Threat Detection and Mitigation with Honeypots: A Modular Approach for IoT

Chapter · October 2022

DOI: 10.1007/978-3-031-17926-6_5

4 authors, including:

Patricia Sousa
University of Porto
15 PUBLICATIONS **40** CITATIONS
SEE PROFILE

João Resende
Universidade NOVA de Lisboa
19 PUBLICATIONS **35** CITATIONS
SEE PROFILE

Luís Antunes
University of Porto
119 PUBLICATIONS **1,184** CITATIONS
SEE PROFILE

Threat Detection and Mitigation with Honeypots: A Modular Approach for IoT

Simão Silva[1[0000−0002−7692−5822]], Patrícia R. Sousa[1,2[0000−0002−0268−9134]], João Resende[1,2[0000−0003−0125−4240]], and Luís Antunes[2[0000−0002−9988−594X]]

[1] CRACS - INESCTEC
[2] University of Porto
simao.sfos@gmail.com, {psousa,jresende,lfa}@dcc.fc.up.pt

Abstract. A honeypot is a controlled and secure environment to examine different threats and understand attack patterns. Due to the highly dynamic environments, the growing adoption and use of Internet of Things (IoT) devices make configuring honeypots complex. One of the current literature challenges is the need for a honeypot not to be detected by attackers, namely due to the delays that are required to make requests to external and remote servers. This work focuses on deploying honeypots virtually on IoT devices. With this technology, we can use endpoints to send specific honeypots on recent known vulnerabilities on IoT devices to find and notify attacks within the network, as much of this information is verified and made freely available by government entities. Unlike other approaches, the idea is not to have a fixed honeypot but a set of devices that can be used at any time as a honeypot (adapted to the latest threat) to test the network for a possible problem and then report to Threat Sharing Platform (TSP).

Keywords: CVE · Honeypot · Internet of Things · Intrusion Detection · Security · Vulnerability.

1 Introduction

The evolution of technology is changing society and the environment around us. With users spending more time looking at a screen than sleeping [10], it is important to alert users to the hardships they may face online regarding security and privacy. As the software is not unbreakable and new vulnerabilities are identified and disclosed daily, devices and data can be compromised. New vulnerabilities are threats that an attacker can use to take advantage of the user, creating severe consequences.

The growing number of cyberattacks and their extensive damage has changed the mindset of organizations, with more resources being applied to cybersecurity. According to statistics from AV-TEST as of January 2021 [3], the security community knew over a billion malicious executable scripts. Digital transformation has also meant increased cybercrime, often associated with significant financial losses for individuals and organizations. As new daily vulnerabilities

are identified and disclosed, attackers constantly find new ways to compromise devices. However, the increase in the number of cyberattacks and their extensive damage has changed organizations' mindset, applying more resources, specific teams, and dedicated tools to deal with this problem.

The use of honeypots has been a vital tool to face the increase in cyberattacks. A honeypot is configured to detect, circumvent, or prevent unauthorized use of information systems. A honeypot is a controlled and secure environment to examine different threats and understand attack patterns. It is possible to analyze attack traffic independently of regular network traffic and away from critical infrastructure. Thus, security teams can focus on analyzing just the threat. Honeypots are versatile and are not addressed exclusively to a specific problem like other standard solutions (firewalls and antivirus). Given its functioning, it becomes an essential information tool to spot existing threats and possibly new ones, which makes them an important asset, especially relevant for Security Operations Center (SOC) teams since they possess a significant database of attacks' artifacts and, consequently, a significant source of threat intelligence. Honeypot systems are becoming a mandatory component for cybersecurity defense by working with threat intelligence platforms.

Although many honeypots are aimed at different purposes, these systems have some constraints. Given the diversity of devices, there are still limitations in using these systems outside of x86 architectures, leaving out capable devices such as Raspberry Pi, which align computational power with low energy consumption. Another issue is regarding its use. With the proliferation of the cloud, we are witnessing the phenomenon of services migration to external servers, which, in the case of honeypots, the trend is to use remote machines as baits, and often, the traffic is redirected from the local to the cloud network. In this case, resorting to this method allows the attacker to quickly detect that it is in a honeypot by measuring the latency times of Internet Control Message Protocol (ICMP) ECHO requests, thus making the honeypot ineffective.

This paper contributes to the existing literature with the creation of an automatic honeypot instance deployment when a vulnerability is present in a device in the network, capable of running a variety of devices, including low-power, performance-constraint devices, and provide a mechanism that can discretely monitor honeypots instances to collect intelligence of tactics and techniques of intruders. Also, overall, we contribute with an autonomous threat detection flow able to analyze the diversity of devices in the network and detect and mitigate, if possible, its vulnerabilities.

2 Related work

In IoT-based networks, new devices entering the network are automatically configured due to their open nature, which leaves these networks subject to many attacks, as described in [18]. Given these systems' complexity and configurations, a common approach is to use low-interaction honeypots as they are easier to install and configure and with low risk. *Dshield* [1], *Glastopf* [26] and *OpenCanary*

[4] can monitor requests and alert of potential unwanted traffic but, due to its characteristics, has limitations on the information gather from attackers which unable the trace of attacks. Also, the attackers can easily identify them due to their appearance and behavior [11,20].

Honeyd [21] is a simulated honeypot environment intended to simulate the virtual network topology to filter network packets based on user preferences. When sending a response packet, the personality engine makes it match the network behavior of the configured operating system personality [22]. As it can emulate operating systems, *Honeyd* can appear to the attacker as a router, web server, or DNS server, allowing the honeypot to blend into existing networks. With this behavior, *Honeyd* can spoof responses about active fingerprint measurements, such as those used by the NMAP tool.

The increasing need for honeypots has led to the concept of Honeypot-as-a-Service (HaaS), where, instead of being configured locally, they are made available to users by cloud providers, thus eliminating the need to worry about hardware, software, and human resources to create and maintain the honeypot. There are public cloud provider honeypots [16] that, although it reduces costs, still require labor to apply and maintain the configurations. An evaluation of this solution [14] shows that it still involves human resources to use and maintain the settings, and the traffic flow is in order.

Honeyd's low-interaction honeypots [6] explore the idea of honeypots that adapt to the needs and changes of organizations and use unassigned IP addresses to launch instances. With traffic being diverted in the background from one honeypot to another, *Honeyd* honeypots are more reliable for intruders. However, this requires configuring physical machines dedicated exclusively to the honeypot network, and as we would need to mimic more rogue systems, more IP addresses also need to be available.

Regarding the industrial sector, there is also a concern about safety, namely for Programmable Logic Controllers (PLCs). *HoneyPLC* [19] overcomes the limitations of current honeypot implementations for PLCs, with easy fingerprinting and low interaction levels being some examples. However, the system remains stagnant and unable to adapt to new changes in existent threats.

A comprehensive solution for honeypot systems focused on IoT environments is the YAKSHA [17] project, which consists of the development of a cloud-based platform that allows an organization to perform continuous monitoring and penetration tests of its infrastructure without the need of need to maintain such infrastructure nor have dedicated staff. It provides a way to create custom honeypots tailored to specific needs that include installing a specific set of services that will be used to lure attackers. Although it is designed for the IoT environment, the honeypot deployment, like the malware analysis tool, requires additional hardware based on the x86 architecture. Also, while a mechanism for patching and configuring IDS is mentioned in vulnerability detection, it is unclear how this process is accomplished.

Table 1 shows the comparison of characteristics of current state-of-the-art against our proposed solution (Sandboxing for Threat Detection and Mitigation (STDM)).

	[1]	[26]	[4]	[21]	[16]	[14]	[19]	[17]	STDM
Honeypot deployment	X	X	X	X	X	X	X	X	X
Device profiling				X	X	X	X	X	
Honeypot customization				X	X	X	X	X	X
Decentralized deployment supported	X		X					X	X
CVE vulnerability scanning									X
IDS connection								X	X
TSP connection									X
ARM compatibility	X	X	X						X

Table 1. State of the art's features summary

All these solutions lack the characteristics of environmental independence and mobility. The presented honeypots cannot be deployed on any device, at any time, regardless of its architecture. Given the highly dynamic environments associated with IoT, honeypots must also be interchangeable between available devices on the network, as well as the ability to be easily replicated across different devices.

3 Architecture

Gathering threat knowledge is one of the most important and necessary tasks in today's cybersecurity. The increase in attacks led to changes in collecting knowledge of (potential) threats and how to take advantage of that knowledge.

Figure 1 shows the overview of the architecture of our system. The Threat Sharing Platform (TSP) component will update their database with the Common Vulnerabilities and Exposures (CVE) entries (1). The *STDM Central Coordinator* will act as a go-between managing the device scanning history and honeypots. From time to time, the devices will send their list of software installed to the *STDM Central Coordinator* that, in turn, requests the information about CVEs from the TSP (2). For each CVE entry, the coordinator parses the Common Platform Enumeration (CPE) list - a standard that specifies a structured naming scheme for software based on the uniform resource identifiers syntax - and searches for a match in the list received from the devices (3-4). When a match is found, the *STDM Central Coordinator* will select a random device and launch a honeypot with the same operating system and software version of the match (5). Inside the honeypot, an admin user can verify if the match is not a false positive and provide that information to the *STDM Central Coordinator* (6). From the honeypot logs (7), an administrator user can observe the malicious requests and

can create, if possible, rules to be applied to the *Host-based Intrusion Detection System (HIDS)* of all devices (8).

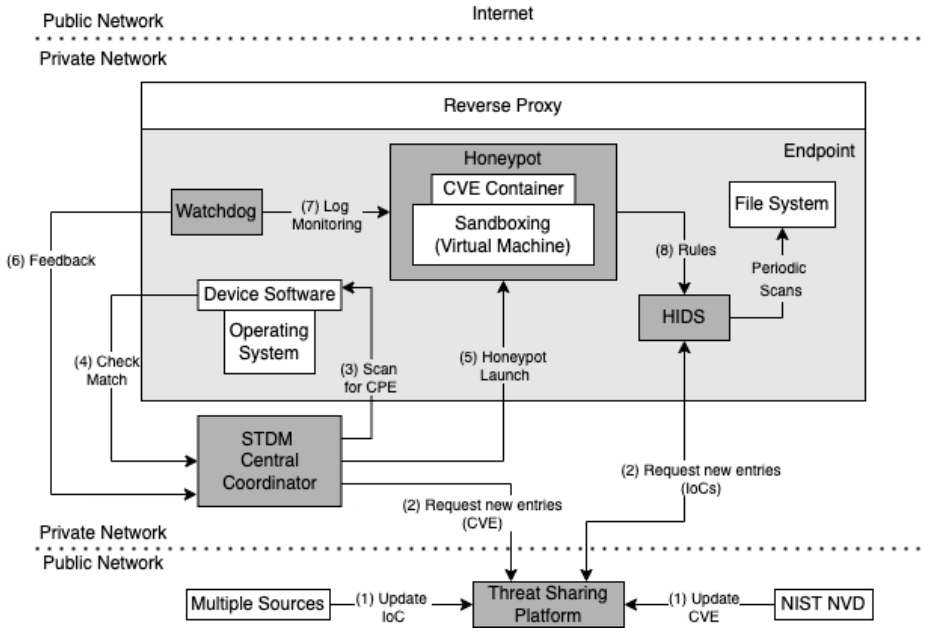

Fig. 1. System architecture

In the following subsections, we will describe the highlighted components of our architecture.

3.1 Threat Sharing Platform

This component is responsible to store information from Indicators of Compromise (IoC) and CVE. From time to time, the platform will retrieve IoC from public and trustworthy feeds and act as a local mirror database of *National Institute of Standards and Technology (NIST)*'s National Vulnerability Database (NVD) database of CVE entries.

Even in security breaches, we can still learn with the shreds of evidence left by the intruder. IoC [9] enhances the honeypot by capturing malicious activities on the network at their initial stage, preventing them from becoming more significant problems and compromising the security of the infrastructures.

Also, in our approach, we intend to enrich our knowledge through known public vulnerabilities, namely CVEs [12]. As they are public, we can recreate the environments that mimic that CVE and use it to gather IoC that can be later used to enhance our HIDS, as shown in steps 5, 7, and 8 of Figure 1.

Those environments are launched as honeypots and are exposed to the Internet as decoy services.

3.2 Host Intrusion Detection System

We use the component of HIDS to detect and automate the honeypot initialization process according to what is happening in the network (environment).

To improve security on our devices, we have deployed two HIDS systems - one that offers anomaly and signature variants and one that offers Intrusion Prevention System (IPS) capabilities - on each device, as these systems are designed to increase protection against internal and external threats. In addition, they can monitor network traffic to and from the machine, observe running processes, and inspect system logs for patterns that match known cyberattacks. While they can be seen as limited due to their low visibility (limited to the host, which decreases the decision-making context), their deep visibility into the internals of the host allows the security teams to analyze activities in a high level of detail. Thus, unlike network-based intrusion detection systems [15], they can directly access and monitor data files and processes of the targeted system.

3.3 STDM Central Coordinator

This component is the gateway between the threat-sharing platform and the devices in the network. It is responsible for checking if a device in our network has vulnerable software, keeping records of those threats, and launching the honeypots containing them.

3.4 Watchdog

Watchdog is an agent installed on the host system to allow users to access any honeypot file without having to directly interact with it or learn the back-end command syntax to access it. From *Watchdog*, a user can monitor the logs of the vulnerable service running on the honeypot instance. With this information, it can analyze logs and analyze malicious queries that can later be manually added to the IoC database in TSP, which allows the platform to be aware of insider attacks viewed or received by honeypots, allowing to prioritize threats in a global overview.

The *Watchdog* can be adapted to any installation of service in the virtual machine.

3.5 Honeypot

The *honeypot* instances are virtual machines with service(s) exposed to the Internet. Those services are installed directly on the file system or using containers (to avoid external changes that can interfere with the service availability and expected behavior). The vulnerable services are represented by the *CVE container*

inside the *Sandboxing Virtual Machine* in the *Honeypot* (in the Figure 1). The instances are prepared to run in x86-64 and ARM architectures, thus enabling (virtually) the launching of an instance on any device on the network. Also, this mobility enables a given instance running a given service can be easily shifted between devices.

These instances require isolation techniques so that the attacker can not gain access to the host system and damage it. The typical strategy used in the cybersecurity world is sandboxing, which is a security mechanism that tricks an application or program into thinking it is running on a regular computer [5]. It allows us to provide services to intruders in a tightly controlled environment without allowing the services and the intruders' actions to harm the host device.

4 Implementation

Following the presentation of our architecture and its components in Section 3, this chapter describes the technical details of the implementation of our system, the decisions made, and problems that emerged during said implementation.

4.1 Threat Sharing Platform

For our architecture, we opt to use Malware Information Sharing Platform (MISP) as our threat (intelligence) sharing platform [27]. Interactions with MISP can be done by its Representational State Transfer (REST) interface. Using PyMISP [23], we can manage the platform and add new functionalities. In our case, we use it to keep our database of IoC and CVEs updated.

Regarding CVEs, we keep our database updated by a custom module-like script we developed, henceforth referred to as *CVE module*. That module will use NIST's REST service to periodically request new or recent modified entries from the CVE database since the last request. The service will answer with an empty response - meaning that we are up to date - or a list of entries with their respective details in JavaScript Object Notation (JSON) format.

For each element in that list, we must verify the contents of the *input* field. If the field is empty, the CVE is under analysis, and the information about this vulnerability is still preliminary. If not, we make a subsequent request to retrieve the CPE software list affected by the vulnerability. Using the information from both requests, namely ID, description, reference links, and CPE list, we use the CVE module to create a MISP event with all these attributes and store it on the platform, making it available to the other components of our system.

4.2 STDM

This component is the bridge between the network devices and the TSP. The component comprises a server with a database storing the values of the device's last authentication timestamp (for debugging) and several vulnerabilities found (for statistical purposes). All communications between devices and our server use

SSL/TLS to provide confidentially and integrity. The devices will authenticate themselves into the component using a certificate-based authentication where each device has its own issued certificate that was later manually installed. This method allowed us easier management of the devices, given that only devices with a valid certificate installed can access our system. Then, they will periodically send the names and versions of the software installed on the host to the server. In turn, the server will request the TSP platform for the list of CVEs and respective CPEs and verify if there is a match against the information received from the device. If a match is found, it increments the positive matches statistics, launches a honeypot instance with the same operating system, and informs the admin of the vulnerable software version to install it on the honeypot later. The honeypot instance starts on a random device with the necessary resources to run the instance and is available in the network at the launch time.

The software matching is not linear and requires a pre-processing step. Given that the names in NVD's database are in CPE syntax and differ from the syntax used by operating systems, we attempt to translate the CPE syntax closer to the names on the operating systems. However, we notice a high rate of false positives in some situations, which we attribute to the versioning naming format. In these cases, a warning is displayed to an admin user that has to check if it is a match manually.

4.3 Host Intrusion Detection System

For our solution, we opt to use a combined solution of HIDS by leveraging the Fail2ban's IPS characteristics. As HIDS are *passive* in its essence, meaning they identify but do not prevent suspicious activity, we chose to add an IPS system as they are active in preventing those suspicious activities. Thus, to take advantage of all of their features, we used OSSEC (HIDS) combined with Fail2ban (HIDS/IPS).

4.4 Isolated environments

Honeypot's security creates a vulnerability in the host that is deliberately singled out for attackers to use. In addition, deploying these systems in secure environments is necessary so that activities on honeypot systems do not affect the host. This section describes the technologies and methods applied to achieve this goal with that goal in mind.

Virtualization was used because the kernel and libraries are not shared between host and guest to prevent an attacker from accessing the host machine. From the functionality point of view, it allows the creation of several isolated environments from single hardware, thus allowing better usage of resources. While containers are highly supported for our architectures (x86 and ARM), the isolation provided by virtual machines was the critical factor to consider.

Virtualization technology has many solutions for x86 architectures, but the same does not apply to devices with ARM architectures. However, continuous improvements in boards, such as the Raspberry Pi, have made it feasible to use

Kernel-based Virtual Machine (KVM), a Linux kernel module that allows the Linux kernel to act as a hypervisor, which has become more accessible for users with Linux distributions with the pre-compiled Linux KVM kernel module.

The chosen virtual machine provider was Multipass [8] due to the support of different architectures and operating systems, allowing it to scale its use to any system. However, to manage virtualization, we had to change Multipass settings. By default, it uses the QEMU hypervisor [7] that, although it runs perfectly on x86-based processors, presented some incompatibility issues when running on ARM processors, making Multipass unable to run. So, we change the hypervisor to LXD [2], which is a REST API that connects to *libxlc*, which is the Linux Containers (LXC) library - a solution for virtualizing software at the operating system level within the Linux kernel [24] that can monitor the virtual machines from the host's filesystem.

The honeypot system is a Multipass virtual machine deployed by the *STDM Central Coordinator* component. Currently, only Ubuntu cloud images can be deployed automatically. However, the usage of custom images is also supported.

The system's image is very similar to a non-graphical version of Ubuntu Server, with no visible distinguishability that can hint to the intruder that it is a decoy. Once up and running, an admin user sets up the vulnerable service, including the additional required software and the necessary port-forwarding with IPtables. The services are installed preferably from the source code of the corresponding vulnerable version and, when possible, deployed on containers to take advantage of its portability so that installation is environment independent and the service easily deployed (it can start in just a few seconds) with minimal overhead.

4.5 Watchdog

We have two ways to access the honeypot: the Multipass interface or the LXC/LXD interface. Both options allow us to access the honeypot and monitor the services' logs, but they can reveal to the attacker that he/she is in a decoy system. Using the Multipass interface, an attacker can monitor the SSH log file and discover a user with *Sudo* privileges monitoring the log files, which can be an admin user and thus jeopardize the decoy system. Instead, if the LXC/LXD interface is used, the attacker can be suspicious of the system if it discovers that the *lxd-agent* is running (given this is necessary for the host-honeypot communication) or if a process monitoring some log file own by the root is running. The resolution of these issues was two: the first one is mounting the */proc* directory with *hide-pid=2* flag, thus denying users access to processes besides theirs; the second one is by unmounting and mounting the system image file (.img) used to support the virtual machine on the host's file system. This second option comes with the inconvenience of repeating the unmount/mount routine due to a limitation on the image file that supports the virtual machine file system. When mounted, we notice that later changes made within the virtual machine's file system are not reflected in the mount point and vice versa. However, we can see earlier changes if we repeat the unmount/mount routine on the image file. This limitation forces

us to redo the unmount/mount routine to get new data from the log files, making real-time log monitoring impossible.

Given the options above, we opt to use the LXC/LXD interface. Even though the flag usage can indicate the presence of a decoy system, it can also be seen as a default security measure employed by an admin. Also, considering the usability trade-off between the two resolutions, the usage of the LXC/LXD interface provides a better user interaction.

To avoid the need for admin users to learn and remember the syntax of the command to access the instance, we developed a script that abstracts the backend command and accepts user input to execute commands as if the users were inside the instance.

5 Evaluation

In this section, we aim to evaluate the performance of our system. Our evaluation focuses on the measurement of delay between the time attack is detected (i.e., the malicious request is detected) on the honeypot and the time the central coordinator receives that information.

The goal of our evaluation is to demonstrate the feasibility of the solution. For the performance evaluation, the main challenge was to measure and compare with a standard service in the literature. An example of this service type is Apache2 [13], which is one of the most used web servers. We use a vulnerability of this service to demonstrate our prototype (CVE-2019-10092) using the Apache HTTPd 2.4.38 server (this issue affects the versions from 2.4.7 up to 2.4.51, included).

5.1 Environment description

For our scenario, we used a Raspberry Pi 4 Model B with 4GB of RAM with a Broadcom BCM2711 1.5 GHz Quad-Core 64-bits processor connected through an Ethernet cable to our router Thomson TG784n. Also, we acquired a virtual machine from Microsoft Azure with 2 GB of RAM and an Intel Xeon E5-2673 v3 2.40 GHz processor truncated at two cores. Both used Ubuntu 20.04 LTS as the operating system.

5.2 Methodology

We designed our testing scenarios to consider different honeypot installations. We tested our system in the following scenarios: Setup 1 (HaaS with physical machines): cloud virtual machine exclusively dedicated to being a honeypot; Setup 2 (HaaS with virtual machines and using LXD): using Multipass instances as honeypots on the cloud virtual machine to evaluate the impact of virtualization overhead; and Setup 3 (*STDM*): our solution runs in a local network with a Raspberry Pi running Multipass instances as honeypots.

The tests were performed to explore the differences against the HaaS concept and usage of cloud machines to compare the delay times between on-premise and off-premise solutions (State-of-the-Art). Given the usage of virtualization, we also tested its impact on the overall results. Scenarios from Setup 1 and Setup 2 allow us to test and compare the State-of-the-Art solutions with our solution (Setup 3).

Regarding deployment, we launch the vulnerable Apache Server from a container available on a Docker Registry to simplify the installation process in all setups. For setups 1 and 2, to allow queries from cloud machines to reach the central coordinator, we had to configure IPTables rules and delete the default Azure forwarding rules causing the requests to be dropped. For setups 2 and 3, we deployed a Multipass virtual machine, installed the Docker software, and configured the necessary redirects from the host to the virtual machine to reach the server from the outside.

The attack scenario was performed on a client-server basis running an environment to mimic the CVE-2019-10092 in the Apache HTTPd 2.4.38 server.

We measure the delay times between the moment the malicious request is detected and when the information is recognized by the *STDM Central Coordinator*. For this, a specific tool was designed, located precisely in the *STDM Central Coordinator*, which provides the network latency generated (in milliseconds) by our solution. The exploitation was performed using a bash script containing a cURL request to the server's IP address.

5.3 Results

This section aims to show the results and comparison with state-of-the-art.

Table 2 presents the results for each of the three scenarios of network throughput. For gathering the network information, we used *iPerf* [25], which is software used to test the network bandwidth. We collected two-time samples to measure the setup performance only in network bandwidth. The results presented in the table are the mean and standard deviation for the three setups and both measurements (TCP bandwidth and UDP jitter).

	Setup 1	Setup 2	Setup 3
TCP bandwidth (Mbits/sec ± sd)	1,22 ± 0,89	1,17 ± 1,06	30,29 ± 2,91
UDP jitter (ms ± sd)	4,02 ± 1,12	4,32 ± 2,88	0 ± 0

Table 2. Latency results for the three setups

Figure 2 represents the delay associated with each implementation scenario. The latency of the process between detecting the malicious request in the honeypot instance and the moment when the *STDM Central Coordinator* receives this information. To do this, we collect three samples for each number of requests.

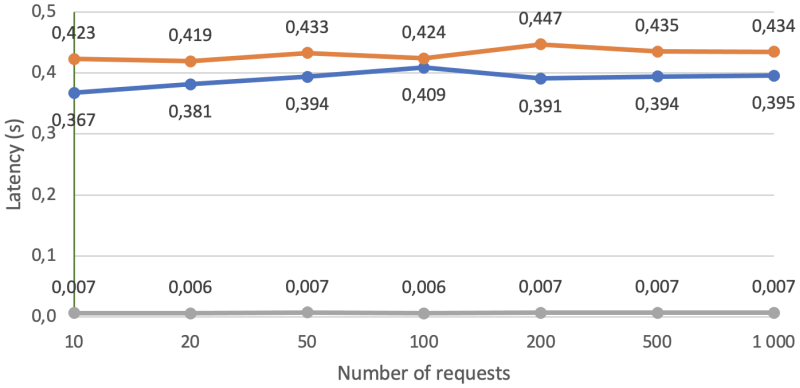

Fig. 2. latency (ms) per number of requests

The results of Setup 1 (HaaS with physical machines) show a delay in the order of 390ms with minimal oscillations. Then, Setup 2 (HaaS with virtual machines and using LXD) presents small delay increments compared to Setup 1, showing that virtual machines have a residual overhead on performance and is, therefore, a good choice when looking for multiple deployments and host isolation. On the other hand, Setup 3 (STDM - our implementation) shows the main advantage of having Honeypots on the network endpoints because the delay associated with this task is reduced to close to 0 with on-premises honeypots. Similar behavior could be achieved with Honeypots running on a separate Local Area Network (LAN), but these solutions raise concerns about the actual usability of the honeypot network (as shown in related works).

6 Conclusion

In this work, we studied and created a solution to autonomously detect and prevent threats based on public information of known treats and specialized software that gathers, parses, and analyzes information.

We offer a new security approach to deploying dynamic local honeypots capable of running on devices not previously used for this purpose. As far as we know, this is the first proposal for a dynamic honeypot that can be installed on any network device in order to test network devices, being capable of confusing the attacker, as he/she never knows if it is a network device or a honeypot placed for this purpose. IoT environments benefit from this solution as sensors are often the most vulnerable devices on a network.

It combines IDS and IPS systems to detect and automate the honeypot initialization process according to what is happening in the network (environment). This process can be used as the first line of defense for active or passive intrusion detection and prevention, as it also allows insights into new attacks.

Contrary to previous work, the idea is not to have a fixed honeypot but to have a set of devices that can be used at any time as a honeypot (adapted to the most recent threat) to test the network for these possible threats and then report to the MISP. Also, we developed a honeypot solution that can run on multiple devices of different architectures without allocating specific hardware.

We have experimentally demonstrated that deploying decoy systems in on-premises infrastructure is possible and feasible, even on power-constrained devices. Regarding latency and comparing to standard HaaS, we reduced it from 360ms to closer to 0ms.

7 Future Work

As future research challenges, the paper should be extended with real-world use cases, which must include an evaluation of the normal functioning of IoT devices while acting as a honeypot and other evaluation parameters such as memory consumption and power consumption in a real-world IoT use case.

We think another exciting path focuses on enhancing the system with human-in-the-loop mechanisms. In addition to the supervision performed automatically by the system, a user must be able to add their domain knowledge that could identify points of failure and new rules for improving the detection of abnormal patterns, reducing false alarm rates and adversarial attacks. As misclassification can have serious consequences, human-in-the-loop should be used to confirm all patterns and decisions. In this sense, human-in-the-loop in IPS and IDS systems can validate the abnormal events and generate rules to feed the first defense layer. On our current implementation, an administrator can add rules with Fail2ban, but it is essential to make more steps for the overall system, allowing the launch of honeypots with new rules.

If an attacker gains access to the system, they may attack the system (as explained in the security analysis section), but one solution to blocking this type of attack is to use multiple software vendors to perform this process. In combination with Byzantine Fault Tolerance (BFT), it will mitigate and detect another vector of attacks on the host machine, allowing to identify wrong answers from the system.

Acknowledgements. This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020.

The work of Simão Silva was partially funded by the SafeCities POCI-01-0247-FEDER-041435 project through COMPETE 2020 program. The work of Patrícia R. Sousa was partially supported by the Project "City Catalyst – Catalisador para cidades sustentáveis", with reference POCI-01-0247-FEDER-046119, financed by Fundo Europeu de Desenvolvimento Regional (FEDER), through COMPETE 2020 and Portugal 2020 programs. João S. Resende's work was partially supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu). National Funds also partially supported this

work through the Agência para a Modernização Administrativa, program POCI - Programa Operacional Competitividade e Internacionalização, within project POCI-05-5762-FSE-000229.1.

References

1. DShield Honeypot. https://isc.sans.edu/honeypot.html, (Accessed on 25/01/2022)
2. LXD - Introduction. https://linuxcontainers.org/lxd/introduction/, (Accessed on 26/07/2021)
3. Malware Statistics & Trends Report. https://www.av-test.org/en/statistics/malware/, (Accessed on 26/01/2021)
4. Opencanary honeypot. https://opencanary.readthedocs.io/en/latest/, (Accessed on 25/01/2022)
5. Virtualization-Based Sandboxes are Vulnerable to Advanced Malware. https://www.lastline.com/blog/virtualization-based-sandboxes/, (Accessed on 28/05/2021)
6. Artail, H., Safa, H., Sraj, M., Kuwatly, I., Al-Masri, Z.: A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks, journal = Computers & Security **25**(4), 274–288 (2006). https://doi.org/https://doi.org/10.1016/j.cose.2006.02.009, https://www.sciencedirect.com/science/article/pii/S0167404806000587
7. Bellard, F.: QEMU, a fast and portable dynamic translator. In: USENIX annual technical conference, FREENIX Track. vol. 41, p. 46. Califor-nia, USA (2005)
8. Canonical: Multipass orchestrates virtual Ubuntu instances. available at: https://github.com/canonical/multipass (Accessed 20 July 2021) (2015)
9. Catakoglu, O., Balduzzi, M., Balzarotti, D.: Automatic extraction of indicators of compromise for web applications. In: Proceedings of the 25th international conference on world wide web. pp. 333–343 (2016)
10. Editors, I.I.: US adults added 1 hour of digital time in 2020. https://www.emarketer.com/content/us-adults-added-1-hour-of-digital-time-2020 (01 2021), (Accessed on 19/08/2021)
11. Franco, J., Aris, A., Canberk, B., Uluagac, A.S.: A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. IEEE Communications Surveys & Tutorials **23**(4), 2351–2383 (2021)
12. Guo, M., Wang, J.A.: An ontology-based approach to model common vulnerabilities and exposures in information security. In: ASEE Southest Section Conference (2009)
13. Hu, Y., Nanda, A., Yang, Q.: Measurement, analysis and performance improvement of the Apache web server. In: 1999 IEEE International Performance, Computing and Communications Conference (Cat. No. 99CH36305). pp. 261–267. IEEE (1999)
14. Jafarian, J., Niakanlahiji, A.: Delivering Honeypots as a Service (01 2020). https://doi.org/10.24251/HICSS.2020.227
15. Javaid, A., Niyaz, Q., Sun, W., Alam, M.: A deep learning approach for network intrusion detection system. Eai Endorsed Transactions on Security and Safety **3**(9), e2 (2016)
16. Khan, N.F., Mohan, M.M.: Honey pot as a service in cloud. International Journal of Pure and Applied Mathematics **118**(20), 2883–2888 (2018)

17. Kostopoulos, A., Chochliouros, I.P., Apostolopoulos, T., Patsakis, C., Tsatsanifos, G., Anastasiadis, M., Guarino, A., Tran, B.: Realising honeypot-as-a-service for smart home solutions. In: 2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM). pp. 1–6. IEEE (2020)
18. La, Q.D., Quek, T.Q., Lee, J., Jin, S., Zhu, H.: Deceptive attack and defense game in honeypot-enabled networks for the internet of things. IEEE Internet of Things Journal **3**(6), 1025–1035 (2016)
19. López-Morales, E., Rubio-Medrano, C., Doupé, A., Shoshitaishvili, Y., Wang, R., Bao, T., Ahn, G.J.: Honeyplc: A next-generation honeypot for industrial control systems. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 279–291 (2020)
20. Mphago, B., Mpoeleng, D., Masupe, S.: Deception in web application honeypots: Case of glastopf. International Journal of Cyber-Security and Digital Forensics **6**(4), 179–185 (2017)
21. Provos, N.: Honeyd - A virtual honeypot daemon. In: 10th DFN-CERT Workshop, Hamburg, Germany. vol. 2, p. 4 (2003)
22. Provos, N., et al.: A Virtual Honeypot Framework. In: USENIX Security Symposium. vol. 173, pp. 1–14 (01 2004)
23. PyMISP, G.: PyMISP - Python Library to access MISP. PyMISP (accessed on 20 February 2021) `https://github.com/MISP/PyMISP`
24. Senthil Kumaran, S.: Practical LXC and LXD: linux containers for virtualization and orchestration. Springer (2017)
25. Tirumala, A.: Iperf: The TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf/ (1999)
26. Tools, K.Y.: Glastopf - a dynamic, lowinteraction web application honeypot
27. Wagner, C., Dulaunoy, A., Wagener, G., Iklody, A.: Misp: The design and implementation of a collaborative threat intelligence sharing platform. In: Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security. pp. 49–56 (2016)