Research Statement

Yinzhi Cao

My thesis research aims to build a principal-based browser architecture to detect, quarantine, and prevent various Web-borne security threats, which happen within, across, and outside Web principals[1]. As an analogy, if we consider a Web principal as a bottle, and the resource as the liquid inside the bottle, the task of my research is to pour the liquid into its corresponding bottle, and then give the bottle the correct label, also known as *origin* in the context of Web security. To achieve the goal, there are three fundamental questions to answer.

(1) How to make and strengthen a principal? A principal defines the boundary of resources isolated from other principals, and the penetration of other principals will cause a privilege escalation attack. Therefore, researchers need to build a principal as strongly as possible. Considering the bottle and liquid analogy, we need to prevent liquid from leaking out of a bottle by strengthening the bottle.

Two pieces of my work, Virtual Browser [2, 3] and WebShield [10], fall into this category. Virtual Browser, a concept comparable to a virtual machine, strengthens a browser principal by creating a virtualized browser to sandbox third-party JavaScript, and prevent drive-by download and cross-site scripting (XSS) attacks. My other work, WebShield, moves heavy and potentially dangerous client-side computation to a proxy sitting in between a client and a Web server. In other words, WebShield strengthens a browser principal by reducing it to a thin terminal, and lets the proxy handle the vulnerable contents.

Like two sides of a coin, defense and detection are complementary to each other. My two other pieces of work, JShield [4] and MPScan [11], try to detect those malicious contents penetrating existing principals, a/k/a drive-by download attacks, in normal Web traffic and Adobe Portable Document File (PDF). In those two works, I propose an opcode level vulnerability signature, including a definitive state automaton, plus a variable pool, to represent the inherent property of a certain vulnerability, and use the signature to match the opcode sequence containing an exploit. Both systems have been adopted by Huawei, the world largest telecommunication equipment maker, and filed in a U.S. patent.

(2) What is to put inside a principal? A principal, being empty after creation, is filled by a server and/or other principals. A careless selection of contents inside a principal will lead to an XSS attack, comparable to the case where a mixture of vinegar and soda causes the explosion of the bottle.

Two pieces of my work, PathCutter [9] and the measurement of add-on XSS [8], fall into this category. PathCutter selects user-generated contents, i.e., those that may contain malicious JavaScript, and puts them inside an isolated Web principal. Although XSS attacks can still succeed, PathCutter cuts off the propagation path of the attacks, and mitigates further infections to other users. On the other hand, my work on measurement of add-on XSS, a type of XSS initiated by a user's fault from the client side, focuses on how and when a user mistakenly puts malicious

[1] In this statement, a Web principal, as borrowed from operating systems and defined by existing work [12], is an isolated security container of resources inside the client browser.

contents into a Web principal. We recruit Amazon Mechanical Turks, and study their behavior given an add-on XSS.

(3) How to connect two principals? Principals, being isolated in a Web browser, sometimes need to talk to each other through a well-established channel, e.g., in the case of single-sign on (SSO), a relying party such as *New York Times* needs to talk to an identity party such as Facebook for a user's credential. There are two sub-questions in this category.

- *(i) How to label a principal?* An origin gives each principal a name, to be distinguished from other principals. For example, a server can differentiate requests from various client-side principals by the attached *origin* header. Similarly, a bottle should have a label to show the contents inside, such as water, vodka, and a cocktail mixed from vodka and gin, so that people could choose the correct bottle that they want.

- *(ii) How to create a channel for two principals?* Once a principal finds the correct target to talk to, those two need to negotiate to create a channel, like a pipe between two bottles to transfer liquid.

Two pieces of my work, configurable origin policy (COP) [5] and securing Web single sign-on [6], fall in the category. COP solves the problem of labeling a principal, by giving each principal a secret identity to represent it. Then, my work on securing Web single sign-on creates a bi-directional, secure channel upon the existing *postMessage* channel between two principals.

Other than my thesis focusing on *Web security*, my **broad** research interest actually spans the network and system areas, including *network diagnosis*, *memory management*, *security of cyber-physical systems (CPS)*, and *smart phone security*. Rake [13, 14], a network diagnosis tool, links packages in a distributed system by their semantics, and finds the problematic nodes. BrowserThread [1], an architecture to save browser memories, combines security and system in the concept of principal. Since a principal may include multiple browser frames, common JavaScript objects between frames can be reduced to one, save browser memory. SafePay [7], a substitute for magnetic credit card connected to a smart phone, protects credit card payments against forgery, and remains backward compatible with existing magnetic card readers. Further, my ongoing work on smart phone security tries to improve the accuracy of static analysis of Android applications by discover the implicit edges in the Android framework.

Future work in Web security, in my opinion, turns on how to solve the three aforementioned fundamental questions, and how to apply techniques for solving these questions to real-world practical problems.

None of the three aforementioned questions are fully solved. A stronger principal creates higher overhead, and it is a tradeoff to find the balance between security and high performance. Similarly, if we define a finer-grained principal by putting less content into a principal, we need to introduce more communication between principals, i.e., additional overhead. On contrary, if we define a coarse-grained principal by putting more content into a principal, security may be compromised. Again, a balance point is necessary.

More importantly, these three fundamental problems do not stand alone by themselves, i.e., they interact greatly with the most cutting-edge Web-borne threats. I will illustrate the point from two possible directions: defending Java zero-day vulnerabilities and preventing third-party Web tracking.

Java has recently exhibited a huge number[2] of zero-day vulnerabilities, most of which belong to privilege escalation. A malicious Java program from the server can bypass the access control enforced by the sandbox of Java virtual machine (JVM) in the browser, and gain high privileges of the JVM. When applying the principal-based browser architecture to Java, a browser plug-in, we need to isolate the server-provided Java program with low privilege, and Java library with high privilege, into different principals. Then, a communication channel will be created for the server-provided Java program to use the Java library. In sum, the principal-based browser architecture can also be adopted for JVM.

Web tracking attracts the public's attention, since privacy becomes an important issue for the Internet. Among those, third-party Web tracking is more important, because your browsing behavior is leaked to a third-party that you may not know. In one sentence, those third-party Web sites use cookies (or its counterparts) to fingerprint a user, and *referer*[3] header to record which Web site the user has visited. When applying the principal-based browser architecture to third-party Web tracking, we need to isolate a third-party web site with *referer* header as `a.com` and that with a *referer* header as `b.com`. Thus, the third-party Web site cannot connect the behaviors of that user when he visits both `a.com` and `b.com`.

In conclusion, in the past, I have worked with many researchers from academic institutions (such as Carnegie Mellon University, University of California Santa Barbara, Texas A&M University, and Tsinghua University in China) and research labs (such as Microsoft Research, NEC Labs America, and SRI International) to those who are from industry (such as Huawei) on those cutting-edge Web security problems by a principal-based browser architecture. In the future, I will continue working on those emerging Web-borne threats, such as Java zero day vulnerabilities and third-party Web tracking, by applying the architecture. Meanwhile, my broad interest across the network and system areas will help me to make inroads into other security-related problems, and achieve even broader effects.

References

[1] CAO, Y., LI, Z., AND CHEN, Y. BrowserThread: Redefining client-side objects management via web browser principals (under review).

[2] CAO, Y., LI, Z., RASTOGI, V., AND CHEN, Y. Virtual browser: a web-level sandbox to secure third-party javascript without sacrificing functionality (poster paper). In *ACM conference on Computer and communications security (CCS)* (2010).

[2] It is reported that Java zero-day holes appear at a rate as high as one a day (`http://www.infoworld.com/t/java-programming/java-zero-day-holes-appearing-the-rate-of-one-day-213898`).

[3] The misspelling of referer comes from RFC 1945 (`http://en.wikipedia.org/wiki/HTTP_referer`).

[3] CAO, Y., LI, Z., RASTOGI, V., CHEN, Y., AND WEN, X. Virtual browser: a virtualized browser to sandbox third-party javascripts with enhanced security. In *ACM ASIACCS* (2012).

[4] CAO, Y., PAN, X., CHEN, Y., AND ZHUGE, J. JShield: Towards complete de-obfuscation and real-time detection of complex drive-by download attacks (under review).

[5] CAO, Y., RASTOGI, V., LI, Z., CHEN, Y., AND MOSHCHUK, A. Redefining web browser principals with a configurable origin policy. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN - DCCS)* (2013).

[6] CAO, Y., SHOSHITAISHVILI, Y., BORGOLTE, K., KRUEGEL, C., VIGNA, G., AND CHEN, Y. Protecting web single sign-on against relying party impersonation attacks through a bi-directional secure channel with authentication (under review).

[7] CAO, Y., AND XIANG PAN, Y. C. SafePay: Protecting against credit card forgery with existing magnetic card readers (under review).

[8] CAO, Y., YANG, C., RASTOGI, V., CHEN, Y., AND GU, G. Abusing your browser address bar for fun and profit - an empirical investigation of add-on cross site scripting attacks (under review).

[9] CAO, Y., YEGNESWARAN, V., PORRAS, P., AND CHEN, Y. PathCutter: Severing the self-propagation path of xss javascript worms in social web networks. In *Network and Distributed System Security Symposium (NDSS)* (2012).

[10] LI, Z., TANG, Y., CAO, Y., RASTOGI, V., CHEN, Y., LIU, B., AND SBISA, C. WebShield: Enabling various web defense techniques without client side modifications. In *Network and Distributed System Security Symposium (NDSS)* (2011).

[11] LU, X., ZHUGE, J., WANG, R., CAO, Y., AND CHEN, Y. De-obfuscation and detection of malicious pdf files with high accuracy. In *Hawaii International Conference on System Sciences* (2013).

[12] WANG, H. J., GRIER, C., MOSHCHUK, A., KING, S. T., CHOUDHURY, P., AND VENTER, H. The multi-principal os construction of the gazelle web browser. In *the conference on USENIX security symposium* (2009).

[13] ZHAO, Y., CAO, Y., GOYAL, A., CHEN, Y., AND ZHANG, M. Rake: Semantics assisted network-based tracing framework. In *IEEE/ACM IWQoS* (2011).

[14] ZHAO, Y., CAO, Y., GOYAL, A., CHEN, Y., AND ZHANG, M. Rake: Semantics assisted network-based tracing framework. *IEEE Transactions on Network and Service Management 10*, 1 (2013), 3–14.