Automated Black-box Detection of Side-Channel Vulnerabilities in Web Applications

Peter Chapman
University of Virginia
pchapman@cs.virginia.edu

David Evans
University of Virginia
evans@cs.virginia.edu

***Abstract*—Web applications divide their state between the client and the server. The frequent and highly dynamic client-server communication characteristic of modern web applications leaves them vulnerable to side-channel leaks, even over encrypted connections. We propose a side-channel leak detection system that monitors network traffic over repeated crawls of a web application. Preliminary results have shown our prototype can effectively detect side-channel vulnerabilities in popular websites.**

***Keywords*-security; web applications; side-channel leaks.**

I. Introduction

As web applications increasingly replace traditional desktop ones, developers must adapt to and protect against the security problems unique to connected computing. Web applications divide their state between the server and the client. Communication between the separated states is necessary for meaningful and efficient operation, but without care, can leak substantial information through a variety of side-channels.

Our threat model includes two scenarios. In one, an adversary wants to accumulate as much private information about a single target as possible by eavesdropping on the encrypted network traffic. The target could be a high profile individual in a government or cooperation whose search terms or personal health is valuable information. Our second scenario includes a government organization monitoring encrypted Internet traffic. Government agencies may want to track encrypted searches for topics that are censored or deemed critical to national security.

Network transfer statistics of encrypted traffic have been shown to distinguish individual pages on the World Wide Web [1]. Chen et al. demonstrated the prevalence of information leaks based on dynamic transfer sequences and sizes in high-profile web applications [2]. Their existence is unsurprising considering the nature of web applications and the lack of available tools to automate the discovery and mitigation of such attacks. Our goal is to create a tool that searches for unique and predictable correlations between network traffic and application state, indicating the possibility of an information leak.

II. Approach

We are building a completely dynamic, black-box side-channel vulnerability detection system. A black-box approach allows developers to use our tool without special customization to a particular development stack. Additionally, tracking the application network traffic negates the risk of missing side-channels hidden in unchecked components of the source code or web server.

Conceptually, the goal of the attacker is to identify the current state of the application using only network information leaked during transitions. Transitions that emit unique traffic patterns are the most useful to a potential attacker. Our prototype explores a web site and logs network traffic between state transitions. A post-crawl analysis applies a variety of heuristics to extract network traffic patterns that uniquely identify application state. The output of our tool identifies these leaks and shows developers where to apply mitigation efforts. In future work, we plan to investigate techniques for automatically mitigating discovered leaks.

A. Crawling Web Applications

Crawling Web 2.0 sites and services has long been identified as a difficult task due to their highly dynamic nature and emphasis on client-side technologies that conflict with the traditional concept of a web page [3]. The Crawljax project attempts to create finite state machines for web applications even in the presence of JavaScript and AJAX [4]. Crawljax drives an instance of the Selenium [5] testing framework for black-box manipulation and state inference of the application. In particular, Crawljax interacts with developer-specified elements and forms while monitoring the browser DOM to construct a corresponding state machine.

We have extended the functionality of Crawljax to record the network traffic generated by state transitions. This was accomplished using the existing plugin architecture built into Crawljax. The network monitoring plugins were written using the Jpcap Java library for network packet monitoring [6]. For our experiments, we log packet source, destination, and length; however, simple modifications would allow us to track inter-packet timings and any other attributes available in an IP packet header. To improve robustness, our network plugins are also aware of basic TCP features such as sequence numbering and retransmission. Importantly, our tools do not use any knowledge unavailable to a potential adversary intercepting SSL-encrypted traffic.

Ideally, we would like to visit every possible state of a web application. For any meaningfully complex utility, such as a search suggestion box, state explosion makes such a goal infeasible. For intelligent navigation the developer creates a crawling specification using a number of XML files. At a minimum, the developer must specify which elements on a site to click, using the W3C XML Path Language (XPath) syntax [7]. The crawler populates form fields using random values, but for many situations, developer input is key to discovering side-channels. Many real-world applications require existing user-accounts to function. So we have added login functionality to Crawljax that allows the developer to enumerate a number of previously created accounts.

We consider an adversary who searches for network flows that illustrate specific state transitions and are predictable between users and over time. This is important because the adversary may have limited information about the user and the user's status within the web application. To find the vulnerable network flows, we crawl the application with a number of trials, each using different user accounts. We then apply a set of heuristics to find network flows corresponding to a certain state transition that are predictable across the trials.

B. *Traffic Analysis*

The crawling stage outputs a series of HTML files for each identified state with accompanying logs detailing the captured network packets during each transition. Basic heuristics check for exact and exclusive transition matches across trials. These are flows that cannot be found in any other transition and therefore can be used to identify web application state by an adversary. We have also implemented heuristics that allow for a threshold of error in matching flows based on the observation that many transfers can vary by a few bytes, often due to different hashes appended to the end of responses and requests.

More advanced heuristics check for exclusive substrings of network traffic. One situation where such a heuristic can be illuminating is a confirmation screen in a financial web application. The confirmation screen may load only for users within a certain income bracket. The contents of the screen may vary with text for the user's name, address, and additional information. This variation may make it difficult for an adversary to identify that page based on the total transfer, but if the page loads a specific image (e.g. a check mark), the separate request and transfer for the image reveals the state of the application and the user's private information. In order for effective identification of such leaks, the state identification algorithm included in Crawljax was modified to consider combining states with a matching stripped-DOM, since the variation occurs within tags.

The post-crawl analysis outputs a HTML-formatted report listing discovered network flows with links to the identified DOM states before and after the transition. The developer can use this report to decide if the leaked information is sensitive and where to apply mitigation efforts.

Web Application	Tested Actions	Uniquely Identifiable Actions
Bing	26	24
Google	26	18
Yahoo	$17,576$	$1,265$
Google Health	98	79

Figure 1: Our prototype can identify side-channel leaks in real-world web applications.

Web Application	Expected Entropy	Reduced Entropy
Bing	4.7	0.08
Google	4.7	0.67
Yahoo	14.1	6.23
Google Health	6.61	0.21

Figure 2: Reduced entropy values (in bits) are much lower than desired.

III. Preliminary Results

With our entirely black-box approach, we can test our prototype tool on real-world web applications. We focused initially on the previously identified side-channel leaks described in Chen et al. [2].

A. *Search Engine Suggestion Fields*

The Google (http://encrypted.google.com), Bing (http://bing.com), and Yahoo (http://search.yahoo.com) search engines have been discovered to leak queries through the network traffic generated by search suggestions [2]. Suggestion fields are particularly vulnerable to side-channel attacks because they update with every keystroke and are rarely personalized to a particular user. We tested the search engine suggestion fields by scripting the typing of a single letter and measuring the accompanying network traffic. Our preliminary results are shown in Figure 1 and Figure 2. We found that for Bing and Google most letters are uniquely identifiable, and letters whose network characteristics are not unique can be identified using the same technique applied to the second letter of a query with the search space limited by the possibilities of the first letter.

In our experiment Yahoo Search did not generate suggestions until three letters were typed. We collected traffic from every combination of three letters, a total of $26^3 = 17,576$ states. While less than 8% of the three letter strings generated unique network traffic, the entropy of a query was still greatly reduced. Although many results were in groups small enough that they could likely be distinguished with a fourth letter, 65% of the strings were in groups of over 100, with the largest being 240 matching flows. We calculated the traffic of adding a fourth letter to a group of 222, a total of $222 \times 26 = 5,772$ states. Adding a fourth letter to this group, one of the most difficult cases for the attacker, reduces the entropy from 12.49 bits to 1.53 bits with 40% of the recorded flows being uniquely identifiable.

Manual analysis of the Yahoo Search data revealed that network flows matching a large number of four letter combinations returned little to no suggestions. An adversary would likely never encounter a four letter combination that was not unique or in a small group. Discarding the strings that return no suggestions, the entropy in our four letter test decreases to 0.97 bits with 48% of the remaining flows now uniquely identifiable. Based on these results we believe that Yahoo Search is also vulnerable to the same side-channel attack as Google and Bing despite not sending search suggestions until the third letter.

B. Google Health

We also tested our prototype on the Google Health (http://health.google.com) Find A Doctor functionality. The Find A Doctor tool has been shown to leak the type of doctor a user searches and by extension a user's medical condition [2]. To find a doctor the user inputs an area of medicine and a location. It is assumed that an adversary would be able to accurately determine the location and there are a total of 98 disciplines, leaving 98 possible states. Using the size of the returned search results, our prototype can uniquely distinguish 79 of the states. Of the remaining 19 states, 16 conflicted with one other result, and 3 conflicted with two others. A determined adversary may still be able to determine the condition using other side-channels leaks within Google Health and other web applications.

IV. Conclusion

Side-channel leaks of private data have been found in popular web applications and without the proper tools to identify the leaks developers cannot create proper defenses. Our detection system infers a web application state machine only using network traffic and the browser DOM. Our dynamic, black-box approach allows us to experiment and identify side-channel vulnerabilities in real-world web applications without access to source code. However, it remains an open question if effective automated mitigation techniques can be developed and if this approach can work on very large and complex websites where traffic flows are harder to distinguish.

References

[1] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *IEEE Symposium on Security and Privacy*. Society Press, 2002.

[2] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow," *The 31st IEEE Symposium on Security and Privacy*, 05/2010 2010. [Online]. Available: http://www.informatics.indiana.edu/xw7/

[3] G. Cormode and E. Krishnamurthy, "Key Differences between Web1.0 and Web2.0," 2008. [Online]. Available: http://www2.research.att.com/~bala/papers/web1v2.pdf

[4] A. Mesbah, E. Bozdag, and A. van Deursen, "Crawling AJAX by Inferring User Interface State Changes," *Web Engineering, International Conference*, pp. 122–134, 2008.

[5] (2010) Selenium. [Online]. Available: http://seleniumhq.org/

[6] (2010) Jpcap. [Online]. Available: http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html

[7] (1999) XML Path Language (XPath). [Online]. Available: http://www.w3.org/TR/xpath/