View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Archivio istituzionale della ricerca - Politecnico di Milano

Poster: Detecting WebInjects through Live Memory Inspection

Nicola Mariani*, Andrea Continella*, Marcello Pogliani*,
Michele Carminati*, Federico Maggi*†, and Stefano Zanero*

*Dipartimento di Elettronica, Informazione e Bioingegneria – Politecnico di Milano, Italy
nicola1.mariani@mail.polimi.it, {andrea.continella, marcello.pogliani, michele.carminati, stefano.zanero}@polimi.it
†Trend Micro Inc.
federico_maggi@trendmicro.com

***Abstract*—Information stealing malware—a growing threat, which provokes billion-dollar losses every year—usually obtains sensitive information by modifying the content that the user's browser renders when visiting specific (e.g., banking) websites. This poster addresses the problem of detecting when a machine is infected by such trojans. We propose IRIS, an automatic kernel-space module that analyzes the memory of the user's browser to spot the artifacts of malicious web-injections by means of a signature matching mechanism. We leverage the signatures generated by Prometheus, an automatic system that analyzes information stealing malware by observing the differences that they produce in the infected DOMs and by generating signatures of the injection behavior. Preliminary results, conducted against real-world variants of financial trojans, show that our system can successfully detect such malware.**

I. Introduction

Despite being a well-known threat, banking trojans have recently evolved, growing by 16% since Q1 2016 [1]. In 2016 financial malware infected about 2,8 million personal devices: a 40% increase since 2015 [2]. Such malware uses Man-in-the-Browser techniques to infect web browsers; they are equipped with a "WebInject" functionality that, leveraging API hooking techniques, intercepts all sensitive data in the browser context and modifies the look of selected web pages on infected hosts.

With Prometheus [3], we introduced an automatic framework for analyzing WebInject-based trojans. Prometheus observes the differences that trojans produce in an infected DOM and generates precise signatures of the injection behavior. Although our experiments showed the effectiveness of Prometheus for malware analysis, we did not fully exploit the generated signatures for *detecting* banking trojans. Indeed, detecting widespread classes of malware by looking at their "generic" effects, instead of focusing on the specific implementation, already proved its efficacy in other contexts [4].

To overcome this limitation, we propose IRIS, a client-side kernel-space module to automatically detect Man-in-the-Browser attacks that result in visible DOM modifications, independently from the malware implementation. Our system analyzes the memory of running browser processes and, leveraging live memory analysis, reconstructs the DOMs of the visited web pages to match injection signatures and spot artifacts of malicious web-injections.

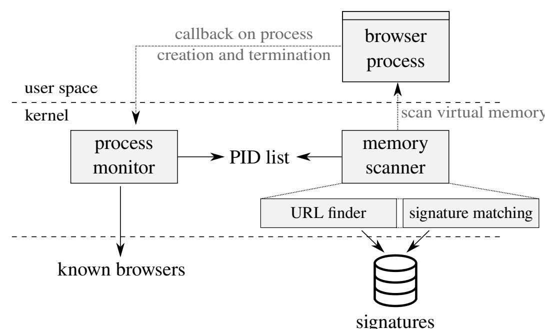

Fig. 1. Architecture of IRIS.

However, signature matching in "raw" memory is not trivial and it requires to overcome several technical challenges. First, malware can easily circumvent any detection performed in the browser context—it infects and controls the browser. For this reason, we operate at a "lower level" (kernel driver). In addition, there is a semantic mismatch between the dynamic DOM objects allocated by the browser and the "raw" and static memory view that our solution must efficiently analyze.

We implement a prototype of IRIS as a Microsoft Windows kernel driver, and we evaluate it on two distinct ZeuS and Citadel samples, analyzing a real, live banking website. We manually verified these preliminary results, showing that IRIS is able to detect when a browser is infected.

II. System Design and Implementation

IRIS is composed of two parallel processes (Figure 1): a process matcher and a memory scanner. The *process matcher* holds a list of PIDs belonging to running browser processes: It is notified whenever a new process is created or deleted, and, if the process image name belongs to a known browser, updates the PID list. Periodically, the *memory scanner* scans the memory space of each browser process, extracting the URLs of the pages being visited. If the signature database contains signatures related to the extracted URLs, it invokes the signature matching module, which scans the process memory to match its content (i.e., in-memory DOM fragments) with known signatures. If there is a match, we detect an infection.

Signatures. A signature is a tuple containing (a) a URL, (b) an XPath expression specifying the location of the injection in the

DOM, and (c) the injected content, which can be a node or an attribute. Such signatures are URL-specific and are based upon the *effect* of the banking trojan on the target web-page rather than the malware implementation: They can be generated through automated dynamic analysis (e.g., Prometheus [3]).

Memory Scanner. The memory scanner cyclically iterates through the running browser processes' PIDs and gathers the valid (i.e., allocated) virtual memory pages. It scans the process memory space to determine the memory state (*ZwQueryVirtualMemory*). If a page region has state `MEM_COMMIT`, it is valid and can be scanned for URLs or signatures: Through the *KeStackAttackProcess* routine, IRIS attaches to the target process memory space and makes a copy of the memory region to analyze. The copied memory buffer is passed to the URL finder and signature matching modules. After that, we move to the next region of addresses in the target process, repeating this procedure until a region has an invalid state (`STATUS_INVALID_PARAMETER`).

URL finder. This module finds the visited URLs in a memory region belonging to a running browser process. In general (e.g., Internet Explorer), multiple browser tabs share the same process: URLs and DOMs are scattered over the process memory. Hence, we need to perform a preliminary complete scan of the process memory to gather the relevant URLs. Instead, the multi-process architecture of some browsers, such as Google Chrome, has a different rendering process for each tab (i.e., each visited URL). In this case, we stop the linear memory scan after identifying the visited URL, performing signature matching only over the remainder of the process memory. This is possible because, as experimentally determined, in the memory address space of each Chrome rendering process, the URL is in a lower memory address than the DOM.

Signature matching. This module matches a memory buffer against the signatures targeting the visited URLs in three steps:

1) String search: First of all, it searches for occurrences of the signature content (i.e., the injected HTML/JS code) in the memory buffer using the Boyer–Moore algorithm [5]. This allows to efficiently discard signatures that do not match.

2) Refinement: If there are candidate matches, it refines the search by checking whether the string found in memory is part of a valid HTML fragment (e.g., in case of an attribute injection, the value found in memory should be an attribute with the same HTML element type described in its signature). We search the start tag of the element backward in memory, starting from the matched value; when we find the node, we use a lightweight HTML parser to validate that the string ranging from the start tag until the injected value is a valid HTML element with the type and attribute (or content in case of text injection) defined by the signature. This phase is not required when whole HTML fragments are injected.

3) XPath search: Lastly, it checks whether the *location* of the match in the DOM is the one specified in the signature's XPath expression. We proceed by walking the XPath expression starting from the node with the injected content and moving backward to the DOM root, while matching DOM nodes in the memory dump. We note that, often, the complete DOM continuously changes, and cannot be retrieved with a single live memory "snapshot." Instead of attempting to reconstruct the DOM correlating multiple memory scans, we overcome this issue by coping with partial and approximate matches (e.g., when only fragments of the DOM are available, or when the DOM is scattered around the memory).

IRIS supports three levels of match: In the best case, it matches the XPath expression in the signature up to the root node; on average, it partially matches the XPath expression; in the worst case, it matches only the signature value. The user can tune the match level required to trigger a detection according to the desired trade-off.

III. PRELIMINARY RESULTS

In order to test the correctness of our tool, we performed an experiment building a custom trojan in a controlled environment. Specifically, we got access to ZeuS and Citadel's builders. Using such tools, we built two samples defining a list of custom web-injections for a real banking website. We then manually created the signatures for such injections and run the samples in a controlled environment (VirtualBox VM running Windows 7 32 bit) where we previously installed IRIS (together with our signatures).

As a result, IRIS detected the presence of the trojans when we visited the targeted web-pages using Internet Explorer, showing it is able to successfully match such signatures only looking at the raw memory of the browser.

IV. CONCLUSIONS

Based on the finding of our previous work, which analyzes trojans equipped with "WebInject" functionalities and generates signatures of their injection behavior, we presented IRIS, an automatic kernel-space module that detects banking trojans by exploiting an in memory signature matching mechanism. Preliminary results, conducted against two distinct variants of trojans, show that our system can successfully detect when a machine is infected with a banking trojan.

Future Work. We plan to extend the evaluation of IRIS on a large dataset of distinct samples of trojans. In particular, we want to study how the different matching techniques (full XPath, partial XPath, content only) affect the performance of the detection.

REFERENCES

[1] D. Emm, R. Unuchek, M. Garnaeva, A. Ivanov, D. Makrushin, and F. Sinitsyn, "IT Threat Evolution in Q2 2016," Kaspersky Lab, Tech. Rep., Aug 2016. [Online]. Available: https://kas.pr/xE7e

[2] "Kaspersky Security Bulletin 2016," Kaspersky Lab, Tech. Rep., Dec 2016. [Online]. Available: https://goo.gl/W9dfol

[3] A. Continella, M. Carminati, M. Polino, A. Lanzi, S. Zanero, and F. Maggi, "Prometheus: Analyzing WebInject-based information stealers," *Journal of Computer Security*, Feb 2017.

[4] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "Accessminer: Using system-centric models for malware protection," in *Proc. 17th ACM Conf. on Computer and Communications Security*. New York, NY, USA: ACM, 2010.

[5] R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, Oct 1977.