Mode Switching for Secure Edge Devices

Michael Riegler[1], Johannes Sametinger[2], Christoph Schönegger[3]

[1,2] LIT Secure and Correct Systems Lab and Department of Business Informatics
Johannes Kepler University Linz, Austria
{michael.riegler,johannes.sametinger}@jku.at
https://www.jku.at/en/lit-secure-and-correct-systems-lab
https://www.se.jku.at

[3] ENGEL AUSTRIA GmbH
christoph.schoenegger@engel.at
https://www.engelglobal.com/

1[0000−0002−6937−986X] 2[0000−0002−0637−6602]

Abstract. Many devices in various domains operate in different modes. We have suggested to use mode switching for security purposes to make systems more resilient when vulnerabilities are known or when attacks are performed. We will demonstrate the usefulness of mode switching in the context of industrial *edge devices*. These devices are used in the industry to connect industrial machines like cyber-physical systems to the Internet and/or the vendor's network to allow condition monitoring and big data analytics. The connection to the Internet poses security threats to *edge devices* and, thus, to the machines they connect to.
In this paper (i) we suggest a multi-modal architecture for *edge devices*; (ii) we present an application scenario; and (iii) we show first reflections on how mode switching can reduce attack surfaces and, thus, increase resilience.

Keywords: Mode switching · Edge device · Security · Linux · systemd · Ansible.

1 Introduction

Manufacturers will increasingly become service providers and use condition monitoring, and big data analytics for predictive maintenance [25]. *Edge devices* connect physical devices like machines, robots, and sensors over various protocols like MQTT, OPC UA, and others to the virtual world. They are widely used in Cyber-physical systems (CPS) and Internet of Things (IoT) systems, and their use will increase in the next decade [24]. *Edge devices* provide computing power for lightweight devices and other facilities on-site and build a bridge between operational technology (OT) and information technology (IT), typically in the cloud. They allow machine-to-machine communication, remote monitoring and control across several locations and increase availability and efficiency.

Edge devices provide load balancing, low latency, and service continuity in case of connection failures, optimize the network bandwidth to the cloud, and process the workload close to its occurrence [15]. Devices increasingly get interconnected, enhancing systems' complexity and unleashing threats and vulnerabilities. In the year 2021, vulnerabilities in *Log4J*, *Kaseya* and *Solar Winds* have affected thousands of companies. Cyberattacks like the one against *Colonial Pipeline* can lead to loss of productivity, business continuity problems, financial losses, and damage of reputation. According to the Allianz Risk Barometer [1], cyber incidents are the most important business risk in 2022.

Time is essential when a vulnerability becomes known. It typically takes a while until an update will be available. During that time, attackers can write exploits and attack systems. Workarounds are sometimes provided to mitigate known vulnerabilities. Nevertheless, manual changes are time-consuming and, if performed hastily, involve the danger of disrupting operations. In the worst case, services have to be completely stopped or shut down.

We have presented how multi-mode systems can provide resilient security in Industry 4.0 [18] and a web server case study using a multi-purpose mode switching solution to overcome vulnerabilities in [19]. Additionally, we have investigated how modes can secure deliberately insecure web applications [17]. In this paper, we will investigate the security of *edge devices* and reflect on how automatic and manual mode switches can reduce attack surfaces.

In Section 2, we present the idea of mode switching and its use in several domains. In Section 3, we describe common methods to secure *edge devices*. We present our considerations about mode switching in *edge devices* in Section 4. Automation of configuration management of modes and deployment are shown in Section 5. We discuss our results in Section 6 and draw conclusions in Section 8.

2 Mode Switching

We have provided first findings of a systematic literature review about mode switching from a security perspective in [16]. In general, modes are used to provide flexible adjustment of behavior, real-time adaptation, and complexity management. They have already been used in domains like automotive [2], aviation [22] and energy [26]. Modes divide and manage complexity, have specific configurations, and consist of specific behaviors. For example, nuclear power plants have multiple modes for power operation, startup, hot standby, hot shutdown, cold shutdown, and refueling [26]. They automatically change their modes and may even be completely switched off if parameters exceed or fall below critical thresholds [21]. Special accident and emergency systems and a degraded mode of operation [7] provide resilience and a better understanding of the system's state if there is any kind of malfunction. However, mode switching is not limited to safety precautions. Modes can also be used to control functionality and to increase security [20]. A web server case study [19] has shown that mode switching has reduced known risks in that specific scenario in 98.9% of the time and has shortened the window of exposure from 536 days to 8 days.

3 Securing Edge Devices

Making *edge devices* more secure begins with the key concepts of the security triad *CIA*: confidentiality, integrity, and availability. According to ISO/IEC 27000:2009 [11], authenticity, accountability, non-repudiation, and reliability are also important. Preventive, detective, and corrective security controls will help to minimize security risks and to recover faster whenever something bad happens. There should be multi levels of protection (defense in depth) for all system parts: cloud, network, device, application, and data. One compromised part should not affect the entire system. Following a *Secure Software Development Live Cycle* (sSDLC) will facilitate delivering effective and efficient systems during lifetime [4,10]. Several guidelines and frameworks [5,10,13,14] exist to make IoT devices more secure. They suggest physical controls like a *Trusted Platform Module* (TPM), mechanical locks, and minimized external ports. TPM goes hand in hand with a *Trusted Execution Environment* (TEE), secure boot, a protected *operating system* (OS), and secure updates. McCormack et al. provide an architecture for trusted edge IoT security gateways and recommend periodic remote attestation with TPM, to control that there are no fraudulent manipulations [12]. It is state-of-the-art to encrypt all communication to the outside of an IoT device. In addition to regulatory requirements and policies, technical and logical controls should also be implemented: authentication, access controls, a firewall, and antivirus software. Each device should have its own client certificate to communicate with manufacturers' cloud services. Even if compromised, only single devices are affected and can be blocked on the server-side, if necessary. Updates should be signed and verified, risky legacy protocols should be avoided, and open ports minimized.

Cejka et. al have compared *Amazon Web Services* (AWS) IoT Greengrass, Microsoft Azure IoT Edge, and a self-implemented framework for secure *edge devices* and distributed control of critical infrastructure [3]. They recommend filtering and monitoring communication for anomaly detection and reacting to deviations with countermeasures.

In our sample scenario, we use *edge devices* to connect industrial machines at the customers' site to the Internet and to allow communication with the machines' manufacturer. Typically, *edge devices* will be delivered to the customer's factory or machine. However, sometimes devices get stolen and even manipulated. Therefore, the hard disk is encrypted, and the device provides only limited initial functionality like configuring the network settings to onboard the device. Onboarding means that the device is registered with the manufacturer and gets a device certificate that activates the device's full functionality, which persists even if later the connection gets lost. Thus, once the device has successfully established a connection with the manufacturer's cloud service, it sends an onboarding request. Then the customer can authenticate at the manufacturer's cloud service to view the request and get a security token to assign the *edge devices* to the customer, similar to [27].

4 Edge Device Modes

In this section, we provide further details about the design and implementation of *edge device* modes. Figure 1 gives an impression about a few sample modes and their services and configurations. A system and its components can be in multiple modes. Initially, an *edge device* is in *factory mode*. After successful onboarding, it switches to the *onboarded mode*. In our sample scenario, *edge devices* are online most of the time. They check for updates in regular time intervals, typically once a day. If an update check was not possible for a specific time period, e.g., for x days, the device is considered to be outdated. A mode switch to *outdated mode* leads to limited functionality for security reasons. Thus, the onboarded mode has two sub-modes: *updated mode* and *outdated mode*. Modes can extend other modes similar inheritance in object-oriented programming (OOP).

We use modes to protect the system and its services from vulnerabilities found in the meantime. We consider it bearable that a system that has not been online for a predefined time period makes updates first. The system itself periodically checks if there is an update. After the successful update or when the last successful update check is younger than x days, the *updated mode* becomes active, which provides all services, like condition monitoring, big data analytics for predictive maintenance, remote support, and others. Additionally, we plan to analyze log files to react to specific events like denial-of-service (DoS) attacks.

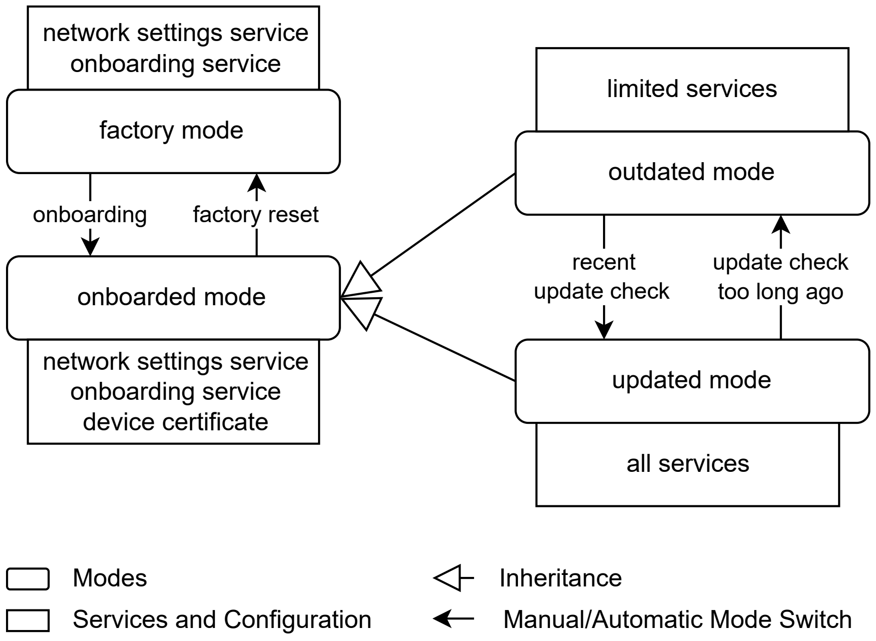

Fig. 1. Edge Device Modes

4.1 Implementation

For implementing the concept of modes on common *edge devices*, we have extended the system and service manager *systemd*[1], which is used in most Linux operating systems. We use *systemd* targets [8] to specify modes. They are similar to *SystemVinit* run levels [23], which were used in previous systems. A target is specified with a unit-file and combines several services. It is possible to define a default target as well as hard and soft dependencies to other modes, which should be started before. Usually the `multi-user.target` (on non-graphical systems) or the `graphical.target` are the default targets. We can switch to the mentioned mode by running `systemctl isolate mode.target`. Nevertheless, it only works if the mode meets the conditions. Services, like to define the network settings, for onboarding or for specific software containers, have their own unit-file and can be attached to one or multiple modes in their `[Install]` section with the command `WantedBy=mode.target`. Since release v250, the `factory-reset.target` has become available. Obviously, the developers think in a similar way. Listing 1.1 shows the `updated-mode.target`. It is started after the `onboarded-mode.target` only when a specified file exists. *Systemd* only supports some basic condition and assertion checks, before a target or service becomes active. For example, we can only check whether a file exists (`ConditionPathExists`), but we cannot check whether a certificate is valid, or whether the device is online. Therefore, we consider these parts in our *ModeSwitcher* shell script. As of now, all mode switches are executed by this shell script. A cron job runs periodically and checks if there are any changes. If an *edge device* is not onboarded, it tries to resolve that. If the device certificate becomes invalid, a mode switch to the *factory mode* is triggered. The cron job performs the updates for the device, software repositories, and the device certificate. If the last successful update check was as long ago as predefined, a mode switch is triggered from *updated mode* to *outdated mode*, and several containers are stopped in order to reduce the attack surface. If the update was successful, the system switches back to *updated mode*. Every mode switch will be notified to the manufacturer's cloud service.

Listing 1.1. Systemd updated-mode.target

```
#/etc/systemd/system/updated-mode.target
[Unit]
Description=Updated Mode
Requires=onboarded-mode.target
Conflicts=
After=onboarded-mode.target
AllowIsolate=yes
#Start target only when file does exists
ConditionPathExists=/var/lib/edgedevice/up-to-date
```

[1] https://systemd.io

4.2 Fail2ban Sample Scenario

We are monitoring log files in our sample scenario with *fail2ban*[2], a Python-based *Intrusion Prevention Software* (IPS) that is mainly used to defend systems from brute-force attacks, e.g., by blocking IP addresses of attackers after several failed login attempts within a specific time frame. We can also specify multiple and less intrusive actions like sending an email to the administrator. It follows the rules of *event condition action* (ECA). New log entries get checked by a *filter* of regular expressions to extract interesting data. *Jail* configurations become relevant, if there is a match. These configurations specify services or attack scenarios to be monitored. They consist of the log file, the port or service, the *filter*, the time span and the maximum number of wrong attempts, the *actions* to be executed, and the ban time. For example, we can specify that after three failed SSH login attempts, the IP address of the possible attacker gets blocked, and the administrator will be informed by email. Then, after a ban time of 10 minutes, a new login attempt from that IP address will be allowed again. The *fail2ban* manual [6] and Hess [9] provide more details about these mechanisms. The standard configuration provides filters for *Apache*, *sshd*, *vsftpd*, *Postfix* and others. We have implemented additional filters, e.g., to detect (unsuccessful) terminal logins, (unsuccessful) SSH logins, port scans, HTTP errors, plugged-in LAN cables, attached USB devices, and lost connections. We imagine using *fail2ban* to send a notification to manufacturers' cloud services in case of abnormal behavior. In addition to that, we will use some of the *fail2ban* events as a source to trigger mode switches. For example, we envision switching from a normal to a denial-of-service (DoS) mode. The typical use case of *Fail2ban* is to block single IP addresses. However, if there is a DoS attack, it is not appropriate to block thousands of IP addresses. In this case, switching to a more secure mode is more practical, where this form of attack is blocked in general. Additionally, IP address ranges or countries could be blocked. Switching to another or degraded mode can reduce the attack surface and protect the system from further attacks. Another scenario is multiple wrong login attempts on the console. If that happens and the device is offline, we switch to the *factory mode* to prevent further attacks. We can react in the same way, if we detect anomalies regarding hardware attacks like sniffing, or port scans. In order to prevent the abusive use of insiders, administrative capabilities have to be limited, and all actions have to be logged and monitored. Procedural or administrative controls are needed to handle incidents and increase security awareness [13].

5 Automation

We have used our own package repository to provide modes for multiple devices and distribute changes. Software packages can be checked, adapted, and activated individually. Thereby, we can install modes as if they belonged to *systemd*. We have also examined system configuration management tools like

[2] https://www.fail2ban.org

Ansible, *Progress Chef* and *Puppet*. They are also used for provisioning, application deployment, and infrastructure as a code. We had a closer look at *Ansible* and have developed a module for mode switching, i.e., switching *systemd* targets. With that implementation, we were able to use mode switching in *Ansible* tasks and playbooks. This feature is helpful to manage multiple *edge devices* and to switch the mode of either one or many devices depending on requirements. For example, Listing 1.2 shows how to switch all *edge devices* from group `devgroup1` to the `oudated-mode.target`.

Listing 1.2. Call developed Ansible module to change systemd target

```
ansible devgroup1 -m systemd_target -a "name=outdated-mode.
    target␣state=isolated"
```

6 Discussion

From a security perspective, it is an advantage to switch to a restricted mode as soon as possible, if attacks or abnormal behavior happen. From a customer perspective, it is highly unsatisfactorily if an *edge device* switches the mode automatically and seemingly unexpectedly denies services and interrupts operational processes. Therefore, full automation of mode switching is not desirable under all circumstances. We propose to inform a manufacturer's cloud service and put a human in the loop. Thus, online devices can be controlled semi-automatically or manually by experts. In addition, the devices can learn over time about false positives of possible attacks, e.g., with machine learning techniques. If *edge devices* were offline for a long time, they have to act autonomously and should be stricter about switching modes. However, even then, a specific threshold needs to be considered. Shutting down the device must be the last resort.

Safety is an issue for a CPS as well. Connected machines and robots can potentially harm people and damage property. In this context, the concepts of fail-secure and fail-safe contradict each other to some extent. Completely stopping an *edge device* in case of any issue is highly secure. Nevertheless, if a person is stuck in a machine, some basic functionality is needed to get her free. A differentiated approach is needed in such a scenario. From the outside, the system has to be in a fail-secure mode and may prohibit external access. A fail-safe mode can provide at least some basic on-site functionality.

We have also experimented with multiple modes with different arranged software containers on top of the *updated mode*. Further research is needed to define how many modes are necessary and are still manageable. However, we think it is more beneficial to stop single vulnerable services. *Ansible* and other configuration management tools are well usable to achieve that goal. They allow us to specify single, groups, or all hosts to make changes. We can go one step further and integrate predefined modes more deeply if a service supports different configurations or some kind of limited operation.

7 Limitations

Mode switching and our proposed *edge device* modes are not intended to replace other traditional hardware and software protection techniques. Secure system architecture and security by design are still necessary. Developers must not become less concerned about security and rely on the fact that they can later install an update or patch. However, modes can contribute to more secure and flexible system architectures. Our research has no empirical results yet. Experiments and more detailed comparisons against other protection techniques will have to underline the effectiveness of the approach.

8 Conclusion

We have given a first impression of how mode switching can increase the security of *edge devices* and enhance resilience. We have shown an example of how modes can be implemented in the Linux operating system and how system configuration management tools can help to manage modes of multiple *edge devices*.

Future work includes monitoring modes of multiple *edge devices* on the manufacturer's cloud service and considering the *edge device* modes of a customer and across different customers in a security information and event management (SIEM) system. Being able to change *edge device* modes manually is highly beneficial in case of known vulnerabilities or when stopping malware from spreading further. In addition, we plan to simulate attacks to demonstrate the effectiveness of our approach.

Acknowledgement. This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

1. Allianz Global Corporate & Specialty SE: Allianz Risk Barometer (2022), https://www.agcs.allianz.com/content/dam/onemarketing/agcs/agcs/reports/Allianz-Risk-Barometer-2022-Appendix.pdf, accessed: February 20, 2022
2. Autosar: Guide to Mode Management. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_ModeManagementGuide.pdf (2017), accessed: February 24, 2022
3. Cejka, S., Knorr, F., Kintzler, F.: Edge Device Security for Critical Cyber-Physical Systems. In: 2nd Workshop on Cyber-Physical Systems Security and Resilience (CPS-SR) (04 2019)
4. European Union Agency for Cybersecurity (ENISA): Good Practices for Security of IoT - Secure Software Development Lifecycle (Nov 2019), https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1

5. European Union Agency for Cybersecurity (ENISA): Guidelines for securing the Internet of Things: secure supply chain for IoT. Publications Office (2020). https://doi.org/10.2824/314452
6. Fail2ban: Manual Fail2ban 0.8, https://www.fail2ban.org/wiki/index.php/MANUAL_0_8, accessed: February 20, 2022
7. Firesmith, D.: System Resilience: What Exactly is it? (2019), https://insights.sei.cmu.edu/sei_blog/2019/11/system-resilience-what-exactly-is-it.html, accessed: February 23, 2022
8. freedesktop.org: systemd.target — Target unit configuration, https://www.freedesktop.org/software/systemd/man/systemd.target.html, accessed: February 22, 2022
9. Hess, K.: Linux security: Protect your systems with fail2ban. Red Hat (Jun 2020), https://www.redhat.com/sysadmin/protect-systems-fail2ban, accessed: February 21, 2022
10. International Electrotechnical Commission (IEC): IEC 62443-4-1:2018 — Security for industrial automation and control systems - Part 4-1: Secure product development lifecycle requirements, https://webstore.iec.ch/publication/33615
11. International Organization for Standardization (ISO): ISO/IEC 27000:2009 (2009), https://www.iso.org/standard/41933.html
12. McCormack, M., Vasudevan, A., Liu, G., Echeverría, S., O'Meara, K., Lewis, G., Sekar, V.: Towards an Architecture for Trusted Edge IoT Security Gateways. In: 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20). USENIX Association (Jun 2020), https://www.usenix.org/system/files/hotedge20_paper_mccormack.pdf
13. National Institute of Standards and Technology (NIST): Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1 (Apr 2018). https://doi.org/10.6028/NIST.CSWP.04162018
14. National Institute of Standards and Technology (NIST): Security and Privacy Controls for Information Systems and Organizations (Sep 2020). https://doi.org/10.6028/NIST.SP.800-53r5
15. Noghabi, S., Kolb, J., Bodik, P., Cuervo, E.: Steel: Simplified Development and Deployment of Edge-Cloud Applications. In: 10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18) (Jul 2018)
16. Riegler, M., Sametinger, J.: Mode Switching from a Security Perspective: First Findings of a Systematic Literature Review. In: Kotsis, G., et al. (eds.) Database and Expert Systems Applications. pp. 63–73. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-59028-4_6
17. Riegler, M., Sametinger, J.: Mode Switching for Secure Web Applications – A Juice Shop Case Scenario. In: Kotsis, G., et al. (eds.) Database and Expert Systems Applications. pp. 3–8. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-87101-7_1
18. Riegler, M., Sametinger, J.: Multi-mode Systems for Resilient Security in Industry 4.0. Procedia Computer Science **180**, 301–307 (2021). https://doi.org/10.1016/j.procs.2021.01.167, proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020)
19. Riegler, M., Sametinger, J., Vierhauser, M., Wimmer, M.: Automatic mode switching based on security vulnerability scores. In: submitted for publication (2022)
20. Sametinger, J., Steinwender, C.: Resilient Context-Aware Medical Device Security. In: International Conference on Computational Science and Computational Intelligence, Symposium on Health Informatics and Medical Systems (CSCI-ISHI). pp. 1775–1778 (2017). https://doi.org/10.1109/CSCI.2017.310

21. Shultis, J.K., Faw, R.E., McGregor, D.S.: Fundamentals of Nuclear Science and Engineering; 3rd Edition. CRC Press (2016), https://cds.cern.ch/record/2245430, accessed: February 24, 2022
22. SmartCockpit: A330-A340 Flight Crew Training Manual (Jul 2004), https://www.smartcockpit.com/docs/A330-A340_Flight_Crew_Training_Manual_1.pdf, accessed: February 24, 2022
23. van Smoorenburg, M.: init, telinit - process control initialization. Debian (Jul 2004), https://manpages.debian.org/testing/sysvinit-core/init.8.en.html, accessed: February 21, 2022
24. Statista: Number of edge enabled internet of things (IoT) devices worldwide from 2020 to 2030, by market, https://www.statista.com/statistics/1259878/edge-enabled-iot-device-market-worldwide/, accessed: February 20, 2022
25. Statista: In-depth: Industry 4.0 2021 (Jun 2021), https://www.statista.com/study/66974/in-depth-industry-40/, accessed: February 20, 2022
26. US Nuclear Regulatory Commission (NRC): Standard Technical Specifications – Operating and New Reactors – Current Versions (2019), https://www.nrc.gov/reactors/operating/licensing/techspecs/current-approved-sts.html, accessed: February 24, 2022
27. Zoualfaghari, M.H., Reeves, A.: Secure & zero touch device onboarding. In: Living in the Internet of Things (IoT 2019). p. 1–3 (May 2019). https://doi.org/10.1049/cp.2019.0133