SURVEYING THE LANDSCAPE OF ACTIONSCRIPT SECURITY

TRENDS AND THREATS

by

Dhiraj V. Karamchandani

APPROVED BY SUPERVISORY COMMITTEE:

Kevin W. Hamlen, Chair

Balaji Raghavachari

Bhavani Thuraisingham

This thesis is dedicated to my advisor Dr. Kevin W. Hamlen
in whose eyes I read the definition of the term 'research', and
to my dearest family members who have
instilled in me the invaluable virtue of selflessness.

SURVEYING THE LANDSCAPE OF ACTIONSCRIPT SECURITY TRENDS AND THREATS

by

DHIRAJ V. KARAMCHANDANI, BE

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2013

ACKNOWLEDGMENTS

The text of this thesis incorporates and extends joint work with Meera Sridhar and Dr. Kevin W. Hamlen, which has been submitted for publication and is currently under review. The author thanks both co-authors for their contributions to the study.

This work was supported in part by the National Science Foundation under award #1065216. All opinions, findings, conclusions, or recommendations expressed are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

November 2013

PREFACE

This thesis was produced in accordance with guidelines which permit the inclusion as part of the thesis the text of an original paper or papers submitted for publication. The thesis must still conform to all other requirements explained in the "Guide for the Preparation of Master's Theses and Doctoral Dissertations at The University of Texas at Dallas." It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this thesis to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts which provide logical bridges between different manuscripts are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the student's contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the thesis attest to the accuracy of this statement.

SURVEYING THE LANDSCAPE OF ACTIONSCRIPT SECURITY

TRENDS AND THREATS

Publication No. _____

Dhiraj V. Karamchandani, MS
The University of Texas at Dallas, 2013

Supervising Professor: Kevin W. Hamlen

As one of the foremost scripting languages of the World Wide Web, Adobe's ActionScript Flash platform now powers multimedia features for a significant percentage of all web sites. However, its popularity and complexity have also made it an attractive vehicle for myriad malware attacks over the past five years. Despite the perniciousness and severity of these threats, ActionScript has been relatively less studied in the scholarly security literature. To fill this void and stimulate future research, this thesis presents a systematic study of Flash security threats and trends, including an in-depth taxonomy of fifteen major Flash vulnerability and attack categories, a detailed investigation of 520 Common Vulnerability and Exposure (CVE) articles reported between 2008–2013, and an examination of what makes Flash security challenges unique. The results of these analyses provide researchers, web developers, and security analysts a better sense of this important attack space, and identify the need for stronger security practices and defenses for protecting users of these technologies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Adobe Flash applets (Shockwave Flash programs) provide web developers a superior platform for creating rich, dynamic web content such as web advertisements, online games, streaming media and interactive webpage animations, resulting in a soaring popularity of the technology on the web. Additionally, most of this content can also be made available to the desktop using the Adobe Integrated Runtime (AIR) cross-platform environment. The following statistics [7, 99] demonstrate the pervasive impact of Flash. More than 20,000 apps in mobile app stores such as Apple App Store and Google Play are created using Flash. A staggering revenue of over US$70 million per month is generated by the top nine Flash enabled games in China. Flash is used in 24 of top 25 Facebook games. Flash is the choice of technology of more than three million developers for creating interactive and animated web environments. Flash is used by 16.9% of all websites.

The popularity of Flash combined with the complexity of its features has made it extremely attractive to attackers. The 2013 Cisco Annual Threat Report marks Adobe Flash as the third highest in the top content types for malware distribution [20]. The 2013 Symantec Internet Security Threat Report mentions that of all plug-in vulnerabilities between 2010 and 2012, Adobe Flash Player constitutes 18%, 20%, and 22% in the years 2010, 2011, and 2012 respectively. Flash-powered attacks have successfully penetrated some of the most security-hardened facilities in the world, such as the famous 2011 penetration of RSA [68], and the massive Luckycat campaign that targeted an entire spectrum of important U.S. industries such as aerospace, energy, engineering, shipping, and military research, as well as top-level international organizations such as Indian military research institutions, and groups in Japan and Tibet [48].

One reason Flash security is so non-trivial is because of the feature-filled complexity of the ActionScript bytecode language [6], which Flash uses internally. Like other ECMAScript languages, ActionScript includes language features such as an object model, function calls, class inheritance, compile time and run-time type checking, packages, namespaces, regular expressions, and direct access to security-relevant system resources [3]. However, unlike JavaScript, ActionScript programs are disseminated as compiled binary Flash files (.swf files) that pack images, sounds, text, and bytecode in a webpage-embeddable form, which is then seamlessly JIT-compiled and/or interpreted by the Adobe Flash Player browser plug-in when the page is viewed. This transparent purveyance of powerful binary content from Flash authors, through page publishers, to end-users educes many security threats.

Security apprehensions have been further exacerbated due to the recent trend of web environments becoming aggressively heterogeneous (e.g., composed of mash-ups that mix mobile code from many mutually distrusting sources), which expand the attack vulnerability surface area. Additionally, Flash's deployment as plug-in VM tends to widen threat windows due to patch lag. Newly discovered VM vulnerabilities are typically resolved by patches released by Adobe, but many consumers are apathetic or inattentive in downloading and installing the patches; in many large companies, older versions of the Flash plug-ins may be required for compatibility with critical business systems, creating reluctance in updating to the latest version. This lag in patch deployment rate has resulted in many home and large organization systems prone to Flash-attacks [96]. Consequently, effective intrusion detection of Flash-based attacks must consider a large array of past AVM versions and configurations. Due to the lack of a consistent, systematic solution to the problem, many security advisories suggest disabling Flash altogether as a fool-proof protection strategy [93]; unfortunately, this strategy is antithetical to the revenue models of many businesses.

Despite significant causes for concern, attention given by the formal research community has been disproportionately low compared to gravity of the Flash security problem. For

example, between 2008 and 2013 only 1.4% of publications in the top six non-cryptography security venues (ranked by Google Scholar h5-index) concerned Flash, and only one venue (the *IEEE Symposium on Security & Privacy*) devoted a large fraction of web security research (42%) to Flash-related threats (see §3.1 for a more detailed description of our scientific research methodology).

Scattered, ad hoc information about Flash security abounds in the literature, especially in the form of news stories and "best practices" tips for Flash programmers. A systematic study of known attacks and attack classes, their potential impact, the landscape of the attack surface, and known strategies for mitigation, is badly needed for organizing this scattered information and helping both researchers and practitioners learn from past mistakes to build stronger defenses for this pervasive web technology.

Towards this goal, our main contributions are three-fold:

1. We present a detailed taxonomy of fifteen Flash-relevant vulnerabilities and attacks. Our categorization provides a more fine-grained, informative classification specifically tuned to the *Flash* attack surface compared to the cross-section of Mitre's Common Weakness Enumeration (CWE) classification system used by the National Vulnerability Database (NVD) [75] for scoring Common Vulnerability and Exposures articles (CVEs) [72].

2. For each category, we highlight ActionScript language and Flash architecture features that make them particularly susceptible to that attack/vulnerability. We also present a compilation of pertinent resources such as academic and news articles, examination of attack-type variants, high-impact real world incidents, and representative CVEs.

3. We present the results of a detailed analysis of Flash threats and research trends using our derived taxonomy. As part of our analysis, we report on all Flash-relevant CVE articles recorded between 2008–2013, and their classification using our taxonomy. We

show why existing classification methodologies such as the CWE numbering used by NVD prove inadequate for systematizing Flash attacks, and highlight why the lack of detail in CVE articles makes meaningful classification extremely challenging. Our analysis also includes a report of attack and vulnerability type distribution per year, and evolution of attack and vulnerability trends over time; we integrate findings from latest annual threat reports from major security organizations such as Symantec, Cisco, and Kaspersky.

As security researchers, we were met with several challenges in conducting this survey on the Flash attack space. First and foremost was the challenge of sifting through and classifying a massive volume of completely disorganized information on Flash security such as thousands of news articles (including new articles that appear daily), numerous research publications, scattered information on past Flash attacks, and dispersed material on various components of the Flash ecosystem, including the Flash browser plug-in, VM, development and analysis tools, and the ActionScript language. The difficulty of taming information volume was further heightened due to the innumerable versions of various Flash software components such as the Player and the ActionScript language, each of which exhibits a multitude of features and weaknesses. CVE articles of Flash-relevant attacks tend to be too terse and coarse-grained to glean any useful technical details of an attack for educational purposes. Therefore, with our analysis, we aim to provide researchers, web developers, and security analysts a substantive sense of the Flash vulnerability and attack space, in a *consolidated* form, crucial for developing better Flash security practices and defenses, and we hope that this attempt will fuel security research towards the betterment of Flash security.

Past and future enforcement mechanisms for these attacks and vulnerabilities are beyond the scope of this thesis; however various prior works explore these topics (e.g., [83, 65, 2, 78]).

CHAPTER 2

A TAXONOMY OF VULNERABILITIES AND ATTACKS

We begin our survey of Flash security with an in-depth taxonomy of prominent Flash-powered vulnerabilities and attacks. The taxonomy is inspired by our detailed study (see §3) of 520 Flash-related CVEs and annual threat reports of major security organizations (e.g., Symantec, Cisco, Kaspersky) published over the past five years. High-impact, real-world attacks and the role of platform features unique to Flash in these attacks are highlighted.

2.1 Flash-based Phishing

In *phishing*, an attacker lures an unsuspecting user by masquerading as a trustworthy entity in order to steal important information such as usernames, passwords and credit card details. This is typically achieved by using various social engineering techniques to redirect the victim to a legitimate-appearing malicious site designed by the attacker.

Flash-based phishing often takes advantage of Flash's advanced animation features to create spoof sites capable of evading automated anti-phishing services. Typical automated anti-phishing services scan webpage text to identify certain suspect phrases (e.g., bank names). However, if the phishing is Flash-based, these tools are typically unable to detect these phrases or understand the scam, since they are not text-based. In fact, Flash-based features in the phishing site are even transparent to much more powerful tools such as spiders (search engines) [74].

Another common Flash-based phishing technique is to use Flash-based web advertisements for phishing; attackers abuse specific ActionScript 2 and ActionScript 3 language features only available through Flash. These features include *Flash Shared Objects* (similar

to HTTP cookies, allowing Flash applets to store information about a user on the user's computer, useful for computing timestamps for attacks; these can store upto 100 KB per host name, are persistent data, and can work cross-browser [18]), methods `MovieClip.getURL()` and `flash.net.navigateToURL()` (to perform actual redirects), and `LoadVars.load()` (to make HTTP requests to the attacker's web domain, in order to keep track of the malicious redirects, or to disable any specific redirects if he or she chooses) [35]. Malicious advertisements are discussed further in §2.15.

Real-world Example:

RSA SecurID Breach. One of the most shocking Flash-based phishing attacks in history was the attack on the website of RSA Security LLC (an American computer and network security company) in 2011 [68, 53, 69, 21, 11]. The attack was allegedly conducted by a nation-state, targeting Lockheed-Martin and Northrop-Grumman to steal military secrets [68]. These companies were using RSA's two-factor authentication product, SecurID, for network authentication.

In the attack, two phishing emails were sent to four EMC (RSA's parent company) employees. The emails carried a malicious Excel spreadsheet attachment with the subject line "2011 Recruitment plan.xls". The attachment used a zero-day exploit targeting vulnerability in the `authplay.dll` component in Flash player, creating a backdoor on the victim's machine. The attackers spoofed the emails as if they originated from a web master at `Beyond.com`, a job search and recruiting site. The email body had a deceptively innocuous simple line: "I forward this file to you for review. Please open and view it." The Excel attachment had just an "X" in its first cell. The attack used the Poison Ivy Remote Administration Tool (RAT) [33] (Trojan backdoor) on the compromised computers, using which the attackers were able to harvest users' credentials to access other RSA network machines, and copy senstive information and transfer data to their own servers [53, 69, 21, 11]. The speculation is that one of the credentials stolen was the unique numbers for the SecurID tokens [69].

The severity of the ramifications is demonstrated by the fact that RSA's only choice was to replace their SecurID tokens for their customers worldwide [68] (CVE-2011-0609).

2.2 Flash-based Pharming and DNS Rebinding

2.2.1 Flash-Based Pharming

Pharming is a more sophisticated version of phishing in which an attacker redirects an unsuspecting user to an unintended website, either by changing the `hosts` file on the victim's computer, or by exploiting a vulnerability in the DNS server software.

Several ActionScript 2 and ActionScript 3 methods facilitate pharming, such as: (i) `getURL()` (AS2) and `flash.net.navigateToURL()` (AS3), which can not only be used to navigate to a website, but also to directly execute JavaScript; (ii) `loadMovie()` (AS2); and (iii) `flash.net.navigateToURL()` in conjuction with `flash.net.URLRequest` objects (AS3) [37].

2.2.2 DNS Rebinding

DNS rebinding allows an attacker to use a victim's browser environment (typically JavaScript or Flash) to connect to internal IP addresses in the victim's network [94]. This in turn can be used to leverage the victim machine for stealing information, spamming, distributed denial-of-service, and other attacks on the victim's internal network. While the *Same-origin Policy* (please see §2.4) restricts communication between objects from differing origins, the DNS rebinding attacker is able to bypass this by dynamically switching the target IP address to a host name that he or she controls [94].

Real-world Example:

Massive DNS Poisoning Attack in Mexico. One of the first *drive-by-pharming* attacks in the wild occurred in Mexico, and exploited a vulnerability in *2wire* modems. The attack was conducted using spam email messages that fooled victims into believing that they received an

electronic postcard from `Gusanito.com`, a popular e-card website. When the victims clicked on the link to view the cards, they were directed to a spoofed `Gusanito` page. This spoofed page had a malicious SWF file that modified the *2wire* modem `localhost` table. Subsequently, the malicious Flash controls involved redirecting users to a fraudulent site whenever they attempt to access pages related to `Banamex.com`, a banking site [77].

2.3 Flash-based Drive-by-Download and Drive-by-Cache

In a Flash-based *drive-by-download* attack, the attacker compromises a website by injecting a malicious Flash binary into the site. The Flash binary loads a malicious payload (also called *shellcode*) into the address space of the browser. The code is usually a series of commands that directs the browser process to retrieve malware (usually from a different domain), write it to disk, and subsequently execute it. This attack is extremely dangerous because not only the usual user warning for download is bypassed, even simply reading a webpage or viewing a document results in the malware being downloaded quietly in the background [31].

Drive-by-cache is a variation of drive-by-download attack, in which the malware is already present in the browser's cache directory and is executed, unlike in a drive-by-download attack, where the malware is downloaded and written to disk. Drive-by-cache makes infection harder to detect than drive-by-download—in a drive-by-download attack, the malware (which is often times the downloader) has to pass through the personal firewall and web filter in order to obtain the malicious payload. This makes the malware susceptible to detection. In drive-by-cache, the payload is pre-downloaded into the browser cache for easy access [25].

Drive-by-download attackers thrive on users visiting the malicious website. A typical method used is to manipulate search engines to list the site high in rankings to try and ensure that visitors will visit it [70]. Often heap spraying (see §2.8) is used in conjunction to inject the shellcode.

Drive-by-download attacks also exploit other vulnerabilities, such as integer-overflow vulnerabilities. For example. several drive-by-download attacks have been conducted exploiting a vulnerability discovered in the `DefineSceneAndFrameLabelData` tag parsing routine in the Flash Player [29]. The vulnerability was caused by the routine reading an unsigned 32-bit integer and subsequently validating it using a signed comparison operator (CVE-2007-0071) [35].

Real-world Examples:

Drive-by-download Attacks on Windows and Apple Users. Drive-by-downloads were conducted on Windows (CVE-2013-0633) and Apple Macintosh (CVE-2013-0634) users, and targeted vulnerabilities through spear phishing email messages [31, 89]. The victims were from several industries, including aerospace (specifically Boeing) [89].

In the Windows attack, users were lured into opening a Microsoft Word document delivered as email attachments that contained malicious Flash files. As reported, one of the attachments used the 2013 IEEE Aerospace Conference schedule, and another was related to the US online payroll system company, ADP, to exploit the vulnerability in CVE-2013-0633 [89]. The exploit targeted the ActiveX version of Flash Player on Windows [31]. One of the malicious payloads (executable) was signed with a fake certificate from a South Korean company called MGAME. This certificate has been used several times in the past as part of targeted attacks [15]. In the Mac attack, the vulnerability in CVE-2013-0634 was exploited by tricking an Apple OS X user to open a webpage, which contained a malicious Flash file hosted on websites, targeting Flash Player in Firefox or Safari on the Macintosh platform [31, 89].

Drive-by-cache attack on the UK Human Rights website. In April 2011, the UK Human Rights website [70] was hit by a Flash drive-by cache attack, in which a Flash zero-day vulnerability (CVE-2011-0611) was exploited to infect multiple pages of the website, and install a malware which allowed the attackers to connect back to a malicious IP in Hong Kong. At

the time of attack, there was no patch available for the zero-day vulnerability. The following fact demonstrates the difficulty of detection of the malicious software—VirusTotal detection was 0 out of 42 for the zero-day exploit, and 1 out of 42 for the malicious payload [45].

2.4 Same-Origin Policy Abuse

2.4.1 Same-origin Policy

Same-origin policy allows major interactions and scripting between pages originating from the same site (determined using a combination of protocol, host and port number), and restricts inter-communication between unrelated sites. Many variations of same-origin policy exist, including ones for DOM access, XMLHttpRequest, cookies, Java, JavaScript, and Flash [103].

The security context for Flash applets is derived from the their originating URL, and not from their embedding site. This is achieved by comparing protocol, host name and port of requestor and requested resource; for a Flash applet from a specific origin, universal access is granted to local disk contents at that origin. Flash applets can request permission for outside-domain resources using a `crossdomain.xml` policy file or the `Security.allowDomain()` directive in the `flash.system` package. For example, consider `foo.swf` in domain `X` and `bar.swf` in domain `Y`. In order for `bar.swf` to access `foo.swf`, `Y` must be added to `crossdomain.xml` policy file at `X` or `Security.allowDomain("Y")` statement must be added in `foo.swf`. Note that these methods give all Flash files in domain `Y` access to `foo.swf` [103].

Lax development practices and subtle differences in same-origin policies have led to myriad security problems, discussed below. Despite Flash's above mentioned methods for controlling access to exposed functions (*viz.*, `flash.system.Security.allowDomain()` and `flash.system.Security.allowInsecureDomain()`), many ad developers commonly use these features unwisely, for example by specifying wildcard ("*") that permits universal access [32, 83].

Often, this is the case because it is difficult for developers to determine which domains are needed by the library at the time of development. However, using the wildcard in this manner is highly imprudent because many sites that have a "*" access policy use cookies for authentication and maintain private information for logged-in users [50].

Malicious Flash applets can exploit subtle differences between Flash and JavaScript same-origin policy to bypass the Flash same-origin policy and deliver malicious JavaScript code to a third-party victim site via the `flash.external.ExternalInterface` class. Subsequently, attackers can successfully implement two-way communication with the victim third-party site, becoming fully capable of conducting attacks such as click forgery, resource theft, or flooding attacks upon victim sites [83]. Please see §2.5 for more details.

More recently, attacks through malvertisements (malicious advertisements) have gained momentum (see §2.15 for more details). Most webpages today contain web ads, an important source of revenue for publishers; many contain ads derived from multiple ad networks. Malvertisements can place both the publisher and the ad network at risk by abusing Flash-JavaScript interaction to extend its privileges to DOM objects and call exposed functions from a more trustworthy ad on the same page [83].

In addition to these attack vectors, Flash's same-origin policy contains various other potentially risky leniencies. Examples include the ability to make cookie-bearing cross-domain `HTTP GET` and `POST` requests via the browser stack, through the `URLRequest` API; the ability for embedding webpages to allow various permissions via the `<OBJECT>` or `<EMBED>` parameters, such as: load external files and navigate the current browser window using `allowNetworking` attribute; interact with on-page JavaScript context `allowScriptAccess` attribute; run in full-screen mode `allowFullScreen` attribute [103].

2.4.2 Case-study: Client-side Flash Proxies

An excellent example of security issues rising from subtle differences in same-origin policy between Flash and JavaScript is the concept of client-side Flash proxies [51]. While Flash

allows cross-domain HTTP requests through the `crossdomain.xml` policy file, cross-domain HTTP requests in JavaScript are achieved via the new Cross-origin Resource Sharing (CORS) feature [42]. CORS uses HTTP response headers to allow or deny requests unlike `crossdomain.xml` [51].

While many newer browsers support CORS (e.g., mobile browsers that do not have plug-ins or browsers where plug-ins have been disabled due to security reasons), many legacy browsers do not. Therefore, developers have to create a CORS and a non-CORS version of cross-domain HTTP requests. Many developers currently use Flash proxies to aid in this process. Flash proxies are Flash applets that include a small JavaScript library that interfaces with the JavaScript on the hosting page, handling HTTP requests to cross-domain targets and responses back to the calling script [51].

If the Flash applet provides a public JavaScript interface, scripts running in the context of the embedding page can abuse this interface to perform functions executed under the cross-domain origin of the Flash applet (possibly higher privileges). Note that, for this type of abuse, the exported methods for external domains have to be reported using the `allowDomain()` directive by the Flash applet. However, as mentioned above, many developers whitelist all domains using the wildcard mechanism `allowDomain("*")`, rendering such attacks feasible [51].

Cross-site request forgery, session hijacking, and leakage of sensitive information attacks can be conducted via an abuse of Flash proxies and gaps in same-origin-policy to obtain trust through transitivity [51]. Additionally, Flash proxies and gaps in same-origin-policy can also be abused to make requests from malicious domain `A` to `B`, even without `B` providing a `crossdomain.xml` policy file [51].

2.5 Attacks using ExternalInterface, URLRequest, and navigateToURL

2.5.1 ExternalInterface Abuse

Flash, through the `ExternalInterface` class in ActionScript 2 and ActionScript 3, provides two methods to interact with external containers, such as JavaScript in the embedding page: `call()` and `addCallback()`. ActionScript method `call(s, ...)` invokes JavaScript function `s` (which is passed as a string to the JavaScript VM and evaluated as JavaScript code at global scope to obtain a JavaScript function reference). ActionScript method `addCallback(s, f)` makes ActionScript function `f` callable from JavaScript under pseudonym `s` (a fresh JavaScript property name). The ActionScript method can return a value, and JavaScript receives it immediately as the return value of the call. Hence, the methods `call()` and `addCallback()` allow two-way communication between ActionScript and JavaScript [83].

The cross-language communication is extremely useful for developing rich web applications. For example, some of the uses include tracking clicks on web advertisements to gather revenue, interactive communication between the embedding HTML page and the Flash movie, including HTML buttons to start/stop the movie, random access to different chapters in the Flash movie, sending usage reporting of user interactions with the movie to Google Analytics, and data transportation between Flex chart and HTML data table [63].

Security for the ActionScript-JavaScript interface is provided by the `allowScriptAccess` property in the `<OBJECT>` and `<EMBED>` tags of the HTML page. In particular, the `call()` method requires the `allowScriptAccess` property to be set to one of three options: `always` (full access), `sameDomain` (same origin access), or `never` (none); same origin access is the default. For the `addCallback()` method, the default setting is that the HTML page can communicate with the ActionScript only if it originates from the same domain. To override the default, one must use the `allowDomain()` method in the `flash.system.Security` class.

Since `ExternalInterface.call` behaves very similarly to JavaScript's `eval`, it can be abused to corrupt the DOM and develop attack back channels similar to BeEF [8]. `ExternalInterface.call`

has also been featured in attacks that use a combination of ActionScript and JavaScript to perform cross-domain code-injection [43] (CVE-2011-0611).

Real-world Examples:

Cross-site scritping (XSS) and cross-site request forgery (CSRF) vulnerabilities were found in two widely-deployed applications SWFUpload [88] and Plupload [73]. The applications have Flash at their core, and allow developers customization of user-interface upload features such as multiple file selection, upload progress, and client-side file size checking for incorporation into sophisticated web publishing software such as Wordpress [102].

XSS in SWFUpload. ActionScript code for SWFupload uses callbacks as the first parameter to `ExternalInterface.call()`, which in turn executes JavaScript in the current page. The value of `movieName` derived from input by the user and direct loading of the applet by passing parameters in the URL result in the XSS attack. Sites where the applet is hosted on the same domain as that of the main website are vulnerable to this kind of attack [84] (CVE-2012-3414,CVE-2013-2205).

CSRF in Plupload. An attacker was able to make a request to the domain where a Plupload applet was hosted, and was able to read the full response; the applet was embedded on a page using JavaScript. This was facilitated by Flash's same-origin policy. As a result, CSRF tokens and other sensitive information were disclosed on Wordpress installations. Plupload v1.5.4 was released with the CSRF issue patched—the issue had been a whitelisting of all domains by default through `Security.allowDomain('*')` [84] (CVE-2012-3415).

2.5.2 `URLRequest` and `navigateToURL` Abuse

Flash applications extensively use URL redirection (*viz.*, `navigateToURL()` in the ActionScript 3 runtime and `getURL()` in ActionScript 2) and HTTP requests (via `URLRequest`) to direct user clicks to advertiser web sites, or load external resources. However, these same methods can be abused to perform highly dangerous attacks [82]. The `URLRequest` class

allows a Flash applet to create HTTP requests using `GET` or `POST` methods; the `navigateToURL()` method takes two parameters: a `URLRequest` object containing all the information needed to perform the HTTP request, and an optional `String` that determines which frame in the current browser to open the new webpage specified in the `URLRequest`. ActionScript allows developers to pass URL values `navigateToURL()` obtained from external sources such as `FlashVars` (see §2.14), creating a vulnerability that attacks can easily manipulate to perform cross-site scripting [5].

Real-world Example:

Reconfiguring Home Router. A proof-of-concept example has been shown describing how these two methods can be used in conjunction to effortlessly reconfigure a well-known home router, *BT Home Hub*, distributed by a leading British telecommunications company [82]. The attack uses these ActionScript methods to request Universal Plug and Play (UPnP) functionality via the Simple Object Access Protocol (SOAP) (CVE-2008-1654).

2.6 Flash-based Cross-Site Scripting (XSS)

A *Cross-Site Scripting (XSS)* attack involves the injection of a malicious, client-side script into a vulnerable website that can be executed at the privileges of the victim page. When an unsupecting user visits the victim page, the script can exploit the user's trust to perform malicious activity.

In a classic XSS involving Flash, an attack can be conducted by passing in a malicious script through *global flash variables* (see §2.14) [18]. In a *Cross-Site Flashing (XSF)* attack [71], the attacker-injected malware is a malicious *Flash applet*. Subsequently, when the applet runs on the client browser's Flash plug-in, it compromises the plug-in and allows the attacker to abuse native Flash functionality in the client browser, creating arbitrary code execution possibilities. Fig. 2.6 presents a list of several ActionScript classes, methods and

ActionScript 2	ActionScript 3
getURL()	flash.net.URLLoader.load()
MovieClip.loadVariables()	flash.net.URLStream.load()
TextField.htmlText	flash.text.htmlText
loadMovie()	flash.external.ExternalInterface.call()
loadMovieNum()	flash.external.ExternalInterface.addCallback()
LocalConnection.connect()	flash.net.LocalConnection
NetStream.play()	flash.net.NetStream.play()
SharedObject.getLocal()	flash.net.SharedObject.getLocal()
SharedObject.getRemote()	flash.net.SharedObject.getRemote()
XML.load()	
XML.sendAndLoad()	
Sound.loadSound()	
LoadVars.sendAndLoad()	
FScrollPane.loadScrollContent	

Figure 2.1. Potentially dangerous ActionScript classes, methods, and properties

variables that pose severe risks for XSS. It is vital for developers to conduct adequate validation and sanitization of user input leaking into any of these methods to defend against XSS and XSF attacks [87, 49, 79].

Real-world Example:

Flash-based XSS in Yahoo! Mail. In June 2013, a Flash XSS vulnerability was discovered [85] in the `IO Utility` of the Yahoo! User Interface library [46]. The utility contained a Flash applet, `io.swf` which used user inputs as parameters in an `ExternalInterface.call()` without validation, rendering malicious JavaScript execution feasible in the `io.swf` container. The applet `io.swf` was hosted in the Yahoo! Mail main domain, creating an appalling vulnerability in Yahoo! Mail; users logged into Yahoo! Mail were able to access the applet at `http://us-mg5.mail.yahoo.com/neo/ued/assets/flash/io.swf`, enabling attacks such as read access to other Yahoo Mail! users' inbox by sending a cleverly crafted URL to them [85].

Please see the *`SWFUpload` example* (§2.5) and the *`Gmail` example* (§2.14) for more examples of Flash-based XSS.

2.7 Flash-based Cross-Site Request Forgery (CSRF)

In a *Cross-Site Request Forgery (CSRF)* attack a malicious website conducts an attack on a trusted website employing a user's browser [104]. While CSRF attacks are often confused

with the more well-known XSS, the two are strategically quite different; with CSRF, the attack is based on the exploitation of naïve web servers, which accept client requests without validation. In XSS, the trust of the user is targeted, whereas in CSRF, the cross-origin trust of the user's browser is targeted.

Flash-based CSRF attacks take advantage of several clever abuses of the language. For example, ActionScript can be used to craft spoofed HTTP headers to bypass HTTP `referer` header checking, thereby defeating a mechanism used to prevent CSRF attacks [18].

Additionally, Flash proves handy for CSRF distribution, because of the following features: (i) Flash applet's same-origin policy is determined from the origin of the Flash, not the embedding page; (ii) malicious CSRF code can be easily obfuscated and placed inside a Flash applet (see §2.10); (iii) Flash shared objects (see §2.1) enable manipulation of date and time of attack easily, and maintain hack status; and (iv) the facts that stolen data can be retrieved back to the Flash applet, and cross-domain POST can be used in place of GET, facilitate theft of large-sized data [39].

Real-world Example:

CSRF vulnerability in IBM Tivoli Endpoint Manager Software Usage Analysis (SUA) application. The application used Flash's *Action Message Format* (AMF) (format used to send messages between a Flash applet and a remote service) to serialize messages between web clients and the SUA server. A CSRF attack was feasible by attackers creating malicious AMF messages and deceiving an authenticated SUA user into visiting an attacker-controlled website (CVE-2013-0452) [98].

Another example of CSRF includes the *`Plupload` example* (§2.5).

2.8 Flash-based Heap Spraying

In a *Heap Spraying* attack, the attacker repetitively writes, or *sprays*, premeditated byte sequences into a large section of the victim program's heap. The shellcode is duplicated, and

augmented with long sequences of *NOP (No Operation) sleds*, to provide an increased jump target to maximize the probability of success. A second exploit is required to point control flow to jump to the sprayed code.

Flash-based heap spraying often employs the `flash.utils.ByteArray` class to conduct heap spraying attacks. The `ByteArray` class, originally meant for facilitating developer interaction with binary data, unfortunately facilitates heap spraying as well, due to this very same characteristic of ease of byte-level access, including byte-level access to chunks of data, read and write access to arbitrary bytes, and read and write access to binary representation of integers, floating point numbers, and strings. Additionally, the implementation of the `ByteArray` class in the ActionScript 3 VM uses a contiguous block of memory and is expanded dynamically for storing array contents [78].

In the attack, two `ByteArray` instances are used—one for the shellcode, and the other as the heap spray target. The shellcode-loaded `ByteArray` is repeatedly copied into the target `ByteArray`, thereby spraying the latter with the desired malicious payload [78].

Real-world Examples:

Watering Hole Attack on the Council on Foreign Relations Website. On December 27, 2012, the Council on Foreign Relations (CFR) website [23] was compromised, and subsequently was used as a medium to serve malware to its visitors [57]. The final stages of the exploit used a Flash applet, `today.swf` to conduct a heap spray attack against users using Internet Explorer version 8 (CVE-2012-4792) [57, 56].

Flash Heap Sprays without JavaScript support. While most heap sprays involving Flash borrow help from JavaScript, several instances of purely Flash-based heap spray attacks exist. One example involves a Microsoft Office Word document containing an embedded uncompressed malicious Flash file with heap spraying code. The document was a news article on iPhone batteries (CVE-2012-1535) [14].

2.9 Flash-based JIT Spraying

Just-In-Time (JIT) spraying attacks abuse JIT compilers to defeat code control-flow protections, such as those based on *Address Space Layout Randomization* (ASLR) and *Data Execution Prevention* (DEP). ASLR reduces the reliability of attacker payloads by randomizing the locations of binary code sections in victim processes. This frustrates attackers' ability to predict valid code pointer values, and therefore invalidates many payloads containing such pointers. DEP restricts write- and execute-access to most code and data bytes, respectively, impeding malicious code-injections. However, JIT compilers typically open loopholes in both defenses by dynamically allocating writable, executable data sections for JIT-compiled code at discoverable locations.

The Adobe Flash player proves a prime target for JIT spraying as the ActionScript 3 Virtual Machine (AVM2) uses JIT-compiler enhancements to speed up execution [3]. Additionally, the Flash player implements a vast number of features that all unfortunately aid in conducting a JIT spray attack, including a large GUI library, a JIT 3D shader language, embeddable PDF support, multiple audio and video embedding and streaming options, and of course the scripting VM [17].

JIT spraying was first introduced, using Flash, in BlackHat D.C. 2010 (CVE-2010-1297) [90, 16, 17].

Real-world Example:

Evolution of Flash-based JIT spraying, and Adobe's Continuous Reactions. Starting from the BlackHat D.C. demo by Blazakis, it has been a back-and-forth war between JIT spraying developers and Adobe [91]. Since the first demo, Adobe has introduced various features in the Flash compiler to mitigate JIT spray vulnerabilities, including constant folding, and introduction of NOP-like instructions that break the continuity of shellcode.

An extremely sophisticated example of JIT Spraying (mitigated by Adobe in Flash version 11.8) uses *ROP [92] info leak gadgets* and heap spraying to defeat prior Adobe mitigations

(such as the introduction of random NOP-like instructions). The attack exploits a vulnerability in Windows 7/Internet Explorer 9 (CVE-2012-4787). Adobe's mitigation to this attack implemented a technique called *constant blinding*—XORing the value of a user-supplied integer, used later in an assignment or function argument, with a random cookie generated at runtime [91].

2.10 Obfuscation

Binary code obfuscation techniques are widely used by legitimate Flash developers to hinder reverse-engineering of Flash applets and protect intellectual property. Consequently, Commercial Off-The-Shelf (COTS) Flash obfuscators, such as SWFEncrypt [9], Kindi secureSWF [55], DoSWF [28], and DCoM SWF Protector [27], are found aplenty in the market. Unfortunately, the same techniques and tools are often exploited by attackers to evade intrusion detection systems.

Malicious obfuscation techniques include name substitution, improper use of keywords, removal of debugging- and meta-information, introduction of redundant and cyclic control flows, and inclusion of unrealizable or illegal code. Well-executed obfuscation makes manual or COTS tool decompilation extremely challenging; typically, decompilation attempts using these techiques lead to invalid or unintelligible source code.

Flash-based malicious obfuscation takes advantage of various ActionScript Virtual Machine features and methods. For example, one obfuscation technique employs a combination of the `Loader.loadBytes()` method and the `DefineBinaryData` SWF tag [3]. `Loader.loadBytes()` allows dynamic loading of Flash applets; the `DefineBinaryData` tag allows arbitrary binary data to be embedded into the tagged section of a SWF file; the data becomes available to ActionScript through a `ByteArray` instance at runtime, which can be used as input to `loadBytes()` for evaluating a new Flash file. This gives attackers the potential to create a series of

encrypted malicious Flash files, embedded within one another [78]. Identifying the embedded exploits by a simple examination of the external Flash file is extremely challenging [35].

Several other SWF tags [3] and ActionScript classes present similar powerful obfuscation-aiding mechanisms, including the `DoAction`, `ShowFrame` tags (ActionScript 2), and `SymbolClass`, `DefineBits`, `DoABC` tags (ActionScript 3) [61, 62]. Obfuscation techniques that adopt any of these features turn out to be deviously powerful because they house arbitrary dynamic code generation capabilities within operations that are widely used for legitimate purposes.

Additionally, ActionScript allows string identifiers of built-in ActionScript variables and methods to be stored in obfuscated form, and de-obfuscated at runtime when needed. Nearly all Flash instructions represent object member names as string values at the binary level, making it acutely difficult to robustly determine which methods are called by even a standard, non-obfuscated Flash program. The ubiquity of obfuscation only makes this frightful situation worse.

Real-world Example:

Peeling Obfuscation Like An Onion. Several instances of real-world Flash malware use the attacker-favorite obfuscation technique of wrapping a series of malicious Flash files one into another. One such real attack example involves wrapping an obfuscated Flash 8 exploit (CVE-2007-0071) into multiple layers of a Flash 9 file [35].

An Avast! Blog article describes in detail an interesting real-world sample. The malware uses the `DefineBinaryData` and `SymbolClass` tags to load one obfuscated Flash into a byte array, and subsequently use the latter in a `DoABC` tagged code section, creating a layered Flash exploit. The main point of the article was to demonstrate the immense ease by which Flash files can be obfuscated, making obfuscators increasingly rampant in the Flash malware world [61].

2.11 Type Confusion Exploitation

In a *type confusion* attack, the attacker abuses a vulnerability created by a discrepancy in data type representation [30]. Type confusion attacks are particularly insidious, since they can bypass DEP and ASLR without any kind of heap or JIT spraying [78]. This kind of vulnerability is often found in software components that bind more than one language [30]. For example, in Flash, type confusion vulnerabilities have appeared in the binding layer between ActionScript and native code. Improper error-checking by compilers while converting between fundamental and user-defined types can also cause type confusion vulnerabilities [30].

Both the ActionScript 2 and ActionScript 3 virtual machines have been targeted for type-confusion vulnerability exploitations. The ActionScript 3 virtual machine uses data types known as *atoms* and *type-tags* to support runtime type detection when a variable's type is not specified at the source level. Additionally, native code resulting from the JIT compilation uses native data types; therefore, when a native method is called, the result is wrapped into a type-tag for use by the VM [78].

This kind of *type-tag wrapping* has led to type confusion vulnerabilities. For example, in one attack, the identifier of a class `A` is changed to the same name as another class `B` in the bytecode, resulting in type confusion'. This results in calls to the `B`'s methods actually calling native code implementations of class `A`. Upon return from the native code method, the wrapped type-tag of the result depends on the types defined in `B`. The mismatch between `A`'s native code methods being called, and `B`'s return types being used creates an exploitable vulnerability, which can be used for various attacks such as leaking objects' memory addresses, reading arbitrary memory addresses, and gaining control of execution (CVE-2010-3654) [78].

FlashDetect [78] presents a very interesting technique for bypassing DEP in Flash. The technique involves discovering the address of the `VirtualProtect` function (used for changing

the protection on a region of committed pages in the virtual address space of a calling process) in the Flash player DLL, through an ActionScript object [78].

Real-world Example:

Massive E-mail Attachment Exploits Targeting Type Confusion Vulnerabilities. Attacks were conducted exploiting type confusion vulnerability in several versions of the Flash Player. The vulnerability was exploitable upon supplying a corrupt response to *ActionScript Message Format0* `error` field, giving attackers the ability to execute arbitrary code with user privileges [86].

Several attacks were conducted—each consisted of sending victims custom crafted emails with malicious attachments. The attachments were `.doc` files that contained references to a malicious Flash file on a remote server. The .doc files also contained a hidden malicious payload in encrypted form. The Flash file, when downloaded and played using a local vulnerable Flash Player, sprays the heap with shellcode and triggers CVE exploit. When executed, the shellcode finds the encrypted malicious payload in the original document, decrypts and executes it [95].

The emails were targeted at several members of the U.S. defense industry, and contacted servers hosted in China, Korea, and the United States to acquire the necessary data to complete the exploitation [95]. Most recently, an emailed called "World Uyghur Congress Invitation.doc" was sent, targeting the World Uyghur Assembly [81].

2.12 Vulnerabilities in Flash Parser and Analysis Tools

Flash parser, runtime analysis and decompilation tools have also been targets of attacks [37].

Several attacks have been conducted due to lack of validation of ActionScript 2 *jumps*, thereby allowing code execution to jump to non-code locations in the Flash file. The ActionScript architecture confines bytecode to specific *tagged* sections in the binary (.swf) file, such as in `DoAction` or `DoInitAction` tags [4]. The Flash VM does not verify that the jump

location exists within the original tagged section, malware can therefore jump outside the section to execute bytecode elsewhere in the file. Many Flash disassemblers and decompilers such as Flasm [59] and Flare [58] only examine tagged sections designated for bytecode, and therefore typically miss malware that has jumped outside. In fact, much malware has used the fact that most Flash analysis tools do not examine tagged sections not designated for bytecode to hide malicious executable code [35].

Ford et al. [35] additionally point out how tag validation is a problem in itself—the Flash VM does not validate data inserted into tags. Furthermore, the Flash VM also quietly ignores invalid tag types; invalid tag types can be created and filled with ActionScript bytecode, which can be used for attacks such as above [35].

Real-world Examples:

Improper Parsing of Various Entities. Flash Player's sloppy parsing has led to innumerable attacks, some examples are highlighted below.

Iranian Oil and Nuclear Situation Used As Bait for Defense Employees. In March 2012, targets were sent emails with an "Iran's Oil and Nuclear Situation.doc" attachment. The attachment consisted of an embedded malicious Flash applet, which when run plays a malformed MP4 file. An MP4 parsing error in the Flash Player (CVE vulnerability CVE-2012-0754) while parsing caused memory corruption and subsequently the downloading and installing of a Trojan, identified by many anti-virus products as "Graftor" or "Yayih.A". The targets of the attack were suspected to be members of the U.S. defense industry [22].

Parsing TrueType Font. The Flash Player did not perform necessary validation while parsing a TrueType font. While parsing, the Flash Player was supposed to calculate the size of data to be copied based on a specific field; however, due to the lack of proper validation, an integer overflow vulnerability was created, which exposed the VM to the execution of arbitrary malicious code [38].

Proof-of-Concept Parsing Error Exploit. Improper validation of integer value by the parser causes vulnerability (CVE-2009-1869) that is exploited by a cleverly executed proof-of-concept heap-spray attack on Windows XP SP3 with IE7. The source code is available on Google Code [40].

2.13 JavaScript and HTML Script Injection

Most JavaScript code injection is conducted through the `ExternalInterface` class that Adobe provides for ActionScript-JavaScript communication. For more information about JavaScript code injection, please see §2.5.

Several security risks are posed by a combination of two ActionScript-HTML interactive features. The first is the ability of the Flash VM to interpret HTML tags such as the *anchor* (`<a>`,`</a>`) and *image* (`<img>`) tags. The second is the ability of HTML code to invoke public and static ActionScript methods through a special protocol for URLs in HTML text fields. In ActionScript 2, this is achieved through the `asfunction` protocol, which takes two arguments `function` and `parameter`, where `function` is a string identifier for an ActionScript 2 function, and `parameter` is the parameter to the former [80]. In ActionScript 3, this is achieved through the `flash.events.TextEvent` class, by listening for click events from HTML, using the `TextEvent.LINK` property for transferring information to ActionScript, and adding an event handler in ActionScript.

Using these features, HTML code can perform cross-scripting to JavaScript through ActionScript, call an ActionScript method directly, call a particular SWF file's public functions, or call native static ActionScript directives such as `flash.system.Security.allowDomain`. While these features provide powerful convenience in ActionScript-HTML interactions, they obviously pose several security risks. For example, with the `Security.allowDomain`, it is easy to allow access to a malicious domain [80].

The HTML image tag allows the `src` attribute to take files with `.jpg` and `.swf` extensions. This of course presents an easy cross-site scripting vulnerability for Flash Player versions 7 or less, or if the `AllowScriptAccess` attribute is used imprudently. Additionally, if a `.swf` extension is added to a malicious JavaScript code in the `src` attribute, such as `<img src='javascript: alert(foo); //.swf'>`, the Flash plug-in will go ahead and run the Flash binary [80, 36].

Real-world Example:

Cross-Platform Attack Using Malicious JavaScript Injection and Flash. Various forums, such as "windows7forums.com" and "www.macrumors.com", were attacked by a malicious JavaScript-Flash combination originating from malicious site "www.priceofinsurance.com". The attack was cross-platform, launched from multiple attack sites, with the JavaScript hosted on a distribution server called "www.googlefreehosting.com". The JavaScript triggered the Flash file `4.swf`, which was used for data collection, and perhaps for click fraud and privacy violation [34].

2.14 Flash Parameter Injection

Users can pass values to a Flash applet from its embedding container (typically an HTML environment) into *global variables* inside the applet. In ActionScript 2, global variables are pre-pended by keywords `_root`, `_global` or `_level0`. In ActionScript 3, these are deprecated, and replaced by a single global variable `root`. Arguments to a Flash movie using ActionScript 3 can be passed into the `root.loaderInfo.parameter` object, which will contain name-value pairs of the parameters passed in from HTML. In *flash parameter injection*, an attacker abuses this facility to take control of other objects within the Flash applet, as well as full control over the embedding page's DOM model.

Three popular ways to pass values to Flash applets include:

1. *Passing arguments using direct reference.* This method references the Flash file directly and passes the arguments through the URI (this is the same as HTTP parameters

```
<body>
  <object>
    type = "application/x-shockwave-flash"
    data= "myMovie.swf?a=5&b=hello"
    width = "600" height="345">
  </object>
</body>
```

Figure 2.2. HTML code with injection of Flash parameters using the Embedded URI method

using the GET method). For example, in `http://URL/myMovie.swf?a=5&b=hello`, the global variable `a` receives the value `5` and `b` receives the value `hello`. When this method is used the Flash file is not embedded in the original HTML page, but is instead embedded in a second "dummy" HTML page that is created automatically.

2. *Embedded URI.* This method passes the arguments in the URI of the embedded object in the original HTML page. For example, consider the HTML code in Fig. 2.

3. *Using the `FlashVars` parameter in `<object>` and `<embed>`* . Variables passed via `FlashVars` will go into the `_root` level of the Flash movie. For example, in the `Object` tag, `<PARAM NAME=FlashVars value="foo=bar">` will assign `bar` to `foo` on the `_level0` timeline. All variables passed in through `FlashVars` have to be strings.

An uninitialized global variable usually has whatever value was in its memory location before it was declared. However, uninitialized global variables are assumed to be `FlashVars`—variables that can be declared and passed into the Flash applet for use from the embedding container such as the embedding HTML webpage, through the `<object>` and `<embed>` tags. Irrespective of ActionScript's version, `FlashVars` can be easily abused since there are many ways to pass them into the Flash applet [80].

Potential unsafe operations include: (1) location of the Flash movie is retrieved through a URL parameter: `http://host/index.cqi?movie=movie.swf?globalVar=e-v-i-l`; (2) a victim is lured to click on a link such as `http://host/index.cqi?language English%26globalVar=e-`

v-i-l, which happens when global flash variables are received from HTML parameters without sanitization; (3) global variable is injected into the Flash movie embedded inside the DOM: http://host/index.htm#&globalVar=e-v-i-l [80].

Real-world Example:

XSS in Gmail Based Services through Flash Parameter Injection. Users of the staple applications Gmail and Google Apps became vulnerable to full account hijacking through a Flash-based XSS vulnerability [10]. Internally, Gmail used a Flash applet called uploader-api2.swf for file uploads; the applet used two user-inputs, (apiInit and apiId), as parameters to ExternalInterface.call(). A proof-of-concept script injection attack was conducted before Google patched the vulnerability; the attacker was able to execute arbitrary JavaScript code on mail.google.com (the Gmail domain) by setting apiInit to eval and apiId to payload code, and then enticing a user to click on a malicious link with these variables set: https://mail.google.com/mail/uploader/uploaderapi2.swf?apiInit=eval&apiId=⟨*payload*⟩.

The malicious JavaScript ran in the context of active Gmail sessions; attackers were able to fully impersonate their victims and steal information from their accounts [10].

An interesting point to note is that this attack can be executed transparently in browsers such as Firefox and Google Chrome—since Flash is executed on the client side, the values of apiInit and apiId (the malicious payload) can be hidden from the server by adding the "#" sign before the query part of the URL: https://mail.google.com/mail/uploader/uploaderapi2.swf#?apiInit=eval&apiId=⟨*payload*⟩. The receiving server sees a parameter-less request: https://mail.google.com/mail/uploader/uploaderapi2.swf, and therefore concludes it to be benign; Gmail loads uploaderapi2.swf without any parameters. However, at the client side, a successful exploitation occurs since the Flash player refers to the whole URL, including the attack payload, which comes after the "#" sign [80, 10].

2.15 Flash-based Malvertisements

Web advertisements are an important source of revenue for webpage publishers. However, recently they are gaining traction as a tenacious vehicle for various malicious activities such as stealing personal and banking details, corrupting data and webpages, and spreading viruses and spyware. According to the Symantec annual *Internet Security Threat Report*, malicious advertising, or *malvertising* may be the primary reason why drive-by-web attacks increased by one-third from 2011 to 2012 [96]. According to Cisco's *Annual Security Report*, malvertisements comprise of 16% of total web malware, mainly because a single online advertisement is typically used to fuel revenue for many webpages [20]. According to a 2010 report by the Internet security company Dasient, there were a staggering 1.3 million malvertisements viewed daily [24].

The user's trust is paramount to a malvertisement; therefore, many malvertisers target popular sites such as The New York Times, the London Stock Exchange pages, and top social networking sites such as Facebook. Malvertisers also wait until the ad has been well-circulated before triggering malicious activity, since alarming users at the start of the malicious campaign would defeat their purpose.

Websites usually contain embedded resource containers for an advertisement, with a reference to the advertisement, which is typically hosted by a third-party ad network. When a user visits the site, the webpage loads and communicates with the advertising third-party network requesting a relevant ad. For a Flash-based ad, after checking that the user's browser is Flash-enabled, the network sends relevant code back to the user's browser that needs to be inserted for the webpage to display the ad. The code in turn downloads the actual ad Flash binary file from the ad network [35].

Flash provides malvertisers a more sophisticated, powerful, and flexible platform than JavaScript/HTML for numerous reasons. Firstly, Flash allows the encoding of detailed

ActionScript instructions for expressing the business/domain logic within the ad itself. Secondly, ActionScript tends to be more difficult to examine than JavaScript, providing attackers more flexibility in their attack code. ActionScript also allows malvertisements to judge time and location targets—for example, they can defer malice until they have been deployed successfully on the ad network and target certain geographic areas using the ActionScript `Date` class. Additionally, *Flash Shared Objects* (see §2.1) contain a timestamp attribute that can be used by malvertisements to determine whether the malicious activity has been performed on a particular machine within a particular time frame, avoiding a redundant attack to evade suspicion. The `LoadVars.load()` method can be used to send HTTP requests, and navigation methods such as `MovieClip.getUrl()` can be used to redirect to particular new sites. Finally, Flash-based malvertisements have access to a plethora of COTS obfuscators such as SWF Encrypt [9] to evade detection and defeat casual SWF decompiler tools [35, 105, 106].

An interesting point to note here is that Flash-based malvertisements borrow heavily from several vulnerability and attack types that this paper presents from §2.1 to §2.14. For example, Flash-based malvertisements often use the obfuscation technique of layered-embedding of malicious Flash files as discussed in §2.10, and `getUrl()` and `navigateToURL()` methods for malicious redirection to conduct phishing, pharming and drive-by-download attacks (§2.1–2.3). Malvertisements also use the Flash platform for `ExternalInterface` attacks (§2.5) and as a vehicle for heap or JIT spraying (§2.8, §2.9).

Real-world Examples:

DDoS Attacks on Stop Malvertising Site. In July 2011, a DDoS attack was conducted on the *Stop Malvertising Site* using a series of cleverly crafted mini Flash files. The files came from three different domains, `data-ero-advertising.com`, `flatfee.ero-advertising.com`, and `www.ero-advertising.com`. Analysis of several of the malicious Flash advertisements showed embedded `Sprite` and `MovieClip` classes with Flash malware [54].

Malvertising Attack on American Idol Website. Another interesting attack was on the American Idol fan page just prior to the competition finale of 2011. The attack consisted of an abuse of the `ExternalInterface` class to perform malicious redirection to the malicious ad network [26].

CHAPTER 3

ANALYSIS

3.1 Scientic Research Survey Methodology

To better understand the scientific community's responsiveness to Flash security threats, we surveyed publications in the six highest-impact, security-themed, computer science venues, excluding venues that focus mainly on cryptography. The top six such venues ranked by Google's h5 index as of November 2013 are *ACM Symposiym on Information, Computer and Communications Security (CCS)*, *USENIX Security Symposium*, *IEEE Symposium on Security and Privacy (S&P)*, *IEEE Transactions on Dependable and Secure Computing (TDSC)*, *ACM Transactions on Information and System Security (TISSEC)*, and *European Conference on Research in Computer Security (ESORICS)*.

We read the abstracts and introductions of all publications in these venues between 2008–2012, and all publications in these venues in 2013 published to-date, manually identifying those papers that are web-related, and conservatively classifying all works that make more than anecdotal reference to Flash or ActionScript as Flash-targeting.

Figure 3.1 illustrates the results. Overall, 9.6% (100/1045) of surveyed publications are devoted to web security. Of these, only $15/100 = 15\%$ papers target Flash (CCS: [66, 1, 41], IEEE: [60, 76, 100, 47, 67, 101, 64, 97, 19, 13], and USENIX [52, 44]). IEEE S&P has the greatest percentage, devoting $10/24 = 41.7\%$ of web security publications to Flash. The remaining five venues collectively devoted only $5/76 = 6.6\%$ of web security publications to Flash. This indicates that in general the scientific community's attendance to Flash security issues has been disproportionately small relative to the role of Flash in real-world attacks.

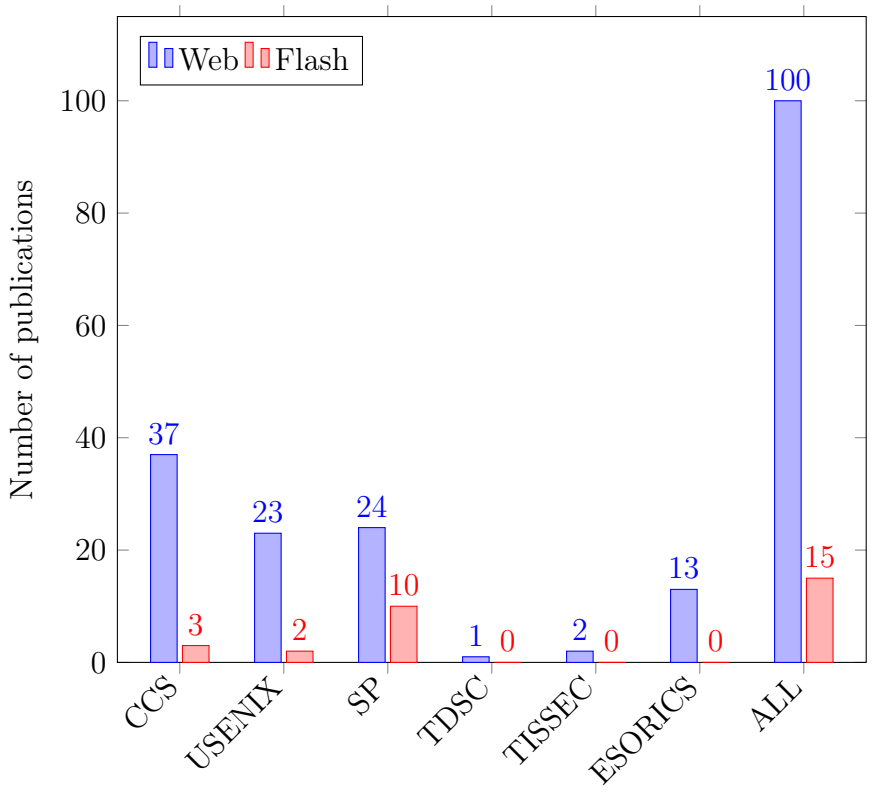

Figure 3.1. Flash presence in the top six security publication venues in 2008–2013.

3.2 CVE Collection and Investigation

3.2.1 Total Flash-relevant CVEs Over the Years

Figure 3.2.1 presents the total number of Flash-relevant CVEs per year. It is interesting to note that there was a drastic increase in the number of recorded Flash-relevant attacks/vulnerbilities from the year 2008 to the year 2010, followed by a gradual decrease until this point in 2013.

3.2.2 Flash-relevant Attack/Vulnerbility Trend Evolution Over the Years

Figures 3.2.2 and 3.2.2 present the evolution of attack/vulnerability types in the years 2008–2013. Each figure shows the evolution of the number of attack/vulnerability type according

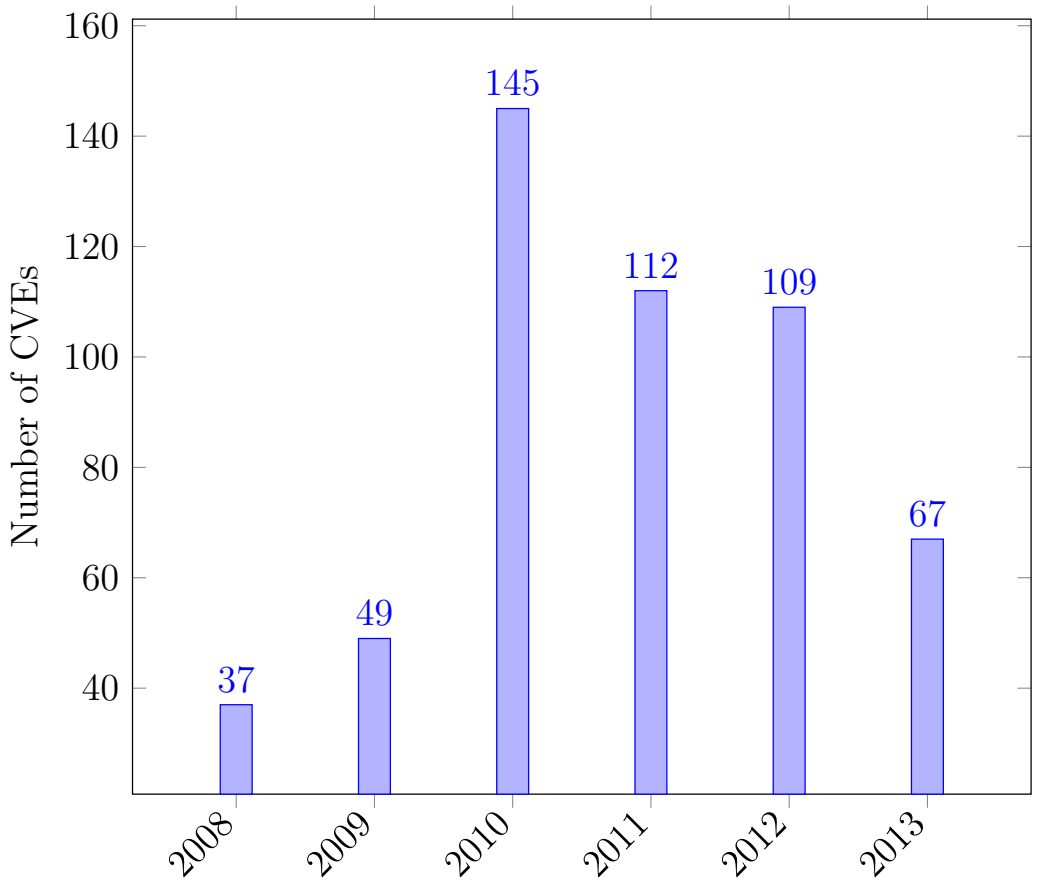

Figure 3.2. Number of Flash-relevant CVEs by Year from 2008–2013.

to the taxonomy we presented in §2. The taxonomy is provided in abbreviated form again in Table 3.1, for the reader's convenience. We perfomed classification in two steps:

1. In the first step, we classified CVEs purely using (i) attack/vulnerability type attribution given in the description provided by each CVE entry in the Mitre database [72], and (ii) corresponding CWE identification scores wherever available. Figure 3.2.2 shows results based on this first step. However, we realized that we could make more refined attribution inferences in several cases based on attack/vulnerability techniques and patterns in exploiting ActionScript language features and Flash architecture details that we studied in §2, leadng to step two (below).

Table 3.1. CVE Classification Legend

Abbr.	Description	Section
PHI	Flash-based Phishing	2.1
PHA	Flash-based Pharming	2.2
DNS	DNS Rebinding	2.2
DBD	Drive-by-Download and Drive-by-Cache	2.3
SOP	Same-Origin Policy Abuse	2.4
EI	Attacks using `ExternalInterface`, `URLRequest`, and `navigateToURL`	2.5
XSS	Cross-site Scripting	2.6
CSRF	Cross-site Request Forgery	2.7
HSP	Heap Spraying	2.8
JIT	JIT Spraying	2.9
OBF	Obfuscation	2.10
TYP	Type Confusion Exploitation	2.11
PAR	Vulnerabilities in Flash Parser and Analysis Tools	2.12
FPI	Flash Parameter Injection	2.14
JHS	JavaScript and HTML Script Injection	2.13
MCC	Malvertisements, Click-Fraud and Click-Jacking	2.15
OVF	Integer- and Buffer-Overflow Vulnerabilities	Not Flash-specific
DoS	Denial of Service	Not Flash-specific
EXC	Arbitrary Code Execution	Not Flash-specific
REA	Unpermitted Read Access, especially sensitive information	Not Flash-specific
UNK	Unknown	Not Flash-specific

2. In the second step, we re-traversed through the entire CVE list and added an extra layer of classification whenever possible. This layer was inferences based on minute details in the attack description, and external sources, such as news articles or research papers, that listed a particular CVE entry in reference to a particular attack/vulnerability reported. Figure 3.2.2 shows results based on this seond step.

Note that Table 3.1 contains five classes not present in our taxonomy, namely OVF (Integer- and Buffer-Overflow Vulnerabilities), DoS (Denial of Service), EXC (Arbitrary Code Execution), REA (Unpermitted Read Access, especially sensitive information), and

UNK (Unknown). The first four categories are standard vulnerabilities, and not Flash-specific, but we decided to add these classifications for completeness in the analysis. "Unknown" classifications were assigned to CVEs whose descriptions lack the necessary specificity for a correct classification, and for which no pertinent external sources were found that clarified the ambiguity.

3.3 Classification Challenges

In this section, we present a series of examples that highlight the challenges we faced in classifying CVE entries.

3.3.1 Example 1: CVE-2013-0634

Figure 3.3.1 displays the contents of the CVE entry CVE-2013-0634. From the entry, the reader cannot obtain much information about the attack, except for the CWE number, affected Player and OS versions, date, and that the vulnerability exposes the system to arbitrary code execution and denial of service. The CWE number, CWE-119, corresponds to *Buffer Errors*, which also does not provide much information. The following is a description of the Buffer Errors category (see Table 3.2):

> Buffer overflows and other buffer boundary errors in which a program attempts to put more data in a buffer than the buffer can hold, or when a program attempts to put data in a memory area outside of the boundaries of the buffer. [75]

Through news articles, however, we realized that CVE-2013-0634 is actually a serious vulnerability which was exploited by a wave of drive-by-downloads, affecting millions of computers, and important groups such as the aerospace industry [89, 31].

3.3.2 Example 2: CVE-2011-2836

Figure 3.3.2 displays the contents of the CVE entry CVE-2011-2836. The description provided in the CVE entry does not provide enough detail to make accurate inferences on what

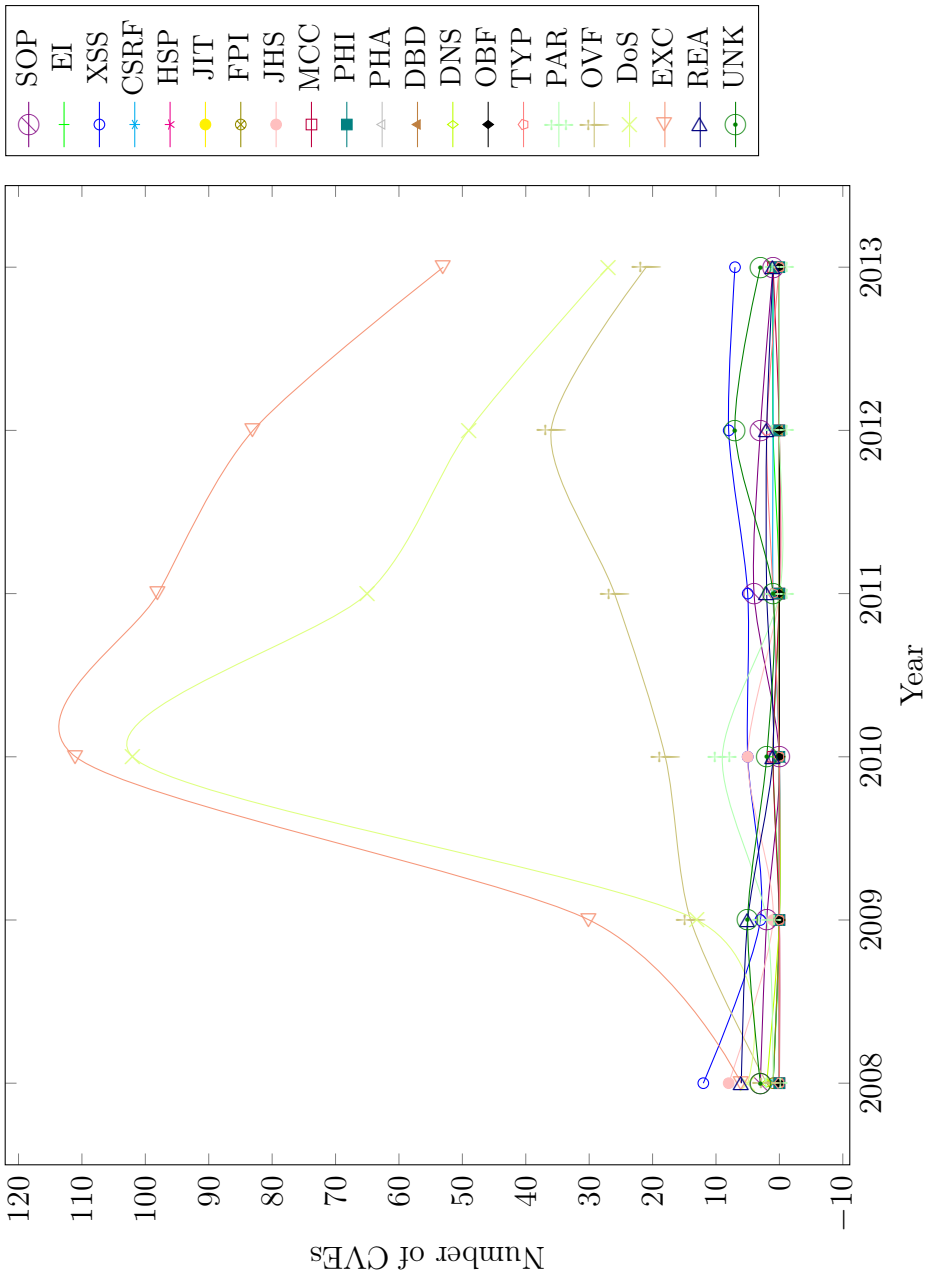

Figure 3.3. Evolution of Flash-relevant vulnerabilities and attacks between 2008-2013.

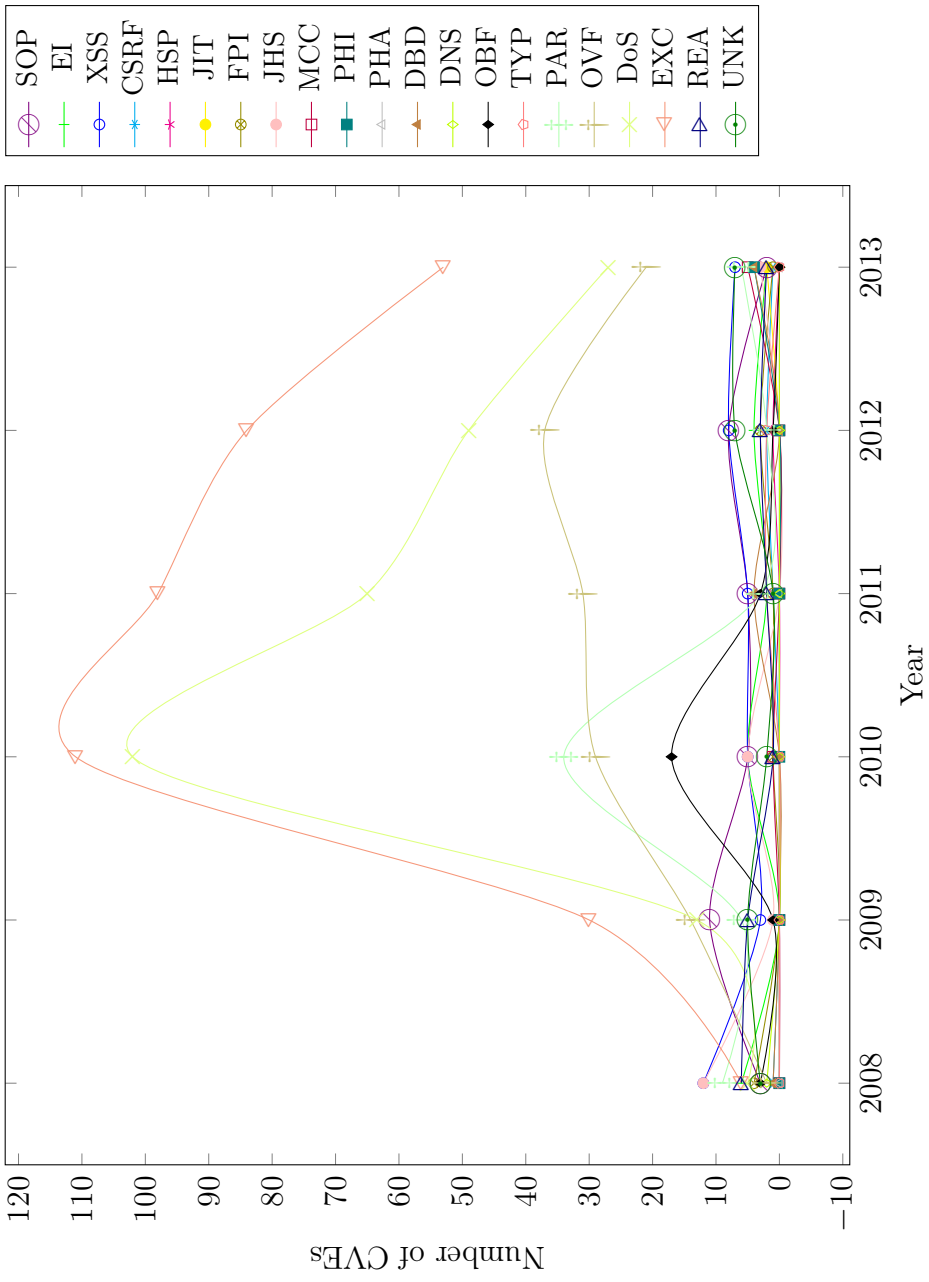

Figure 3.4. Evolution of Flash-relevant vulnerabilities and attacks between 2008-2013, including inferences.

CVE Entry No.	CWE-ID	Description
CVE-2013-0634	CWE-119	Adobe Flash Player before 10.3.183.51 and 11.x before 11.5.502.149 on Windows and Mac OS X, before 10.3.183.51 and 11.x before 11.2.202.262 on Linux, before 11.1.111.32 on Android 2.x and 3.x, and before 11.1.115.37 on Android 4.x allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via crafted SWF content, as exploited in the wild in February 2013.

Figure 3.5. Text of CVE-2013-0634

CVE Entry No.	CWE-ID	Description
CVE-2011-2836	CWE-264	Google Chrome before 14.0.835.163 does not require Infobar interaction before use of the Windows Media Player plug-in, which makes it easier for remote attackers to have an unspecified impact via crafted Flash content.

Figure 3.6. Text of CVE-2011-2836

the attack type or vulnerability type is for any useful mitigation strategy. The maximum pertinent information one can glean is that there are multiple components involved, including the Windows Media Player plug-in, the Google Chrome Infobar, which should involve user interaction (but is failing to do so here), and a Flash content used as a vehicle for attack. The CWE number 264 corresponds to "Permissions, Privileges and Access Control", whose description is "Failure to enforce permissions or other access restrictions for resources, or a privilege management problem" (see Table 3.2).

From these, we cannot make any safe inferences with a reasonable probability; the scope of possibilities includes a wide range of attack/vulnerability types.

3.3.3 Example 3: CVE-2011-0627

Figure 3.3.3 displays the contents of the CVE entry CVE-2011-0627. CWE-20 corresponds to "Input Validation", whose description says "Failure to ensure that input contains well-formed, valid data that conforms to the application's specifications. Note: this overlaps

CVE Entry No.	CWE-ID	Description
CVE-2011-0627	CWE-20	Adobe Flash Player before 10.3.181.14 on Windows, Mac OS X, Linux, and Solaris and before 10.3.185.21 on Android allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via crafted Flash content, as possibly exploited in the wild in May 2011 by a Microsoft Office document with an embedded .swf file.

Figure 3.7. Text of CVE-2011-0627

other categories like XSS, Numeric Errors, and SQL Injection." Once again, the description provided in the CVE entry and CWE table clearly do not provide enough detail to make accurate inferences on what the attack type or vulnerability type is for any useful mitigation strategy.

Apart from the obvious OVF and EXC classes, the cause of the vulnerability could be a range of classes—the most likely one is PAR (perhaps the input validation categorization by the CWE score corresponded to a improper parsing vulnerability, which is often the cause of input validation problems); however, the vulnerability could be open to attacks such as drive-by-download (DBD)—consider CVE entries CVE-2013-0633 and CVE-2013-0634 in §2.3, where the vulnerability is very similar to this CVE, but the attack was a famous drive-by-download exploit. See example 1 for more details.

3.3.4 Example 4: CVE-2011-0578

Figure 3.3.4 displays the contents of the CVE entry CVE-2011-0578. CWE-119 corresponds to "Buffer Errors" (described in Example 1). Using this information and the description provided, we added classes DoS and EXC. However, we also inferred TYP based on "ActionScript3 object and improper type checking". This example demonstrates that the CWE numbers used by NVD [75] are not fine-tuned to Flash.

CVE Entry No.	CWE-ID	Description
CVE-2011-0578	CWE-119	Adobe Flash Player before 10.2.152.26 allows attackers to execute arbitrary code or cause a denial of service (memory corruption) via unspecified vectors related to a constructor for an unspecified ActionScript3 object and improper type checking, a different vulnerability than CVE-2011-0559, CVE-2011-0560, CVE-2011-0561, CVE-2011-0571, CVE-2011-0572, CVE-2011-0573, CVE-2011-0574, CVE-2011-0607, and CVE-2011-0608.

Figure 3.8. Text of CVE-2011-0578

CVE Entry No.	CWE-ID	Description
CVE-2012-3414	CWE-79	Cross-site scripting (XSS) vulnerability in swfupload.swf in SWFUpload 2.2.0.1 and earlier, as used in WordPress before 3.3.2, TinyMCE Image Manager 1.1, and other products, allows remote attackers to inject arbitrary web script or HTML via the movieName parameter, related to the "ExternalInterface.call" function.

Figure 3.9. Text of CVE-2012-3414

3.3.5 Example 5: CVE-2012-3414

This CVE entry gives enough information to add classes XSS, EI and JHS. However, the interesting missing class is FPI. A news article describing the attack [84] mentions that the value of `movieName` derived from input by the user and direct loading of the applet by passing parameters in the URL result in the XSS attack (see §2.5). Therefore, we were able to add the FPI attribution here.

3.3.6 Example 6: CVE-2012-2399

Similar to example 8, example 9 also provides enough information to add classes XSS and JHS. However, once again, the missing class is FPI, where the `buttonText` parameter is used to inject malicious script into the Flash component.

CVE Entry No.	CWE-ID	Description
CVE-2012-2399	Not Available	Cross-site scripting (XSS) vulnerability in swfupload.swf in SWFupload 2.2.0.1 and earlier, as used in WordPress before 3.5.2, TinyMCE Image Manager 1.1 and earlier, and other products allows remote attackers to inject arbitrary web script or HTML via the buttonText parameter, a different vulnerability than CVE-2012-3414.

Figure 3.10. Text of CVE-2012-2399

CVE Entry No.	CWE-ID	Description
CVE-2012-3415	Not Available	** RESERVED ** This candidate has been reserved by an organization or individual that will use it when announcing a new security problem. When the candidate has been publicized, the details for this candidate will be provided.

Figure 3.11. Text of CVE-2012-3415

3.3.7 Example 7: CVE-2012-3415

This CVE entry is actually a CSRF vulnerability in the Plupload software, where the attacker was able to make a request to the domain where a Plupload applet was hosted, and was able to read the full response; the applet was embedded on a page using JavaScript. This was facilitated by Flash's same-origin policy problem, where the software developer had whitelisted all domains by default through `Security.allowDomain('*')`, and the attack itself was carried out using the `ExternalInterface` class [84] (see §2.5).

The existing description in the CVE entry is inadequate to infer any of the above. However, using the additional information gleaned by our article survey, we were able to assign categories SOP, EI and CSRF for the attack.

Table 3.2. CWE Cross Section Mapped into by NVD [75].

Name	**CWE-ID**	**Description**
Authentication Issues	CWE-287	Failure to properly authenticate users.
Credentials Management	CWE-255	Failure to properly create, store, transmit, or protect passwords and other credentials.
Permissions, Privileges, and Access Control	CWE-264	Failure to enforce permissions or other access restrictions for resources, or a privilege management problem.
Buffer Errors	CWE-119	Buffer overflows and other buffer boundary errors in which a program attempts to put more data in a buffer than the buffer can hold, or when a program attempts to put data in a memory area outside of the boundaries of the buffer.
Cross-Site Request Forgery (CSRF)	CWE-352	Failure to verify that the sender of a web request actually intended to do so. CSRF attacks can be launched by sending a formatted request to a victim, then tricking the victim into loading the request (often automatically), which makes it appear that the request came from the victim. CSRF is often associated with XSS, but it is a distinct issue.
Cross-Site Scripting (XSS)	CWE-79	Failure of a site to validate, filter, or encode user input before returning it to another user's web client.
Cryptographic Issues	CWE-310	An insecure algorithm or the inappropriate use of one; an incorrect implementation of an algorithm that reduces security; the lack of encryption (plaintext); also, weak key or certificate management, key disclosure, random number generator problems.
Path Traversal	CWE-22	When user-supplied input can contain ".." or similar characters that are passed through to file access APIs, causing access to files outside of an intended subdirectory.

Table 3.2 continued

Name	**CWE-ID**	**Description**
Code Injection	CWE-94	Causing a system to read an attacker-controlled file and execute arbitrary code within that file. Includes PHP remote file inclusion, uploading of files with executable extensions, insertion of code into executable files, and others.
Format String Vulnerability	CWE-134	The use of attacker-controlled input as the format string parameter in certain functions.
Configuration	CWE-16	A general configuration problem that is not associated with passwords or permissions.
Information Leak / Disclosure	CWE-200	Exposure of system information, sensitive or private information, fingerprinting, etc.
Input Validation	CWE-20	Failure to ensure that input contains well-formed, valid data that conforms to the application's specifications. Note: this overlaps other categories like XSS, Numeric Errors, and SQL Injection.
Numeric Errors	CWE-189	Integer overflow, signedness, truncation, underflow, and other errors that can occur when handling numbers.
OS Command Injections	CWE-78	Allowing user-controlled input to be injected into command lines that are created to invoke other programs, using system() or similar functions.
Race Conditions	CWE-362	The state of a resource can change between the time the resource is checked to when it is accessed.
Resource Management Errors	CWE-399	The software allows attackers to consume excess resources, such as memory exhaustion from memory leaks, CPU consumption from infinite loops, disk space consumption, etc.

Table 3.2 continued

Name	**CWE-ID**	**Description**
SQL Injection	CWE-89	When user input can be embedded into SQL statements without proper filtering or quoting, leading to modification of query logic or execution of SQL commands.
Link Following	CWE-59	Failure to protect against the use of symbolic or hard links that can point to files that are not intended to be accessed by the application.
Other	No Mapping	NVD is only using a subset of CWE for mapping instead of the entire CWE, and the weakness type is not covered by that subset.
Not in CWE	No Mapping	The weakness type is not covered in the version of CWE that was used for mapping.
Insufficient Information	No Mapping	There is insufficient information about the issue to classify it; details are unkown or unspecified.
Design Error	No Mapping	A vulnerability is characterized as a "Design error" if there exists no errors in the implementation or configuration of a system, but the initial design causes a vulnerability to exist.

CHAPTER 4

RELATED SURVEYS

Related work on specifically surveying the Flash attack space has been minimal. In this section, we discuss a few works that focus on this domain.

Ford et al. [35] discuss many interesting attacks and vulnerabilities in the Flash attack space, such as obfuscation techniques, malvertisements, parser and decompilation tool issues. However, their survey is about four years ago, in which there have been innumerable changes to the Flash architecture and ActionScript language [35].

A more recent work earlier this year analyzed security threat reports as reported by the United States Computer Emergency Readiness Team (US-CERT). In their work, they record vulnerabilities and threats by company, in which Adobe was ranked third with 14% amongst top seven software giants [12]. The goal of their work was to simply provide a high-level perspective of security threats by company; they do not analyze details of security challenges specifically from the perspective of the ActionScript language or Flash architecture.

In FlashDetect [78], the authors analyze language and architecture features that aid in Flash-based malware. The paper specifically covers obfuscation, heap spraying, JIT spraying, and the usage of ActionScript 3 as an exploit facilitator. However, their study is limited to this scope.

Finally, none of the works attempt to make an exhaustive report of all recorded Flash-relevant attacks and vulnerabilities, including an analysis, as we have provided in this work.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Adobe's Flash platform has undoubtedly become a pervasive technology with a spectrum of rich features. The same flexibility and power, however, lead to a vast range of security issues. Despite the gravity of the problem, little formal study has been done on systematizing this large body of knowledge.

This thesis progresses towards this goal in the following ways. It presents a systematic study of ActionScript security threats and trends, including an in-depth taxonomy of fifteen major Flash vulnerability and attack categories that highlight ActionScript language and Flash architecture features that contribute to those particular attack/vulnerability types. It also presents a detailed investigation of 520 Common Vulnerability and Exposure (CVE) articles reported between 2008–2013. As part of this investigation, we report the evolution of attack/vulnerability trends using both the basic information provided in CVE entries and also smart inferences made using information learned in our taxonomy study. Additionally, we discuss examples that reveal why Flash attack/vulnerability is so challenging and why existing classification techniques such as the CWE scoring system used by NVD does not suffice for the Flash attack realm.

We sincerely hope that our systematic study and classification will raise awareness of several causes of Flash security problems and provide researchers and other members of the Flash community important information needed for building better mitigation techniques.

In future work, we hope to keep pace with the state of the art along with advanced exploits and attacks that surround Flash and endanger the platforms that support it. After analyzing overall Flash attack space, we also plan to study past and future enforcement strategies for the issues that we examined in this work.

REFERENCES

[1] Acar, G., M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel (2013). FPDetective: Dusting the web for fingerprinters. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, pp. 1129–1140.

[2] Acker, S. V., N. Nikiforakis, L. Desmet, W. Joosen, and F. Piessens (2012). FlashOver: Automated discovery of cross-site scripting vulnerabilities in rich internet applications. In *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS)*, pp. 12–13.

[3] Adobe Systems (2007). ActionScript virtual machine 2 overview. http://www.adobe.com/devnet/actionscript/articles/avm2overview.pdf.

[4] Adobe Systems (2012). SWF file format specification, version 19. http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/swf/pdf/swf-file-format-spec.pdf.

[5] Adobe Systems (2013a). ActionScript 3.0 reference for the Adobe Flash platform, package flash.net. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/package.html#navigateToURL().

[6] Adobe Systems (2013b). ActionScript technology center. http://www.adobe.com/devnet/actionscript.html.

[7] Adobe Systems (2013c). Adobe Flash runtimes statistics. http://www.adobe.com/products/flashruntimes/statistics.edu.html.

[8] Alcorn, W. (2013). BeEF: Browser exploitation framework project. http://beefproject.com.

[9] Amayeta (2013). SWF ENCRYPT 7.0. http://www.amayeta.com/software/swfencrypt.

[10] Amit, Y. (2010, March). Cross-site scripting through Flash in Gmail based services. *IBM Security Insider*. http://blog.watchfire.com/wfblog/2010/03/cross-site-scripting-through-flash-in-gmail-based-services.html.

[11] Anthony, S. (2011, April). Security firm RSA attacked using Excel-Flash one-two sucker punch. http://downloadsquad.switched.com/2011/04/06/security-firm-rsa-attacked-using-excel-flash-one-two-sucker-punc.

[12] Baker, Y. S., R. Agrawal, and S. Bhattacharya (2013). Analyzing security threats as reported by the united states computer emergency readiness team (US-CERT). In *Proceedings of the 11th IEEE Intelligence and Security Informatics Conference (ISI)*, pp. 10–12.

[13] Bau, J., E. Bursztein, D. Gupta, and J. Mitchell (2010). State of the art: Automated black-box web application vulnerability testing. In *Proceedings of the 31st IEEE Symposium on Security & Privacy (S&P)*, pp. 332–345.

[14] Blasco, J. (2012). CVE-2012-1535: Adobe Flash being exploited in the wild. *AlienVault Blogs*.

[15] Blasco, J. (2013, February). Adobe patches two vulnerabilities being exploited in the wild. *AlienVault Blogs*. http://www.alienvault.com/open-threat-exchange/blog/adobe-patches-two-vulnerabilities-being-exploited-in-the-wild.

[16] Blazakis, D. (2010a). BHDC2010 - JITSpray demo #1. http://www.youtube.com/watch?v=HJuBpciJ3Ao.

[17] Blazakis, D. (2010b). Interpreter exploitation: Pointer inference and JIT spraying. Technical report, Semantiscope.

[18] Chatterji, S. (2008). Flash security and advanced CSRF. Presented at the OWASP Delhi Chapter Meet.

[19] Chen, S., R. Wang, X. Wang, and K. Zhang (2010). Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proceedings of the 31st IEEE Symposium on Security & Privacy (S&P)*, pp. 191–206.

[20] Cisco (2013). 2013 Cisco annual security report.

[21] Clark, J. (2011, April). RSA hack targeted Flash vulnerability. *ZDNet*. http://www.zdnet.com/rsa-hack-targeted-flash-vulnerability-4010022143.

[22] Constantin, L. (2012, March). Iranian nuclear program used as lure in Flash-based targeted attacks. *CSO Security and Risk*. http://www.csoonline.com/article/701565/iranian-nuclear-program-used-as-lure-in-flash-based-targeted-attacks.

[23] Council on Foreign Relations (CFR) (2013). http://www.cfr.org.

[24] Danchev, D. (2010, May). Research: 1.3 million malicious ads viewed daily. *ZDNet*.

[25] Darryl (2011a, August). Drive-by-cache: Payload hunting. *Kahu Security*. http://www.kahusecurity.com/2011/drive-by-cache-payload-hunting.

[26] Darryl (2011b, May). Flash used in Idol malvertisement. *Kahu Security*. http://www.kahusecurity.com/2011/flash-used-in-idol-malvertisement.

[27] DCOMSoft (2013). SWF protector. http://www.dcomsoft.com.

[28] DoSWF (2013). DoSWF – Professional Flash SWF Encryptor. http://www.doswf.com.

[29] Dowd, M. (2008, April). Application-specific attacks: Leveraging the ActionScript virtual machine. Technical report, IBM Global Technology Services.

[30] Dowd, M., R. Smith, and D. Dewey (2009). Attacking interoperability. Whitepaper, Black Hat USA. http://www.hustlelabs.com/stuff/bh2009_dowd_smith_dewey.pdf.

[31] Ducklin, P. (2013, February). Adobe patches Flash — heads off in-the-wild attacks against Windows and Apple users. *Sophos NakedSecurity*. http://nakedsecurity.sophos.com/2013/02/08/adobe-patches-flash-heads-off-attacks-on-windows-and-apple.

[32] Elrom, E. (2010, July). Top security threats to Flash/Flex applications and how to avoid them. http://www.slideshare.net/eladnyc/top-security-threats-to-flashflex-applications-and-how-to-avoid-them-4873308.

[33] F-Secure (2013). Backdoor:W32/PoisonIvy. http://www.f-secure.com/v-descs/backdoor_w32_poisonivy.shtml.

[34] Fara, M. and J. Stackhouse (2013, August). Cross-platform malicious code discovered in "in the wild". http://forum.bitdefender.com/lofiversion/index.php/t47561.html.

[35] Ford, S., M. Cova, C. Kruegel, and G. Vigna (2009). Analyzing and detecting malicious flash advertisements. In *Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC)*, pp. 363–372.

[36] Fukami (2007). Testing and exploiting Flash applications.

[37] fukami and B. Fuhrmannek (2008). SWF and the malware tragedy. In *In Proc. OWASP Application Security Conference*.

[38] Gavrun, A. (2012). Adobe Flash Player TrueType font parsing integer overflow vulnerability. http://www.verisigninc.com/en_US/products-and-services/network-intelligence-availability/idefense/public-vulnerability-reports/articles/index.xhtml?id=1001.

[39] Guya (2008). Encapsulating CSRF attacks inside massively distributed Flash movies—real world example. http://blog.guya.net/2008/09/14/encapsulating-csrf-attacks-inside-massively-distributed-flash-movies-real-world-example.

[40] Hay, R. (2009). Exploitation of CVE-2009-1869. http://roeehay.blogspot.com/2009/08/exploitation-of-cve-2009-1869.html.

[41] Heiderich, M., T. Frosch, M. Jensen, and T. Holz (2011). Crouching tiger – hidden payload: Security risks of scalable vectors graphics. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, pp. 239–250.

[42] Hossain, M. (2013). *CORS in Action*. Manning Publications.

[43] Howard, F. (2012). Exploring the blackhole exploit kit. Technical report, Sophos Labs, UK. http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/.

[44] Huang, L.-S., A. Moshchuk, H. J. Wang, S. Schechter, and C. Jackson (2012). Clickjacking: Attacks and defenses. In *Proceedings of the USENIX Conference on Security Symposium*, pp. 22–22.

[45] Huang, W. (2011, April). Newest Adobe Flash 0-day used in new drive-by-download variation: Drive-by-cache, targets human rights website. *Armorize Technologies Blog*. http://blog.armorize.com/2011/04/newest-adobe-flash-0-day-used-in-new.html.

[46] Inc., Y. (2013). Yahoo! user interface library. http://yuilibrary.com.

[47] Invernizzi, L. and P. M. Comparetti (2012). EvilSeed: A guided approach to finding malicious web pages. In *Proceedings of the 33rd IEEE Symposium on Security & Privacy (S&P)*, pp. 428–442.

[48] Irinco, B. (2012). Luckycat leads to attacks against several industries. In *Threat Encyclopedia*. Trend Micro. http://about-threats.trendmicro.com/us/webattack/111/Luckycat+Leads+to+Attacks+Against+Several+Industries.

[49] Jagdale, P. (2009). Blinded by Flash: Widespread security risks Flash developers don't see. Presented at Black Hat D.C.

[50] Jang, D., A. Venkataraman, G. M. Sawka, and H. Shacham (2011). Analyzing the cross-domain policies of flash applications. In *Proceedings of the Web 2.0 Security & Privacy Workshop (W2SP)*.

[51] Johns, M. and S. Lekies (2011). Biting the hand that serves you: A closer look at client-side Flash proxies for cross-domain requests. In *Proceedings of the International Conference On Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pp. 85–103.

[52] Johns, M., S. Lekies, and B. Stock (2013). Eradicating DNS rebinding with the extended same-origin policy. In *Proceedings of the 22nd USENIX Security Symposium*.

[53] Keizer, G. (2011, April). RSA hackers exploited Flash zero-day bug. *ComputerWorld*. http://www.computerworld.com/s/article/9215444/RSA_hackers_exploited_Flash_zero_day_bug.

[54] Kimberly (2011, July). DDoS attacks – a new twist in malvertisements. *StopMalvertising*. http://stopmalvertising.com/malvertisements/ddos-attacks-a-new-twist-in-malvertisements.html.

[55] Kindi Software (2013). secureSWF. http://www.kindi.com.

[56] Kindlund, D. (2012, December). CFR watering hole attack details. *FireEye Blog*. http://www.fireeye.com/blog/technical/malware-research/2012/12/council-foreign-relations-water-hole-attack-details.html.

[57] Kindlund, D. (2013, February). Holiday watering hole attack proves difficult to detect and defend against. *ISSA Journal*, 10–12. http://c.ymcdn.com/sites/www.issa.org/resource/resmgr/journalpdfs/feature0213.pdf.

[58] Kogan, I. (2005). Flare. http://www.nowrap.de/flare.html.

[59] Kogan, I. (2007). Flasm. http://www.nowrap.de/flasm.html.

[60] Kolbitsch, C., B. Livshits, B. Zorn, and C. Seifert (2012). ROZZLE: De-cloaking internet malware. In *Proceedings of the 33rd IEEE Symposium on Security & Privacy (S&P)*, pp. 443–457.

[61] Kovac, P. (2011a, September). Breaking through Flash obfuscation. *Avast! Blog*.

[62] Kovac, P. (2011b, June). Flash malware that could fit a twitter message. *Avast! Blog*.

[63] Lance, B. (2009, November). Connecting JavaScript and Flash. Presented at Flash Camp Philadelphia. http://www.slideshare.net/BeautifulInterfaces/connecting-flash-and-javascript-using-externalinterface-2452543.

[64] Levchenko, K., A. Pitsillidis, N. Chachra, B. Enright, T. Halvorson, C. Kanich, C. Kreibich, H. Liu, D. McCoy, N. Weaver, V. Paxson, G. M. Voelker, and S. Savage (2011). Click trajectories: End-to-end analysis of the spam value chain. In *Proceedings of the 32nd IEEE Symposium on Security & Privacy (S&P)*, pp. 431–446.

[65] Li, Z. and X. F. Wang (2010). FIRM: Capability-based inline mediation of Flash behaviors. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, pp. 181–190.

[66] Magazinius, J., B. K. Rios, and A. Sabelfeld (2013). Polyglots: Crossing origins by crossing formats. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, pp. 753–764.

[67] Mayer, J. R. and J. C. Mitchell (2012). Third-party web tracking: Policy and technology. In *Proceedings of the 33rd IEEE Symposium on Security & Privacy (S&P)*, pp. 413–427.

[68] Mikko (2011). How we found the file that was used to hack RSA. http://www.f-secure.com/weblog/archives/00002226.html.

[69] Mills, E. (2011, April). Attack on RSA used zero-day Flash exploit in Excel. *CNET News*. http://news.cnet.com/8301-27080_3-20051071-245.html.

[70] Ministry of Justice, United Kingdom (2013). Human rights. http://www.justice.gov.uk/human-rights.

[71] MITRE Corporation (2013a, June). CAPEC-178: Cross-site flashing. http://capec.mitre.org/data/definitions/178.html.

[72] MITRE Corporation (2013b). Common vulnerabilities and exposures. http://cve.mitre.org/.

[73] Moxiecode (2013). Plupload v2.0.0. http://www.plupload.com.

[74] Nambiar, S. N. (2009). Flash phishing. *Symantec Official Blog*. http://www.symantec.com/connect/blogs/flash-phishing.

[75] National Institute of Standards and Technology (NIST) (2013). CWE — common weakness enumeration. http://nvd.nist.gov/cwe.cfm.

[76] Nikiforakis, N., A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna (2013). Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the 34th IEEE Symposium on Security & Privacy (S&P)*, pp. 541–555.

[77] Oliveria, P. (2008). Targeted attack in Mexico, part 2: Yet another drive-by pharming. *TrendLabs Security Intelligence Blog*. http://blog.trendmicro.com/trendlabs-security-intelligence/targeted-attack-in-mexico-part-2-yet-another-drive-by-pharming.

[78] Overveldt, T. V., C. Kruegel, and G. Vigna (2012). FlashDetect: ActionScript3 malware detection. In *Proceedings of the 15th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pp. 274–293.

[79] OWASP (2013). Testing for cross site flashing (OWASP-DV-004). https://www.owasp.org/index.php/Testing_for_Cross_site_flashing_(OWASP-DV-004).

[80] Paola, S. D. (2007). Testing Flash applications. Presented at the 6th OWASP AppSec USA Conference.

[81] Parkour, M. (2012, May). May 3 – CVE-2012-0779 World Uyghur Congress Invitation.doc. *Contagio Malware Dump*.

[82] Petkov, P. D. (2008, January). Hacking the interwebs. *GNUCitizen Information Security Think Tank*. http://www.gnucitizen.org/blog/hacking-the-interwebs.

[83] Phung, P. H., M. Monshizadeh, M. Sridhar, K. W. Hamlen, and V. Venkatakrishnan (2013). Between worlds: Securing mixed JavaScript/ActionScript multi-party web content. Submitted for publication.

[84] Poole, N. (2012). XSS and CSRF via SWF applets (SWFUpload, Plupload). https://nealpoole.com/blog/2012/05/xss-and-csrf-via-swf-applets-swfupload-plupload.

[85] Rad, M. B. (2013). Flash based XSS in Yahoo Mail. http://miladbr.blogspot.com/2013/06/flash-based-xss-in-yahoo-mail.html.

[86] Rapid7 (2012). Adobe Flash Player object type confusion. http://www.rapid7.com/db/modules/exploit/windows/browser/adobe_flash_rtmp.

[87] Rapid7 (2013). Cross-site Flashing. http://www.rapid7.com/db/vulnerabilities/spider-actionscript-cross-site-flashing.

[88] Roberts, J. and R. Loach (2013). swfupload: JavaScript & Flash upload library. https://code.google.com/p/swfupload.

[89] Romang, E. (2013). Posts tagged CVE-2013-0633; Gong Da/Gondad exploit pack add Flash CVE-2013-0634 support. http://eromang.zataz.com/tag/cve-2013-0633.

[90] Seltzer, L. (2010). New JIT spray penetrates best Windows defenses. *SecurityWatch*. http://securitywatch.pcmag.com/apple/284124-new-jit-spray-penetrates-best-windows-defenses.

[91] Serna, F. J. (2013, July). Flash JIT—spraying info leak gadgets. http://zhodiac.hispahack.com/my-stuff/security/Flash_Jit_InfoLeak_Gadgets.pdf.

[92] Shacham, H. (2007). The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pp. 552–561.

[93] Sophos (2013). Security threat report 2013: New platforms and changing threats.

[94] Striegel, J. (2007, August). DNS rebinding: How an attacker can use your web browser to bypass a firewall. *Makezine*. http://makezine.com/2007/08/01/dns-rebinding-how-an-attacker.

[95] Symantec (2012, May). Targeted attacks using confusion (CVE-2012-0779). http://www.symantec.com/connect/blogs/targeted-attacks-using-confusion-cve-2012-0779.

[96] Symantec Corporation (2013, April). Internet security threat report, volume 18.

[97] Thomas, K., C. Grier, J. Ma, V. Paxson, and D. Song (2011). Design and evaluation of a real-time URL spam filtering service. In *Proceedings of the 32nd IEEE Symposium on Security & Privacy (S&P)*, pp. 447–462.

[98] U.C. Berkeley and IBM (2013). Security bulletin: Tivoli Endpoint Manager for software use (CVE-2013-0452). http://www-01.ibm.com/support/docview.wss?uid=swg21631350.

[99] W3Techs (2013). Usage of Flash for websites. http://w3techs.com/technologies/details/cp-flash/all/all.

[100] Wang, R., S. Chen, and X. Wang (2012). 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 33rd IEEE Symposium on Security & Privacy (S&P)*, pp. 365–379.

[101] Weinberg, Z., E. Y. Chen, P. R. Jayaraman, and C. Jackson (2011). I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Proceedings of the 32nd IEEE Symposium on Security & Privacy (S&P)*, pp. 147–161.

[102] WordPress (2013). WordPress.org. http://wordpress.org.

[103] Zalewski, M. (2009). *browsersec Browser Security Handbook, part 2, Same-origin Policy for Flash*. Google.

[104] Zeller, W. and E. W. Felten (2008). Cross-site request forgeries: Exploitation and prevention. Technical report, Princeton University.

[105] Zeltser, L. (2011a). Malvertising: The mechanics of malicious ads. http://blog.zeltser.com/post/6275464150/malvertising-mechanics-of-malicious-ads.

[106] Zeltser, L. (2011b). Malvertising: The use of malicious ads to install malware. http://www.infosecisland.com/blogview/14371-Malvertising-The-Use-of-Malicious-Ads-to-Install-Malware.html.

VITA

Dhiraj Karamchandani earned his Bachelor's of Engineering in Information Technology from University of Mumbai in May 2012. He started his Master's journey in Computer Science in Fall 2012. After graduating with a MS Thesis in December 2013, he is planning to pursue new avenues in the field of Software Security.