Sensitive Behavior Acquisition Procedure of Midas (Supplementary Material)

In order to acquire behavioral characteristics of IoT malwares from the data source, we refer to the surveys conducted by Cozzi et al. [1], Alrawi et al. [2] and Rieck et al. [3], which have been proven to be systematic and effective. We tailor these existing malware analysis methods for our sensitive behavior acquisition process, in particular, static, dynamic and capability differential analysis to collect the suspicious concrete behaviors conducted by IoT malwares.

I. Static Analysis

During static analysis, we mainly focus on the typical static characteristics of malicious behaviors conducted in each step of the IoT malware's lifecycle. This procedure consists of three main phases: binary header parsing, filesystem-like path extraction and attack vectors analysis.

First, we analyze the header information to determine the binary file type, especially the target architecture, byte-order, section information and linkage type. Such information helps to guide the subsequent dynamic analysis process by choosing an appropriate environment with the correct architecture, byte-order and libraries (e.g. *uclibc* or *musl* setup). Second, we identify and unpack UPX-packed samples statically with the *UPX executable packer* for further filesystem-like path extraction. Some malware developers may invalidate the UPX header for anti-analysis, which can crack the UPX decompression but not hinder the execution during the subsequent dynamic analysis. Next, we utilize regular expressions to extract filesystem-like paths. These paths implicitly indicate the critical resources accessed by IoT malware to reach its goals at every stage.

Finally, we conduct the YARA signature-based analysis to establish attack vector databases. The YARA signatures are built from the proof-of-concept codes of NVD and CVE verified in [2]. Since the payload or attack vector is often encapsulated in the protocol packet for remote attacks, like RCE, we rely on custom IDA scripts to check whether the offset of the signature-matched string is referenced by the *.text* section and packed in the packet. If the conditions are met, the string will be considered as an attack vector. In this way, the extracted attack vector database can provide commands executed by the malware's payloads to achieve its malicious attempts. Overall, we can parse characteristic resources as shown in Figure 1, like file paths or commands, utilized by IoT malware and combine identical static characteristic resources for further analysis.

II. Dynamic Analysis

The main goal of dynamic analysis is to collect the full trace of system calls, refine each characteristic resource with a specific operation type as a pair to record the behavior of IoT

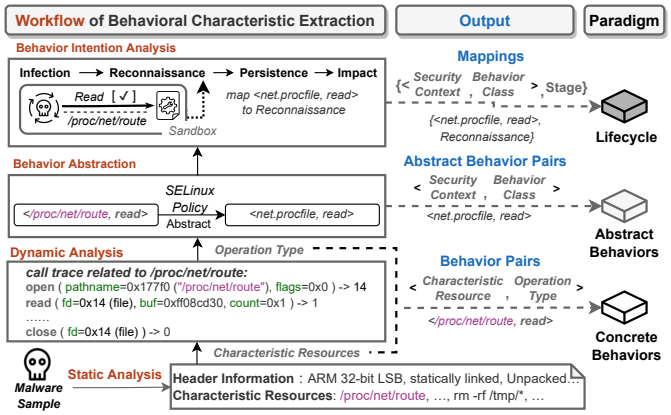

Fig. 1: The example workflow of behavioral characteristic extraction. The behavioral characteristics generated in each step will be appended to the behavior paradigm.

malwares. In order to reveal more operations of the analyzed samples, we apply the binary emulation tool, Zelos [4] with DNS response simulation and a root directory. Meanwhile, we utilize QEMU-based emulated sandboxes with OpenWrt firmwares [5], syscall tracing tools and additional libraries for full-system analysis. We carry out a 60-second analysis for each malware sample suggested as prior works [2], which is enough for over 95% of samples to get blocked or stuck in an infinite network loop.

Then, we cross-check and combine the system call trace results of full-system emulation and binary emulation, mainly focusing on aspects associated with filesystems, network and signals. With parameters and return values in call traces, we perform automatic syscall-level context analysis to determine the operation type for each characteristic resource. For instance, for filesystems, when the analysis script finds an *open()* syscall of a filesystem-like path */proc/net/route* with the return file descriptor (*fd*) valued 0x14, it will iterate the subsequent syscalls in the limited round to identify an operation with *fd* also valued 0x14 as a parameter, e.g. *read(fd=0x14, ...)*. In this way, we can obtain one behavior pair *<*/proc/net/route, read*>* of the malware sample as shown in Figure 1. For network, we obtain ports of the destination socket address. If the sample uses them in payloads recorded in the attack vector database, the access of these port resources, e.g., *connection()*, will be acquired. For signals, we are mainly concerned with signal-related calls, such as *kill()* or *rt_sign()* to identify operation types to processes, like *sigkill*. In addition, we introduce manual analysis to a few static characteristic resources, for which the script cannot extract the corresponding syscall or fail to determine the operation type. In general, the dynamic analysis generates the refined behavior pairs *<characteristic*

resource, operation type> and gives us insights on behaviors across the entire lifecycle of IoT malwares.

III. Capability Differential Analysis

Malwares tend to invoke some system capabilities across their lifecycle. For example, some malwares of the Mirai family invoke the *sys_ptrace* capability during reconnaissance against the debugging environment to prevent themselves from being analyzed. Thus, to further expose sensitive behaviors utilizing exceptional system capabilities to compromise devices, we carry out an additional analysis.

Given the significant difference of system capabilities between normal applications and malware, we apply differential analysis to delineate the scope of suspicious capabilities by comparing all existing capabilities defined in Linux [6] with the capabilities needed by normal programs of the typical IoT firmware, defined in [7]. Capabilities that are not needed by any programs, or only by highly privileged essential system modules, are considered sensitive capabilities. Finally, along with behavior pairs extracted during static and dynamic analysis, sensitive capabilities will be also included in the behavior paradigm as similar pairs <$*$ [1] , *capability*>, so that relevant behaviors can be audited and analyzed when defending against break-in malicious programs.

References

[1] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 161–175.

[2] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis, "The circle of life: A Large-Scale study of the IoT malware lifecycle," in *USENIX Security 21*, 2021, pp. 3505–3522.

[3] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, D. Zamboni, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 108–125.

[4] Z. D. LLC, "zeropointdynamics/zelos: A comprehensive binary emulation and instrumentation platform." https://github.com/zeropointdynamics/zelos, 2020, (Accessed on 12/13/2021).

[5] F. Fainelli, "The openwrt embedded development framework," in *Proceedings of the Free and Open Source Software Developers European Meeting*. sn, 2008, p. 106.

[6] SELinux Project, "SELinux Object Classes and Permissions Reference," 2013. [Online]. Available: https://selinuxproject.org/page/ObjectClassesPerms

[7] Defensec, "dssp SELinux Policy," 2021. [Online]. Available: https://git.defensec.nl/

[1] "$*$" indicates the program that uses the sensitive capability. In abstract behavior pairs of behavior paradigm, "$*$" can be converted to the security context of the intruding program, i.e. *suspicious.subj*.