Gassais *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2020) 9:62
https://doi.org/10.1186/s13677-020-00206-6

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Multi-level host-based intrusion detection system for Internet of things

Robin Gassais[1], Naser Ezzati-Jivan[2*], Jose M. Fernandez[1], Daniel Aloise[1] and Michel R. Dagenais[1]

Abstract

The growth of the Internet of things (IoT) has ushered in a new area of inter-connectivity and innovation in the home. Many devices, once separate, can now be interacted with remotely, improving efficiency and organization. This, however, comes at the cost of rising security vulnerabilities. Vendors are competing to create and release quickly innovative connected objects, without focusing on the security issues. As a consequence, attacks involving smart devices, or targeting them, are proliferating, creating threats to user's privacy and even their physical security. Additionally, the heterogeneous technologies involved in IoT make attempts to develop protection on smart devices much harder. Most of the intrusion detection systems developed for those platforms are based on network activity. However, on many systems, intrusions cannot easily or reliably be detected from network traces. We propose a novel host-based automated framework for intrusion detection. Our work combines user space and kernel space information and machine learning techniques to detect various kinds of intrusions in smart devices. Our solution use tracing techniques to automatically get devices behavior, process this data into numeric arrays to train several machine learning algorithms, and raise alerts whenever an intrusion is found. We implemented several machine learning algorithms, including deep learning ones, to achieve high detection capabilities, while adding little overhead on the monitored devices. We tested our solution within a realistic home automation system with actual threats.

Keywords: Host-based intrusion detection system, Internet of things, Anomaly detection, Machine learning, Execution tracing

Introduction

Cisco estimates approximately 50 billion smart devices connected to the Internet in 2020, or 6.58 things per inhabitant [1]. This refers to the connection of various embedded devices such as sensors, actuators, and vehicles able to interact with each other [2]. While this growth induces the production of innovative objects, like connected speakers able to respond to a verbal request or order products, it creates a huge security threat for consumers and companies, as attackers can gain access to devices within a home or office. In the race to develop innovative and profitable technology, security concerns are often secondary. The targeting of insecure devices can have far-reaching consequences, like the Mirai botnet infecting poorly secured devices, using a default password, to launch one of the most powerful DDoS campaigns ever seen in 2016 [3] against the Dyn DNS server. This cyber-attack succeeded in making unreachable for many hours some of the most popular websites on the American west coast. Most of the time, these devices have limited resources, and there is a huge heterogeneity in the connected device software, including the operating system, and the network protocols. For those reasons, the security community keeps raising alarms on the vulnerability of IoT devices, as did OWASP with its TOP 10 IoT vulnerability list [4].

To improve the security of IoT devices, some work has been done to detect intrusions on smart devices. ZarpelÃčo et al. [5] highlighted the need to install host-based intrusion detection systems (HIDS) since they can monitor more information about the connected device,

*Correspondence: n.ezzati@polymtl.ca
[2]Brock University, St. Catharines, Ontario, L2S 3A1, Canada
Full list of author information is available at the end of the article

and therefore can help to detect attacks that could not be identified with network information. Furthermore, the author explains that traditional HIDS are not effective for IoT. All of the intrusion detection systems (IDS) presented in this article are network-based, while we could only find a few host-based intrusion detection systems for IoT.

Many host-based solutions have been developed for traditional systems such as OSSEC [6] or Sagan [7], which provide multi-level monitoring of systems, with alerts correlation or active response. In addition, work by that of Bezerra et al. [8] and Breitenbacher et al. [9] are able to offer lightweight solutions for intruder detection while maintaining a high degree of accuracy. However, even if those solutions can be lightweight, for instance when using the OSSEC agent, the limited resources of smart devices cannot let those tools process the collected pieces of information directly on the device. Moreover, those tools may not be compatible with smart objects hardware such as ARM CPUs. As a consequence, those solutions cannot detect intrusions, based on host information on smart devices. Our goals are therefore to determine if a lightweight and efficient HIDS for smart home devices can be developed and to study how machine learning algorithms could be used with tracing data to achieve good detection capabilities.

In this paper, we propose a complete framework for the dynamic automated analysis of IoT. We collect multi-level information on the monitored devices, with tracing techniques, and stream this data to an analysis engine, either located on a device inside the network system, on a dedicated machine inside the home, or in the cloud. The analysis engine can use several machine learning algorithms to detect anomalies on device behavior. Then, when appropriate, alerts are raised by the analysis system with the name of the device and the kind of threat that has been detected.

We have several contributions within this article. First, we provide a complete tracing architecture where a user only has to specify what tracepoints (s)he wants to monitor and how often (s)he wants to receive this information. Our solution then activates tools that will send snapshots of the trace events at the user-specified interval. Then, we provide an automatic tool for trace analysis, which can extract features from events and process them to become usable with open-sourced machine learning libraries. We also developed a convenient way to label each event. Finally, we optimized and compared several machine learning, and deep learning algorithms to get the best detection capabilities.

The paper is structured as followed: "Related work" discusses the proposed works about intrusion detection, smart homes and tracing that have been proposed. "Proposed solution" focuses on the solution proposed, with the framework architecture and a detailed review of each of its components. In "Use case", we explain our experimental set-up and the proposed implementation. "Evaluation" sections highlights and discusses some results about the efficiency and the overhead introduced by our solution. Finally, "Conclusion and future work" section concludes on the whole framework developed.

Related work

Smart home and associated threats

We will refer to a smart home as a residence where smart devices such as sensors and actuators are connected to provide monitoring and/or new entertainment capabilities to its inhabitants. The typical architecture of such a residence is described in Fig. 1, where all the smart devices are part of the IoT Network, managed by a home controller that is the gateway with other Internet-based devices. This device is responsible to link the Z-Wave or the ZigBee network to the traditional IP network. Some authors like Cheng et al. [10] distinguished the IoT networks according to the network protocol used by the devices or according to the primary purpose of the device (e.g., healthcare, entertainment or surveillance). However, even with the supporting infrastructure, the devices often rely on a home controller as a gateway to the traditional network.

Whether it is for personal comfort, with the possibility to manage the electricity consumption more efficiently, or for healthcare, Gubbi [11] explains that homes will be the first area where connected devices will be deployed, before entering companies and more large-scale deployments. However, the article highlights that the security of such devices and networks is a crucial challenge, both at the device scale, with software and protocols, and at the cloud scale, with data protection for privacy and identity management. The work of Hui et al. [12] comes to the same conclusion regarding security and privacy. They are important challenges to achieve consumer adoption of smart homes. Moreover, the author identifies different threats, such as personal data theft like photos, videos, and documents, or the spying on personal habits that could be revealed to third parties. Moreover, smart devices are usually produced in large quantities, and vulnerabilities are often found after their release. One of the major issues is that those devices are often not updated, and if available the update process should be secured [13].

As smart devices are more widespread over homes, attackers are increasingly targeting such devices. The Mirai attack involved more than 200,000 infected devices [14]. Saxena et al. [15] focuses on several attacks that strike smart devices, while Sikder et al. [16] highlights the threats on sensors inside such a network. While Babar et al. [17] proposed a taxonomy for IoT attacks, we decided to summarize the main ideas from those article as follow :

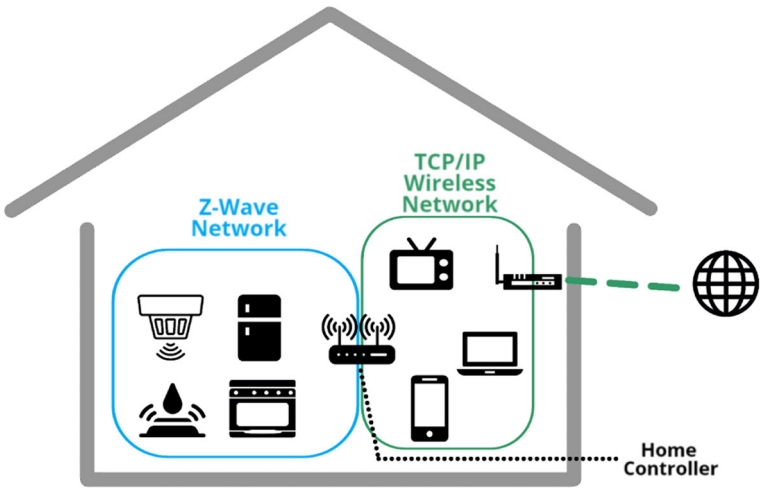

Fig. 1 Smart home architecture

- Data exfiltration: either eavesdropping on private life, information theft, or inference, the attacker's goal is to obtain some confidential pieces of information that (s)he could benefit from;
- Data alteration: this may be achieved via the injection of false data or command to some device, via a replay or person-in-the-middle attack. The attacker's goal here is to change a systems behavior in order to abuse it.
- Denial of service: the intrusion's purpose is to make a system unreachable for the user. The target system can be a smart device that the intruder will infect, like the Brickerbot malware [18] which bricks the device by wiping its files and corrupting its memory. Alternatively, the attacker's target may be a remote system. This was the case for the infamous Mirai botnet, responsible for the largest DDoS attack at the time, on remote servers such as OVH and DYN DNS [19].
- Intrusion inside a network: connected devices, especially home controllers, may be used to enter a traditional network and then allow the attacker to bypass some security protections such as an external firewall.
- Physical security: smart devices could threaten its user physical security. For instance, if an intruder can control a connected oven and a smart gas valve, (s)he might be able to make the oven explode and therefore hurt objects and persons around. Furthermore, with connected pacemakers which are proved to be vulnerable, or connected cars being remotely controlled, the threats associated with cyber-physical systems are rising.

Intrusion detection in IoT context

Intrusion detection system

A distinction is made between network-based IDS (NIDS) and host-based IDS (HIDS), as explained by Sabahi et al. [20], even if some work has been done using both approaches with a hybrid IDS. However, the author highlighted that regardless of the type of data collected, the detection can be obtained with two techniques: misuse detection or anomaly detection. The first type of detection relies on an expert system, which can be based on the known threats signature, with rules about the collected data or on state transitions in the system, while the second monitors the system behavior to detect anomalies and therefore potential intrusions. Both methods have benefits, with misuse detection being accurate and working well on known threats, according to ZarpelÃčo et al. [5], but being unable to detect new attacks, unlike anomaly detection. However, anomaly detection requires to characterize and identify the normal behavior of the system, which can be difficult to achieve.

Intrusion detection systems have been studied for a long time, however, as highlighted by ZarpelÃčo et al. [5], traditional IDS failed at protecting connected device for three main reasons: the devices have limited resources, often insufficient to run a traditional agent, smart devices usually work under a mesh network where they can forward packets while being an endpoint, and there is considerable heterogeneity in the technologies and network protocols used in IoT. Furthermore, little work has been done with HIDS, as ZarpelÃčo et al. [5] only studied NIDS. However, Nobakht developed a whole framework to deploy a host-based intrusion detection system on software-defined networks (SDN) [21], but the researcher still relies on net-

work information to detect intrusions. Moreover, SDNs are not yet deployed in homes, which makes this solution inconvenient for now.

While tracing has been used for a long time to detect intrusions, especially system anomalies, either in traditional or embedded systems [22], Eskandari et al. [23] explained that intrusions can be detected via tracing from incoherent events (for instance, the execution of /bin/bash on systems that are not supposed to) or from sequences of events that are unusual.

Anomaly detection

Chandola et al. [24] defined anomaly detection as a set of techniques that aim at finding data that does not match the expected behavior. According to the author, this problem can occur in many fields, including fraud detection, fault detection, and intrusion detection. Furthermore, the article highlights several techniques that have been used for years to find anomalies, including classification, clustering, and statistics.

Murtaza et al. [25] provide several examples of tracing to detect anomalies in software. In doing this they highlighted the need to use kernel traces with classification techniques to improve the accuracy of anomaly detection. Other work has been proposed to combine tracing events and anomaly detection, such as another work by Murtaza et al. [26] in a paper where he and his co-authors compared three anomaly detection techniques that only rely on syscalls to detect intrusions. They used three different machine learning algorithms, but without studying the overhead introduced by their solution or trying more recent deep neural network techniques.

It is very important to distinguish classification, clustering, and statistics when it comes to detecting anomalies on a system.

- Classification aims at associating a class to each input element fed to the algorithm. For instance, when focusing on intrusion detection, one may want to know if the input is an intrusion or belongs to the normal system behavior. One may also want to detect if this is a network intrusion or a software intrusion. Classification can be achieved for each input event or for the sequence of input data. When focusing on each input, one might consider using machine learning algorithms, like those explained later in the article. When focusing on sequences, Chandola et al. [27] distinguished three categories of techniques: kernel-based techniques, window-based technique, and Markovian techniques (including Hidden Markov Models). In the field of Machine Learning, this is an example of supervised learning techniques.
- Clustering aims to find some relations between input data to create groups of elements. For instance, while tracing a system, a user could obtain network events, syscall events, and scheduler events. It is very likely that network events look similar, quite different from syscall events. In that case, clustering could be used to automatically split the input data into three sets: network events, syscall events, and scheduler events. Several techniques have been studied, and Jain distinguished hierarchical algorithms, partitional algorithms, and some other techniques [28]. The user may not know how many different groups should be used to split the data. Those techniques belong to the second category, unsupervised Machine Learning.
- Statistics could be used to detect anomalies if the user has a good knowledge of the system behavior and can easily quantify this behavior with figures. For instance, when the goal is to know whether there is an anomaly on a machine, one can monitor some metrics such as the CPU usage, the network latency, or the memory usage. When detecting that one of those metrics is above an arbitrary threshold, one can assume there is an anomaly on the system.

Tracing and debugging embedded device

Several techniques can be used to collect data about a system, such as tracing, debugging, or profiling. Tracing is a set of techniques that can provide a user with detailed information on a system, either hardware or software. Tracing is different from profiling, as detailed in [29]. While profiling aims at getting general statistics on a system, tracing gets a precise and timestamped log of the system execution when an event occurs. Therefore, tracing can be used to investigate short duration episodes, while profiling only detects changes that affect the system for a longer period of time. Tracepoints are small pieces of code for collecting information, that are available in instrumented kernels, such as the Linux kernel, or can be added in software. When the runtime hits a tracepoint, the tracer is called, and will record an event which contains information about the state of the system at the time the tracepoint was hit.

Several tracers exist with different functionality. One of the most effective for Linux systems is LTTng [30], which can trace both user space and kernel space [31]. This tool can help users to get detailed information about a system. It is very flexible, the user can choose which tracepoints (s)he wants to monitor and produce a list of accurate timestamped events in a binary format called CTF (Common Trace Format). In addition, Desnoyers et al. explained how easy it was to port LTTng to new embedded architectures [32]. The main requirement for using LTTng is to have a Linux system, which is the case for most smart devices. Nonetheless, some tracers such as Barectf can be run on bare-metal systems [33]. This tracer has the advantage of producing CTF traces, and Bertauld

explained how it is possible to correlate Barectf traces and LTTng ones [34]. With these tools, it is possible to efficiently trace most IoT devices.

Proposed solution

Architecture

To achieve the best intrusion detection, according to metrics that are specified later in the article, we propose a whole infrastructure, whose architecture is shown in Fig. 2. It is composed of several sensors and actuators and an analysis system. The smart devices are running the tracer, while the analysis device aggregates the collected traces from devices, to detect anomalies and raise an alert when an intrusion is found. In addition, the analysis engine can be used to automatically correlate raised alerts and take action to prevent the intrusion. According to the definition of ZarpelÃčo et al. [5], this host-based intrusion detection system has a hybrid placement strategy, which enables us to use powerful analysis techniques to monitor various devices, while introducing very little overhead on each device. This architecture can be quickly integrated into a home automation system, since it only requires the use of a tracer on the monitored devices, and the use of an analysis system that can run Python. The analysis system can be another device or a server (inside the home or in the cloud). The heart of the solution is the analysis engine, which is composed of four main components: the trace aggregation engine, the data processing engine, the automated analysis engine, and the alert raising engine.

Trace collection

We rely on traces to detect intrusions. For more precision, we get multi-level information from the monitored systems, such as kernel-space and user-space data, hardware information, and even network input. Furthermore, our tracer imparts minimal overhead, which is a major requirement for working in the IoT field. For a Linux-based smart device, the kernel is already instrumented (since version 2.6.32), and several powerful tracers are available. However, for devices running custom firmware, or for bare-metal systems, tracing can be done via barectf, a minimal tracer that is easy to port to such environments but that produces compatible traces in the Common Trace Format (CTF). Consequently, since our HIDS only relies on tracing information, it could detect intrusions in most of the connected devices.

The LTTng tracer is known for its very low overhead [26]. Moreover, it provides detailed information on the systems, thanks to the huge number of tracepoints available in the Linux kernel. It can quickly send trace data through the network in an optimized binary format called CTF. It is easy to enable or disable tracepoints. In recent LTTng releases, one can even apply filters to tracepoints, in order to only trace some events in specific interesting cases, optimizing the trace size. This enables the user to have a precise control over data collected from the system. In addition, LTTng is able to get multi-level information on the system, such as syscalls, scheduler events, user-space events, network events, and some hardware information, which helps to detect intrusions by monitoring the whole system.

Furthermore, LTTng can collect data through two modes: live monitoring, where the tracer sends trace data to a remote server while it is being collected, or snapshot mode, where the tracer sends a snapshot of the current content of the trace buffer. Our solution can work with

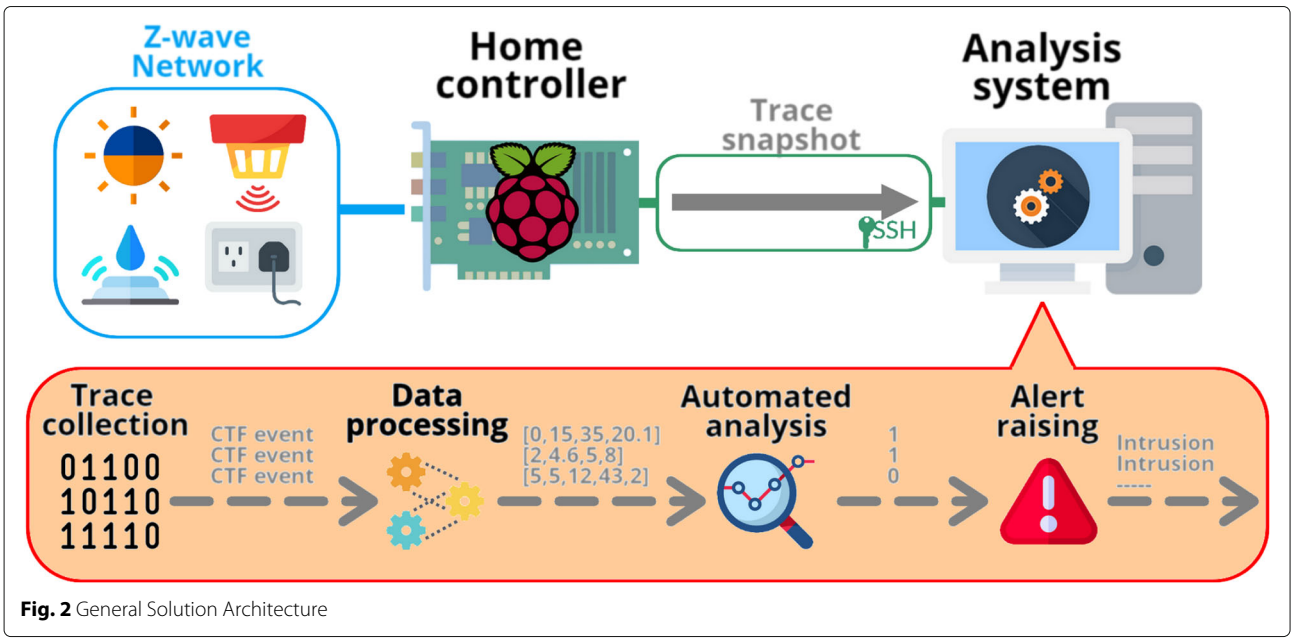

Fig. 2 General Solution Architecture

either of those two modes, but many devices such as sensors usually send information at regular intervals, not continuously. For instance, this can be the case with a light sensor, sending measurements of the room luminosity twice a second. This explains why we decided to use snapshot recording, since it is more suitable for sensors or actuators that only open communications at a certain frequency. Nonetheless, the proposed solution can be used with the live mode as well.

To simplify trace collection, we developed a tool able to generate an activation file from a configuration file. The activation file collects the specified events on the traced device and sends snapshots through an SSH tunnel to the analysis system, based on a user-specified frequency. This enables adapting for each device the information we want to collect. This allows us to optimize the intrusion detection in that device, as we can avoid collecting irrelevant information and thus reduce resource usage (bandwidth, cpu). The flexibility of our tool, and of the LTTng tracer, can easily accommodate every connected home device that runs a Linux-based kernel. Our tool on the analysis system can in parallel receive snapshots from devices and perform the intrusion detection analysis, in order to detect intrusions as quickly as possible.

Data processing

The next step is to be able to feed machine learning algorithms, expecting vectors of metrics as input, with our collected binary CTF traces. The process of extracting numerical data for use in machine learning is known as representation learning, as explained by Bengio et al. [35]. For this purpose, we developed a complete toolchain (shown in Fig. 3) that extracts features from CTF trace events, merges some features and creates others, and finally converts string values to numeric ones.

To be able to read CTF traces and extract the features, we rely on the Babeltrace API, an open-source tool developed by the same group that produced LTTng. This software can convert binary CTF traces to Python dictionaries containing the event fields. Those will become features for the analysis algorithm, and other information such as statistics about the consumption of the device resources. Some information like the process identifiers (PID) are already numerical, others like the filename in an open syscall must be converted to a numerical identifier. We then have to select the fields that are used to detect intrusions and remove the others, to get a manageable dimensionality for our arrays. This is mandatory for most of the analysis algorithms. Indeed, the curse of dimensionality is a well-known machine learning issue that has to be taken into account when training a model.

The next step is to process extracted data from sensors, as explained by Chandola et al. [24]. To that extent, we need to improve upon our raw events, so that they contain relevant information, useful for the analysis engine. Consequently, we created a layer of abstraction by merging some events, such as entry_syscall_X and exit_syscall_X with a finite state machine (FSM), as proposed by Ezatti-Jivan et al. [36], that can synthesize the information of two events into one and therefore enhance the detection. A user can quickly create new synthetic events with other FSMs, either by adding some short sections of code in the processing source code or via a configuration file. In addition, we also created new features in this step, like the duration of the event, based on the beginning and the end timestamps recorded by the tracer, or the current process name associated with the event, which can be retrieved by following the execution flow from the scheduling events. As a result, we obtain many categorical and non-categorical features that are used to detect intrusions on the device.

Thereafter, a one-hot encoder is used to convert the string values into numerical identifiers. One-hot encoding is a well-used step in data processing, as shown in Fig. 2 of Springenberg et al. [37]. For instance, since the filename feature contained originally the filename string

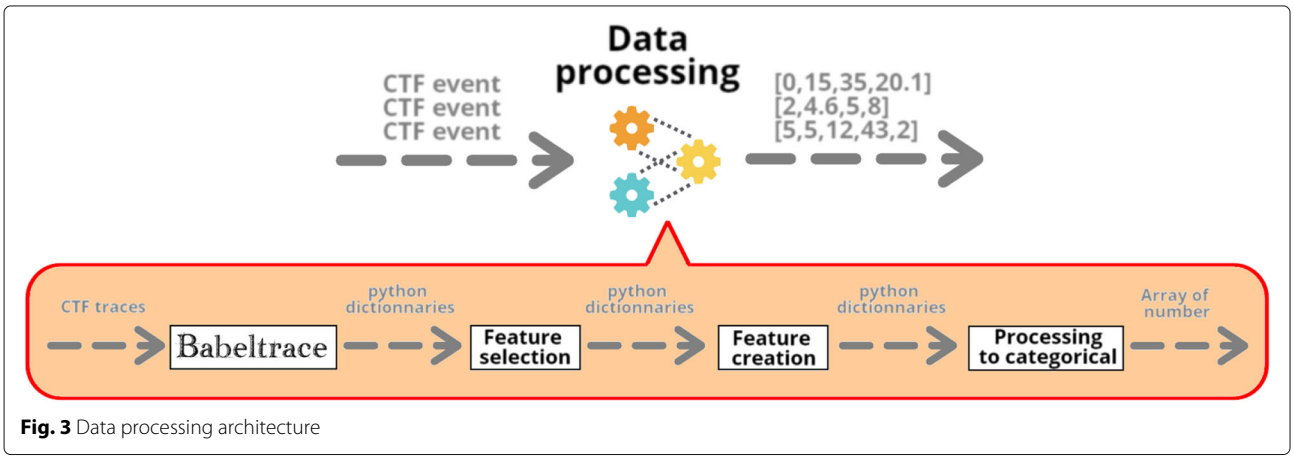

Fig. 3 Data processing architecture

of the associated event, {filename : X}, we have to convert that string into a binary value that the algorithm can process, in this case, is_X. It will get the value 1 if the event is associated with filename X and 0 otherwise. This has the cost of significantly increasing the dimensionality of the array representing the collected events, but this information is crucial for the intrusion detection part. At the end of this step, we have an array which only contains categorical values, which can then feed the machine learning algorithm.

The resulting features are detailed in Table 1

Automated analysis

The final step is to use the collected trace data to automatically detect intrusions on the device. For this, one must know the normal behavior of the connected device to be able to analyze a different behavior. While it might be easy to know the standard behavior of a device, either from the vendor or by manually checking the device behavior, it can require a lot of time and a deep knowledge about the device operation. Therefore, we propose an automated solution that may be efficient with a wide range of devices. This is possible with the use of machine learning techniques, learning the difference between a benign behavior and an Intrusion. Furthermore, it can be very complicated to manually define the rules characterizing a standard behavior, while machine learning will learn by itself the boundary between good behavior and intrusions, even in some sophisticated attacks where the attacker can mimic the standard system behavior [38].

However, machine learning can require more work than specifying some intrusion detection rules. Indeed, any machine learning algorithm has to be fed with enough high-quality data to ensure accurate detection. The supervised learning algorithms also need the output classification of the training data to be able to learn what behavior is good or not. Consequently, we developed a tool able to create a data-set and automatically label it. This is done by tracing the device during two phases: first tracing the device in normal activity, then tracing the device under attack. The collected traces of the first step are then processed as described in the last paragraph, and the information about the events are stored in a database. This includes the network addresses and the ports used, the name of the process that generated the event, the filename associated with an event, and more. The information stored in the database is precise enough to perform the detection at the event scale. Indeed, if the Home Assistant software has to access file /etc/passwd as part of the controller normal behavior, then the information will indicate that the Home Assistant process has accessed this file, with the return value of this operation and its duration.

In the second phase, we collected traces of the device while subjected to attacks. To label the generated traces, we process the recorded events and then check for each event if its information is contained in the database of normal behavior. If so, we label the event as part of the good behavior. If not, we label it as an intrusion. We then store the processed event and the label in a CSV file, which will be the input for our algorithm. Thus, if an attack during this phase reads the /etc/passwd file from a bash script, this will not be found in the database of good behavior, since this file is not accessed in this way during normal operation. As a conclusion, the collected information is sufficiently large to distinguish between similar events in the normal and the intrusion case.

Once the dataset is generated, we can use different machine learning algorithms to find the boundary between the normal behavior and an intrusion. With this pre-processing, the events are suitable for feeding to many machine learning algorithms, including those of the popular open-source libraries Scikit-learn and Keras. Once our models have been trained with the dataset, we can trace the device again and give the collected processed events to the models. They can then detect attempted intrusions on the device.

Table 1 Post processing feature list

Feature	Type	Description
filename	string	Filename manipulated by the syscall
source_port	int	Port of the source network packet
dest_port	int	Port of the destination network packet
p_name	string	Name of the process that created the event
protocol	int	Value representing the used network protocol
parent_comm	string	Parent's process name
child_comm	string	Children's process name
pathname	string	Pathname manipulated by the syscall
ret	int	Value of return of the syscall
saddr	string	Address of the source network packet
daddr	string	Address of the destination network packet
d_timestamp	int	Event duration
a_nomEvent	string	Event name

Alert raising

The final stage of the analysis engine is to raise an alert whenever an anomaly is found, with information about the suspicious event. This is achieved with the display of an alert message on the analysis system. In operation, such an alert would normally be fed to a monitoring system.

Although it has not yet been implemented, the alert engine could correlate the events associated with an intrusion to take actions to stop the intrusion. In the Mirai botnet case, once the alert has been raised, the

analysis system could shut down the infected device and restart it, removing the injected piece of malware. It could also recognize the Mirai infection pattern and alert the user to change the device's passwords to prevent another infection.

Use case

Attacks

To train our model, we decided to use real threats that IoT face today on our home assistant controller. As shown by Homoliak et al. [39], Obfuscation of TCP communication can reduce the chance of detection significantly. It is important that any security system is trained on a variety of intrusion attempts that actively attempt to hide their presence. We thus implemented a Mirai like infection, a scan of the IoT network, a Hail Mary attempt with Metasploit [40], a directory listing and a vulnerability scanning on the controller web interface, a basic ransomware, and a spying tool. Those attacks are described in the next paragraphs.

First, we set up the Mirai botnet using its source code on GitHub, and we studied the traces that have been generated by this infection. Infected Mirai devices try to brute force other devices passwords with a list of default factory passwords. If it succeeds, it then downloads via the *busybox wget* command a malware from the C&C server, changes its name on the device, does a *chmod*, and then runs it. For this reason, because we only wanted to trace the infection part, to be detected by our IDS, we implemented this attack with a modified malware that would just echo "Infected" on the device, once the malware process is created.

The second intrusion was a nmap scan to discover other devices on the traditional TCP/IP network. Indeed, whenever an attacker wants to exploit a network, (s)he first has to survey the network to find potential targets. This is usually done with a network scan via the nmap tool. To be efficient against hacking attempts, our solution has to be able to detect such network activity when it comes from a monitored device.

Then, we used Metasploit [40] to launch several exploits on our target. In a Hail Mary attack, the intruder attempts a variety of common attacks that (s)he believes may succeed. This is not subtle, however it is often effective against unprotected and un-patched systems. Since we patched the system with the latest security fixes and used up-to-date software, the Hail Mary process did not manage to exploit our controller. Nonetheless, the traces left by this attempt can then be used with our models, which will then be able to detect such attempts.

Then, we used two attacks against the web interface of the device. As OWASP highlighted in its Top 10 IoT Vulnerabilities [4], it was the major attack vector: a directory listing done with the DirBuster tool with its standard wordlist, and a vulnerability scanning with Nikto that did not find any vulnerability.

A ransomware [41] attack is the act of encrypting files on the target's system in order to deny access until a ransom is paid. In our case we used a ransomware that encrypts all the files in a specified directory with AES 256 ECB mode. We only traced the exploitation phase, i.e. the encryption phase. This mimics the behavior of other ransomware such as Brickerbot [18] that heavily targeted smart devices last year.

Finally, we developed a spying tool that records all the information sent to the home assistant controller via its API and resends that information to another server through the Internet. It is a specific malware that targets this device, and it does not change the device behavior much, unlike to the other intrusions.

Resulting dataset

As described earlier, we first traced the system while only issuing standard actions. We switched on a light via a Z-Wave command, powered on a smart plug with the controller web panel, activated a motion sensor so that a notification was sent to a remote smartphone and laptop,... Our home automation system is only composed of affordable devices that are described in the next section.

We traced the controller for one hour with a snapshot sent every 30 seconds, except the specific test where we took a snapshot of the controller every second. We then processed the recorded snapshot as described earlier, and created our events database. To reduce the risk of getting an imbalanced dataset, we removed several instances of events that were too frequent in our snapshots. Undersampling the dataset is a common practice in machine learning when dealing with an imbalanced dataset, as explained in Kotsiantis et al. [42], but several other methods exist, as highlighted in the article. As explained by Pendelberry et al. [43], intrusion detection is naturally imbalanced (benign events are far more common than malign ones) and therefore undersampling can introduce bias on naturally imbalanced data sets. However, it also improves the performance of classifiers, as shown by Rani et al. [44], thus resulting in overall greater accuracy.

The second tracing phase was with the device under attack. We took a snapshot of the trace buffers at every second to record every event associated with each attack. To label those events, we compared each with our database. Whenever the information of the event is already in the database, we remove the event. The events that do not match the database are labeled as intrusions.

For both tracing parts, we used several tracepoints, including syscall, network queue event, scheduler event, CPU events and kernel module events. An exhaustive list of the tracepoints used is presented in Table 2. We launched a new tracing session for each attack,

Table 2 Enabled tracepoints list

Syscall	Network	Scheduler	Kernel module	CPU
access,chdir,chmod ,chown				
,chroot,clone,close,connect,				
copy_file_range,creat,delete_module				
,execve,execveat,exit,exit_group				
,faccessat,fallocate,fchdir,fchmod				
,fchmodat,fchown,fchownat,finit_module				
,fork,getcpu,getcwd ,getdents,getdents64				
,getegid,geteuid,getgid,getpgid,getpgrp				
,getpid,getppid,gettid,getuid,kexec_file_load				
,kexec_load,kill,lchown,link,linkat,listen				
,migrate_pages,mkdir,mkdirat,mount				
,mq_unlink,open,openat,open_by_handle_at				
,newstat,pipe,pipe2,pivot_root,read				
,readahead,readlink,readlinkat,readv		sched_process_exit		
,reboot,remap_file_pages,rename	net_dev_queue	sched_switch,	module_load	power_cpu_frequency
,renameat,renameat2,restart_syscall		sched_process_fork		
,rmdir,sched_getparam,setdomainname		sched_process_exec		
,setfsgid,setfsuid,setgid,setgroups				
,sethostname,setitimer,set_mempolicy				
,setns,setpgid,setpriority,setregid				
,setresgid,setresuid,setreuid,setrlimit				
,set_robust_list,setsid,setsockopt				
,set_tid_address,settimeofday,setuid				
,setxattr,shutdown,socket,symlink				
,symlinkat,sysctl,sysfs,sysinfo,syslog				
,tgkill,tkill,umount,unlink,unlinkat,unshare				
,vfork,write				
,writev				

ensuring that there was no event from a previous attack in a recorded snapshot. As a result, we have a dataset composed of 58% of benign events and 42% of intrusion-related events. The repartition of the various events is highlighted in Fig. 4.

Learning methodology

To train our models while avoiding over-fitting, we split our dataset between a training set and test set, with 66% of the data used in the training set and 34% in the test set. Splitting the dataset into training and test sets is a common practice in machine learning, as explained by Hsu et al.[45]. There is, however, no consensus in the field about the optimal repartition. We used a k-fold cross-validation, which consists in splitting the dataset into k sets, and then compute the model learning on k-1 sets and test on the last set. The performance is measured by the average score of some metrics on the computed loop. To optimize the hyper-parameters of our various algorithms, we used a grid search, which consists of testing all the combinations of various hyper-parameters and keeping those yielding the best score for their model. The metrics used to measure the efficiency of our algorithms are described later in this paper.

Classification algorithm

To enhance our detection performance, we implemented several algorithms that can be divided into different categories: supervised learning vs semi-supervised[46] learning or event-based detection vs sequence of events-based detection. Table 3 provides a quick overview of those algorithms. We decided to use both very recent deep neural network techniques and more traditional machine learning algorithms to measure the benefits of using newer

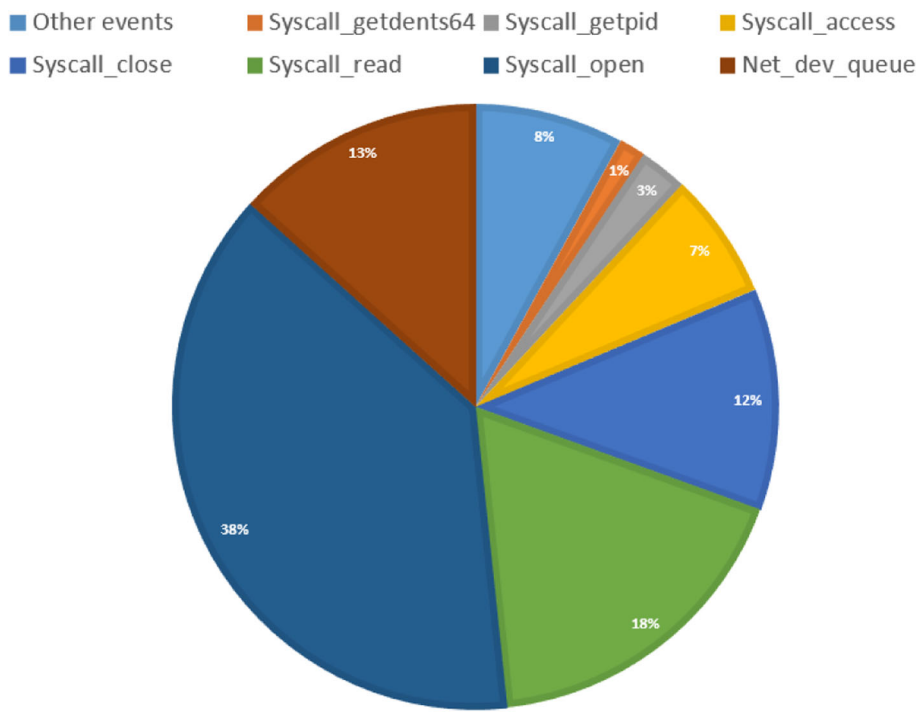

Fig. 4 Dataset event repartition

methods, that are often more resource hungry than older ones. The algorithms we implemented use various methods to detect anomalies :

- Decision Trees (DT) use a graph like structure to classify the dataset by creating nodes from previous nodes that maximize the purity of each node. We use Gini Impurity as a criterion to split the data in each node and maximize the information gain.
- Random forest (RF) is a bagging ensemble method where several decision trees are independently created during training. Each tree is created in a random sub-ensemble of the dataset with a random number of features. Each tree will then vote for a class, and the class with the maximum number of voters will be associated with the input.

Table 3 Machine learning algorithm used

	Event classification	Sequence classification
Supervised learning	Decision Tree	
	Random Forest	
	GBT	
	SVM	
	MLP	
Semi-Supervised learning	OneClass SVM	LSTM

- Gradient Boosted Trees (GBT) is a boosting ensemble method where classifiers are built sequentially to reduce the bias and the variance of the global classifier. We use decision trees as the basics estimators.
- Support Vector Machines (SVM) is a set of techniques that aim at finding the boundary between the different classes by mapping the data into another space. To predict new data, the same mapping is used.
- MultiLayer Perceptron (MLP) is a feed-forward artificial neural network where several layers of neurons are connected that can separate even not linearly separable data.
- One Class SVM is a particular use-case of SVM techniques where the classifier only learns the boundary of one class, here the standard behavior of the system. Then, whenever an event is outside of the determined boundary, it is classified as an anomaly. This technique is able to detect new intrusions since we do not need to train this algorithm with attacks samples. One class SVM is effective on imbalanced data sets, as explored by Devi et al.[47]
- Long Short-Term Memory (LSTM) is a recurrent neural network, meaning that there are some recurrent connections in the network. Thus, recurrent networks have two sources of input: present data and data that has already been through the network. Therefore, the prediction of an entry

event will affect the prediction of the later. Each unit of an LSTM network is composed of an LSTM cell, which has the ability to store, write or read a piece of information, according to the input it receives. Thus, LSTM networks are able to classify a sequence of data.

Evaluation

Set up

To implement our solution, we used several devices that would typically be used in a home automation system. Our home automation controller, the device responsible for the action of actuators according to data received from the sensors, is a Raspberry Pi 3, an ARM based embedded system running Linux. This device has the exact same hardware characteristics as the HomeSeer HomeTroller Zee S2, another home automation controller. Our Raspberry Pi 3 runs Home Assistant, a modified Rasberry Pi Debian distribution with one of the most popular open-source home automation platforms. Home Assistant is compatible with many smart devices and IoT protocols. To get a representative home automation network, we plugged a Z-Wave USB stick, the Z-Stick Gen5 from AEOTEC. Our controller can then manage z-wave sensors and actuators.

We used several sensors to be able to get data from the environment, with the Multisensor 6 from AEOTEC, a z-wave device that can monitor motion, temperature, light, humidity, and vibration. As an actuator for our home automation system, we used the smart switch 6 from Aeotec. It can be turned on or off via a z-wave command and can monitor the energy consumption of its electrical outlet.

In order to provide a realistic networked IoT system, we set up some rules on the home assistant controller. For instance, whenever the light is above 166 Lumix, i.e. the light is switched on, the controller will send a notification to a remote desktop and a smartphone through the Pushbullet API, a service aiming at sending a message to multiple platforms. Another example is a rule that switched on the smart plug when the motion sensor detects activity.

The analysis machine is a Debian based computer with an Intel i7-3770 CPU clocked at 3.40GHz with 8 cores. It has 16 GB RAM and is connected to the same VLAN as the home controller.

Metrics

To evaluate the efficiency of our detection engine, we used several metrics commonly employed in model evaluation. Those metrics are defined in the next paragraphs in the context of binary classification, since we want to predict good or abnormal behavior.

In our binary classification problem, True Positive (TP) is the number of correctly classified events as positive, True Negative (TN) is the number of correctly classified events as negative, False Positive (FP) is the number of falsely classified events as positive and False Negative (FN) is the number of falsely classified events as negative.

The common information to get from a model is its accuracy, which is the ratio between the number of good predictions and the total number of predictions. It can be defined by Eq. 1.

$$Accuracy = \frac{TP + TN}{N_{events}} \tag{1}$$

However, accuracy is not enough to determine a model efficiency. Indeed, a high accuracy cannot assure that the model will have a great detection power, especially if the dataset is imbalanced. Even if our dataset is balanced, it can still be useful to consider the Precision and Recall metrics, which can help to select among the different algorithms.

When it comes to classifying an input, the precision tells how many of the predicted events of a class were correctly predicted. This is the ratio between good predictions and the total number of events from a class. Recall reports how many of the classified events that should have been selected were actually selected. Since both of those metrics are relevant regarding our detection problem, we also introduced the F1-score, which is based on the precision and recall of the system. Those metrics can be obtained from Eq. 2

$$\begin{aligned} Recall &= \frac{TP}{TP + FN} \\ Precision &= \frac{TP}{TP + FP} \\ F1 &= 2 * \frac{Precision * Recall}{Precision + Recall} \end{aligned} \tag{2}$$

We want to maximize those metrics which reflect the detection performance of our system. As described earlier, we computed our metrics 5 times with 5-fold validation, with a training dataset and a testing dataset.

Tracing overhead

To measure the tracing overhead with LTTng, we used the sysbench benchmark first while tracing was enabled, and second while it was not. We used the integrated CPU benchmark with the following arguments. In this computation, the system verifies prime numbers by dividing the number with all values between 2 and the square root of the number. The argument –cpu-max-prime sets the highest number to verify during the benchmark.

```
$ sysbench —test=cpu —cpu–max–prime
=20000 run
```

We traced our device with the tracepoints described in Table 2. This contains 108 syscalls (including open,

Table 4 CPU overhead according to snapshot frequency

Snapshot frequency	Total time	Average time (per request)	95 percentile time (per request)
No tracing	371.51 s	37.15 ms	37.23 ms
0.5 s	+ 1.52%	+ 1.53%	+ 7.49%
1 s	+ 1.15%	+ 1.15%	+ 6.28%
2 s	+ 0.80%	+ 0.81%	+ 4.78%
3 s	+ 0.50%	+ 0.51%	+ 3.82%
4 s	+ 0.44%	+ 0.45%	+ 2.94%
5 s	+ 0.39%	+ 0.39%	+ 2.21%
10 s	+ 0.25%	+ 0.25%	+ 1.24%

close, access, write, fork, chmod, chown, mkdir), network queue with net dev queue kernel event, scheduler events (switch, process fork, process exit), kernel module (load and free) and some CPU events (frequency). We analyzed the system overhead while tracing and sending a snapshot through the network by varying the duration between two snapshots. The benchmark was repeated ten times, the mean results are presented in Table 4.

Moreover, we used the sysbench memory benchmark, which allocates a 1 kB buffer that will be read or written, sequentially or randomly, until the memory-total-size argument is reached. As before, we ran our benchmark ten times and present in Table 5 the mean results. We used this specific command :

```
$ sysbench --test=memory --memory
-total-size=20G run
```

These results highlight the low overhead of the LTTng tracer. Indeed, we traced a powerful device connected to an electrical outlet. Moreover, because the overhead introduced by tracing is low, our solution can be used even on limited resources devices. The CPU overhead can be reduced by decreasing the snapshot frequency. This will not, however, reduce the memory overhead, which is about 2.21% for this application.

Table 5 Memory overhead according to snapshot frequency

Snapshot frequency	Memory overhead
No tracing	3.75 s
0.5 s	+ 2.73%
1 s	+ 1.82%
2 s	+ 2.25%
3 s	+ 2.52%
4 s	+ 2.31%
5 s	+ 1.99%
10 s	+ 1.87%

Training efficiency

While we focus on the previously defined metrics to measure the efficiency of our models, we also noticed that some of our models needed significantly more computing time than others during the training phase. Indeed, the MLP took an 8 times longer training phase, while GBT needed more than 6 000 times the duration. The testing phase results are presented in the next table.

While those results are very good, we have to ensure that our models did not overfit our training set and can predict well future input events.

Analysis efficiency

Precision, recall, and f1 score

As explained earlier, we want to maximize the precision, recall, and F1 score of our detection engine. The computation of those efficiency metrics has been done with the scikit-learn built-in function *score*. The test set results are the average of the five cross-validations and are presented in Table 6.

Those results highlight two major points. First, the choice of the algorithm has a great impact on the detection accuracy. While our results emphasize that some algorithms could provide a top detection quality, some others fail at detecting intrusions on our dataset. This can be explained by the heterogeneity of the events that make our dataset. Indeed, as explained earlier, each field of a processed event is a feature for the machine learning algorithms. However, events have very different fields, as explained in Fig. 5. Some algorithms such as Decision Trees (DT), Random Forest (FT) or Gradient Boosted Trees (GBT) are very robust to that kind of data issues, while others like MultiLayer Perceptron (MLP) and Support Vector Machines (SVM) are not. We could apply some data transformations, like removing some fields or putting some default values, even normalizing some fields like the source IP address, but then we would have lost some relevant knowledge from the event. Indeed, it is essential to know the origin of the intrusion to have an adapted response to the security event.

Secondly, we can achieve very high detection rates with our solution. Indeed, because a smart device has been designed to have a specific behavior, it is easier than

Table 6 Anomaly detection efficiency of various algorithm

	Accuracy	F1	Precision	Recall
DT	99.97%	99.96%	99.98%	99.93%
RF	99.99%	99.99%	99.99%	99.99%
GBT	100%	99.99%	99.99%	99.99%
MLP	57.85%	23.07%	46.78%	27.25%
SVM	53.32%	99.99%	47.33%	99.99%

Syscall_open: (filename: "sudoTracing.sh", d_timestamp: 1780004334, pathname: "/etc/login.defs", p_name: "sudoTracing.sh")
Syscall_read: (filename: "sudoTracing.sh", d_timestamp: 1780004334, p_name: "sudoTracing.sh")
Syscall_readlinkat: (pathname: "/proc/sudoTracing.sh/exe", d_timestamp: 1780004334, p_name: "system-journal")
Net_dev_queue: (p_name: "sshd", d_timestamp: 890024167, source_port: 22, dst_port: 41640, Saddr: 1322077235, dadr: 1322077219)

Fig. 5 Heterogeneous Events

for traditional computers to learn this behavior, thus to detect any anomaly. Furthermore, it prevents an intruder from breaking our system using mimicry attacks that have been described by Sabahi et al.[20]. Our framework can precisely detect various kinds of intrusions while avoiding too many false alarms or failing to detect intrusions.

Analysis latency

To measure the analysis latency, we generated new snapshots containing a basic attack that every classifier studied was able to detect. This attack only run a freshly downloaded bash script which echoes "Infected" every 0.5 seconds. The created events have not been used in the training dataset or in the testing dataset; this is novel data for the classifiers. We defined the analysis latency as the mean time required for each classifier to predict the class of one event, making our measure independent of the network latency or the snapshot frequency. However, the more often snapshots are recorded, the quicker the intrusion alert will be raised, but the snapshot frequency should not be higher than the time needed to classify each event of the previously received snapshot. In most cases, we want to reduce the analysis latency to detect a security event as soon as possible. However, some users may want to run many analysis engines at the same time to do alert correlation when the priority is to reduce the number of false alarms.

The total detection latency can be defined as follows, if the analysis engine has been previously loaded, where the network latency and the intrusion event raised (and its time) can vary :

$$\begin{aligned} Latency_{detection} = Latency_{network} + \\ Time_{intrusion\_event} - Time_{first\_event} \end{aligned} \tag{3}$$

The results are exposed in Table 7.

These results highlight the need to carefully select the anomaly detection model. Indeed, GBT provides better results than the DT algorithm, but the time needed to predict the class of an event is nearly 7.5 times more than the former algorithm.

Discussion

Our results showed that some algorithms can provide better detection performance than others, but may require more time for learning or for classification.

Indeed, Decision Trees, Random Forest, and Gradient Boosted Trees can provide really great intrusion detection capabilities, based on tracing events, but those algorithms cannot detect intrusions that have not been previously learned. This is unlike OneClass SVM, which is much slower than other algorithms to predict the class of the incoming events.

Furthermore, only LSTM focuses on sequences of events, which can help detect some complicated intrusions. For instance, the attacker may attempt a mimicry attack, even if this is more complicated on smart devices than on traditional systems.

Moreover, our results, with our enabled set of tracepoints, show that DT based algorithms (including GBT of RF) provide very efficient intrusion detection capabilities, because they are very robust with input events that have different features. However, if the monitored events have nearly the same structure, for instance when only syscall events are being monitored, some algorithms may provide better results, especially deep-neural network based algorithms.

Finally, tracing the home controller is very useful to detect attacks on the system. For instance, it can detect attempts to eavesdrop information with a malware similar to the one tested. This attack can be very difficult to detect when only relying on the network information, if the covert channel is efficient enough. However, tracing the other devices could really improve the performance of our solution since it could prevent more threats, such as

Table 7 Analysis latency

	Avg. time per event
DT	1.23 ms
RF	1.68 ms
GBT	9.28 ms
OneClass SVM	6.79 ms
MLP	2.16 ms

ransomware on a smart TV or a Brickerbot attack on an alarm system.

Limitation

Our solution can be quickly integrated into home automation systems since it only requires having a tracer sending traces in CTF format, such as LTTng or BareCTF. However, those tools rely on the TPC/IP stack, which is a limitation for some devices. Indeed, some connected devices lack such connections and instead rely on Z-Wave, Zigbee or Insteon to send data in smart homes. Those protocols work in mesh networks where each device can send and forward information, and where the network topology can vary with time. This is very different from traditional TCP/IP networks.

Most of the algorithm that worked well with our experiments are supervised learning techniques, implying that they can only detect attacks similar to what they have learned. As a consequence, another limitation of our work is the number of attacks that have been learned, and the lack of real malware being used. Indeed, since most IoT threats are not open-source, we could only create attacks that mimic the behavior of real-world threats, like we did with our ransomware and spying tools. As noted by Pendlebury et al.[43], cross validation can introduce positive bias due to similarities between malware attacks. As intrusion detection is naturally biased, collecting malware samples targeting our monitored device could have been achieved with a honeypot emulating ARM hardware and the same OS, for instance by running Home Assistant on a QEMU virtual host.

In addition, to enhance our detection capabilities, we could have tried to develop a misuse detection engine to combine with our anomaly detection engine. This would require more powerful hardware for the analysis system, and could threaten the detection speed.

Finally, it should be noted that the proposed framework exists as a single point of failure, excluding other security measures taken. When installing security systems on IoT networks, measures should be taken to avoid unnecessary exposure to threats and malware, and to regularly patch and update software.

Conclusion and future work

In this paper, we presented a complete novel and flexible framework to detect intrusions on smart devices. It combines several machine learning algorithms and tracing techniques on the monitored devices. Our solution has shown very accurate intrusion detection results. We explained how it could be tuned to adapt to different devices, and explained why this solution performs well, benefiting from its host-based approach. Our tool could also have been adapted to work with the LTTng live tracing mode, to detect intrusions in real time.

Although we achieved great intrusion detection precision with our solution, some work can be done to improve our system usability. One future work would be adapting the open-source code of LTTng to support IoT protocols such as Z-Wave, ZigBee, and Insteon, and studying the traffic created with this tool in such a mesh network.

It would also be interesting to study the scalability of the analysis engine. Indeed, since the recorded snapshots can be analyzed independently from each other, our solution can be parallelized and scale to numerous devices, in small or bigger networks. However, the resources necessary to scale the monitored networks should be studied. The solution presented obtained excellent results and was optimized for quick detection. Further study could focus on the learning step processing time, since the learning phase has to be updated frequently to account for always evolving new threats. Another interesting area for further study is to select different models according to the targeted device, and even to combine more than one algorithm to improve further the detection accuracy.

Acknowledgments

We would like to gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC), Prompt, Ericsson, Ciena, and EfficiOS for funding this project.

Authors' contributions

The author(s) read and approved the final manuscript.

Funding

This project is funded by Natural Sciences and Engineering Research Council of Canada (NSERC), Prompt, Ericsson, Ciena, and EfficiOS.

Availability of data and materials

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Ecole Polytechnique Montreal, Montreal Quebec, H3T 1J4, Canada. [2]Brock University, St. Catharines, Ontario, L2S 3A1, Canada.

Received: 29 January 2020 Accepted: 1 October 2020
Published online: 23 November 2020

References

1. Evans D (2011) The internet of things : How the next evolution of the internet is changing everything. https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. Accessed: 24 Nov 2019
2. Atzori L, Iera A, Morabito G (2010) The internet of things: A survey. Comput Netw 54(15):2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010
3. Kambourakis G, Kolias C, Stavrou A (2017) The mirai botnet and the iot zombie armies. In: MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM). IEEE, Baltimore. pp 267–272. https://doi.org/10.1109/MILCOM.2017.8170867
4. OWASP OWASP top 10 IoT vulnerabilities. https://www.owasp.org/index.php/Top_IoT_Vulnerabilities. Accessed: 24 Nov 2019
5. ZarpelÃčo BB, Miani RS, Kawakani CT, de Alvarenga SC (2017) A survey of intrusion detection in internet of things. J Netw Comput Appl 84:25–37. https://doi.org/10.1016/j.jnca.2017.02.009
6. Hay A, Bray R, Cid D (2008) OSSEc host-based intrusion detection guide. Syngress Publishing

7. Security QI The sagan log analysis engine. https://quadrantsec.com/sagan_log_analysis_engine/. Accessed: 24 Nov 2019
8. Bezerra VH, da Costa VGT, Barbon Junior S, Miani RS, ZarpelÃčo BB (2019) Iotds: A one-class classification approach to detect botnets in internet of things devices. Sensors 19(14):3188. https://doi.org/10.3390/s19143188
9. Breitenbacher D, Homoliak I, Aung YL, Tippenhauer NO, Elovici Y (2019) Hades-iot: A practical host-based anomaly detection system for iot devices. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (Asia CCS '19), New York. pp 479–484. https://doi.org/10.1145/3321705.3329847
10. Cheng S-T, Wang C-H, Horng G-J (2008) Osgi-based smart home architecture for heterogeneous network. In: 2008 3rd International Conference on Sensing Technology. IEEE, Tainan. pp 527–532
11. Gubbi J, Buyya R, Marusic S, Palaniswami M (2013) Internet of things (iot): A vision, architectural elements, and future directions. Futur Gener Comput Syst 29(7):1645–1660. https://doi.org/10.1016/j.future.2013.01.010. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications – Big Data, Scalable Analytics, and Beyond
12. Hui TKL, Sherratt RS, SÃąnchez DD (2017) Major requirements for building smart homes in smart cities based on internet of things technologies. Futur Gener Comput Syst 76:358–369. https://doi.org/10.1016/j.future.2016.10.026
13. Acosta Padilla FJ, Baccelli E, Eichinger T, Schleiser K (2016) The future of IoT software must be updated. In: IAB Workshop on Internet of Things Software Update (IoTSU), Dublin, Ireland. ffhal-01369681f. https://hal.inria.fr/hal-01369681. Internet Architecture Board (IAB)
14. Antonakakis M, April T, Bailey M, Bernhard M, Bursztein E, Cochran J, Durumeric Z, Halderman JA, Invernizzi L, Kallitsis M, et al (2017) Understanding the mirai botnet. In: USENIX Security Symposium, Vancouver. pp 1092–1110
15. Saxena U, Sodhi JS, Singh Y (2017) Analysis of security attacks in a smart home networks. In: 2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence. IEEE, Noida. pp 431–436. https://doi.org/10.1109/CONFLUENCE.2017.7943189
16. Sikder AK, Petracca G, Aksu H, Jaeger T, Uluagac AS (2018) A survey on sensor-based threats to internet-of-things (IoT) devices and applications. arXiv preprint arXiv:1802.02041
17. Babar S, Stango A, Prasad N, Sen J, Prasad R (2011) Proposed embedded security framework for internet of things (iot). In: Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference On. IEEE, Chennai. pp 1–5
18. Wagner C, Dulaunoy A, Wagener G, Mokkadem S (2017) An extended analysis of an IoT malware from a blackhole network
19. Kolias C, Kambourakis G, Stavrou A, Voas J (2017) Ddos in the iot: Mirai and other botnets. Computer 50(7):80–84. https://doi.org/10.1109/MC.2017.201
20. Sabahi F, Movaghar A (2008) Intrusion detection: A survey. In: 2008 Third International Conference on Systems and Networks Communications. IEEE, Sliema. pp 23–26. https://doi.org/10.1109/ICSNC.2008.44
21. Nobakht M, Sivaraman V, Boreli R (2016) A host-based intrusion detection and mitigation framework for smart home iot using openflow. In: 2016 11th International Conference on Availability, Reliability and Security (ARES). IEEE, Salzburg. pp 147–156. https://doi.org/10.1109/ARES.2016.64
22. Burguera I, Zurutuza U, Nadjm-Tehrani S (2011) Crowdroid: Behavior-based malware detection system for android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '11. ACM, New York, NY, USA. pp 15–26. https://doi.org/10.1145/2046614.2046619
23. Eskandari S, Khreich W, Murtaza SS, Hamou-Lhadj A, Couture M (2013) Monitoring system calls for anomaly detection in modern operating systems. In: 2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, Pasadena. pp 19–20. https://doi.org/10.1109/ISSREW.2013.6688856
24. Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: A survey. ACM Comput Surv 41(3):15–11558. https://doi.org/10.1145/1541880.1541882
25. Murtaza SS, Sultana A, Hamou-Lhadj A, Couture M (2012) On the comparison of user space and kernel space traces in identification of software anomalies. In: 2012 16th European Conference on Software Maintenance and Reengineering. IEEE, Szeged. pp 127–136. https://doi.org/10.1109/CSMR.2012.23
26. Murtaza SS, Hamou-Lhadj A, Khreich W, Couture M (2014) Total ads: Automated software anomaly detection system. In: 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation, Victoria. pp 83–88. https://doi.org/10.1109/SCAM.2014.37
27. Chandola V, Banerjee A, Kumar V (2012) Anomaly detection for discrete sequences: A survey. Knowl Data Eng IEEE Trans 24:1–1. https://doi.org/10.1109/TKDE.2010.235
28. Jain AK, Murty MN, Flynn PJ (1999) Data clustering: A review. ACM Comput Surv 31(3):264–323. https://doi.org/10.1145/331499.331504
29. Shende S (1999) Profiling and tracing in Linux. In: Proceedings of the Extreme Linux Workshop (Vol. 2). http://www.cs.uoregon.edu/research/paraducks/papers/linux99.pdf
30. Desnoyers M, Dagenais MR (2006) The LTTng tracer: A low impact performance and behavior monitor for gnu/Linux. In: OLS (Ottawa Linux Symposium) Vol. 2006. pp 209–224
31. Wininger F, Jivan NE, Dagenais M (2016) A declarative framework for stateful analysis of execution traces. Softw Qual J 25:201–229
32. Desnoyers M, Dagenais M (2009) Deploying lttng on exotic embedded architectures. In: Embedded Linux Conference, vol. 2009, San Francisco
33. Proulx P Tracing bare-metal systems: a multi-core story - LTTng. https://lttng.org/blog/2014/11/25/tracing-bare-metal-systems/. Accessed: 24 Nov 2019
34. Bertauld T, Dagenais MR (2017) Low-level trace correlation on heterogeneous embedded systems. EURASIP J Embed Syst 2017(1):18. https://doi.org/10.1186/s13639-016-0067-1
35. Bengio Y, Courville A, Vincent P (2013) Representation learning: A review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828
36. Ezzati-Jivan N, Dagenais MR (2012) A stateful approach to generate synthetic events from kernel traces. Adv Soft Eng 2012:6–666. https://doi.org/10.1155/2012/140368
37. Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F (2015) Efficient and robust automated machine learning. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R (eds). Advances in Neural Information Processing Systems 28. Curran Associates, Inc., Montreal. pp 2962–2970. http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf
38. Wagner D, Soto P (2002) Mimicry attacks on host-based intrusion detection systems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02. ACM, New York, NY, USA. pp 255–264. https://doi.org/10.1145/586110.586145
39. Homoliak I, Teknös M, Ochoa M, Breitenbacher D, Hosseini S, Hanácek P (2018) Improving network intrusion detection classifiers by non-payload-based exploit-independent obfuscations: An adversarial approach. CoRR abs/1805.02684. http://arxiv.org/abs/1805.02684
40. Holik F, Horalek J, Marik O, Neradova S, Zitta S (2014) Effective penetration testing with metasploit framework and methodologies. In: 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI). IEEE, Budapest. pp 237–242
41. Kharraz A, Robertson W, Balzarotti D, Bilge L, Kirda E (2015) Cutting the gordian knot: A look under the hood of ransomware attacks. In: Almgren M, Gulisano V, Maggi F (eds). Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham. pp 3–24
42. Kotsiantis S, Kanellopoulos D, Pintelas P, et al (2006) Handling imbalanced datasets: A review. GESTS Int Trans Comput Sci Eng 30(1):25–36
43. Pendlebury F, Pierazzi F, Jordaney R, Kinder J, Cavallaro L (2019) TESSERACT: Eliminating experimental bias in malware classification across space and time. In: 28th USENIX Security Symposium (USENIX Security 19). USENIX Association, Santa Clara, CA. pp 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury
44. Yap BW, Abd Rani K, Abd Rahman HA, Fong S, Zuraida K, Abdullah NN (2014) An application of oversampling, undersampling, bagging and boosting in handling imbalanced dataset. In: Proceedings of the first international conference on advanced data and information engineering (DaEng-2013). Springer, Singapore. pp 13–22. https://doi.org/10.1007/978-981-4585-18-7-2
45. Hsu C-W, Chang C-C, Lin C-J (2003) A practical guide to support vector classification:1396–1400
46. Chapelle O, Scholkopf B, Zien, Eds. A (2009) Semi-supervised learning (chapelle, o. et al., eds.; 2006) [book reviews]. IEEE Trans Neural Netw 20(3):542–542

47. Devi D, Biswas S, Purkayastha B (2019) Learning in presence of class imbalance and class overlapping by using one-class svm and undersampling technique. Connect Sci 31:1–38. https://doi.org/10.1080/09540091.2018.1560394

Publisher's Note