FLASH IN THE DARK: ILLUMINATING THE LANDSCAPE OF ACTIONSCRIPT WEB SECURITY TRENDS AND THREATS

Meera Sridhar[1], Mounica Chirva[1], Benjamin Ferrell[2], Kevin W. Hamlen[2], and Dhiraj Karamchandani[2]

[1]University of North Carolina, Charlotte, USA; [2]The University of Texas, Dallas, USA

Abstract

As one of the foremost scripting languages of the World Wide Web, Adobe's Action-Script 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 six years. Despite the perniciousness and severity of these threats, ActionScript has been significantly less studied in the scholarly security literature than the other major web scripting language—JavaScript.

To fill this void and stimulate future research, this paper presents a systematic study of Flash security threats and trends, including a finer-grained taxonomy of Flash software vulnerability classes, a detailed investigation of over 700 Common Vulnerability and Exposure (CVE) articles reported between 2008–2016, and an examination of the fundamental research challenges that distinguish Flash security from other web technologies. 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.

Keywords: Workplace Common Vulneraqbilities and Enumeration; Adobe Flash; ActionScript; Virtual Machine

1 Introduction

Adobe Flash applets (Shockwave Flash programs) provide web developers a powerful platform for creating rich, dynamic web content, such as web advertisements, online games, streaming media, and interactive webpage animations. This has resulted in a soaring popularity of the technology on the web. Recent studies (Adobe Systems 2016b; W^{3}Techs 2016) conclude that Flash is the technology of choice for over three million developers for creating interactive and animated web environments, and that it is used by 8.3% of *all websites*. More than 20,000 apps in mobile app stores such as Apple App Store and Google Play are created using Flash, and Flash powers 24 of the top 25 Facebook games. In China alone, a revenue of over US$70 million *per month* is generated by the top nine Flash-enabled games.

The popularity of Flash combined with the complexity of its features has made it extremely attractive to attackers. In the first quarter of 2015 alone, 42 new Adobe Flash vulnerabilities were discovered—an increase of 50% from Q4 of 2014—along with a 317% increase in the number of new Flash malware samples (over 200K total) (Cisco 2015; Garnaeva et al. 2015; McAfee Labs 2015; Symantec Corporation 2015). McAfee Labs concludes that Flash is "a favorite of designers and cybercriminals" (McAfee Labs 2015), and Kaspersky Lab identifies at least 12 major Adobe Flash Player vulnerabilities that gained exceptional popularity among cybercriminals in 2015, and are now integrated into many common exploit packs (Garnaeva et al. 2015). The precipitous increase in vulnerability reports and attacks has been attributed in part to rapidly escalating Flash presence in mobile devices, increased security attention through Adobe's ongoing vulnerability disclosure program and Google Project Zero (Uhley 2015), and the July 2015 compromise of Hacking Team which inadvertently disclosed numerous Flash vulnerabilities to the public—including at least two zero-days (Zetter 2015).

Flash-powered attacks have successfully penetrated some of the most security-hardened facilities in the world, such as the famous 2011 penetration of RSA (Hypponen 2011), and the massive Luckycat campaign that targeted a large spectrum of important U.S. industries, including 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 (Trend Micro Forward-Looking Threat Research Team 2012).

One reason Flash security is so challenging is the feature-filled complexity of the ActionScript (AS) bytecode language (Adobe Systems 2016a) in which Flash apps are expressed. Like other ECMAScript languages, AS includes an object model, function calls, class inheritance, packages, namespaces, and direct access to security-relevant system resources (Ado 2007). However, unlike JavaScript (JS), AS adds significant language complexity, including a sophisticated object-oriented gradual typing system

(Siek and Taha 2007), and is disseminated as compiled binary Flash files (`.swf` files) that pack images, sounds, text, and bytecode into 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 challenges.

Another source of security challenges relates to the increasing heterogeneity of web environments, which often mix interoperating Flash and JavaScript code, and which cohabitate script code from mutually distrusting sources in a shared browser environment. Such practices often evade defenses that secure only one web scripting language in isolation without first-class support for cross-language scripting (cf., Phung et al. 2015). Additionally, Flash's deployment as a plug-in VM tends to widen threat windows due to patch lag. Many consumers are apathetic or inattentive in downloading and installing VM patches, and many companies persist with outdated VM versions for compatibility reasons (Cisco 2015; Garnaeva et al. 2015; McAfee Labs 2015). Consequently, effective intrusion detection of Flash-based attacks must consider a large array of *legacy* VM versions and configurations. Some security advisories have even resorted to recommending an abandonment of Flash altogether (Sophos 2013), but this strategy is antithetical to the revenue models of many businesses.

Despite significant causes for concern, attention given by the scientific research community has been disproportionately low compared to the gravity of the Flash security problem. For example, we show that between 2008 and 2016 only 0.95% 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 (30.55%) to Flash-related threats (see §2.2).

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 and future media-heavy web technologies.

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

- We present a detailed taxonomy of Flash software vulnerabilities, and describe how major classes of attacks are executed. Our categorization provides a more fine-grained, informative classification specifically tuned to the *Flash* vulnerability and attack surface, as compared to more general categorizations (e.g., the NVD's CWE classification system (National Institute of Standards and Technology 2016) for scoring general CVEs (Mitre Corporation 2016)).

- For each category, we highlight AS language and Flash architectural features that are particularly challenging to defend. 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.
- 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 2008–2016, 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 often impedes meaningful classification.

As security researchers, we faced many challenges in conducting this survey of the Flash attack space. Flash security knowledge comprises a massive volume of remarkably disorganized information, including hundreds of research publications and thousands of (sometimes inaccurate) news articles, 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 AS language. The difficulty of taming this volume of information was further heightened by the vast array of differing versions of various Flash software components, such as the Player and the AS language, each of which exhibits a multitude of unique features and weaknesses. CVE articles of Flash-relevant attacks tend to be too terse and coarse-grained to glean useful technical details of an attack for research purposes. Our analysis therefore provides researchers, web developers, and security analysts a substantially improved sense of the Flash vulnerability and attack space, in a *consolidated* form, crucial for developing better Flash security practices and defenses. This will fuel security research towards strengthening web security defenses.

The rest of the paper is organized as follows. Section 2 begins with a high-level summary of Flash security trends uncovered by our analysis of the attack space and scholarly research publication histories. Sections 3, 4, and 5 expand upon these observations by presenting a detailed taxonomy of three important Flash vulnerabilities classes: those arising from (1) *web environment heterogeneity*, (2) *language semantics and implementation* issues, and (3) *networking and communications*, respectively. Section 6 discusses recent research on mitigations. Finally, Section 7 summarizes related work and Section 8 concludes.

2 Security Trends Analysis and Overview

2.1 Vulnerability Analysis

To construct the Flash threat taxonomy detailed in §3–5, we researched and studied all 711 Flash-related CVE articles reported from 2008 to February 2016 (cf., Karamchandani 2013). We performed classification in two steps:

1. We first classified CVEs purely by (i) the attack/vulnerability type attribution given in the CVE description in the Mitre database (Mitre Corporation 2016), and (ii) corresponding CWE identification scores whenever available. However, these classifications were often too coarse to be useful, due to the sparsity and generality of the information related by the CVEs.
2. To refine the classifications, we then re-traversed the entire CVE list with an extra layer of classification based on a broader research of external sources related to each CVE, including news articles and research papers. This included manually analyzing (and, when possible, reproducing) sample attack code in order to resolve ambiguous or conflicting reports. "Unknown" (UNK) 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.

Figure 1 presents the total number of Flash-relevant CVEs per year. The sharp increase in CVE disclosures in 2015 is due in part to increased mobile Flash presence, increased scrutiny (e.g., Adobe's vulnerability disclosure program through Google Project Zero), and a data breach by Hacking Team (Uhley 2015; Cisco 2015; Symantec Corporation 2015; Garnaeva et al. 2015; McAfee Labs 2015).

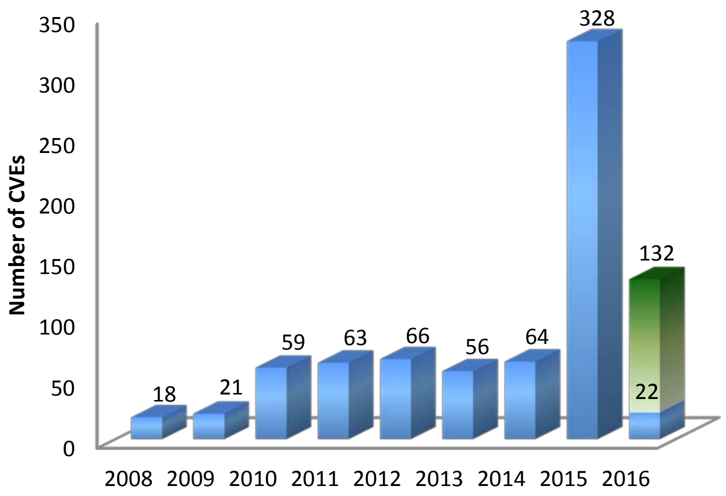

Figure 1: Flash-relevant CVEs from 2008–2016. CVE count for 2016 is projected from January-February numbers.

JISSec
Journal of Information System Security
[authors' draft copy]

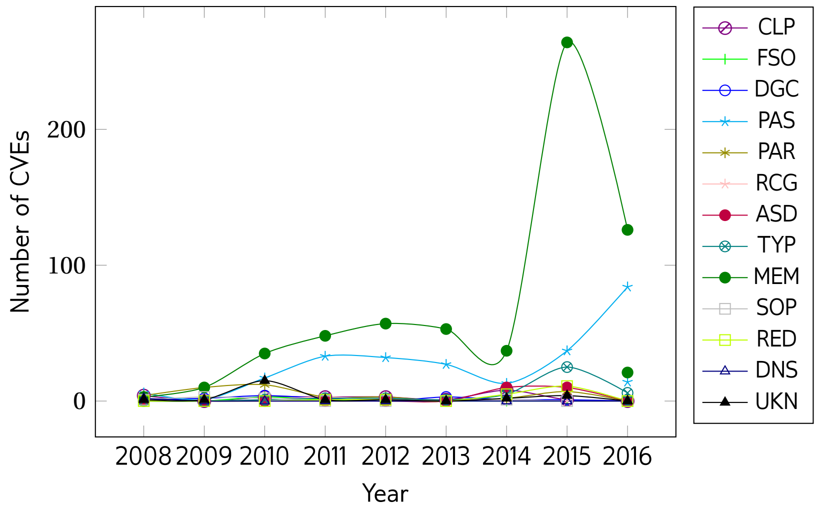

Figure 2: Evolution of Flash-relevant CVE disclosures over 2008–2016.

Environment Heterogeneity		**§3**
CLP	Cross-language Procedure Calls	§3.1
FSO	Flash Shared Objects	§3.3
DGC	Disguised Graphical Content	§3.4
PAS	Parameter Passing	§3.5
Language Implementation		**§4**
PAR	Parsing Inconsistencies	§4.1
RCG	Runtime Code Generation	§4.2
ASD	Address Space Derandomization	§4.3
TYP	Type-Tagging	§4.4
MEM	Memory Errors	§4.5
Networking & Communications		**§5**
SOP	Unique Same-Origin Policy	§5.1
RED	URL Redirection	§5.2
DNS	DNS Rebinding	§5.3
UKN	Unknown	–

Table 1: CVE Classification Legend

Figure 2 tracks the evolution of vulnerability types in the years 2008–2016 according to the taxonomy presented in §3–5. The taxonomy legend is provided in Table 1, with references to the respective sections that detail each category. Figure 3 visualizes this data as a bump chart that exhibits the evolving ranks of the vulnerability classes over time. Memory management errors (e.g., buffer overflow bugs) are consistently the top category, due in part to the massively diverse collection of binary media and object formats that Flash software must dynamically accommodate when streaming, parsing, executing, and rendering Flash apps. Parameter-passing vulnerabilities are a surprising second-ranked category, and are a direct consequence of increasingly aggressive web environment heterogeneity. Type confusion vulnerabilities populate the third rank, and are the fastest escalating vulnerability class we studied. These arise in part from difficulties involving the complexities of AS's gradual, polymorphic type system.

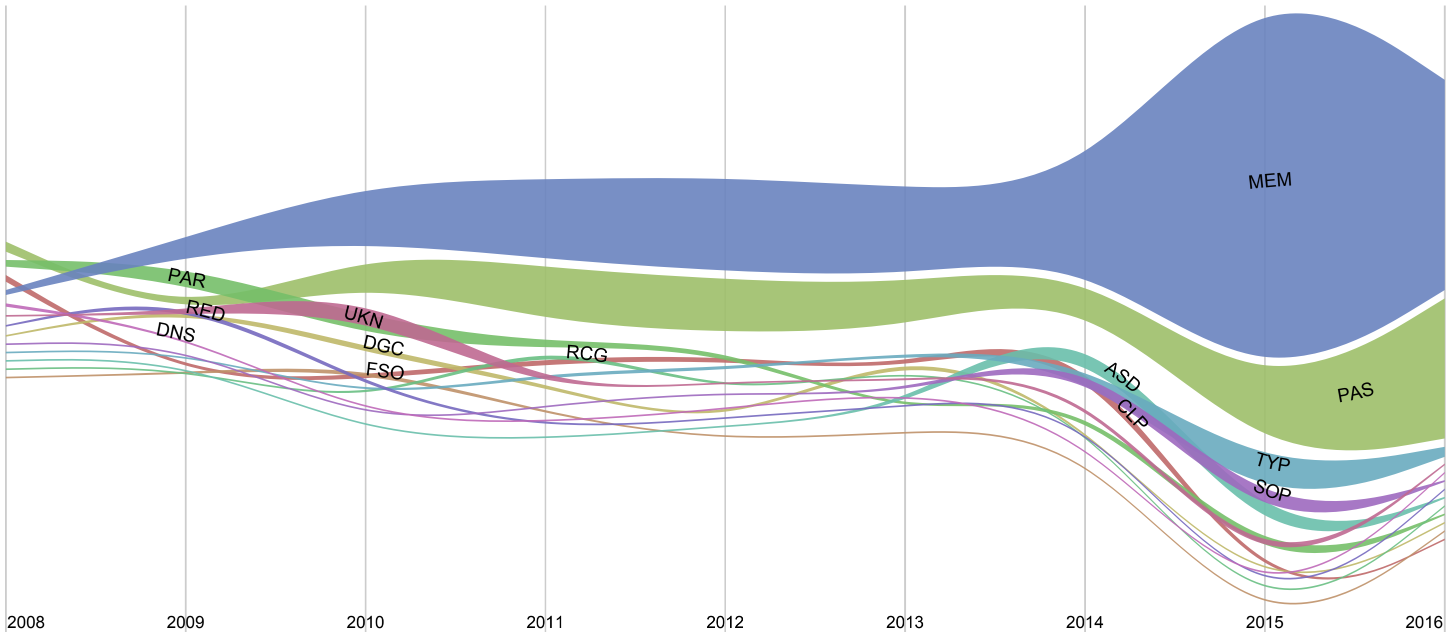

Figure 3: Bump chart illustrating relative rankings of vulnerability classes over time

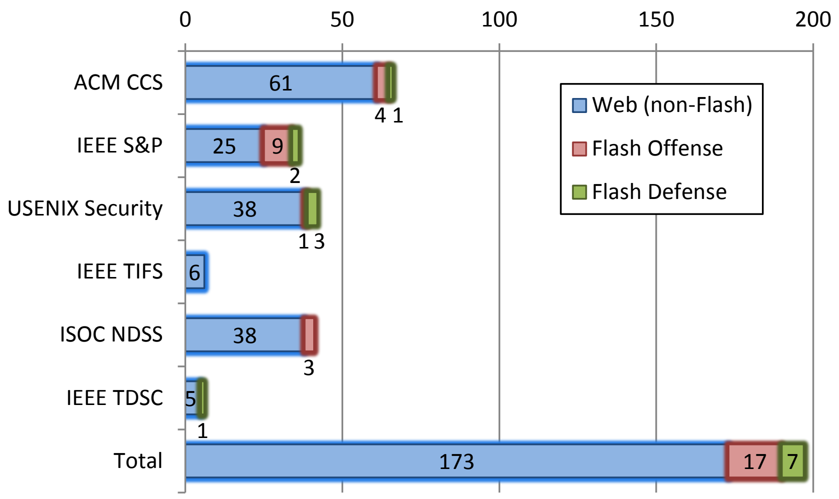

Figure 4: Flash presence in the top six academic, non-cryptography, computer security publication venues in 2008–2016.

2.2 Scholarly Research Survey

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 February 2016 are:

1. *ACM Symposium on Information, Computer and Communications Security (CCS),*
2. *IEEE Symposium on Security and Privacy (S&P),*
3. *USENIX Security Symposium,*
4. *IEEE Transactions on Information Forensics and Security (TIFS),*
5. *Network and Distributed System Security Symposium (NDSS),* and
6. *IEEE Transactions on Dependable and Secure Computing (TDSC).*

We examined all publications in these venues between 2008 and February 2016, manually identifying those papers that are web-related, and conservatively classifying all works that make more than anecdotal reference to Flash or AS as Flash-targeting.

Figures 4 and 5 illustrate the results. Overall, 7.83% (197/2514) of surveyed publications are devoted to web security.

Of these, only 12.18% (24/197) investigate Flash (CCS: Acar et al. 2014, Magazinius et al. 2013, Acar et al. 2013, Heiderich et al. 2011; S&P: Kolbitsch et al. 2012, Nikiforakis et al. 2013, Wang et al. 2012, Invernizzi and Comparetti 2012, Mayer and Mitchell 2012, Weinberg et al. 2011, Levchenko et al. 2011, Thomas et al. 2011, Chen et al. 2010, Bau et al. 2010, Thomas et al. 2015; USENIX: Johns et al. 2013, Huang et al. 2012, Lekies et al. 2015, Nelms et al. 2015; NDSS: Kranch and Bonneau 2015, Pan et al. 2015, Song et al. 2015; and TDSC: Phung et al. 2015). Figure 5 visualizes the data as ring-sectioned pie charts in which each ring has width proportional to the number of publications contributed by a given venue to each category of research. IEEE S&P has the greatest percentage, devoting 11 / 36 = 30% of web security publications to Flash; its ring in is especially wide in the Flash offense category in Figure 5. The remaining five venues collectively devoted only 13 / 161 = 8% of web security publications to Flash. Most Flash-targeting publications we surveyed were purely offensive in nature, revealing and analyzing new attacks; only 7 / 24 = 29% introduce new defensive technologies.

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 cyber attacks. We believe this is due in part to the relative difficulty of apprehending the Flash ecosystem and threat landscape, given the often scattered and disorganized nature of the information. Our research therefore purposes to address this obstacle through a more systematic treatment.

In the following three sections we present the taxonomy of Table 1 in detail, highlighting prominent research challenges, and relating high-profile, in-the-wild exploits that showcase each category's importance.

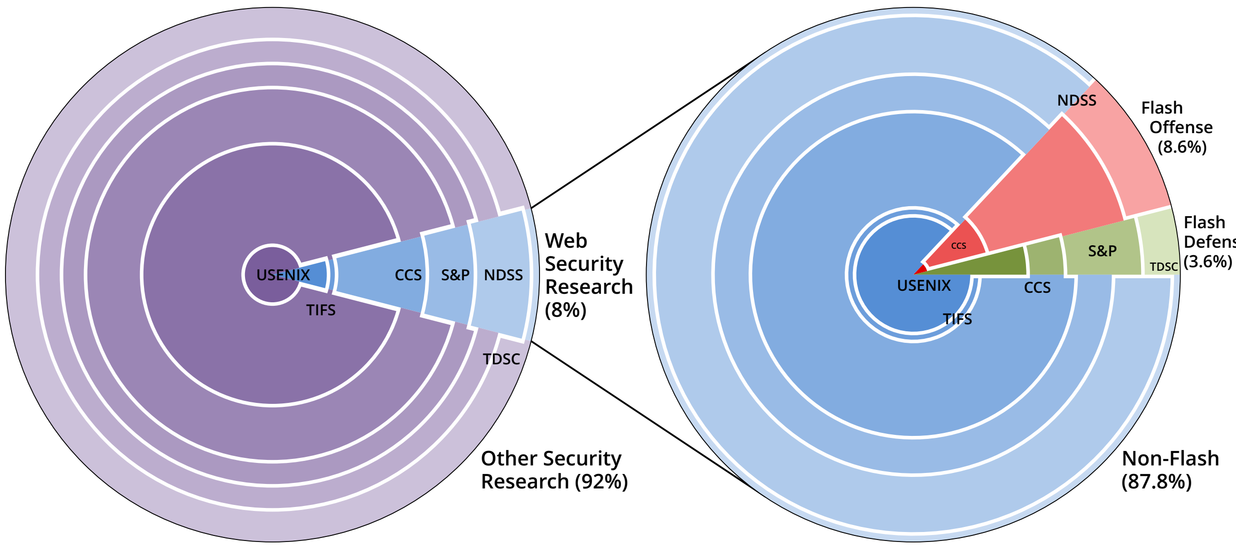

Figure 5: Proportion of publications on Flash defense, Flash offense, and web security research contributed by top scientific venues in 2008–2016.

3 Environment Heterogeneity

Web browser environments are increasingly heterogeneous, mixing interoperating code from a multitude of different scripting languages and origins in a common execution environment. This is important for an industry that prioritizes captivating, dynamically interactive, multimedial user experiences, since different technologies tend to be best suited to facilitating differing forms of interaction and differing media formats. For example, click-tracking for web advertisement revenue generation, interactive control of HTML-embedded Flash movies, movie usage reporting for analytics and billing purposes, and data transport between Flex charts and HTML data tables (Lance 2009), all rely on cross-language web communication.

Flash tends to be a focal point for such heterogeneity, since it supports first-class, dynamic manipulation of many binary web data formats, and is designed to directly access cross-language APIs, such as the Domain Object Model (DOM). Suitable protections and secure practices for such programs can be highly unintuitive because it must span the disparate security models of *both languages*, and exploits that span two languages make it difficult for security tools to identify and prevent them (Cisco 2015). In this section we discuss how such interoperability has led to security vulnerabilities and exploits.

3.1 Cross-language Procedure Calls

Cisco Security Research reports a significant growth in JS/AS cross-language malware starting in 2014 (Cisco 2015). This section describes how insecure Flash cross-language communication practices underly a large class of high-impact web attacks.

From AS2 onward, cross-language communication is achieved in Flash via the `ExternalInterface` class of the runtime system. That interface's `call(`$s, \ldots$`)` method invokes JS function s, and its `addCallback(`s, f`)` method makes AS function f callable from JS under pseudonym s (a fresh JS property name). However, because the computational models of AS and JS differ (AS being a compiled bytecode language, and JS being a dynamically parsed and interpreted string-centered language), this interface between AS and JS has a potentially unintuitive and often misunderstood semantics: Argument s of `call` is passed as a string to the JS VM and *evaluated as JS code at global scope* to obtain a JS function reference. The treatment of s as code and not a verbatim function name (which many developers erroneously expect) is a root of many vulnerabilities.

Abuse of the JS function reference argument as code (similar to JS `eval()`) has been exploited to corrupt the DOM and develop attack back channels, as demonstrated by the Browser Exploitation Framework Project (BeEF) (Alcorn 2011). Many scripts flow unsanitized or insufficiently sanitized, user-controllable string inputs into `ExternalInterface` arguments, resulting in myriad script injection opportunities for attackers. `ExternalInterface` communication is thus a primary facilitator of attacks that use a combination of AS and JS to perform cross-domain code-injection (Howard 2012) (CVE-2011-0611), cross-site scripting (XSS), and cross-site request forgery (CSRF) (Poole 2012).

Flash and JS also have subtly different Same-Origin Policy (SOP) formulations, allowing cross-language scripts to bypass both (see §5.1). Malicious Flash applets exploit this to deliver malicious JS code to third-party victim sites, facilitating click forgery, resource theft, and flooding attacks upon victims (Phung et al. 2015).

Real-world Examples:

SWFUpload XSS Attack: In 2012 and 2013, SWFUpload—a popular upload widget used on many websites—was found to be vulnerable to XSS due to its failure to sanitize user input (parameter `movieName`) that leaks to parameter s of `call()`, where it is interpreted as code (Poole 2012). This exposed many sites where the applet is hosted from the same domain as the embedding page to XSS attacks (CVE-2012-3414, CVE-2013-2205).

Yahoo! Mail XSS Attack: In June 2013, a Flash XSS vulnerability was discovered in the `IO Utility` of the Yahoo! User Interface library (Rad 2013). The utility contained a Flash applet (`io.swf`) which used user inputs as parameters in a `call()` without sanitization, permitting malicious JS execution. The applet was hosted in the Yahoo! Mail main domain, creating a significant privacy vulnerability—Yahoo! Mail users could read other users' inboxes by submitting cleverly crafted URLs to the applet.

3.2 HTML Interactive Capabilities

AS and HTML can also interoperate directly without JS. Flash interprets a subset of HTML tags, including anchors (`<a>`) and images (`<img>`), via its `TextField.htmlText` property, which renders formatted text. Dually, HTML can invoke public and static AS methods expressed as URLs in HTML text fields. In AS2, this is achieved using `asfunction`, which accepts a string identifier for an AS2 function and a string argument passed as input to the function (Paola 2007). In AS3, 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 AS, and adding an AS event handler.

While these features provide powerful convenience in AS-HTML interactions, they can also pose security risks. For example, unsanitized user inputs that flow to `htmlText` can contain `<a>` or `<img>` tags. The HTML `<img>` tag allows the `src` attribute to have a `.swf` extension, resulting in an XSS attack. Input sanitization defenses that merely secure JS are here ineffective, since there is no JS involved. In some cases, even JS plus HTML sanitization does not suffice; for example, by injecting a `.swf` extension (which contains no JS- or HTML-escaped characters) onto a JS `src` attribute, attackers have contrived to replace otherwise safe JS code with a malicious Flash script (Paola 2007; Fukami 2007).

Moreover, because AS apps frequently communicate with many different domains (such as the many different principals that cooperatively purvey web ads), such apps often open cross-domain scripting channels imprudently—for example, by calling the Flash API's `allowDomain()` method with a wildcard that permits universal communication (see §5.2). This escalates AS-HTML attacks by allowing universal cross-domain scripting of the injected payloads.

Example Attacks:

The anchor tag combined with the `asfunction` method are commonly abused by attackers in the following ways:

- direct XSS (e.g., `<a href ='javascript:`⟨*malicious code*⟩`'>`),
- calling AS (e.g., `<a href='asfunction:`⟨*victim callee and arguments*⟩`'>`),
- calling Shockwave Flash (SWF) public functions (e.g., `<a href='asfunction: _root.`⟨*victim object, method, and args*⟩`'>`), or
- calling a native, static, AS API function (e.g., `<a href='asfunction:System.Security.allowDomain,`⟨*malicious host*⟩`'>`).

3.3 Shared Objects

Flash shared objects (a.k.a., "Flash cookies") are similar to HTTP cookies, and are used to store persistent, cross-session data (Chatterji 2008). They are delegated by the `SharedObject` class and are commonly used to enhance UI customizability (e.g., allowing users to personalize website appearance). Applications may only access their own (same domain) `SharedObject` data.

However, unlike traditional HTML cookies, Flash cookies are *cross-browser* and can store large amounts of data up to 100 KB in size. These features have made them attractive vehicles for a number of malicious uses, including phishing, cross-site Flashing (XSF), XSS, and CSRF. For instance, phishing and CSRF often frequently abuse shared objects to compute timestamps and maintain persistent attack status-tracking across sessions and across separate browsers (Ford et al. 2009). Shared objects can also respawn previously removed HTTP cookies (Acar et al. 2014) and are a means for fingerprinting (see §6), resulting in privacy violations.

The 2015 Kaspersky Security Report discusses the increase in the usage of Flash shared objects as a method for concealing exploit packs for attack (Garnaeva et al. 2015; Davydov et al. 2015). For example, the Neutrino Exploit Kit was embedded in

Flash pack and attacked roughly 2,000 users every day, sometimes reaching even 5,000 or 6,000 (Davydov et al. 2015).

Real-world Example:

A Double Free Story (Hayak and Davidi 2014): An Adobe Flash Player zero-day (CVE-2014-0502) facilitated a significant attack that targeted several nonprofits in 2014. The vulnerability is a double-free triggered by a bug in how shared objects are handled. During the termination of a Flash Player instance, all shared objects are destroyed, but not before the object's data is flushed to disk. If an attacker manages to trigger the VM's garbage collector during the flush, shared objects become concurrently double-freed, leading to a remote code execution vulnerability.

3.4 Disguised Graphical Content

One of Flash's most compelling features is its powerful support for web multimedia content. Flash is widely employed to realize web animations, advertisements, games, and rich Internet applications. Unfortunately, attackers find these features useful as well.

Flash has become a compelling platform for maliciously impersonating legitimate sites, luring users (phishing/pharming), and stealing user clicks (click-jacking). For example, Flash-based phishing attacks leverage Flash's advanced animation features to create spoof sites or email messages in which malicious text is rendered purely graphically (e.g., within a compressed animation) not textually.

Such attacks present significant fundamental research challenges for the defense community, because they are largely immune to automated anti-phishing technologies that rely upon textual analysis to identify suspicious phrases and links. Since the Flash content is not text-based, the typical mining algorithms fail to identify the scam. As a result, Flash-based features in phishing sites have proven transparent to many anti-phishing spiders (Nambiar 2009). This highlights the need for new data-mining algorithms that target these binary fusions of multimedia, text, and scripting content.

Along with social engineering attacks that clothe malicious sites in attractive graphical content, certain functions that facilitate the loading of media in Flash applications are regular targets of abuse:

- Flash API method `MovieClip.loadVariables()` reads movie player variables from an external file. Attack payloads trigger it to maliciously change the values of variables in the active SWF file.
- `loadMovie()` and `loadMovieNum()` load a file into a movie clip and allow switching between simultaneously displayed SWF files. `NetStream.play()` plays

media from a local disk, a web server, or Flash Media Server. Attackers probe these for input sanitization lapses to load malicious content.

- `Sound.loadSound()` loads audio content either statically or as a stream, and is also frequently susceptible to input sanitization failures.
- `FScrollPane.loadScrollContent()` displays objects, images, or SWF files loaded locally or from the Internet. It is another frequent hijacking victim for loading malicious script content.

These functions are all regularly exploited for XSS or XSF attacks. For example, `loadVariables()` can perform XSF by tricking the victim into clicking a specially crafted link to send the users' credentials to the attacker. If any of the functions are combined with an insecure same-domain or cross-domain policy (see §5.1), then all image, sound, and movie functions become vulnerable to XSF.

3.5 Parameter Passing

One of the most significant classes of Flash attacks are those that pass corrupted parameters to victim Flash apps from their embedding containers (e.g., the HTML environment). Since the embedding environment can be untrustworthy (e.g., remote content sourced from a third-party domain), Flash apps must conservatively treat such parameters as untrusted. The difficulty of securing them is exacerbated by the many channels via which parameters can be passed, some of which are obscure and may be overlooked by app developers. In a *Flash parameter injection* (FPI) attack, an adversary abuses one or more of these unsecured channels to take control of objects within the Flash applet, and possibly hijack the embedding page's Document Object Model (DOM) API.

Parameter values exported by embedding containers become *global variables* inside the embedded Flash object. Global variables in Flash function similarly to global variables in other languages, but some of them have unique functionalities and effects that can be dangerous if set by an untrusted caller. The three main methods of parameter-passing are:

1. *Embedded URI.* HTML pages embed Flash objects using the `<object>` tag, whose `data` attribute specifies the Flash object's URI. This URI may set global variables within the embedded script. For example,

   ```
   http://host/myMovie.swf?a=5&b=hello
   ```

   sets global variables `a` and `b` to values `5` and `hello`, respectively, within the invoked Flash script.
2. *Direct URI reference.* URIs passed directly to the browsing agent can also include parameters. When this method is used, the Flash file is embedded into a fresh "dummy" HTML page created automatically.

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

Figure 6: Setting Flash global variables via HTML object parameters

3. *FlashVars parameters.* HTML `<object>` elements may also contain `<param>` elements that pass parameters to the embedded object. Figure 6 demonstrates how to use the `FlashVars` parameter to set Flash global variables. Such parameters populate the `_root` level of the Flash script.

The AS language semantics assign default values to uninitialized variables (e.g., zero to integers). Many developers dangerously rely on this behavior, forgetting that the default values of global variables can be effectively determined by untrusted callers via one of the mechanisms above. Attackers frequently exploit such vulnerabilities to hijack or abuse Flash scripts (Paola 2007; Chatterji 2008). Common unsafe practices include:

- The location of a movie is retrieved through an unsafe URL parameter:

  `http://host/index.cqi?movie=movie.swf?globalVar=...`

- A victim is lured to click on a link with malicious Flash parameters:

  `http://host/index.cqi?langEnglish%26globalVar=...`

- A global variable is injected by assigning it via the DOM (Paola 2007):

  `http://host/index.htm#&globalVar=...`

Real-world Example:

Gmail Services XSS FPI: Users of Gmail and Google Apps became vulnerable to full account hijacking in 2010 through a Flash-based XSS vulnerability (Amit 2010). Internally, Gmail used a Flash applet (`uploaderapi2.swf`) for file uploads. Two Flash parameters (`apiInit` and `apiId`) in the applet flowed to an `ExternalInterface` call, where they are interpreted as code.

A proof-of-concept script injection was conducted before Google patched the vulnerability; the attacker was able to execute arbitrary JS code in the `mail.google.com` domain by enticing users to click on links that set `apiInit=eval` and `apiID=`$\langle$*malicious script*$\rangle$. The malicious script ran in the context of active Gmail sessions; attackers were able to fully impersonate their victims and steal information from their accounts.

Alarmingly, this attack is not reliably detectable on the server side. 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 # symbol before the query part of the URL:

```
https://host/uploader-api2.swf#?apiInit=eval&apiId=...
```

The receiving server then sees a parameter-less request. 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 # symbol (Paola 2007; Amit 2010).

4 Flash Language Implementation Issues

Flash seamlessly incorporates an impressive array of binary media formats in aggressively compressed forms for effective streaming. It also boasts a sophisticated hybrid of static and dynamic semantic features—such as gradual typing, runtime code generation, concurrency, and asynchronous memory management—in order to deliver highly flexible, interactive user experiences. This combination of features educes significant implementation challenges, whose security implications are surveyed in this section.

4.1 Parsing Inconsistencies

Flash files are delivered using the *Shockwave Flash (SWF)* file format (Ado 2012), which packs AS bytecode with vector graphics, text, video, and sound for efficient mobility. In contrast to most ECMA scripting languages (e.g., JS), SWF files are designed to be *streamed live*, so that execution can begin on the client before the file is completely downloaded or fully parsed. This requires a sophisticated binary parsing engine, since efficient players must have the capacity to parse, validate, and render scripts lazily on-demand. In addition, the rich multimedia capabilities of Flash demand that it support a dizzying array of binary data formats, including virtually every major video, audio, graphic, and font format, plus binary bytecode scripts. Parsers for all of these formats are packed into one interpreter for quick streaming without the overhead of plug-in loading.

Unsurprisingly, this high complexity has given rise to a host of parsing-related security vulnerabilities. For example, properly validating jump instructions in a lazily parsed binary format is notoriously tricky. The AS language specification is type-safe, confining control-flows to well-formed tagged sections and bytecode indexes (Ado 2012); however, during streaming, some jumps may target destinations or content that has not yet downloaded, and can therefore only be validated once that content appears. AS parsers have a history of getting this wrong, leading to vulnerabilities that allow malicious scripts to jump to non-code destinations (Ford et al. 2009).

The failure of parsers to match the language's specification disinforms static malware detection tools, which typically parse and analyze only the portions of SWF files that the specification says are executable. Many Flash disassemblers and decompilers, such as Flasm (Kogan 2007) and Flare (Kogan 2005), therefore miss malware payloads hidden outside the scriptable portions of the SWF (Ford et al. 2009). Myriad Flash-based malware abuses this loophole to evade detection.

The high flexibility and open-endedness of the SWF format presents related security challenges. SWF files are structured as a series of *tagged blocks* of binary data, to facilitate streaming. *Definition tags* encode content, such as images, text, and sounds, and are stored in a dictionary which *control tags* reference in their flow of execution. Flash VMs quietly ignore invalid tag types during parsing, so that specialized, player-specific blocks can be supplied in a way that is transparent to players that don't support them. However, this means that many players support an array of obscure, often undocumented tags, interpreting them in various unexpected ways. Attackers abuse this to conceal malicious scripts within blocks that are unrecognized by analysis tools, or that attack only certain player versions to evade simulation-based detection strategies (Ford et al. 2009).

Real-world Examples:

APT Campaign Against Defense Employees. In March 2012 an advanced persistent threat campaign was uncovered that exploited a Flash parsing vulnerability (CVE-2012-0754) to achieve remote code injection on victim machines (Constantin 2012). Targets were sent emails with an attachment named "Iran's Oil and Nuclear Situation.doc", suggesting that U.S. defense industries were the intended targets. The attachment consisted of an embedded malicious Flash applet, which downloads and plays a malformed MP4 media file. An MP4 parsing error in the Flash Player corrupted memory, leading to arbitrary code execution, which the attack leveraged to execute a Trojan identified by some antivirus products as "Graftor" or "Yayih.A". At the time of exploit, only 16% of antivirus products detected the Trojan.

Proof-of-Concept Parsing Error Exploit. In 2009 the AS2 parser was found to be vulnerable to an integer overflow bug (CVE-2009-1869) in its treatment of the `intrf_count` field, which counts the number of interfaces implemented by a given AS class. Maliciously crafted SWFs could overflow the count to write arbitrary data to arbitrary addresses in the process address space. A proof-of-concept heap-spray attack on Windows XP SP3 with IE7 is available on Google Code (Hay 2009).

TrueType Font Parsing Bug. A similar integer parsing error for TrueType font resources was discovered in August 2012, which allowed SWFs with maliciously crafted TrueType fonts to achieve arbitrary remote code execution (Verisign 2012). At the time of discovery, targeted attacks were exploiting the vulnerability in the wild via malicious Word documents.

Signed vs. Unsigned Integer Parsing. Drive-by-download attacks have exploited a vulnerability in the `DefineSceneAndFrameLabelData` tag parsing routine in the Flash Player (Dowd 2008). The parser erroneously validated an unsigned 32-bit field with a signed comparison (CVE-2007-0071) (Ford et al. 2009).

4.2 Runtime Code Generation

The binary streaming capabilities of Flash equip authors with runtime code generation and code obfuscation channels that are significantly more difficult to analyze and reverse than in most other scripting languages. While this can be attractive for legitimate intellectual property protection, such as digital rights management, it also facilitates malware obfuscation.

A prominent example is the Flash API's `ByteArray` class, which offers scripts dynamically expandable storage spaces for raw binary data. Legitimate scripts use this facility to download and manipulate binary data streams in arbitrary, custom formats (e.g., encrypted movie streams), and convert them into playable media on-demand. However, malicious scripts use the same functionality to dynamically unpack and execute malicious scripts hidden in the binary data (Overveldt et al. 2012). For example, the `loadBytes()` method allows Flash applets to dynamically reinterpret any binary resource as a fresh binary SWF file and execute it. This gives attackers the potential to create a series of nested, encrypted Flash files embedded in arbitrary data sources. Identifying the embedded exploits by static examination of the external Flash file is extremely challenging (Ford et al. 2009).

Several other SWF tags and AS classes present similar powerful obfuscation-aiding mechanisms, including the `DoAction` and `ShowFrame` tags in AS2, and the `SymbolClass`, `DefineBits`, and `DoABC` tags in AS3 (Ková 2011a,b). Obfuscation techniques that adopt any of these features are deviously powerful because they house arbitrary dynamic code generation capabilities within operations that are widely used for legitimate purposes. While dynamic code analysis has been applied to heuristically detect some of these malicious obfuscations (e.g., Jung et al. 2015; Wressnegger et al. 2015), it cannot reliably detect malware that downloads and launches its malicious content selectively—a common gambit in malvertising.

`ByteArray` objects are also an inviting tool for implementing *heap spraying* attacks (Overveldt et al. 2012). Here, the malicious script populates a `ByteArray` with shellcode containing many entry points (e.g., a long *sled* of no-operation instructions ending in a malicious payload). A separate control-flow hijacking vulnerability is then exploited to redirect control to a random destination address. The destination address is typically not under the control of the attacker, but with a large enough shellcode having many entry points, the hijack targets it with high probability, enabling the attack. *Cross-language heap sprays* (Wolf 2009) implement the spray and the control-flow hijack in different scripting languages (e.g., AS and JS, respectively), so that detection tools must identify and piece together both halves of the attack to spot it.

Real-world Example:

Peeling Obfuscation Like An Onion. Numerous Flash malware families use the attacker-favorite obfuscation technique of wrapping a series of malicious Flash files one into another. One common approach involves wrapping an obfuscated Flash 8 exploit (CVE-2007-0071) into multiple layers of a Flash 9 file (Ford et al. 2009).

4.3 Address Space Derandomization

Many Flash players include a *Just-In-Time* (JIT) compiler that improves performance by converting AS instruction streams to more efficient, processor-specific native code during streaming (Ado 2007). Unfortunately, JIT compilers can be abused to execute *JIT-spraying attacks* (Seltzer 2010; Blazakis 2010a,b), which defeat code control-flow protections such as those based on *Address Space Layout Randomization* (ASLR) and *Data Execution Prevention* (DEP).

OS-implemented ASLR protections reduce the reliability of attacker control-flow hijacks by randomizing the locations of binary code sections in victim processes each time the process loads. This frustrates attackers' ability to predict valid code pointer values, and therefore invalidates many payloads containing such pointers. Similarly, DEP restricts write- and execute-access to most code and data bytes, respectively, impeding malicious code-injections.

However, JIT compilers can open loopholes in both defenses by dynamically allocating writable, executable data sections for JIT-compiled code at discoverable locations. The Flash player implements many optimizations that unfortunately aid JIT spraying attacks, including a large GUI library, a JIT 3D shader language, embeddable PDF support, multiple audio and video embedding and streaming options, and the scripting VM (Blazakis 2010b). JIT spraying was first introduced using Flash in BlackHat D.C. 2010 (CVE-2010-1297) (Seltzer 2010; Blazakis 2010a,b).

Real-world Examples:

Flash-based JIT Spraying Evolution. Starting from the BlackHat D.C. demo by Blazakis (Blazakis 2010a), there has been a back-and-forth war between JIT sprayers and defenders at Adobe (Serna 2013). 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 info leak gadgets* (Shacham 2007) and heap spraying to defeat prior Adobe mitigations. 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 (Serna 2013). Subsequent research has sought to detect Flash-based ROP attacks at the microarchitectural level (Pfaff et al. 2015).

4.4 Type-Tagging

To combine performance-enhancing static typing with flexibility-enhancing dynamic typing, AS is gradually typed (Siek and Taha 2007). This allows developers to statically type some variables, inviting compiler optimization of those operations, while leaving other variables untyped at the source level for convenience. The latter are type-checked at runtime—implemented in the AVM2 as *atoms* and *type-tags*. 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 (Overveldt et al. 2012).

This *type-tag wrapping* has led to type confusion vulnerabilities. Our study identifies type confusion as one of the fastest-evolving threat classes for Flash, due in part to growing awareness of gradual typing technologies and their security implications within the black hat communities. In a *type confusion* attack, the attacker abuses a vulnerability created by a discrepancy in a datum's type representation (Dowd et al. 2009). Type confusion attacks are particularly insidious, since they can bypass DEP and ASLR without any kind of heap or JIT spraying (Overveldt et al. 2012). This kind of vulnerability is often found in software components that bind more than one language (Dowd et al. 2009). For example, in Flash, type confusion vulnerabilities have appeared in the binding layer between AS and native code.

Improper error-checking by compilers while converting between primitive and user-defined types can also cause type confusion vulnerabilities (Dowd et al. 2009). 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 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 callees yielding B's return types creates an exploitable vulnerability usable for various attacks, such as leaking objects' memory addresses, reading arbitrary memory addresses, and hijacking execution (Overveldt et al. 2012).

Real-world Example:

World Uyghur Congress Invitation Attack. In May 2012, attacks in the wild exploited a type confusion vulnerability in several versions of Flash Player to achieve arbitrary code execution (sinn3r and Vazquez 2012). The vulnerability was exploitable by supplying a corrupt response to an AS Message Format0 (AMF0) `error` field. Attackers sent victim members of the World Uyghur Assembly emails with a malicious attachment entitled "World Uyghur Congress Invitation.doc" (Parkour 2012). Members of the U.S. defense community were also targeted.

The document contained references to a malicious Flash file on a remote server, as well as a hidden malicious payload in encrypted form. When downloaded and played using a local vulnerable Flash Player, the Flash file sprays the heap with shellcode and triggers the exploit. The shellcode then finds and unpacks the encrypted malicious payload in the original document, and executes it (Symantec Security Response 2012). The malware contacted servers hosted in China, Korea, and the U.S. to acquire the necessary data to complete the exploitation.

4.5 Memory Errors

Almost all large VM implementations are prone to at least some miscellaneous memory errors (e.g., buffer overflows). However, memory errors have proven particularly problematic for AS VM implementations. In the time frame of 2008 to July 2014, JS memory exploits account for only 76 CVEs, while a staggering 219 CVEs pertain to AS memory exploits (nearly 3 times as many). In one security audit (Naraine 2011), around 400 unique vulnerabilities were discovered and forwarded to Adobe. The CVEs cited by the resulting patch (APSB11-21) are almost all related to memory exceptions. Overall, memory exceptions account for roughly 70% of all Adobe Flash Player CVEs (see §2.1).

As a result, attackers regularly seek out and exploit memory errors in the Flash Player as an unusually fertile source of high-impact zero-days.

One possible reason for the higher prevalence of Flash memory exploits relative to JS is the relatively small number of independent Flash VM development teams relative to JS VM development teams. Many different browser developers independently implement widely deployed JS VM products, creating opportunities for cross-compatibility testing and fuzzing, whereas Flash VM development is dominated by only one developer (Adobe). Multiple production-scale JS VMs are also open source, which widens the pool of software engineers performing security audits relative to closed-source products. Although there are some open-source Flash Player components available (e.g., the largely abandoned (Paul 2010) Tamarin project), the majority of Adobe's Flash Player remains closed-source. Finally, Flash supports an impressive number of multimedia binary formats not directly supported by JS, increasing the opportunities for implementation error.

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 (Hypponen 2011; Keizer 2011; Mills 2011; Clark 2011; Anthony 2011). The attack was allegedly conducted by a nation-state targeting Lockheed-Martin and Northrop-Grumman to steal military secrets (Hypponen 2011). 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 (CVE-2011-0609), 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) (F-Secure 2017) (Trojan backdoor) on the compromised computers, using which the attackers were able to harvest users' credentials to access other RSA network machines, and copy sensitive information and transfer data to their own servers (Keizer 2011; Mills 2011; Clark 2011; Anthony 2011).

It has been speculated that the stolen credentials may have included the unique numbers for the SecurID tokens (Mills 2011). This single breach required RSA to replace the SecurID tokens of their customers worldwide (Hypponen 2011).

5 Networking and Communications

In contrast to attacks that exploit heterogeneity of the local environment (§3), a third important collection of attack categories exploit network heterogeneity—the diverse collection of remote web resources and communication protocols available to Flash apps. Chief among these are non-uniformities and idiosyncrasies in the communications security policies enforced by interoperating web principals.

5.1 Non-uniform Same-Origin Policy

Same-Origin Policy (SOP) is a staple of web security enforcement that disallows most interactions between scripts and resources having different origins (defined by the protocol, host, and port of the originating server). This affords mutually distrusting scripts a form of process isolation in the browser. Although many developers presume that SOP enforcement is uniform across different types of resources, browsers actually enforce SOPs that differ subtly for different resource types, including slightly different policies for DOM access, XMLHttpRequests, cookie accesses, Java permissions, JS permissions, and Flash permissions (Zalewski 2011).

Flash's SOP is particularly complex in order to meet the frequent need for Flash apps to inter-communicate across numerous principals. For example, contextual advertisements typically need read-access to the embedding page in order to mine it for keywords, write-access in order to add links, and network access to both advertiser-owned and advertising network-owned sites (often on separate domains) in order to report analytics and perform click-tracking. As a result, Flash's SOP has some elaborate and obscure features that attackers exploit in order to circumvent SOP restrictions imposed on other resources, such as JS scripts.

For example, unlike JS scripts, AS scripts may voluntarily open cross-domain communication channels by calling the `allowDomain()` method of the `Security` Flash API class with an arbitrary domain name. AS programs may therefore voluntarily *relax the SOP restrictions enforced upon other principals* with respect to themselves. Unfortunately, lax ad developers frequently use this feature imprudently, such as by supplying a wildcard ("`*`") that permits universal access (Elrom 2010). Such misuse is common, since it is a tempting quick-fix for apps whose legitimate communications are being blocked by security errors. When such a script becomes embedded alongside content owned by other principals, its resources can be abused (Jang et al. 2011) or it can become a confused deputy facilitating other attacks, such as CSRF (Phung et al. 2015).

Since SOP-relaxations of this form are effected by runtime Flash script operations (not static declarations), it is infeasible to reliably detect insecure scripts statically. Page publishers and advertising networks that purvey third-party scripts must therefore place a certain degree of trust in the scripts they purvey—trust that oftentimes proves undeserved.

A related idiosyncrasy of Flash SOP is its support for *cross-domain policies* (Zalewski 2011). Remote hosts may voluntarily accept cross-origin communications (waiving SOP restrictions) by serving a `crossdomain.xml` policy file. Such policy files can become very complex, leading to unexpected loopholes that attackers exploit (Kalra 2013). Since the policy file is separate from the code and only requested and consulted at runtime as the cross-domain communication is attempted, insecure Flash SOPs cannot be detected merely by analyzing the code of untrusted scripts. Moreover, an app deemed secure (and possibly code-signed) may later become insecure when the cross-domain policy changes independently of the code.

Real-world Example:

Facebook SSO vulnerability. In 2012, researchers performing penetration tests of

Single-Sign-On (SSO) web services discovered a vulnerability in Facebook exploitable by abusing Flash cross-domain SOP (Wang et al. 2012). The analysis showed that in order to thwart impersonation attacks, Facebook relied upon the browser's SOP to prevent secret authentication tokens from flowing from its Flash sign-on widget to untrusted embedding hosts. By opening a cross-domain communication channel, a malicious embedding page could steal these tokens and acquire private data about victim Facebook users.

Client-side Flash Proxies. Cross-domain HTTP communication is prohibited by JS SOP, but is an oft-requested feature desired by many JS developers.[1] Developers have successfully circumvented and defeated JS's security policy by implementing *Flash proxies* (Johns and Lekies 2011) consisting of a small AS-JS cross-language script that tunnels JS communications through Flash. Flash SOP is significantly more permissive than JS SOP, supporting a `crossdomain.xml` policy that can open arbitrary cross-domain channels to accepting hosts. This practice exemplifies how interoperation of languages with different security policies effectively weakens the security of both to the intersection of the two permission sets.

5.2 URL Redirection

Flash applications extensively use URL redirection (*viz.*, `navigateToURL()` in AS3 and `getURL()` in AS2) and HTTP requests (via `URLRequest`) to direct user clicks to advertiser web sites, or to load external resources. Unfortunately URL redirection can be abused to execute CSRF attacks or obscure links to malicious web pages in a sea of dynamic redirections. The ubiquity of Flash-implemented redirections for legitimate advertising purposes makes malicious uses especially difficult to distinguish.

For example, Flash-based URL redirection has been used to facilitate pharming attacks (Li et al. 2012), which redirect victim users to unintended sites, either by changing the `hosts` file on the victim's computer, or by exploiting a vulnerability in the DNS server software (Petkov 2008). AS scripts commonly accept URLs obtained from external sources, such as `FlashVars` (see §3.5), creating a vulnerability that attacks can easily manipulate to perform cross-site scripting. Examples include the ability to make cookie-bearing cross-domain HTTP GET and POST requests via the browser stack, through the `URLRequest` API. The cross-domain POST can be used in place of GET, which can aid CSRF in the theft of large-sized data (Guya 2008).

Real-world Example:

Reconfiguring Home Router. These two methods were used in conjunction to attack

[1]Cross-Origin Resource Sharing (CORS) (van Kesteren 2014) supports JS cross-domain HTTP using response headers instead of language-level code origins to authorize communications. However, it is not fully supported by all browser environments and versions, motivating Flash Proxies as a fallback workaround.

and reconfigure a well-known home router, *BT Home Hub*, distributed by a leading British telecommunications company (Petkov 2008). The attack uses these AS methods to request Universal Plug and Play (UPnP) functionality via the Simple Object Access Protocol (SOAP) (CVE-2008-1654).

5.3 DNS Rebinding

DNS rebinding attacks circumvent SOP by obfuscating or corrupting domain names in resource requests, misleading defenses into accepting the request as a same-origin communication when it is not. Legacy versions of Adobe's Flash Player (e.g., version 9) have been discovered to be susceptible to DNS rebinding attacks due to subtleties in how they parse domain names (Striegel 2007). For example, the domains "`evil.com`" and "`evil.com.`" (notice the latter's terminating dot) are considered by Flash Player 9 to be the same domain. A SWF file originating from "`evil.com`" can make a request to "`evil.com.`" and hand it off to the browser, which then performs a DNS lookup. This can cause DNS rebinding to occur. In this way, the attacker is able to bypass SOP by dynamically switching the target IP address to an attacker-controlled host name.

Flash's powerful networking capabilities, such as the `Socket` API class's facilities for conducting a full internal network scan, sending email through a corporate SMTP server, or creating a general purpose VPN bridge through a firewall, can be abused through DNS rebinding to create potent networking attacks (Jackson et al. 2009).

6 Selected Mitigations

Most scholarly research on Flash security concerns new attacks; few publications focus on mitigation technologies. Of the 24 top-venue Flash security papers surveyed in §2.2, we found that 7 propose new mitigations. These defensive contributions are summarized in this section in relation to the vulnerability taxonomy outlined in §3–5.

Malvertising Paths. A large category of research on malvertising and drive-by download attacks focuses on the chains of web resources or network paths that culminate in attacks. The URL Redirection attacks discussed in §5.2 are highlighted by such approaches.

Three of the seven Flash-defensive papers covered by our survey propose path-based mitigations. WebWitness (Nelms et al. 2015) significantly reduces drive-by download attacks by monitoring and data-mining user web page visitation histories for suspicious chains. Monarch (Thomas et al. 2011) detects potential spam and phishing scams by monitoring outgoing URL requests and redirections, API calls (e.g., pop-up window creations), and a variety of other static and dynamic features. Further empirical study of these malvertising paths has led to proposals that interventions should target the payment tier of the attack ecosystem, such as through appropriate public policy action (Levchenko et al. 2011).

DNS Rebinding and Same Origin Policy. Johns et al. (2013) demonstrate that variants of the DNS rebinding attacks discussed in §5.3 continue to be feasible in the face of modern defenses, such as DNS pinning. As a more comprehensive defense, they propose lightweight extensions to SOP that would suffice to detect and thwart these threats. In order to be effective, such extensions would need to be uniformly enforced across all web scripting languages—a perennial challenge for the web development community (see §5.1).

Between Worlds. FlashJaX (Phung et al. 2015) monitors web script cross-language procedure calls (§3.1) to enforce security policies related to several classes of our vulnerability taxonomy, including certain parameter passing attacks (§3.5), heap sprays (§4.3), and SOP violations (§5.1). To avoid forcing modifications to web browsers or adoption of new web standards, it takes an *in-lined reference monitoring* approach, which modifies and secures untrusted Flash and JavaScript code in-flight before it is parsed and executed by the browser or VM. However, the policies it can enforce are limited to those expressible in terms of API call traces.

Clickjacking. InContext (Huang et al. 2012) targets a particularly insidious form of graphical subterfuge (§3.4) in which malicious Flash apps impersonate or conceal security settings dialogue boxes and steal user clicks to manipulate them. Flash's exceptional animation and interactivity features can be abused to create almost perfect replicas of browser and VM dialog boxes and widgets, making this a particularly difficult deception for users to discern.

Adobe has attempted to mitigate these attacks by, for example, blocking cross-origin frames that are not fully visible for web cam control settings, but this defense only protects the web cam settings dialog box and not other web content. Recent Flash Players now also impose a delay when UI changes are made, to give the user time to comprehend the current environment and possibly spot a deception, but this may still require exceptional alertness on the part of the user.

InContext therefore implements a more robust solution that computes and compares separate bitmaps of what the user sees versus the content of security-sensitive dialog boxes. If the bitmaps do not match, this indicates a possible deception. This image-comparison approach offers a more general and complete form of graphical deception detection that may be useful for mitigating other Flash graphical obfuscation attacks as well.

Fingerprinting. FPDetective (Acar et al. 2013) combats web privacy violations by crawling web sites and logging activities indicative of *fingerprinting*—a controversial method used by many advertisers and page publishers to track and potentially identify remote web users by collecting distinguishing information about their computers and correlating it with similar information collected by other sites. Flash apps are particularly attractive vehicles for fingerprinting attacks because they can typically access a richer set of system-specific data, such as hardware and OS configuration information, than other web scripting languages.

Fingerprinting falls outside our vulnerability taxonomy because what distinguishes an attack from legitimate application behavior is the use to which the collected information is put, rather than the act of collecting it. For example, a Flash app that consults the graphical capabilities of a client machine in order to select the best movie format and resolution to stream would not typically be considered malicious; but an app that collects the same information in order to determine whether the same machine was recently used to visit sites relating to a particular medical condition would be considered by many to be a privacy violation. Thus, the correct classification hinges upon factors beyond the scope of software vulnerability classification.

7 Related Surveys and Classifications

The most recent Flash security survey paper of similar scope is that by Ford et al. (2009). Published in 2009, it summarizes attacks and vulnerabilities that predate our survey, most of which are now obsolete, including early obfuscation techniques, malvertisements, parsing errors, and decompilation tool issues. The Flash threat landscape and technology maturity has changed considerably in the past seven years, motivating our updated survey of the space.

A more recent 2013 work (Baker et al. 2013) analyzed all security threat reports (not those relating specifically to Flash) as reported by US-CERT, and concluded that Adobe ranked third (at that time) among the top seven software giants in terms of vulnerability disclosures, with 14% of CVEs analyzed, mostly relating to Flash.

In their 2012 publication of the FlashDetect malware detection framework (Overveldt et al. 2012), the authors survey language and architecture features exploited by contemporary Flash-based malware. These include several topics examined in our survey, including script obfuscation, heap spraying, and JIT spraying. Like our study, they conclude that researchers have disproportionately focused on JS security over Flash security, and that more research in the Flash space is needed to adequately address modern web security threats.

Adobe supplies a very brief classification of some Flash vulnerabilities on their web pages (Tenable Network Security 2016; Adobe 2016).

8 Conclusion

Adobe's Flash platform has become a pervasive web technology with a spectrum of rich features. This flexibility and power, however, leads 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. Scholarly research has focused disproportionately on other threats, such as those implemented purely in JS.

In order to fill this void and stimulate future research, we presented a systematic study of Flash security threats and trends, including an in-depth taxonomy of thirteen major components of Flash that can be exercised as attack vectors, and a detailed investigation of 711 Common Vulnerability and Exposure (CVE) articles reported between 2008–2016. The results show that many Flash security threats present unique fundamental research problems not reflected in other scripting languages, including design and implementation challenges for gradually typed, object-oriented, streamed, JIT-compiled programming languages; security policy analysis in the face of aggressive heterogeneity at both the local and distributed environmental levels; and better data mining for phishing and scam detection in media-heavy binary formats. Our systematic organization and detailed summary of these unique challenges, along with representative, real-world attack examples, aids researchers, web developers, and security analysts seeking to address this important threat landscape.

Acknowledgments

This research was supported in part by AFOSR award FA9550-14-1-0173; NSF awards #1054629, #1065216, and #1513704; NSF CRII award #1566321; ONR award N00014-14-1-0030; and the Department of Software and Information Systems at UNC Charlotte. The authors thank Abhinav Mohanty for help with CVE classification. The authors would also like to thank Matthew Pettersson for his help with data collection and classification.

References

G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: Dusting the web for fingerprinters. In *Proc. 20th ACM Conf. Computer and Communications Security (CCS)*, pages 1129–1140, 2013.

G. Acar, C. Eubank, S. Englehardt, M. Juárez, A. Narayanan, and C. Díaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proc. 21st ACM Conf. Computer and Communications Security (CCS)*, pages 674–689, 2014.

Adobe. Adobe security bulletin: Security updates available for Adobe Flash Player. https://helpx.adobe.com/security/products/flash-player/apsb15-32.html, March 2016.

ActionScript Virtual Machine 2 Overview. Adobe Systems, 2007. http://www.adobe.com/content/dam/Adobe/en/devnet/actionscript/articles/avm2overview.pdf.

SWF File Format Specification, Version 19. Adobe Systems, 2012. http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/swf/pdf/swf-file-format-spec.pdf.

Adobe Systems. ActionScript technology center. http://www.adobe.com/devnet/actionscript.html, 2016a.

Adobe Systems. Adobe Flash runtimes statistics. http://www.adobe.com/products/flashruntimes/statistics.edu.html, 2016b.

W. Alcorn. BeEF: The browser exploitation framework project. http://beefproject.com, 2011.

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

S. Anthony. Security firm RSA attacked using Excel-Flash one-two sucker punch. *Huffpost Tech*, April 2011. http://downloadsquad.switched.com/2011/04/06/security-firm-rsa-attacked-using-excel-flash-one-two-sucker-punc.

Y. S. Baker, R. Agrawal, and S. Bhattacharya. Analyzing security threats as reported by the United States Computer Emergency Readiness Team (US-CERT). In *Proc. 11th IEEE Intelligence and Security Informatics Conf. (ISI)*, pages 10–12, 2013.

J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art: Automated black-box web application vulnerability testing. In *Proc. 31st IEEE Sym. Security & Privacy (S&P)*, pages 332–345, 2010.

D. Blazakis. BHDC2010 – JITSpray demo #1. Presented at BlackHat Technical Conf. USA, July 2010a. http://www.youtube.com/watch?v=HJuBpciJ3Ao.

D. Blazakis. Interpreter exploitation. In *Proc. 4th USENIX Conf. Offensive Technologies (WOOT)*, 2010b.

S. Chatterji. Flash security and advanced CSRF. Presented at the OWASP Delhi Chapter Meet, 2008.

S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proc. 31st IEEE Sym. Security & Privacy (S&P)*, pages 191–206, 2010.

Cisco. Cisco annual security report, 2015.

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

L. Constantin. Iranian nuclear program used as lure in Flash-based targeted attacks. *CSO*, March 2012. http://www.csoonline.com/article/701565/iranian-nuclear-program-used-as-lure-in-flash-based-targeted-attacks.

V. Davydov, A. Ivanov, and D. Vinogradov. How exploit packs are concealed in a Flash object. *SecureList*, April 2015. https://securelist.com/analysis/publications/69727/how-exploit-packs-are-concealed-in-a-flash-object.

M. Dowd. Application-specific attacks: Leveraging the ActionScript virtual machine. Technical report, IBM, April 2008. http://www.inf.fu-berlin.de/groups/ag-si/compsec_assign/Dowd2008.pdf.

M. Dowd, R. Smith, and D. Dewey. Attacking interoperability, 2009. http://www.hustlelabs.com/stuff/bh2009_dowd_smith_dewey.pdf.

E. Elrom. 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, July 2010.

F-Secure. Backdoor:W32/PoisonIvy. http://www.f-secure.com/v-descs/backdoor_w32_poisonivy.shtml,F-Secure, 2017.

S. Ford, M. Cova, C. Kruegel, and G. Vigna. Analyzing and detecting malicious Flash advertisements. In *Proc. 25th Annual Computer Security Applications Conf. (ACSAC)*, pages 363–372, 2009.

Fukami. Testing and exploiting flash applications. Presented at Chaos Communication Camp, 2007. http://events.ccc.de/camp/2007/Fahrplan/events/1994.en.html.

M. Garnaeva, J. van der Wiel, D. Makrushin, A. Ivanov, and Y. Namestnikov. Kaspersky security bulletin 2015: Overall statistics for 2015. Technical report, Kaspersky Labs, December 2015. https://securelist.com/analysis/kaspersky-security-bulletin/73038/kaspersky-security-bulletin-2015-overall-statistics-for-2015.

Guya. 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, September 2008.

R. Hay. Exploitation of CVE-2009-1869. http://roeehay.blogspot.com/2009/08/exploitation-of-cve-2009-1869.html, August 2009.

B. Hayak and A. Davidi. Deep analysis of CVE-2014-0502 – a double free story. http://blog.spiderlabs.com/2014/03/deep-analysis-of-cve-2014-0502-a-double-free-story.html, March 2014.

M. Heiderich, T. Frosch, M. Jensen, and T. Holz. Crouching tiger – hidden payload: Security risks of scalable vectors graphics. In *Proc. 18th ACM Conf. Computer and Communications Security (CCS)*, pages 239–250, 2011.

F. Howard. Exploring the blackhole exploit kit. Technical report, Sophos, March 2012. http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit.

L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schechter, and C. Jackson. Clickjacking: Attacks and defenses. In *Proc. 21st USENIX Security Sym.*, pages 413–428, 2012.

M. Hypponen. How we found the file that was used to hack RSA. http://www.f-secure.com/weblog/archives/00002226.html, August 2011.

L. Invernizzi and P. M. Comparetti. EvilSeed: A guided approach to finding malicious web pages. In *Proc. 33rd IEEE Sym. Security & Privacy (S&P)*, pages 428–442, 2012.

C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. Protecting browsers from DNS rebinding attacks. *ACM Trans. Web (TWEB)*, 3(1), 2009.

D. Jang, A. Venkataraman, G. M. Sawka, and H. Shacham. Analyzing the cross-domain policies of Flash applications. In *Proc. 5th Work. Web 2.0 Security and Privacy (W2SP)*, 2011.

M. Johns and S. Lekies. Biting the hand that serves you: A closer look at client-side Flash proxies for cross-domain requests. In *Proc. Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pages 85–103, 2011.

M. Johns, S. Lekies, and B. Stock. Eradicating DNS rebinding with the extended same-origin policy. In *Proc. 22nd USENIX Security Sym.*, pages 621–636, 2013.

W. Jung, S. Kim, and S. Choi. Poster: Deep learning for zero-day Flash malware detection. In *Proc. 36th IEEE Sym. Security & Privacy (S&P)*, 2015. http://www.ieee-security.org/TC/SP2015/posters/paper_34.pdf.

G. S. Kalra. Exploiting insecure crossdomain.xml to bypass same origin policy (ActionScript PoC). http://gursevkalra.blogspot.com/2013/08/bypassing-same-origin-policy-with-flash.html, August 2013.

D. V. Karamchandani. Surveying the landscape of ActionScript security trends and threats. Master's thesis, The University of Texas at Dallas, Richardson, Texas, December 2013.

G. Keizer. RSA hackers exploited Flash zero-day bug. *Computer World*, April 2011. http://www.computerworld.com/s/article/9215444/RSA_hackers_exploited_Flash_zero_day_bug.

I. Kogan. Flare: ActionScript decompiler. http://www.nowrap.de/flare.html, 2005.

I. Kogan. Flasm: Command line assembler/disassembler of ActionScript bytecode. http://www.nowrap.de/flasm.html, 2007.

C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert. ROZZLE: De-cloaking internet malware. In *Proc. 33rd IEEE Sym. Security & Privacy (S&P)*, pages 443–457, 2012.

P. Ková. Breaking through Flash obfuscation. Avast! Blog, September 2011a. https://blog.avast.com/2011/09/09/breaking-through-flash-obfuscation.

P. Ková. Flash malware that could fit a Twitter message. Avast! Blog, June 2011b. http://blog.avast.com/2011/06/28/flash-malware-that-could-fit-a-twitter-message.

M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Proc. 22nd Annual Network & Distributed System Security Sym. (NDSS)*, 2015.

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

S. Lekies, B. Stock, M. Wentzel, and M. Johns. The unexpected dangers of dynamic JavaScript. In *Proc. 24th USENIX Security Sym.*, pages 723–735, 2015.

K. Levchenko, 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. Click trajectories: End-to-end analysis of the spam value chain. In *Proc. 32nd IEEE Sym. Security & Privacy (S&P)*, pages 431–446, 2011.

Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *Proc. 19th ACM Conf. Computer and Communications Security (CCS)*, pages 674–686, 2012.

J. Magazinius, B. K. Rios, and A. Sabelfeld. Polyglots: Crossing origins by crossing formats. In *Proc. 20th ACM Conf. Computer and Communications Security (CCS)*, pages 753–764, 2013.

J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *Proc. 33rd IEEE Sym. Security & Privacy (S&P)*, pages 413–427, 2012.

McAfee Labs. McAfee Labs threats report. Technical report, Intel Security, May 2015. http://www.mcafee.com/in/security-awareness/articles/mcafee-labs-threats-report-may-2015.aspx.

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

Mitre Corporation. Common vulnerabilities and exposures. http://cve.mitre.org, 2016.

S. N. Nambiar. Flash phishing. Symantec Security Blog, January 2009. http://www.symantec.com/connect/blogs/flash-phishing.

R. Naraine. Did Adobe hide 400 vulnerability fixes in latest Flash player patch? *ZDNet*, August 2011. http://www.zdnet.com/blog/security/did-adobe-hide-400-vulnerability-fixes-in-latest-flash-player-patch/9249.

National Institute of Standards and Technology. CWE – common weakness enumeration. http://nvd.nist.gov/cwe.cfm, 2016.

T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad. WebWitness: Investigating, categorizing, and mitigating malware download paths. In *Proc. 24th USENIX Security Sym.*, pages 1025–1040, 2015.

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

T. V. Overveldt, C. Kruegel, and G. Vigna. FlashDetect: ActionScript 3 malware detection. In *Proc. 15th Int. Sym. Recent Advances in Intrusion Detection (RAID)*, pages 274–293, 2012.

X. Pan, Y. Cao, and Y. Chen. I do not know what you visited last summer: Protecting users from third-party web tracking with TrackingFree browser. In *Proc. 22nd Annual Network & Distributed System Security Sym. (NDSS)*, 2015.

S. D. Paola. Testing Flash applications. Presented at the 6th OWASP AppSec Conf., 2007.

M. Parkour. CVE-2012-0779 World Uyghur Congress Invitation.doc. Contagio, May 2012. http://contagiodump.blogspot.com.es/2012/05/may-3-cve-2012-0779-world-uyghur.html.

R. Paul. Mozilla borrows from WebKit to build fast new JS engine. *Ars Technica*, 2010.

P. D. Petkov. Hacking the interwebs. GnuCitizen, January 2008. http://www.gnucitizen.org/blog/hacking-the-interwebs.

D. Pfaff, S. Hack, and C. Hammer. Learning how to prevent return-oriented programming efficiently. In *Proc. 7th Int. Sym. Engineering Secure Software and Systems (ESSoS)*, pages 68–85, 2015.

P. H. Phung, M. Monshizadeh, M. Sridhar, K. W. Hamlen, and V. Venkatakrishnan. Between worlds: Securing mixed JavaScript/ActionScript multi-party web content. *IEEE Trans. Dependable and Secure Computing (TDSC)*, 12(4):443–457, 2015.

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

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

L. Seltzer. New JIT spray penetrates best Windows defenses. *PC Magazine*, February 2010. http://securitywatch.pcmag.com/apple/284124-new-jit-spray-penetrates-best-windows-defenses.

F. J. Serna. Flash JIT – spraying info leak gadgets. http://zhodiac.hispahack.com/my-stuff/security/Flash_Jit_InfoLeak_Gadgets.pdf, July 2013.

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

J. Siek and W. Taha. Gradual typing for objects. In *Proc. 21st European Conf. Object-Oriented Programming (ECOOP)*, pages 2–27, 2007.

sinn3r and J. Vazquez. Adobe Flash player object type confusion. Rapid7, 2012. http://www.rapid7.com/db/modules/exploit/windows/browser/adobe_flash_rtmp.

C. Song, C. Zhang, T. Wang, W. Lee, and D. Melski. Exploiting and protecting dynamic code generation. In *Proc. 22nd Annual Network & Distributed System Security Sym. (NDSS)*, 2015.

Sophos. Security threat report 2013: New platforms and changing threats, 2013.

J. Striegel. DNS rebinding: How an attacker can use your web browser to bypass a firewall. *Make Magazine*, August 2007. http://makezine.com/2007/08/01/dns-rebinding-how-an-attacker.

Symantec Corporation. Internet security threat report (ISTR), volume 20, 2015.

Symantec Security Response. Targeted attacks using confusion (CVE-2012-0779). http://www.symantec.com/connect/blogs/targeted-attacks-using-confusion-cve-2012-0779, January 2012.

Tenable Network Security. Adobe Flash Player <= 19.0.0.245 multiple vulnerabilities (APSB15-32). https://www.tenable.com/plugins/index.php?view=single&id=87244, 2016.

K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time URL spam filtering service. In *Proc. 32nd IEEE Sym. Security & Privacy (S&P)*, pages 447–462, 2011.

K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. McCoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab. Ad injection at scale: Assessing deceptive advertisement modifications. In *Proc. 20th ACM Conf. Computer and Communications Security (CCS)*, pages 151–167, 2015.

Trend Micro Forward-Looking Threat Research Team. Luckycat redux: Inside an APT campaign with multiple targets in India and Japan. Trend Micro Research Paper, March 2012. http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_luckycat_redux.pdf.

P. Uhley. Community collaboration enhances Flash. https://blogs.adobe.com/security/2015/12/community-collaboration-enhances-flash.html, December 2015.

A. van Kesteren. Cross-origin resource sharing. W3C Recommendation, January 2014. http://www.w3.org/TR/cors.

Verisign. 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, August 2012.

W^{3}Techs. Usage of Flash for websites. http://w3techs.com/technologies/details/cp-flash/all/all, 2016.

R. Wang, S. Chen, and X. Wang. Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed single-sign-on web services. In *Proc. 33rd IEEE Sym. Security & Privacy (S&P)*, pages 365–379, 2012.

Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Proc. 32nd IEEE Sym. Security & Privacy (S&P)*, pages 147–161, 2011.

J. Wolf. Heap spraying with ActionScript: Why turning off JavaScript won't help this time. FireEye Malware Intelligence Lab, July 2009. http://blog.fireeye.com/research/2009/07/actionscript_heap_spray.html.

C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck. Analyzing and detecting Flash-based malware using lightweight multi-path exploration. Technical Report IFI-TB-2015-05, Institute of Computer Science, University of Göttingen, December 2015.

M. Zalewski. Same-origin policy. In *Browser Security Handbook, Part 2*. Google, 2011. https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy.

K. Zetter. Hacking team shows the world how not to stockpile exploits. *Wired*, July 2015. http://www.wired.com/2015/07/hacking-team-shows-world-not-stockpile-exploits.