DISSERTATION

Threats to Privacy Sensitive Data

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Priv.-Doz. Dipl.-Ing. Dr. Engin Kirda
und
Priv.-Doz. Dipl.-Ing. Dr. Christopher Krügel

Institut für Rechnergestützte Automation
Arbeitsgruppe Automatisierungssysteme (E183-1)

eingereicht an der Technischen Universität Wien,
Fakultät für Informatik

von

Dipl.-Ing. Mag. Gilbert Wondracek

Fasangasse 43/2/8
1030 Wien
9925208

Wien, 30. November 2009

Kurzfassung

In den letzten Jahren sind hunderte Millionen von Benutzern zu Opfern von Cybercrime geworden. Bösartige Software (Malware) oder Aktivitäten wie Daten- oder Identitätsdiebstahl, Phishing, Botnetze, Trojaner oder gezielte Spamkampagnen sind eine ernsthafte Bedrohung für die Sicherheit und den Schutz von sensiblen und privaten Daten von Benutzern. Diese Dissertation präsentiert neuartige Lösungsansätze und Techniken für drei Problemfelder innerhalb des Gebiets der Computersecurity.

Zuerst stellen wir ein neuartiges Bedrohungsszenario für soziale Netzwerke (z.B. Facebook, LinkedIn, Xing) vor, welches es einem Angreifer ermöglicht, eine große Anzahl von Benutzern zu deanonymisieren. Wir zeigen, sowohl theoretisch als auch experimentell, dass ein derartiger Angriff mit relativ geringem Aufwand in der Realität durchführbar ist, und die persönlichen Daten und die Privatsphäre von Millionen von Benutzern gefährdet.

Des Weiteren demonstrieren wir anhand einer Studie die Verbindung zwischen Cybercrime und der Internet-Schattenwirtschaft (underground economy). Wir führen eine technische und wirtschaftliche Untersuchung der Online-Adult Branche durch, und zeigen, dass undurchsichtige Geschäftsmodelle mit traditionellen Securitybedrohungen Hand in Hand gehen. Dies berührt im Besonderen die Themen traffic trading, Betrug in Partnerprogrammen, das Ausspähen von privaten Browserdaten und Malware Bedrohungen (drive-by-downloads).

Schließlich präsentieren wir "Prospex", ein System zum automatischen Reverse-Engineering von Netzwerkprotokollen. Durch dynamische Taint-Analyse ist es möglich, Rückschlüsse auf das interne Verhalten von Programmen, die ein Protokoll implementieren, zu erhalten. Wir führen neuartige Methoden ein, mit denen genaue Format- und Typbeschreibungen für Protokollnachrichten generiert werden können, und ein Zustandsautomat abgeleitet werden kann. Als konkrete Anwendung zeigen wir, dass automatisch generierte Protokollbeschreibungen zum Fuzz-Testing von existierender Software verwendet werden kann und damit reale Sicherheitslücken gefunden werden können.

Abstract

In recent years, security and privacy threats like data or identity theft, phishing, credential stealing trojans, botnets, or targeted spam campaigns have affected millions of users and online businesses. Researchers have acknowledged these threats, and are actively exploring potential attack vectors and developing solutions and countermeasures. In this doctoral thesis, we present new approaches and techniques to three problems in the domain of computer security that severly impact user privacy.

First, we introduce a novel attack scenario against social networks (e.g., Facebook, LinkedIn, Xing), that potentially allows a miscreant to de-anonymize a large amount of social network users. We demonstrate, both theoretically and practically, that this is feasible in a real-world scenario, thus compromising the privacy and security of millions of users.

Second, we conduct a study on cybercrime and the underground economy. Specifically, we investigate shady business practices using the example of the online adult industry and perform an economic and technical analysis. Furthermore, we provide a real-world evaluation of security issues in this domain, including traffic trading, affiliate fraud, history stealing, and malware (drive-by-downloads) vulnerability assessments.

Finally, we present "Prospex", a system that aims at automatic network protocol reverse engineering. By applying dynamic taint analysis on binaries, we can observe the internal behavior of programs that implement an application level protocol. Then, we use novel techniques to identify message formats and types, and infer a protocol state machine. As an application of our system, we show that we successfully used the recovered protocol specifications as input to a fuzz testing tool, allowing us to find security vulnerabilities in real-world software.

Contents

List of Figures

List of Tables

Acknowledgments

Writing this thesis would not have been possible without the help from many people. First of all, I would like to express my gratitude to my advisors, Prof. Dr. Engin Kirda and Prof. Dr. Christopher Krügel, who made this work possible. Their inspiring work and advice helped me to develop new skills and complete this thesis, and I am looking forward to working with them in the near future.

In addition, I would like to thank all colleagues and members of the International Secure Systems Lab and the Institut für Rechnergestützte Automation at the Vienna University of Technology for the great cooperation and the fun we had.

Last but not least, my gratitude goes to my family and my girlfriend Cornelia for their endless love and support during the past years.

Chapter 1

Introduction

In recent years, security researchers and professionals have experienced a global paradigm shift from "hacking for fun" to profit-driven cybercrime. This development is fueled by an underground economy that is estimated to have a volume in the range of billions of dollars.

In the wake of real-world threats like data or identity theft, phishing, malware, trojans, botnets, spam, or credit card fraud, security and privacy issues have become a major concern for hundreds of millions of users. For example, every day, web surfers are tricked into installing scareware, hackers use mass exploits to automatically compromise thousands of systems, while personal and private data or credit card information are openly traded in underground marketplaces on the Internet.

Researchers have acknowledged these threats, and are actively exploring potential attack vectors and developing solutions and countermeasures in areas that range from web security to binary analysis. The ongoing race between researchers and miscreants constantly leads to the exploration of new theoretical and practical approaches to security, combining aspects of many computer science disciplines.

1.1 Security and Privacy

One particularly interesting area within computer security is privacy. For example, the renowned security researcher Bruce Schneier emphasizes this, by defining privacy as "Privacy is an inherent human right, and a requirement for maintaining the human condition with dignity and respect." [26]. The scope of the challenges and issues that are related to privacy is constantly expanding, as more and more users use (online) services and applications that store personal and sensitive data (for example, search engines, social networks, photo sharing websites, online banking, or discussion forums). While, intuitively, most people would agree that their personal data should be protected, many people willingly, and even more worrisome, unwillingly, give up information such as their names, addresses, religion, sexual and political preferences,

friends, location, and even medical conditions.

Ideally, users should be aware of how their personal information is being used, and able to retain control on how it is processed and propagated. However, this is an ambitious goal, as the high complexity of many applications makes it increasingly difficult or even impossible for users to anticipate the implications of their actions. For example, users might not be aware that a seemingly private messaging area in a social network is accessible by search engine crawling bots, or that web site owners may share their customer data with data mining or marketing companies that then use this information for targeted advertising. Also, many (legitimate) services make it overly complicated to change privacy settings or do not provide adequate options at all. These shortcomings, along with lax restrictions and a large amount of users that are not aware of privacy issues, are facilitating cybercrime scenarios, such as identity theft, unsolicited e-mail (spam), or phishing.

From a research point of view, privacy challenges are particularly interesting, as they play an important role in a wide array of security research directions. Throughout this work, we will show how security issues from seemingly different domains are linked by the underlying implications to the privacy of user data.

We think that it is crucial to the security of users to research and develop novel, privacy enhancing solutions, as well as to find potential attack vectors and weaknesses in existing systems and techniques. It is equally important to create awareness among users about privacy issues, and educate them on how to protect their private information.

1.2 Contributions

In this thesis, we also want to show that privacy issues and challenges act as a common link between individual aspects of computer security. To this end, we contribute to the domains of web security, a study on the underground economy, and protocol reverse engineering and highlight privacy-relevant implications.

In summary, we make the following main contributions in this thesis:

- First, we present a novel de-anonymization scenario that is directed against social networks. These networks (for example, Facebook, MySpace) are used by millions of users, who often publish sensitive personal information in their member profiles. We demonstrate, both theoretically and experimentally, that a malicious attacker can, with relatively low effort, use a web based attack to de-anonymize a large amount of so-

cial network members, thus potentially compromising the privacy of millions of users. Furthermore, we evaluate the de-anonymization scenario on real-world social networks and show that an attacker only requires access to publicly available data.

- Second, we investigate the relationship between the underground economy and the online industry. To this end, we observe a whole business branch, the online adult industry, and examine its associated security and privacy issues. Furthermore, we show the economic and technical interdependencies between its participants and demonstrate attacks against website visitors. Our study is the first on this subject to combine a technical analysis with economical aspects.

- Finally, we present a system for automatically reverse engineering application level network protocols. By utilizing binary analysis, our system is able to perform a behavioral analysis of server applications. Our system can be used to analyze unknown or proprietary server programs as well as malicious bots that are a threat to users and their private information. We have developed novel techniques that allow the reversing of individual protocol message formats, discover message types, and infer a protocol state machine. As an application, we used the protocol specifications generated by our system to instrument fuzz testing software. This allowed us to successfully find security vulnerabilities in real-world server applications.

1.3 Organization of this Thesis

This dissertation is organized as follows. In Chapter 2, we present our work on de-anonymizing users of social networks. In Chapter 3, we present our findings on the relationship between the online adult industry and cybercrime. Chapter 4 presents "Prospex", our system for automatically extracting protocol specifications from binaries. Chapter 5 lists the related work for each of the systems presented in this work. Finally, in Chapter 6, we conclude this thesis.

1.4 List of Publications

Parts of this thesis are based on publications listed in this section. In particular, Chapter 4 was published as a paper together with Milani-Comparetti,

Kirda and Krügel [85]. Additionally, two papers based on the content of Chapter 2 and Chapter 3 are currently under submission to academic, peer-reviewed conferences.

Chapter 2

De-Anonymization of Social Networks

2.1 Introduction

Social networking sites such as Facebook, LinkedIn, Twitter, and Xing have been increasingly gaining in popularity [6]. In fact, Facebook has been reporting growth rates as high as 3% per week, with more than 300 million registered users as of September 2009 [2]. Furthermore, this site has more than 30 billion page views per month, and it is reported to be the largest photo storage site on the web with over one billion uploaded photos. Clearly, popular social networking sites are critical with respect to security and especially privacy due to their very large user base.

Of course, social networking sites are not less secure than other types of websites. However, the difference to other sites lies in the amount of private and possibly sensitive data that they store. Social networking sites are typically used to contact friends, discuss specific topics in online discussion forums and groups, establish new business contacts, or simply to keep in touch with other people. Along with the information about friendships and acquaintances, users often provide a great deal of personal information that may be interesting for attackers. Although social networking sites employ mechanisms to protect the privacy of their users, there is always the risk that an attacker can correlate data or abuse the structure of a social network to infer information about registered individuals [32, 4, 128].

In this work, we introduce a novel de-anonymization attack against users of social networking sites. In particular, we show that information about the *group memberships* of a user (i.e., the groups of a social network to which a user belongs) is often sufficient to uniquely identify this user. When unique identification is not possible, then the attack might still significantly reduce the size of the set of candidates that the victim belongs to.

To make the de-anonymization attack practical, we present a way in which an adversary can learn information about the group memberships of a user who is just browsing the web. To do this, an attacker can leverage the well-known technique of *history stealing* [67, 83]. More precisely, using history stealing, an attacker can probe the browser history of a victim for certain

URLs that reveal group memberships on a social network. By combining this information with previously collected group membership data from the social network, it is possible to de-anonymize any user (of this social network) who visits the attacker's website. In some cases, this allows an attacker who operates a malicious website to uniquely identify his visitors by their name (or, more precisely, the names used on the corresponding social network profiles).

Previous work in the area of de-anonymization was mostly focusing on correlating information from several independent data sets (datasets from different sources). For example, Griffith and Jakobsson used public records such as marriage and birth information to derive a mother's maiden name [58]. Narayanan and Shmatikov showed, in two recent papers, that information from different data sources can be combined to de-anonymize a user [89, 4]. In contrast, our attack uses only information from a single social networking site, and combines it with intrinsic information that is generated while users access the site. That is, our attack makes use of the fact that the browser records the URLs of the social networking site that a user visits (since browsers typically keep an access history for some time).

To demonstrate that our attack works, we performed both a theoretical analysis and empirical measurements for users of the Xing social network. The results suggest that our attack can be used to potentially de-anonymize millions of users. Due to the limited resources that were available to us, we focused our empirical evaluation on Xing, a medium-sized network that has eight million registered users. We managed to extensively collect data for this network and achieved a high coverage of its groups and members. However, to demonstrate that the attack is not conceptually limited to one social network, we also performed an empirical feasibility study on two other, significantly larger networks: Facebook and LinkedIn. Furthermore, we also briefly study five other social networks and find that they are also vulnerable to our attack.

Our attack can also be generalized to other websites that generate *sparse datasets* (i.e., the information about each individual user covers only a small fraction of the overall attributes) [89]. In the case of social networks, our attack works because even the most active user is only a member of a small fraction of all groups, and thus, the group membership information serves as a fingerprint. Sparse datasets are common with websites that deal with user data. For example, Amazon and eBay use concepts similar to groups on social networks ("Customer Communities" and "Groups," respectively), meaning that they are both potentially vulnerable to our de-anonymization attack.

In summary, we make the following three contributions:

- We introduce a novel de-anonymization attack, and show how it can be applied to social networking sites. The key idea of the attack is to

exploit information about specific users (in this case, membership in groups) combined with the victim's browsing history.

- We demonstrate several techniques to scale our attacks to real-world social networking sites. This is a challenging task since these websites often have tens of millions of users.

- We provide both a theoretical analysis and empirical measurements to demonstrate the feasibility of our attack. The results indicate that about 42% of users in the social network Xing can be reliably de-anonymized. Furthermore, we empirically show that both Facebook and LinkedIn are also potentially vulnerable.

2.2 Background

In this section, we provide a brief introduction to the background concepts to allow the reader to better understand our attack. We first present a model of social networks, and define the terminology we use within this thesis. We then list our assumptions about the attacker. We continue with an overview of the common structure of social networks, and discuss the aspects of this structure that we exploit. Finally, we explain why social networks are commonly prone to history stealing attacks.

2.2.1 Model and Definitions

Throughout this work, we use the following models and definitions to describe our de-anonymization attack.

Social Networks A social network S is modeled as a graph $G = (V, E)$ containing nodes V representing users, and undirected edges E representing the "friendship" relation between two users. Furthermore, the social network contains G groups. Each group $g \in G$ contains a set of users from V: $\forall g \in G : g \subseteq V$. Social networks typically do not allow empty groups without any user (and also actively delete such groups). Thus, we can assume, without loss of generality, that $\forall g \in G : g \neq \emptyset$.

Each user $v \in V$ is a member of n groups, with $n \geq 0$. We model this information as a vector $\Gamma(v) := (\Gamma_g(v))_{g \in G}$ such that:

$$\Gamma_g(v) = \begin{cases} 1 & \text{if } v \text{ is a member of group } g \\ 0 & \text{if } v \text{ is not a member of group } g \end{cases} \tag{2.1}$$

For each group g in which v is a member, one dimension of $\Gamma(v)$ is set to one. Otherwise, this dimension is set to zero. For the case of $n = 0$ (i.e., the user is not a member in any group), the vector $\Gamma(v)$ contains only zeros. This is the worst case for our attack.

As we will show, the vector $\Gamma(v)$ can be used to de-anonymize users within a social network. In particular, $\Gamma(v)$ serves as the *group fingerprint* of a user, and we demonstrate in our experiments that this fingerprint, in practice, is characteristic for a given user.

Browser History A building block that we use during our attack is the browsing history β_v of a user v. A web browser maintains a list of web pages that the user has recently visited. Every time a user visits a page p, the URL ϕ_p that was used to load this page is added to β_v. Moreover, entries in β_v expire. That is, after a time interval τ has elapsed, the URL related to p is removed from β_v. The timeout itself depends on the browser and user settings. For example, Mozilla Firefox uses $\tau = 90$ days by default, while Microsoft Internet Explorer only uses $\tau = 20$ days. Apple Safari is between both browsers with $\tau = 1$ month by default, whereas Google Chrome has an unlimited history timeout $\tau = \infty$.

Attacker Model We have two basic assumptions about an attacker. First, we assume that the attacker can determine which web pages, from a given set, a specific user v has accessed in the recent past (within time τ). This means that the attacker can determine whether or not a given URL ϕ_p is in β_v. The attacker, thus, has a method to compute, for a given victim v, the function $\sigma_v(\phi_p)$, which is defined as follows:

$$\sigma_v(\phi_p) = \begin{cases} 1 & \text{if } \phi_p \in \beta_v \text{ for the user } v \\ 0 & \text{if } \phi_p \notin \beta_v \text{ for the user } v \end{cases} \tag{2.2}$$

It is reasonable to assume that such a function exists and that the attacker can perform the computation based on *history stealing*, as we show in Section 2.2.3.

The second assumption is that the attacker has a way to learn about the members of groups for a given social network S. As defined above, a group g is a non-empty subset of the overall users of S. The attacker does not need to have the membership information for all groups $g \in G$. However, knowledge about more groups makes the attack more efficient. In Section 2.3.3, we discuss how an attacker can obtain the necessary group membership information.

We believe that our two assumptions about an attacker can be (easily) satisfied in practice, and our empirical experiments support this claim. Moreover, as we will discuss in Section 2.3, our attack is able to tolerate a certain amount

Name	# users	Focus	Alexa rank	Groups
Facebook	300,000,000+	general audience, global	2	✔
MySpace	260,000,000+	music, global	11	✔
Friendster	90,000,000+	general audience, global	111	✔
LinkedIn	50,000,000+	business, global	53	✔
StudiVZ	15,000,000+	students, Germany	179	✔
Xing	8,000,000+	business, Europe	285	✔
Bigadda	5,500,000+	teenage audience, India	3,082	✔
Kiwibox	2,500,000+	teenage audience, global	74,568	✔

Table 2.1: Overview of popular social networking websites. The data is based on information provided by individual social networks, public sources such as Alexa [6], and our analysis.

of inaccuracy. That is, even when the history stealing attack does not produce a perfect group fingerprint $\Gamma(v)$ for a victim v, or when the attacker's view of the social network is different than the network's actual state (e.g., due to users who join and leave groups), the attack can still be successful. However, in such cases, it typically proceeds slower and produces larger candidate sets.

2.2.2 Structure of Social Networking Sites

Overview

Most social networking sites share the same basic structure. Each user v within the network has a *profile* p_v that contains (partially sensitive) information. This information, for example, can be the user's full name, photographs, date of birth, relationship status, former and current employers, and education. One of the core technical components of a social network is its website, and the underlying web application. The web application provides the main functionalities of the social network. This functionality often comprises of features that allow a web visitor to become a member, to edit personal profiles, to view other user profiles, or to join groups. To become a member of a social network, users can sign up at the website. This process usually only requires a valid e-mail address for verification purposes.

Since social networks can have millions of users, most popular social networks (see Table 2.1) include features that allow users to be organized in *groups*. This feature allows users of a social network to easily find other users with whom they might share specific interests (e.g., same hobbies), demographic groups (e.g., studying at the same university), political or sexual-

orientation, and even medical conditions. Typically, there exists some kind of hierarchy within a group. That is, particular members can hold the role of administrators or moderators, which grants them some special privileges (e.g., sending messages to the whole group, or removing members). In general, two different types of groups exist:

- *Public groups*: These groups allow all members of the social network to join. Typically, members are automatically added to the group when they wish to join. Interestingly, we found that some social networks even allow non-group members to list the members of public groups (e.g., Facebook).
- *Closed groups*: On the other hand, *closed groups* require some sort of authorization before a member is allowed to join. In practice, this means that a group administrator or moderator needs to manually approve each membership request.

The different social networks vary widely in the number of available groups. Networks that target a general audience typically have a large number of groups, and the average user is a member of many groups. Social networks that target business users, on the other hand, have a smaller number of groups, and the average user is only a member in a few groups (see Section 2.5 for more specific results).

Web Applications

```
(1) http://www.facebook.com/home.php?ref=home
(2) http://www.facebook.com/ajax/profile/picture/upload.php?id=[userID]
(3) http://www.facebook.com/group.php?gid=[groupID]&v=info&ref=nf
(4) https://www.xing.com/net/[groupID]/forums
```

Figure 2.1: Examples of distinct types of web application hyperlinks for different social networks.

The web applications for the most popular social networks (see Table 2.1) rely on hyperlinks and HTTP GET parameters to implement the communication between a user (more precisely, her browser) and the actual web application. For example, Figure 2.1 shows four real-world examples from the web application of Facebook and Xing that are representative for two groups of hyperlinks. The first link is used to tell the web application to display the

currently logged-in user's "home" area. Since the hyperlink for this operation is the same for every user of the social network, we refer to links of this type as *static hyperlinks*. In contrast, the other links are used to inform the web application of state changes requested by a user. For example, the second link sends a request to the web application that the user with the ID `userID` wishes to upload a new profile picture. This link contains a dynamic token (in this case, the ID of user v), so we consequently call it a *dynamic hyperlink*. This type of links explicitly contains information about a user since the link is unique for each user of the social network (i.e., this link identifies a particular $v \in V$).

Besides dynamic hyperlinks that contain information about users, there also exist dynamic hyperlinks that contain (or embed) information about groups. The third and fourth link of Figure 2.1 show two examples in which information about a specific group (i.e., a `groupID` parameter) is encoded within the hyperlink. Note that each link uniquely refers a group $g \in G$.

From the web application's point of view, these hyperlinks facilitate the "internal" state keeping and communication between the web application and the user's web browser. Since web browsers are agnostic to the semantic interpretation of links, they simply add the URLs of all visited web pages to the browsing history β_v of a user v. Note that since the interesting information is already encoded in the URL itself, it does not matter if the website is using security-enhancing protocols such as HTTPS for protecting the actual content. The URL is nevertheless added to the browser's history. From an attacker's point of view, this behavior is interesting, since it enables the attacker to identify groups a user has visited, and even potentially identify a specific user. That is, if the attacker is able to determine which pages are in the victim's browsing history (i.e., she can compute the function $\sigma_v(\phi_p)$ for pages loaded via dynamic hyperlinks ϕ_p), she can use this information to de-anonymize a user v (as shown in more detail later).

2.2.3 History Stealing

History stealing is a known attack in which a malicious website can extract the browsing history of a visitor. One of the first descriptions of this attack dates back to the year 2000 [106], and the technique was re-discovered several times in the recent years (e.g., [16, 24]). The core observation behind this attack is the fact that a web browser treats hyperlinks differently depending on whether or not a hyperlink was previously accessed by a user. This means that a browser implements the function $\sigma_v(\phi_p)$ (that is, the browser implicitly checks whether a target URL ϕ_p is in the browsing history β_v). Typically, hyperlinks to web pages in the browsing history are displayed in a different

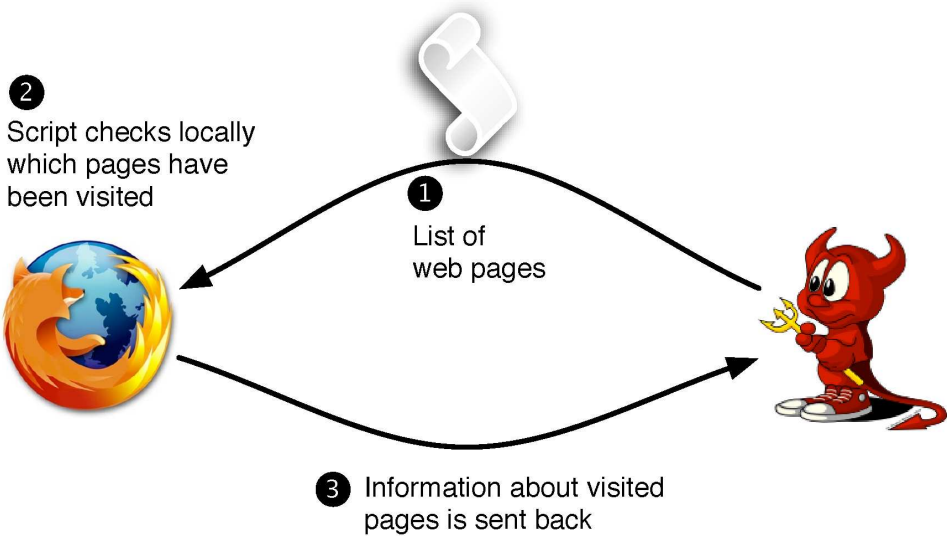


Figure 2.2: Schematic overview of history stealing attack.

color to indicate to the user that this link has been clicked in the past. An attacker can use various techniques to probe whether or not a web page is in the browsing history:

- An attacker can create an HTML page with links to target web pages of interest and use background image tags in the `a:visited` style information. Since images can be referenced with URLs, the user's browser will then access these URLs if a target web page is in β_v.
- Alternatively, an attacker can also use client-side scripting (for example, JavaScript) to generate and iterate over a list of target links and programatically check for the `a:visited` style to see if a link is present in β_v.

Note that an attacker has to probe for each URL (and cannot simply access the full browsing history of a victim), obtaining one bit of information per URL that determines whether it is contained in β_v or not.

From a schematic point of view, each attack scenario is the same (see Figure 2.2): First, the attacker sends a list of URLs to the victim's browser. Second, the attacker forces the browser to check for each URL whether or not it is contained in the browsing history using one of the methods discussed above. Finally, a report is sent back to the attacker, who then obtains a list of URLs that are contained in the victim's browsing history.

History stealing can be used for different kinds of attacks. For example, it can be used for more effective phishing. Jakobsson and Stamm presented

a scenario where the attacker checks the browsing history for various bank site URLs. If the attacker sees that the victim has visited a certain bank, she can then launch targeted phishing attacks [83] that target this bank. In the context of web applications, this means that an attacker can apply this technique to reconstruct knowledge on past interaction between the victim and the web application. While this knowledge alone might not be sufficient for many attack scenarios (e.g., an attacker would still need the online banking credentials to withdraw funds – requiring a phishing step), we show that we can successfully improve this technique to de-anonymize users of social networks.

All popular browsers (e.g., IE, Firefox, Safari) are vulnerable to history stealing attacks in their default settings (i.e., when the browser keeps a history of visited pages). To date, the problem has not been solved as it is often viewed a usability feature/design issue rather than a browser bug.

2.2.4 Possible Attack Scenarios

De-anonymizing website visitors allows an adversary to launch targeted attacks against unsuspecting victims. Such attacks could be targeted phishing attempts [68] or support social engineering efforts to spread malware (e.g., a message such as "Hello Mr. Smith, we have discovered that your computer is infected. Please download and install this file." might be displayed to Mr. Smith). In addition, many people in political or corporate environments use social networks for professional communication (e.g., LinkedIn). Identifying these "high value" targets might be advantageous for the operator of a malicious website, revealing sensitive information about these individuals. For example, a politician or business operator might find it interesting to identify and de-anonymize any (business) competitors checking her website. Furthermore, our attack is a huge privacy breach: any website can determine the identity of a visitor, even if the victim uses techniques such as onion routing [45] to access the website – the browser nevertheless keeps the visited websites in the browsing history.

Of course, analogous to the situation where attackers compromise and abusing legitimate websites for drive-by downloads, the de-anonymization technique presented in this work can be used in a large-scale setup. That is, an attacker could abuse several compromised (but otherwise legitimate) websites as a vehicle for a de-anonymization attack.

2.3 De-Anonymization Attacks

With the background concepts introduced in the previous section, we are now ready to present our attack in more detail. We first introduce a basic variation of the attack, which is not feasible in practice. We then show how this basic approach can be refined to work for real-world social networks. Finally, we discuss how group membership information, a critical component for the advanced attack, can be obtained with a reasonable (limited) amount of resources.

2.3.1 Basic Attack

As mentioned in the previous section, certain dynamic hyperlinks contain explicit information about individual groups $g \in G$ and users $v \in V$ within a given social network S. An attacker can take advantage of this fact by using history stealing to probe for URLs that encode user information. In particular, the attacker can probe for a URL ϕ that contains an identifier of user v. When a link is found that contains this identifier for v, then the attacker can reasonable assume that the browser was used by v in the past to access the user-specific URL ϕ.

To find a suitable URL ϕ, an attacker would first perform an information gathering step and join the target social network. In particular, he would analyze the website and look for dynamic hyperlinks that *(a)* contain identifiers that are indicative for a specific personal profile (e.g., because they contain a user ID) and *(b)* easy to predict for arbitrary users. For example, the second link in Figure 2.1 satisfies these properties: The user IDs are numerical and, hence, easy to predict. Also, the link is indicative for a specific user because the corresponding web application command (i.e., modifying the profile image) can only be performed by the owner of the profile. Thus, it is very unlikely that a user other than v has this link in her history.

Of course, the basic attack is not feasible in practice. The reason is that the attacker has to generate and check one URL for every user in the social network, and each potential victim's browser would have to download all links and process them. In the case of Facebook, this would mean that more than 300 million links would have to be transferred to each victim. Thus, using the basic attack technique, the size of the search space (the candidate set) is far too large to be practical. In the following paragraphs, we show how group information can be used to to significantly reduce the search space. Moreover, we need to keep in mind that the basic attack is still a valuable tool to identify a specific user among a (small) group of possible candidates.

2.3.2 Improved Attack

For our improved attack, we leverage the fact that many social network users are members in groups. Social networks commonly provide special features for groups in order to facilitate communication and interaction between group members. Often, discussion forums or mailing lists are provided. Since these features are incorporated into the social network's web application, they are also prone to the history stealing technique. Similar to per-member actions, dynamic hyperlinks are used to incorporate group features into the web application. The main difference is that the identifiers in these links are not related to individual users within the group, but to the group itself. For example, the URLs (3) and (4) in Figure 2.1 are used for opening the group forums for two social networks.

An improved attack that leverages group membership information proceeds in two steps: First, the attacker needs to obtain group membership information from the social network. That is, the attacker has to learn, for some (possibly many) groups, who the members of these groups are. This step will be discussed in detail in the next section.

In the second step, the attacker uses history stealing to check the victim's browser for URLs that indicate that this user has recently accessed a page related to group g, and hence, is likely a member of g. By preparing URLs for a set of n groups, the attacker can learn a partial group fingerprint of the victim $\Gamma'(v)$. More precisely, the attacker can learn the entry $\Gamma_k(v)$ for each group k that is checked. The remaining entries are undefined. Clearly, being able to check more groups allows the attacker to learn more about the group fingerprint of a victim (i.e., he can obtain a larger, partial group fingerprint). This increases the chances that at least one entry of the partial group fingerprint is non-zero, which is necessary to carry on with the attack.

Once the partial group fingerprint of a victim is obtained, the attacker checks for the presence of entries where $\Gamma_k(v) = 1$. Whenever such an entry is found, we assume that the victim v is member of the corresponding group k. At this point, the attack can continue in one of two ways.

A slower, but more robust, approach is to leverage the group membership information and generate a candidate set C that contains the *union* of all members $\{u\}_k$ in those groups k for which $\Gamma_k(v) = 1$. That is, $C = \cup\{u\}_k : \Gamma_k(v) = 1$. Then, we use the basic attack for each element c in the candidate set C. More precisely, we use the basic attack to determine whether the victim v is one of the users $c \in C$. If so, then the attack was successful, and the user is successfully de-anonymized.

A faster, but more fragile, approach is to leverage the group membership information and generate a candidate set C that contains the *intersection* of all

members $\{u\}_k$ in those groups k for which $\Gamma_k(v) = 1$. That is, $C = \cap\{u\}_k : \Gamma_k(v) = 1$. Again, the basic attack is used to check for each user c in the candidate set C. Since the second technique uses set intersection instead of set union, it produces much smaller candidate sets and thus, it is faster.

Robustness. To see why the first attack is more robust than the second, we have to realize that the information that the attacker learns might be not entirely accurate. There are two reasons for this: First, the browsing history may contain incomplete information about the victim's past group activity (e.g., a user might have deleted the browsing history at some point in the past). Second, the group membership information that the attacker has collected "degrades" over time, deviating increasingly from the real group and membership configuration as users join and leave groups.

As a result of inaccuracies, some entries $\Gamma_k(v)$ in the partial group fingerprint might be wrong. Two cases need to be distinguished. The first case is that the entry $\Gamma_k(v)$ for a group k is 0 (or undefined), although v is a member of k. In general, this is not a problem, as long as the attacker finds at least one group k that the victim belongs to (and $\Gamma_k(v) = 1$). The reason is the following. Since the entry for k is zero, the first attack will not add the members of k (including v) to the candidate set C. However, we assume that there is another group that contains v. This means that v will be added to C, and the attack succeeds. For the second attack, the candidate set C can only shrink when a new group is considered (since set intersection is used). Thus, the attacker might need to check a larger candidate set, but he will still find v eventually.

The second case describes the situation where the entry $\Gamma_k(v)$ for a group k is 1, although v is *not* a member of k. This causes no problem for the first attack, which simply adds additional users (all members from group k) to the candidate set C. However, it is a problem for the second technique. The reason is that the intersection operation now includes a group that does not contain the victim user v. As a result, v will not be a part of the candidate set C, and hence, the attack will fail to find the victim.

In practice, an attacker would first attempt to use the fast (but fragile) approach based on set intersection. Only if this fails, one fall-backs onto the slower, more robust approach based on set union.

2.3.3 Efficiently Obtaining Group Information

To carry out the advanced attack, the adversary requires information about groups and group memberships. In this section, we demonstrate how an attacker can obtain this knowledge with relatively little effort.

The number of groups is typically considerably smaller compared to the

number of users. Nevertheless, collecting information about all groups and the members of each group is a challenging task. Therefore, we now discuss two techniques to efficiently obtain information about groups: *group directories* and *group member crawling*.

Group Directory

Typically, groups within social networks aim at attracting members that share a common interest with the group. To this end, social networks either offer a search functionality to find groups with a specific keyword, or they publish a list of the existing groups, called a *group directory*, via their website. This directory can be listed and searched by members of the social network to find groups related to their interests.

In our attack, it is desirable for the attacker to have knowledge on as many groups as possible. More specifically, the attacker is interested in the group identifiers to construct the hyperlinks for the history stealing attack. An attacker can use standard web crawling techniques to download the group directory, and then extract the group IDs from the web page's source code. Several social networks even allow the group directory to be viewed by non-members, which enables an attacker to use commercial crawling services for this task (see Section 2.4.3 for details).

Directory Reconstruction Some social networks do not publish a group directory or only do so partially (i.e., not all information about groups can be accessed this way). We implemented three methods to successfully circumvent this obstacle in practice.

First, the group identifiers that we observed in our experiments were either systematic (for example, numerical) or names. If group IDs can be guessed by an attacker, the group directory can be reconstructed by simply iterating over all (or at least a large fraction of) the ID space. The presence of the individual groups can be verified by trying to access each group's web page. In Section 2.5, we show that this brute-force technique can be used in practice effectively with a relatively small effort.

Second, an attacker can use the built-in search functionality of social networking websites to expose the group directory by listing all groups within a specific category of groups. Group search results are usually ranked by member size, which means that even if the result size is limited to a fixed value, an attacker gains valuable information.

Finally, we found that social networks may provide special "public" member profiles that can be accessed by non-members (i.e., they serve as "virtual"

business cards). For privacy reasons, these profiles usually contain less personal information than the original member profiles. However, these public profiles typically reveal the groups for a specific member. In this case, an attacker can reconstruct the group directory (including the group members) by crawling the public member profiles. Note that this technique is rather costly, since it requires to crawl member profiles.

Group Member Crawling

In addition to group IDs, an attacker needs to obtain the IDs of the group members for a significant amount of groups to successfully de-anonymize group members. This step can also be automated and performed on a large-scale, as we discuss below.

If we deal with a public group, the easiest case is that the social network allows all members of this group to be listed. Then, we can use a standard crawling approach to discover the group members and use them for our attack. As we show in Section 2.5, even tens of millions of group members can be crawled with only a limited amount of resources.

Some social networks limit the size of the member list that can be browsed. For example, Facebook only returns the first 6,000 members of a group. Hence, this limits a crawling attempt to only fully discover groups with up to 6,000 members. While this is still useful in practice, clearly, we would like to also be able to crawl larger groups.

In order to overcome this limitation, we take advantage of the fact that social networks typically allow searching within groups for members. This limits the amount of members returned per search, but we can efficiently extract most group members by searching for common first or last names. We use publicly available data from the U.S. Census Bureau [114] to determine common names, and then utilize this information to search within large groups to extract their members.

If we are dealing with a closed group, we cannot easily access the membership information for this group since only members can access this information. Hence, we send a request to join the group from a legitimate user account by using a script (i.e., "I would like to become member of this group"). If our application is accepted, we leave the group after we have collected membership information. Surprisingly, such a simple automated demand is successful in practice as we show in Section 2.5.

Note that, depending on the resources of an attacker, the member crawling may optionally be performed on-the-fly instead of offline before the actual attack. In an online setting, the attacker would crawl the groups a victim is a member of on demand, and then use the collected information for performing

the second round of history stealing (i.e., verification step). From a conceptual point of view, both attacks are similar. They just vary in the amount of resources needed.

2.4 Crawling Experiments

In this section, we describe our empirical experiments to extract group information from social networks and present the results we obtained for three social networks.

2.4.1 Ethical and Legal Considerations

Crawling data in social networks is an ethically sensitive area. Clearly, one question that arises is if it is ethically acceptable and justifiable to conduct crawling experiments in social networks. Similar to the experiments conducted by Jakobsson et al. in [69, 70], we believe that realistic experiments are the only way to reliably estimate success rates of attacks in the real-world.

First, in the crawling experiments we conducted, we only accessed user information that was publicly available. Second, note that the crawler we wrote was not powerful enough to influence the performance of any social network we investigated. Third, the commercial crawling services we used had misuse protection mechanisms like bandwidth throttling in place that prevented them from launching denial of service-like attacks against the websites that they were crawling (i.e., because of a higher crawling load).

We also consulted the legal department of our university (comparable to the IRB in the US), and we were informed that our experiments are approved.

2.4.2 Overview

For our experiments, we performed an in-depth analysis of the Xing platform. Furthermore, we carried out feasibility studies for Facebook [2] and LinkedIn [3].

We chose these networks as they are representative of the different categories of popular social networks. For example, Facebook aims at an audience that would like to maintain and create friendships, whereas LinkedIn and Xing are more focused towards business users who would like to maintain and extend their professional networks. Furthermore, each network has a unique way of representing its member and group directories.

Because of resource limitation, and because we had access to more Xing users for real-world user experiments, we chose to perform an in-depth analysis

of Xing (Xing's size is considerably smaller than Facebook or LinkedIn, but it still has more than eight million users).

In the following, we discuss how an attacker can automatically extract group information (i.e., which is a prerequisite for the de-anonymization attack) from each of these social networks.

2.4.3 Social Network Crawling Approaches

In order to access group information, an attacker can either run crawlers on her machines, or use third-party crawling services. For our experimental evaluation, we followed both approaches by implementing a custom web crawler, and by using commercial crawling services.

Custom Crawler

We implemented a web crawler that works by following the hyperlinks on a given starting public web page and then continues to download the HTML source code of each hyperlinked web page. To be able to also access parts of the social network that are only restricted to members, we added features that allow the crawler to login using provided member credentials. To this end, we manually registered three members to the social network using valid registration data (e.g., e-mail for our lab, etc.).

To extract the desired data, the crawler matches a set of regular expressions against the HTML source code. The extracted data (for example, group IDs and group names) are then stored in a database. To speed up the crawling process, we ran up to four instances of our crawler simultaneously.

Anti-Crawling Techniques Most social networks employ anti-crawling techniques to protect the data of their members. Typically, if a member behaves suspiciously (for example, if he tries to access an overly large amount of user profiles in a short amount of time), this member's account will be temporarily, or permanently disabled. In contrast, no similar restrictions are in place for group directories. We believe that this mainly has two reasons:

1. The content is regarded as being public, and not security-relevant.
2. As a means of promoting groups, it should be intentionally easy to access the directory.

In addition, we observed that group directories often contain additional information that is relevant for an attacker (e.g., group size, or additional meta data). In our scenario, an attacker benefits from these factors, as it allows her to prepare the history stealing attack with relatively little effort.

Commercial Crawling Services

Attackers with limited computing or network resources might resort to commercial crawling services instead of operating their own web crawler. Such services allow customers to specify which websites to visit. Typically, the crawling service might accept a given list of web pages and regular expressions. Such a service can be very cost effective. For example, services such as 80legs [34] charge as low as $0.25 per one million crawled URLs. In our experiments, we used such a service provider to perform some of our experiments.

2.4.4 Crawling Experiments

We applied the two different crawling strategies to the three social networks Xing, Facebook and LinkedIn. In the following, we elaborate on how the group directories for each network can be retrieved, and provide an overview of the results.

Xing

We mainly concentrated our crawling experiments on this network, as its smaller size allowed us to fully crawl its public groups. This network is being actively policed by administrators who, for example, quickly remove empty, or inactive groups.

By directing our custom crawler to Xing, we could download the data of 6,574 groups containing more than 1.8 million unique members. Xing claims to have about 8 million members in total (i.e., including members that do not use groups at all). Hence, the users in these groups represent a substantial fraction of the entire social network.

Closed Groups On Xing, the largest groups are *public*. That is, there are no restrictions on who is allowed to join these groups. On the other hand, an attacker might also be interested in crawling closed groups (that require manual approval by a moderator) in order to increase the effectiveness of the de-anonymization attack. To test how restrictive these groups are, we sent automated member requests from a legitimate account to the groups that had a large number of members.

We automatically applied for membership in 1,306 groups, and were accepted to 108 groups (8.2%). This allowed us, as an attacker, to see the user IDs of a total of 404,331 group members. Note that membership was denied by 1,199 groups (91.8%). However, despite the high rejection rate, we believe

that our test demonstrates that a malicious user can successfully launch automated social engineering attacks to become member of closed groups. In practice, a real attacker would probably use fake fotos, detailed fake information, and a corresponding application text to increase her success rate. In our experiments, we simply asked if we could become member of the group.

Interestingly, our success rate was higher for the larger (i.e., more important from the attacker's point of view) groups. We were often instantly added to the group without receiving any feedback. Hence, membership application seems to be a formality for many large, closed groups.

Facebook

Recovering the group directory for Facebook initially appeared straightforward. The network publishes a group directory on its website. Access to the directory is not restricted. As a result, everyone can download it from the web. The directory itself is organized in a multi-level hierarchical collection of alphabetically ordered lists that provide pointers to individual web pages to make it convenient for a human to navigate.

Due to the large size of the dictionary, we decided to use a commercial crawling service to download it. In total, the dictionary consisted of 7.1GB of HTML data in about 7,4 million files that contain 39,156,580 group IDs. The crawling service cost us $18.47 and we received the data after five days.

To enumerate Facebook's group members, we extracted the group IDs from the group directory, and then used our custom crawler to enumerate the members for each group. Facebook permits each logged-in user to search within the member lists of arbitrary groups. This search can be used to filter the member lists to only show members whose first and/or last name fully matches the search token. Using an empty string as the search token returns random sample of the group members. The size of each search result is limited to 6,000 members, and can be retrieved in chunks of 100 members per HTTP request.

Using Member Search Functionalities Since most Facebook groups have less than 6,000 members, this threshold is high enough and often allows us to obtain full member lists. An attacker can additionally use the search functionality to enumerate members in larger groups by searching for common first or last names. For example, with only 757 first names (233 male, 524 female), an attacker would be able to cover a cumulative percentage of more than 75% for each gender. According to the public 1990 US census [114] statistics, the most common first name, "James", has a 3.318% probability among males, and 1.635% among the overall population. Hence, for groups with about 367,000 members, an attacker could obtain all members with this name (i.e., the search

returns about 6,000 members for each name) on average. An attacker could even refine this approach by computing a more accurate name distribution by downloading Facebook's own (also publicly available) member directory.

Note that enumerating very large groups that contain millions of members only provides a limited benefit for an attacker. Apart from the high crawling effort, the time required to check for so many members via history stealing would defeat the purpose of using groups as a means of search space reduction in a realistic attack scenario (e.g., see throughput rates in Section 2.5.3). However, an attacker can also use the group member search functionality to verify the membership of individual members.

Results In total, we successfully crawled more than 43.2 million group members from 31,853 groups in a period of 23 days using only two machines. While this is still only a fraction of the overall Facebook groups and members, it demonstrates the feasibility of the approach.

In general, we believe that an attacker could also use a malicious botnet in real-life, or crawl for a longer period of time, and collect significantly more data compared to our effort with only limited resources.

LinkedIn

Just like Xing, LinkedIn focuses on business users. LinkedIn is a popular service and is widely-known.

Third-Party Crawling Use-Cases LinkedIn does not publish a full group directory, but provides a group search functionality for logged-in users. Theoretically, this functionality could be exploited in a similar fashion to the group member search functionality of Facebook. However, this requires a much larger effort due to the higher variation in possible group names as opposed to first or last names of individuals.

LinkedIn uses easy to predict group IDs. Specifically, LinkedIn uses numerical group IDs that range from 0 (oldest) to about 3,000,000 (the newest groups). In addition, the group ID space seems to be sparsely populated, as according to the network itself, it currently has only 442,434 registered groups.

In a two-phase crawling scenario, we first started a crawling pass with a commercial service [34]. In a preparation step, we first generated three million hyperlinks for the observed group ID space, and then used these links as "seed" for the commercial crawling service. The results of the crawling service can be used to identify which group IDs actually exist (i.e., a group profile page is returned if the ID exists). The cost for this experiment was $7.49.

After probing for the existing groups, we performed a second crawling run to retrieve additional information for each group such as its group size and group description. As this operation requires a logged-in user, we had to use our custom crawler.

While LinkedIn restricts access to its group directory, we found this not to be the case for its public member directory. For privacy reasons, the public member profiles are much less detailed than the regular profiles within the social network. Surprisingly, though, they do contain the membership status and group IDs for all groups that a member has joined. Clearly, data on groups and group memberships does not seem to be regarded as being security-relevant. As the public member profiles can be freely accessed over the web, an attacker can use automated legal third-party services to fully "outsource" the information gathering phase of the de-anonymization attack.

In a different crawling scenario, we performed an experiment and used an external crawling service to crawl the public profiles of three million members that we randomly picked from LinkedIn's member directory. The costs for the crawling were $6.57. Assuming a linear cost model, we estimate overall costs of about $88 for crawling all 40 million public profiles. This small investment would allow an attacker to target all group members of LinkedIn in a de-anonymization attack.

Other Social Networks

In order to find out how generic the problem of group information disclosure is, we manually analyzed five additional popular social networks that share features with the three networks that we analyzed in more detail.

Table 2.2 shows the features that are related to our de-anonymization scenario for these networks. All networks are vulnerable to history stealing and de-anonymization via groups. While we did not conduct crawling experiments for these networks, we expect the results and techniques to be similar to the ones we described in this section. Our empirical results demonstrate that group memberships are generally not considered as being privacy-relevant in many social networks.

2.5 Evaluation

In this section, we evaluate the practical feasibility of the de-anonymization attack.

	Facebook	MySpace	Friendster	LinkedIn	StudiVZ	Xing	Bigadda	Kiwibox
Uses dynamic links	✔	✔	✔	✔	✔	✔	✔	✔
Group directory	Full	Searchable	Full	Searchable	Searchable	Searchable	Searchable	Full
Member directory	Full	Searchable	Full	Full	Searchable	Searchable	Searchable	Searchable
Group member enumeration	≤6,000	Unlimited	Unlimited	≤500	Unlimited	Unlimited	Unlimited	Unlimited
Public member profiles	✔	✔	✔	✔	✔	✔	✔	×
Vulnerable	✔	✔	✔	✔	✔	✔	✔	✔

Table 2.2: Vulnerability Comparison of Social Networks.

2.5.1 Analytical Results

To assess the effectiveness of our de-anonymization attack, we first perform an analytical analysis of Xing. For this social network, we have a comprehensive overview. More precisely, we have crawled all public and several closed groups, together with all member information for these groups. We start with an overview of the different parameters of that network related to our attack scenario.

In total, we collected membership information for more than 1.8 million unique users in 6,466 public and 108 closed groups. Based on our data, the average Xing user is a member of 3.49 groups. By using data from Xing's group directory, we found that for the whole network, the sum of group member sizes (i.e., the number of membership relations) for the 6,466 public groups totals to more than 5.7 million. In contrast, the 15,373 closed groups only contain about 4.4 million membership relations. Furthermore, the average size of public groups is 914 members, whereas it is only 296 members for closed groups. The closed groups that we crawled contain 404,548 unique users. However, of these users, 329,052 (81.34%) were already covered by the public groups in our dataset.

These figures indicate, that an attacker can already collect a substantial amount of information by only focussing on public groups. The practical impact of closed groups appears to be rather low, given the increased effort that is necessary for gaining membership data for closed groups.

As mentioned in Section 2.3, the improved de-anonymization attack requires that the history stealing step finds at least one group k that the victim is a member of (that is, $\Gamma_k v = 1$ for at least one k). Thus, when probing for an increasing number of groups, it is interesting to see how the number of unique users grows that appear in at least one of these groups. When this number grows quickly, the attacker has a good chance to get a "hit" after inspecting a small number of groups.

Of course, one also needs to consider the order in which the history stealing step should probe for group membership: clearly, the attacker wants to optimize the process such that he has seen each user at least once after as few attempts as possible. One approach is to perform a *greedy* search. That is, the attacker first probes the largest group, then the second largest group, and so on. The problem is that this order might not be optimal, since it does not take into account group membership overlaps. An alternative approach is to choose the groups by *information gain*, i.e., in each step, test the group which has most members *not* seen before. This might leads to a better probing procedure, since the attacker covers in each step the largest possible number of previously-unseen users.

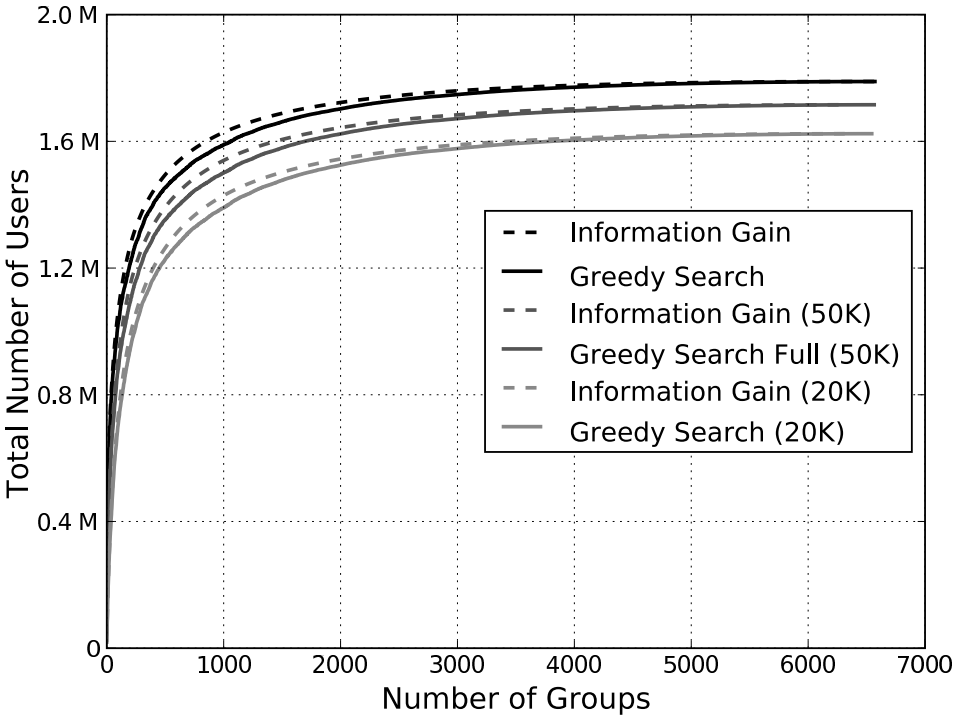


Figure 2.3: Cumulative distribution for number of unique users seen after crawling a specific number of groups (Xing).

Figure 2.3 shows the cumulative distribution for the number of unique users seen after crawling a specific number of groups. Besides considering all groups, we also studied the effect of limiting the search space to groups with at most 50,000 and 20,000 members, respectively. This takes into account that really large groups with hundreds of thousands of members are too large to probe efficiently, thus we restrict an attack to groups with an upper bound on the group size. The results indicate that an attack is very effective in practice: Even after testing only a few hundred groups, we have seen a significant percentage of all users at least once, i.e., a specific user is a member in at least one of the groups we have tested so far. In fact, we have seen more than 50% of the users after testing only 61 groups. After testing 1,108 groups, we have seen 90% of the users at least once. When restricting the search space, we can observe that we do not find each member at least once since some users are only member of large groups. Nevertheless, we can find more than 90% of the overall users when only considering groups smaller than 20,000 members.

The figure also shows that the difference between the greedy and the information strategy is small. More precisely, the overall shape is very similar, only the number of tested group is significantly different: with the information gain strategy, an attacker only needs to probe 6,277 groups until he has seen each user at least once, whereas the brute-force approach requires to test 6,571 of all 6,574 groups before the complete set of users (who are part of at least one group) is covered.

For our next analysis, we assume that the attacker has successfully launched a history stealing attack and computed an accurate group fingerprint $\Gamma(v)$ for a victim v. This allows the attacker to use the fast de-anonymization attack based on set intersection. To demonstrate the effectiveness of this attack, we show in Figure 2.4 the cumulative distribution of the candidate set sizes after set intersection. Each user in the candidate set needs to be inspected by the basic attack, thus, a smaller size if favorable for the attacker. Interestingly, for 42.06% of the users (= 753,357 users), the group fingerprint is *exact*, i.e., only a single user in the social network is a member of exactly these groups. These users can be uniquely identified just based on their group fingerprint, and no additional steps are required. For one million users, we can narrow down the candidate set to less than 32 users, and for 90% of all users, the candidate set is reduced from initially ∼1.8 million to less than 2,912 user. This highlights that one can dramatically narrow down the search space of candidates (who are then compared against the victim, one by one, using the basic attack).

Extracting a partial group fingerprint for a victim v might not always work flawlessly (for reasons discussed in Section 2.3). As a result, the fast de-anonymization attack based on set intersection could fail. In this case, the attacker needs to resort to the slower but more robust attack based on set

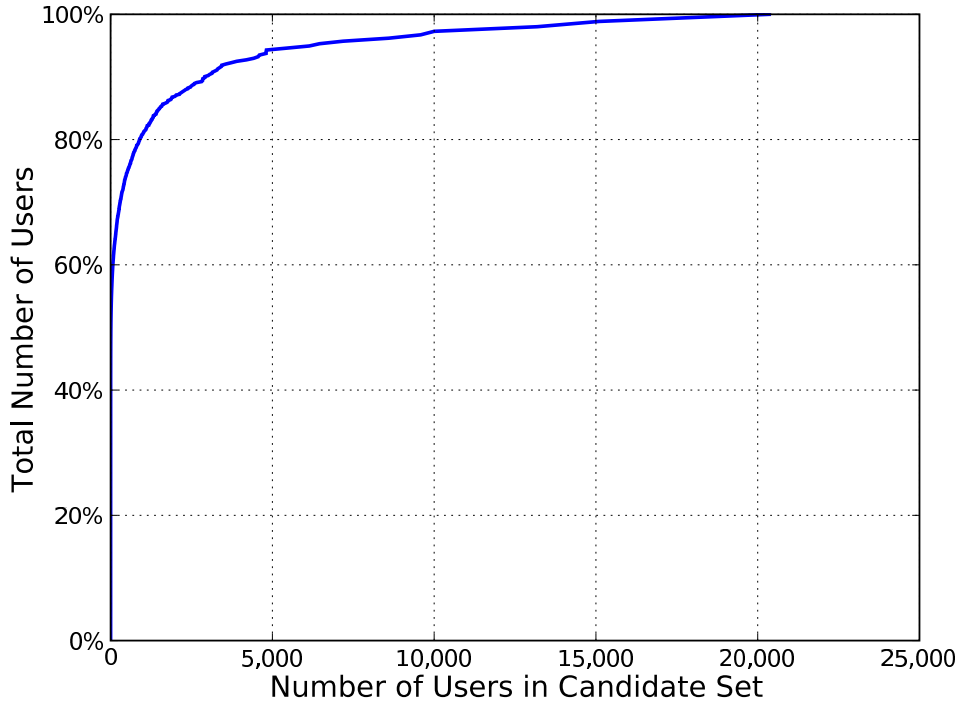


Figure 2.4: Cumulative distribution for the size of candidate sets (Xing).

union. In Figure 2.5, we show the cumulative distribution of the candidate set sizes when performing set union. Compared to the candidate sets from Figure 2.4, the union sets are considerably larger due to the fact that we merge each group for which we have a match. Second, there is still a significant reduction in size compared to the overall number of users in Xing for a larger fraction of victims. For example, the set union attack still reduces the size of candidate set to less than 50,000 for more than 72% of all users on Xing.

We performed the same kind of analysis for Facebook. Since we did not completely crawl this social network, the results in Figure 2.6 provide an overview for the snapshot of group information we obtained during our experiment. For the 43.2 million users we have seen in 31,853 groups, 9.9 million have an exact group fingerprint $\Gamma(v)$. Furthermore, for 25.2 million users, the size of the candidate set is smaller than 1,000. The distribution of users in a candidate set of a given size is different compared to the Xing experiment, but we expect that also for Facebook the shape of the cumulative distribution will be similar if an attacker has crawled the complete network.

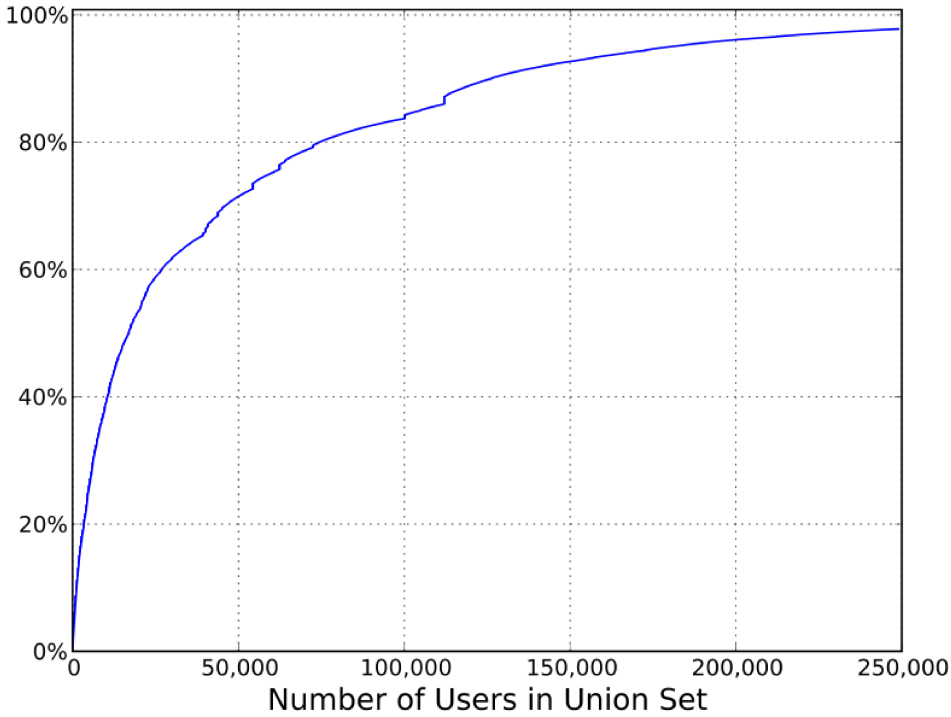


Figure 2.5: Cumulative distribution for the size of group union sets (Xing).

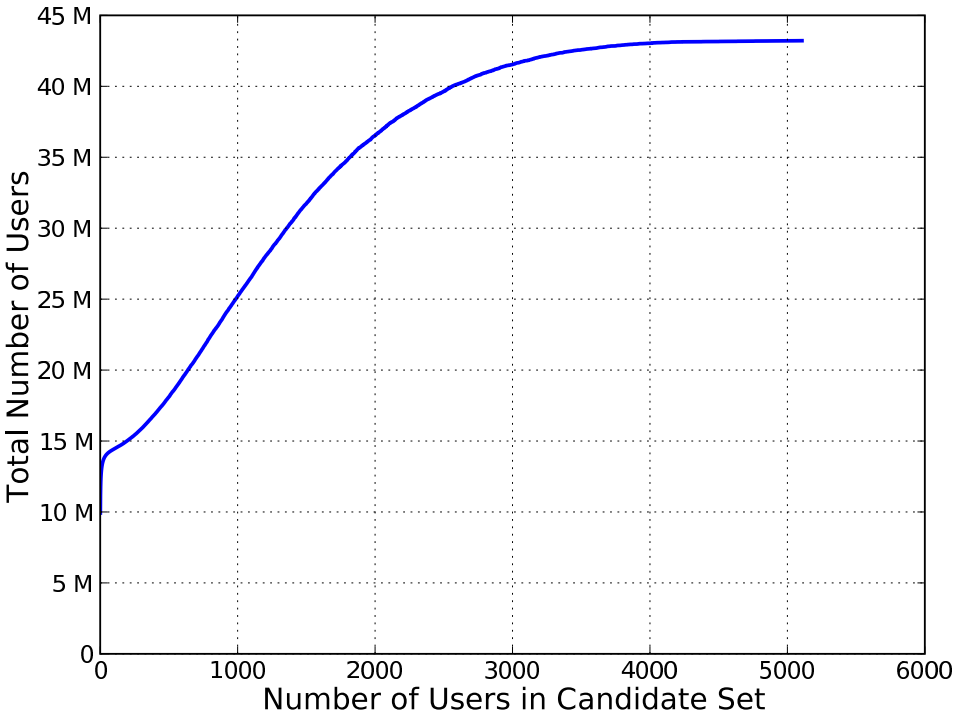


Figure 2.6: Cumulative distribution for the number of users seen in the candidate set (Facebook).

2.5.2 Real-World Experiments

To demonstrate the practical feasibility of our attack, we created a website that performs a de-anonymization attack against Xing. While there is no technical restriction that limits our attack scheme to a specific network, we chose to only implement it for Xing, as our crawling experiments had covered the highest fraction of groups and members for this network.

For ethical reasons, we only promoted this website to volunteers that we picked from our personal contacts. Before the actual de-anonymization process starts, a warning page explains the purpose of the experiment to visitors. Then, the visitors need to explicitly acknowledge that they would like to participate in the experiment. Once a visitor decides to take part, she also needs to fill out a brief questionnaire that asks her about how she uses the social network, if she shares her computer or user account with someone else, and how often she participates in social networking groups.

Once the experiment starts, the visitor's browsing history is probed for the URL of the main web page of Xing. If this URL is not present in the history, we assume that the visitor has not used Xing recently, or that she has disabled her browsing history (see the discussion in Section 2.6). In this case, we abort the de-anonymization attempt, and do not proceed.

If we do identify that a user has visited Xing before, we then use history stealing to probe for the URLs of the groups within Xing. That is, for each group, we check the visitor's browsing history for dynamic group links. We only expect these links to be present in the browsing history of users who are indeed members of these groups. We then perform the analysis based on the obtained group fingerprint, and present the result to the user.

Results In total, we launched our attack on 26 volunteers from our Xing contacts. For 11 of these visitors, we could not find any dynamic links that indicated interaction with groups in their browsing history. The reasons for this can be manifold, for example only seldom usage of groups or regularly flushing the browsing history.

For the remaining 15 visitors, we could determine group fingerprints, and successfully launched de-anonymization attacks. More precisely, we could leverage the faster group intersection variant of the attack, and could compute intersection sets for 11 visitors. The median size of these intersection sets was only 570 members, thus a search within this set can be performed easily. For 4 visitors, we had to fallback to the more robust, but slower union set-based variant of our attack. As expected, the union set is significantly larger compared to the intersection set: the median size was 30,013 members, which can nevertheless be probed during a history stealing attack.

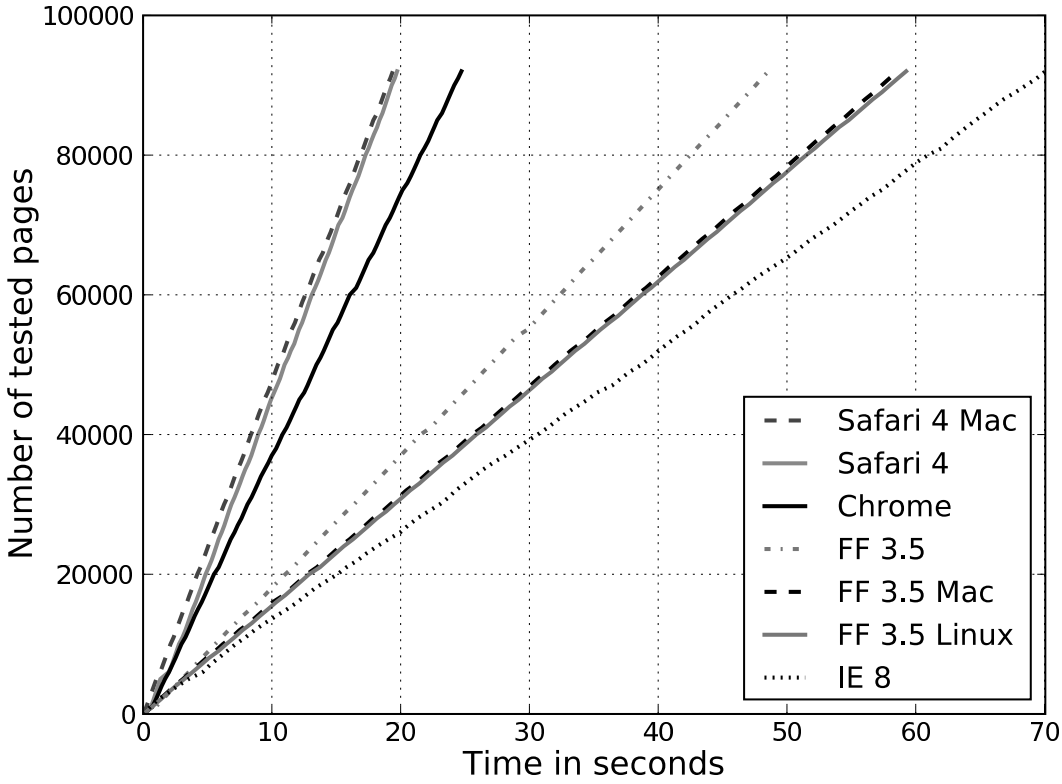


Figure 2.7: Runtime benchmark for different web browsers on different operating systems. The legend corresponds to the browsers from left to right.

In summary, our experiments with real users show that our attack works in practice. We managed to de-anonymize 15 of 26 users who participated in our experiment.

2.5.3 Run-Time and Throughput Rate

The runtime of the attack significantly influences its success rate in practice. Recall that we perform an active attack, and probe the victim's browsing history. Thus, the victim needs to interact with the attacker for a certain amount of time. There are many techniques to convince a victim to stay longer at the attacker's website. These techniques range from benign attempts such as showing a video or music clip, to offensive techniques such as "hijacking" the user's browser with the help of JavaScript to prevent her from leaving the site. In any case, from the point of view of the attacker, it is desirable that the attack takes as little time as possible.

We measured the typical time it takes to perform a history stealing attack in practice. We performed experiments with the four major web browsers (i.e., Internet Explorer, Firefox, Safari, Chrome) on different operating systems (i.e., Windows Vista, Ubuntu Linux, Mac OS X). The test machine was a MacBook Pro with a 2.8 GHz Intel Core 2 Duo processor and 4 GB RAM. We booted the operating system natively on the machine, and used no virtualization software to prevent side-effects. In each test case, we measured the time it takes to perform a history stealing attack by checking a specific number of URLs (i.e., we check if these URLs have been visited or not): That is, we started with 1,000 URLs, and we increased the number of URLs to be checked in steps of thousand until we reached 90,000. We performed each test ten times, and calculated the mean values. These values are shown in Figure 2.7. In order to increase readability, and since the mean error between individual runs was always below 5%, we omit error bars.

Safari on both Mac OS X and Windows achieved the best results in our experiments: A history stealing attack with 90,000 tests can be performed in less than 20 seconds. Chrome is about 25% slower, while Firefox requires between 48 and 59 seconds, depending on the operating system. The slowest performance was measured for Internet Explorer, which took 70 seconds to probe all pages. Nevertheless, even for Internet Explorer, we could probe more than 13,000 URLs in less than 10 seconds. Together with the results from Figure 2.3, this show that an attacker can detect many groups of a victim in a small amount of time.

2.5.4 Fluctuation in Groups

Another aspect we need to consider is the fluctuation rate in groups. From the viewpoint of an attacker, it is interesting to understand how the groups and members in a social network change over time.

First, it is unlikely that an attacker has access to the networking and computing capacity that would enable her to take a complete snapshot of a social network (i.e., permit her to collect the necessary data for a de-anonymization attack in a time span that is short enough to prevent *any* change in both groups and members). In practice, depending on the size of the network and the attacker's resources, the duration from start to end of crawling and the reconstruction of the groups directory might take days, or even weeks.

Second, there will also be changes that are caused by normal user behavior after the initial crawling phase. For example, members will join or leave groups, or new groups will be created. Over time, these changes cause the crawled data to increasingly deviate from the real configuration of groups and members in the network. Determining how stable the group data is, is

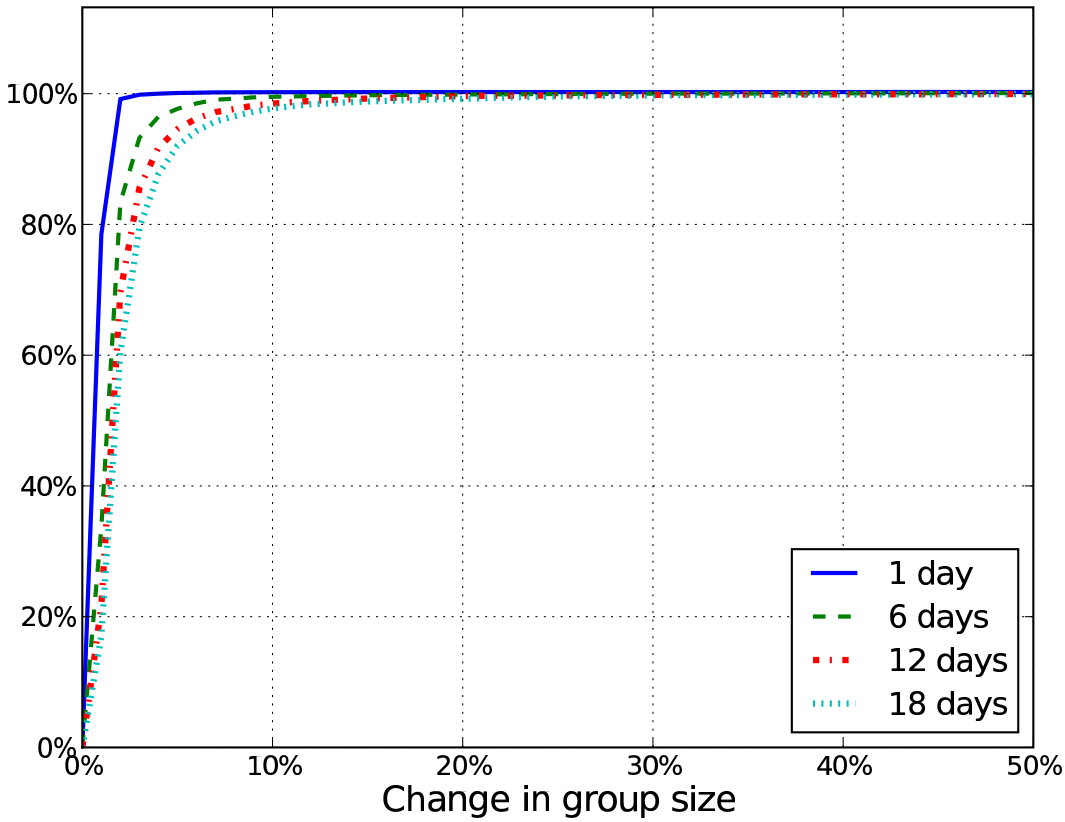


Figure 2.8: Degradation of group data in Xing.

related to the question of how often an attacker would have to recrawl parts or the entire social network. Hence, this directly influences how much effort an attacker has to invest for a de-anonymization attack. An attacker can also develop iterative approaches for keeping the collected information up-to-date, e.g. social networking features that explicitly allow the listing of only new members in groups and newly created groups can be used.

For measuring the fluctuation in groups, we conducted experiments for Xing. Instead of repeatedly crawling the entire network, we only downloaded the group directory and the member size for each group. This permitted us to repeat this operation every four hours over a period of 18 days.

Figure 2.8 shows the CDF for the changes in group size for four different periods. Interestingly, while the results from our measurements confirm that the quality of the collected group and member data degrades over time, they also show that the data stays relatively stable, significantly reducing the necessary crawling effort for an attacker.

While we are aware of the possibility that the amount of users that either

join or leave a group might lead to the same overall group size, we believe that the result of our experiment is still accurate enough to give an indication of the amount of change that affects the Xing's group configurations.

2.6 Possible Mitigation Techniques

The approach presented in this work allow a malicious user to launch de-anonymization attacks against a large number of victims with relatively little effort. Whereas history stealing by itself is often not enough to identify individual users, combined with the misuse of group membership information stored in social networks, it becomes a critical weakness. In this section, we list mitigation techniques that aim to thwart our attack.

2.6.1 Server-side Mitigation

As a server-side mitigation, web applications could use dynamic hyperlinks that an attacker cannot easily predict. For example, existing systems could be hardened against the history stealing attack by automatically adding HTTP GET parameters that contain random tokens to all hyperlinks. Depending on the utilized web server, it might be possible to retrofit existing web applications by using URL rewriting to automatically add such tokens to each URL.

Even adding a simple, alphanumerical string of length 2 would increase the attacker's search space by a factor of 3844 (62^2). Hence, the attack would effectively be prevented.

Also, web applications should preferably use HTTP POST instead of HTTP GET in order to send parameters. This is because only GET parameters are stored in the browsing history. In fact, a server-side mitigation solution that randomizes web-application links is presented in existing work [71].

Note that one difficulty with server-side mitigation is that the usability of the web applications may be affected. For example, it may become more difficult to bookmark parts of the application, or links to certain groups may become more difficult to remember.

2.6.2 Client-side Mitigation

On the client-side, history stealing is more difficult to fix without sacrificing functionality. Obviously, the goal is to prevent browsers from leaking sensitive and private information via style information. As a solution, browsers could generally restrict client-side scripts from accessing the CSS properties of hyperlinks. Unfortunately, this could also break existing websites that legitimately

do so.

In [67], the authors offer a clever solution by extending the *same-origin* concept of web browsers to visited links. Unfortunately, so far, none of the published countermeasures to history sniffing have experienced wide-spread adoption, whether on the server, nor on the client-side.

Current web browsers only provide limited options for protection against attacks that are based on history stealing. Because the attack can be implemented without the need for client-side scripting, turning off JavaScript, or using browser add-ons that protect against script-based attacks (for example, NoScript [84]) may only provide limited help.

Users can also permanently, or temporarily disable the browsing history. They can, for example, use the "private browsing modes" that are supported by several current browsers (e.g., Firefox, Safari). Unfortunately, all of these methods also require some effort on behalf of the user, and reduce the usability of web browsers and web applications.

2.7 Summary

In this chapter, we introduced a novel de-anonymization scenario against social networking websites. We have shown, theoretically and experimentally, that an attacker can crawl social networking sites and use publicly available group membership data to de-anonymize large amounts of members of social networks with relatively low effort. Clearly, as social networking websites store large amounts of sensitive data for millions of users, this attack is a serious security threat,

In the next chapter of this thesis, we are going to widen the scope of our observations. In contrast to analyzing the privacy issues of a single type of website, we are going to examine a more complex system, a whole branch of the online business. Specifically, we are performing a case study on the online adult industry, as it is widely assumed that many websites in this industry are privacy threatening. We perform a technical and economical analysis to show the industry's privacy and security issues, as well as its relationship with the underground economy.

Chapter 3

Privacy Threats in Online Services

Typically, popular online services (such as news websites, video and media sharing websites, online shops) attract large amounts of visitors. As many users are not aware of the potential security and privacy risks, this also creates an incentive for cyber criminals to participate in online services themselves. In this chapter, in contrast to "direct" privacy threats against websites (as shown in the previous chapter for social networks), we are interested in examining a whole branch of online business in more detail.

As a case study, we have chosen the online adult industry, as it is among the most profitable business branches on the Internet, its participants clearly are interested in protecting their privacy, and its web sites attract large amounts of visitors and traffic. Nevertheless, no study has yet characterized the industry's economical and security-related structure. As cyber-criminals are motivated by financial incentives, a deeper understanding and identification of the economic actors and interdependencies in the online adult business is important for analyzing security-related aspects of this industry.

This knowledge can be used to answer the following questions: (1) Which economic roles exist in the online adult industry? (2) Is there a connection between the online adult industry and cyber-crime? (3) What specific threats target visitors of adult web sites? (4) Is there domain-specific malicious activity originating from adult web sites? To answer these questions, we applied a combination of automatic and manual analysis techniques to investigate the economic structure of the online adult industry and its business cases. To gain additional insider information that can be used for a security assessment, we also created and operated two adult web sites. In this study, we provide a survey of the different economic roles that adult web sites assume, and highlight their economic and technical features. We provide insights into security flaws and potential points of interest for cyber-criminals. Furthermore, we also performed several experiments to gain a better understanding of the flow of visitors to these sites and the related cash flow, and report on the lessons learned while operating adult web sites.

3.1 Introduction

"The Internet is for Porn" is the title of a satirical song that has been viewed several million times on YouTube. Its popularity indicates the common belief that consuming pornographic content via the Internet is part of the modern pop-culture. Compared to traditional media, the Internet provides fast, easy, and anonymous access to the desired content. That, in turn, results in a huge number of users accessing pornographic content. According to the Internet Pornography Statistics [65], 42,7% of all Internet users view pages with pornographic content. From the male portion of these users, 20% admittedly do it while at work.

With a total worth of more than 97 billion USD in 2006 [65], the Internet porn industry yields more revenue than the top technology companies *Microsoft, Google, Amazon, eBay, Yahoo!*, and *Apple* combined. Interestingly, however, to the best of our knowledge, no study has yet been published that analyzes the economical and technological structure of this huge industry from a security point of view. We aim at answering the following questions:

Which economic roles exist in the online adult industry? Our analysis shows that there is a broad array of economic roles that web sites in this industry can assume. Apart from the purpose of selling pornographic media over the Internet, there are much less obvious and visible business models in this industry, such as *traffic trading* web sites or *cliques* of business competitors who cooperate to increase their revenue. We identify the main economic roles of the adult industry and show the associated revenue models, organizational structures, technical features and interdependencies with other economic actors.

Is there a connection between the online adult industry and cybercrime? According to web statistics, adult web sites regularly rank among the top 50 visited web sites worldwide [6]. Anonymous and free access to pornographic media appeals to a huge audience, and attracts large amounts of Internet traffic. A global increase in available consumer bandwidth has made adult web sites even more popular over the past years. We show that this highly profitable business is an attractive target for cyber-criminals, who are mainly motivated by financial incentives [52, 64].

What specific threats target visitors of adult web sites? Common belief suggests that adult web sites tend to be more dangerous than other types of web sites, considering well-known web-security issues such as malware, or script based attacks. Our results verify this assumption, and in addition, we

show that many adult web sites use aggressive marketing and advertisement methods that range from "shady" to outright malicious. They include techniques that clearly aim at misleading web site visitors and deceiving business partners. We describe the techniques we identified, and their associated security risks.

Is there domain-specific malicious activity originating from adult web sites? To be able to assess the abuse potential of adult web sites, we describe how we created and operated two adult web sites. This enabled us to identify potential attack points, and participate in adult traffic trading. We conducted several experiments and performed a security analysis of data obtained from web site visitors, evaluating remote vulnerabilities of visitors and possible attack vectors. We also identified and experimentally verified scenarios involving fraud and mass infection that could be abused by adult site operators, showing that we could potentially exploit more than 20,000 visitors spending only about $160.

To summarize, we make the following contributions in this work:

1. We provide a detailed overview of the individual actors and roles within the online adult industry. This enables us to better understand the mechanisms with which visitors are redirected between the individual parties and how money flows between them.

2. We examine the security aspects of more than 250,000 adult pages and study, among other aspects, the prevalence of drive-by download attacks. In addition, we present domain-specific security threats such as disguised traffic redirection techniques, and survey the hosting infrastructure of adult sites.

3. By operating two adult web sites, we obtain a deeper understanding of the related abuse potential. We participate in *adult traffic trading*, and provide a detailed discussion of this unique aspect of adult web sites, including insights into the economical implications, and possible attack vectors that a malicious site operator could leverage. Furthermore, we experimentally show that a malicious site operator could benefit from domain-specific business practices that facilitate click-fraud and mass exploitation.

This chapter is structured as follows: Section 3.2 discusses the analysis techniques we used to study adult web sites. Section 3.3 presents the observations we made and insights we gained on the economic and technical structure of

adult sites. In Section 3.4, we describe how we mimic adult webmasters and operate two adult websites, allowing us to collect and evaluate security relevant insider information.

3.2 Analysis Techniques

In this section, we describe the experimental setup that we used to perform the analysis that allowed us to gain insights into the online adult industry. As part of this study, we first manually examined about 700 pornographic web sites. This allowed us to infer a basic model of the industry's economic system. In the second step, we created a system that crawls adult web sites and extracts information from them to automatically gather additional data.

3.2.1 Manual Inspection

Given the minimal amount of (academic) information currently available for this very specific type of Internet content, we basically had to start from scratch by projecting ourselves into a "consumer" role. By using traditional search engines, we located 700 distinct web sites related to adult content. This initial sample set provided the first insights into the general structure of adult web pages. For example, we observed that many web sites contain parts that implement similar functionality, such as preview sections and sign-up forms. In addition, we also looked for specialized services and web sites that appeal to "producers" of pornographic web sites. We used information gained from industry-specific business portals [124] to identify business-to-business web sites, such as adult hosting providers and web payment systems.

We identified several web site "archetypes" that represent the most important business roles present in the online adult industry. The majority of web sites that we analyzed fits into exactly one of these roles. The economic relationships between these entities are shown in Figure 3.1. Whenever suitable, we named the roles according to the industry jargon. In the following section, we provide a detailed overview of each role. Based on these observations, we then created an automated crawling and analysis system to gain a broader insight into the common characteristics of adult web pages, operating on a large sample set of about 270,000 URLs (on more than 35,000 domains).

3.2.2 Identified Site Categories

Based on our observations, we can classify the market participants in the following categories.

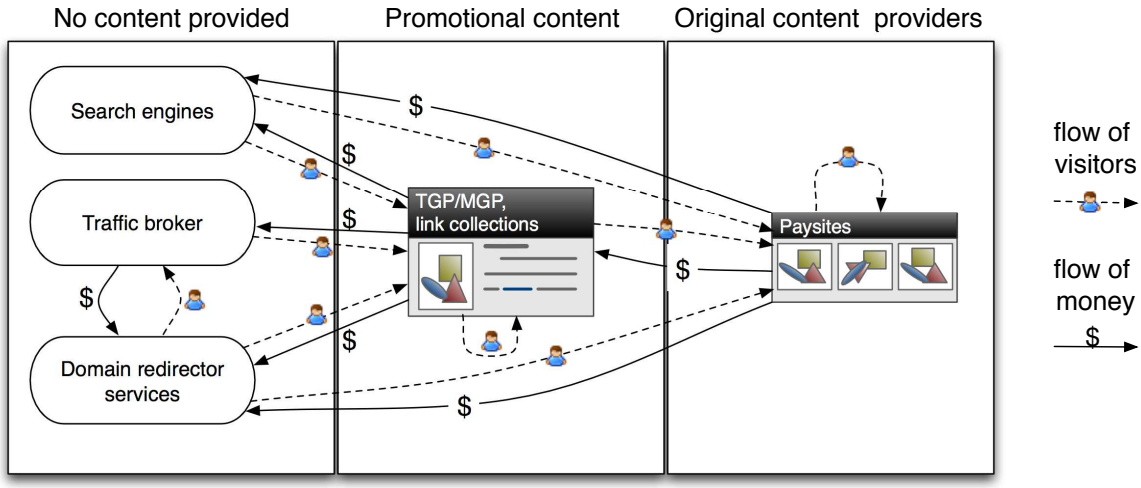


Figure 3.1: Observed traffic and money flows for different roles within the online adult industry.

Paysites

This type of web sites constitutes the economic core of the online adult industry. These web sites typically act as "content providers", producing and distributing pornographic media such as images and videos via their web pages, charging money in return. Most common users would consider these sites to be representative for this genre.

Link Collections, TGP / MGP

Complementary to paysites, a large number of pornographic web sites promise free content. These sites often call themselves *link collections*, *thumbnail gallery posts* (TGPs) or *movie gallery posts* (MGPs), depending on the provided form of pornographic media. We use the term *free site* to denote these types of web sites.

Link collections typically consist of a series of hyperlinks (often adding textual descriptions of the underlying media) to other web sites. TGP and MGP sites are structurally similar, with the addition of displaying miniature preview (still) images next to each link. It is indicative for free sites that they do not produce their own content. Our evaluation shows that they receive media from other content providers, as their main economic role is marketing for paysites. A secondary role is *traffic trading*, as it will be explained in Section 3.2.2.

Search Engines

With the multitude of different providers, specialized search engines evolved to fit the need of every potential customer. Functionally similar to general purpose search engines such as Google, adult search engines [59] allow users to search for web sites that match certain criteria or keywords. Unlike traditional search engines, adult search engines claim to manually classify the web sites in their index, instead of relying on heuristics or machine learning techniques. However, this claim - suggesting that their results are more accurate than other search engines - is highly questionable, considering the fact that pornographic pages account for 12% of the total number of web pages on the Internet [65]. Adult search engines generally generate revenue by displaying advertisements and selling higher-ranked search result positions.

Domain Redirector Services

Interestingly, there are services that specialize in managing adult *domain portfolios*. They are similar to commercial *domain parking* services that display web pages with advertisements (which are often targeted towards the domain name) in lieu of "real" content [118].

Adult domain redirector services such as [18] not only allow their clients to simply park their domains, but are rerouting any web traffic from their clients' domains to adult web sites. Adult sites that wish to receive traffic from the redirector service have to pay a fee for being registered as a possible redirection target. The exact destination of the redirections is typically based on the string edit distance between the domain name of the web site participating in the redirector service, and the domain name of the adult web sites which wish to receive traffic. For example, a user might browse to `www.freehex.com`, not knowing that this site participates in a redirector service. The user will then be redirected to an adult web site with a domain name that has a low edit distance to this domain name. The destination adult web site initially has to pay a fee for being considered by the redirection service, while the domain owner is rewarded for any traffic that originates from his domains. Technically, these redirector services work by using a layer of HTTP redirections, giving no indication to the user that a redirection has occurred.

From a miscreant's point of view, these redirector services appear to be an ideal tool for *typo-squatting* [118]. Typo-squatting is the practice of registering domain names that are syntactically very close to the names of legitimate web sites. The idea behind typo-squatting is to parasitize web traffic from users that want to go to the legitimate site, but make a typographical error while entering the URL.

Keyword-Based Redirectors

Several businesses offer a service that aims at increasing the visibility and (traditional) search engine ranking of their clients (adult web sites). To this end, keyword based redirector services operate websites that have a large numbers of subdomains. The names of these subdomains consist of combinations of adult-related search engine keywords.

Similar to domain redirector services, these subdomains are configured to redirect visitors to "matching" web sites, e.g. the redirector's clients. Clearly, this technique is an attempt to exploit ranking algorithms to achieve higher search result positions, effectively subverting the search engine's business model of selling search result positions. Furthermore, it is an efficient way to prepare a web site for spam advertisement. Unsolicited bulk (spam) mails tend to yield a higher penetration rate when embedded links differ from mail to mail [109].

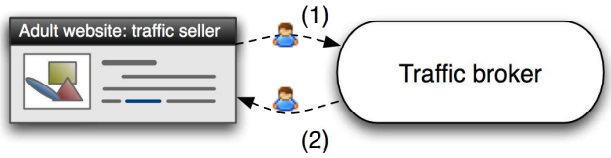
Traffic Brokers

This unique type of service provider allows its clients to directly *trade* adult web traffic for money, and vice versa (i.e., web traffic can be turned into real money with this kind of providers). Prospective clients who want to *buy* traffic can place orders (typically in multiples of 1,000 visitors) that will then be directed to a URL of their choice. Usually, the buyer can select the source of the web traffic according to several criteria, such as interest in certain niches of pornography or from specific countries. Available options also include traffic that originates from other adult sites, e-casinos, or from users who click on advertisements such as pop-up or pop-under windows, or even links in YouTube comments. Another option is traffic that is redirected from recently expired domains, which have been re-registered by the traffic broker.

On the other hand, clients who want to *sell* traffic can do so by redirecting their visitors to URLs that are specified by the traffic broker, receiving money in return. If the broker has no active orders from buyers for the type of traffic that is provided, the traffic is sent back to a link specified by the client. However, if the broker has an active order, the traffic is redirected to the site of the buyer's choice and the seller is credited a small amount of money. Figure 3.2 visualizes the flow of visitors and money for both scenarios.

Before a client can participate in traffic trading, brokers typically claim that they check the source or destination site of the traffic to prevent potential abuse. For example, many traffic brokers state that they do not tolerate hidden frames on target web sites. However, in our experiments with traffic brokers, we found this claim to be false: We successfully managed to buy large

(a) Traffic buyer is interested in receiving traffic and pays for it.



(b) No traffic buyer available, traffic broker returns visitor to a specific URL.

Figure 3.2: Schematic overview of traffic trading and the flow of visitors/money.

quantities of traffic for a web site that makes extensive use of hidden `iframes` and even performs vulnerability checks on its visitors (see Section 3.4 for more details).

3.2.3 Automated Crawling and Experimental Setup

To acquire real-world data and to perform a large-scale validation of the initial results from our manual analysis, we created a web crawler system. Based on our observations, we added several domain-specific features. Our system consists of the following components.

Search Engine Mining

For our crawling system, it was necessary to acquire a set of adult web sites that were suitable as initial input. To mimic the way a consumer would look for adult web sites, we made use of search engines. We manually compiled a set of domain-specific search queries and automatically fed it as input to a set of 13 search engines. This included three general purpose search engines (Google, Yahoo, and Microsoft Live) and ten adult search engines. We then automatically extracted the URLs from the search results and stored them in a database. The result set consisted of 95,423 URLs from 11,782 unique domains. These URLs were the seed used in the crawling step.

Crawling Component

The core component of our system is a custom web crawler we implemented for this purpose. We configured it to follow links up to a depth of three for each domain. For performance reasons, we additionally limited the maximum amount of URLs for a single domain to 500. Starting from the previously-mentioned seed, we crawled a total of 269,566 URLs belonging to 35,083 web sites. For each crawled URL, we stored the web page source code, and the embedded hyperlinks. This formed the data set for our subsequent analysis. In addition to the crawling, we used the following heuristics to further classify the content, and detect a number of features.

Enter Page Detection. A characteristic feature of many adult web sites (unrelated to their economic role) are "doorway" web pages that require visitors to click on an `Enter` link to access the main web site. These *enter pages* often contain warnings, terms of use, or reminders of legal requirements (for example, a required minimum age for accessing adult material).

In order to automatically detect enter pages, we used a set of 16 manually compiled regular expressions to scan textual descriptions of links. Since some enter pages use buttons instead of text-only descriptions, we also checked the HTML alternative text for images. For example, if a link description matches $.*enter\ here.*$ or $.*over.*years.*$, we classify the page as an enter page.

Adult Site Classifier. Since we wish to avoid crawling non-adult web sites, and since not all outgoing links lead to adult web sites, we created a simple, light-weight keyword-based classifier to identify adult web sites. To this end, we first check for the appearance of 45 manually selected, domain-specific keywords in the web site's HTML meta description tags. In case no matches are found, we also extend our scan to the HTML body of the web page. If at least two matches are encountered, we consider the web site to contain pornographic content.

According to our experience, this *naïve* classification works surprisingly well, as porn sites usually promote their content openly. To evaluate the true positive (TP) and false positive (FP) rate of our classifier, we ran it on a hand-labeled subset of 102 web sites that we chose randomly our manual-analysis test set. It achieved rates of 81.5% TP and 18.5% FP. Moreover, a limitation of our current implementation is that it currently only works with English-language web sites. After excluding non-English web sites, the rate improved to 90.1% TP and 9.9% FP. We are aware that far more advanced classifiers for adult sites exist, for example systems that include image recognition techniques [60]. However, these classifiers are typically aimed towards filtering

pornographic content and are not readily and freely available, and our current heuristic yields sufficiently accurate results for our purposes.

Client Honeypots

Malicious web sites are known to direct a multitude of different types of attacks against web surfers [102, 103, 117]. Examples include drive-by downloads, Flash-based browser attacks, or malformed PDF documents that exploit third-party software. To detect such attacks, we used two different client honeypots to check the web sites that we crawled in our study.

Capture-HPC. We used an adapted version of the Capture-HPC [112] *client honeypot*. The tool detects and records changes to the system's filesystem and registry by installing a special kernel driver. We set up Capture-HPC in virtual machines (VMs) with a fully patched Windows XP SP2, resembling a typical PC used for web browsing. We then instrumented the VMs to open the URLs from our crawling database using Internet Explorer 7 (including the popular Flash and Adobe PDF viewer plugins). This allowed us to detect malicious behavior triggered by (adult) web sites. In our experimental setup, we ran eight instances of the VMs in parallel, to achieve a higher throughput rate.

Wepawet. To complement the analysis performed by Capture-HPC, we used another client honeypot, namely Wepawet [82], in parallel. The software features special capabilities for detecting and analyzing Flash-based exploits, and for handling obfuscated JavaScript, which is commonly used to hide malicious code. Wepawet also tries to match identified code signatures against a database of known malware profiles, returning human-readable malware names.

Whois Mining

Our system also performs Whois lookups for each crawled domain. Then, the registrant (the entity that owns the domain name) and the registrar (the company that sold the domain) are determined for each entry. Since Whois records do not have a homogeneous data representation, we had to implement a parsing tool that can process Whois records. We subsequently use this information to determine if web sites belong to the same owner and also search for anomalies in this data set.

Economic Classification

To decide if paysites are more or less secure (i.e., trustworthy) than free sites, we created a heuristic for automatically classifying each web site depending on its economic role. Our classifier is limited to determining if a web site is either a paysite or a free site; otherwise, the web site's economic role remains undefined.

Paysite Indicators. We identity paysites based on manual observations and by using information we found on adult business-to-business web sites: we compiled a list of 96 adult payment processors, i.e., companies appointed by a web site operator to handle credit card transactions on behalf of him. If a web site links to a payment service provided by one of these processors, we immediately mark it as a paysite. In case no payment processor is found, we look for additional features of paysites. To this end, we match the web site source code against a set of regular expressions to determine if it contains a "tour", "member section", or membership sign-up form. We assume these structural features to be indicative for paysites, as we did not find any counter-examples in our manual observations.

Free Site Indicator. To identify free web sites, we examine their hyperlink topology. For this classification, we only regard *outgoing links* as a reliable feature, as it is not feasible to recover (all) incoming links for a web site. We analyze the number of hyperlinks pointing to different domains for each web site, and additionally compare the Whois entries for both the source and destination domains. If a web site exceeds a threshold t of links to "foreign" domains (e.g., the Whois entries show different registrants), we label it as a free site. To evaluate this classifier and instantiate a value for t, we tested it on a hand-labeled set of 384 link collection web sites that we selected randomly from our database. Based on this experiment, we chose $t = 25$ for the evaluation.

3.3 Observations and Insights

During our crawling experiments, we observed several characteristics of adult sites. In this section, we provide an overview of the most interesting findings, and discuss how they are security-relevant.

3.3.1 Revenue Model

Like with every economy, the ultimate goal for commercial web site operators is to earn a maximum of money, and the slogan "sex sells" is a clear testimony to this fact. In the following, we analyze the revenue model of the major categories identified in Section 3.2.2.

Paysites

We found the revenue model of paysites to be centered around selling *memberships* to customers. A membership grants the customer access to an otherwise restricted *member area* with username/password credentials. In the member area, an archive of pornographic media can be browsed or downloaded by the customer. Memberships typically have to be renewed periodically, causing recurring fees for the customer and, therefore, providing a steady cash-flow for the paysite. To appeal to customers and to create a stimulus for purchasing a membership, paysites rely heavily on a number of marketing and advertising techniques, like for example:

A "Tour" of the Web Site. Similar to traditional advertising methods (for example cinematic trailers for movies), preview media content is published for free on the paysites' web pages, eventually directing the user to membership sign-up forms.

Search Engines and Web Site Directories. Specialized promotion services, such as *adult search engines* and web site directories, allow users to submit hyperlinks to web sites. These links are then categorized (depending on the nature of the content), and made available on a web site where they can be searched and browsed. While these services are typically free of charge, higher ranked result positions can be purchased for a fee.

Affiliate Programs. The main purpose of an *affiliate program* is to attract more visitors to the paysite. The business rationale is that more visitors translates to more sales. To this end, paysites allow business partners to register as affiliates, thus giving them access to promotional media. This media is designated for marketing the paysite. It consists of hyperlinks pointing to the paysite and optionally includes a set of pornographic media files. In return for directing visitors to the paysite, affiliates are rewarded a fraction of the revenue that is generated by those customers that were referred by the affiliate.

By using affiliate programs, paysites are effectively shifting part of their marketing effort towards their affiliates. Additionally, those sites that dis-

tribute the media files (instead of just providing hyperlinks) can reduce their resource consumption (such as bandwidth costs) as an additional benefit. Many paysites even offer specialized services to their affiliates, for example, by providing preview images and textual descriptions of the content, or even creating administrative shell scripts. Also, Internet traffic statistics are made available to affiliates, so that they can optimize their marketing efforts.

Free Sites

Free sites typically participate in multiple affiliate programs. We found examples of sites participating in more than 100 different programs, generating revenue by directing visitors to paysites. To account for the origin of customer traffic, paysites usually identify their affiliates by unique tokens that are assigned on registration. These tokens are then used to associate traffic with affiliates, for example, by incorporating them as HTTP parameters in hyperlinks pointing from the affiliate site to the paysite. The same technique is used to identify links originating from spam mails, providing the site with the means to evaluate a spammers' advertising impact.
Often, affiliates can choose between two revenue system options:

- Pay-per-sign-up (PPS): The affiliate receives a one-time payment from the paysite for each paysite member that was referred by the free site.
- Recurring income: In contrast to PPS, the affiliate can choose to receive a fraction of each periodic fee as long as the membership lasts.

We found that the payment systems that are used to transfer money from paysites to affiliates offer a wide variety of options, including wire transaction, cheques, and virtual payment systems. In addition to affiliate programs, free sites display advertisements to increase their revenue.

3.3.2 Organizational Structure

Paysites We noticed that many paysites are organized in *paysite networks*. Such networks act as umbrella organizations, where each paysite contains hyperlinks to other members of its network. Additionally, networks often offer customers special membership "passes" that grant collective membership for multiple paysites.

Interestingly, however, upon inspection of the Whois [92] entries for member sites within several networks, we found the registration information to often match (e.g., the sites were belonging to the same owner). Apparently, the individual network members prefer to create the outward impression of

representing different enterprises, when they are in fact part of the same organization. This indicates that a diversification among paysites, depending on the sexual specifics of the offered content, is advantageous for the owners. These specialized sites are called *niche sites* in the industry jargon.

Free Sites Similar to paysite networks, we found free sites to be also organized in networks. However, in contrast to paysites, free sites also frequently link to each other even if the site owners differ. This means that business competitors are collaborating. This appears counter-intuitive at first. However, one has to take into account that cross-linking between free sites is a search engine optimization method. Thus, the search engine ranking of all sites participating in a "clique" of free sites improves, as the sites are artificially increasing their "importance" by creating a large number of hyperlinks pointing towards them.

3.3.3 Economic Roles

From a consumer perspective, paysites and free sites are the most important types of adult web sites. To get an overview of the distribution of paysites and free sites with regard to the total population of adult web sites, we applied our economic classification heuristic to the 35,083 adult web sites (domains) in our data set.

Our classifier was able to determine the role of 87,7% of these web sites. For the remaining 12,3%, whose roles remained undefined, we found a high percentage of web sites that either served empty pages, returned HTTP error codes (for example, HTTP 403 "Forbidden"), or were parked domains. We assume that many of these sites are either still under construction or simply down for maintenance during our crawling experiment.

Our results indicate that 8.1% of the classified sites are paysites and 91.9% are free sites (link collections). This is consistent with the intuition that we gained from our initial, manual analysis, showing that most adult site operators make money by indirectly profiting from the content provided by paysites.

3.3.4 Security-Related Observations

For either economic role, we found a relatively large number of web sites that use questionable methods and techniques that can best be described as "shady." Unlike well-known web-based attacks and malicious activities (such as drive-by downloads [102, 117]), these practices directly aim at manipulating and misleading a visitor to perform actions that result in an economic profit

for the web site operator. Overall, we found free sites to employ at least one of these techniques more often (34.2%, see Figure 3.3) when compared to paysites (11.4%). In particular, we frequently found the techniques listed below on adult web sites.

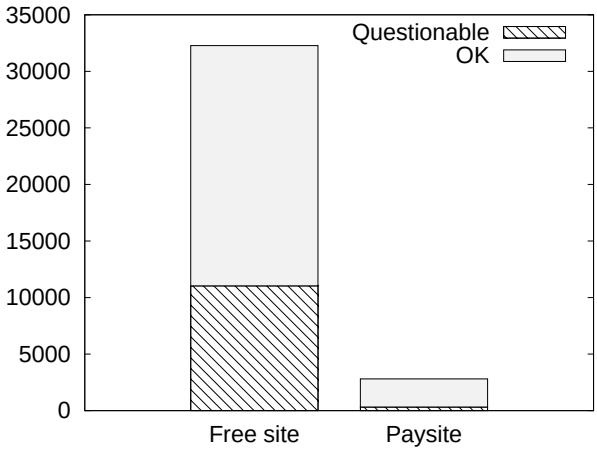


Figure 3.3: Distribution of economic roles.

JavaScript Catchers

These client-side scripts "hijack" the user's browser, preventing him from leaving the web site. To this end, usually JavaScript code is attached to either the `onunload` or `onbeforeunload` event handlers. Anytime the user tries to leave the web site (e.g., by entering a new address, using the browser's "Back" button, or closing the browser) a confirmation dialog is displayed. The user is then asked to click on a button to leave the web site, while, at the same time, advertisements are displayed or popup windows are spawned. Apart from the obvious annoyance, this could easily be used in a *clickjacking* attack scenario [61]. We detected catcher scripts in 1.2% of the paysites and 3.9% of free sites.

Blind Links

This technique uses client-side scripting via JavaScript to obscure link destinations, effectively preventing the addresses from being displayed in the web browser's status bar. The most popular methods that we found in the wild either work by overwriting the `window.status` or `parent.location.href` variables. We scanned the source code of the web sites for occurrences of these

variable names, and found 10.9% of paysites, and 26.2% of free sites to use blind links.

While the destination addresses are still contained in the web page source code, we believe it is fair to assume that most users will be unable to extract them. This is problematic, as it not only leaves the user unaware of the link's destination (leading to different web sites), but could also potentially be used to mask malicious activities such as cross site scripting (XSS) or cross site request forgery (CSRF) attacks.

Redirector Scripts

Redirector scripts make use of server-side scripting (for example PHP scripts) to redirect users to different web sites. In contrast to blind links, the link targets are determined at the server at run-time, making it impossible for a client to know in advance where a link really points to.

Typically, these redirector scripts are presented in combination with pornographic media. For example, small preview images usually have links to full-size versions attached. Instead of this expected behavior, users are redirected with a probability p to different web sites (so called *skimming rate*). The rationale behind redirector scripts is that users will know from experience that by keeping on clicking on the preview image, the desired media will eventually be shown at some point. At the same time, they "generate" artificial outgoing traffic for the web site, even though the user originally never intended to leave the site.

In our crawler implementation, we use a simple, yet effective technique to detect redirector scripts. Whenever our system finds hyperlinks with a destination address that contains a server-side script (currently `*.php` and `*.cgi` scripts), it resolves the link 10 times. If there is more than one destination address, the script is regarded as a redirector script, and the set of targets is added to our crawling queue. We chose a value of 10, because in our initial tests, we observed this as an upper bound for the number of redirection targets. When tested on a sample of 100 redirector scripts, none of them exceeded this threshold.

We found examples of p ranging from 0 (no *random* redirection) to 1 (the promised content is never shown). Also, the number of possible target addresses n varied from 1 to 6 destinations. Interestingly, only 3.2% of paysites but 23.6% of free sites contained redirector scripts. This implies that free sites have an incentive for using this technique.

The most likely explanation of this phenomenon are traffic brokers (see Section 3.2.2). These services have specialized in (adult) *traffic trading* and allow visitor traffic to be sold, a unique feature available only in this type of

online industry. This means that a miscreant could lure unsuspecting visitors who click on pornographic media to click on redirector links. The resulting traffic can then be sold to such a traffic trading service, which redirects it to targets of the buyer's choice. The web site operator earns money with every click, even if a single visitor clicks on one links many times – something not possible in traditional online advertisement.

Redirection Chains

If web sites which contain redirector scripts link to other sites with redirector scripts, we call this a *redirection chain*. This topology can be abused to further increase the revenue from artificial traffic generation.

We observed that JavaScript catchers are frequently used in conjunction with redirector chains, effectively "trapping" the user in a network of redirections. In our evaluation, we found 34.4% of those web sites that use redirector scripts to be part of redirector chains. Potentially, this could easily be abused for performing click-fraud or similar traffic-based cyber-crime because it enables the redirection operators to direct large amounts of "realistic" traffic to destinations of their choice. We study this phenomenon in more detail in Section 3.4.

3.3.5 Malware

To find more "traditional" web-based attacks, we applied our client honeypot analysis (see Section 3.2.3) to all 269,566 pages in our data set (which represents the adult web sites' main pages, subdomain pages, and enter page targets). Of these, 3.23% were found to trigger malicious behavior such as code execution, registry changes, or executable downloads. This percentage is significantly higher than what we expected based on related work [102], where slightly more than 0.6% of adult web sites were detected as malicious.

We used Anubis [17], a behavior-based malware analysis tool, to further analyze the malware samples that were collected by the honeypots. Also, Wepawet could successfully identify several families of exploit toolkits used by the malicious sites. This gave us human-readable malware names for the malware, showing that the most popular types of malware that we found are Spyware and Trojan downloaders (e.g., `rootkit.win32.tdss.gen` or `backdoor.win32.bandok`).

Whenever `iframes` were used as infection vectors, we extracted the hosting location of the injected code, finding the malicious code to be mostly (98.2%) not stored on the adult web sites themselves. We believe this is a clear indication that the web sites that distribute the malware were originally exploited themselves, and are not intentionally serving malware. This

was also confirmed by results from Wepawet, which automatically attributed several exploits to the "LuckySploit" malware campaign [51].

3.3.6 Hosting Infrastructure

In this section, we provide an overview of the information we collected on adult web site hosting.

Geographic Distribution

As part of our work, we analyzed the geographical distribution of adult web site hosting. To this end, we used a publicly available database [91] that contains a mapping of IP addresses to countries. We then queried this database with the IP addresses extracted from DNS lookups of the 35,083 adult web sites domains in our data set. Table 3.1 shows the most popular countries for hosting adult web sites.

Not surprisingly, the order of countries in this list appears largely similar to statistics that rank countries by the revenue generated from adult web sites [65]. However, several very small countries are ranked surprisingly high, for example, the Virgin Islands, Gibraltar, or Dominica. We speculate that this might be related to these countries' well-known roles as business off-shoring centers, or legal requirements for hosting adult content.

Pos	Country	Web Sites
1	United States	26,793
2	Netherlands	2,502
3	Germany	1,122
4	United Kingdom	1,048
5	Canada	779
6	Spain	720
7	France	638
...	...	...
14	Virgin Islands (brit.)	217
22	Gibraltar	75
26	Dominica	64

Table 3.1: Distribution of domains per country.

Whois Anonymization

We also matched the Whois entries against signatures for Whois anonymization services, such as Go Daddy's *Domains by Proxy* [46] or Namecheap's *WhoisGuard* [121] service. These commercial services provide "privacy" by substituting the name of the registrant with a generic entry, and an anonymous email address, while still allowing the original registrant to be contacted.

We found anonymization service signatures in 9.4% of the Whois entries in our data set. In contrast, we also performed this check on the Whois entries for the top 5,000 ranking web sites of Alexa [6], and found anonymization signatures in only 1.9% of these entries. On manual inspection, almost all of these sites were either gambling or file sharing related web sites, or of pornographic nature. Clearly, this reflects an increased interest for privacy for operators of adult web sites.

3.4 Experimental Evaluation

The analysis methods and findings presented in the previous sections allow us to gain information from an external observer's point of view, enabling us to outline the online adult industry's business relationships and studying some security-related aspects. However, we are also interested in more technical, security relevant information that is only available to adult web site operators themselves, for example, data about the web site visitors or the mechanisms behind traffic trading. One of the goals of our research is to estimate the malicious potential of adult web sites, for example, as a mass exploitation vector. Therefore, we also need the *internal* point of view to understand this area of the Internet in detail.

Unfortunately, we are not aware of any available real-world data set that could be used for such an analysis. Therefore, we took over the role of an adult webmaster and created two adult web sites from scratch to conduct our experiments.

3.4.1 Preparation Steps

To be able to interact with the adult industry, we performed the following operations to mimic an adult web site. First, we created two relatively simple web sites. We designed both sites' layout to resemble existing, genuine adult web sites, allowing us to blend in with the adult web site landscape. We chose to mimic two popular types of free sites, one "thumbnail gallery" web site and one link collection web site. After registering domain names that are

indicative for adult web sites, we put the sites online on a rented web hosting server.

Affiliate Programs. To receive promotional media, we then registered as an adult web site operator at eight adult affiliate programs. Surprisingly, the requirements for joining affiliate programs appear to be very low. In our case, only the web site URL, a contact name, and an email address had to be provided. There is no verification of neither the contact identity information nor is a proof of ownership required for the web site.

Immediately before signing up to an affiliate program, we created a snapshot of our web server access logs. As soon as an affiliate program accepted our application, we compared the current access logs to the snapshot. We found that six of the eight affiliate programs were accepting our application, even though no access to our web sites happened during the period between sign-up and acceptance. This means, that they were blindly accepting our application, performing no check of the web sites at all.

Traffic Brokers. Furthermore, we also registered our web sites at four traffic brokers that we chose due to their popularity among adult site operators, allowing us to participate in traffic trading. The registration procedure was almost identical to affiliate programs, and again, most brokers accepted our application without looking at the web sites. Only one broker checked our site and subsequently declined our application after detecting our analysis scripts (see next section).

Payment System. To be able to buy traffic, we had to send money to the traffic brokers. To this end, we used the "ePassporte" electronic payment system, that is popular among adult site operators, as it is widely accepted in the adult industry. We spent slightly more than $160 for our traffic trading experiments (including transaction fees).

3.4.2 Traffic Profiling

Our main goal in operating these web sites is to acquire as much security relevant information about web traffic coming to the sites as possible. To this end, we added several features to the web sites that allow us to collect additional information from each visitor. Since the collected data may contain detailed information about a unique visitor, his surfing habits, and other privacy related information, we implemented several precautions to protect the user's privacy (e.g., anonymization of the collected raw log data). This information

is then used in subsequent analysis steps, e.g., to determine if a user is vulnerable to remote exploits like arbitrary code execution or drive-by downloads, or to analyze what other kinds of sites the visitor surfed before coming to our site. Specifically, we collect the following information from each visitor:

Browser Profiling. First, we store general information for each visitor that is available through the web server log files, for example, the `User-Agent` string and the HTTP request headers that are sent by the user's browser.

Additionally, we added several JavaScript functions to the web site. These routines gather specific data about a visitor's web browser capabilities, for example, the supported data types or installed languages. We also collect information about any installed browser plugins, including their version numbers. This information is security relevant, as browser plugins are frequently vulnerable to remote exploits, and we can infer from this data if the visitor is potentially vulnerable to a drive-by download attack.

In particular, we are interested in the Flash browser-plugin [5], which is typically used to embed videos in web sites, as it is known for its bad security record [113]. Our intuition is that visitors to multimedia-rich adult web sites will most likely have Flash installed. Therefore, in addition to the plugin detection, we implemented a JavaScript-independent Flash detection mechanism that uses a small Flash script to check if the user has Flash installed. This allows us to detect vulnerable clients, even if they have JavaScript turned off (see Section 3.4.4). In addition to Flash, we also check for vulnerable versions of browser plugins for the Adobe PDF document viewer and Microsoft Office as they are the most prevalent targets for malicious attackers [14].

Browsing History. For an unscrupulous web site operator, it is desirable to know which other web sites a particular user has visited in the past. Exposing the user's browsing history can be used for advertising purposes or – in a malicious variation – to launch targeted phishing attacks.

As shown by Jakobsson and Stamm [83], it is possible to actively query for URLs in the user's browsing history by abusing cascading style sheets (CSS). In Chapter 2 of this thesis, we already introduced the background of this technique in more detail (although in a different context). The basic idea of the attack is that a URL is displayed differently by the browser once a user has accessed this URL (to indicate to the user that he has surfed this URL in the past). This visual difference can be queried via CSS, enabling an attacker to enumerate which URLs a visitor has accessed in the past. In addition to the 290 web sites proposed by Jakobsson and Stamm, we extended the list with the top 100,000 web sites from the Alexa [6] index to collect a larger data set

about each visitor's browsing history. Tests with different browsers showed that this technique can result in a suspiciously high CPU load and sluggish GUI behavior for the client if many domain names are queried in parallel.

This might indicate that something unusual is going on, and therefore, we adapted our system to perform the domain name lookup in chunks of 1,000 domains each. This technique successfully solved this problem when testing history stealing on several different machines. Note that if a user decides to leave our web site before the history query finishes, we would still receive matching domain names up to the last chunk that was processed.

Outgoing Links. To be able to verify statistics provided by affiliate program partners, we track all outgoing (i.e., leaving the web site) hyperlinks that a user has clicked. This is implemented by scripts that operate similar to redirector scripts often employed on adult web sites (see Section 3.3.4 for details).

3.4.3 Traffic Buying Experiments

After having prepared the web sites with our profiling tools, we placed orders for buying web site visitors at three different traffic brokers. We tested different brokers to study the differences in delivered traffic and to gain a better understanding of their intricacies. In total, we ordered almost 49,000 visitors at the three different traffic brokers during a period of seven weeks. We spent a total of $161.84 on these traffic orders (average $3.30 per thousand visitors). Surprisingly, each traffic broker redirected traffic to our site (almost) instantly after placing an order. This suggests that they have an automated traffic distribution system in place, capable of flexibly rerouting traffic to customers, and enough incoming traffic that they can handle orders in a timely manner. Checking our web server logs confirmed that we indeed received the correct amount of visitors (e.g., clients with unique IP addresses) at the correct rate for *all* orders.

In addition to the rate limit, we also chose the more expensive "high quality" option when buying traffic, which is regarded by traffic brokers as synonymous with traffic coming mostly from the US and Europe. To verify the geographical origin of traffic, we performed an IP to country lookup for the bought traffic. We found that 98.22% of the traffic really originates from the US and Europe, thus the origin is correct for the vast majority of visitors.

3.4.4 Profiling Results

After having received the ordered amount of traffic, we analyzed the output of the profiling steps outlined in Section 3.4.2. An overview of the results of

this analysis is shown in Table 3.2. All brokers sent a similar type of visitors to our site and there are no major differences between the brokers. Therefore, we discuss the overall results in the following sections.

Browser Profiling

When a visitor accesses one of our web sites, we automatically start to collect information about him (e.g., all request headers and information about browser extensions). In certain cases, our system cannot obtain this profiling information for a web site visitor. The reasons can be manifold, for example a client can have JavaScript support disabled, it can be an "exotic" web browsers with reduced functionality, the visitor might stay for only a few seconds on our web site, or it might not be a human visitor but a bot. The most prevalent case were visitors that did not correctly execute our JavaScript-independent Flash detection: 18,794 (38.43%) of our overall visitors behaved in this way. In contrast, 30,106 (61.57%) visitors correctly performed the test, and of those 96.24% had Flash installed. Furthermore 10,214 visitors (about 20.89%) did not download any images, but just requested the HTML source code of the site. While we cannot coherently explain this behavior, we think that it is caused by bots (e.g., click-bots [43]), since the browser of a human visitor would start to download the complete content of the site.

For about 47% of all visitors we were able to build a *complete* browser profile, which includes all the information we are interested in. For the remaining visitors only certain types of information were collected (e.g., only HTTP headers and no other information since the visitor spent not enough time on our site). We opted to analyze only the cases in which we have collected the complete browser profile to be conservative in our analysis.

During our analysis we also detected some noteworthy anomalies that prohibit browser profiling. For example, about 0.53% of the visitors used browser versions typically found in mobile phones or video game consoles (such as Nintendo Wii, Playstation Portable, or Sony Playstation). These devices do not fully support JavaScript or have a limited set of features, preventing our profiling scripts from executing correctly. We also found that in about 0.14% of the cases our profiling did not work since the HTTP headers were purged, a fact that we could attribute to clients which have the Symantec Personal Firewall installed.

Vulnerability Assessment

We determine if a client is vulnerable to *known* exploits by matching the visitor's browser properties (e.g., version number of common plugins and add-ons)

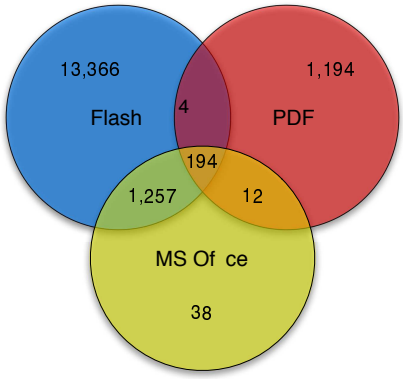
	Broker A	Broker B	Broker C	Total
Ordered Visitors	12,000	7,900	29,000	48,900
Performed Flash Detection	8,638 (71.98%)	5,010 (63.42%)	16,458 (56.75%)	30,106 (61.57%)
↦ Flash Found	8,401 (97.26%)	4,876 (97.33%)	15,697 (95.38%)	28,974 (96.24%)
Complete Browser Profiles	6,183 (51.53%)	3,682 (46.60%)	13,176 (45.43%)	23,041 (47.12%)
↦ Vulnerable	5,251 (84.93%)	3,242 (88.05%)	11,847 (89.91%)	20,340 (88.28%)
↦ Browsing History	3,635 (58.79%)	2,066 (56.11%)	8,211 (62.32%)	13,912 (60,38%)
# Clicked Links	3,662	2,742	8,997	15,401

Table 3.2: Statistics about the visitors studied during our traffic buying experiments.

against a list of common vulnerabilities we compiled manually. We focused on only the most prevalent browser plugins such as those related to Adobe Flash and PDF, and Microsoft Office. These three plugins had seven vulnerabilities in the recent past, and an attacker can buy toolkits that exploit these vulnerabilities to compromise a visitor [14]. Since realistically, additional exploits (even some that are not publicly known yet) exist in the wild, this provides us with a lower bound for the number of vulnerable systems among visitors to our web sites. Using this heuristic, we found that more than 20,000 visitors had at least one vulnerable component installed and more than 5,700 visitors had multiple vulnerable components. Figure 3.4(a) shows a Venn diagram depicting the prevalence of different types of vulnerabilities.

A malicious site operator could take advantage of these vulnerabilities and compromise the visitor's browser with a drive-by download [102]. Besides the opportunity to build a botnet with only a small investment (e.g., we spent $160 and could potentially infect more than 20,000 machines), an operator could also earn money with the help of so called *Pay-Per-Install* (PPI) affiliate programs. In a PPI program, the "advertiser" pays the partner a commission for every install of a specific program by a user. The exact amount of this commission depends on the countries that the users come from. For example, we registered at one PPI program (note that we did *not* install any software to clients) and found the rate for 1,000 installs to computers located in the US and parts of Europe to be set to $130, while it would be as low as $3 for most Asian countries. This is consistent with information that we manually compiled from five other PPI program web sites. Related work that focuses on PPI (for example [111]) lists even higher prices per installation. Since we only bought US and European traffic in our experiments, we found a large fraction of traffic to fall into the highest selling PPI category (more than 95%). While an in-depth analysis of PPI programs is outside the scope of this work, these figures clearly show that it would be highly profitable for a malicious site operator to participate in PPI programs, and covertly trigger installs of unwanted software at vulnerable clients.

In addition to vulnerable browser versions and plugins, we also analyzed the `User-Agent` strings obtained from the visitor's browser. This enables us to detect certain cases of clients that are already compromised: While the `User-Agent` string can be arbitrarily set by a client, it is still a good indicator for clients that are infected with certain types of malware, which intentionally "mark" infected clients to avoid re-infection or change the behavior of web sites that act as an infection vector. We found 915 clients (1.87%) that contain known malware marker strings, such as for example the adware "Simbar", or scareware like "Fake Antivirus 2008". Figure 3.4(b) provides an overview of the most common suspicious `User-Agent` strings we observed in the visitor's



(a) Distribution of the three vulnerability types we examined. Note that the display format is not proportional.

User-Agent	# Suspicious Clients	Type
FunWebProducts	260	Adware
SIMBAR	136	Adware
DesktopSmiley	93	Spyware
JuicyAccess	85	Nagware
Antivir XP 2008	52	Fake AV
Seekmo	46	Adware
3P_UV	45	Fake AV
Other	198	-
Total	915	-

(b) Overview of suspicious User-Agent strings that we observed frequently, indicating that these clients are presumably infected with some kind of malware, e.g., scareware or adware.

Figure 3.4: Results for vulnerability assessment of clients studied during traffic experiments.

browser.

Browser History Evaluation

For more than 13,900 visitors (about 60% of the complete browser profiles), we were able to probe the browser history of the visitor and found at least one domain with this technique. In total, we found 218,255 domains in the visitor's browser history. We noticed that the distribution is highly biased: the average per visitor is 15.7 sites, while the mean is 3 sites. For 3,973 visitors, we found only one domain during our probing. We suspect that this bias is mainly introduced by three factors. First, probing of the browser's history commonly takes at least 30 seconds: If a visitor leaves the page earlier, we cannot probe all sites. The high number of incomplete profiles already hints that bought visitors often do not spend a long time on a redirected site. Second, since our probing technique did in almost 40% of the cases not even find popular web sites, we speculate that visitors to adult web sites more carefully control their browser's history. Third, recent versions of different browsers introduced a feature that enables *private browsing*, i.e., in this mode, the browser does not record browsing history.

One malicious use-case of browser history information are targeted phishing attacks, for example to obtain online banking credentials or to facilitate identity theft by displaying fake versions of web sites that the user is indeed using [83]. Another use case would be to change the content of the site depending on the visitor's history.

To understand the browsing habits of a typical user, we determined the category of each domain with the help of the OpenDNS domain tagging database [94]. This database assigns, based on human classification, a number of tags such as for example *Nudity*, *Webmail*, or *Auctions* to domains according to their content. We found that the domains we detected in our experiments are highly biased towards adult themes: about 40% of the domains had the tag *Pornography*, and 36% the tag *Nudity*.

3.4.5 Traffic Selling Experiments

Traffic brokers also allow their clients to *sell* web traffic, paying them for visitors that are redirected to the broker's web site; from there the visitors are forwarded to traffic buyers (see Section 3.2.2 for details). The commission a traffic seller receives mainly depends on the sexual niche that is attributed to the traffic, and is influenced by the type of web site the seller operates. To explore the security aspects of traffic selling, we included traffic selling links on our web sites and participated in this business.

Click Inflation Fraud Scenario

The first thing we noticed is the fact that traffic brokers do not require traffic selling web sites to include any content (for example a script) that is hosted by the traffic broker or by a third party. This stands in contrast to other types of web businesses that rely on partner web sites to publish information. For example, online advertisers such as Google typically require the inclusion of JavaScript code that is hosted by the advertiser himself on the publisher's web site. This code enables the content provider to acquire information about the publishing web site that can be used for abuse and fraud detection, for example by computing the click-through-ratio (CTR) or by checking the cookie information of a user that clicks on a link.

Since traffic brokers do not use this technique, they cannot implement these well-known techniques for fraud detection and are thus subject to specific abuses. However, we found that traffic brokers check the HTTP `Referrer` header of redirected traffic to see if it really originates from the seller's web site. If this is not the case, the traffic is either rejected (redirected back to the seller), or only a very low price is paid.

These observations led us to the assumption that the level of sophistication of anti-fraud techniques employed by traffic brokers is rather low. To verify this assumption, we devised a simple, yet effective fraud scenario to test the vulnerability of traffic brokers to click fraud. In this scenario an attacker (legitimately) buys traffic from at least one traffic broker, and then "resells" this traffic to n different traffic brokers in parallel by forwarding the incoming traffic. Figure 3.5 illustrates the concept of our attack, which is a variation of click inflation attacks [10].

With the help of this *n-fold click inflation*, an attacker can earn money if the total earnings from selling the traffic n times exceeds the amount of money she needs to spend for buying the traffic. Furthermore, she could even earn more money by abusing each visitor she bought, for example by compromising vulnerable visitors with the help of a drive-by download.

From a technical point of view, we found that simple HTTP or JavaScript redirections to the traffic selling URLs would not suffice, as many popular web browsers such as Internet Explorer and Mozilla Firefox incorporate pop-up blocking features that prevent opening new browser windows without the user's interaction. However, during our traffic buying experiment, we noticed that a relatively high amount of visitors clicked on links on the web site: overall, we had more than 15,400 clicks based on just 48,900 visitors (see Table 3.2).

Since pop-up blockers do not trigger if user interaction is involved in opening links, we were able to attach JavaScript code to the `onclick` event handler of

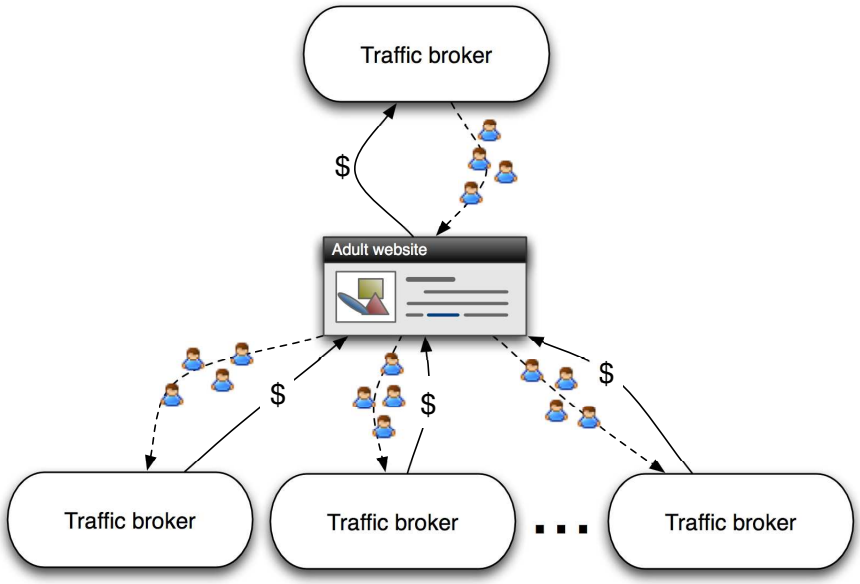


Figure 3.5: Overview of n-fold click inflation fraud scenario.

hyperlinks. This allows us to perform n-fold traffic selling every time a user clicks on a hyperlink on the web site.

We performed an experiment that shows that this attack is effective against traffic brokers: We signed up as traffic seller at two different traffic brokers and bought visitors from a third broker. Each click on our web site was redirected to both brokers. We implemented only a 2-fold click inflation attack to test the setup in practice, but higher values for n can be implemented without problems. In total, we bought 17,000 visitors to our site, from which more than 1,800 visitors clicked at least one link and thus we could sell them to both brokers. The visitors generated a total of 4,100 clicks, which could also be abused in other ways by a malicious site operator. During our experiment, we successfully accumulated funds of slightly less than $10 (on average, we received $2.22 for 1,000 sold visitors). To prevent damage to the traffic broker, we did not withdraw any funds, but forfeited our traffic trading accounts.

Based on the insights gained from this experiment, we think that other, even more powerful types of click fraud (such as *clickjacking* [61]) would be equally easy to employ. The success of this experiment also suggests that traffic brokers do not share information among each other about traffic sold, and that no advanced fraud detection systems are in place.

Malware Distribution

The main drawback of traffic selling is that the seller does not have any control over the final destination of sold traffic. This is worrisome, as this can be abused in a number of ways. For example, we discussed in Section 3.4.4 the possibility that a malicious operator could buy traffic and then use drive-by download attacks to infect visitors. Since traffic brokers seem to have low standards for traffic buyers (e.g., no background check was performed when we signed up), a benign traffic seller could unintendedly become part of malicious activity by redirecting traffic to malicious web sites. Due to the fact that redirection by traffic brokers is transparent to a client, it could directly harm the reputation of the seller.

To study the prevalence of malicious redirects, we followed our own traffic selling links to check the final destination of our traffic. To our surprise, we were redirected to a web site that provides a fake video player that would trigger the download of an executable upon clicking the "Play" button. By closer examination, we found that the downloaded file contains the JuicyAccess toolbar, which we detected previously in the context of infected clients (see Section 3.4.4). Thus, there are, indeed, malicious web site operators that buy traffic to infect visitors on purpose.

3.5 Summary

In this chapter, we examined the challenges and issues for security and privacy of a whole branch of online business. The online adult industry is one of the largest industries, and the general consensus is that these websites are privacy threatening. In order to empirically determine if these claims are founded, we performed a case study on this industry, and conducted a technical and economical analysis. We observed that questionable business practices (e.g., traffic trading, redirection chains) are frequently employed. In our experimental evaluation, we found that an attacker mimicking adult website operators would only require little effort to perform large-scale malicious activity, such as mass exploitation, drive-by downloads, or browser history stealing.

Clearly, this facilitates criminal activity like malware distribution, bot herding, affiliate fraud, or theft of private data. Many of these activities aim at compromising the computers of website visitors to install malware, for example, bots, keyloggers, or trojans. The malicious software is then used to steal private information from the victim, and send it to the criminal, for example, bank account numbers, passwords, or whole documents. Often, such malware programs use unknown or custom communication protocols, making it costly

and tedious for human analysts to understand their function.

In the following chapter, we will present a system that aims at automatically reverse-engineering unknown application-level network protocols. From a security point of view, such a system has many interesting applications, ranging from the analysis of legitimate software (that might use proprietary or undocumented protocols) to malicious, privacy-threatening bots. Specifically, we show that our system can reverse the protocols used in real-world applications, including a malicious bot's command and control channel.

Chapter 4

Extracting Privacy-Relevant Information from Network Protocols

Protocol reverse engineering is the process of extracting application-level specifications for network protocols. Such specifications are very useful in a number of security-related contexts, for example, to perform deep packet inspection and black-box fuzzing, or to quickly understand custom botnet command and control (C&C) channels. Furthermore, reverse-engineering application level protocols also has important implications for the privacy of users. The resulting protocol specifications can be used to monitor and understand if private or sensitive information is sent or received by applications - for example, by malicious bots or keyloggers.

Since manual reverse engineering is a time-consuming and tedious process, a number of systems have been proposed that aim to automate this task. These systems either analyze network traffic directly or monitor the execution of the application that receives the protocol messages. While previous systems show that precise message formats can be extracted automatically, they do not provide a protocol specification. The reason is that they do not reverse engineer the protocol state machine.

In this thesis, we focus on closing this gap by presenting a system that is capable of automatically inferring state machines. This greatly enhances the results of automatic protocol reverse engineering, while further reducing the need for human interaction. We extend previous work that focuses on behavior-based message format extraction, and introduce techniques for identifying and clustering different types of messages not only based on their structure, but also according to the impact of each message on server behavior. Moreover, we present an algorithm for extracting the state machine. We have applied our techniques to a number of real-world protocols, including the command and control protocol used by a malicious bot. Our results demonstrate that we are able to extract format specifications for different types of messages and meaningful protocol state machines. We use these protocol specifications to automatically generate input for a stateful fuzzer, allowing us to discover security vulnerabilities in real-world applications.

4.1 Introduction

Reverse engineering is the process of analyzing a device or a system to understand its structure and functionality. In the context of network protocols, reverse engineering describes the process of deriving the application-level protocol specification of an unknown protocol. To this end, an analyst can monitor the exchange of messages over a network or observe how the communication end-points (such as client and server) process network input. The detailed knowledge of a protocol specification is important for addressing a number of security problems.

Given a protocol specification, it can be used to generate protocol fuzzers [93] that perform black-box vulnerability analysis of network applications. In fact, many vulnerabilities have been found in the past that resulted from programming errors in protocol parsing code [72]. Moreover, detailed protocol specifications are required by intrusion detection systems (e.g., Bro [99]) that perform deep packet inspections. Also, the ability to generate protocol specifications is useful for generic protocol analyzers that require protocol grammars as input (e.g., binpac [97] and GAPA [22]). Furthermore, protocol reverse engineering can help to identify (subtle) variations in the way that different applications implement the same protocol. These differences can be used for application fingerprinting [115] or to discover security vulnerabilities [27]. Finally, the analysis of malware is another important area where protocol reverse engineering can be applied. Botnets [42] increasingly make use of non-standard communication protocols [101, 11]. For a security analyst who attempts to understand and take down botnets, the ability to automatically reverse engineer the command and control protocol is clearly helpful.

In general, reverse engineering is largely a manual, tedious, and time-consuming process. To support a human analyst with this task, a number of automatic protocol reverse engineering techniques have been proposed. These techniques aim to automatically generate the specifications of an application-level protocol. Two possible input sources can be used to analyze a protocol: network traffic and an application that implements the protocol.

A number of approaches [38, 39, 78, 79] have been presented that use network traffic as input. These systems typically analyze traces generated by recording the communication between a client and a server. Then, heuristics are applied to extract different protocol fields and delimiters. Although useful in practice, the precision of these systems is often limited. That is, it is not always possible to extract all required information about a protocol from the network traffic alone. To address the limited precision of techniques that operate directly on the network traces, a number of systems [29, 80, 123, 40] were introduced that focus on the (server) application. More precisely, these sys-

tems operate by observing the execution of the application while it is processing input messages. This allows them to infer the structure of a message (i.e., its constituent fields) with higher precision, and it provides insight into field semantics that are not available to network-trace-based approaches. A common property of *all* previous systems (whether network- or behavior-based) is that they only extract the format of individual protocol messages.That is, these systems do not aim at reverse engineering the protocol state machine, and, therefore, cannot produce specifications for *stateful network protocols*.

In this thesis, we introduce Prospex (Protocol Specification Extraction), a system that can automatically infer specifications for stateful network protocols, i.e. including state machine information. To the best of our knowledge, this is the first system with this capability. Our analysis builds upon a system introduced in previous work [123], which can extract the format specifications of individual messages by monitoring the application as it processes its inputs. For this thesis, our system was extended in two main directions. First, we developed a mechanism to identify messages of the same type. This information is leveraged to combine similar messages into clusters. The second extension is related to the inference of a protocol state machine. The protocol state machine encodes all sequences of messages that are permitted by the protocol. Information about the state machine is required to be able to engage in a "meaningful" conversation with a communication partner, e.g., knowing when a certain message can be sent.

In summary, the contributions of this work are the following:

- We propose several features to determine when two messages in a network session are similar. These features take into account not only the format of messages, but also the effect that receiving each message has on server execution. This allows us to automatically identify and cluster messages of the same type. Automatically recognizing different message types allows us to use a set of messages of the same type to generate a corresponding message format specification.

- We present a technique to automatically infer the protocol state machine. This state machine specifies the order in which messages can be exchanged, given no prior knowledge about the protocol under analysis. We further show that our technique consistently outperforms existing approaches for state machine inference.

- We applied our system to a number of real-world applications that implement complex, stateful protocols. The results demonstrate that our techniques are capable of extracting meaningful message formats and

protocol state machines. This is true both for protocols used by benign applications (such as SMTP, Samba, or SIP) and protocols used by malicious software (such as the C&C protocol used by Agobot).

- We leverage the output of our system to automatically produce protocol specifications for the open-source Peach fuzzing platform [100]. To this end, we actively contributed to the development of Peach and extended its support for stateful protocols. Running Peach with our specifications allowed us to automatically find vulnerabilities in real-world applications.

4.2 System Description

The input to our system are a number of application sessions. An application session is a connection between hosts that allows the involved machines to exchange data. Each session typically consists of a sequence of *messages*. Each of these messages has a message *type*, which is defined by a message format specification. The message format specifies the structure of a message, typically as a number of *fields*. The structure of the whole application session is determined by the *protocol state machine*. The protocol state machine defines the order in which messages of different types can be sent.

The objective of our system is to automatically infer the specification of an unknown protocol that a client uses to communicate with a server. More precisely, given a sequence of messages that a client sends to a server, we are interested in the specifications of these messages, as well as the protocol state changes that these messages result in.

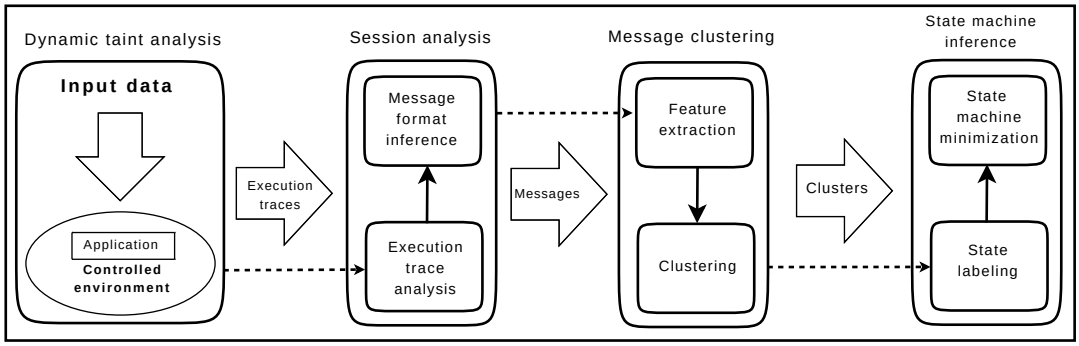


Figure 4.1: System overview.

To this end, our system proceeds in several phases, as shown in Figure 4.1.

Dynamic taint analysis. In this phase, dynamic data tainting is used to observe the application as it processes incoming messages. The resulting execution traces show the operations performed on data that was read from the network.

Session analysis. Initially, we analyze these execution traces, splitting them into individual messages. Then, we perform *message format inference* on each message, resulting in detailed format specifications for single messages.

Message clustering. We extract a number of features for each message from the execution trace. These features take into account the previously inferred message formats as well as the effect of each message on the application's behavior. The similarity between messages is determined using these features. Then, we apply the partitioning around medoids (PAM) clustering algorithm [74] to group similar messages into types. Finally, we derive a generalized message format specification for each type.

State machine inference. In this phase, we infer a state machine that models the order in which messages of different types may be sent. Initially, we construct a state machine that accepts exactly the sequences of messages observed during training. Then, we use a novel algorithm, based on domain-specific heuristics, to label the states of this state machine. States that may be similar are assigned identical labels. Finally, we apply the Exbar [75] algorithm to produce a more general, minimal state machine by merging similar states. Together with the generalized message format specifications for the different types of messages, this minimal state machine represents the reverse-engineered network protocol.

Fuzzing. Optionally, our tool can translate the extracted protocol specifications into input for the Peach fuzzing platform [100]. As we will show in Section 4.3, this allows Peach to test code that is only accessible in later protocol stages, finding "deeper" security vulnerabilities in real-world applications.

Message clustering. To recognize the types of messages, we assume that messages of the same type are similar. As a result, the goal of the message clustering phase is to identify and combine (cluster) similar messages. We regard two messages as similar (and hence, of the same type) when they share similar message formats *and* when the server "reacts" in a similar fashion upon receiving them.

Thus, in addition to comparing message formats, we propose a number of novel similarity features that are based on the analysis of the application's actions as it processes different messages.

Once all similar messages are clustered, we label each cluster (and all corresponding messages) with a type. Moreover, for each cluster, we generate

a generalized message format specification that describes all messages in this cluster.

Generation of fuzzing specifications. Optionally, our tool can translate the extracted protocol specifications into input for the Peach fuzzing platform [100]. As we will show in Section 4.3, this allows Peach to test code that is only accessible in later protocol stages, finding "deeper" security vulnerabilities in real-world applications.

4.2.1 Session Analysis

The purpose of the session analysis phase is to automatically retrieve the message format specifications for the messages that are passed between client and server within an application session. To this end, we leverage an observation that was exploited in previous work [29, 123] for the analysis of individual messages. This observation suggests that it is advantageous to monitor how a program *processes* its input messages instead of analyzing the traffic that is exchanged between hosts at the network-level, because this allows more precise inference of message formats. By using dynamic taint analysis [33, 35, 37], we can precisely track how the application, which "understands" the messages and implements the protocol state machine, handles input data. As a result of the session analysis phase, we obtain a sequence of messages for an application session, each represented as a tree of high-level messages fields. Previous work on message format inference includes systems that analyze either single inputs [29, 80] or, more generally, multiple inputs [123, 40]. Our current implementation extends work from [123] and automatically splits sessions into sequences of messages. Thus, the need for human assistance during this phase is removed.

Like previous work, our system is not capable of reverse engineering encrypted traffic. While this is an intrinsic limitation of approaches that only analyze network traffic, it could be overcome by systems that observe server behavior. For this, one could manually or automatically [119] identify buffers that hold decrypted messages and then use these buffers as a starting point for the analysis.

The following paragraphs outline the steps performed by our system to retrieve the message formats.

Recording execution traces. Initially, we execute the application that implements the protocol that we are interested in (e.g., a server program). The program is run in a controlled environment that supports dynamic data tainting [33, 35, 37]. This allows us to record all operations that involve data read from protocol messages. Then, we engage the server in a series of

application sessions, typically by connecting with a client program, performing some common tasks.

In this work, we limit ourselves to the analysis of the communication in a single direction. That is, we infer the protocol state machine only for one of the communication partners. Also, we only determine the specifications of the messages that this communication partner receives. For ease of presentation, we will refer to this communication partner as "server," and to the other as "client." Note that it would be possible to use our techniques to simultaneously monitor both the client and the server, eventually combining the two different state machines and sets of message formats.

With dynamic data tainting, the system assigns a unique label to each input byte and tracks the propagation of these labels throughout the execution of the program. The output of this step is, for each application session, an execution trace that contains all executed instructions as well as the taint labels of all instruction operands.

Splitting a session into messages. The execution traces that we record contain all instructions that are executed during an application session. As the next step, this trace needs to be split according to the individual messages. Since we assume no prior knowledge of message boundaries, we use a simple heuristic: The first message starts with the first input byte that the server receives. All subsequent input is considered to be part of this message. This continues until the server writes data to the socket from where it had received the input (that is, the server sends a reply). The next byte received from the client is considered to denote the start of the next message. This is repeated until all execution traces are split into segments, where each segment corresponds to one message. While this approach is not fully general, it is significantly more accurate than considering each network packet as a message by itself. The reason is that clients sometimes break a message into several packets (for example, interactive protocols). The server collects these packets until a complete message has arrived before a response is sent. Our heuristic correctly handles this case and combines multiple packets into a single message.

Inferring message formats. Once our system has identified an execution trace segment for each protocol message, we use the techniques presented in previous work [29, 123] to determine the format of each message. Using these techniques, we analyze an execution trace segment and split the corresponding message into fields. As a result, each message is represented as a tree of fields with associated semantics (i.e., delimited field, length field, pointer field, keyword, file name, etc.). The output of the session analysis step is a sequence of messages for each application session. Each message is represented as a tree

of nested fields.

4.2.2 Message Clustering

After the session analysis phase, the system has extracted a format specification for every individual message. However, there is no information about similarities between messages or their types.

Previous systems that perform message format inference operate on messages of a *single*, known type. However, our goal is to infer a protocol specification assuming no prior knowledge about message types (in fact, we do not even know *a priori* how many message types there are for a certain protocol). Therefore, we require a step that can recognize the types of different messages.

Thus, the goal of the message clustering phase is to assign a type to each message. To this end, we define a metric of similarity between messages, and use it to cluster together similar messages. Our similarity metric is based on the assumption that messages of the same type share similar message formats *and* that the server "reacts" in a similar fashion upon receiving them. Thus, in addition to comparing message formats, we propose a number of similarity features that are based on the analysis of the application's actions as it processes different messages. Once all similar messages are clustered, we label each cluster (and all corresponding messages) with a type.

By assigning types to messages, we can operate on a more abstract representation of protocol sessions. Moreover, for each cluster, we generate a generalized message format specification that describes all messages in this cluster.

Feature Extraction and Similarity Computation

To be able to cluster related messages, we require a way to assess their similarity. For this, we introduce a number of features and corresponding distance functions that allow our system to calculate the similarity between two messages. These features can be divided into three groups that are discussed in the following paragraphs. For each feature, we compute a normalized similarity score between 0 (meaning completely different) and 1 (meaning identical).

Input similarity. Clearly, when comparing two messages, the structure and order of the fields that these messages are composed of play an important role. That is, we would assume that two messages of the same type also contain similar fields in a similar order. To capture this intuition, we use a sequence alignment algorithm (the Needleman-Wunsch algorithm [90], to be more precise). The goal of this algorithm is to take two sequences as input and find those parts that are similar, respecting the order of the elements. These

similar parts are then aligned, exposing differences or missing elements in the sequences. In our case, we use the sequences of fields that each message is composed of. For more details on how the comparison between two messages is implemented, we refer the reader to [123].

Execution similarity. In addition to the format of the input messages, we also expect that messages of the same type are handled by similar code in the application. That is, when a message of a certain type is received and processed, we assume that the program uses the same code fragments, library calls, and system calls, at least to a certain degree. This intuition is captured by the following execution similarity features, which can be directly derived from the recorded execution traces.

- *System call feature*: This feature takes into account the types of system calls (as indicated by their system call number) that were invoked during the processing of a message. These system calls are stored as a set, that is, the order is not taken into account for this feature.

- *Process activity feature*: This feature is related to the system call feature, but focuses on system calls related to the generation or destruction of processes (such as **clone** and **kill**). Process-activity-related system calls are typically very indicative for the behavior of an application. When considered together with all other system calls (as part of the previous feature), these calls would not have sufficient weight in the similarity calculation.

- *Invoked function feature*: For this feature, we store in a set the target addresses of `call` operations that are executed during the processing of a message (if these addresses are within the application's text segment).

- *Invoked library functions feature*: This feature is used to track (dynamically-linked) library calls that are made by an application. To capture this feature, we record the target addresses of `call` operations that are outside the program's text segment. In the case of statically-linked binaries, we would recognize a library call as a regular function call.

- *Executed addresses feature*: For this feature, we use the set of addresses of instructions that are executed by the application while it is processing a specific message (if these addresses are within the application's text segment).

For each of the execution features listed above, we record a set of numbers or addresses that are associated with a certain message. To compare two messages, we employ the Jaccard index [66] to determine the similarity between

two features, defined as:

$$J(a,b) = \frac{|a \cap b|}{|a \cup b|}$$

In the equation above, a is the set of elements associated with a feature of the first message, while b is the set that represents the same feature of the second message. Clearly, $J(a,b)$ yields 0 when the sets are disjoint and 1 when they are identical. Note that we calculate five execution similarity scores, one for each of the five execution features.

Impact similarity. The last group of features captures the response of the server to a message that is received. Typically, a server application will execute a series of actions when receiving a (legitimate) request from a client. The goal of the following two impact similarity features is to represent some of these actions at a high level of abstraction.

- *Output feature.* This feature captures the output behavior of the server while processing a message. In particular, we are interested in all system calls that cause the server to write out data. More precisely, we consider the following four destinations for data write operations: the socket to which the client is connected, other network sockets, files, and the terminal. The socket to which the client is connected captures cases in which data is returned to the client (thus ending the message, as explained in Section 4.2.1), while other network sockets refer to cases in which a server sends data over a different connection. File and terminal destinations simply represent operations where the application writes data to one of these sinks.

  For each write operation, we also analyze the taint status of the data that is written. This allows us to distinguish between operations that write tainted data (i.e., data previously received from the client) and those that write other data. We then label each byte of output with a tuple $\langle$*sink, tainted*$\rangle$ that specifies where the data was written to and whether it was tainted or not. The output feature is represented by a sequence of such tuples, with consecutive duplicates removed.

  Finally, we use the Needleman-Wunsch sequence alignment algorithm to compare output sequences, as for the input similarity. The result is the output similarity score.

- *File system feature.* This feature captures the file system activity of the server when handling an input message. Therefore, we consider system calls that perform file system actions, such as opening or closing a file, or obtaining information about a file or directory. In a first step, we represent the file system activity as a set of $\langle$*operation, path*$\rangle$ tuples, where operation is one of

{**open**, **close**, **read**, **write**, **rename**, **stat**, **mkdir**, **rmdir**}, and path is the path of the file that the system call operates on.

The name of the file or directory on which a system call operates may be specific to the individual execution trace, and, therefore, needs to be generalized. Specifically, we look for prefixes of the path that are either hardcoded in the binary (by scanning for strings in the program's file on disk) or are found in one of the program's configuration files (if provided by an analyst). We can also detect those parts of a path that are tainted, and as such, represent a parameter of a client request. To perform generalization of a path, we first attempt to look for the longest prefix that matches a string that is found in the binary. The remaining parts are then replaced with one of the special tokens *TAINT*, *CONFIG*, or *VARIABLE*. The VARIABLE token is used for all parts that are neither tainted nor appear in a configuration file.

As a result of the previous step, the file system feature is represented by a set of tuples, for example, ⟨open, "*/CONFIG/TAINT*"⟩ or ⟨write, "*/var/log/samba/VARIABLE*"⟩. To compute the similarity measure between the file system features of two messages, we use the Jaccard index, as for the execution similarities.

Clustering

Based on the features and similarity functions described in the previous section, we compute the distance between a pair of messages a and b as $d(a,b) = 1-\sum_i w_i s_i(a,b)$, where s_i is the similarity measure for feature i, and $\sum_i w_i = 1$. The weights w_i are selected in such a way that each of the three groups of features described above has the same overall weight of $\frac{1}{3}$, and that features in the same group have equal weight. Once a distance matrix is computed, we can use off-the-shelf clustering techniques to classify our data. Specifically, we employ the partitioning around medoids (PAM) algorithm [74].

Like most partitioning-based clustering techniques, the PAM algorithm takes as input the number k of clusters to generate. To determine a suitable value for k, we employ a generalization of the Dunn index. The Dunn index [47] is a standard intrinsic measure of clustering quality, defined as:

$$D(k) = \frac{min_{1\leq i\leq k}\{\min_{1\leq j\leq k}\{\delta(C_i, C_j)\}\}}{max_{1\leq i\leq k}\{\Delta(C_i)\}} \tag{4.1}$$

where $C_1, .., C_k$ are the clusters, $\Delta(C_i)$ is the diameter of cluster C_i, and $\delta(C_i, C_j)$ is the distance between the two clusters. Since the numerator of Equation 4.1 is a measure of cluster separation, while the denominator is a measure of cluster compactness, k should be chosen so that the Dunn index is

maximized. To compute the distance between two clusters (δ in Equation 4.1), we use the single-linkage distance defined as $\delta(C_i, C_j) = min_{a \in C_i, b \in C_j}\{d(a, b)\}$. To compute the diameter of a cluster (Δ in Equation 4.1), we use one of the measures defined in [95], which is based on *Relative Neighborhood Graphs*. Once clustering has been performed, we derive a format specification for each message type by merging the formats of all messages in the corresponding cluster. This merging step leverages techniques from [123].

4.2.3 State Machine Inference

The previous clustering phase identifies similar messages in application sessions, assigning a different type to each cluster. As a result, each session s_i can be represented as a sequence $S_i = (\sigma_1, .., \sigma_h)$, where $\sigma_1, .., \sigma_h \in M$ and M is the set of message types. The goal of the state machine inference phase is to infer an acceptor machine that can recognize sequences of message types that represent valid sessions of the protocol under analysis. Unfortunately, this problem cannot be solved exactly, even if we assume that the language comprising all valid sequences is a regular language. The reason is that Gold [56] has proved that a regular language cannot be learned from positive examples only. Moreover, the problem is even more difficult for more powerful languages.

A commonly-used approach to infer a regular language from a labeled training set (a labeled training set is a set of example strings, labeled *accept* or *reject*), is to find the smallest automaton that is consistent with that training set [28]. Such an approach selects the simplest, most-generic hypothesis consistent with the observations. Unfortunately, this technique cannot be directly applied to our problem. The reason is that only positive examples are available (all sessions are labeled *accept*), therefore, the minimal automaton consistent with our training set accepts all possible sequences of message types. To avoid such an over-generalization, we need to restrict the hypothesis space using domain-specific knowledge.

Augmented Prefix Tree Acceptor (APTA)

As mentioned previously, the input to the state machine inference phase is a set Λ of message sequences S_i, where each S_i represents one observed application session. In a first step, we can represent the set Λ as an *augmented prefix tree acceptor* (APTA) T [28].

An APTA is an incompletely-specified deterministic finite state automaton (DFA), with a state transition graph that is a tree. The root of the tree is the initial state of the DFA, and each branch represents an application

session. As an example, consider that we observe two application sessions of the Agobot malware. The sequences of the message types of these two sessions are: (**login**, **bot.dns**, **bot.status, mac.logout**) and (**login**, **mac.logout**, **login**, **bot.status**, **bot.dns**, **mac.logout**). The APTA for this example is shown in Figure 4.2. States of T may be labeled either *accept*, or *reject*. In our example, all states are labeled *accept* (marked with an "A") because any prefix of a valid protocol session is also a valid session (if this were not the case, only the two final states would be *accept* states, and other states would be unlabeled). T is an incompletely-specified acceptor DFA that accepts only the sequences in the training set (and their prefixes). For any other sequence, the result is unspecified.

The APTA T is used as a starting point to find the protocol state machine. This is done by finding the *minimal* DFA that is consistent with T. To find such a DFA, we can leverage existing algorithms (such as Exbar [75]) that start from T and successively merge pairs of states. Clearly, states with different labels can never be merged. However, in our training set, all states of T are labeled *accept*. Thus, the result of directly applying an existing algorithm would be an over-general DFA with only a single state. To address this problem, we introduce an algorithm that assigns different labels to the states of T (discussed in the next Section 4.2.3). This restricts the possible merges, since only states with the same label may be merged. Finally, we use Exbar to obtain a minimal DFA that is consistent with that labeling, as discussed in Section 4.2.4. This minimal DFA represents our state machine.

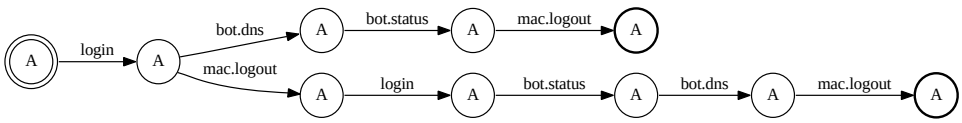


Figure 4.2: APTA for the Agobot example.

State Labeling Algorithm

The goal of the state labeling algorithm is to find states in the APTA that are different. By assigning different labels to these states, we can prevent them from being merged. To this end, we leverage the observation that a common pattern in application layer network protocols is that a message or a sequence of messages must be sent before the server can perform certain actions. As an example, in the Agobot command and control protocol, a login is required before other commands become available. In SMB/CIFS,

a "TREE CONNECT" operation must be performed to connect to a share before file operations can be issued. In addition, certain commands may lead the server away from a state where it can perform these actions. For instance, a logout command in Agobot or a "TREE DISCONNECT" in SMB/CIFS make previously available commands impossible to execute.

Our state labeling algorithm attempts to identify states that represent similar application conditions. That is, we attempt to identify cases in which an application can process similar commands, based on the sequence of messages that it previously received. To this end, we extract simple patterns from the observed application sessions. These patterns have the form of regular expressions on sequences of message types. More precisely, each pattern has the form:

$$.*r(a_1|..|a_j)*, \quad (r, a_1, .., a_j \in M) \tag{4.2}$$

where "$*$" means zero or more repetitions of the previous term and "." matches any message type. We call such a pattern a prerequisite.

A prerequisite requires that, for the server to be in a state where it can meaningfully process a message of type m, it must first receive a message of type r, optionally followed only by messages in the set $A_r = a_1, .., a_j$.

The message of type r is a message that always occurs before m. That is, in all application sessions, a message of type r was found before m. This is to capture the case where a connect or login message must be sent before message m. Note that Equation 4.2 allows messages of any type to occur before r (including more occurrences of r).

The set of optional messages A_r is the set of all messages that, in at least one application session, have been seen between the last occurrence of r and a message of a type m. In the Agobot example, *login* always occurs before messages in the set $M_{login} = \{bot.dns, bot.status, mac.logout\}$. Furthermore, only *bot.dns* and *bot.status* occur between the last *login* and messages in the M_{login} set. Therefore, the prerequisite $.^*login(bot.dns|bot.status)*$ will be added for all three message types in M_{login}. We provide a more precise description of our algorithm for inferring prerequisites in the following section.

4.2.4 Prerequisites Inference Algorithm

In this section, we detail the algorithms that we use to infer the prerequisites described in Section 4.2.3. Algorithm 1, together with Algorithms 2 and 3, computes prerequisites in the form of Equation 4.2. A prerequisite for message type m is specified by the tuple $\langle m, r, A_r \rangle$. To implement the hitting set heuristic and to infer prerequisites in the form of Equation 4.3, we replace the call to the `get_required` function in Algorithm 1 with a call to `get_required_sets`

(Algorithm 4).

The `hitting_set` function in Algorithm 4 finds a solution to the minimum hitting set problem for sets in Y. That is, it finds the smallest set ρ of message labels such that $\rho \cap y \neq \emptyset, \quad \forall y \in Y$. That is, we want to find the minimum number of message types such that at least one type is present on any path from the start state to a state where m is received. The minimum hitting set problem is NP-complete [73], but we impose the restriction $|\rho| \leq K$ and solve it by exhaustive search. The constant K was set to 5 in our experiments; we do not expect a protocol specification to have more than 5 message types leading to the same state transition, or our clustering algorithm to be so inaccurate as to split messages of a single message type into more than 5 clusters. Since `get_required_sets` returns a set of sets, Algorithms 1 and 3 must also be modified accordingly.

Algorithm 1 `infer_prerequisites`

Input: The set of message types M. The training set composed of n application sessions $S_1, .., S_n$, where $S_i = \sigma_{i,1}, ..\sigma_{i,|S_i|}$ and $\sigma_{i,j} \in M$.
Result: the set of prerequisites P.
  for each $m \in M$ **do**
    $R_m = get\_required(m)$
  $R = \bigcap_{m \in M} R_m$
  $P = \emptyset$
  for each $r \in R$ **do**
    $M_r = m \in M | r \in R_m$ // the set of messages which share requirement r
    $A_r = get\_allowed(r, M_r)$
    for each $m \in M_r$ **do**
      add $\langle m, r, A_r \rangle$ to P

Algorithm 2 `get_required`

Input: A message type $m \in M$. The training set $S_1, .., S_n$.
Result: $R_m \subset M$, the set of msg. types required before m
  $Y = \emptyset$
  for each instance $\sigma_{i,j}$ of m in the training set **do**
    $y = \{\sigma_{i,1}, .., \sigma_{i,j-1}\}$
    // the set of message types found before $\sigma_{i,j}$ in S_i
    add y to Y
  return $\bigcap_{y \in Y}$

Once all prerequisites have been computed, we label each state q of T with the set of message types that are allowed as input in that state. A message

Algorithm 3 `get_allowed`

Input: A message type $r \in M$. A set of message types $M_r \subset M$. The training set $S_1, .., S_n$.
Result: $A \subset M$, the set of message types allowed after r

```
A = ∅
for each instance σ_{i,j} of m in the training set do
    consider the application session S_i = σ_{i,1}, .., r, σ_{i,k+1}, .., σ_{i,j}, .. // σ_{i,k} = r
    is the last occurrence of r in S_i before σ_{i,j}
    a = {σ_{i,k+1}, .., σ_{i,j-1}}
    add a to A
return A
```

Algorithm 4 `get_required_sets`

Input: A message type $m \in M$. The training set $S_1, .., S_n$.
Result: $R_m \subset 2^M$, the set of requirements for m

```
R_m = ∅
compute Y as in get_required
while (y ≠ ∅  ∀y ∈ Y) do
    ρ = hitting_set(Y)
    add ρ to R_m
    set y to y − ρ for each y ∈ Y
return R_m
```

type m is allowed in a state q if the sequence of message types leading to q exactly matches all prerequisites for m (since T is a tree, there is only one path leading from the root to state q). The labeled state tree for the Agobot example is shown in Figure 4.3.

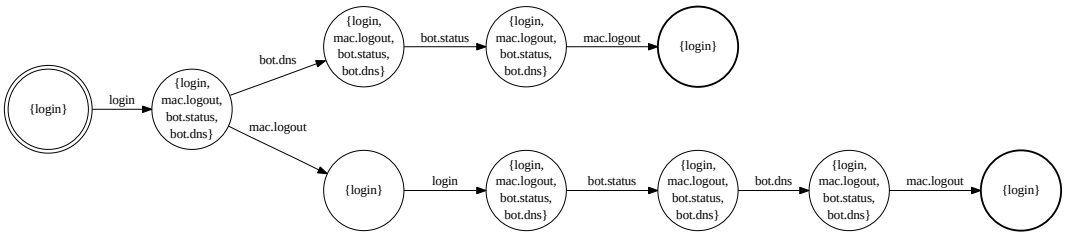


Figure 4.3: Labeled State Tree for Agobot example.

Hitting set heuristic. The technique described above fails to detect a prerequisite for a message m when there are multiple, alternative paths to a state where m is allowed. As an example, in an SMTP session, either $HELO$ or $EHLO$ may be the first message, but one of these two is required before a $RCPT\_TO$ message may be sent. Furthermore, even if there is only one login message type according to the specification of a protocol, this message type may be split into several clusters by our tool (for instance, when the login message can have significantly different, optional parameters). To be able to handle such situations, we generalize Equation 4.2 and infer prerequisites in the form:

$$.*(r_1|..|r_k)(a_1|..|a_j)* \tag{4.3}$$

That is, we require only one of the messages $r_1, .., r_k$ to be present in a session before m can be received. To infer such prerequisites, we generalize the algorithm described above, as detailed in Section 4.2.4.

End-state heuristic. In addition to the techniques described previously, we also use a simple heuristic to detect end-states in the protocol state machine. It is common for application layer protocols to have one (or more) message types that request the termination of the protocol session. To detect those message types, we simply look for messages that, throughout all observed application sessions, appear only last in a session. In T, we mark all states that follow such messages as end-states, setting their label to the empty set (since no messages of any type are allowed in those states).

Exbar

Based on a state tree (APTA) that is labeled by our heuristics, we can now infer a minimal DFA. The problem of deriving the smallest DFA consistent with a labeled training set is an important problem in grammar inference, and has been proven NP-complete by Gold [57]. Both approximate and exact algorithms have been proposed to solve it (see [28] for an up to date survey of existing techniques). Exbar [75] is the state-of-the-art, exact algorithm for minimal consistent DFA inference. Thus, we apply Exbar to the state tree T, once it has been labeled by the previously-discussed algorithm. The result is the generalized protocol state machine.

The state machine for the Agobot example (Figure 4.3) is shown in Figure 4.4. Here, we have once more replaced the state labels with *accept*. Once this generalization phase is complete, we assume that any sequence of message types that leads to an unspecified state transition is not a valid protocol session. Therefore, we add an additional *reject* state (not shown in Figure 4.4) to the state machine, and make it the endpoint for all unspecified transitions. In the state machines shown throughout this work, this *reject* state is also omitted for ease of presentation. In Section 4.3.5, we evaluate the performance of Exbar on our datasets.

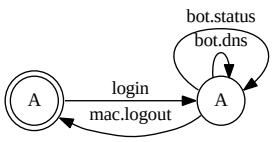


Figure 4.4: Inferred state machine for Agobot example.

4.2.5 Creating Fuzzing Specifications

As a final step, our tool is able to export the state machine and message format descriptions to the XML-based protocol specification format used by the Peach fuzzing platform [100].

Fuzzing is a black-box software testing technique that is based on the principle of feeding an application with random input, while observing crashes or other undesired behavior [86]. To achieve better code coverage of the tested application, advanced fuzzers (such as Peach [100]) generate test data based on the grammar of the file formats or network messages understood by the target application (we refer the reader to [110] for a recent overview of fuzzing

techniques). Unfortunately, without any knowledge of the protocol state machine, a (stateful) network protocol cannot be effectively fuzzed. The reason is that a server will typically discard messages with types that are not acceptable in the current protocol state. Thus, stateful protocol fuzzers such as Snooze [15], additionally use a specification of a protocol state-machine to reach deep protocol states.

Prospex is able to automatically extract a grammar for protocol messages, as well as a protocol state machine; stateful, grammar-based fuzzing is, therefore, a natural application. We chose to leverage an existing tool for fuzz testing, and selected Peach [100], mainly because it is an open-source project under active development, and it provides most of the required features. The main limitation of Peach was the limited support for statefulness. To address this limitation, we contributed to the design and development of improved statefulness features for Peach, which have been integrated into release 2.2. To use Prospex specifications for fuzzing, we simply translate the message formats and state machine extracted by our tool to Peach XML. The Peach fuzzing framework then provides all the mechanisms necessary to perform stateful fuzz testing of real-world applications that implement the target protocol.

4.3 Evaluation

We have tested our tool on a number of applications that implement stateful, application-layer protocols. In particular, we chose a bot protocol, SMB, SMTP and SIP, as they are all stateful protocols implemented in complex server applications that are widely deployed. Because of a limitation of our current system (our taint tool only runs under Linux), we only analyzed server programs that are available to us as Linux binaries. However, this does not represent a general limitation of our approach.

The quality of the specifications produced by our tool is limited by the quality and variety of the data used to train it. As for all trace-based approaches, our system cannot learn behaviors that do not occur in the training data. For the purpose of this evaluation, we trained our system using small datasets that covered a meaningful subset of the functionality of each protocol, such as using SIP to perform phone calls or SMB to browse shared files and folders. The goal of this evaluation is to demonstrate that, provided suitable training data, we can produce accurate state machines and message formats for complex, stateful protocols. Furthermore, our tool can help a human malware analyst to understand a previously-unknown malware protocol. Finally, we show that we can automatically generate fuzzing specifications that are subsequently used to find security vulnerabilities in real-world server programs.

4.3.1 State Machine Inference

We applied our system to one malware protocol (Agobot C&C), two text-based protocols (SMTP, SIP), and one binary protocol (SMB). For each protocol, the system created state machines that ranged from four states (Agobot) to 13 states (SMB).

Agobot. We selected the well-known Agobot as the malware example. The reason is that Agobot implements a custom C&C protocol and is representative for a whole family of bots, for example, Phatbot and Forbot [63]. For C&C, Agobot uses a text-based protocol that resembles the IRC protocol. However, the malware author has extended the protocol by incorporating additional command keywords. These commands typically trigger malicious bot behavior, for example, spreading via scanning and remote exploits, relaying traffic, or downloading binaries from the web. The automatic analysis of bots can provide valuable information about the malware's communication protocol and the available commands, which can help an analyst to better and faster understand the bot's internal functionality.

For our experiments, we set up an IRC server and configured an Agobot instance such that the bot connected to a local IRC channel, listening for commands. We then mimicked a bot herder, issuing several commands to the bot while monitoring it. We then ran our tool on the collected traces and obtained the state machine in Figure 4.5. Moreover, the system has correctly produced format specifications for the commands that we sent to the bot. Of course, for a more realistic scenario, it would be desirable to trace the bot while a real bot herder is issuing commands.

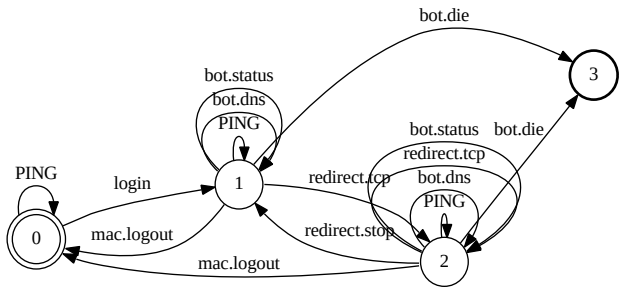


Figure 4.5: Inferred state machine for Agobot command and control protocol.

SMTP protocol. As an example of an application that implements a stateful, text-based protocol, we have chosen the widely-deployed mail transport agent `sendmail` (version 8.13.8). The application implements SMTP (Simple Mail Transfer Protocol). To infer the state machine, we first recorded 16

SMTP application sessions on our group's e-mail server. We then replayed this small training set to a sendmail server instance that we were tracing. Figure 4.6 shows the SMTP state machine that our system inferred. It can be seen that two different message types were created for each the MAIL FROM and RCPT TO commands. This is due to the fact that those mail clients that initially send an EHLO command are typically using extended options (additional flags and keywords) in subsequent SMTP commands (for example "ORCPT" in the RCPT TO command). Because of the resulting, different message formats, our system distinguishes between simple and extended versions of these SMTP commands.

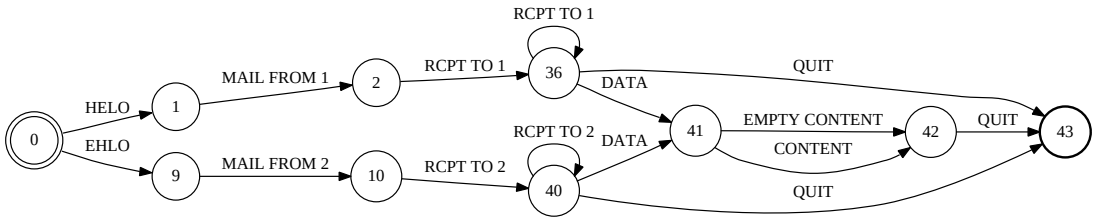


Figure 4.6: Inferred state machine for the SMTP protocol.

Server Message Block (SMB) protocol. As an example of a complex, stateful, binary protocol, we have chosen SMB/CIFS. In our experiments, we used version 3.0.26a of the `Samba` software suite, and traced the `smbd` daemon while using the `smbclient` utility to browse shared directories, performing common operations such as writing, reading, and deleting files and directories. Using this setup, we produced a training set of 31 recorded sessions. The state machine inferred from the SMB dataset can be seen in Figure 4.7. The login sequence leading to State 3 is clearly visible. After that, when the DFS (distributed file system) option is enabled, the client first connects to the IPC$ share to obtain a DFS referral for the requested share. Otherwise, the client directly connects to the requested share in State 6. In this state, most of the file system operations are available. Operations on a file are performed by opening the file, reading or writing, and finally closing it (States 8-10). According to this state machine, only one file may be opened at any given time. Of course, this is not a limitation of the SMB/CIFS protocol, but a peculiarity of how the `smbclient` tool employs it. In fact, `smbclient` always closes a file before operating on the next one. Finally, notice that States 11 and 12 are artifacts of our system, caused by the limited variety of the training set. The reason is that, in the training set, "DELETE" and "QUERY DISK" requests

were always preceded by find requests. This highlights the dependence of our system on the quality and variety of data in the training set.

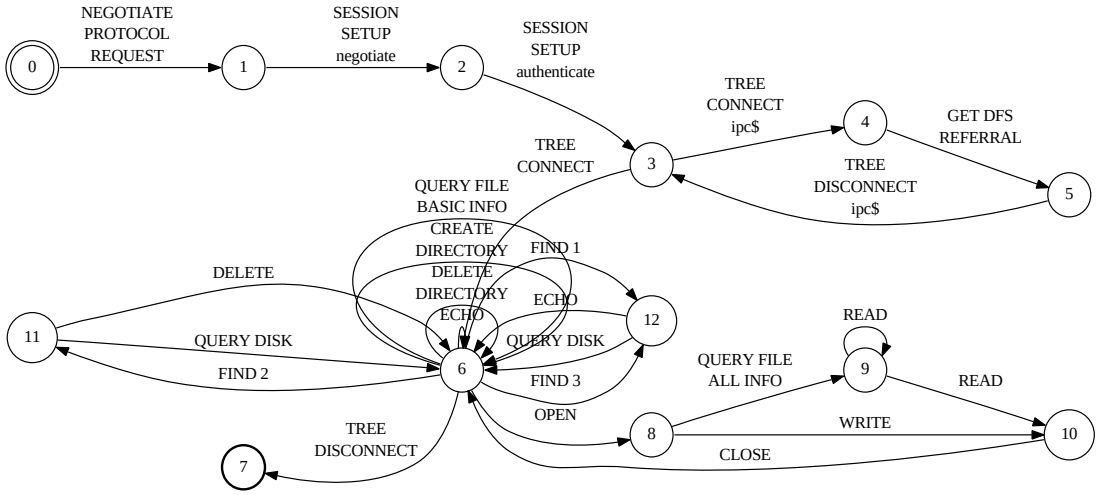


Figure 4.7: Inferred state machine for the SMB/CIFS protocol.

Session Initiation Protocol. The text-based Session Initiation Protocol (SIP) [105] is used for setting up and controlling communication connections. In our experiment, we traced the well-known, open-source telephony server `Asterisk` [12] (version 1.4.0), which is typically used as part of a Voice-Over-IP (VOIP) infrastructure. Our test environment consisted of three (virtual) machines, one of them running the Asterisk server, while the two other additional machines served as clients. In our test configuration, we created two SIP peers, each including a voice box. The client machines had installed either CounterPath Corporation's proprietary softphone `X-Lite` [125] (version 2.0) or the open-source softphone `Ekiga` [48] (version 2.11). To simulate different client behavior, the softphones were configured to either automatically answer incoming calls using a built-in auto-answer feature, automatically answer after a short delay (e.g., permit ringing) by using a GUI automation tool [77], or to not answer at all (triggering the voice box). Then, we initiated a number of phone calls to these peers by using a softphone, including simultaneous phone calls on multiple lines. These training calls were used by our tool to generate protocol specifications for the observed call initialization use-cases. Figure 4.8 shows the SIP state machine.

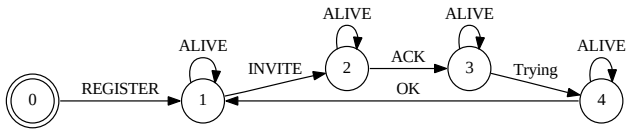


Figure 4.8: Inferred state machine for the SIP protocol.

4.3.2 Quality of Protocol Specifications

To evaluate the quality of the protocol specifications inferred by Prospex, we need to assess their *soundness* and *completeness*. For the purpose of this thesis, we say that a protocol specification is *complete* if it is not overly restrictive. That is, the protocol specification accepts (parses) valid protocol sessions. Conversely, we say that a protocol specification is *sound* if it is not overly permissive. That is, it rejects invalid protocol sessions.

As for all trace-based approaches, the *completeness* of inferred specifications is limited by the variety of behaviors observed in the training data. Therefore, we evaluate *completeness* with respect to the subset of protocol functionality that was exercised during training. For instance, for the SMB protocol we take into account the browsing of shared files and directories, but not the use of printing services.

Protocol Completeness

In the first step, we demonstrate that our protocol specifications are complete. To this end, we used our protocol specifications to parse real-world network traces (that were not part of the training data) of SMTP, SMB, and SIP traffic . Note that this shows the *completeness* of both the message formats and the state machines inferred by our tool. The reason is that, for successful parsing, Prospex has to correctly determine the format of each individual message and recognize their correct ordering.

For parsing, we used an enhanced version of the single message parser presented in [123], which includes support for multiple states. Each result was achieved by using the value of k where the generalized Dunn index reaches its maximum (as discussed in Section 4.2.2).

SMTP results. For SMTP, we recorded our group's `Postfix` [122] e-mail server traffic (incoming traffic on port 25) during a period of four weeks. Then, we split the dumps into TCP sessions and parsed them, using the automatically-generated SMTP protocol specification with $k = 10$.

Out of 31,903 total sessions, we were able to parse 29,832 sessions (93.5%)

successfully. We found that the remaining 2,071 sessions (6,5%) were all using TLS encryption, which we cannot handle properly as one of the limitations of our system is its inability to handle encrypted traffic. This shows that our system can fully parse (unencrypted) real-world traffic, generated by a number of clients and sent to a different mail server implementation than the one used to infer the protocol specifications.

SMB results. To test our SMB protocol specification, we used `smbclient` to browse shared directories on both Windows and Linux servers, and recorded 80 sessions. For $k = 23$, only 8 sessions fail to parse. We examined these sessions and determined that parsing fails because of (1) error conditions not present in the training set (such as attempting to read from a non-existing file), (2) writing of long files; a limitation of our training set was that only short files were written, small enough to be sent in a single write message, and (3) insufficient generalization of the state machine (as discussed in Section 4.2.3, states 11 and 12 in Figure 4.7 are artifacts of our system).

SIP results. For generating SIP traffic, we used `X-Lite` [125] to initiate phone calls to different SIP peers in a laboratory Voice-Over-IP environment. We recorded a set of 80 SIP sessions during these calls. Using the state machine for the indicated optimum of $k = 6$, we were able to parse all of the traffic successfully.

Protocol Soundness

In the next step, we evaluate the *soundness* of the inferred specifications.

Soundness of the Message Formats. To show that our protocol specifications are not overly permissive, we first need to demonstrate that the inferred message format for each cluster is not too general, as it should neither parse arbitrary messages nor messages that have a different type. To this end, we compute the *message format specificity*.

Initially, we manually label every message in the training set with its actual message type (such as "HELO" or "TREE CONNECT"). Then, we mark each cluster with the message label of the majority of its messages (while, ideally, all messages in a cluster would share the same label, this step is necessary if different message types are incorrectly clustered together). In the next step, we select a certain cluster. Then, we find all training set messages that are *not* labeled with the label of this cluster. These messages are then parsed with the cluster's message format. When the clustering phase was successful and the message formats are *sound* (not too general), we would expect most parsing attempts to fail. This step is then repeated for all clusters. Finally, we calculate the ratio r of successfully parsed messages to the number of total

parsing attempts. The format specificity is then computed as $1 - r$.

For the value of k where the Dunn index reaches its maximum, our tool achieves a message format specificity of 1 for all four datasets (Agobot, SMTP, SMB and SIP). This means that (a) no cluster contains messages of multiple types, and (b) the message format for a cluster does *not* parse any messages of a different type.

Soundness of the State Machines. The next goal is to evaluate the *soundness* of the inferred state machines. To do so, we require a reference state machine for each protocol. We created these reference state machines manually, using information from specification documents (if available), and integrating it with our own testing and reverse engineering efforts. Clearly, our tool cannot learn parts of the protocol that were not exercised in our training data, so we do not include them in the reference state machine. We then performed n (for $n = 50,000$) random walks over our inferred state machine, generating n sessions that our specification considers valid. These sessions were fed to the reference state machines. The idea is that an overly permissive state machine would generate sessions that are not recognized by the true protocol. We found that, for all four protocols, all sessions were accepted. Thus, our inferred state machines are sound.

4.3.3 Comparative Evaluation

The previous section has shown that our techniques allow us to infer accurate specifications. However, there exist alternative approaches to infer an automaton from positive examples only. One popular approach is based on the minimum message length (MML) principle [116]. According to this principle, a solution should minimize the length of the description of the state machine together with the dataset it tries to account for. Since minimizing this quantity is an NP-complete problem, several approximate algorithms have been proposed to attempt to solve it. The sk-strings algorithm [98] is one such algorithm, which has previously been applied to mine specifications [8] of a software component from program execution traces. The authors of [98] also introduced the beams algorithm [104], which outperforms sk-strings. We obtained the implementations of both algorithms from the authors [1].

To compare the performance of previous techniques with our system, we leverage the *precision* and *recall* metrics introduced in [81]. *Precision* is closely related to the *soundness* metric described in the previous section. It measures the ratio of sequences generated by a random walk over the inferred automaton that are accepted by the reference automaton. Conversely, *recall* measures the ratio of sequences generated by the reference automaton that are accepted by

the inferred automaton. It is a measure of *completeness*. For details on how these metrics are computed, we refer the reader to [81].

Results are shown in Table 4.1. For sk-strings, we show results using the OR heuristic (which was the best performer in [98]) and the AND heuristic (which is evaluated in [81]). We run sk-strings with tail lengths of $1, 2$ and 3 and $s = 0.5, 0.75, 1$, and select the best solution based on MML. Similarly, we run the beams algorithm with beam widths of $1, 2, 4, 8, 16$ and 32.

Prospex clearly outperforms previous tools on all four datasets. The sk-strings algorithm using the OR heuristic does not produce *sound* results on most datasets ($P \simeq 0.12$). Sk-strings with the AND heuristic and beams produce better results. However, only Prospex consistently provides a *sound* state machine ($P = 1$). Previous algorithms over-generalize on at least one dataset. Somewhat surprisingly, neither sk-strings nor beams succeed in learning a state machine for the rather simple Agobot dataset.

	Agobot		SMTP		SMB		SIP	
	P	R	P	R	P	R	P	R
Prospex	1	1	1	1	1	.58	1	1
beams	.56	1	.89	1	1	.50	1	1
skstrings(and)	.79	.20	1	.88	1	.30	1	.01
skstrings(or)	.11	.92	.11	1	.12	.62	1	1

Table 4.1: Precision (P) and Recall (R) of inferred automata with respect to reference automaton.

4.3.4 Robustness of k

In this section, we examine the robustness of our tool to the choice of the parameter k, the number of message types (clusters) that need to be provided as input to the clustering algorithm. The number of clusters is a trade-off between *soundness* and *completeness*. With too many clusters, we expect the inferred model to be *sound* but over-specific, and therefore *incomplete*. Conversely, with too few clusters, we expect a *complete* but over-permissive (*unsound*) model. We wish to demonstrate that, for a relatively large range of values for k, Prospex produces a *sound* and *complete* protocol specification. Therefore, we measure the following two properties over a range of values for k:

Parsing success rate. To compute this property, we use the generated protocol specification for each k to parse network traces of real-world traffic, and measure the ratio of successfully parsed sessions to the total number of

sessions. This is a measure of *completeness*, so we expect it to to *decrease* as k increases.

Message format specificity. This is the property introduced in Section 4.3.2. It is a measure of *soundness*, so we expect this property to *increase* as k increases. The reason is that more clusters imply fewer messages per cluster, so the message formats for each cluster become more specific.

In Figure 4.9, we show both properties for the SMTP and SMB protocols over a range of values for k. In addition, the figures show the generalized Dunn index, which, as described in Section 4.2.2, is used to choose the value of k. The Dunn index is normalized to the $(0, 1)$ range. It can be seen that the maximum of the Dunn index correlates with the optimal choice of k with regard to parsing quality and message specificity. This confirms that the Dunn index is a good predictor to select k, resulting in protocol specifications that are specific *and* successful in parsing.

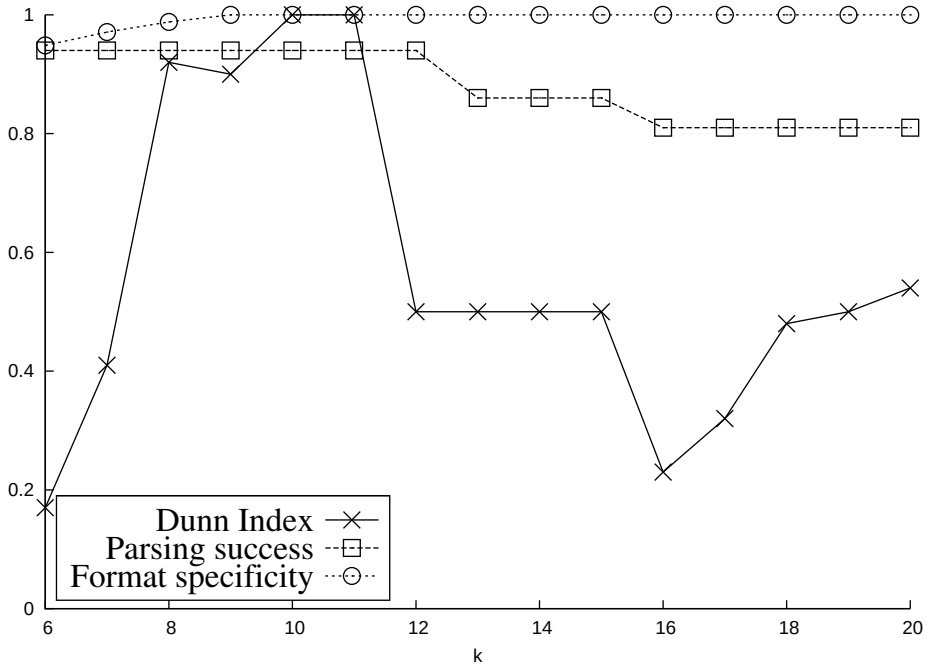
Specifically, in Figure 4.9(a), the Dunn index reaches its maximum at $k = 10$. This corresponds to the optimal parsing results and a message format specificity of 1, which demonstrates the suitability of our approach. Similarly, Figure 4.9(b) shows that values of k around the choice of 23 (between 22 and 25) produce high parsing results, while having a message format specificity of 1.

4.3.5 Exbar Performance

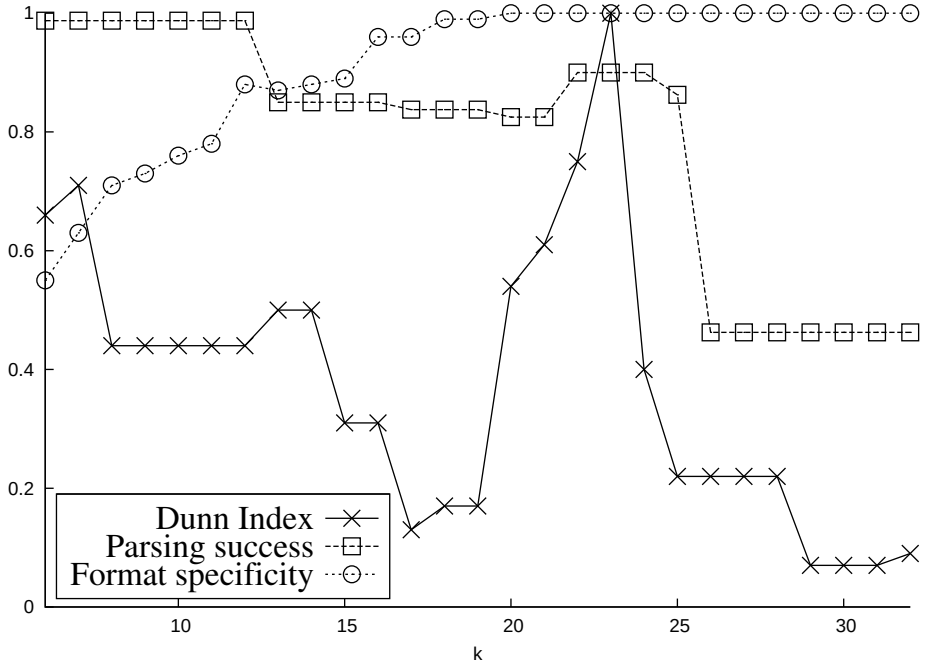
We allowed Exbar to run for up to 5 minutes for each value of k, and were able to infer state machines of up to 25 states (starting from state trees with over 200 states). For the optimal values of k, selected using the Dunn index, Exbar terminated in less than 0.03 seconds for all of our test cases. Nonetheless, for very large values of k, Exbar might not terminate in the allotted time. In such cases, approximate algorithms could be used instead [28], but a better solution is to increase the size of the training set, since DFA learning is harder when the training set is sparse (as was empirically demonstrated by the Abbadingo competition [76]).

4.3.6 Fuzzing Experiments

As discussed in Section 4.2.5, our system can be used to create input specifications for the Peach fuzzer. This allows the fuzzer to use the automatically inferred state machine while fuzzing the message's field values according to the inferred field types.



(a) SMTP



(b) SMB

Figure 4.9: Robustness of k

SMB fuzzing. We automatically converted the protocol specifications generated by Prospex for the SMB/CIFS protocol into more than 2,100 lines of Peach XML. This allowed us to use Peach to fuzz the latest versions of the Samba server and the Windows XP SMB/CIFS implementation. Unfortunately, we did not find any vulnerabilities in these programs. This may be due to the fact that both are mature, widely-deployed services that have been patched for many vulnerabilities related to input validation errors in the past. Therefore, we target an older version of Samba (version 3.0.2a, which is subject to an arbitrary file access vulnerability [41]) to validate our tool. By searching the network traces captured during the fuzzing run, we were able to verify that the fuzzer had been able to find the vulnerability. More specifically, the fuzzer downloaded the `/etc/passwd` file, which should not have been accessible through the SMB service. The `/etc/passwd` file is commonly used to test for file traversal vulnerabilities on UNIX variants. Notice that successful exploitation of this vulnerability requires the fuzzer to navigate deep into the protocol state machine (to State 10 in Figure 4.7). Furthermore, this attack is only possible because the message format inference has automatically identified a field in the "OPEN" message as a file name. Peach only makes directory traversal attempts on fields that are marked as file names.

SIP fuzzing. We generated fuzzing specifications for SIP and ran the Peach fuzzer on the `Asterisk` server. After checking the fuzzer logs, we noticed that sending the server an "OK" message with status code "0" triggered a segmentation fault that crashed `Asterisk`. This could be used to launch a denial-of-service (DOS) attack. For the server to successfully accept and parse the message that crashes it, the fuzzer has to navigate successfully to State 4 in the SIP state machine (shown in Figure 4.8). Finding vulnerabilities of this kind by using stateless fuzzing is practically infeasible. Even though we found this to be a known vulnerability [13] that has already been addressed in the newest versions of Asterisk, it shows that our system is capable of creating fuzzing specifications that can be used to automatically find vulnerabilities in real-world software.

4.4 Summary

In this chapter, we introduced Prospex, a system for automatically reverse-engineering application level network protocols. We presented novel approaches, algorithms and techniques for automatically extracting message formats, identifying message types and inferring a protocol state machine. In an experimental evaluation with real-world server software, we were able to show that we can produce accurate protocol specifications. Furthermore, we extended

our system to produce specifications for a stateful fuzz testing tool, which allowed us to automatically find several security vulnerabilities.

Chapter 5

Related Work

Each of the three systems described in this thesis is related to a different field of security research. To reflect this, the sections in this chapter individually highlight the related work for each of these systems.

5.1 Social Network De-anonymization

Clearly, de-anonymization of privacy-sensitive data is not a new concept. Research initially focused on anonymization and de-anonymization of network level data. For example, work by Pang et al. [96] presents techniques for anonymizing network packet traces with the intent of sharing data between researchers. As a reaction to anonymization research, Coulls et al. [36] introduced approaches that allow an attacker to de-anonymize network traces, and recover sensitive data on network topologies.

Information Leakage and Social Networks Due to the popularity of social networks and the large amounts of sensitive data they store, the focus of de-anonymization research has recently extended to this area. Several publications have shown that seemingly non-sensitive data from publicly available sources can be used to recover private information about individuals. For example, Griffith and Jakobsson [58] use public records to infer individuals' mothers' maiden names, and Heatherly et al. [62], as well as Zheleva and Getoor [128], show how public data provided by social networks can be used to infer private information.

In addition, several publications have analyzed and measured features of social networks that are privacy-related. For example, Mislove et al. present a measurement study on social networks [87] while Bonneau and Preibusch evaluate the privacy settings and policies of a large number of social networks in [21]. Closely related to this context, several recent papers focus on scenarios for malicious activity directed against social networks. For example, Jagatic et al. evaluate the success rates of phishing attacks [68], and Brown et al.

discuss context-aware spam [25]. Another study [20] by Bilge et al. shows the feasibility of automated identity theft attacks in social networks.

Attacks on Browsing Privacy The de-anonymization scenario presented in this work leverages a browsing history stealing technique that is based on CSS and has been known since the year 2000. This technique has been discussed in several browser bug reports [106, 16, 24], and has been shown to be practical for targeted phishing attacks by Jakobsson and Stamm [83]. Despite its malicious potential, browser history stealing has not lead to any changes in browser software.

There are also other techniques that aim at exposing private browsing information. Several systems use timing properties to recover private information. For example, Felten and Schneider show an attack on web browsing history by analyzing caching operations [50], while Bortz and Boneh [23] use timing attacks to recover private information from web applications.

De-Anonymization of Social Networks Narayan and Shmatikow have shown that statistical methods can be applied to de-anonymize micro-data by cross-correlating multiple datasets [89]. They extend their approach to social networks in [4], and prove that it is possible to de-anonymize members by mapping known, auxiliary information on the (social) network topology.

In [44], Diaz et al. present a de-anonymization approach that uses information gained from observing communication patterns between social network members.

In contrast to existing work, our attack uses only information from a single social networking site, and combines it with the browsing history of a user to identify individuals. Furthermore, our attack is highly practical, and works effectively in the real-world. In fact, as we demonstrate in Chapter 2, the attack has the potential to affect the privacy of millions of registered social network users.

5.2 The Online Adult Industry and Cybercrime

Little academic information is available about the online adult industry, yet it consists of thousands of web sites that generate a revenue of billions of dollars every year. Many publications that analyze general web security issues have been published in recent years. For example, Wang et al. developed client honeypots to detect and capture web-based malware samples [117].

Existing work on web-based threats often targets specific types of malware. For example, Moshchuk et al. provide an analysis of web-based spyware by

utilizing a web-crawling system [88]. Provos et al. focus on analyzing technical exploitation details [102]. The authors did mention the adult industry, however, the scope of their work is limited to drive-by downloads found on adult web sites and no other aspects were studied.

Several studies show parallels and draw connections between malicious Internet activity and the underground economy. For example, Provos et al. provide technical details on how cyber-criminals use web-based malware to their advantage [103]. The aspect of an underground economy that is fuelled by financially motivated cyber-criminals is highlighted by Franklin et al. [52]. In a recent paper, Holz et al. study the structure and profits of keyloggers [64].

To the best of our knowledge, our study (presented in Chapter 3) is the first that combines an economic analysis of the online adult industry with a security analysis from a technical and a cyber-crime perspective.

5.3 Protocol Reverse-Engineering

Since proprietary, closed protocols started to emerge on the Internet (e.g., such as the OSCAR protocol, used by ICQ and AIM [53]), there has been interest to reverse engineer these protocols with the goal of providing free, open-source alternatives. For example, Samba [107] aims to offer a free implementation of the Microsoft SMB/CIFS file sharing protocol. Although popular, protocol reverse engineering is still a largely manual task. It is tedious and labor-intensive.

Session replay. The first automated protocol analysis approaches emerged within the context of honeypots. In order to capture malicious code that delivers its payload after a series of interactions over the network, researchers started working on systems that could replay application sessions automatically. To this end, systems such as RolePlayer [39] and ScriptGen [78, 79] analyze network traffic and attempt to generalize the traces so that correct replies can be generated to new requests. Although useful, the main focus of these systems is not to reverse engineer and understand the entire protocol that is analyzed, but to continue the interaction with a malicious program long enough so that its payload can be intercepted. Hence, these systems only focus on the protocol format to the extent necessary for replay, in particular, on the recognition of fields that contain cookie values or IP addresses. ScriptGen is the only previous work that attempts a kind of state machine inference. However, the proposed technique is limited because no generalization takes place. Thus, the resulting state machine is a tree, similar to the APTA in Section 4.2.3, which can only parse sessions identical to those previously observed.

Protocol analysis. Reacting to the emerging need for the automated analysis and reverse engineering of entire protocols, systems were proposed that attempt to discover the complete protocol format. In [19], the authors propose to apply bio-informatics techniques (such as sequencing algorithms) to network traffic. The goal of the system is to identify protocol structure and fields with different semantics. In [38], an improved technique was proposed that uses recursive and type-based clustering instead of byte-wise alignment. The advantage of such network trace-based approaches is that it is straightforward to gather large datasets for training. Their shortcoming is that network traces provide a limited amount of information and no information on field semantics, making classification of messages into types extremely challenging.

Recently, four approaches were presented that propose to extract protocol information by observing the execution of a program while it processes input messages [29, 80, 123, 40]. However, these systems focus on reversing message formats, and leave state machine inference for future work.

Specification mining. Automatically extracting a protocol state machine from a set of observed protocol interactions is related to the problem of extracting temporal specifications for software components (such as API or method call sequences) from program traces [8, 120, 49, 126, 7]. Here, we focus on how work in this field performs state machine inference. In [120], each relevant event is directly mapped to a state in the automaton. This approach is not suitable for protocol inference, where typically there exist message types that are valid in many different states (such as the "ALIVE" message in Figure 4.8). In [8], the sk-strings algorithm is used to infer a state machine. As discussed in Section 4.3.3, this algorithm does not provide acceptable performance for most of our datasets. Other works [49, 126] only infer properties conforming to simple patterns, such as alternation between two events. Finally, [7] uses an active learning approach, and learns state machines using the $L*$ algorithm [9]. The $L*$ algorithm requires a teacher that can answer membership queries. In [7], the teacher is implemented using model checking techniques. This approach cannot be easily applied to network protocol inference.

Automated white-box testing. Performing fuzzing of an application based on an automatically reverse-engineered network protocol is related to concolic testing [108], white-box fuzzing [55, 54], and related approaches [31, 30]. These techniques leverage symbolic execution of a target application to generate test cases that provide better code coverage than black-box fuzzing approaches. They have been successfully applied to a wide variety of software such as Linux file system implementations [127], the entire GNU coreutils [30], and the JavaScript interpreter of Internet Explorer 7 [54]. To the best of our knowledge, none of these tools have yet been applied to real-world implementations

of stateful network protocols. Also, we believe that these techniques are complementary to ours. That is, symbolic execution could be added to Prospex to overcome some of its limitations, such as its inability to express arbitrary relationships between protocol fields. Conversely, the protocol specifications generated by Prospex could be used to enhance white-box fuzzing of complex network applications by leveraging a grammar-based constraint solver [54].

Chapter 6

Conclusion

In this thesis, we presented novel approaches and techniques for solving problems within three research areas in the domain of computer security. The challenges and issues covered in this work range from web security to automatic protocol analysis, and are crucial to the security and privacy of users.

Social Network De-Anonymization. Social networking is a recent phenomenon on the Internet. Sites that offer social networking features are reporting exponential growth rates (e.g., [2]). In many ways, the growth and popularity of social networking sites is showing similarities to the early days of e-mail. Just as e-mail revolutionized communication in the early days of the Internet, social networking sites are now making it easier for Internet users to maintain friendships, and to meet new people.

Social networking sites are useful, and as a result, they have millions of registered users. However, these sites are interesting from a security and privacy point of view as they store large amounts of sensitive personal user data. For an attacker, social networking sites are interesting targets as the sensitive information stored by them can be used in many attack scenarios (e.g., targeted phishing, targeted spam, etc.).

In this work, we have introduced a novel, practical de-anonymization attack that makes use of the group information in social networking sites. Using empirical, real-world experiments, we show that the group membership of a user in a social network (i.e., the groups within a social network in which a user is a member), may reveal enough information about an individual user to identify her when visiting web pages from third parties.

The implications of the attack we present are manifold. The attack requires a low effort, and has the potential to affect millions of registered social networking users who have group memberships.

The theoretical analysis and empirical measurements we present demonstrate the feasibility of the attack on the Xing, Facebook, and LinkedIn social networks. Furthermore, our investigations suggest that many more social networks that support group memberships can potentially be misused for similar

attacks.

Cybercrime and the Underground Economy. In Chapter 3, we performed a technical and economic analysis of the online adult industry. The novel insights that we gained show that shady business practices and well-known security threats often go hand in hand with each other. We analyzed the economic structure of this industry, and found that apart from the expected "core business" of adult sites, more shady business models exist in parallel. Our evaluation shows that many adult web sites try to mislead and manipulate their visitors, with the intent of generating revenue. To this end, a wide range of questionable techniques are employed, and openly offered as business-to-business services. The tricks that these web sites employ range from simple obfuscation techniques such as relatively harmless *blind links*, over convenience services for typo-squatters, to sophisticated redirector chains that are used for traffic trading. Additionally, the used techniques have the potential to be exploited in more harmful ways, for example by facilitating CSRF attacks or click-fraud.

By mimicking adult web site operators ourselves, we gained additional insights on unique security aspects in this domain. For example, we discovered that a malicious operator could infect more than 20,000 with a minimal investment of about $160. We successfully conducted an experiment that shows that the lack of appropriate security measures in this domain indeed facilitates click-fraud. Furthermore, adult web sites seem to have a high potential for being abused in malicious Pay-Per-Install schemes.

We conclude that many participants of this industry have business models that are based on very questionable practices that could very well be abused for malicious activities and conducting cyber-crime. In fact, we found evidence that this kind of abuse is already happening in the wild.

Automatic Protocol Reverse Engineering. As the final main contribution of this thesis, we presented Prospex, a system to automatically extract application layer protocol specifications. The goal of our techniques is to provide precise specifications of unknown network protocols. Our system monitors the execution of a (server) program that processes network input in a controlled environment to perform a behavioral analysis of a binary. Based on the recorded execution traces, the tool produces accurate message format specifications for different types of messages and a generalized protocol state machine.

Our technique proceeds in three main steps: First, we split application sessions into individual messages and extract their formats. The second step

is responsible for clustering similar messages. The notion of similarity is established not only by comparing message formats, but also by analyzing the overall behavior of the server in reaction to each input. Based on the clusters, we can assign a type to each message, a process that required manual analysis in previous work. Finally, the third step infers a generalized protocol state machine that reflects the sequences in which messages may be exchanged.

Our experiments demonstrate that the presented approach works well in practice. Our system can analyze real-world programs, producing specifications for complex protocols such as SMB/CIFS. Moreover, our system is able to help malware analysts by automatically reverse-engineering a non-standard protocol used by a malicious bot program. Additionally, our system can create detailed input specifications for a stateful fuzzer. For a number of real-world server applications, this allowed us to automatically find security vulnerabilities.

Bibliography

[1] PFSA Toolkit. http://www.cs.usyd.edu.au/~rcdmnl/PFSA.

[2] Facebook. http://www.facebook.com, 2009.

[3] LinkedIn. http://www.linkedin.com, 2009.

[4] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, 2009.

[5] Adobe Systems Incorporated. Adobe Flash Player. http://www.adobe.com/de/products/flashplayer/, 2009.

[6] Alexa. Top 500 Global Sites. http://www.alexa.com/topsites, 2009.

[7] R. Alur, P. Černý, P. Madhusudan, and W. Nam. Synthesis of Interface Specifications for Java Classes. *SIGPLAN Not.*, 40(1), 2005.

[8] Glenn Ammons, Rastislav Bodík, and James R. Larus. Mining Specifications. *SIGPLAN Not.*, 37(1), 2002.

[9] D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2), 1987.

[10] Vinod Anupam, Alain Mayer, Kobbi Nissim, Benny Pinkas, and Michael K. Reiter. On the Security of Pay-per-click and Other Web Advertising Schemes. In *Proceedings of the Eighth Conference on World Wide Web (WWW)*, 1999.

[11] Danmec / Asprox SQL Injection Attack Tool Analysis. http://www.secureworks.com/research/threats/danmecasprox/?threat=danmecasprox, 2008.

[12] Asterisk: The Open Source PBX and Telephony Platform. http://www.asterisk.org, 2008.

[13] Asterisk DOS Vulnerability. http://secunia.com/advisories/24579, 2007.

[14] B. Stone-Gross and M. Cova and L. Cavallaro and B. Gilbert and M. Szydlowski and R. Kemmerer and C. Kruegel and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[15] Greg Banks, Marco Cova, Viktoria Felmetsger, Kevin Almeroth, Richard Kemmerer, and Giovanni Vigna. Snooze: Toward a Stateful Network Protocol Fuzzer. In *Proceedings of the 9th Information Security Conference (ISC)*, 2006.

[16] David Baron. :visited support allows queries into global history. `https://bugzilla.mozilla.org/show_bug.cgi?id=147777`, 2002.

[17] U. Bayer, P. Milani Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *Symposium on Network and Distributed System Security (NDSS)*, 2009.

[18] Beano Publishing. Domain Players Club. `http://www.domainplayersclub.com`, 2009.

[19] M. Beddoe. The Protocol Informatics Project. In *Toorcon*, 2004.

[20] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *18th International Conference on World Wide Web (WWW)*, 2009.

[21] Joseph Bonneau and Sören Preibusch. The Privacy Jungle: On the Market for Privacy in Social Networks. In *Eighth Workshop on the Economics of Information Security (WEIS)*, 2009.

[22] Nikita Borisov, David Brumley, Helen J Wang, John Dunagan, Pallavi Joshi, and Chuanxiong Guo. A Generic Application-Level Protocol Analyzer and its Language. In *14h Symposium on Network and Distributed System Security (NDSS)*, 2007.

[23] Andrew Bortz and Dan Boneh. Exposing Private Information by Timing Web Applications. In *16th International Conference on World Wide Web (WWW)*, 2007.

[24] Zbigniew Braniecki. CSS allows to check history via :visited. `https://bugzilla.mozilla.org/show_bug.cgi?id=224954`, 2003.

[25] Garrett Brown, Travis Howe, Micheal Ihbe, Atul Prakash, and Kevin Borders. Social networks and context-aware spam. In *ACM 2008 Conference on Computer Supported Cooperative Work (CSCW)*, 2008.

[26] Bruce Schneier. The Eternal Value of Privacy. `http://www.schneier.com/essay-114.html`, 2006.

[27] D. Brumley, J. Caballero, Z. Liang, J. Newsome, and D. Song. Towards Automatic Discovery of Deviations in Binary Implementations with Applications to Error Detection and Fingerprint Generation. In *Usenix Security Symposium*, 2007.

[28] M. Bugalho and A. L. Oliveira. Inference of Regular Languages Using State Merging Algorithms with Search. *Pattern Recognition*, 38(9), 2005.

[29] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: Automatic Extraction of Protocol Format using Dynamic Binary Analysis. In *14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[30] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

[31] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: Automatically Generating Inputs of Death. In *13th ACM Conference on Computer and Communications Security (CCS)*, 2006.

[32] Monica Chew, Dirk Balfanz, and Ben Laurie. (Under)mining Privacy in Social Networks. In *Proceedings of Web 2.0 Security and Privacy Workshop (W2SP)*, 2008.

[33] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum. Understanding Data Lifetime via Whole System Simulation. In *Usenix Security Symposium*, 2004.

[34] Computational Crawling LP. 80legs. `http://www.80legs.com`, 2009.

[35] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P Barham. Vigilante: End-to-End Containment of Internet Worms. In *20th ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

[36] Scott Coulls, Charles Wright, Fabian Monrose, Michael Collins, and Michael Reiter. Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Traces. In *Symposium on Network and Distributed Systems Security (NDSS)*, 2007.

[37] J. Crandall and F. Chong. Minos: Control Data Attack Prevention Orthogonal to Memory Model. In *37th International Symposium on Microarchitecture (MICRO)*, 2004.

[38] W. Cui, J. Kannan, and H. Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Traces. In *16th Usenix Security Symposium*, 2007.

[39] W. Cui, V. Paxson, N. Weaver, and R. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In *13th Symposium on Network and Distributed System Security (NDSS)*, 2006.

[40] W. Cui, M. Peinado, K. Chen, H. Wang, and L. Irun-Briz. Tupni : Automatic Reverse Engineering of Input Formats. In *ACM Conference on Computer and Communications Security (CCS)*, 2008.

[41] Potential Arbitrary File Access. `http://www.securityfocus.com/archive/1/377618`, 2004.

[42] D. Dagon, G. Gu, C. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Annual Computer Security Applications Conference (ACSAC)*, 2007.

[43] Neil Daswani and Michael Stoppelman. The Anatomy of Clickbot.A. In *First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[44] Claudia Diaz, Carmela Troncoso, and Andrei Serjantov. On the Impact of Social Network Profiling on Anonymity. In *8th International Symposium on Privacy Enhancing Technologies (PETS)*, 2008.

[45] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, 2004.

[46] Domains By Proxy, Inc. Domains By Proxy. `http://domainsbyproxy.com`, 2009.

[47] J.C. Dunn. Well Separated Clusters and Optimal Fuzzy Partitions. *Journal of Cybernetics*, 4, 1974.

[48] Ekiga - Free your speech. `http://www.ekiga.org`, 2008.

[49] Dawson Engler, David Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. In *ACM Symposium on Operating Systems Principles*, 2001.

[50] Edward W. Felten and Michael A. Schneider. Timing Attacks on Web Privacy. In *7th ACM Conference on Computer and Communications Security (CCS)*, 2000.

[51] Finjan Inc. LuckySploit Toolkit Exposed. `http://www.finjan.com/MCRCblog.aspx?EntryId=2213`, 2009.

[52] Jason Franklin, Vern Paxson, Stefan Savage, and Adrian Perrig. An inquiry into the nature and causes of the wealth of internet miscreants. In *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[53] A. Fritzler. UnOfficial AIM/OSCAR Protocol Specification. `http://www.oilcan.org/oscar/`, 2007.

[54] Patrice Godefroid, Adam Kieżun, and Michael Y. Levin. Grammar-based Whitebox Fuzzing. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2008.

[55] Patrice Godefroid, Michael Y. Levin, and David A Molnar. Automated Whitebox Fuzz Testing. In *Network and Distributed Security Symposium (NDSS)*. Internet Society, 2008.

[56] E. Mark Gold. Language Identification in the Limit. *Information and Control*, 10(5), 1967.

[57] E. Mark Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3), 1978.

[58] V. Griffith and M. Jakobsson. Messin' with Texas, Deriving Mother's Maiden Names using Public Records. In *Third Conference on Applied Cryptography and Network Security (ACNS)*, June 2005.

[59] Guywire, Inc. Booble. Adult Search Engine. `http://www.booble.com`, 2009.

[60] M. Hammami, Y. Chahir, and L. Chen. Webguard: A web filtering engine combining textual, structural, and visual content-based analysis. *IEEE Transactions on Knowledge and Data Engineering*, 18(2), 2006.

[61] R. Hansen and J. Grossman. Clickjacking. Technical report, SecTheory – `http://www.sectheory.com/clickjacking.htm`, 2008.

[62] Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. Preventing Private Information Inference Attacks on Social Networks. Technical Report UTDCS-03-09, University of Texas at Dallas, 2009.

[63] T. Holz. A Short Visit to the Bot Zoo [Malicious Bots Software]. *Security & Privacy, IEEE*, 3(3), 2005.

[64] Thorsten Holz, Markus Engelberth, and Felix Freiling. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. In *European Symposium on Research in Computer Security (ESORICS)*, 2009.

[65] Internet Filter. Internet Pornography Statistics. `http://internet-filter-review.toptenreviews.com/internet-pornography-statistics.html`, 2006.

[66] P. Jaccard. The Distribution of Flora in the Alpine Zone. *The New Phytologist*, 11(2):37–50, 1912.

[67] Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting Browser State From Web Privacy Attacks. In *15th International Conference on World Wide Web (WWW)*, 2006.

[68] Tom N. Jagatic, Nathaniel A. Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *Commun. ACM*, 50(10):94–100, 2007.

[69] M. Jakobsson, P. Finn, and N. Johnson. Why and How to Perform Fraud Experiments. *Security & Privacy, IEEE*, 6(2):66–68, March-April 2008.

[70] Markus Jakobsson and Jacob Ratkiewicz. Designing ethical phishing experiments: a study of (ROT13) rOnl query features. In *15th International Conference on World Wide Web (WWW)*, 2006.

[71] Markus Jakobsson and Sid Stamm. Web Camouflage: Protecting Your Clients from Browser-Sniffing Attacks. *IEEE Security and Privacy*, 5(6), 2007.

[72] R. Kaksonen, M. Laakso, and A. Takanen. Software Security Assessment through Specification Mutations and Fault Injection. In *IFIP Joint Working Conference on Communications and Multimedia Security (CMS)*, 2001.

[73] Richard M. Karp. "Reducibility Among Combinatorial Problems". In *Complexity of Computer Computations*. Plenum Press, 1972.

[74] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.

[75] K. J. Lang. Faster Algorithms for Finding Minimal Consistent DFAs. Technical report, NEC Research Institute, 1999.

[76] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In *ICGI 98: Proceedings of the 4th International Colloquium on Grammatical Inference*. Springer-Verlag, 1998.

[77] The Linux Desktop Testing Project. `http://ldtp.freedesktop.org`, 2008.

[78] C. Leita, M. Dacier, and F. Massicotte. Automatic Handling of Protocol Dependencies and Reaction to 0-Day Attacks with ScriptGen-based Honeypots. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2006.

[79] C. Leita, K. Mermoud, and M. Dacier. ScriptGen: An Automated Script Generation Tool for Honeyd. In *21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

[80] Z. Lin, X. Jiang, D. Xu, and X. Zhang. Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. In *15th Symposium on Network and Distributed System Security (NDSS)*, 2008.

[81] D. Lo and S. Khoo. QUARK: Empirical Assessment of Automaton-based Specification Miners. In *13th Working Conference on Reverse Engineering (WCRE)*. IEEE Computer Society, 2006.

[82] M. Cova and S. Ford. Wepawet: Detecting and Analyzing Web-Based Malware. `http://wepawet.iseclab.org`, 2009.

[83] M. Jakobsson and S. Stamm. Invasive Browser Sniffing and Countermeasures. In *15th International World Wide Web Conference*, 2006.

[84] Giorgio Maone. NoScript. `https://addons.mozilla.org/de/firefox/addon/722`, 2009.

[85] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol Specification Extraction. In *IEEE Symposium on Security and Privacy*, 2009.

[86] B.P. Miller, L. Fredriksen, and B. So. An Empirical Study of the Reliability of UNIX Utilities. *Communications of the ACM*, 33(12), 1990.

[87] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *7th ACM SIGCOMM Internet Measurement Conference (IMC)*, 2007.

[88] Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A crawler-based study of spyware on the web. In *Symposium on Network and Distributed System Security (NDSS)*, 2006.

[89] Arvind Narayanan and Vitaly Shmatikov. Robust De-anonymization of Large Sparse Datasets. In *IEEE Symposium on Security and Privacy*, 2008.

[90] S. Needleman and C. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48(3), 1970.

[91] Net Industries, LLC. IP Address Lookup and GeoTargeting. `http://www.hostip.info`, 2009.

[92] Network Working Group. WHOIS Protocol Specification. `http://tools.ietf.org/html/rfc3912`, 2004.

[93] P. Oehlert. Violating Assumptions with Fuzzing. *IEEE Security and Privacy*, 3(2), 2005.

[94] OpenDNS community. Domain Tagging. `http://www.opendns.com/community/domaintagging`, 2009.

[95] N R. Pal and J. Biswas. Cluster Validation Using Graph Theoretic Concepts. *Pattern Recognition*, 30(6), 1997.

[96] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The Devil and Packet Trace Anonymization. *SIGCOMM Comput. Commun. Rev.*, 36(1), 2006.

[97] Ruoming Pang, Vern Paxson, Robin Sommer, and Larry Peterson. binpac: A yacc for writing application protocol parsers. In *Internet Measurement Conference (IMC)*, 2006.

[98] Jon Patrick and Palmerston North. The sk-strings Method for Inferring PFSA. In *Workshop on Automata Induction, Grammatical Inference and Language Acquisition at the 14th International Conference on Machine Learning (ICML97)*, 1997.

[99] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Usenix Security Symposium*, 1998.

[100] Peach Fuzzing Platform. `http://peachfuzzer.com`, 2008.

[101] P. Porras, H. Saidi, and V. Yegneswaran. A Multi-perspective Analysis of the Storm (Peacomm) Worm. Technical report, Computer Science Laboratory, SRI International, 2007.

[102] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All Your iFRAMEs Point to Us. In *17th Usenix Security Symposium*, 2008.

[103] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The Ghost In The Browser. In *First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[104] A. Raman, P. Andreae, and J. Patrick. A Beam Search Algorithm for PFSA Inference. *Pattern Analysis and Applications*, 1, 1998.

[105] RFC 3261 - SIP: Session Initiation Protocol. `http://www.ietf.org/rfc/rfc3261.txt`, 2008.

[106] Jesse Ruderman. CSS on a:visited can load an image and/or reveal if visitor been to a site. `https://bugzilla.mozilla.org/show_bug.cgi?id=57351`, 2000.

[107] How Samba Was Written. `http://samba.org/ftp/tridge/misc/french_cafe.txt`, 2007.

[108] Koushik Sen, Darko Marinov, and Gul Agha. CUTE: A Concolic Unit Testing Engine for C. In *ESEC/FSE-13: Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2005.

[109] Spam Assassin. List of performed Tests. http://spamassassin.apache.org/tests_3_2_x.html, last accessed: 23.04.2009, 2009.

[110] M. Sutton, A. Greene, and P. Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, 1st edition, 2007.

[111] Symantec Corporation. Misleading Applications. http://www.symantec.com/connect/blogs/misleading-applications-show-me-money-part-3, 2009.

[112] The Honeynet Project. Capture-HPC Client Honeypot. https://projects.honeynet.org/capture-hpc, 2009.

[113] Trusteer, Inc. Flash Security Hole Advisory. http://www.trusteer.com/files/Flash_Security_Hole_Advisory.pdf, 2009.

[114] U.S. Census Bureau. Frequently Occurring Names and Surnames. http://www.census.gov/genealogy/www, 2009.

[115] S. Venkataraman, J. Caballero, P. Poosankam, M. Kang, and D. Song. Fig: Automatic Fingerprint Generation. In *Symposium on Network and Distributed System Security (NDSS)*, 2007.

[116] C.S. Wallace and M.P. Georgeff. A General Objective for Inductive Inference. Technical report, Department of Computer Science, Monash University, 1983.

[117] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated Web Patrol with Strider HoneyMonkeys. In *Symposium on Network and Distributed System Security (NDSS)*, 2006.

[118] Yi-Min Wang, Doug Beck, Jeffrey Wang, Chad Verbowski, and Brad Daniels. Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting. In *2nd Conference on Steps to Reducing Unwanted Traffic on the Internet*, 2006.

[119] Zhi Wang, Xuxian Jiang, Weidong Cui, and Xinyuan Wang. ReFormat: Automatic Reverse Engineering of Encrypted Messages. Technical Report 2008-26, NC State University, 2008.

[120] J. Whaley, M. C. Martin, and M. S. Lam. Automatic Extraction of Object-Oriented Component Interfaces. *SIGSOFT Softw. Eng. Notes*, 27(4), 2002.

[121] WhoisGuard. WhoisGuard. `http://www.whoisguard.com`, 2009.

[122] Wietse Venema. Postfix. `http://www.postfix.org`, 2008.

[123] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. Automatic Network Protocol Analysis. In *15th Symposium on Network and Distributed System Security (NDSS)*, 2008.

[124] XBIZ. The Adult Industry Source for Business News and Information. `http://www.xbiz.com`, 2009.

[125] X-Lite softphone. `http://www.counterpath.com`, 2008.

[126] J. Yang and D. Evans. Perracotta: Mining Temporal API Rules from Imperfect Traces. In *28th Internl. Conf. on Software Engineering (ICSE 2006*.

[127] Junfeng Yang, Can Sar, Paul Twohey, Cristian Cadar, and Dawson Engler. Automatically Generating Malicious Disks Using Symbolic Execution. In *IEEE Security and Privacy*, 2006.

[128] Elena Zheleva and Lise Getoor. To Join or Not To Join: The Illusion of Privacy in Social Networks with Mixed Public and Private User Profiles. In *18th International Conference on World Wide Web (WWW)*, 2009.

Curriculum Vitae

02.07.1980	geboren in Eisenstadt, Österreich
1986 - 1990	Besuch der Volksschule Großhöflein
1990 - 1998	BG/BRG/BORG Eisenstadt
1999 - 2006	Diplomstudium Informatik an der TU Wien,
2006 - 2008	Magisterstudium Informatikmanagement an der TU Wien, (mit Auszeichnung)
2006 - 2009	PhD Studium Informatik an der TU Wien