EXEIN

EXEIN CORE

A host-based, run-time anomaly detection mechanism for Linux-based embedded systems.

ALAN VIVONA, ALESSANDRO CARMINATI, GIANNI CUOZZO, GIANLUIGI SPAGNUOLO, GIOVANNI ALBERTO FALCIONE.

JANUARY 2020

EXEIN S.P.A.

1. INTRODUCTION

In recent years the number of devices connected to the Internet has grown significantly and a substantial part of those are embedded devices such as routers, VOIP phones, home automation devices, industrial devices and even cars.

Unfortunately, the aforementioned are typically ripe for exploitation as little to no defensive technologies (such as AV scanners or IDS's) are available to protect them.

In the following pages we will propose a host-based defence mechanism capable of providing an embedded system with run-time anomaly detection functionality. The natural position of a piece of software providing these features is within the Linux kernel using the Linux Security Module ecosystem [1] [2].

Even though Exein is not Linux version-specific, making it run on a given Linux version may require some customization due to Linux versions specificities. In order to prove that an Exein Core can run on different Linux versions and multiple hardware platforms, it was compiled and tested on the following environments:

- Emulated device: Qemu ARM32
- Emulated device: Qemu MIPS Malta
- Real device: Raspberry PI bcm2709 (Raspberry PI 3+ in arm32 mode)
- Real device: Ramips MT7688

The chosen base systems were OpenWrt [3] versions 18.06.5 & 18.06.6 running Linux Kernel versions 4.9.198, 4.9.208, 4.14.113, 4.14.151 & 4.14.162.

In all cases, the Exein Machine Learning Engine was trained to recognize the normal behavior of OpenWrt's default web server (the httpd service [4]) and tested using both common interactions and real attacks.

Exein LSM allowed regular operations to be carried out as usual and recognized the attacks, effectively blocking them and thus successfully demonstrating the effectiveness of the solution.

Having carried on tests using more than one architecture and given the fact that the solution can also be ported to more architectures in the future, we may conclude that Exein Core represents a practical and effective protection mechanism for a wide range of devices from IoT to ICS.

2. WHAT EXEIN IS ALL ABOUT

In recent years, there have been several well-documented attacks on embedded devices [5] [6]. Despite embedded devices including password-protected logins and making use of encrypted protocols such as SSH (Secure Shell) or SSL (Secure Sockets Layer), this has been proven not enough to make them secure because they are still somewhat lacking in terms of security within their design. An example of this is the fact that IoT and embedded devices, unlike traditional computers, often lack a reasonable upgrade pipeline to keep them up to date with the latest security patches.

A common philosophy of implementing “whatever works” when it comes to developing embedded devices is now showing some consequences as we are facing a world where devices are constructed with software containing well-known security issues that are dangerous both for the users and the Internet as a whole. The issue is so common that quite often, attacks on IoT/embedded devices make the headlines on top media sites.
Moreover, a lot of these systems are operating in critical and/or strategically important roles, and it is not uncommon that their performance can have an impact on industrial networks and national security matters. [7] [8] [9] [10]

The resources allocated to the firmware constitute only a minimal part of the budget for each device, and of this amount, a smaller part is intended to secure the firmware.

Although progress has been made in strengthening firmware security in recent years, there is still no definitive solution for its critical condition. Taking into consideration that the number of attacks increases substantially every year, the need for a solution is clear. [11]

Exein Core was designed to address and fix this situation by providing the tools

needed to produce hardened devices with anomaly detection capabilities that can also be monitored with ease.

3. DESCRIPTION OF THE EXEIN CORE

3.1. DESCRIPTION OF THE EXEIN CORE

Exein Core's goal is to protect the target system from undesirable behavior, introducing the self-protecting and remote-monitoring set of tools into the embedded systems arena. In order to achieve this, the solution needs to extract and analyze the system's behavior by enumerating its events. This is carried out in three steps (or main functions):

- Collecting events at OS level (LSM Exein)
- Providing a mean of communication between the kernel space implementation and the user space applications (Exein interface)Real device: Raspberry PI bcm2709 (Raspberry PI 3+ in arm32 mode)
- Analyzing the events using machine learning algorithms (MLE Player)

Derived from this functional division, Exein Core has three main components: a Linux Security Module (LSM Exein), a Linux Kernel Module (Exein interface) and a Machine Learning Engine (MLE Player).

As can be seen in Figure 1, each executable that's meant to be tracked by Exein is labeled with a unique tag. These tags are added to the executables during the firmware build by inserting a new section within the ELF header containing the unique value.

Each tag will then be used by the Exein kernel modules to keep record of which ML Model is responsible for tracking each group of processes (parent and child processes spawned or forked by the first tagged binary).

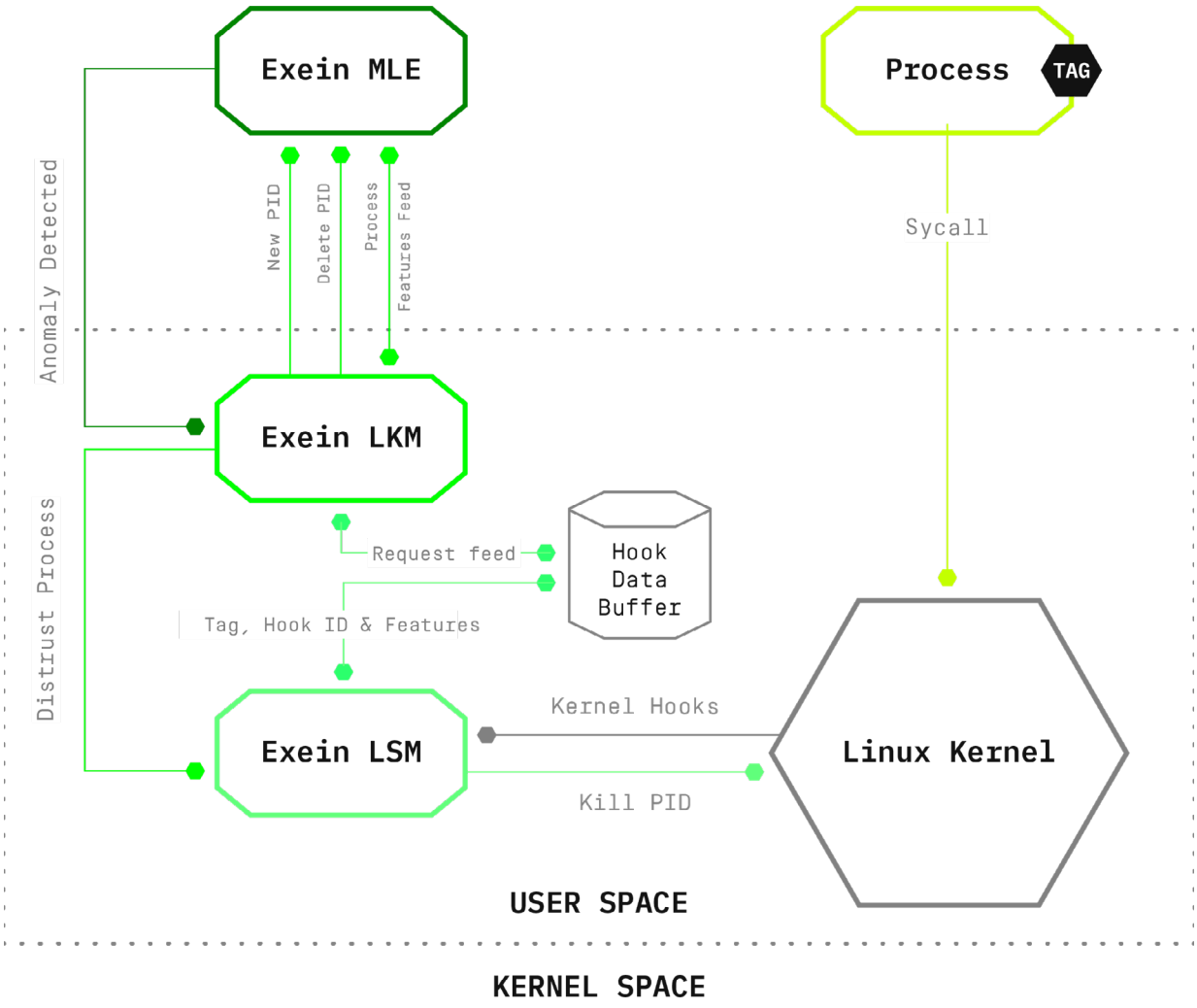

Figure 1: Representation of the Exein Core solution tracking a process.

One machine-learning model is created and subsequently used to track each individual tag. As the tagged processes issue system calls to the Linux Kernel, Exein extracts rich features from kernel hooks and sends them to the machine learning models in order to identify anomalous behavior and — if a certain threshold is passed — block the process actions.

3.2. THE EXEIN MACHINE LEARNING ENGINE (MLE)

The MLE is a user-space process responsible for deciding whether the behavior of a device is to be considered normal or, instead, if the device appears to be under attack.

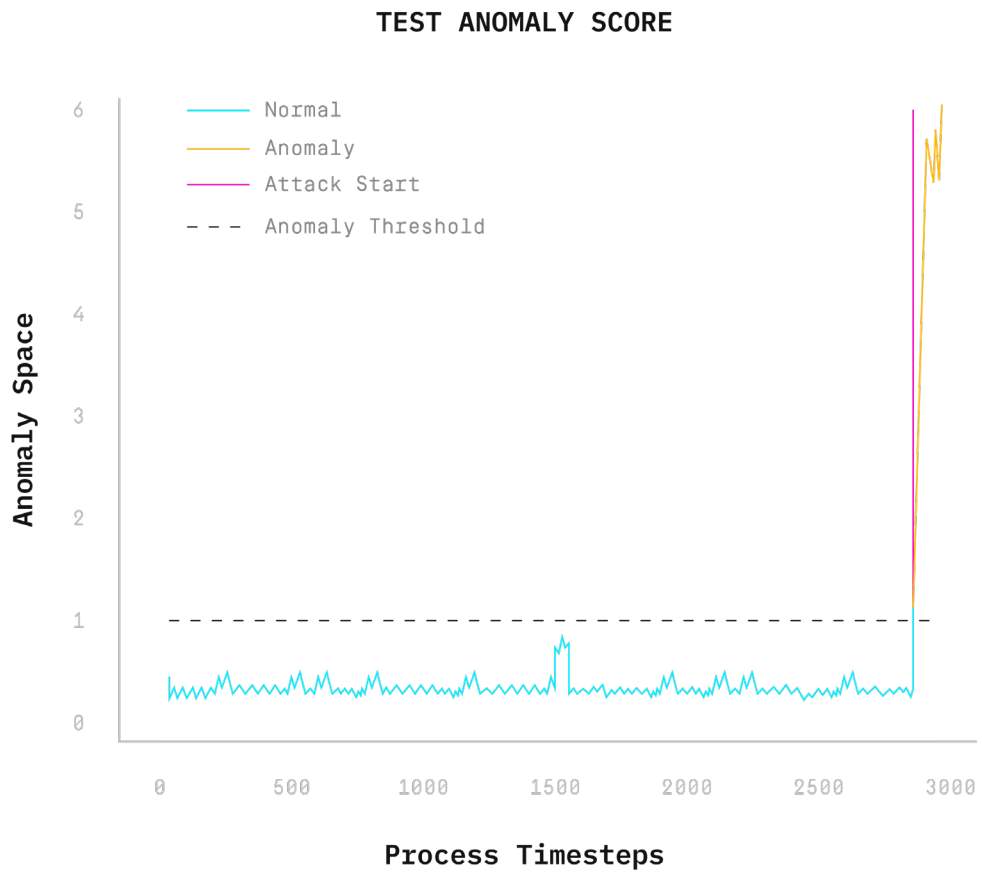

Figure 2: Example of anomaly score responding to a synthetic attack.

In order to accomplish this task, the Exein MLE employs a deep Convolutional Neural Network that studies the behavior of the device processes at kernel level during the training phase, which consists in learning the expected behavior of a variable using a “clean”, normal dataset in a semi-supervised fashion.

Once the training phase is completed, the model trained on the clean dataset can be used as a tool for detecting anomalies in new data that has not been labeled a-priori (i.e. it might or might not contain anomalies) by predicting the future evolution of the device behavior and comparing the model predictions with the actual observations. If these match, then the device is operating under the learned normal behavior; if, otherwise, the new observations diverge significantly from the model’s predictions, then a change of behavior has occurred and the new behavior can be identified as anomalous with respect to the learned one, i.e. the one that is considered to be normal.

We refer to the discrepancy between the machine learning models' predictions for a process and its actual observations as the process "anomaly score". During real time execution, the Exein MLE Player keeps track of the anomaly score of each monitored process.

As shown in Figure 2, when the anomaly score for a given process exceeds a certain threshold, the process is considered to be malicious, and the LSM is informed about it so that it can respond accordingly.

3.3. THE EXEIN LINUX SECURITY MODULE (LSM)

This module of the Exein Core interacts in a direct way with the Linux Kernel. It receives hook calls from the processes being tracked as they issue system calls to the kernel and extracts relevant data given the context of every received hook.

The contextual data available on each case depends on the type of hook being executed but it can range from file descriptors, names and paths, inode attributes, details of memory regions, capabilities and processes attributes to users, groups and effective permissions being used when accessing a resource.

After being extracted, the data is delivered to the MLE through an interface with the third main component: The Exein Linux Kernel Module (or LKM).

There are hundreds of different security hooks that are provided by the Linux Kernel and could be tracked by the Exein LSM implementation.

As the amount, type, and signature of the hooks vary between kernel versions we developed tools to keep track of these changes and automate the configuration of the LSM build for several kernel versions.

In case the MLE detects a process trying to do something unexpected, the LSM will receive a message with this information and will enforce a policy over the process, thus eliminating the threat.

3.4. THE EXEIN LINUX KERNEL MODULE (LKM)

This module plays the role of interfacing between the LSM and the MLE, which are running at different execution levels (kernel and user space respectively).
It provides a communication channel carried over netlink protocol [12][13] and provides some debug functionalities as well.

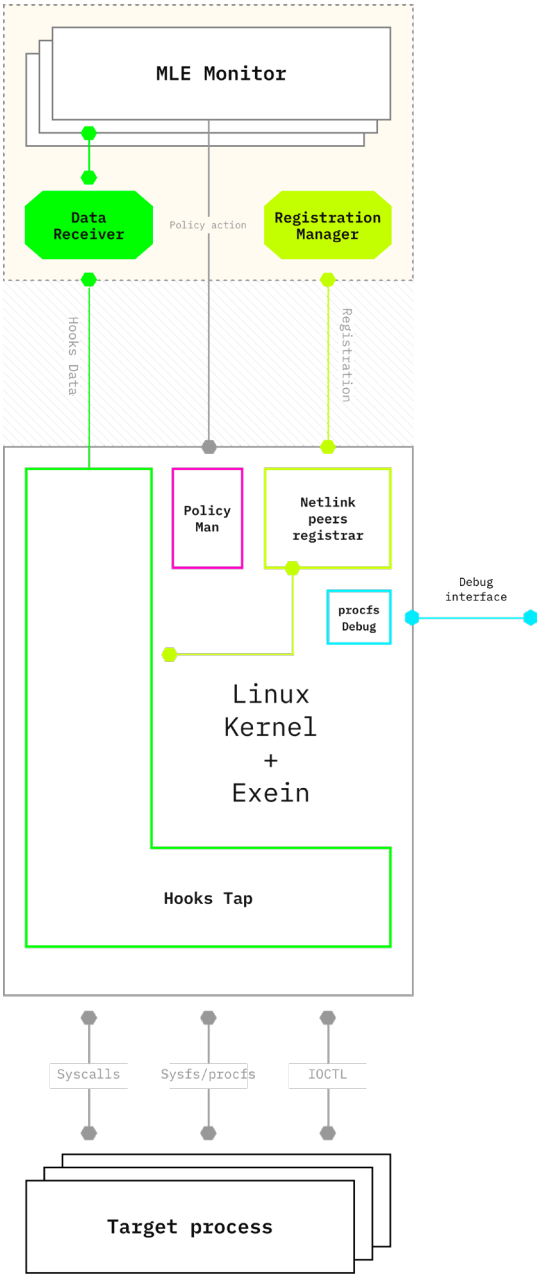

Figure 3: Representation of the kernel “live” mode and MLE player

It supports two modes of operation, the first, named "**live**" is meant to be used on high-end devices and to collect data for training a new ML model. Its main feature is to export in a "live" way all the data produced by a target process. The second mode of operation is named "**snapshot**", and it is the default mode of operation, meant to run on the low-end devices. It minimizes Kernel - user-space communications by exporting only the data that can be actually analyzed, keeping the current image of each monitored process within the kernel address space.

There are currently 4 defined methods for the "**live**" mode of operation:

I. REGISTRATION

Is issued by the MLE with the purpose of registering itself as feed receiver for a particular tag. The MLE needs to provide the tag and a proof of trust that's derived from the current build key.

- Request format: [Key, REGISTER_ID=1, Tag, DC]
- Response format: [Answer, Tag]

II. KEEP-ALIVE

In order for the kernel modules to keep a consistent track of the currently running MLEs, each MLE is expected to issue a keep-alive message periodically.

- Request format: [Key, KEEPALIVE_ID=2, Tag, DC]
- Response format: no answer is expected

III. FEED

This is the request containing the actual data extracted by the LSM and sent to the corresponding MLE to make its prediction.

- Request format: [Key, FEED_ID=3, Tag, Hook data]
- Response format: NONE

IV. BLOCK

This request is issued by the MLE to inform the LSM that it has detected an anomaly and that the process needs to be stopped.

- Request format: [Key, BLOCK_ID=4, DC, PID]
- Response format: [Answer, PID]

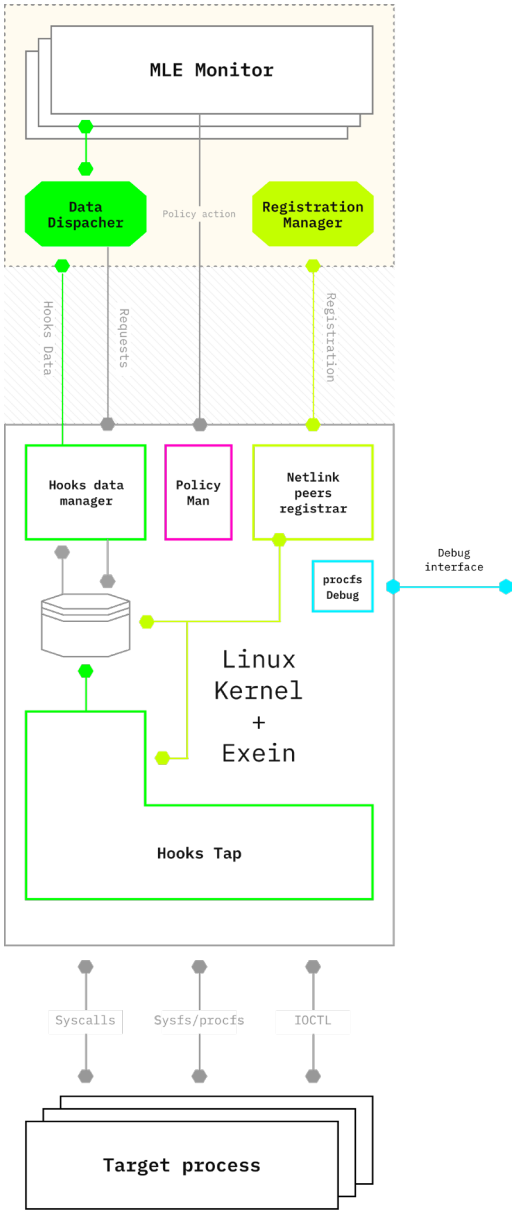

Figure 4: Representation of the kernel “snapshot” mode and MLE player

For the “**snapshot**” mode of operation, currently implemented methods are as follows:

I. REGISTRATION

It is issued by the MLE with the purpose of registering itself as feed receiver for a particular tag. The MLE needs to provide the tag and a proof of trust that’s derived from the current build key.

- Request format: [Key, REGISTER_ID=1, Tag, DC]
- Response format: [Answer, Tag]

II. KEEP-ALIVE

In order for the kernel modules to keep a consistent track of the currently running MLEs, each MLE is expected to issue a keep-alive message periodically.

- Request format: [Key, KEEPALIVE_ID=2, Tag, DC]
- Response format: no answer is expected

III. BEHAVIOR IMAGE

This method is invoked by the MLE Player each time it is ready to infer the behavior for a specific PID.

- Request format: [Key, KEEPALIVE_ID=5, Tag, PID]
- Response format: [Answer, Behavior image data]

IV. BLOCK

This request is issued by the MLE to inform the LSM that it has detected an anomaly and that the process needs to be stopped.

- Request format: [Key, BLOCK_ID=4, DC, PID]
- Response format: [Answer, PID]

V. ADD PROCESS

This method is invoked by the kernel to inform the user-space process that a new PID exists for the TAG the MLE Player is registered to.

- Request format: [Key, NEW_PID=6, TAG, PID]
- Response format: no answer is expected

VI. DELETE PROCESS

This method is invoked by the kernel to inform the user-space process that a PID that existed for the TAG the MLE player is registered to, has been terminated.

- Request format: [Key, DEL_PID=7, TAG, PID]
- Response format: no answer is expected

3.5. THE LIBEXNL LIBRARY

The libexnl library was implemented to ease the development of the Exein MLE and solve the synchronization issues that arise when dealing with communications

between kernel-space and user-space applications.

It provides a simple interface between the MLE and the Exein's kernel interface module, encapsulating the details when it comes to the management of the netlink protocol and the different types of requests that can be issued to and from the MLE. It also takes care of complex data synchronization aspects by providing callback functions for important events and mutexes. The library exposes a simple API that lets you instantiate a multithread agent and provides the current data feed.

It's important to note that because under certain circumstances the MLE may be slower than the actual data acquisition process, this data must be managed so that each time the MLE requests for a sample, it receives the most recent one. The library accomplishes this task by using a circular buffer that is accessed linearly using the provided APIs.

4. SCOPE AND EFFECTIVENESS

Exein LSM allows users to guarantee classes of processes behave in the way they were designed to. For each class of processes, an MLE instance running a specialized model for the class watches all the processes operation. Deviations from the intended behavior result in alarms and policies to be enforced.

Exein LSM's approach results in being very effective when all the external communicating processes are watched. In these cases, attacks such as buffer overflows, or even misconfigurations leading the process acting in a way different from what the model learned during its training phase, result in the MLE stored policies activation, preventing the process from continuing its current misbehaving thread.

Exein is a powerful tool that ensures processes behave in the way they're intended to, but its scope isn't infinite, and its effectiveness isn't universal. The following are some points to take into consideration when evaluating Exein as a solution.

4.1. PERFORMANCE IMPACT

In an ideal world, Exein could take care of the whole system, making it 100% secure by watching for all the processes running on it. In reality, this would impose a huge toll on performance as the computational power required to watch for all the running processes on a system is significant and embedded devices are not built with spare CPU power in mind.

In order to solve this issue, we decided to go with a fair trade-off configuring Exein to monitor only the threads that expose any sort of communication channel to the outside world (the network).

The logic behind this decision is that if an external threat wants to get access to the device through the network, it has no other way in but through abusing or exploiting one of those exposed services.

Depending on the firmware's composition, the task of identifying the processes with access to external communication can be non-trivial. Firmware is often built using several third-party components with functionalities that are sometimes not well understood by the firmware developers themselves. This leads to a potential situation of a firmware running processes that expose services to the outside without the developers being aware of it.

4.2. PHYSICAL ACCESS AND BOOT SEQUENCE ATTACKS

The second thing to take into consideration is that, as is the case with any other Linux Kernel Module, Exein can protect the device only after its start and boot sequence has finished and the kernel modules have been started by the kernel.

This solution is not meant to defend the device from someone who has physical access to the device or who can abuse the boot process at any point before the Exein module is started. If attackers can gain control before Exein starts, they have basically beaten you to the punch.

4.3. CONCLUSION

Given the facts that tracking every process is unfeasible and both physical and boot attacks are out of the scope of the solution, for Exein Core to effectively protect the system it must be coupled with tools that take care of all the remaining aspects.

This would give space to the notion of a framework composed of Exein Core, tools for restricting the permissions of the processes not being monitored, and tools that provide a safe boot mechanism.

5. CONCLUDING REMARKS

As detailed above, have presented Exein Core, a novel host-based, run-time anomaly detection mechanism for Linux-based embedded systems.

We succeeded in training Exein Core in order to identify attacks on a commonly used web server running on one of the most used Linux distributions for embedded systems. The solution was tested on multiple environments, both emulated and real ones and on two of the most popular architectures.

Our ongoing research aims to demonstrate the advantages of the Exein Core approach over traditional solutions, to produce a second-stage solution (the Exein Framework) that will expand the scope and reliability of the current one by covering the aspects discussed in section 4, and to produce models for other types of processes and environments.

REFERENCES

[1] “Linux Security Modules: General Security Hooks for Linux”
Smalley, Fraser, Vance
https://www.kernel.org/doc/html/latest/security/lsm.html

[2] Linux Security Module Framework
Wright, Cowan, Morris, Smalley, Kroah-Hartman. 2002

[3] OpenWrt: Wireless Freedom
https://openwrt.org/

[4] OpenWrt package page for the uHTTPd webserver
https://openwrt.org/docs/guide-user/services/webserver/http.uhttpd

[5] Stratosphere Laboratory. A labeled dataset with malicious and benign IoT network traffic. January 22th 2020
Agustin Parmisano, Sebastian Garcia, Maria Jose Erquiaga
https://www.stratosphereips.org/datasets-iot23

[6] OWASP Internet of Things Project
https://owasp.org/www-project-internet-of-things/

[7] Understanding the Mirai Botnet
Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, Yi Zhou. 2017

[8] Article from Forbes site titled “Criminals Hacked A Fish Tank To Steal Data From A Casino”
Lee Mathews. 2017
https://www.forbes.com/sites/leemathews/2017/07/27/criminals-hacked-a-fish-tank-to-steal-data-from-a-casino/

[9] “HACKING IoT: A Case Study on Baby Monitor Exposures and Vulnerabilities” Mark Stanislav, Tod Beardsley. 2015 https://www.rapid7.com/globalassets/external/docs/Hacking-IoT-A-Case-Study-on-Baby-Monitor-Exposures-and-Vulnerabilities.pdf

[10] “The 2018 SANS Industrial IoT Security Survey: Shaping IIoT Security Concerns” Barbara Filkins. 2018

[11] F-Secure “2019 Attack Landscape H1 2019” https://blog-assets.f-secure.com/wp-content/uploads/2019/09/12093807/2019_attack_landscape_report.pdf

EXEIN S.P.A.
VAT N.IT14881441001

PIAZZALE FLAMINIO 19
00196 ROME, ITALY

INFO@EXEIN.IO
WWW.EXEIN.IO