Intrusion Survivability for Commodity Operating Systems and Services: A Work in Progress

Ronny Chevalier*† David Plaquin* Guillaume Hiet†
*HP Labs
Bristol, United Kingdom
Email: {firstname.lastname}@hp.com
†CIDRE Team, CentraleSupélec, Inria, CNRS, IRISA
Rennes, France
Email: {firstname.lastname}@centralesupelec.fr

***Abstract*—This paper presents a work-in-progress of our approach for intrusion survivability in commodity operating systems. Our approach relies on an orchestration of recovery and mitigation actions. We rollback infected services (i.e., their processes) and infected files to a previous known safe state, and we apply per-service mitigations (i.e., privileges removal) before unfreezing the restored processes. Such approach effectively puts the previously compromised service into a degraded mode, allowing the system to withstand ongoing intrusions and ensures the availability of core functions to the users. A prototype for Linux-based systems is currently in development.**

***Index Terms*—Intrusion Tolerance, Intrusion Recovery, Survivability, Resiliency, Intrusion Response, Availability**

I. Introduction

Organizations rely on the security of their computing platforms, such as laptops, servers, or phones, to operate properly and to provide services to their users in an uninterrupted manner. Despite progress in computer security (e.g., cryptography, coding practices, hardware protections, or access controls) to reduce the likelihood of an intrusion impacting those platforms, given time, an intrusion will eventually occur. Such case may happen due to technical reasons (e.g., a misconfiguration, a system not updated, or an unknown vulnerability) and economic reasons [1].

To limit the damage done by intrusions, intrusion recovery systems help administrators to restore a compromised system in a sane state. One common limitation, however, affecting prior work [2, 3, 4, 5], is that they stop neither the intrusion from reoccurring nor the attackers from achieving their goals (e.g., data integrity violation). If the recovery system manages to restore the system in a sane state, the system continues to run with the same vulnerabilities and nothing stops attackers to reinfect it and to accomplish their goals. Thus, the system could enter in a loop of infections and recoveries.

Our work aims at designing a system able to withstand ongoing intrusions and to allow business continuity despite the presence of an active adversary. We focus on the implementation of an architecture to provide intrusion survivability in commodity Operating Systems (OS) and their services. In particular, our approach relies on two components: service recovery and threat mitigation.

One key point of our approach is that after an intrusion is detected, it puts the system into a degraded mode following a predefined policy. Some non-essential functions of a service are no longer available due to some mitigations applied, while core functions are still operating (e.g., a core function of web server would be to always provide read access to a website). The goal here is to recover infected services and to put them in a degraded mode with fewer privileges to ensure the integrity of the OS despite the presence of an adversary, while providing availability of core functions of the service. Such degraded mode allows the system to mitigate the threat and to provide business continuity, since users of the system still have access to the essential functions. Thus, it gives time for administrators to plan an update of the system to fix the vulnerability, and potentially to wait for a patch to be released if necessary.

Our work relies on existing work in Intrusion Detection Systems (IDS) and malware characterization, but our contributions concentrate on the ability to recover from and withstand ongoing intrusions, while providing availability to core functions of a system.

This work is part of a broader approach to achieve survivability for a whole platform (i.e., its hardware, firmware, and OS). Previously, we worked on intrusion detection, and more specifically, on improving the security of firmware [6].

The rest of this document is structured as follows. First, in section II, we mention related concepts about our work, and we review state-of-the-art work on intrusion recovery systems. Then, in section III, we provide a more detailed description of our approach. In section IV, we describe the current state of our proof-of-concept. Finally, we give the next steps regarding our work-in-progress in section V.

II. Related work

In this section, we first discuss related concepts close to our work such as dependability, resiliency, or survivability. Then, we review existing work on intrusion recovery systems.

A. Concepts

Laprie [7] defined the concept of resilience as “the persistence of dependability when facing changes”, where dependability is “the ability of a system to avoid service failures that are more frequent or more severe than is acceptable” [8].

Avizienis et al. [8] suggested that dependability and survivability (based on an earlier definition [9]) were similar concepts. Knight et al. [10] weighed that survivability distance itself from dependability, since it should encompass the notion of degraded service, and a trade-off between the availability of some functions and the cost to maintain and provide them.

Thus, from these concepts, we consider our work closer to survivability. More specifically intrusion survivability, since our approach focuses on withstanding ongoing intrusions and does so by service degradation. In our trade-off, the cost to provide some core functions without applying some mitigations represents a security risk on the system.

To achieve dependability, resiliency, survivability, or other close concepts, four categories of techniques exist: fault prevention, fault tolerance, fault removal, or fault forecasting [8]. In our case, we focus on fault tolerance, since we use recovery techniques to roll back the state of objects in the system, and we isolate recovered services by applying mitigations.

B. *Intrusion Recovery Systems*

Intrusion recovery systems [2, 3, 4, 5] focus on system integrity by recovering legitimate persistent data. Except for SHELF [2], they do not preserve availability since their restore procedure either forces a system shutdown or it does not record the state of the processes. These systems log all system events to later replay legitimate operations [2, 3, 4] or rollback illegitimate ones [5], thus providing a fine-grained recovery. However, they generate gigabytes of logs per day inducing a high storage cost.

Most related to our work is SHELF [2]. SHELF is an intrusion recovery system focusing on business continuity able to recover the state of processes while performing damage assessment. It performs periodic asynchronous snapshots of files and process states on the system. It keeps a log of system call events and it uses dependency tracking to perform damage assessment when the IDS notifies it of an intrusion. During recovery, SHELF quarantined infected objects by freezing processes or forbidding access to files. However, such quarantined state is removed as soon as the system is restored.

One common limitation affecting prior work is that they prevent neither the attacker from reinfecting the system nor the attackers from achieving their goals (e.g., data theft, integrity violation, or propagation) when successfully reinfecting the system. They rely on the implicit assumption that after the recovery the attacker will stop attacking the system, that the vulnerability is fixed, or that the cost of the recovery is negligible. Unfortunately, these assumptions do not necessarily hold even if an intrusion is detected and recovered from. A worst-case scenario could be a loop of infections and recoveries impacting the availability of the services.

III. Approach description

In this section, we describe the assumptions made and the different components of our approach, illustrated in Figure 1.

A. *Intrusion Detection*

Since our work focuses on intrusion survivability, we assume an IDS able to detect intrusions on the system and able to associate such intrusion to a specific service in the OS (e.g., a web server or an SMB server). After the IDS detected an intrusion, it notifies the recovery and mitigation system about which service is infected in the OS and what are the potential characteristics and attributes of the intrusion. For the attributes of the intrusion, we rely on the Malware Attribute Enumeration and Characterization (MAEC) [11] and Structured Threat Information eXpression (STIX) [12] standard languages. These languages are used to encode threat information, such as malware attributes, behaviors, artifacts, or attack patterns.

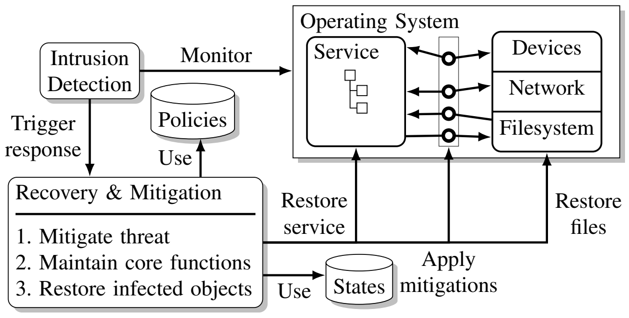

Fig. 1. High-level overview of the approach

B. *Recovery and Mitigation*

After the recovery and mitigation system received the notification from the IDS, it recovers as soon as possible the core functions of the system to maintain business continuity. The procedure must meet the following goals: maintain core functions, mitigate the threat, and restore infected objects (e.g., files and processes).

These goals are achieved using an orchestration of recovery and mitigations actions. A recovery action restores the state of a service (i.e., the state of its processes and metadata describing the service), a file, or a filesystem to a previous known safe state. A mitigation action removes privileges or isolate components of the system from the service according to the MAEC and STIX information provided by the IDS. Such mitigation either stops the service from being reinfected or stops attackers from achieving their goals (e.g., data theft or data integrity violation). After a mitigation is applied, however, not only the attackers but the service is restricted in its capacity. It effectively puts the system in a degraded mode, because some functions are not available anymore.

In order to perform recovery actions, we create periodic snapshots of the filesystem and the services, during the normal operation of the OS. In addition, we log all the files modified by the monitored services. Hence, when restoring infected services, we only restore the files they modified.

For example, let us consider a web server infected by some ransomware. After being notified, our mechanism restores the infected processes to a previous known safe state (pre-infection), and it restores all the files modified by the service since the last snapshot. In addition, before unfreezing the service, it degrades the state and applies a mitigation removing the write access privilege of the service to the filesystem. Here we achieve multiple goals: we restore the data, we continue to provide access to the web server, and we stop a reinfection from having any impact.

C. Policy

Our approach relies on policies describing a set of constraints of what can be and cannot be done for each service. Such policies help to maintain the availability of core functions of the services and provide flexibility to fit the different needs depending on the context of the system. A server, a desktop, or an embedded system have different constraints due to the different functions they provide. Hence, some mitigations or recovery actions should not be attempted due to their availability cost. The goal is to provide a trade-off between the availability of a function in a service and the security risk due to an intrusion that could benefit from the absence of some mitigations.

In our previous example with a web server, one constraint was that the server always has read access on the filesystem. Another constraint could have been that the service has always write access to a specific directory in the filesystem. Thus, the writer of the policy accepts the risk that in case of an infection such directory might be compromised.

IV. Proof of concept

In this section, we describe the work we started on the development of a proof-of-concept for Linux-based systems. Since this is still a work-in-progress, not all the aspects described in section III are implemented. In addition, we give a brief overview of some limitations of our current work.

A. Current state

We use CRIU [13], a checkpoint and restore project implemented in userspace for Linux, mainly developed to perform the checkpointing and restoring of containers. In addition, we use snapper [14] to perform snapshots of the filesystem and to fetch previous versions of files when restoring.

We modified systemd [15], a system and service manager for Linux-based systems, to checkpoint and restore services with the help of CRIU and snapper. We perform periodic atomic snapshots of the monitored services (i.e., their processes and service metadata) and the filesystem, by freezing their processes (i.e., they are removed from the scheduling queue) during the checkpointing procedure, thus avoiding any inconsistencies.

Finally, we modified systemd and CRIU to apply mitigations when restoring an infected service, before unfreezing the processes. We use seccomp [16] to filter system calls and we apply Linux namespaces [17] to isolate or remove access of parts of the system (e.g., files or devices) from the service.

B. Limitations

The periodic snapshot of the services we monitor requires freezing their processes to avoid any inconsistency during the procedure. Hence, services will be longer to respond to any request made by a user during the snapshotting procedure.

If a process has opened a device to have direct access to some hardware, checkpointing its state may not be possible (except for virtual or pseudo devices not corresponding to any physical devices). This technical limitation is because we cannot be sure that when the process is restored, the physical device (e.g., a printer) has the same state as when the process was checkpointed. One would need a standard interface to control the state of such devices in a generic way. Finally, even if there was such interface, restoring its state may create inconsistencies with other users of the device.

V. Future work

We are currently developing a proof-of-concept implementation of our approach for Linux-based systems. When finished, we plan to evaluate the performance impact during normal system workload and when an intrusion is detected, to evaluate its efficacy against different attacks, and to evaluate its efficacy to maintain the availability of core functions. After, we would like to investigate how, based on our current work, we could adapt the system to gradually remove the mitigations and achieve an optimal number of recovered functions.

References

[1] N. Kshetri, “The simple economics of cybercrimes,” *IEEE Security & Privacy*, vol. 4, no. 1, pp. 33–39, 2006.

[2] X. Xiong, X. Jia, and P. Liu, “Shelf: Preserving business continuity and availability in an intrusion recovery system,” in *Proceedings of the 25th Annual Computer Security Applications Conference*, ser. ACSAC ’09. IEEE Computer Society, 2009, pp. 484–493.

[3] A. Goel, K. Po, K. Farhadi, Z. Li, and E. de Lara, “The taser intrusion recovery system,” in *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, ser. SOSP ’05, 2005, pp. 163–176.

[4] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek, “Intrusion recovery using selective re-execution,” in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI’10. USENIX Association, 2010, pp. 89–104.

[5] F. Hsu, H. Chen, T. Ristenpart, J. Li, and Z. Su, “Back to the future: A framework for automatic malware removal and system repair,” in *Proceedings of the 22nd Annual Computer Security Applications Conference*, ser. ACSAC ’06, 2006, pp. 257–268.

[6] R. Chevalier, M. Villatel, D. Plaquin, and G. Hiet, “Co-processor-based behavior monitoring: Application to the detection of attacks against the system management mode,” in *Proceedings of the 33rd Annual Computer Security Applications Conference*, ser. ACSAC ’17, 2017.

[7] J.-C. Laprie, “From dependability to resilience,” in *38th IEEE/IFIP International Conference On Dependable Systems and Networks*, 2008.

[8] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[9] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, and T. Longstaff, “Survivable network systems: An emerging discipline,” Software Engineering Institute, Carnegie Mellon University, Tech. Rep., Nov. 1997.

[10] J. C. Knight, E. A. Strunk, and K. J. Sullivan, “Towards a rigorous definition of information system survivability,” in *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*, vol. 1. IEEE, 2003, pp. 78–89.

[11] I. Kirillov, D. Beck, P. Chase, and R. Martin, “Malware Attribute Enumeration and Characterization,” 2011.

[12] S. Barnum, “Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX),” Feb. 2014.

[13] “CRIU,” 2018. [Online]. Available: https://criu.org/

[14] “Snapper, the ultimate snapshot tool for linux,” 2018. [Online]. Available: http://snapper.io/

[15] “systemd system and service manager,” 2018. [Online]. Available: https://www.freedesktop.org/wiki/Software/systemd/

[16] J. Corbet, “Seccomp and sandboxing,” LWN, May 2009. [Online]. Available: https://lwn.net/Articles/332974/

[17] M. Kerrisk, “Namespaces in operation, part 1: namespaces overview,” LWN, Jan. 2013. [Online]. Available: https://lwn.net/Articles/531114/