Hiding in Plain Sight: An Empirical Study of Web Application Abuse in Malware

Mingxuan Yao, *Georgia Institute of Technology;* Jonathan Fuller, *United States Military Academy;* Ranjita Pai Kasturi, Saumya Agarwal, Amit Kumar Sikder, and Brendan Saltaformaggio, *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity23/presentation/yao-mingxuan

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: Hiding in Plain Sight: An Empirical Study of Web Application Abuse in Malware

Mingxuan Yao[1], Jonathan Fuller[2], Ranjita Pai Sridhar[1], Saumya Agarwal[1], Amit K. Sikder[1], Brendan Saltaformaggio[1]
[1]*Georgia Institute of Technology* [2]*United States Military Academy*

A Artifact Appendix

A.1 Abstract

The artifact is a code repository (with supporting documentation) for Marsea, an automated concolic analysis pipeline used to perform a scalable and retroactive study of malware that abuses web applications. Marsea consists of the backend for malware's symbolic execution and the hook project for dynamic binary instrumentation.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact should not pose any inherent security, privacy, or ethical concerns. No preexisting data is read or transmitted. If the user decides to install and run active malware for the purpose of verifying Marsea's claims, they do so at their own risk. For security and ethical reasons, the artifact does not include any active malware.

A.2.2 How to access

The artifact is a code repository and well-documented tutorial thta can can be accessed on GitHub: https://github.com/CyFI-Lab-Public/MARSEA/tree/fc53c4629065eeaad78258a11d950265cb059c5d

A.2.3 Hardware dependencies

Marsea requires a Linux machine and a Windows Machine.

A.2.4 Software dependencies

The preferred environment for running Marsea is Ubuntu 22.04 LTS (Long Time Support). However, Marsea *should* work with any recent version of Ubuntu. Given the fact that Ubuntu is Debian based operataion system. Marsea *should* also work on Debian 11 and up (64 bit).

Another important component of Marsea is its customized DLL (Dynamic Linked Library), which enables Marsea to instrument the malware and introduce symbolic value amid the execution The maintaining and the building of this DLL project requires a Windows machine (Windows 7 or above). The preferred environment for the DLL project is Microsoft Visual Studio (2017 or above), with Windows 8.1 SDK and version 141 Platform Toolset. However, the newer version of SDK and Platform Toolset *should* works as well.

A.2.5 Benchmarks

The primary bench mark used in the paper is a collection of Web-App-Engaged (WAE) malware, such as information stealer, dropper, and other types of malware that target web applications. because it represents a common and important threat to web applications and it allows for a thorough evaluation of the effectiveness of the Marsea. The benchmark was run on the Ubuntu 22.04 LTS operating system with Marsea deployed, and the results are show in Table 2 of the paper. We also performed the baseline concrete execution comparison using Marsea with no instrumentation.

A.3 Set-up

A.3.1 Installation

Users should follow the Setup section of the README to deploy Marsea.

A.3.2 Basic Test

Users should follow the Usage section of the README, which covers includes a step-by-step tutorial of running Marsea against malware, *Razy*, which abuses the Twitter to resolve the C&C server address. Notably, to avoid the possible security risk, we verified that the resolved C&C server has already been mitigated.

Running Marsea against *Razy* should reveal:

1. The context-rich execution trace of the target malware. For *Razy*, Marsea explores different paths and reveals the its connection to VirusTotal, Twitter, and the backend C&C server.
2. Reconstructed network sessions initiated by the target malware. For *Razy*, Marsea reveals (a) a connection to

the VirusTotal with an API key in the request and the malware itself as payload; (b) a connection to Twitter with an user specified.

3. The malicious vectors the malware performed given the abused web apps. For *Razy*, Marsea reveals (a) the flooding attack the malware performed towards VirusTotal; (b) the backend C&C server resolving by abusing the tweet posted on the Twitter.

The README contains step-by-step instructions for deployment and provides running examples to assist users in verifying the successful completion of each phase.

A.4 Evaluation workflow

This subsection serves to illustrate the assertions made in our paper. However, due to ethical considerations, we are unable to release the malware dataset utilized in our research at this time. Consequently, users are required to obtain their own malware dataset for analysis.

A.4.1 Major Claims

(C1): Marsea is able to identify 40 abuse vectors across 20 malware samples. This is proven by experiment (E1) described in Section 4.1 of the paper and illustrated in Table 2.

(C2): Marsea identified 86% of vectors compared with concrete execution. This is proven by experiment (E2) described in Section 4.1 of the paper and illustrated in Table 2.

(C3): Marsea revealed a 226% increase in malware only relying on web apps since 2020, showing malware's growing adoption of web app abuse. This is proven by experiment (E3) described in Section 5.1 of the paper and illustrated in Table 3.

(C4): Marsea revealed 893 WAE malware in 97 families abusing 29 web apps. This is proven by experiment (E4) described in Section 5.2 of the paper and illustrated in Table 4.

(C5): Marsea found that 48% of 893 WAE malware remained actiev until the day of our study. This is proven by experiment (E5) described in Section 5.3 of the paper and illustrated in Table 5.

(C6): Marsea revealed that WAE malware could have infected up to 909,788 victims from 33 abused web app content. This is proven by experiment (E6) described in Section 5.4 of the paper and illustrated in Table 6.

A.4.2 Experiments

(E1): [10 human-days + 3 compute-days + 300GB storage]: Evaluate the performance of Marsea in ground truth dataset.
Preparation: Collected malware are manually reverse-engineered to derive ground truth.
Execution: Run Marsea against the malware in the ground truth dataset and collect the generated results, such as the execution trace, malicious vectors, and reconstructed network sessions.
Results: Marsea should be able to identify the web apps that have been abused and detect most malicious vectors.

(E2): [5 human-days + 3 compute-days + 400GB storage]: Compare the performance of Marsea with the concrete execution.
Preparation: Prepare for concrete execution analysis and set up Marsea.
Execution: Analyze the malware using both Marsea and concrete analysis techniques, and compare the malicious vectors identified by Marsea and the concrete analysis.
Results: Marsea should be able to identify more malicious vectors and abused web apps than concrete analysis.

(E3): [10 human-days + 10 compute-days + 5TB storage]: Execute Marsea on the large scale to evaluate the prevalence of WAE malware.
Preparation: Collect malware from online resources. To ensure an unbiased dataset, the collection should be random and normalized across the timeline (i.e., the same number of samples should be taken for each evaluated time slot). Use domain reputation resources to evaluate the maliciousness of the communication targets.
Execution: Run Marsea against the malware and collect the communication targets. Extract the effective Second-Level Domain (eSLD) for each communication target and measure its maliciousness using domain reputation resources.
Results: Marsea should be able to identify the increase of WAE malware in the last three years.

(E4): [3 human-days + 20 compute-days + 10TB storage]: Run Marsea on large-scale WAE malware to evaluate the capabilities the abused web apps provide attackers.
Preparation: Collect WAE malware.
Execution: Run Marsea on a large-scale WAE malware collection, collect the execution trace, and identify the vectors.
Results: Marsea should be able to a wide range of vectors the malware could perform using web apps.

(E5): [10 human-days + 2 compute-days + 50GB storage]: Examine the effectiveness of the current mitigation of WAE malware by evaluating the activity of the malicious web app content.
Preparation: The communication targets as the intermediate results from E4. VirusTotal access is required to identify the first seen date and the last seen date of the malware.

Execution: Query the first seen and last seen date of the analyzed malware on VirusTotal. Then, run the web app harvest component of Marsea to extract the first creation time and the last update time of abused web app content.
Results: Marsea should be able to show that some web app content can remain on the platform for a long time, even after the corresponding malware has been identified. Note that users may need to write their own harvest components to support additional web apps.

(E6): [3 human-days + 2 compute-days + 20GB storage]: Use the engagement data on the web app platform to prove the large scope of infection caused by WAE malware.
Preparation: The communication targets results intermediate results from E4.
Execution: Run Marsea's web app harvest component to extract the engagement data from the abused web app platforms.
Results: Marsea should be able to identify a significant scope of infection on specific web apps.

A.5 Notes on Reusability

Marsea has a wide range of in-house scripts that can be used directly or easily extended for other research purposes.

- **`custom-hook.cpp`**: A general framework to inject the DLL into the target program.
- **`utils.cpp`**: An in-house extension built on top of S2E symbolic analysis framework. It supports symbolic tag extraction, on-demand concretization, VM-to-host dropped file transferring, memory symbolic expression extraction, and taint analysis logic.
- **`forkprofiler.py`**: This is an investigation-oriented analysis tool that iterates through the execution traces generated by Marsea and reports the triaged path explosion source (i.e., system APIs) if there is any.
- **`NewCodeSearcher.cpp`**: This is a code-coverage-driven exploration technique that has been implemented as an S2E plugin. Unlike the default exploration technique, which picks a random module and then a random execution state to explore, our technique prioritizes unexplored code regions in the target module being analyzed.
- **`LibraryCallMonitor.cpp`**: This is a customized built-in S2E plugin that provides detailed execution tracing by logging all system APIs invoked by the target binary during analysis.
- **`CyFiFunctionModels.cpp`**: This is an in-house S2E plugin that provides the backbone functionality for the injected DLL.
- **`pipeline.py`**: This is the pipeline script used to create a Marsea project and terminate the analysis in case of a timeout.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.