SmartDroid [148]		
AppsPlayground [149]	•		TaintDroid [64]	•	
CopperDroid [147]		•	TraceDroid [146]		•
DroidBox [142]	•		vetDroid [141]		
DroidScope [144]	•		VisualThreat		•

6.3 Evaluation Criteria

We present an overview of all dynamic analysis frameworks discussed so far in Table 6.1 and list, whether the source code is publicly available (src) or the service is available through a web interface (www).

6.3.1 Features of Interest

In order to compare the discussed dynamic analysis frameworks, we manually examined them and compiled a feature list for further evaluation. First of all, we are interested in implementation details such as the *Android version* that is supported by the sandbox. Android is under active development. At the time of writing Android 4.4 (KitKat) is the most recent version. We are also interested in the *inspection level* and which methods the sandboxes use to capture the dynamic behavior of an app. As discussed in Section 6.2.2, multiple methods are suitable: Modifying stock Android for additional logging capabilities is one convenient way to capture app behavior, in particular the kernel or the APIs. ANANAS, for example, uses a loadable kernel module for its analysis. The downside of this approach is that the kernel patches cannot be easily ported to other versions of Android, as the internals can change between updates. Secondly, as Android programs are executed within the Dalvik VM, some approaches modify the Dalvik VM to capture the dynamic behavior of an app. TaintDroid and DroidBox use this approach. The downside of this approach is that only Java calls are traceable and native code is not captured. Lastly, it is possible to modify QEMU for enhanced logging mechanisms and to use true virtual machine introspection. This is a holistic method, as it works on a lower level compared to the other methods. However, `arm-to-x86` emulation is known to be slow, and as such it has a high overhead. Some sandboxes even use combinations of various methods e.g., Andrubis uses virtual machine introspection through QEMU in

combination with a modified Dalvik VM.

Additionally, some sandboxes use *taint tracking* to detect if private information is leaked. Furthermore, sandboxes usually perform some form of *static analysis* such as parsing the requested permissions from the manifest. *GUI interactions* are usually employed to simulate user activity and enhance the coverage of dynamic analysis. We can further distinguish the sandboxes by the features they analyze. *Network activity* as well as *file activity* is of particular interest in malware analysis, as malware usually reads and writes local files and relies on obtaining commands from a command and control server in case of a botnet [152]. As discussed in Section 6.1.1, malware is often used to generate revenue for the malware creator by sending premium SMS or calling premium phone numbers causing financial damage to the victim. As such, we interesting if the sandbox is able to capture *phone activity*. Furthermore, we evaluate if the analysis frameworks are able to analyze *native code*.

6.3.2 Framework Interdependency

Many sandboxes build on previous work and incorporate existing tools in their service. In particular a combination of the open source solutions TaintDroid and DroidBox for dynamic analysis and Androguard for static analysis are heavily used. As we will show in Section 6.4, this can be a problem if the malware is capable of evading these underlying analysis tools. Furthermore, many sandboxes rely on the Application Exerciser Monkey[107] (or Monkey) to simulate user input, even though the intended use case of Monkey was to stress test applications with pseudorandom user events. In Figure 6.1 we provide the full interdependency graph depicting which sandboxes rely on which other tools or services.

6.3.3 Sandbox Fingerprinting

Dynamic sandboxes are often rather easy to fingerprint, which is a problem that has been extensively studied for desktop malware [153]–[155]. First of all, system emulators like QEMU [139], [156] and virtual machines [157] are detectable. With the numerous sensors and interfaces of a smartphone, a real device is even more challenging to simulate than PC hardware. GPS, screen size, and the motion sensor are just a few examples of what has to be considered when building sandboxes for mobile devices. Malware can also simply wait for a certain amount of time, as dynamic sandboxes cannot execute an app indefinitely [158] due to time constraints. Other methods to detect if the app is running within a sandbox include checking the application's signature. This makes it possible to detect platforms like the recent version of DroidBox that modify the app's bytecode.

[107] `http://developer.android.com/tools/help/monkey.html`

Fingerprinting methods can also rely on implementation details of specific sandboxes. For example, AASandbox uses 500 monkey events to simulate user input, with 1 second between events [123]. While this is of course just a configuration parameter, malware could easily detect AASandbox if these parameters are not modified. TraceDroid on the other hand initializes the Monkey with the same seed value for the pseudo-random number generator [146]. This is intended to ensure that re-runs of the exerciser generate the same sequence of events for debug purposes. This is, however, also detectable by malware. SmartDroid [148] waits for 10 seconds before starting UI interactions during dynamic analysis.

6.4 Evaluation Results

Table 6.2: Comparison of Android malware analysis sandboxes.

	Implementation Details		Analysis Type			Analyzed Features			
Framework	**Android Version**	**Inspection Level**	**Static**	**Tainting**	**GUI Interactions**	**File**	**Network**	**Phone**	**Native Code**
AASandbox	—	Kernel	•		•	•	•	•	
AppIntent	2.3	Kernel	•	•	•				
ANANAS	2.3-4.2	Kernel	•		•	•	•	•	•
Andrubis	2.3.4	QEMU & Dalvik	•	•	•	•	•	•	•
AppsPlayground	—	Kernel	•	•	•				
CopperDroid	2.2.3	QEMU	•		•	•	•	•	•
DroidBox	2.3-4.1	Kernel		•		•	•	•	
DroidScope	2.3	Kernel & Dalvik		•		•	•	•	•
ForeSafe	?	?	•		•	•	•		
Joe Sandbox Mobile	4.0.3 / 4.0.4	Static Instrumentation	•		•	•	•	•	
Mobile Sandbox	2.3.4	Dalvik	•	•	•		•	•	•
SandDroid	?	?	•	•	?	•	•	?	?
SmartDroid	2.3.3	Kernel	•	•	•	•	•	•	
TraceDroid	2.3.4	Dalvik	•		•	•	•	•	
vetDroid	2.3	Kernel & Dalvik	•	•	•	•	•	•	
VisualThreat	?	?	•			•	•	•	•

We evaluated a total of 10 dynamic sandboxes based on the feature set outlined above, skipping sandboxes that are not available to us in any form. Our analysis is based on the academic publications as well as available documentation. Table 6.2 lists the extracted features for all sandboxes. We furthermore tested the sandboxes with selected malware samples, which we discuss in the following paragraphs.

6.4.1 Malware Samples

For our evaluation we used real-world malware samples from four different families or categories. For each of those families we used two different samples. All of the samples have been publicly analyzed and described, either as part of the Android Malware Genome Project [133] or in blogposts or technical reports from antivirus vendors. All four families were chosen due to either feature coverage (regarding [133] coverage in terms of financial charges and personal information stealing) or due to interesting behaviour like sophisticated privilege escalation techniques or evasion of specific sandboxes.

[108] http://www.securelist.com/en/blog/8106/The_most_sophisticated_Android_Trojan

Obad In 2013, Kaspersky Lab[108] described Obad as one of the most sophisticated mobile malware to date. The user has to install the malware manually with 24 different permissions, which grant nearly total access to the device. Obad is able to send SMS, send data to an URL, download and install files, and transfer files via Bluetooth. Additionally, the application tries to connect to a command and control server and is able to join a botnet. Malware of the Obad family tries to evade detection from several sandboxes with anti-emulation and anti-decompilation techniques, like checking for the *Android.os.build.MODEL* value to quit the applications execution if the default value of the emulator is present [159], [160].

Geinimi The second Android malware family we selected for our analysis is Geinimi. Geinimi includes a wealth of malicious payloads: remote control over the Internet, starting phone calls, sending SMS as well as leaking sensitive data stored on the phone.

DroidKungFu DroidKungFu[109] is part of the Android Malware Genome Project as well, and is a malware family that can perform various forms of privilege escalation. It collects various phone related data, such as the IMEI number and the phone model, and sends it to a server. This malware family is of particular interest since it is able to execute various exploits, for example privilege escalation using the RageAgainstTheCage exploit[110]. This exploit decrypts itself during runtime to exploit the `adb` resource exhaustion bug for root access. After this procedure the application is able gain root privileges and hide itself, drop other malware, or perform other malicious activities.

Basebridge/Nyleaker The fourth malware category we used is the malware samples Basebridge and Nyleaker. Both samples are able to successfully evade analysis by abusing the Androguard tool (which is used by many sandboxes) by presenting itself as an invalid application by either using a corrupt APK or an invalid manifest file within the APK. Their malicious behavior includes sending SMS to premium services, executing local privilege escalation exploits and leaking personal information.

Master Key The last evaluated category is the Master Key set that exploits several Android vulnerabilities related to the handling of ZIP files[111].

- *Bug 8219321*[112]: The ZIP format allows multiple occurrences of the same filename within one archive. This can lead to a serious vulnerability, if the implementation of the unpacking of the archive differs in parts of the systems: The Android

[109] `http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html`
[110] `http://thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage/`
[111] APK files are ZIP compressed files, based on the JAR format.
[112] `http://nakedsecurity.sophos.com/2013/07/10/anatomy-of-a-security-hole-`

signature verifier checks the first file for integrity and compares it with the META-INF directory. The Android installer however uses the second file for installation. Symantec reported that this bug was already exploited by malware in the wild[113]. Specifically malware samples added an additional malicious classes.dex as well as a second manifest file to a benign app without breaking its signature.

- *Bug 9695860*[114]: This bug is a simple signed-unsigned integer mismatch between different parts of the Android code. The ZIP file header contains the fields for filename length and extra field length. The signature verification code treats the contents of these fields as signed 16-bit integers, which converts the input of "0xFFFD" to "-3". Thus, the Android verifier jumps three bytes backwards, instead of forward and therefore skips the extra field by reading the part "dex" of "classes.dex". The application loading code however treats the fields as unsigned 16-bit integers, which could match the start of a second uncompressed dex file supplied in the file data section.

- *Bug 9950697*[115]: This last version of the Master Key vulnerability is caused by the redundant storage of the "filename length" field in the ZIP header. This field indicates how many bytes between the filename and the actual file data exist and also how many bytes in the central directory exist, to reach the next directory entry in the archive. It is possible to provide a real filename length for the verifier, that verifies the trustworthy original file data and fake a filename length in the local header for the loader, that later executes the malware code.

Additionally, issue 14315 in Python[116] can be used to create APK files which are not processed correctly by certain sandboxes. This Python bug triggers an exception when the length field in the ZIP header is zero. While this is a Python issue and not related directly to Android, many tools and sandboxes which are based on Python are affected and malware authors could craft APK files that trigger this bug to evade analysis.

6.4.2 Analysis Results

We submitted the 12 malware samples (8 from known corpora, and 4 exploiting one Master Key vulnerability each) to all sandboxes which were available at the time of writing and which use some form of dynamic analysis within a sandbox. Sandboxes

googles-android-master-key-debacle-explained

[113] http://www.symantec.com/connect/blogs/first-malicious-use-master-key-android-vulnerability-discovered

[114] http://nakedsecurity.sophos.com/2013/07/17/anatomy-of-another-android-hole-chinese-researchers-claim-new-code-verification-bypass/

[115] http://nakedsecurity.sophos.com/2013/11/06/anatomy-of-a-file-format-problem-yet-another-code-verification-bypass-in-android/

[116] http://bugs.python.org/issue14315

which were not available during our evaluation were excluded, leaving ten sandboxes. The results of our evaluation are outlined in Table 6.3.

For every family of malware, we submitted two different samples (separated by a "/" in Table 6.3). Andrubis was able to analyze every submitted sample, while the Nyleaker malware was classified as benign. ForeSafe was able to successfully flag every submitted sample as high risk malware. SandDroid was unable to analyze any submitted sample. As for CopperDroid, just one version of Geinimi and DroidKungFu were detected. For all other samples the analysis was aborted with an "Installation Error". Joe Sandbox Mobile and TraceDroid on the contrary were able to analyze every version of every sample we submitted. Joe Sandbox Mobile furthermore correctly flagged every sample as malicious in the generated reports.

Table 6.3: Analysis results of online sandboxes for two samples per malware family ("•"=detected, "○"=not detected, "-"=analysis error).

Framework	**Obad**	**Geinimi**	**DroidKungFu**	**Basebridge/ Nyleaker**
Andrubis	• / •	• / •	• / •	• / ○
CopperDroid	- / -	• / -	- / •	- / -
ForeSafe	• / •	• / •	• / •	• / •
Joe Sandbox Mobile	• / •	• / •	• / •	• / •
Mobile Sandbox	- / -	- / -	- / -	- / -
SandDroid	- / -	- / -	- / -	- / -
TraceDroid	• / •	• / •	• / •	• / •
VisualThreat	• / -	• / •	• / •	• / •

We also compiled an APK for each of the described Master Key bugs and submitted them to our selection of sandboxes to see if they could analyze malicious APKs exploiting those vulnerabilities. We present the results in Table 6.4.

6.4.3 Limitations of Existing Sandboxes

We observed that many sandboxes were not able to fully analyze our samples, were prone to bugs or evasion techniques, or were either no longer maintained or not publicly available anymore at the time of writing. As some versions of the Obad family evade DroidBox, Mobile Sandbox was not able to analyze these. We verified this in a personal communication with the author of Mobile Sandbox. The other samples, namely Geinimi, DroidKungFu, Basebridge and Nyleaker, are also marked as errors in Table 6.3 as there are currently over 300,000 samples in the queue to be analyzed by the dynamic sandbox. Any submitted sample would thus be analyzed after approximately 400 days, which is too long for any useful analysis. This number does not seem to decrease. SandDroid

Table 6.4: Evaluation of Sandboxes with Master Key Samples: "•" indicates that the sample was successfully executed, "-" indicates, that the sandbox was not able to execute the sample.

Framework	**Bug 8219321**	**Bug 9695860**	**Bug 9950697**	**Python ZIP Bug**
Andrubis	•	-	-	•
CopperDroid	-	-	-	-
ForeSafe	•	•	•	•
TraceDroid	•	-	-	•
VisualThreat	•	•	-	•

accepts samples for analysis but is not able to perform any analysis on the samples. After submitting the samples and waiting for about two weeks, the samples are either not in the report database or marked as still being analyzed by the framework.

Some frameworks mentioned in Table 6.2 were excluded from our evaluation: AASandbox, AppIntent, ANANAS, SmartDroid, and vetDroid were neither available as source code nor as an online submission. We were not able to run any samples for analysis within these frameworks. Because AppsPlayground has no online submission possibility[117], it was out of scope and therefore also excluded from our evaluation.

6.4.4 Discussion

Every sandbox carries out a multitude of analysis techniques, ranging from basic static to rigorous dynamic analysis techniques. Due to the large number of available analysis frameworks, on- and offline, it is impossible for a user to determine which framework offers the most and comprehensive set of features.

Supporting these facts, there is no Swiss-army-knife-sandbox that on one hand offers every possible feature and on the other hand is readily available. Nevertheless, also on mobile devices the cat-and-mouse-game between malware authors and security researchers is continuously ongoing, as it has been on desktop computers for many years. One possibility to deal with that problem could be one non-commercially-driven analysis framework that implements all static and dynamic analysis techniques discussed in the literature so far, and is fully maintained and continuously extended by research volunteers. Until this goal is reached, malware authors will be ahead of defending researchers and industry practitioners.

[117] http://list.cs.northwestern.edu/mobile/

CHAPTER 7

Conclusions and Future Directions of Work

7.1 Conclusions

Privacy and Security in mobile environments are a sensitive topic, especially since so many different applications nowadays rely on mobile services. Either as host for all means of different applications, or as a communication hub for additional devices, like smart watches or other Internet of Things appliances. In this thesis, we provide several improvements to enhance security of applications and thereby provide better protection for users.

We conducted a large scale analysis on the widespread practice of online tracking by third-party services. Third-party tracking has serious implications for the privacy and security of users. We provided insights into the limitations and short-commings of current tracker-blocking tools. Our results are based upon the analysis of over 100,000 websites in combination with the state-of-the-art tracker blocking tools. Our findings suggest that some browser extensions can effectively block the majority of stateful third-party trackers. However, they still have blind spots regarding blocking stateless fingerprinting scripts and smaller third-party trackers. We showed that over 60% of third-party tracking services communicate without the use of appropriate preotection mechanisms, like TLS, to protect the transmitted data. Overall, our study advances the field of web privacy by providing the largest study on the effectiveness of tracker-blocking tools on websites to date. We furthermore showed the most pressing challenges for mitigating online tracking.

We conducted a longitudinal study on the availability and functionality of different TLS notary services. We performed daily scans over a period of one year and analyzed the

collected data. We explored the ecosystem of TLS notary services and analyzed their behavior on a large scale. We developed a new proxy-based system to transparently query different notary services for increased protection against Man-in-the-Middle (MitM) attacks and provided an overview on the inner workings of these notary services. Lastly, we discussed the the available of TLS notary services. We identified the problems and pitfalls that can arise by using the existing systems and proposed extensions to improve existing systems to enhance security and privacy for users.

While many applications already employ certificate pinning, they still need to update the application, each time the certificate changes. Based on the results of our study on TLS notary services, we showed that notary-assisted certificate pinning can offer a layer of defense in the meantime that an application is exposed due to a certificate change and the inherent delay to update through the app store procedure. We proposed a design and system architecture for notary-assisted certificate pinning in Google Android devices. We realized a proof-of-concept Android application to evaluate its usefulness Currently administrative privileges are needed to be able to intercept and replace TLS certificate validation in installed applications.

The PoC implementation confirmed that there is no noticeable performance penalty in those cases that a notary service must be consulted; yet the security improvement is very significant. Overall, our proposal for a notary-assisted TLS certificate pinning increases *both* the security and the usability of mobile devices, while reducing the burden of the users being involved in system security and trust decisions.

We conducted a comprehensive study on detection methods of Android malware. First, we provided an overview on current Android malware distribution techniques and the motivation of malware authors. Secondly, we analyzed available dynamic analysis platforms for Android and examined their interdependencies. An evaluation of ten sandboxes which are available as online services, using samples from four real-life Android malware families, showed that detection rates vary. In addition, we found that popular dynamic sandboxes are susceptible to well-known vulnerabilities like the Master Key vulnerabilities, which could be potentially misused by Android malware to thwart analysis. Our findings show that while current dynamic sandboxes are valuable for academic research to analyze Android malware, they cannot be considered an effective defense against malware on their own. Interdependencies between platforms caused by code reuse can lead to challenges in detecting malware that targets specific platform limitations, as all analysis platforms which share the same code base are affected.

7.2 Future Directions of Work

To improve our implementation of TLS notary assisted certificate pinning for Android applications a larger-scale validation is deemed necessary, possibly using the available

dataset of [34], in order to study scalability issues. A production-ready implementation is envisioned for the future. For this implementation, a coordination with the notary operators is necessary, so as to implement a secure communication interface. Towards this direction, it would be helpful to see system vendors integrate such functionality in the operating system itself and thus, make the solution available for all Android devices and not only rooted ones.

While our studies showed that the distribution of notary services seems to be diminishing, we will investigate various application domains, where the use of notaries could prove useful. Such domains include IoT or smart devices like DVRs or TVs, where application updates might not happen as often as server side certificate changes. The use of notary services could help to improve security of communication channels. Existing TLS notary services face the challenge that the initial request for an unknown page can introduce extra latency, since the notary has to query the server for the certificate. Future research could evaluate the possibilities to use data sources provided by internet wide scans (e.g., `scans.io` [161] and `Censys.io` [31]) either as alternative initial data providers to bootstrap notaries or to wrap the data into a separate notary service.

With our implementation of notary assisted certificate pinning, we have already shown that security of applications can be improved by employing privileged applications directly on the device. Future efforts could continue this line of work of using privileged applications by, e.g., improving the firewalling of Android application intents to better protect inter-application communication.

The main future challenge is in the analysis of Android applications. On the one hand, we need to find security flaws to make applications more secure. To fix these flaws in applications, it is necessary to get a clean representation of the applications, before they can be fixed and recompiled. On the other hand, identification of malicious applications is vital to protect users from installing malware on their devices. While many tools already exist to analyse Android applications, analysis of obfuscated code is still a hard challenge. Especially on native-code level obfuscations pose hard problems to analysis tools, since they also allow for dynamic modification of code during run-time. While automated approaches for de-obfuscation of specific obfuscation methods already exist, we are still missing methods that can efficiently handle different, previously unknown, obfuscation methods.

We argue that generalized automated de-obfuscation methods need to be developed, to provide better methods for applications analysis. This needs deeper research in possibilities for obfuscating applications and also on how to deal with de-obfuscation of previously unknown obfuscation schemes. We therefore consider this an interesting extension of methods for code analysis of applications. These methods will on the one hand help to analyse malware, but also to analyse software concerning security flaws.

List of Figures

List of Tables

Bibliography

[1] G. Greenwald, "XKeyscore: NSA tool collects' nearly everything a user does on the internet'", *The Guardian*, vol. 31, 2013.

[2] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web", in *NSDI*, USENIX Association, 2012.

[3] B. Krishnamurthy and C. E. Wills, "Generating a privacy footprint on the internet", in *ACM Internet Measurement Conference*, ACM, 2006, pp. 65–70.

[4] C. J. Hoofnagle, A. Soltani, N. Good, D. J. Wambach, and M. Ayenson, "Behavioral Advertising: The Offer You Cannot Refuse", *Harvard Law & Policy Review*, 2012.

[5] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting", in *Security and privacy (SP), 2013 IEEE symposium on*, IEEE, 2013, pp. 541–555.

[6] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The Web never forgets: Persistent tracking mechanisms in the wild", *ACM CCS'14*, 2014.

[7] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "FPDetective: Dusting the web for fingerprinters", in *ACM CCS'13*, ACM, 2013, pp. 1129–1140.

[8] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis Draft: July 11th, 2016", [Technical Report], Jul. 2016, [Online]. Available: `http://randomwalker.info/publications/OpenWPM_1_million_site_tracking_measurement.pdf`.

[9] N. Leavitt, "Internet security under attack: The undermining of digital certificates", *Computer*, vol. 44, no. 12, pp. 17–20, 2011.

[10] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security", in *Proceedings of the 2012 ACM conference on Computer and Communications security (CCS '12)*, ACM, 2012, pp. 50–61.

[11] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl, "To Pin or Not to Pin - Helping App Developers Bullet Proof Their TLS Connections", in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 239–254.

[12] IDC, *Smartphone OS Market Share, 2016 Q3*, `https://www.idc.com/promo/smartphone-market-share/os`, Dec. 2016.

[13] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", Internet Engineering Task Force, Internet-Draft, 2014, Work in Progress.

[14] C. Evans, C. Palmer, and R. Sleevi, *Public key pinning extension for HTTP (HPKP)*, RFC 7469 (Draft), Internet Engineering Task Force, 2015.

[15] C. Soghoian and S. Stamm, "Certified lies: Detecting and defeating government interception attacks against ssl (short paper)", in *Financial Cryptography and Data Security*, Springer, 2012, pp. 250–259.

[16] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl, "Spoiled onions: Exposing malicious Tor exit relays", in *Privacy Enhancing Technologies*, Springer, 2014, pp. 304–331.

[17] P. Syverson and G. Boyce, "Genuine onion: Simple, Fast, Flexible, and Cheap Website Authentication", in *Proceedings of the 9th Workshop on Web 2.0 Security and Privacy (W2SP) 2015*, 2015.

[18] R. Holz, T. Riedmaier, N. Kammenhuber, and G. Carle, "X.509 forensics: Detecting and localising the SSL/TLS men-in-the-middle", in *Computer Security–ESORICS 2012*, Springer, 2012, pp. 217–234.

[19] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving SSH-style Host Authentication with Multi-path Probing", in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ser. ATC'08, Boston, Massachusetts: USENIX Association, 2008, pp. 321–334.

[20] M. Marlinspike, "SSL and the future of authenticity", *Black Hat USA*, 2011.

[21] A. Bates, J. Pletcher, T. Nichols, B. Hollembaek, and K. R. Butler, "Forced Perspectives: Evaluating an SSL Trust Enhancement at Scale", in *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC 2014)*, ACM, 2014, pp. 503–510.

[22] K.-P. Fuchs, D. Herrmann, A. Micheloni, and H. Federrath, “Laribus: privacy-preserving detection of fake SSL certificates with a social P2P notary network”, *EURASIP Journal on Information Security*, vol. 2015, no. 1, pp. 1–17, 2015.

[23] G. Merzdovnik, K. Falb, M. Schmiedecker, A. Voyiatzis, and E. Weippl, “Whom You Gonna Trust? A Longitudinal Study on TLS Notary Services”, in *Data and Applications Security and Privacy XXX - 30th Annual IFIP WG 11.3 Conference (DBSec 2016)*, Springer, 2016.

[24] P. Eckersley and J. Burns, “An observatory for the SSLiverse”, *Talk at Defcon*, vol. 18, 2010.

[25] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, “The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements”, in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, ACM, 2011, pp. 427–444.

[26] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, “Analysis of the HTTPS certificate ecosystem”, in *Internet Measurement Conference*, 2013.

[27] W. Mayer, A. Zauner, M. Schmiedecker, and M. Huber, “No Need for Black Chambers: Testing TLS in the E-mail Ecosystem at Large”, *CoRR*, vol. abs/1510.08646, 2015. [Online]. Available: `http://arxiv.org/abs/1510.08646`.

[28] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kâafar, “TLS in the wild: an Internet-wide analysis of TLS-based protocols for electronic communication”, *CoRR*, vol. abs/1511.00341, 2015. [Online]. Available: `http://arxiv.org/abs/1511.00341`.

[29] Z. Durumeric, E. Wustrow, and J. A. Halderman, “ZMap: Fast Internet-wide Scanning and Its Security Applications.”, in *Usenix Security*, vol. 2013, 2013.

[30] R. D. Graham, “MASSCAN: Mass IP port scanner”, *URL: https://github. com/robert-davidgraham/masscan*, 2014.

[31] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, “A search engine backed by Internet-wide scanning”, in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2015, pp. 542–553.

[32] N. Vallina-Rodriguez, J. Amann, C. Kreibich, N. Weaver, and V. Paxson, “A tangled mass: The Android root certificate stores”, in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ACM, 2014, pp. 141–148.

[33] T. Fadai, S. Schrittwieser, P. Kieseberg, and M. Schmiedecker, "Trust Me, I am a Root CA! Analyzing SSL Root CAs in Modern Browsers and Operating Systems", in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security (ARES 2015)*, IEEE Computer Society, 2015, pp. 174–179.

[34] D. Buhov, M. Huber, G. Merzdovnik, and E. Weippl, "Pin It! Improving Android network security at runtime", in *Proceedings of the 15th IFIP Networking Conference, Networking 2016*, IEEE, 2016, pp. 297–305.

[35] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking SSL development in an appified world", in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (ACM CCS)*, ACM, 2013, pp. 49–60.

[36] J. R. Mayer and J. C. Mitchell, "Third-party web tracking: Policy and technology", in *Security and Privacy (SP), 2012 IEEE Symposium on*, IEEE, 2012, pp. 413–427.

[37] J. Mayer, *FourthParty Web Measurement Platform*, `http://fourthparty.info/`, 2015.

[38] R. Balebako, P. Leon, R. Shay, B. Ur, Y. Wang, and L. Cranor, "Measuring the effectiveness of privacy tools for limiting behavioral advertising", in *W2SP*, 2012.

[39] D. Reisman, S. Englehardt, C. Eubank, P. Zimmerman, and A. Narayanan, "Cookies that give you away: Evaluating the surveillance implications of web tracking", in *WWW*, 2014.

[40] P. Eckersley, "How unique is your web browser?", in *Privacy Enhancing Technologies*, Springer, 2010, pp. 1–18.

[41] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi, "Host Fingerprinting and Tracking on the Web: Privacy and Security Implications.", in *NDSS*, 2012.

[42] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5", in *Web 2.0 Workshop on Security and Privacy (W2SP)*, 2012.

[43] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Vienna, Austria: ACM, 2016, pp. 1388–1401, ISBN: 978-1-4503-4139-4. DOI: `10.1145/2976749.2978313`. [Online]. Available: `http://doi.acm.org/10.1145/2976749.2978313`.

[44] S. Guha, B. Cheng, and P. Francis, "Privad: Practical Privacy in Online Advertising.", in *NSDI*, 2011.

[45] X. Pan, Y. Cao, and Y. Chen, “I Do Not Know What You Visited Last Summer: Protecting Users from Third-party Web Tracking with TrackingFree Browser”, in *NDSS*, 2015.

[46] H. J. Christof Torres and S. Mauw, “FP-Block: usable web privacy by controlling browser fingerprinting”, in *ESORICS*, 2015.

[47] N. Nikiforakis, W. Joosen, and B. Livshits, “Privaricator: Deceiving fingerprinters with little white lies”, in *WWW*, International World Wide Web Conferences Steering Committee, 2015, pp. 820–830.

[48] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson, “An Analysis of Chinas Great Cannon”, in *5th USENIX FOCI Workshop*, Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: `http://blogs.usenix.org/conference/foci15/workshop-program/presentation/marczak`.

[49] L. F. Cranor, “Can users control online behavioral advertising effectively?”, *Security & Privacy, IEEE*, vol. 10, no. 2, pp. 93–96, 2012.

[50] P. Leon, B. Ur, R. Shay, Y. Wang, R. Balebako, and L. Cranor, “Why Johnny can’t opt out: a usability evaluation of tools to limit online behavioral advertising”, in *SIGCHI*, ACM, 2012, pp. 589–598.

[51] J. Bau, J. Mayer, H. Paskov, and J. C. Mitchell, “A Promising Direction for Web Tracking Countermeasures”, *W2SP*, 2013.

[52] S. Bhagavatula, C. Dunn, C. Kanich, M. Gupta, and B. Ziebart, “Leveraging Machine Learning to Improve Unwanted Resource Filtering”, in *AISec*, ACM, 2014, pp. 95–102.

[53] D. Gugelmann, M. Happe, B. Ager, and V. Lenders, “An Automated Approach for Complementing Ad Blockers Blacklists”, *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 282–298, 2015.

[54] M. Keller, “Geek 101: What is jailbreaking?”, *Geek Tech*, 2012.

[55] X. Jiang and Y. Zhou, “A Survey of Android Malware”, in *Android Malware*, Springer, 2013, pp. 3–20.

[56] L. Xing, X. Pan, R. Wang, K. Yuan, and X. Wang, “Upgrading your android, elevating my malware: Privilege escalation through mobile os updating”, in *Security and Privacy (SP), 2014 IEEE Symposium on*, IEEE, 2014, pp. 393–408.

[57] A. Mylonas, "Security and Privacy in Ubiquitous Computing: The Smartphone Model and Paradigm", Athens University of Economics & Business, Dept. of Informatics, Tech. Rep., 2013.

[58] H. Li, H. Zhu, S. Du, X. Liang, and X. Shen, "Privacy leakage of location sharing in mobile social networks: Attacks and defense", *IEEE Transactions on Dependable and Secure Computing*, 2016.

[59] S. Zhao, X. Luo, B. Bai, X. Ma, W. Zou, X. Qiu, and M. H. Au, "I Know Where You All Are! Exploiting Mobile Social Apps for Large-Scale Location Privacy Probing", in *Australasian Conference on Information Security and Privacy*, Springer, 2016, pp. 3–19.

[60] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting Data-Usage Statistics for Website Fingerprinting Attacks on Android", in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '16, Darmstadt, Germany: ACM, 2016, pp. 49–60, ISBN: 978-1-4503-4270-4. DOI: 10.1145/2939918.2939922. [Online]. Available: http://doi.acm.org/10.1145/2939918.2939922.

[61] H. Fu, Y. Yang, N. Shingte, J. Lindqvist, and M. Gruteser, "A field study of run-time location access disclosures on android smartphones", *Proc. USEC*, vol. 14, 2014.

[62] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal, "Your location has been shared 5,398 times!: A field study on mobile app privacy nudging", in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, ACM, 2015, pp. 787–796.

[63] M. Van Kleek, I. Liccardi, R. Binns, J. Zhao, D. J. Weitzner, and N. Shadbolt, "Better the Devil You Know: Exposing the Data Sharing Practices of Smartphone Apps", *forthcoming in CHI-2017*, 2017.

[64] W. Enck, P. Gilbert, B.-G. Chunn, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones", in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[65] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)", in *Security and privacy (SP), 2010 IEEE symposium on*, IEEE, 2010, pp. 317–331.

[66] K. Lu, Z. Li, V. P. Kemerlis, Z. Wu, L. Lu, C. Zheng, Z. Qian, W. Lee, and G. Jiang, "Checking More and Alerting Less: Detecting Privacy Leakages via Enhanced Data-flow Analysis and Peer Voting.", in *NDSS*, 2015.

[67] M. Shin and J. Kim, "Privacy Preserving Watchdog System in Android Systems", in *Platform Technology and Service (PlatCon), 2017 International Conference on*, IEEE, 2017, pp. 1–5.

[68] R. Herbster, S. DellaTorre, P. Druschel, and B. Bhattacharjee, "Privacy capsules: Preventing information leaks by mobile apps", in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ACM, 2016, pp. 399–411.

[69] Y. Song and U. Hengartner, "Privacyguard: A vpn-based platform to detect information leakage on android devices", in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, ACM, 2015, pp. 15–26.

[70] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "ReCon: Revealing and controlling privacy leaks in mobile network traffic", *CoRR, abs/1507.00255*, 2015.

[71] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "Mockdroid: trading privacy for application functionality on smartphones", in *Proceedings of the 12th workshop on mobile computing systems and applications*, ACM, 2011, pp. 49–54.

[72] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications", in *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, 2011, pp. 639–652.

[73] K. Fawaz and K. G. Shin, "Location privacy protection for smartphone users", in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2014, pp. 239–250.

[74] J.-Y. Pan and S.-H. Ma, "Advertisement removal of Android applications by reverse engineering", in *Computing, Networking and Communications (ICNC), 2017 International Conference on*, IEEE, 2017, pp. 695–700.

[75] H. Kawabata, T. Isohara, K. Takemori, A. Kubota, J. Kani, H. Agematsu, and M. Nishigaki, "SanAdBox: Sandboxing third party advertising libraries in a mobile application", in *2013 IEEE International Conference on Communications (ICC)*, Jun. 2013, pp. 2150–2154. DOI: 10.1109/ICC.2013.6654845.

[76] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation", in *Acm sigplan notices*, ACM, vol. 40, 2005, pp. 190–200.

[77] E. Bodden, "Easily Instrumenting Android Applications for Security Purposes", in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13, Berlin, Germany: ACM, 2013, pp. 1499–1502, ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516759. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516759.

[78] A. Bartel, J. Klein, M. Monperrus, K. Allix, and Y. L. Traon, "Improving Privacy on Android Smartphones Through In-Vivo Bytecode Instrumentation", *CoRR*, vol. abs/1208.4536, 2012.

[79] R. Neisse, G. Steri, D. Geneiatakis, and I. N. Fovino, "A privacy enforcing framework for Android applications", *Computers & Security*, vol. 62, pp. 257–277, 2016.

[80] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky, "Appguard–enforcing user requirements on android apps", in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2013, pp. 543–548.

[81] S. Rasthofer, S. Arzt, E. Lovat, and E. Bodden, "Droidforce: Enforcing complex, data-centric, system-wide policies in android", in *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, IEEE, 2014, pp. 40–49.

[82] Q. Zhang, X. Li, X. Yu, and Z. Feng, "ASF: Improving Android Security with Layered Structure Instrumentation", in *Contemporary Research on E-business Technology and Strategy*, ser. Communications in Computer and Information Science, V. Khachidze, T. Wang, S. Siddiqui, V. Liu, S. Cappuccio, and A. Lim, Eds., Springer Berlin Heidelberg, 2012, pp. 147–157, ISBN: 978-3-642-34446-6. DOI: 10.1007/978-3-642-34447-3_13. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34447-3_13.

[83] S. Dai, T. Wei, and W. Zou, "DroidLogger: Reveal suspicious behavior of Android applications via instrumentation", in *Computing and Convergence Technology (ICCCT), 2012 7th International Conference on*, Dec. 2012, pp. 550–555.

[84] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly", in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS '03, Washington D.C., USA: ACM, 2003,

pp. 290–299, ISBN: 1-58113-738-9. DOI: 10.1145/948109.948149. [Online]. Available: http://doi.acm.org/10.1145/948109.948149.

[85] P. Schulz, "Code Protection in Android", 2012.

[86] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries", in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 255–270.

[87] H. Cai, Z. Shao, and A. Vaynberg, "Certified self-modifying code", *SIGPLAN Not.*, vol. 42, no. 6, pp. 66–77, Jun. 2007, ISSN: 0362-1340. DOI: 10.1145/1273442.1250743. [Online]. Available: http://doi.acm.org/10.1145/1273442.1250743.

[88] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon", in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ser. IMC '06, Rio de Janeriro, Brazil: ACM, 2006, pp. 41–52, ISBN: 1-59593-561-4. DOI: 10.1145/1177080.1177086. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177086.

[89] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors", in *USENIX workshop on large-scale exploits and emergent threats (LEET)*, 2009.

[90] D. Ehringer, "The dalvik virtual machine architecture", *Techn. report (March 2010)*, 2010.

[91] *dex2jar*, http://code.google.com/p/dex2jar/.

[92] J. Oberheide and C. Miller, "Dissecting the android bouncer", *SummerCon2012, New York*, 2012.

[93] R. Unuchek, *The most sophisticated Android Trojan*, http://www.securelist.com/en/blog/8106/The_most_sophisticated_Android_Trojan, Jun. 2013. (visited on 06/06/2013).

[94] J. Mayer, "Tracking the trackers: Early results", *The Center for Internet and Societys Blog*, 2011.

[95] G. Kontaxis and M. Chew, "Tracking Protection in Firefox For Privacy and Performance", *W2SP*, 2014.

[96] Engadet, *iOS 9's web browser can block annoying ads*, 2015. [Online]. Available: http://www.engadget.com/2015/06/10/safari-in-ios-9-can-block-ads/ (visited on 07/22/2015).

[97] D. Malandrino, A. Petta, V. Scarano, L. Serra, R. Spinelli, and B. Krishnamurthy, "Privacy awareness about information leakage: who knows what about me?", in *12th ACM WPES workshop*, ACM, 2013, pp. 279–284.

[98] S. Kamkar, "Evercookie-virtually irrevocable persistent cookies", *His Blog*, vol. 9, 2010.

[99] B. Krishnamurthy, K. Naryshkin, and C. Wills, "Privacy leakage vs. protection measures: the growing disconnect", *W2SP*, vol. 2, pp. 1–10, 2011.

[100] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in android ad libraries", in *Workshop on Mobile Security Technologies (MoST)*, Citeseer, 2012.

[101] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements", in *WISEC*, ACM, 2012, pp. 101–112.

[102] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of android ad library permissions", *arXiv preprint arXiv:1303.0857*, 2013.

[103] J. Mayer, *The Turn-Verizon Zombie Cookie*, http://webpolicy.org/2015/01/14/turn-verizon-zombie-cookie/, 2015. (visited on 04/14/2015).

[104] Ars Technica, *How a banner ad for H&R Block appeared on apple.com without Apple's OK |*, http://arstechnica.com/tech-policy/2013/04/how-a-banner-ad-for-hs-ok/, Apr. 2013. (visited on 04/14/2015).

[105] The Register, "SSL-busting adware: US cyber-plod open fire on Comodo's PrivDog", 2015, http://www.theregister.co.uk/2015/02/24/comodo_ssl_privdog/.

[106] C. Evans, C. Palmer, and R. Sleevi, *Public key pinning extension for HTTP*, http://www.rfc-editor.org/rfc/rfc7469.txt, 2015.

[107] M. Kranch and J. Bonneau, "Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning.", in *NDSS*, 2015.

[108] Ars Technica, *Over 300 businesses now whitelisted on AdBlock Plus, 10% pay to play*, http://arstechnica.com/business/2015/02/over-300-businesses-now-whitelisted-on-adblock-plus-10-pay-to-play/, 2015.

[109] S. Englehardt, C. Eubank, P. Zimmerman, D. Reisman, and A. Narayanan, "OpenWPM: An automated platform for web privacy measurement", 2015.

[110] A. Cortesi, *mitmproxy*, https://mitmproxy.org/, Feb. 2016.

[111] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of Google Play", in *SIGMETRICS*, ACM, 2014, pp. 221–233.

[112] M. Falahrastegar, H. Haddadi, S. Uhlig, and R. Mortier, "Anatomy of the third-party web tracking ecosystem", *arXiv preprint arXiv:1409.1066*, 2014.

[113] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246 (Proposed Standard), Updated by RFCs 5746, 5878, 6176, Internet Engineering Task Force, Aug. 2008. [Online]. Available: `http://www.ietf.org/rfc/rfc5246.txt`.

[114] P. Hoffman and J. Schlyter, *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*, RFC 6698 (Proposed Standard), Updated by RFC 7218, Internet Engineering Task Force, Aug. 2012. [Online]. Available: `http://www.ietf.org/rfc/rfc6698.txt`.

[115] CMU, *Perspectives Project*, `http://www.perspectives-project.org/`, 2016.

[116] Thoughtcrime Labs, *Convergence Notary Protocol*, `https://github.com/moxie0/Convergence/wiki/Notary-Protocol`, 2016.

[117] ICSI, *ICSI Certificate Notary*, `http://notary.icsi.berkeley.edu/`, 2016.

[118] Alexa, *Alexa Top Domains*, `http://www.alexa.com/topsites`.

[119] C. Fidas, A. Voyiatzis, and N. Avouris, "When Security Meets Usability: A User-Centric Approach on a Crossroads Priority Problem", in *14th Panhellenic Conference on Informatics (PCI 2010)*, IEEE, 2010, pp. 112–117.

[120] D. Buhov, M. Huber, G. Merzdovnik, E. Weippl, and V. Dimitrova, "Network Security Challenges in Android Applications", in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security (ARES 2015)*, IEEE Computer Society, 2015, pp. 327–332.

[121] X. Jiang, *An Evaluation of the Application ("App") Verification Service in Android 4.2*, `http://www.cs.ncsu.edu/faculty/jiang/appverify`, Dec. 2012.

[122] J. Oberheide and C. Miller, "Dissecting the Android Bouncer", in *SummerCon*, 2012.

[123] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An Android Application Sandbox System for Suspicious Software Detection", in *Pro-*

ceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE), 2010.

[124] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox", *IEEE Security and Privacy*, vol. 5, no. 2, 2007.

[125] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A Tool for Analyzing Malware", in *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR) Annual Conference*, 2006.

[126] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools", *ACM Computing Surveys Journal*, vol. 44, no. 2, 2012.

[127] Alcatel-Lucent, *Kindsight Security Labs Malware Report - Q2 2013*, `http://www.kindsight.net/sites/default/files/Kindsight-Q2-2013-Malware-Report.pdf`, Jul. 2013.

[128] Trend Micro, *TrendLabs 2Q 2013 Security Roundup*, `http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-2q-2013-trendlabs-security-roundup.pdf`, Aug. 2013.

[129] E. Kalige and D. Burkey, *A Case Study of Eurograbber: How 36 Million Euros was Stolen via Malware*, `https://www.checkpoint.com/products/downloads/whitepapers/Eurograbber_White_Paper.pdf`, Dec. 2012.

[130] Juniper Networks, *Juniper Networks Third Annual Mobile Threats Report*, `http://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2012-mobile-threats-report.pdf`, Jun. 2013.

[131] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces", in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012.

[132] H. Yao and D. Shin, "Towards Preventing QR Code Based Attacks on Android Phone Using Security Warnings", in *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2013.

[133] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution", in *Proceedings of the 33rd Annual IEEE Symposium on Security and Privacy (S&P)*, 2012.

[134] F. Maggi, A. Valdi, and S. Zanero, "AndroTotal: A Flexible, Scalable Toolbox and Service for Testing Mobile Malware Detectors", in *Proceedings of the 3rd*

ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), 2013.

[135] A. Desnos and G. Gueguen, "Android: From Reversing to Decompilation", in *Black Hat Abu Dhabi*, Dec. 2011.

[136] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security", in *Proceedings of the 20th USENIX Security Symposium*, 2011.

[137] D. Octeau, W. Enck, and P. McDaniel, "The ded Decompiler", Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, Tech. Rep. NAS-TR-0140-2010, Sep. 2010.

[138] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection", in *Proceedings of the 10th Annual Network & Distributed System Security Symposium (NDSS)*, 2003.

[139] T. Raffetseder, C. Kruegel, and E. Kirda, "Detecting System Emulators", in *Proceedings of the 10th Information Security Conference (ISC)*, 2007.

[140] G. Sarwar, O. Mehani, R. Boreli, and M. A. Kaafar, "On the Effectiveness of Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices", in *Proceedings of the 10th International Conference on Security and Cryptography (SECRYPT)*, 2013.

[141] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. Wang, and B. Zang, "Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis", in *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[142] *DroidBox: An Android Application Sandbox for Dynamic Analysis*, `https://code.google.com/p/droidbox/`.

[143] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger, "ANANAS - A Framework For Analyzing Android Applications", in *Proceedings on the 1st International Workshop on Emerging Cyberthreats and Countermeasures (ECTCM)*, 2013.

[144] L. K. Yan and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis", in *Proceedings of the 21st USENIX Security Symposium*, 2012.

[145] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. van der Veen, and C. Platzer, "Andrubis: Android Malware Under The Magnifying

Glass", Vienna University of Technology, Tech. Rep. TR-ISECLAB-0414-001, 2014.

[146] V. van der Veen, "Dynamic Analysis of Android Malware", *Internet & Web Technology Master thesis, VU University Amsterdam*, 2013.

[147] A. Reina, A. Fattori, and L. Cavallaro, "A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors", in *Proceedings of the 6th European Workshop on System Security (EUROSEC)*, 2013.

[148] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications", in *Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012.

[149] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: Automatic Security Analysis of Smartphone Applications", in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2013.

[150] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection", in *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[151] D. Barrera, J. Clark, D. McCarney, and P. C. van Oorschot, "Understanding and Improving App Installation Security Mechanisms Through Empirical Analysis of Android", in *Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012.

[152] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta, "On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core", in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[153] M. G. Kang, H. Yin, S. Hanna, S. McCamant, and D. Song, "Emulating Emulation-Resistant Malware", in *Proceedings of the 2nd Workshop on Virtual Machine Security (VMSec)*, 2009.

[154] D. Balzarotti, M. Cova, C. Karlberger, E. Kirda, C. Kruegel, and G. Vigna, "Efficient Detection of Split Personalities in Malware", in *Proceedings of the 17th Annual Network & Distributed System Security Symposium (NDSS)*, 2010.

[155] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti, "Detecting Environment-Sensitive Malware", in *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection (RAID)*, 2011.

[156] F. Matenaar and P. Schulz, *Detecting Android Sandboxes*, `http://dexlabs.org/blog/btdetect`, Aug. 2012.

[157] P. Ferrie, "Attacks on More Virtual Machine Emulators", Symantec Research White Paper, Tech. Rep., 2007.

[158] C. Kolbitsch, E. Kirda, and C. Kruegel, "The Power of Procrastination: Detection and Mitigation of Execution-Stalling Malicious Code", in *Proceedigns of the 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[159] Comodo Malware Analysis Team, *Android OBAD - Technical Analysis Paper*, `http://www.comodo.com/resources/Android_OBAD_Tech_Reportv3.pdf`, 2013.

[160] Z. Ashraf, *DIY: Android Malware Analysis – Taking apart OBAD (part 1)*, `http://securityintelligence.com/diy-android-malware-analysis-taking-apart-obad-part-1/`, Oct. 2013.

[161] U. of Michigan, *Scans.io - Internet-Wide Scan Data Repository*, `https://scans.io/`, Feb. 2016.