Exploiting Split Browsers for Efficiently Protecting User Data

Angeliki Zavou, Elias Athanasopoulos, Georgios Portokalidis, and Angelos D. Keromytis
Columbia University, New York, NY, USA
{azavou,elathan,porto,angelos}@cs.columbia.edu

ABSTRACT

Offloading complex tasks to a resource-abundant environment like the cloud, can extend the capabilities of resource constrained mobile devices, extend battery life, and improve user experience. Split browsing is a new paradigm that adopts this strategy to improve web browsing on devices like smartphones and tablets. Split browsers offload computation to the cloud by design; they are composed by two parts, one running on the thin client and one in the cloud. Rendering takes place primarily in the latter, while a bitmap or a simplified web page is communicated to the client. Despite its difference with traditional web browsing, split browsing still suffers from the same types of threats, such as cross-site scripting. In this paper, we propose exploiting the design of split browsers to also utilize cloud resources for protecting against various threats efficiently. We begin by systematically studying split browsing architectures, and then proceed to propose two solutions, *in parallel* and *inline* cloning, that exploit the inherent features of this new browsing paradigm to accurately and efficiently protect user data against common web exploits. Our preliminary results suggest that our framework can be efficiently applied to Amazon's Silk, the most widely deployed at the time of writing, split browser.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Client/Server;
D.4.6 [**Security and Protection**]: Information flow controls

Keywords

Split browser architectures, information flow tracking, data protection, cloud, cross-site scripting

1. INTRODUCTION

The proliferation of smartphone devices and ubiquitous Internet connectivity over wireless protocols (WiFi, 3G, 4G, *etc.*) has multiplied the number of Internet users [3], as well as the services offered to them. Web browsers have become the preferred "portal" to access these services, since they allow developers to create a uniform interface that is accessible from different platforms (PCs, smartphones, tablets, *etc.*), and requires minimal (if any) changes to support new platforms. For the same reason, many mobile applications (*e.g.,* the NY Times and Facebook apps) simply encapsulate browsers, acting in essence, as site-specific browsers [24].

Their key role and popularity, their size, and in the case of mobile devices, a growing monoculture,[1] are probably the main reasons browsers are frequently targeted by attackers [5, 11, 17]. In the past, they have suffered severe attacks that exploit vulnerabilities like buffer overflows [17], which enable remote parties to fully control the browser or even the entire operating system after compromise. More frequently though, they enable cross-site scripting (XSS) attacks [26], that are in-fact exploiting vulnerabilities in web applications and not the browser itself. Altogether, the goal of attackers is predominantly obtaining sensitive user information (*e.g.,* credit cards, SSNs, and passwords) that can be used to make profit. Thus, browser security plays a crucial role in the protection of user data.

Concurrently, an increasing number of services migrate to the cloud to reduce costs and gain increased availability and reliability. The ample processing power available in existing cloud infrastructures is also utilized to improve the performance of computationally intensive tasks, or offload certain tasks (even security) entirely to the cloud [9, 22]. The latter is particularly interesting, since it allows us to overcome certain fundamental constraints of mobile devices, such as battery life and limited processing power.

Rendering a web page can be considered as such an intensive task. Split (or else cloud-based) browser architectures [2, 16] split the load of page rendering in two parts, and offload one part to the cloud. Briefly, they consist of two components: the client and the server (cloud) part. To access a web site or application, the client part first communicates with the cloud side, which is responsible for making all required connections and fetching the content required to assemble the final page. Second, the collected components are "rendered" and delivered to the client. Rendering in this context can mean producing a bitmap of the web site, or performing a series of optimizations on the content (*e.g.,* downsizing images) and sending a compressed HTML page with all third-party elements embedded in-line.

The gains from using split browsers in mobile devices are

[1]Webkit powers the browsers on Android, iPhone, and Blackberry smartphones.

CCSW'12, October 19, 2012, Raleigh, North Carolina, USA.

straightforward. Bandwidth can be saved, and precious processing cycles and energy need not be wasted, as a significant part of the computation is moved off the device. Moreover, since the cloud side of the browser acts as a proxy, content caching is made easy. Although the idea is not new, *split browsers* are on the rise. Amazon, one of the largest IT vendors globally, promotes this particular architecture through its Amazon Silk web browser that comes with their KindleFire [4] device.

The split browser paradigm presents us with new opportunities in improving security and protecting user data, as users interact with services hosted in the cloud. In this short paper, we propose a cloud-wide data tracking architecture, which will be transparently offered to all entities running in the cloud. Our goal is to enable various defensive measures that can be deployed on-demand. For example, intra-host data flow tracking can be used to protect the cloud-side of the browser (*i.e.,* the back-end) from memory corruption vulnerabilities [21], while cloud-wide information tracking can be used to detect and prevent information leaks [31], or just provide data accounting services, when interacting with services and applications within the cloud. The latter actually becomes increasingly desirable in split browser architectures, as more information and functionality is transferred to the cloud.

However, simply introducing data tracking support for everything already hosted on the cloud is not sufficient to protect from other types of attacks, like XSS. To extend the abilities of our approach, we propose replicating the client-side of the browser (*i.e.,* the front end) in the cloud. This "cloning" enables us to extend data tracking to the front end, and apply protection mechanisms like XSS prevention [20].

We envision that the underlying data tracking framework can be deployed in various levels using different techniques. For example, by employing virtualization [13,15], or extending languages [30] and the OS [32].

In summary, this paper contributes the following:

- We explore the split browser architecture for web browsing. We investigate security and privacy issues, as well as new features, which are inherited from switching from traditional web browsing to the split browser model.
- We propose a data-tracking framework, specifically designed for split browsers like Amazon's Silk, for countering web exploitation in this new browsing paradigm.

This paper is organized as follows. In Section 2, we present the split browsing paradigm as it has evolved during the last two decades. In Section 3, we propose a security architecture for cloud-based browsers, building on the design of Amazon's Silk. We review related work in Section 4, and conclude in Section 5.

2. SPLIT BROWSERS

Split browsers are web browsers that can offload client-side execution by performing a partial pre-rendering at the server side. The concept was first introduced in 1999 by iCentrix Ltd for the design and implementation of MarioNet [2], a web browser developed for operating in thin clients with low memory and processor capabilities. The core idea behind MarioNet was to perform all rendering at the server and maintain a minimal engine that can uncompress and display graphics at the client. As such, all communications

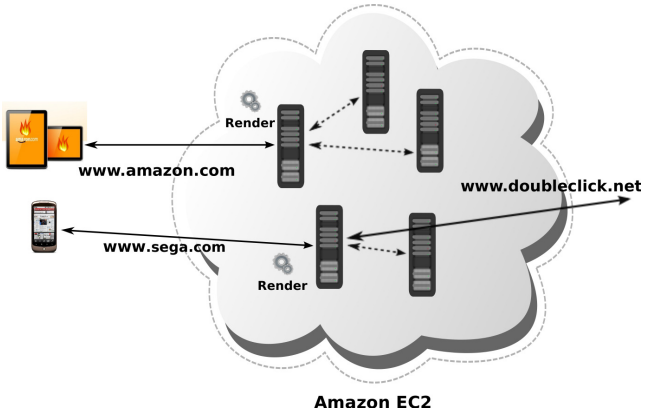

Figure 1: Split browser architecture overview. Notice that all rendering, as well as fetching external resources, for example ads from `www.doubleclick.net`, is performed at the cloud. The device finally receives a compressed HTML document with all external resources embedded in-line, which has been composed at Amazon's servers.

for browsing a web page were performed in a single round trip, using a proprietary protocol called OPTIC, which could deliver a pre-rendered bitmap image. Subsequently, all UI events, such as mouse clicks and keyboard input, would be relayed to the server; this is why there is a loose connection in spelling MarioNet and *marionette*.

The design of MarioNet was not followed by immediate success. It took more than six years for another vendor to follow the paradigm of MarioNet. Opera, one of the largest browser vendors, published Opera Mini in 2005. The target of Opera Mini is mobile devices and smartphones with constrained capabilities, such as tiny screens (even less than 128 pixels wide). The browser by default operates in Small-Screen Rendering (SSR) mode. That means, that all rendering is performed at servers owned by Opera and a final bitmap is delivered to the client-side part of Opera mini. JavaScript support is limited. The server runs all scripts for a couple of seconds, and then, initiates rendering. This means that all scripts that exceed a time threshold are not going to be operational in the final web page. Asynchronous calls, such as `onSubmit` and `OnClick` for example, are also relayed to the Opera server for execution, and the result web page is transmitted back to the client. This essentially resembles the operation of MarioNet.

Apart from MarioNet and Opera Mini, the most recently announced browser that supports this split architecture was developed by Amazon for the KindleFire platform. Silk has been the largest success story of this browser paradigm. Below, we discuss its architecture in more detail.

2.1 Amazon Silk

Silk is a browser designed by Amazon for use with the KindleFire tablet. This split browser utilizes Amazon's Elastic Cloud computing base [28], also known as EC2, for performing part of the rendering in the cloud. Amazon calls this process web acceleration. Silk has many components that can be accelerated, like networking, HTML rendering, external scripts, and so forth. The set of components that will be eventually accelerated is dynamically determined at

run time. An example operation of Silk is the following. If the browser is in acceleration mode, instead of sending an initial HTTP request toward the target web server, it sends it to Amazon's infrastructure and the back end part of Silk in particular. Amazon then performs the HTTP request on behalf of the user. The initial HTTP request frequently triggers subsequent HTTP requests, for fetching images, scripts, and advertisements. All requests, including the ones that target third-party servers, are performed by the cloud, and the corresponding responses are collected. Finally, a web page containing all external resources embedded in-line is assembled, compressed, and returned to the user.

The rendering part is loosely based on WebKit [27], while Google's SPDY protocol [1] is employed between client and back end for speeding up network operations. Although, it has been estimated that some pages load slower with Silk than with a traditional browsing [16], the savings in terms of battery consumption for the clients can be significant. Bandwidth savings are also important in the case of bandwidth oriented data plans because the final page being transmitted is optimized for the client device (*e.g.,* images are downsized to fit its screen), compressed, and the whole communication involves just a single party (*i.e.,* the Amazon server) and requires only one round-trip. Although, at the time of writing, KindleFire does not support 3G, it is very likely that in the future we will see more mobile devices featuring Silk or Silk-like browsers. Finally, in certain cases, caching all frequently accessed information in the cloud can reduce latency even further.

Figure 1 depicts a split browser architecture, and an example of operation of Silk while browsing `www.sega.com`. Notice that all rendering, as well as fetching external resources, for example advertisements from `www.doubleclick.net`, is performed at the cloud. The device finally receives a compressed HTML document with all external resources embedded in-line, which has been composed at Amazon's servers.

Many privacy concerns have been raised because all communication is carried out by Amazon on behalf of the user. The Electronic Frontier Foundation (EFF) has discussed all these issues with Amazon [6]. The company has publicly responded that web acceleration is an opt-in feature and that encrypted traffic (*i.e.,* HTTPS), which is usually associated with more sensitive resources, is never routed through the cloud. In this paper, we do not address such privacy concerns, but we mainly focus in investigating efficient security mechanisms for architectures like Silk. *However, the approach presented here, and assuming that the provider (e.g., Amazon) behaves honorably, can be also used to protect users when encryption is employed.*

We believe that split browser architectures will receive major attention and adoption in the following years, mainly for two reasons. First, cloud computing has became mainstream with large infrastructures offering great computational capabilities. Second, even though thin clients evolve in computational capabilities, some fundamental constraints, like battery consumption, remain. Cloud-based browsers can reduce processing on devices, which translates to increased battery life. Moreover, the fact that rendering takes place in the cloud gives us the opportunity to take advantage of the intrinsic characteristics of the cloud architecture to efficiently apply protection mechanisms.

2.2 Site-specific Split Browsers

Nowadays, we are experiencing the trend of custom single-site applications, which encapsulate browser features for accessing popular services otherwise available over the web. These custom and site-specific browsers are popular in mobile platforms, such as Android and iOS. Examples of this trend are Google services such as Google Search and Maps, social services such as Facebook, Twitter, and Foursquare, various popular applications that evolved in the mobile platform such as Instagram, and popular web sites such as TripAdvisor, NYTimes, and so on. Technically, these applications embed a browser component, usually based on the WebKit engine, for rendering web content. However, they extend the functionality of the browser in numerous ways to provide a more desktop-like user interface. Since large companies invest in the development of these custom browsers, we expect that in the near future they will be enhanced further. Notice, that many of these companies earn more revenue from mobile than desktop users, and they already operate in the cloud. For example, Facebook is considered as the biggest host of photographic content. We consider it to be highly likely that split browser architectures will be adopted by many other IT vendors, apart from Amazon.

3. APPROACH

Split browsing is not exempt from common security risks suffered by the traditional web browsing model, *e.g.,* XSS attacks and code injection flaws. Solutions for all these web exploits have been explored, and some of them are already integrated in traditional web applications. Currently, most of these approaches attempt to prevent web exploits on the server side [29], by inspecting and employing sanitization checks on the data exchanged between the web server and the browser. However, server side mitigation is not adequate for efficient protection against all categories of web exploitation, and therefore protection mechanisms need to be also applied on the client-side (*i.e.,* the web browser) as well. Unfortunately, client-side security solutions tend to be resource intensive, precluding the use of efficient but heavyweight protection mechanisms for XSS and data leak prevention, such as dynamic data flow tracking (DFT) techniques [21, 29]. The problem is only exacerbated when cast to the thin client model, where resources (*e.g.,* battery and memory) are limited.

In this paper, we uncover an opportunity for applying otherwise costly protection mechanisms using cloud resources, and propose a framework for protecting user data when using this new browsing paradigm. The advantage of split-browser architectures, over traditional ones, lies in the existence of a pre-rendering phase, which takes place in the cloud. We can exploit this phase to apply particular security mechanisms [30, 31] in a more efficient way than in the traditional browsing model. More specifically, we *propose a mechanism that capitalizes on the architectural model of split browsers, as well as DFT, in order to protect user data from common web exploitation techniques.*

3.1 Architecture Overview

The design of our architecture for protecting split browsers was driven mainly by two requirements: first the accurate detection and prevention of a multitude of attacks, and second performance. To achieve the first, we turn to a well established technique that has been used by many protection

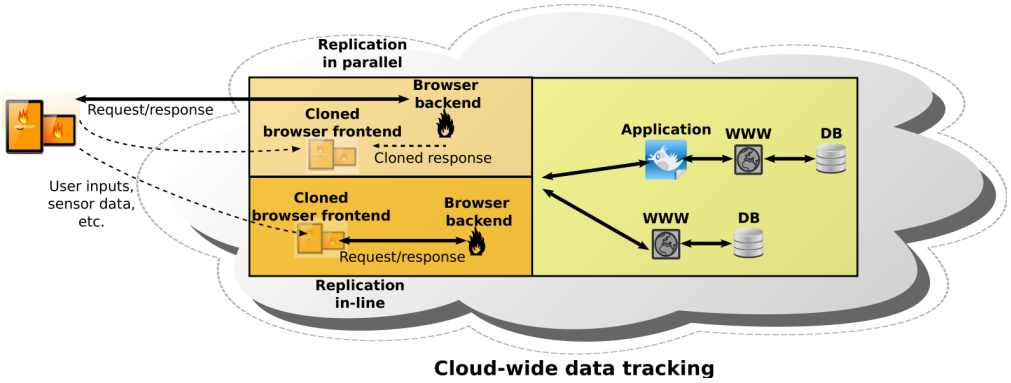

Figure 2: Architecture overview. The two possible designs, in-parallel and inline, of the clone of the client-side (front end) split browser part. In the in-parallel replication, the pre-rendered responses are forwarded from the browser back end in parallel to both the client-side front end and to its clone residing on the cloud, that process them independently. In the inline design, all responses pass through the clone before reaching the client-based front end, which happens only in the absence of exploits.

solutions. Data flow tracking (DFT) has powered many security mechanisms in the past, and can offer protection from the most common web exploits (*i.e.,* SQL injection, code injection and XSS exploitation) with high accuracy. For the second, we propose exploiting two facts: (a) most of the processing in split-browser architectures occurs in the cloud, and (b) the cloud has ample computing resources available, to apply DFT-based defenses with low-overhead.

A high level overview of the proposed architecture is illustrated in Figure 2. We envision that all communicating services residing on the cloud (*i.e.,* the cloud-based component of the split browser, and the web and servers, *etc.*) will incorporate taint tracking mechanisms [10, 30, 31] in their software stack, if such mechanisms are not already a security service offered by default by the cloud provider. At a high level, our security model performs an in-depth and precise analysis of how *unsafe* user inputs, that are usually the main source of most web exploits, propagate inside the split browser component residing on the cloud. *Inputs originating from services residing outside the cloud can be considered tainted in their entirety, if desired. A fact that we believe will encourage the migration of additional services to the cloud.*

3.2 Front-end Cloning

Although some common web exploits can be detected and prevented by the *enhanced* cloud-based component of split browsers, unfortunately the client-side part of the browser still remains vulnerable to certain web exploits that could be detected only on the client-side. In order to protect from this subset of exploits, and taking into consideration the limited resources on the client side in terms of computation and energy reserves, we propose running a *synchronized clone* of the client-side component of the split browser in the cloud.

The idea of replicating the image of a thin client in the cloud, as well as relevant synchronization issues, has been discussed broadly in previous works that explored ways to extend the capabilities of smartphones by offloading execution to the cloud [9, 22], and thus we omit technical details for brevity. At a high level, an execution trace with all necessary information for the accurate replay of user actions is recorded on the client and then transmitted for replaying on the *enhanced clone* running on an emulator in the cloud. As the cloud does not have the resource constraints (*i.e.,* primarily battery constraints) of the client, we can perform security checks that would otherwise be too expensive to run on the client itself.

We envision data tracking techniques being only one of the defenses enabled by our clone architecture for discovering intrusions that could otherwise only be detected at the web browser. Briefly, our approach will flag data from user input as *tainted* and will track this data throughout the execution of the system, so that all data depending on them will also be flagged and tracked. More precisely, our design suggests that the client's clone runs over a cloud where all communicating services are DFT-enabled and exchange marked data.

Although defense mechanisms for web browsers using data tracking techniques in the cloud have been proposed in previous works, they have failed to prove that they are practical enough for deployment on production systems. The major strengths of our approach over previous protection mechanisms are summarized in the following:

1. The client is not burdened with heavyweight DFT, it is the resource-rich, cloud-based clone that will carry this burden.
2. Traffic to the client is not stressed with extra bytes for carrying the taint labels, since these only need to be transferred to the clone in the cloud.
3. Finally, in our approach DFT is *computationally affordable*, since it is only applied on components running on the cloud, and therefore can be significantly accelerated by the cloud infrastructure.

We plan to explore two different designs of the cloud-based synchronized clone, *in parallel* and *inline* in respect to the main execution path. We will evaluate the two models in terms of response latency, application functionality, and resource usage.

In parallel clone. In this design, the clone will be running in parallel with the original split browser part residing on the client machine. The clone receives all the traffic that the original component receives and serves mostly as a complementary debugging tool without interfering in the communications between the original two parts of the split browser. The clone is instrumented with taint tracking tech-

niques and in case of a security violation detection, *i.e.,* data leak, memory corruption *etc.*, an alarm is raised to inform the original components of the browser of the event. A notification system receives the alarm and decides on further actions, *i.e.,* logging, notifying the client-side of the violation, preventing the data transfer, or terminating the connection with an error. We omit further details on the notification system and the possible actions in case of a violation detection due to space constraints.

Inline clone. Unlike the previous approach, this design is based on placing the clone interspersed between the two components of the split browser. Responses from the cloud-based component are directed through the cloud-based clone before reaching the client. If no violation is detected, results are forwarded to the client as is. As expected, this model evaluates the HTTP responses in advance, and therefore protects against potential exploits in real-time. Unfortunately, it introduces further latency in the communication between the split browser and the cloud, especially if the content is highly dynamic and cannot be cached. It is unclear, if this added latency will significantly affect end-user experience. We plan to perform cost analysis to evaluate and quantify the scale of the latency using real-world payloads, and taking into account standard acceleration techniques, such as caching.

To our knowledge, our approach is the first one to take advantage of split browser architectures for protecting user data from common types of attacks launched against web applications. We also believe that it will find a lot of supporters in the future, since it does not only offer a practical solution, but it is also consistent with the current trend of moving execution loads, including security-related functions, to the cloud.

4. RELATED WORK

Decoupling resource-intensive execution, such as heavyweight security mechanisms, from resource-constrained thin clients and offloading their execution on the cloud has been explored in the past in different contexts, especially in the field of resource-poor smartphones [8,9,22]. The main goal of CloneCloud [9] is the acceleration of CPU intensive and low interaction applications, nonetheless its authors recognize the potential use of the approach in offloading heavyweight security mechanisms from phones. On the other hand, in Paranoid Android [22] the authors propose a framework for collecting the state of a smartphone, which is periodically sent to the cloud, where various security checks are applied.

Research literature is rich in papers employing taint-tracking mechanisms for detecting and defending against web exploits [7, 10, 21, 23, 29, 33]. All these frameworks employ DFT mechanisms on the server side of web applications, and their scope of protection ranges from classical buffer overflow and format-string vulnerabilities to the detection of XSS and other types of injection attacks. Some of them are problem-specific and require modifications for protecting web applications against a wider set of web exploits. For instance, DBTaint [10] applies information flow tracking on databases.

Lately, server-side defense mechanisms are shifted to the cloud. To date, most research on the problem of data leakage through cloud-based web services reflects continuations of established lines of security research, such as web security, and data outsourcing and assurance, rather than exclusively focusing on cloud security with a few exceptions. Mundada *et al.* present Silverline [18], a system that allows cloud providers to offer data and network isolation for cloud-based services with the goal of auditing and preventing data leaks resulting from misconfigurations and side-channel attacks from co-resident cloud tenants. While the authors recognize the efficiency of existing information flow control techniques on the cloud in tracking user data and preventing data leakages, they seem to focus on server- side protection mechanisms without investigating data leakages through the web browser.

Finally, there are many proposals for defending against XSS exploitation in the traditional client/server web model. There are hybrid frameworks that require modifications to both server and client [12, 14, 19], server-side only [25], and client-side only [20] approaches. Although these frameworks have been designed towards the right direction they experience various limitations. One fundamental problem they have to deal with is that web applications are separated in a server-side part, which is organized and delivered by a web server, and a client-side part, which is executed in the browser. The lack of semantics between policies expressed at the server and policies executed in the client, and the variety of different browser brands, limit these solutions. Overcoming these issues in the split browser model is however possible because the pre-rendering process occurs in the cloud.

5. CONCLUSION

In this paper, we explored a new browsing paradigm, namely split browsing. This model has been gradually becoming more popular, mainly due to the fact that Amazon ships KindleFire, a tablet purchased by millions of users, with its own custom split browser, Silk. We discussed how this kind of browsers operate and highlighted their major differences with traditional web browsers. We further took advantage of this new browsing paradigm for offering protection against web exploitation. Our two architecture proposals, *in parallel* and *inline* cloning, utilize the inherent features of split browsing to offer more accurate, and with less overhead, protection based on dynamic data tracking. We envision this paper to be the beginning of an ongoing research effort for new browser paradigms that will revise the traditional security technologies as a transition from the traditional client/server web browsing model to the cloud-based browsing trend is expected to happen.

Acknowledgments

This work was supported by DARPA through Contract FA8650-11-C-7190. Any opinions, findings, conclusions or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or DARPA. It was also supported in part by the FP7-PEOPLE-2010-IOF project XHUNTER, No. 273765.

6. REFERENCES

[1] SPDY: An experimental protocol for a faster web. The Chromium Projects. `http://www.chromium.org/spdy/spdy-whitepaper/`.

[2] Internet Appliance - new cheap Internet access for schools from Icentrix. trainingzone, April 1999. `http://www.trainingzone.co.uk/item/3756`.

[3] IDC: More Mobile Internet Users Than Wireline Users in the U.S. by 2015, September 2011. http://www.idc.com/getdoc.jsp?containerId=prUS23028711.

[4] iSuppli Report: Kindle Fire Takes Off, Apple Loses Grip. ANDROID AUTHORITY, February 2012. http://www.androidauthority.com/isuppli-kindle-fire-gaining-on-ipad-54230/.

[5] D. Alperovitch and G. Kurtz. Hacking Exposed: Mobile RAT Edition. In *RSA*, Febrary 2012.

[6] D. Auerbach. EFF Gets Straight Privacy Answers From Amazon About New "Silk" Tablet Browser. https://www.eff.org/2011/october/amazon-fire%E2%80%99s-new-browser-puts-spotlight-privacy-trade-offs, October 2011.

[7] P. Bisht and V. N. Venkatakrishnan. XSS-GUARD: Precise dynamic prevention of cross-site scripting attacks. In *Proc. of the* 5^{th} *DIMVA*, pages 23–43, July 2008.

[8] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proc. of the 2009 CCSW*, pages 85–90, November 2009.

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: elastic execution between mobile device and cloud. In *Proc. of EuroSys'11*, pages 301–314, April 2011.

[10] B. Davis and H. Chen. DBTaint: cross-application information flow tracking via databases. In *Proc. of WebApps'10*, June 2010.

[11] D. Goodin. At hacking contest, Google Chrome falls to third zero-day attack. Arstechnica, March 2012. http://arstechnica.com/business/2012/03/googles-chrome-browser-on-friday/.

[12] M. V. Gundy and H. Chen. Noncespaces: Using randomization to enforce information flow tracking and thwart cross-site scripting attacks. In *Proc. of the* 16^{th} *NDSS*, February 2009.

[13] A. Ho, M. Fetterman, C. Clark, A. Warfield, and S. Hand. Practical taint-based protection using demand emulation. In *Proc. of EuroSys'06*, pages 29–41, Arpil 2006.

[14] T. Jim, N. Swamy, and M. Hicks. Defeating script injection attacks with browser-enforced embedded policies. In *Proc. of the* 16^{th} *WWW*, pages 601–610, May 2007.

[15] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis. libdft: Practical dynamic data flow tracking for commodity systems. In *Proc. of the* 8^{th} *VEE*, pages 121–132, March 2012.

[16] A. Ku. Amazon Silk: Assisted Web Browsing (Sort Of). tom's hardware http://www.tomshardware.com/reviews/amazon-kindle-fire-review,3076-7.html, November 2011.

[17] H. Moore. Cracking the iPhone (part 1). Metasploit, October 2010. https://community.rapid7.com/community/metasploit/blog/2007/10/11/cracking-the-iphone-part-1.

[18] Y. Mundada, A. Ramachandran, and N. Feamster. SilverLine: Data and network isolation for cloud services. In *Proc. of* 3^{rd} *HotCloud*, June 2011.

[19] Y. Nadji, P. Saxena, and D. Song. Document Structure Integrity: A robust basis for cross-site scripting defense. In *Proc. of the* 16^{th} *NDSS*, February 2009.

[20] F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-site scripting prevention with dynamic data tainting and static analysis. In *Proc. of the* 14^{th} *NDSS*, February 2007.

[21] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proc. of the* 12^{th} *NDSS*, February 2005.

[22] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: versatile protection for smartphones. In *Proc. of the* 26^{th} *ACSAC*, pages 347–356, December 2010.

[23] R. Sekar. An efficient black-box technique for defeating web application attacks. In *Proc. of the* 16^{th} *NDSS*, February 2009.

[24] Site Specific Browser. Turn any web site into a Windows Program or Mac App. http://sitespecificbrowser.com/.

[25] M. Ter Louw and V. Venkatakrishnan. Blueprint: Precise browser-neutral prevention of cross-site scripting attacks. In *Proc. of the* 30^{th} *IEEE Symposium on Security & Privacy*, May 2009.

[26] theharmonyhuy. Recent Facebook XSS Attacks Show Increasing Sophistication. Social Hacking, April 2011. http://theharmonyguy.com/oldsite/2011/04/21/recent-facebook-xss-attacks-show-increasing-sophistication/.

[27] S. J. Vaughan-Nichols. The mobile web comes of age. *Computer*, 41(11):15–17, November 2008.

[28] E. Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *LOGIN*, 33(5):18–23, October 2008.

[29] W. Xu, S. Bhatkar, and R. Sekar. Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks. In *Proc. of the* 15^{th} *USENIX Security Symposium*, July 2006.

[30] A. Yip, X. Wang, N. Zeldovich, and M. F. Kaashoek. Improving application security with data flow assertions. In *Proc. of the* 22^{nd} *SOSP*, pages 291–304, October 2009.

[31] A. Zavou, G. Portokalidis, and A. D. Keromytis. Taint-exchange: a generic system for cross-process and cross-host taint tracking. In *Proc. of the* 6^{th} *IWSEC*, pages 113–128, November 2011.

[32] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in HiStar. In *Proc. of* 7^{th} *OSDI*, November 2006.

[33] D. Zhu, J. Jung, D. Song, T. Kohno, and D. Wetherall. TaintEraser: Protecting sensitive data leaks using application-level taint tracking. *ACM Operating Systems Review*, 45(1):142–154, 2011.