Towards Rootkit Detection on Embedded IoT Devices

Roland Nagy Levente Buttyán[1]

Abstract: Rootkits are malicious programs that try to maintain their presence on infected computers while remaining invisible. They have been used to attack traditional computers (PCs and servers), but they may also target embedded IoT devices. In this work, we propose a rootkit detection approach for such embedded IoT devices, where the detection mechanism is executed in an isolated execution environment that protects it from manipulation by the rootkit. Our rootkit detection approach is focused on detecting Direct Kernel Object Manipulation (DKOM) and it is based on detecting inconsistencies caused by the presence of a rootkit in various Linux kernel data structures such as the process list, the process tree, and different scheduling queues. We also report on the current status of our implementation using OP-TEE, an open source Trusted Execution Environment.

Keywords: Embedded Systems, Internet of Things, Security, Malware

1 Introduction

Connecting embedded devices to the Internet (a.k.a. the Internet of Things or shortly IoT) enables new applications such as smart homes, intelligent transportation systems, and personalized healthcare. However, many of these new applications have stringent security and privacy requirements. Unfortunately, IoT systems are notoriously insecure. One of the reasons is that IoT devices are rather easy to compromise by exploiting weaknesses in the way they are operated and vulnerabilities of the software components running on them. A consequence of this is that malware for IoT has appeared [1, 3] and gaining momentum [4].

Sophisticated malware tries to maintain its presence on infected devices while remaining invisible for the operators of those devices. This sort of malware is called *rootkit*. Typically, rootkits run with elevated (root) privileges and they modify system commands and/or low level data structures in the operating system (OS) kernel such that their files and running processes do not appear in the output of various system tools used to monitor the operation of the devices. Detecting such a rootkit is challenging, mainly because any detection program running at the same or lower privilege levels than the rootkit may also be compromised or may be misled by the tricks used by the rootkit to hide itself.

In this work, we aim at rootkit detection on embedded IoT devices, and we address the above challenge by running our rootkit detection tool in a Trusted Execution Environment (TEE), which is isolated from the main OS of the device, and hence the rootkit – even running with root privileges on the main OS – cannot interfere with its operation. In this extended abstract, we introduce the concept of TEE and describe how our rootkit detection tool running in the TEE detects active rootkits.

2 Trusted Execution Environments

A TEE is an execution environment which is isolated from the main OS and applications running on the device (the so called Rich Execution Environment or shortly REE) by software and hardware mechanisms (e.g., based on the ARM TrustZone[2] technology). Within the TEE, trusted applications (TAs) run on top of a trusted OS. The isolation mechanisms ensure, that system resources (e.g., memory) of the REE can be accessed from the TEE, but not vice versa.

[1]All authors are with the Laboratory of Cryptography and System Security (CrySyS Lab), Department of Networked Systems and Services, Budapest University of Technology and Economics

[2]`https://developer.arm.com/ip-products/security-ip/trustzone`

Thus, secrets (e.g., keys) can be kept and critical computations can be executed within the TEE without the risk of being observed or manipulated by potentially malicious software running in the REE.

Our thesis is that an active rootkit must introduce inconsistencies in the data structures of the main OS kernel, since it must remove its own processes from some data structures used by system monitoring tools in order to maintain stealthiness, while it must keep its own processes in other data structures for being eventually scheduled and executed. Hence, our rootkit detection approach is based on detecting inconsistencies in OS kernel data structures by a TA running in the TEE. For this, our TA needs to access the memory of the main OS running in the REE.

As a TEE implementation, we use OP-TEE[3], which is an open source TEE, compliant with a widely accepted standard[4] for TEEs. In OP-TEE, by default, simple TAs are not capable for accessing the memory regions of the REE; such an access requires a so called Pseudo-TA (PTA). Our PTA is running with the privileges of the trusted OS kernel, and we can instruct this kernel to map the memory regions used by the main OS (typically Linux on embedded devices) in the REE, such that our PTA can access them.

3 Rootkit detection

Rootkits use different cloaking mechanisms to hide their presence on infected systems. A simple idea, for instance, is to hide something by corrupting the tool used for gathering information about it. On Linux, an example would be modifying the `ps` program such that it does not list some specific processes. This type of attack can be easily detected by verifying the integrity of important system programs, which we do not discuss here.

In this work, we focus on rootkits that use Direct Kernel Object Manipulation (DKOM) [2]. Such rootkits modify the underlying data structures that the kernel uses to maintain information about its specific components. For instance, if one can determine what data structure is used to populate the `/proc` virtual filesystem on Linux, then he may be able to remove a specific process from that data structure, which will then remain hidden from the `ps` command.

As, in the IoT domain, the main OS running on embedded devices is often Linux, we describe some relevant Linux kernel data structures that may be manipulated by DKOM:

task_struct: Inside the Linux kernel, this structure holds most of the information associated with processes. Internally, tasks are equivalent to threads, and any process may have several threads. Tasks of the same process share one virtual address space and many more resources.

Process list: The task structures inside the kernel memory are chained into a doubly linked circular list. In previous kernel versions, the kernel iterated through this list to populate the `/proc` directory.

Process tree: Processes are related to each other via a parent-child relationship. Every process has a parent that created it, and processes might start other processes that become their children. The `task_struct` holds a pointer to the parent of the given task, a list of pointers for its children, and another list of pointers for its siblings.

Pid namespace, IDR and the struct pid: Each namespace maintains a radix tree[5], containing pointers to pid structures[6]. These structures have lists of pointers for the tasks using

[3] `https://www.op-tee.org`
[4] `https://globalplatform.org`
[5] `https://lwn.net/Articles/175432/`
[6] `https://lwn.net/Articles/195627/`

them. This data structure is responsible for accounting for taken pids and for fast access to tasks via their pids[7]. In recent kernel versions, this mechanism populates the `/proc` directory.

Run queues: Each CPU has a runqueue structure, holding inline structures for the available schedulers. These have their own methods to keep track of runnable processes. The CFS and DL schedulers are using red-black trees[8] for this, while the real-time scheduler has a so-called `rt_prio_array`; a bitmap and an array of lists for each priority level[9].

Wait queues: Every time a process must wait for something, it is placed in a waitqueue[10], containing wait entries. Each such entry has a pointer to the task waiting, and to a function to execute when it is time to wake up the task.

The basic idea of our rootkit detection mechanism is simple: We iterate through the previously mentioned data structures, collect pids into different lists, and look for inconsistencies in the obtained lists. For instance, it is abnormal if a pid can be found in the process tree, but it is missing from the process list.

4 Implementation

Our rootkit detection solution is implemented in an OP-TEE TA, which uses a PTA to access the Linux kernel's memory in the REE as we mentioned before. In order to be able to use types and structures of the Linux kernel, we generated header files from the DWARF section of a dummy kernel module with the `dwarfparse` script[11]. In addition, we were able to retrieve useful addresses from the `System.map` file of the compiled kernel, with which we were able to locate the data structures described in the previous section.

So far, we managed to implement the following: We can request a copy of the `init` task, from which we can iterate through the list of all processes using the process list. We save the pid of each task found in the process list into an arraylist. Then, from the `init` task again, we run a depth-first search on the process tree, saving the pids found there into a separate arraylist. For each element of these lists, we look for that pid in the other list, and if not found, we save it to yet another list. If this differential list is not empty, then we found suspicious processes that appear in one of the kernel data structures but missing from the other one. After this check, a unified list is created from the first two collections of pids, and we check if all the pids found in the schedulers of all the CPUs are also a part of this list.

We tested our implementation with a simple rootkit, which creates a bind shell and removes it from the process list. We detected the rootkit by identifying a pid in the process tree that was missing from the process list.

5 Conclusion and future work

Rootkits are malicious programs that try to maintain their presence on infected devices while remaining invisible for the operators of those devices. They have been used to attack traditional computers (PCs and servers), but they may also target embedded IoT devices. In this work, we proposed a rootkit detection approach for such embedded IoT devices, where the detection mechanism is executed in a TEE, which protects it from manipulation by the rootkit running in

[7] `https://lore.kernel.org/patchwork/patch/834401/`
[8] `https://lwn.net/Articles/184495/`
[9] `https://www.linuxjournal.com/article/10165`
[10] `https://lwn.net/Articles/577370/`
[11] `https://github.com/realmoriss/dwarfparse`

the REE. Current technologies (e.g., ARM TrustZone) supports the implementation of TEEs on embedded devices, hence, our approach does not rely on far fetched assumptions, but can be readily used even today on commodity embedded boards.

The work presented here is work-in-progress. We are currently experimenting with iterating through the IDR of the initial pid namespace and collecting pids from waitqueues. We also plan to extend the functionality of our detection tool with features beyond DKOM, and we would like to cover kernel resources other than processes. Finally, we would like to test our implementation against real rootkits captured in the wild.

Acknowledgment

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai botnet. In *USENIX Security Symposium*, August 2017.

[2] J. Butler. DKOM – Direct Kernel Object Manipulation. In *BlakHat USA*, 2004.

[3] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin. Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In *Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[4] P.-A. Vervier and Y. Shen. Before toasters rise up: A view into the emerging IoT threat landscape. In *IoT Security Foundation Conference*, 2018.