Poster: OpenAPT – Open-Source Advanced Persistent Threat for Academic Research

Mordehai Guri, Tom Sela, Yuval Elovici
Department of Information Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva, Israel
gurim@post.bgu.ac.il, tomse@post.bgu.ac.il, elovici@bgu.ac.il

***Abstract*— Modern advanced malware developers are always adapting new techniques in order to evade security systems. Typical Advanced Persistent Threat (APT) might utilize sophisticated stealth mechanisms, polymorphism engines, anti-forensic capabilities, unique covert channels, and new infection vectors. Security companies such as AV vendors are constantly updated with the state-of-the art threats, which allows them to develop new defense mechanisms. However, academic security research suffers from the lack of access to the latest APTs information. Malware source-code, implementation details and even binaries are commonly not available publicly, preventing innovative research in the scientific community. In this paper we present the work-in-progress of OpenAPT, a community supported, open-source advanced malware development and documentation framework. Providing researchers code-samples and documentation of malware and set of APT mechanisms to compile and test against their new security mechanisms. The framework's contents are all available under the GPL license, inviting the community to freely use and contribute to the collaborative knowledge.**

Keywords— Cyber Security; Academic Research; APT; Malware; Datasets; Open Source; Advanced Persistent Threat

I. Introduction

The academic world of cyber security is at a constant race to keep up with new threats encountered "in the wild". Modern malware and APTs deploy an increasingly wide range of mechanisms while executing their malicious actions. Attacker's collaborations further increase the variety and the spread of sophisticated and innovative malware mechanisms [1].

Conducting an academic research in the field of cyber security tends to rely on testing the theories on malicious code found in the wild or on software created by the researcher aimed to simulate the actions of the threat. For academic research to reflect prudent and innovative science, the work needs to address a number of concerns relating to the correct use of the datasets, and presentation methodology in a fashion sufficiently transparent to enable reproducibility.

A research done in 2012 [2] tested 36 academic publications from 2006–2011 that rely on malware execution. They found frequent shortcomings, including problematic assumptions regarding the use of execution-driven datasets (25% of the papers), and often times insufficient description of the experimental setup.

Researchers face difficulties with:

- Obtaining information about latest malware and APTs mechanisms.
- Obtaining code/binary samples executing the desired mechanisms.
- Conducting scientific experiments reflecting practical malware behavior.
- Presenting their experiments and evaluation in a transparent reproducible fashion.

The absence of an accessible and updatable source of information for academic researchers is detrimental in the quality, quantity and reliability of the academic work being done. The OpenAPT Framework is designed to satisfy the requirements of a prudent and innovative scientific research. A framework updated by a collaboration of academic researchers, white-hat hackers and black-hat hackers dedicated for creating a single database of open-source code of various mechanisms used by modern malware and APTs. Using the OpenAPT, academic researchers could conduct research and test their assumptions against the most updated malware techniques currently known. Furthermore, by referring to the specific modules used in the research, the experiment could be reproduced and validated by other cyber security researchers around the world.

II. Open Source And Security

There's been a lot of debate about the impact of open source approaches on security. One of the key issues is that open source exposes the information to everyone, both the attackers and defenders. Attackers already distribute information about vulnerabilities and threats through a large number of channels; secrecy would leave academic security defenders vulnerable, while doing nothing to inhibit attackers. Moreover, as seen in other open-source projects, having a community cooperative can often lead to more diverse and cutting edge research [3].

III. Design and implementation

OpenAPT project will be hosted in a dedicated website[1], consists of three main parts: project source-code, wiki documentation and community forums. As Microsoft Windows is the most malware targeted OS [4], OpenAPT currently focuses on Windows OS.

1will be launched in "*http://www.openapt.org*"

A. *OpenAPT Source-Code*

The source-code section is compound of eight different modules. These modules represent mechanisms used in the various stages of the APT lifecycle [5] divided into separated groups. A module is a set of source code samples; each sample is an implementation of a known technique. A researcher can use these samples to test his work against the various known techniques. Each module will be consistently updated by the community to include newly discovered techniques.

The modules are:

1. Spreading and Infection Module – Techniques of spreading in the file system and the network. (Various APT lifecycle stages, particularly in "Expansion" phase)
2. Code-Obfuscation Module - Techniques of manipulating the code to conceal its purpose or its logic, in order to prevent tampering and reverse engineering. This module will be taking a place at the pre-compilation phase using macros. (Various APT lifecycle stages, particularly in "Persistence" phase)
3. Anti-Forensic Module – Techniques of Anti-Debugging, Virtualization/Sandboxing Detection, Anti-Honeypot, Anti-Anti-Virus, and forensic tools scanners. (Various APT lifecycle stages, particularly in "Persistence" phase)
4. Polymorphism and Encryption Module – Techniques of modifying the malware in order of making detection by security programs difficult. (Relevant to all APT lifecycle stages, particularly "Internal compromise" and "Maintain Presence")
5. Reconnaissance Module - Techniques to collect information on the surrounding infrastructure, information assets and domain structure. (APT lifecycle stage "Internal Reconnaissance")
6. Covert Channels Module – Techniques to discover and leak information by using the network in a manner that violate security policies. (APT lifecycle stages "Initial intrusion" and "Outbound connection initiated")
7. Exploits Module – Techniques of taking advantage of vulnerabilities in order to cause unintended or unanticipated behavior such as acquiring administrator privileges, creating backdoors, exporting stolen data, etc. (Relevant to all stages of the APT's lifecycle)
8. Stealth Module – Techniques of Hiding Process, Files, Objects, I/O Operations, Outgoing Connection, CPU Activity etc. (Relevant to all stages of the APT's lifecycle)

B. *Wiki Documentation*

Wiki Documentation regarding the various mechanisms and techniques. The documentation pages contain explanations and information such as: related academic research, course of action, origin malware, discovery dates and prevention techniques.

C. *Forums*

Open discussion boards on various subjects for all users and researchers, to ask questions and share knowledge regarding security research and development.

IV. Usage Senarios

- Obtaining individual code snippet of mechanisms from the framework, to integrate in the researcher's work. E.g. using the code-obfuscation or encryption techniques to test the researcher's newly developed detection methods.
- Using the framework to build an experimental malware to evaluate the research.

```
// Initialize Modules
SteslthModule* pSthealth = init_StealthModule();
AntiForensicModule* pAntiForensic= init_AntiForensicModule();
CovertChannels* pCovertChannels = init_CovetChannels();

int main() {
...
// check whether the malware is running inside sandbox using
// all implemented methods
If (pAntiForensic->detectSandbox(DETECT_VMWARE_ALL) ||
(pAntiForensic->detectSandbox(DETECT_VBOX_ALL))
     terminateProcess(0);
...
// Set process, file and registry hiding using various methods
pStealth->setProcessHideMethod(PH_3);
pStealth->setFilesHideMethod(FH_FILTER_DRIVER_7);
pStealth->setRegistryHideMethod(RH_REGHOOK_1);
...
// hidden channel to C&C server using DNS requests
pCovertChannels->setMethod(CC_DNS_9);
pCovertChannels->sendMessage(buff, server_ip);
...
}
```

Fig. 1. Abstract example of using the framework - choosing the components of the expeimental malware.

- Updating the database by Researchers, black-hat and white-hat hackers to enrich the modules with new functionality and implementation methods.

References

[1] A. Raff, "Citadel – An Open-Source Malware Project," Seculert, 8 Feb 2012. [Online]. Available: http://www.seculert.com/blog/2012/02/citadel-open-source-malware-project.html.

[2] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos and M. Steen, "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook," in IEEE Symposium on Security and Privacy (SP), San Francisco, CA, 2012.

[3] C. Raasch, C. Herstatt and N. Abdelkaf, "Creating Open Source Innovation: Outside the Software Industry," in PICMET, Cape Town, South Africa, 2008.

[4] Kaspersky, "Kaspersky Security Bulletin 2012. The overall statistics for 2012," 10 Dec 2012. [Online]. Available: http://www.securelist.com/en/analysis/204792255/Kaspersky_Security_Bulletin_2012_The_overall_statistics_for_2012

[5] SecureWorks, "Lifecycle of an Advanced Persistent Threat," Dell, Counter Threat Unit research , 2012.