Hardening the Internet of Things: Toward Designing Access Control For Resource Constrained IoT Devices

Manuel Bessler
Manuel.Bessler@xylem.com
Xylem
Washington, DC, USA

Paul Sangster
Paul.Sangster@xylem.com
Xylem
Washington, DC, USA

Radhika Upadrashta
Radhika.Upadrashta@xylem.com
Xylem
Washington, DC, USA

TJ OConnor
toconnor@fit.edu
Florida Institute of Technology
Melbourne, FL, USA

ABSTRACT

A myriad of security and privacy threats have accompanied the rapid proliferation and widespread adoption of Internet-of-Things (IoT) devices. IoT vendors often justify this challenge by citing the difficulty of securing resource-constrained embedded devices. However, the monolithic nature of IoT devices introduces the opportunity to leverage mandatory and role-based access control to create a comprehensive yet flexible access control design. We identify that the TOMOYO and CaitSith Linux Security Modules offer opportunities to implement practical access control policies for IoT but there is a dearth of publications in this area. In the following work, we design and implement an access control approach for a Linux-based IoT gateway. Specifically, we explore how to reduce the attack surface by defining the policies to restrict the device to the minimum required behaviors. We empirically evaluate our approach's ability to withstand a network penetration test. Through these efforts, we demonstrate the capacity of Linux security modules to enforce access control on resource-constrained IoT devices.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; **Access control**; **Embedded systems security**; *Domain-specific security and privacy architectures.*

KEYWORDS

Access Control Policies, Internet-of-Things Security, Embedded Device Security, Linux Security Module, CaitSith, TOMOYO

ACM Reference Format:
Manuel Bessler, Paul Sangster, Radhika Upadrashta, and TJ OConnor. 2024. Hardening the Internet of Things: Toward Designing Access Control For Resource Constrained IoT Devices. In *Workshop on Cyber Security Experimentation and Test (CSET 2024), August 13, 2024, Philadelphia, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3675741.3675744

1 INTRODUCTION

IoT device gateways serve a critical role in IoT ecosystems. These gateways broker the on-demand and heartbeat traffic from individual devices to cloud-based content-delivery networks [4]. Typically, they are dual-homed - connected to the Internet over WiFi and connected to various devices over lightweight protocols like ZigBee, Z-Wave, or Bluetooth LE [17]. While commonly implemented on the Linux operating system, these devices often fail to enact well-designed security mechanisms, enabling pervasive attacks against devices [3, 11, 18–20, 26]. In a similar problem space, insecure WiFi routers connect many of our WiFi-enabled devices to the Internet. These problem spaces suffer the same flawed designs and implementations, with poorly secured services and default credentials [3, 19, 26]. However, IoT gateways and WiFi routers' widespread reliance on the Linux operating system introduces unique opportunities. Specifically, we identify that the limited nature of these devices offers the opportunity to establish well-designed practical access control lists that restrict the device to a minimum set of necessary behaviors.

To understand these opportunities, we briefly explore how characteristic access control delivers expressiveness and flexibility to mandatory access control. Mandatory access control mechanisms enforce the principle of least privilege by restricting access to a limited set of subjects. For example, TOMOYO's *pathname-based policy* enforcement mechanism enables access to resources based on their explicit call chains [10]. While mandatory access control's *all or nothing* strategy is effective for systems requiring strict control, users often disable in practice for usability [23]. However, recent work has extended TOMOYO with practical predefined access control lists that define the actions that subjects may or may not perform on objects. For example, *the Apache webserver may never execute a bash shell or open a raw socket.* Specifically, the CaitSith Linux Security Module extends TOMOYO by defining these characteristic action lists for rule-based access control [8]. Developing extensive characteristic action lists may prove a prohibitive

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CSET 2024, August 13, 2024, Philadelphia, PA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0957-9/24/08.
https://doi.org/10.1145/3675741.3675744

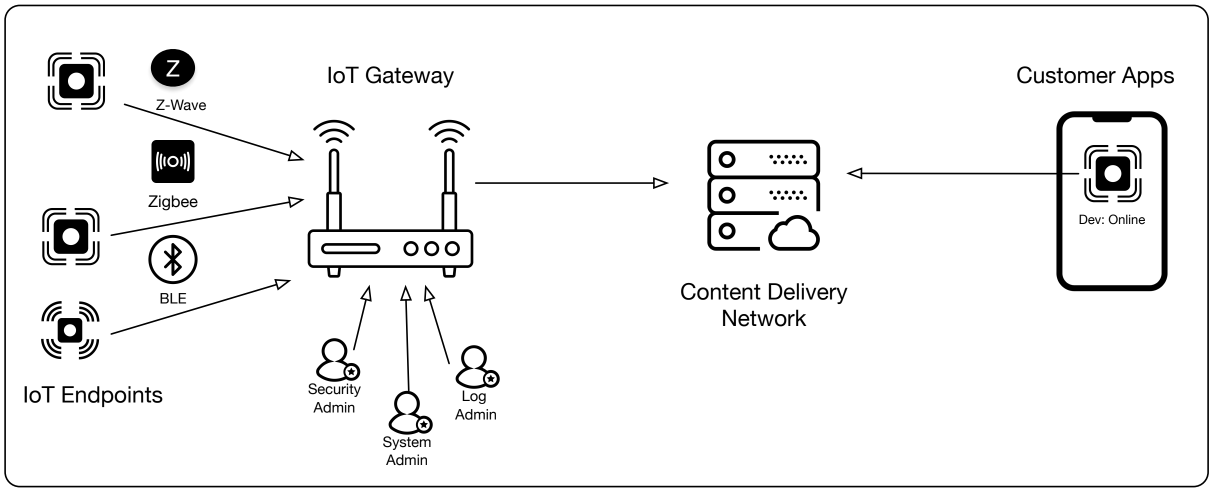

Figure 1: Our access control approach protects an IoT gateway connected to various IoT endpoints, a content delivery network that delivers customer data, while still remaining accessible to different levels of administration.

task for desktop systems or servers running a myriad of applications and services [7]. However, this approach is pragmatic for the monolithic nature of IoT gateways or WiFI routers with limited functionality.

This paper makes the following contributions:

(1) We explore how resource-constrained IoT devices can leverage comprehensive access control designs to overcome security challenges.
(2) We apply this access control design to a production IoT gateway and empirically evaluate our approach's ability to withstand a network penetration test.

Organization: Section 2 investigates previous work related to access control and explores our attacker threat model. In Section 3, we provide an overview of our approach. Section 4 examines how we design characteristic access control in a defense-in-depth approach. Section 5 proposes an experiment to explore the impact of our approach. Section 6 explores the results of our experiment. Section 7 summarizes our conclusions.

2 BACKGROUND

Our approach is motivated by the pervasive insecurity of IoT devices. In this section, we provide the background behind why attackers succeed against IoT devices and describe the threat model for our defensive approach.

2.1 Motivating Example

Nearly a decade ago, the Russian *Insecam* website rose to popularity by providing illicit access to poorly secured IoT camera systems [26]. During its tenure, the site granted access to tens of thousands of camera systems. Xu et al. measured Insecam for 18 days in November 2017 and identified 28,386 unique, active cameras from 31 timezones, 136 countries, and 25 manufacturers [26]. Xu attributed the illicit access to predominately default credentials and remotely exposed unauthenticated services like telnet and ssh [26]. The *Mirai Botnet* echoed these findings the following year as it gained access to 65,000 IoT devices within 20 hours before ultimately infecting over 200,000 devices [3]. Despite widespread knowledge of the problem, these design and implementation flaws continue to exist in IoT due to the growing demand for inexpensive devices [19]. Either by malice or incompetence, IoT devices often enable systemic backdoors that attackers later discover. The pervasive problem has ultimately forced the US White House administration in July 2023 to propose a *U.S. Cyber Trust Mark* to educate and inform consumers about the security of IoT devices [25].

2.2 Threat Model and Assumptions

Our threat model assumes the attacker's goals include obtaining unauthorized access to deny service [3], access confidential materials [19], or pivot towards co-located devices [16]. To achieve this goal, we imagine an attacker will initially gain access to a limited user account, such as the default *www* account responsible for establishing an Apache webserver. We imagine an attacker must escalate privileges to access confidential materials, tamper with logs, or disable services. We consider an attacker will leverage a privilege escalation technique such as *PwnKit* (CVE-2021-4034) vulnerability [12]. We assume that devices will likely be running a legacy operating system and prone to several privilege escalation attack vulnerabilities. Our trusted computing base (TCB) includes the administrator account and the kernel. We do not protect against a malicious administrator that can disable Linux Security Modules (LSMs). We consider the attestation of the Linux kernel as important orthogonal work [22].

3 OVERVIEW

This section outlines an overview of the problem space, identifying the key ideas and challenges.

3.1 IoT Gateway Architecture

Our design is motivated by our real need to secure an IoT gateway in practice. As depicted in Figure 1, our gateway is an intermediary between various IoT endpoints and a customer-facing content delivery network. Further, the IoT gateway must remain accessible for different levels of remote administration. The gateway is running on a resource constrained embedded device on a ARMv5 processor, a little under 500MHz, and 128/256M RAM. There is no external security solution in the IoT architecture, so a lot of thought was given to implement security controls during the design of the Gateway.

The IoT Gateway is running a Yocto Project-based Linux, using TOMOYO and CaitSith LSMs. The defense-in-depth design can be seen in different layers of the appliance right from the choice of the Linux distribution, continued through file system hardening, and extended security through LSM policies.

3.2 Challenges and Key Ideas

Future Vulnerability Prevention: Any access control approach for IoT must consider that users may elect to avoid applying future firmware updates. While the device may be secure from vulnerabilities today, the access control policies should protect the device even in the event of future exposures. For example, a well-designed IoT access control policy should be capable of securing against a future local privilege escalation technique against the operating system. In Section 4.5, we explore how our defense-in-depth strategy enables us to deliver a restricted shell through a public key authentication, where the user (even admin) is severely limited in their ability to perform destructive actions. And in Section 5, we demonstrate this by exploring how the system responds to the PwnKit technique.

Pragmatic Approach: In practice, users often disable comprehensive access control policies as usability competes for security [2]. A well-designed access control policy should permit the set of necessary actions while restricting unnecessary or malicious actions [1, 5]. In Section 4.3, we explore how TOMOYO's concept of process call chain domains introduces unique and pragmatic opportunities for IoT.

Memory and CPU Resource Constraints: The IoT Gateway runs on a resource-constrained embedded architecture. Any feasible access control must consider how the rules are efficiently stored and processed. In Section 4.3, we explore how TOMOYO's implementation as a Linux Security Module performs efficient policy checking and overhead by enforcing on a per-domain basis. Further, in Section 4.4 we see how CaitSith and TOMOYO complement each other.

4 DESIGN

In the following section, we expand on our design and implementation, exploring the base operating system, system hardening, mandatory access control scheme, access control schemes, and additional security controls of our IoT gateway.

4.1 L1: Base Operating System

We selected the *Yocto Project* hardware-independent open source project builder to construct the Linux base operating system for our IoT gateway [28]. Widely used in resource-constrained embedded devices, Yocto Project provides tools to customize and build the Linux environment. Yocto Project expands the *OpenEmbedded* build system (containing BitBake and OpenEmbedded Core) with the *Poky* reference embedded distribution system, which provides the build instructions (also called recipes) and layers (collection of related recipes) [13, 28]. Different layers can be used to logically separate the features of an application. This allows for customization of the application.

4.2 L2: System Hardening

After defining and building the base operating system, we applied a series of security changes to harden the system. First, we removed unnecessary packages, such as the X11 packages (as there is no graphical interface in embedded devices), using the *Yocto BitBake* tool. In addition to reducing the threat surface, removing unnecessary packages frees already-constrained system resources such as memory and storage. Further, we prevented modifying or backdooring system binaries by implementing a *SquashFS* read-only filesystem. Marking the rootfs as read-only prevents unintended modification or introduction of configuration files, such as persistent ssh-keys. In addition to mounting the rootfs as read-only, we mounted the /tmpfs filesystem as *noexec, nodev, nosetuid* and kept it small in size. As /tmpfs is the only *writable* directory, these immutable permissions would restrict users using it to download and execute malicious binaries or interact with the kernel through special devices. Finally, we leveraged the Yocto *cve-check* tool to warn of any vulnerabilities introduced by other packages or software [24]. This helpful tool aids in identifying software supply-chain vulnerabilities that emerge from software components and their dependencies. Finally, we selected and enabled a Linux Security Module (LSM). The LSM framework is useful to Linux distributions as it provides extensions for various security checks, and can be selected at build time using the CONFIG_DEFAULT_SECURITY argument or at boot time using the "security=…" kernel argument. Major LSMs include AppArmor, SELinux, Smack, and TOMOYO [6, 23]. As of Linux kernel 4.19, only one major LSM can be enabled at a time, other than the "capabilities" module which is always included in the distribution. We selected and enabled TOMOYO. TOMOYO offers simplicity in policy and efficient resource utilization over the more complex SELinux, which presents a challenge for resource-constrained IoT devices. In the next paragraphs, we explore these benefits and examine applying to an IoT gateway.

4.3 L3: TOMOYO Mandatory Access Control

TOMOYO enables our first step towards comprehensive access control by implementing mandatory access control mechanisms. Specifically, TOMOYO declares the behavior of each process and the resources needed to achieve that behavior [14]. With the TOMOYO LSM enabled, the kernel restricts each process to a particular behavior and resource chain. TOMOYO is built on the concept of a *domain*. TOMOYO defines domains by their process execution tree execution sequence [9]. For example, the domain for a user on the local system executing the *ls* command to list a directory would be:

```
<kernel>    /bin/sh    /bin/ls
```

TOMOYO automatically constructs these domains, enabling the administrator to define the access mode of the domains using the *Policy Editor*. By identifying this *process call chain*, we can detect incorrect behavior that deviates from the correct call chain [14]. TOMOYO also gives us fine-grained control to restrict elevating privileges to effective User ID(UID)=0, create per-application firewall and restrict services and operations through Access Control Lists (ACLs), and control file system access to prevent read/write to */dev* filesystem[15]. To do these, we leveraged TOMOYO's self-learning mode to build all the domains our IoT gateway would exercise in the normal intended behavior. While TOMOYO exclusionary listing may not be practical for desktop systems or servers running many different applications, it offers a pragmatic solution for monolithic IoT devices that perform limited behaviors. After learning all the behaviors of an IoT device, we took a snapshot, which defines the kernel mandatory access control policy. TOMOYO then restricts further learning to prevent bypassing the policy and directly loads the policy into the kernel to avoid runtime modification that would bypass the policy.

TOMOYO is an appropriate solution for the resource constraints of IoT devices. In contrast, SELinux and Smack require storing the policy in the inode's extended attributes. The overhead introduced by the granularity of SELinux policy enforcement adds undesirable memory and storage demands. Policy checking under SELinux overwhelms IoT device CPU and memory utilization, slowing the device to unacceptable levels. Conversely, TOMOYO controls enforcement on a per-domain basis. Domains without rules inherit the parent's policy. This design allows writing policy incrementally by securing the critical subsystems before defining additional rules.

Consider the following example to understand the expressiveness of a TOMOYO rule. Examine the case where we wanted to enable a local login shell but restrict a remote shell derived from a SSH session. Under TOMOYO, the local and remote login shells are entirely different domains. The domain for a sudo session via an SSH login shell on our IoT gateway is:

<kernel> /usr/sbin/sshd /bin/sh /usr/bin/sudo

Conversely, the domain for a sudo session running under a local login shell is:

<kernel> /bin/sh /usr/bin/sudo

The distinct granularity of this approach enabled us to treat remote users uniquely and differently than local users, allowing us to apply aggressive restrictive lists to remote users who should only interact with devices in minimal behaviors.

4.4 L4: CaitSith Rule-Based Access Control

The CaitSith LSM compliments TOMOYO by establishing specific operations as keys for checking policy permissions [8]. While derived from TOMOYO, CaitSith rule-based access control policy syntax differs. While TOMOYO policy describes what a domain can do, CaitSith policy describes ACLs on operations through allow and deny criteria for subject attributes (for example, userid, groupid, process). to protect various resources in kernel. While TOMOYO's rules define policies for the subjects (process call trees), CaitSith's policies restrict operations such as access to files, executing binaries, and binding to network sockets. This access control approach extends our policy to restrict IoT devices to a limited set of behaviors.

```
110 acl inet_dgram_bind port=161
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/snmpd"
    200 deny

110 acl inet_dgram_bind port=162
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/snmpd"
    200 deny

111 acl inet_stream_listen task.exe="/usr/sbin/snmpd"
    audit 1
    100 allow port=161
    101 allow port=705 ip=127.0.0.1
    200 deny

112 acl inet_dgram_bind task.exe="/usr/sbin/snmpd"
    audit 1
    100 allow port=161
    100 allow port=162
    100 allow port=0 ip=0.0.0.0
    200 deny

113 acl inet_stream_listen port=705
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    2 deny ip!=127.0.0.1
    100 allow task.exe="/usr/sbin/snmpd"
    200 deny
```

Listing 1: CaitSith Policy To Prevent SNMP LoL Attacks.

Since TOMOYO is enabled as a major LSM and CaitSith is built as an external module in our design, CaitSith is loaded last and comes last in the order of execution of the rules.

To illustrate how our policy design defends our IoT gateway, consider the case of the Simple Network Management Protocol (SNMP) protocol that facilitates the exchange of management information between networked devices. Under IoT, SNMP shares device information, networking configuration and enables event notifications to provide seamless setup and maintenance of devices. Usually, the kernel init manager should start the SNMP daemon to read only through a configuration file setting. However, a malicious user could bypass this and launch the SNMP daemon with a separate configuration file to infiltrate or exfiltrate data in a *living-off-the land* (LoL) attack. Consider how we leverage CaitSith to define a generic policy that restricts the behavior of the SNMP daemon. Listing 1 depicts CaitSith rules that restrict SNMP to binding the appropriate UDP ports 161 or 162. Further, the policy establishes that only the kernel (uid == 0, euid == 0) may start the SNMP daemon (/usr/bin/snmpd) and bind the ports. This strengthens authorization and integrity of the running processes, and restricts the service to logged-in users of the IoT gateway.

4.5 L5: Finalizing Defense-in-Depth

We finalized our access control policies by adding a series of defense-in-depth measures that leveraged CaitSith to reduce the threat surface:

- We established a CaitSith rule to restrict turning off the firewall or modifying the firewall rules. Our firewall prevents binding new ports, accepting new connections, or opening connections beyond the minimum necessary communication.
- We enabled an SSH service and restricted logins to SSH Certificate Authentication issued by a trusted Certificate Authority (CA). The SSH service creates on-demand temporary accounts based on the information from the certificate. We remove accounts when the user logs out. There is no root login or *sudo* access.
- We protected our IoT gateway configuration database from all users (remote or local) by leveraging CaitSith to restrict only the init-manager to start our gateway daemon and read the configuration database.
- Only authorized system services on the IoT gateway can read the configuration files with sensitive information, passwords, and keys.

5 EXPERIMENT

In the previous section, we described our approach to applying comprehensive and flexible access control in the context of a defense-in-depth strategy for an IoT gateway. In this section, we empirically evaluate the security of this approach by answering the following questions.

RQ1 (*Confidentiality*): Is our approach effective in restricting confidential files (configurations, private certificates, sensor data) from unauthorized access?

RQ2 (*Integrity*): Is our approach effective in maintaining the trustworthiness of configuration files, services, and sensor data by limiting modification to a minimum necessary set of services initiated by the kernel?

RQ3 (*Availability*): Is our approach effective in preventing disabling critical services, sensor data delivery, and security policies?

Experiment Setup and Scope: After securing the IoT gateway as described in Section 4, we conducted a network penetration test on our system by creating SSH certificates for a privileged account and a non-privileged account. The service reflects the same type of vulnerabilities described in our motivating example in Section 2.1. During the initial black-box testing, the device was not visible because of the firewall restrictions that only allowed access from the Content Delivery Network (CDN). We then logged in from the CDN and performed a penetration test against the device, trying to compromise the system's confidentiality, integrity, and availability. While the device is connected to the CDN, we declared only the IoT gateway to be in scope. We did not assess the communications security of the channel or CDN endpoints.

	Limited User	**Admin**
Confidentiality		
Read VPN configuration	✗	✗
Read VPN private certificate	✗	✗
Dump network traffic	✗	✗
Enumerate firewall policies	✗	✗
List open tcp/udp ports	✗	✗
Read sshd configuration	✗	✗
Read snmp configuration	✗	✗
Read logs of the services	✗	✗
Read /etc/shadow	✗	✗
List contents of root directory	✗	✗
List sudo enabled binaries	✗	✗
Integrity		
Modify /etc/shadow	✗	✗
Modify /etc/passwd	✗	✗
Modify gateway database	✗	✗
Modify gateway configuration	✗	✗
Modify SNMP configuration	✗	✗
Modify sshd configuration	✗	✗
Establish netcat backdoor access	✗	✗
Download and execute pwnkit	✗	✗
Compile pwnkit.c	✗	✗
Escape restricted shell	✗	✗
Enable SUID bit on binary	✗	✗
Availability		
Disable SNMP daemon	✗	✗
Disable sshd dameon	✗	✗
Disable VPN	✗	✗
Disable gateway service	✗	✗
Halt system	✗	✗

Table 1: Results of our network penetration test

6 RESULTS

The following section discusses how our access control approach defended the system. Table 1 provides a high-level overview of our penetration test activities on the IoT gateway.

6.1 RQ1: Confidentiality

Our IoT gateway passes sensitive information between the IoT gateway and a CDN. To compromise the system's confidentiality, we attempted to read the VPN configuration that connects the service to the CDN. However, CaitSith restricted read access to only the gateway service daemon for the VPN configs and certificates. While we attempted to enumerate the firewall policies and log files of the services to examine other potential opportunities, we also could not due to restrictive rules. We then looked for different ways to escalate our access by reading the sshd, SNMP configurations, the user account files and examining the root directory. However, both accounts (privileged and non-privileged) lacked read access to these directories and files.

6.2 RQ2: Integrity

Next, we examined opportunities to modify the system by changing configuration files, user accounts, services, and even tampering with sensor data. However, these files were restricted to only the immutable service daemon. So we looked for opportunities to escape the restricted shell. First, we explored opening a Netcat listener backdoor on a TCP port. However, we could not bind any TCP ports due to the firewall policy. So we explored using a reverse netcat listener, which also failed due to the restrictive firewall policies. We then attempted several well-known privilege escalation techniques by downloading the linPEAS script that automatically searches for possible paths to escalate privilege [21]. However, the tool relies on executing several commands not enabled in the restricted environment and fails to enumerate any privilege escalation paths. Next, we tried to understand if a user could escalate using a tool like *pwnkit* that exploits the CVE-2021-4034 vulnerability. However, we could not download and execute any binaries due to *noexec* mount. Further, we could not compile a source code version of the tool because the environment lacked a compiler or build utilities, and we could not introduce any new binary tools to compile.

6.3 RQ3: Availability

Finally, we attempted to disable services and halt the system. However, our CaitSith ruleset restricted enabling or disabling any critical services to the init manager. Further, the accounts did not have the rules to execute any *systemd* tools that interface with the init manager. Using TOMOYO and CaitSith policies correctly prevented users from executing most of the Linux commands they are not authorized to. This prevented privilege escalation attempts during the pen test.

7 CONCLUSION

This work examined our approach to developing a secure IoT gateway that leverages the TOMOYO and CaitSith Linux Security Modules to implement comprehensive access control policies. We have explored how the monolithic nature of IoT enables the opportunity to restrict the IoT gateway to a minimum necessary set of device behaviors. Combining mandatory and role-based access control approaches, we have facilitated fine-grained user and application restrictions to defend the environment. Further, we have empirically evaluated this approach by performing a network penetration test to understand the strength of the design. We demonstrate that our comprehensive access control design is a feasible approach to defend the threat surface of IoT devices. With the ability to significantly reduce the threat surface, our pragmatic approach offers a promising opportunity for resource-constrained IoT devices.

ARTIFACTS

Sample Yocto recipe files, TOMOYO and CaitSith policy files are available for research and education at [27]. Using these in a real-life solution could require an experienced person and a substantial amount of time.

ACKNOWLEDGMENTS

Thanks to Mark Scandariato (Embedded Firmware Engineer) for working on the development of the solution. Thanks to Gabriel Gwartney (Application Security Engineer) and Baxter McCoy (Product Security Engineer) for their help in testing the solution. Thanks to Dr. Kenneth Crowther (Product Security Leader) for reviewing the paper and for providing overall guidance.

DISCLAIMER

Xylem provides no representation, warranty, or guaranty of outcome based on any statements or perceived recommendations in this paper. Use or implementation of any statements in this paper are at the user's sole risk.

REFERENCES

[1] Ahmed Alhazmi, Khulud Alawaji, and TJ OConnor. 2022. MPO: MQTT-Based Privacy Orchestrator for Smart Home Users. In *Computers, Software, and Applications Conference (COMPSAC)*. IEEE, Virtual Event, 988–993.

[2] Ahmed Alhazmi, Ghassen Kilani, William Allen, and TJ OConnor. 2021. A replication Study for IoT Privacy Preferences. In *Conference on Omni-Layer Intelligent Systems (COINS)*. IEEE, Virtual Event, 1–8.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *USENIX Security Symposium (USENIX Security 17)*. USENIX, Vancouver, BC, 1093–1110.

[4] Daniel Campos and TJ OConnor. 2021. Towards Labeling On-Demand IoT Traffic. In *Cyber Security Experimentation and Test (CSET)*. USENIX, Virtual Event, 49–57.

[5] Parth Ganeriwala, Anubhav Gupta, Daniel Campos, Siddhartha Bhattacharyya, TJ OConnor, and Adolf Dcosta. 2023. Modeling Internet-of-Things (IoT) Behavior for Enforcing Security and Privacy Policies. In *Computing Conference*. Springer, London, UK, 1451–1473.

[6] Tao Guo, Puhan Zhang, Hongliang Liang, and Shuai Shao. 2013. Enforcing multiple security policies for android system. In *2nd International Symposium on Computer, Communication, Control and Automation*. Atlantis Press, Singapore, 165–169.

[7] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. 2013. A capability-based security approach to manage access control in the internet of things, In Mathematical and Computer Modelling. *Mathematical and Computer Modelling* 58, 5-6, 1189–1205.

[8] T. Handa. 2012. CaitSith: a new type of rule based in-kernel access control. http://kernsec.org/files/CaitSith-en.pdf. In *LinuxCon North America*. The Linux Foundation, San Diego, CA, 1–117.

[9] T. Harada. 2007. TOMOYO Linux: A practical method to understand and protect your own Linux box. https://ja.osdn.net/projects/tomoyo/docs/PacSec2007-en-no-demo.pdf. In *PAC SEC*. Dragos Ruiu, Tokyo, JP, 1–34.

[10] T. Harada. 2008. TOMOYO Linux for Secure Embedded. https://osdn.net/projects/tomoyo/docs/fosdem2008.pdf/en/18/fosdem2008.pdf.pdf. In *Free and Open Source Software Developers European Meeting (FOSDEM)*. Université libre de Bruxelles, Brussels, BE, 1–47.

[11] Blake Janes, Heather Crawford, and TJ OConnor. 2020. Never Ending Story: Authentication and Access Control Design Flaws in Shared IoT Devices. In *IEEE Security and Privacy SafeThings Workshop (SafeThings)*. IEEE, Virtual Event, 104–109.

[12] Raphaël Khoury. 2022. A Taxonomy of Software Flaws Leading to Buffer Overflows. In *International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, Guangzhou, CN, 1–8.

[13] A. Murray. 2021. Yocto Security Hardening: CVEs. https://www.thegoodpenguin.co.uk/blog/yocto-security-hardening-cve/.

[14] NTT DATA Corporation. 2023. About TOMOYO Linux. https://tomoyo.osdn.jp/about.html.en. Accessed: July 4, 2024.

[15] NTT DATA Corporation. 2023. TOMOYO Linux functionality comparison table. https://tomoyo.osdn.jp/comparison.html.en. Accessed: July 4, 2024.

[16] TJ OConnor, William Enck, W Michael Petullo, and Akash Verma. 2018. PivotWall: SDN-based information flow control. In *Proceedings of the Symposium on SDN Research*. ACM, Los Angeles, CA, 1–14.

[17] TJ OConnor, William Enck, and Bradley Reaves. 2019. Blinded and Confused: Uncovering Systemic Flaws in Device Telemetry for Smart-Home Internet of Things. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. ACM, Miami,FL, 140–150.

[18] TJ OConnor, Dylan Jesse, and Daniel Camps. 2021. Through the Spyglass: Toward IoT Companion App Man-in-the-Middle Attacks. In *Cyber Security Experimentation and Test (CSET)*. USENIX, Virtual Event, 58–62.

[19] TJ OConnor, Dylan Jessee, and Daniel Campos. 2023. Towards Examining The Security Cost of Inexpensive Smart Home IoT Devices. In *Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, Turin, IT, 1293–1298.

[20] Stian Olsen and TJ OConnor. 2023. Toward a Labeled Dataset of IoT Malware Features. In *Computers, Software, and Applications Conference (COMPSAC 2023)*. IEEE, Torino, Italy, 924–933.

[21] Carlos Polop. 2023. LinPEAS - Linux Privilege Escalation Awesome Script. https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS. Accessed: July 4, 2024.

[22] Red Hat, Inc. 2023. Enhancing security with the kernel integrity subsystem. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/kernel_administration_guide/enhancing_security_with_the_kernel_integrity_subsystem. Accessed: July 4, 2024.

[23] Z Cliffe Schreuders, Tanya McGill, and Christian Payne. 2011. Empowering end users to confine their own applications: The results of a usability study comparing SELinux, AppArmor, and FBAC-LSM. In *Transactions on Information and System Security (TISSEC)*. ACM, New York, NY, 1–28.

[24] The Kernel Development Community. 2024. Linux Security Module Usage. https://www.kernel.org/doc/html/latest/admin-guide/LSM/index.html. Accessed: July 4, 2024.

[25] The White House. 2023. Biden Harris Administration Announces Cybersecurity Labeling Program for Smart Devices to Protect American Consumers. https://www.whitehouse.gov/briefing-room/statements-releases/2023/07/18/biden-harris-administration-announces-cybersecurity-labeling-program-for-smart-devices-to-protect-american-consumers/. Accessed: July 4, 2024.

[26] Haitao Xu, Fengyuan Xu, and Bo Chen. 2018. Internet protocol cameras with no password protection: An empirical investigation. In *Passive and Active Measurement (PAM)*. Springer, Berlin, DE, 47–59.

[27] Xylem.com. 2024. artifacts. https://www.xylem.com/siteassets/support/case-studies/case-studies-pdf/tomoyo-caitsith-recipes.tar.zip.

[28] Yocto Project. 2024. Yocto Project. https://www.yoctoproject.org/. Accessed: July 4, 2024.