Detecting Malware in Firefox Replay Recordings

Sean Liao
University of Amsterdam
Amsterdam, The Netherlands
sean.liao@os3.nl

Thomas Ouddeken
University of Amsterdam
Amsterdam, The Netherlands
thomas.ouddeken@os3.nl

I. Introduction

A. Firefox Replay

Firefox Replay [1] is a debugging tool developed by Mozilla for MacOS, implemented in 2018, available since December 2019 in Nightly builds and yet to be fully released. [2] It is only available on MacOS because the tool requires alterations to the running system code and has not yet been developed for other operating systems. Compared to the standard debugger, its main feature is the ability to record and replay JavaScript processes within a browser tab in a deterministic manner. It does this by hooking into the MacOS kernel system calls and system library calls. The call is wrapped and subsequently executed in a normal fashion, after which the thread passes the resulting data through a data stream to Replay. Locks are used to guarantee and perform atomic operations, forcing a deterministic order of inter process communications.

B. Malware analysis

Malware is one of the largest current online threats. [3] Signature based detection is a well-known defense mechanism, however, a lot of malware evolves dynamically. Being able to analyse and generally recognize and describe the behaviour of certain malware allows for detection of such a process. Being able to accurately describe what malware does and how it works will help in understanding and preventing further (ab)use of malware.

II. Related Work

Neasbitt [4] implemented record/replay (WebCapsule) for chromium and proposed that it could be used to record interactions with phishing websites.

Chen et al [5] created a replay tool, WebPatrol, that allows for researchers to analyse and gather malware information. The goal was to include more than just the binaries of malware, on which most traditional malware analysis is based. WebPatrol collects so-called web-infection trails, but does not collect as much, deterministic, and detailed information as Firefox Replay does.

Mickens et al [6] have created a generic debugging replay tool for JavaScript, however, this tool will not allow for the interception and recording of system level calls, much like WebPatrol.

rr [7] is a general time travel debugger for C/C++ projects. Webkit WebReplay [8] is a time travel debugger that works at a higher abstraction level. Edge Time Travel Debugger [9] was an experimental project for the now deprecated Edge browser.

III. Research Questions

Can we use Firefox Replay to detect, identify, and analyse the behaviour of malware in the browser and use it to detect possible artifacts may remain on the system after the attack? Since Replay is a debugger focused specifically on javascript within a single browser tab, we will be focusing our research on javascript based malware. The general problem of malware delivery through the browser may leverage other non-javascript bugs within the browser, which we think is out of scope for capturing by Replay.

A. Sub question 1

Is it possible to detect and identify malicious processes from the resulting recording file? Playing back a browser process is slow and expensive, the recording file should have enough information to allow for the detection of known (and possibly unknown) malware processes. This aims to validate that assumption.

B. Sub question 2

How to perform automated collection and archival of recordings of web based malware execution runs? This expands on the first question, building out an automated pipeline to capture and process recordings in the wild.

IV. Methodology

We will initially use malware that has already been analysed as a proof of concept. Depending on the time constraints we will then look for malware that has not yet been analysed and contribute to the community in that way. MacOS will have to be used when performing our tests, because Firefox Replay is only available in the Nightly build of Firefox on MacOS. We will be using VM's to perform our tests, not only because this allows us to run MacOS, but also add a level of security when it comes to malware analysis. The chances of browser/javascript based malware breaking out of a VM are slim and when initially, using known malware, we can make sure it does not have the capability of breaking out of a VM. Even though breaking out of a VM has happened in the past and is a possibility, we argue that the amount of malware that has this capability is small enough that a well chosen malware target along with the inherent design of VM's will reduce

the risk significantly. The VM's connections to the outside will be limited to prevent any unforeseen behaviour by the malware, such as (trying to) start infecting other machines over the internet.

V. Planning

We will be following the planning as shown in Table I

VI. Ethical Considerations

We will be using VM's on a local machine, we will make sure to limit outside connectivity to prevent any possible further infection over the internet. If any vulnerabilities are found during our research, responsible disclosure will be done as per the guidelines of the University of Amsterdam.

References

[1] Mozilla, "Replay." https://developer.mozilla.org/en-US/docs/Mozilla/Projects/WebReplay, 2015.

[2] M. Wiki, "Firefox/roadmap/updates." https://wiki.mozilla.org/Firefox/Roadmap/Updates, 2018.

[3] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," 2014.

[4] C. Neasbitt, *Advancing cyber forensics via record and replay of user-browser interactions*. PhD thesis, University of Georgia, Athens, GA, USA, 2015.

[5] K. Chen, G. Gu, J. Zhuge, J. Nazario, and X. Han, "Webpatrol: Automated collection and replay of web-based malware scenarios," pp. 186–195, 01 2011.

[6] J. Mickens, J. Elson, and J. Howell, "Mugshot: Deterministic capture and replay for javascript applications," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, (USA), p. 11, USENIX Association, 2010.

[7] Mozilla, "rr." https://rr-project.org/, 2017.

[8] B. Burg, "The mechanics of webreplay." https://trac.webkit.org/wiki/WebReplayMechanics, 2015.

[9] marron, "Time-travel debugging for javascript/html applications." https://channel9.msdn.com/blogs/Marron/Time-Travel-Debugging-for-JavaScriptHTML.

Week 9	Week 10	Week 11	Week 12	Week 13
- Update proposal - Literature research - Prepare experiment	- Prepare experiments - Run experiments	- Run experiments - Process the results	- Run experiments - Process the results - Write the paper	- Finish the paper - Prepare presentation - Presentation day

TABLE I
SUMMARISED PLANNING