

Gapfruit OS for the Internet of Things in Critical Infrastructures

Sid Hussmann
CTO & Co-Founder
Gapfruit

February 19, 2025

Abstract

Digitalizing critical infrastructures and industrial automation requires a new holistic approach, especially given the exponentially rising cyber security threats. Securing these sectors needs a solid foundation, part of which is the operating system that drives the infrastructure. This is where Gapfruit OS comes into play.

Gapfruit OS is a microkernel operating system with capability-based security developed using the Genode Framework [2]. The capability-based architecture of Gapfruit allows governing the trust relationship of each subsystem down to the hardware root of trust. This "capabilities all the way down" approach provides rational arguments as to why the system is trustworthy. Gapfruit OS is in production use in financial services as part of a TEE and in manufacturing as a secure IoT gateway.

This paper describes the benefits of Gapfruit OS for highly secure, mass deployment of IoT devices. IoT is a field where the domains OT (Operational Technology), IT (Information Technology), and Telco (Telecommunications) come together. We explain the individual challenges of these fields and propose a solution.

We then bring seven properties of highly secured devices into the context of hardware, operating system, and the cloud. Section 4 explains how Gapfruit OS holistically leverages these seven properties.

Further, we will show why critical infrastructures still have ancient and vulnerable operating systems deployed and how Gapfruit OS mitigates this problem and why greenfield products based on Gapfruit OS are future-proof. Section 5 describes a concrete setup of an industrial IoT gateway deployed on a factory floor and managed via Azure DPS/IoT Hub [3].

The appendix contains a threat analysis of a deployment containing Gapfruit OS as part of a secure door system for banks.

©2025 gapfruit AG. All rights reserved.

Contents

1	The Internet of Things	3
1.1	IoT: Converging Domains	3
1.2	Difference in Security Priorities	4
1.3	Challenges of Scaling IoT	4
1.4	Safety vs Security	5
2	The Problem with Current Operating Systems	5
2.1	Why are Linux BSP's Chronically Outdated?	6
3	Properties of Highly Secured Devices	7
4	Gapfruit OS Technology Overview	8
4.1	Core Principles	8
4.1.1	Strong Isolation	8
4.1.2	Control Over all Dependencies	8
4.2	Comparison of Attack Surface	10
4.3	Resilience and Availability	10
4.4	TPM and PKI	11
4.5	Linux Device Driver Environment	11
5	Case-Study: Industrial IoT Gateway	12
5.1	Challenges in the Realm of Digitalization of the Factory Floor	12
5.2	Zero-Trust in OT	13
5.3	Enable Plug and Produce with Azure DPS	14
5.4	Benefits	14
6	Threat Models	16
6.1	Attack Trees	16
6.2	STRIDE Model	18

1 The Internet of Things

We consider IoT devices as interconnected products that are not directly operated by end-users. In contrast to end-user devices such as personal computers, laptops, and smartphones, IoT products are often in use for decades. Additionally, IoT devices are being trusted for safety, security, and privacy. The need for such products is rapidly rising in sectors such as energy, industrial, health, transportation, logistics, retail, building, agriculture, security, and public safety [7].

Many IoT solutions are used to automate critical systems where the utilization of the machine directly affects productivity, safety, and the supply chain of goods and bring enormous financial opportunities.

1.1 IoT: Converging Domains

IoT lies in the middle of OT (Operational Technology), IT (Information Technology), and Telco (Telecommunications) systems. OT, IT, and Telco systems all involve similar technologies to manage and control various aspects of business and industry. However, each domain has its strengths and weaknesses.

OT systems monitor and control physical processes in industries such as energy, water, industrial automation, and transportation. These systems often use sensors, actuators, and control systems to collect data, monitor, automate, and control processes.

IT systems store, retrieve, transmit, and manipulate data in the context of business and organizations.

Telco systems interconnect millions of devices and maintain communication between these connections.

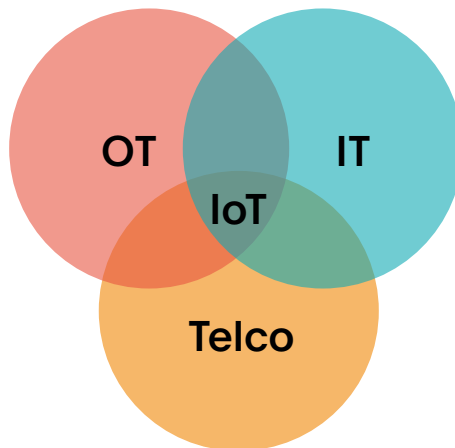


Figure 1: Converging domains

The main difference between IT and OT is that IT is primarily concerned with managing and processing information - while OT focuses on operating and controlling physical devices and processes. While IT and OT systems may use similar technologies, they are typically designed and used for different purposes and environments.

IT engineering practices evolved in a way to meet the constant-changing business needs. The established agile development practices made the technological leaps we see every day possible. These practices resulted in a “ship fast - fix later” mentality that often is at odds with the OT mentality “never touch a running system”. OT systems need to run uninterrupted for decades and their availability - or lack of - are critical for the success or failure of the manufacturing company. In the past 15 years, IT has made massive progress in developing tools to write, build and ship software as rapidly as possible. Further, Docker [8] innovated this field to de-couple software components so that these containers can be written by different teams and deployed to any Linux server in the cloud. These micro-services communicate with each other over clearly defined APIs. The rise of WebAssembly [9] and WASI [10] is taking this concept to the next level by making executables platform-independent and very lightweight.

Telco systems excel at managing and interconnecting millions of devices and are designed to guarantee the availability and integrity of communication. To ensure these requirements, Telco systems have their strengths in configuration management, monitoring, and the provisioning and discovery of devices.

1.2 Difference in Security Priorities

OT, IT, and Telco share similar security goals, however the priorities are often quite different.

The security goals of IT systems generally focus on protecting the confidentiality, integrity, and availability (CIA) of data at rest, in transit, and in use.

In contrast, the security goals of OT systems typically focus on ensuring the safety, integrity, reliability, and availability of physical processes and devices. This includes protecting against unauthorized access or tampering with physical systems, protecting against system failures or disruptions, and ensuring that processes and devices are functioning as intended. These requirements can make OT systems more challenging to secure, as delays or disruptions could have serious consequences. Further, updates in OT systems are only allowed in constrained time windows. Any downtime massively impacts productivity, cost, and the overall supply chain. Another difference is that OT systems are often deployed in more challenging or hazardous environments, such as industrial manufacturing, oil rigs, or power plants. Integrity is an often underestimated priority, as a change in a configuration, e.g., a composition of materials for the automobile industry, could lead to expensive recall campaigns of already sold cars.

Telco companies are responsible for protecting their network and customers' data and complying with many standards and regulations such as GDPR [4] or PCI DSS [5]. On a network management level, they have security principles such as role-based access control [6] in use.

1.3 Challenges of Scaling IoT

This section summarizes the challenges to massively scale IoT projects described in the Beecham report "Getting to Mass IoT Deployment: Challenges and Opportunities" [7].

IoT projects are now mainstream in that they exist in all market sectors and have been shown to bring benefits. Therefore, the time has come to expand these projects from proof of concept to large-scale deployments. More sensors will be deployed at lower costs, providing more data across a broader range of use cases; early results show this is not straightforward.

Once a deployment starts to scale, manually performing even the most basic operations, such as onboarding, configuration, security patches, and maintenance, will be increasingly difficult.

Low data rate connectivity may take a very long time to upgrade a piece of firmware unless it is organized for efficient upgrade. Thus, managing a large fleet of devices remotely is a whole different ballgame than managing a few. An IoT solution designed for a small deployment may be totally unable to scale to a large deployment.

The paper also addresses security concerns: "having good data security has never been more important". Security in IoT and digital transformation solutions can't be an afterthought and has to be an integral part of any development. Unfortunately, many companies still rely on inadequate legacy solutions. They cannot detect and respond to today's advanced attack strategies. Moreover, there is a cybersecurity skills shortage. Relying on manual threat analysis and detection, as well as a security-as-you-go strategy, cannot keep pace with the advanced capabilities of today's cybercriminals. This calls for a zero-trust architecture in contrast to the legacy perimeter approach. In a nutshell, zero trust is security by design on an infrastructure level.

Gartner says SASE (Secure Access Service Edge) will transform the "legacy perimeter" into "a set of cloud-based, converged capabilities created when and where an enterprise needs them, and edge computing is one of many drivers. The key difference (to other endpoint computing solutions) will be the assumption that the edge computing location will have intermittent connectivity and the risk of physical attacks on the system.

According to one of the experts, "The purpose of IT is literally to support IoT maintenance. It needs to either be completely maintenance-free or near to it. What is very important is the update mechanism is designed from the very beginning. And very importantly, it is using a powerful network management platform."

1.4 Safety vs Security

The engineering principles required for products used in safety-critical scenarios are at odds with best practices for security. In a nutshell, here are the goals between these two categories:

- Safety: The system must not harm the outside world.
- Security: The outside world must not be able to harm the system.

Industries such as aerospace, med-tech, automotive, or OT have rigorous certification procedures for their products. Domain-specific certification agencies review the development process, the design, the testing, etc., and sometimes even each line in the source code. Certifying a safety-critical product can therefore become tremendously expensive. Any change to the product would lead to a costly re-certification of the system. This fact incentivizes manufacturers never to change running systems. E.g., many hospitals or industrial production factories still have life-preserving devices running a specific version of Windows XP or even older.

Systems with a focus on security are handled differently. Some software components are complex by nature. The more features a software product has, the larger is their complexity - and ultimately, the larger the attack surface. Commercial software typically has 20 to 30 bugs for every 1000 lines of code, according to Carnegie Mellon University's CyLab [11]. If a bug or vulnerability is found, that component needs an update. Very quick. Even with a secure operating system, such as Gapfruit OS, where the impact of bugs is significantly reduced (see section Comparison of Attack Surface), it is essential to update certain aspects such as e.g., OpenSSL/libcrypto [12] rapidly.

We are returning to our example with the hospital's medical device running Windows XP above. The call for more digitalization in hospitals results in interconnecting these devices that contain vulnerable software. Any script kiddie can hack these devices remotely, resulting in fatal incidences. In other words, there is no safety without security. The same is true in other fields such as OT, where machines are built to be used for 15 to 30 years and, until recently, were not intended to be connected to external networks - especially not the hostile internet [13].

2 The Problem with Current Operating Systems

Current operating systems were designed more than half a century ago. At the time, there were different requirements important than we have today. Systems needed to work, be stable and fit the constrained computing resources that were common back then. They were not designed to face the hostile interconnected world we live in today. After the design was set in stone, different generations tried to add some security concepts. However, the core concepts of these operating systems are still used today. And among the most popular operating systems, the rough architecture is very much the same:

They lack proper isolation mechanisms so that any subsystem has a global impact on the overall system.

Here a rough overview of this legacy design: Figure 2 shows the hardware on the bottom. The kernel is abstracting that hardware. On the top, applications use this abstraction via system calls.

Now, what is wrong with this approach?

One major security issue with monolithic operating systems is that the entire system may be at risk if a single process is compromised. For example, if an attacker can exploit a vulnerability in something inherently complex such as a network driver, they can gain access to the entire system with potentially devastating consequences. Linux, for instance, contains close to 40 Million lines of code, where each line is critical. Each line you have to trust.

Software that runs on these operating systems can access a vast number of system calls provided by the kernel. Applications use these syscalls to access system resources such as file systems, networking sockets, devices, etc. While user applications run in a de-privileged mode, the attack vector to other applications and to that monolithic kernel with over 300 syscalls is enormous. Further, applications operate within one global namespace and typically share one common file system, which results in having to trust all applications not to misbehave.

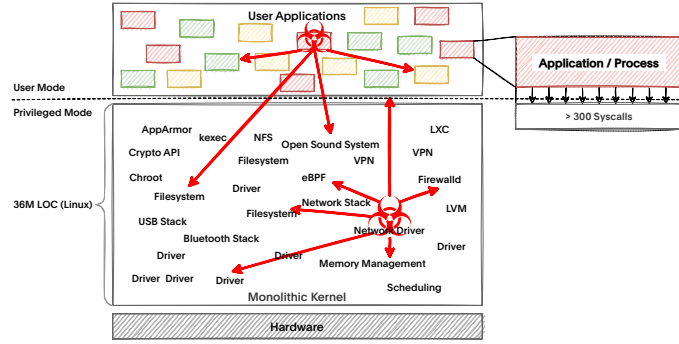


Figure 2: Attack surface of monolithic operating systems

2.1 Why are Linux BSP's Chronically Outdated?

A blog post [14] by Kees Cook, a Linux security expert and Google security engineer, illustrates the challenges in maintaining the security of the Linux kernel. One of the main takeaways from the blog is: "If you're not using the latest kernel, you don't have the most recently added security defenses (including bug fixes)."

Going the "upstream first" route is the absolute best way of keeping the Linux kernel secure. However, it is only part of the story. The challenges faced by device manufacturers running Linux on embedded devices are vastly different.

Most device manufacturers rely on the semiconductor vendor-provided Linux kernel as part of the Board Support Package (BSP), particularly for non-x86 devices. In the interest of going to market early, the semiconductor vendors typically freeze a version of the upstream kernel and add their patches to support their System On Chip (SoC) / Processor.

Now, the device manufacturers are stuck on a version of the Linux kernel that no longer receives security or bug fixes. In contrast, the fixes are available in both the stable and long-term supported (LTS) upstream kernel releases. To get these fixes, the device manufacturer must apply the semiconductor vendor patches (vary from 10s to 10000s) on top of the upstream kernel, which tends to result in conflicts that are difficult to resolve. Once the conflicts are resolved, testing the updated kernel poses a whole other set of challenges where all the sub-systems and drivers need to be retested to ensure they did not break anything.

Now imagine doing this over and over every week because, on average, that is how frequently a new version of the LTS kernel is released [15]!

In an ideal world, the semiconductor vendors would timely upstream all of the SoC-specific patches so that the device manufacturers could use the upstream kernel as-is to get security/bug fixes. Unfortunately, the reality is far from that. Upstreaming patches is time-consuming, and semiconductor vendors typically rush to support new processors/variants while fixing bugs in their older BSP releases. While many semiconductor vendors actively contribute back their SoC-specific patches to the upstream kernel, only a subset makes it to the upstream kernel. Hence, the next BSP release from the SoC vendor ends up with a combination of patches supporting old and new processors/variants, resulting in the number of vendor patches in the frozen vendor kernel being more or less the same... and the cycle repeats with each release. As a device manufacturer, it is not apparent if the vendor has upstreamed all of the patches for your processor. Hence, the safest starting point for the Linux kernel is the vendor-provided kernel. This means that "upstream first" is not an option for most device manufacturers [16].

In other words, devices running Linux are not only architecturally vulnerable; the fact that they have all known security fixes applied is rather the exception than the norm.

3 Properties of Highly Secured Devices

Microsoft Research published an outstanding paper regarding what it takes to build highly secured devices [17]. The researchers identified seven necessary properties, which we bring into the context of HW, OS, and cloud. Figure 3 illustrates which parts of the computing stack are affected by a particular property.

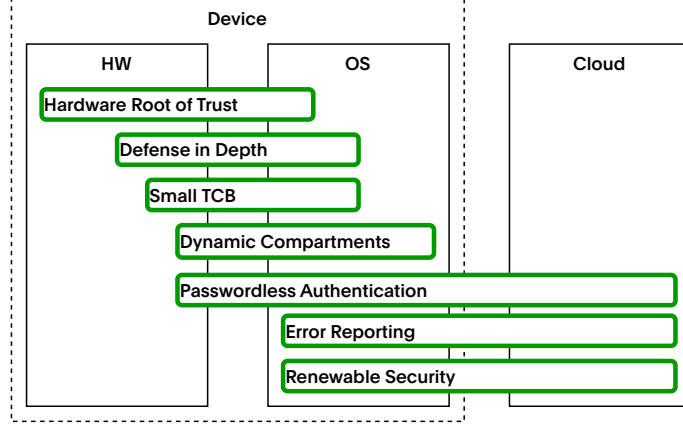


Figure 3: 7 Properties of Highly Secured Devices

A *hardware root of trust* (RoT) is a hardware-based security feature that highly secured devices use to establish a secure foundation for the trustworthiness of the device's computing stack. This trust anchor is typically a TPM [18], which is a discrete chip or part of the SoC. For a trusted boot, all of the boot stages verify the next one. As the name states, this requires hardware. However, the OS must also be capable of establishing the trust graph up to the applications.

Defense in depth is a mechanism that involves implementing multiple layers of security controls to protect against threats. With this approach, an attacker will have a significantly harder time breaching the system. If they breach one line of defense, the other lines can still provide protection. Devices achieve effective in-depth defense with the right combination of hardware primitives, a small trusted computing base (TCB), and compartmentalization.

A *small trusted computing base* (TCB) is a concept that refers to the subset of a system's components that enforce other security concepts. The goal of a small TCB is to minimize the complexity of components and hardware primitives that must be trusted to ensure the system's integrity, confidentiality, and availability. A common misbelief is that this goal stops at the hardware. For effective guarantees, e.g., for isolation, we must also consider the choice of hardware primitives. E.g., the usage of virtual memory in combination with nested page tables is magnitudes less complex and can therefore be verified for correctness. This simplicity contrasts with complex offerings such as Intel TDX, which implements a whole virtual machine monitor in micro-code, which some consider "hardware" - that hides the complexity from the average software developer [20].

Compartmentalization is a mechanism to isolate different parts of a system from each other. The last few years have proven that the trust relationship between these compartments is very nuanced, and we cannot categorize them in ultimate trusted vs. untrusted. Hence, the need for *dynamic compartments*. The isolation mechanism for these compartments needs to be designed with a small TCB to be effective. Compartmentalization acknowledges that software is generally flawed, which is proven by reality. It allows for the planning for the worst case. When a breakage happens, the damage remains constrained.

Password-less authentication in the context of IoT boils down to using certificates to authenticate a device to a cloud offering or other remote systems. We can achieve the highest level of security when we anchor this property cryptographically with a hardware RoT that attests to the trustworthiness of the entire device.

Error reporting states the need for each subsystem to report its state and any failures that affect the system's overall health. This error reporting needs to be accessible from the cloud that

manages the fleet of devices.

Renewable security is a concept that allows you to update the security measures of a device. We can divide this into proactive updates and a technique that involves detection and recovery. Proactive updates are necessary when, e.g., a vulnerability has been found in a crypto library such as OpenSSL/libcrypto. The confidentiality and integrity of data in transit of any component that uses this library may be at risk. Thus, it is essential to update this library as fast as possible. In contrast, some system faults, such as zero-day exploits, have been unknown for a long time. To protect from these threats, we need isolation. And if this does not help, the system needs a way to detect and recover from these vulnerabilities.

It becomes apparent that contemporary operating systems do not fit the properties required to build highly secured devices (see section The Problem with Current Operating Systems).

4 Gapfruit OS Technology Overview

Gapfruit OS solves the seven properties of highly secured devices in a holistic approach. Gapfruit OS is a microkernel operating system with capability-based security developed using the Genode Framework [2]. The capability-based architecture of Gapfruit allows governing the trust relationship of each subsystem down to the hardware root of trust. This "capabilities all the way down" approach provides rational arguments as to why the system is trustworthy.

4.1 Core Principles

This section describes a short overview of the core principles of Gapfruit OS.

There are two core principles of Gapfruit OS: Strong isolation and control over the trusted computing base (TCB) of all software stacks. Control over the TCB of all software stacks means that each component's dependency graph is concisely defined and verified during build, deployment, and run time.

4.1.1 Strong Isolation

The building blocks in a Gapfruit system are called components. Each component on Gapfruit OS runs within a strict sandbox. The microkernel guarantees strong isolation to protect the application and data at runtime.

The microkernel guarantees the quality of the isolation, containing a minimal trusted computing base of roughly 10k lines of code. With such a small TCB, there is a realistic chance that the kernel is entirely free from vulnerabilities.

Each component only receives access to the resources and services it absolutely requires. Components are grouped into a deployable subsystem called SLICE (*Secure and Light Instance of Contained Enclave*). A nested configuration mechanism defines the SLICE topology, which forms a mandatory access control system [19] for every possible resource.

4.1.2 Control Over all Dependencies

Apart from the strong isolation, another core concept is how Gapfruit OS governs dependencies. The first type of dependency is *Resource Distribution*. A *child* component depends on its *parent*. Each dependee is designed as simple as possible so we can verify it for correctness. At the root of this dependency tree lies the microkernel. A parent component provides its children with resources and establishes service connections to other components.

These connections form the second type of dependency: *Service Topology*. Components and SLICES are connected via a service-oriented architecture. A service is a means of abstraction that provides access to a resource or functionality. There are roughly two dozen service types in Gapfruit OS, like file system, networking, GPU, USB, and real-time clock, to name a few. A client depends on a server providing a service. The topology inherently governs the access control to the different services. The underlying technique is called capability-based security. Note that even though the server is more critical regarding availability to the client, confidentiality and integrity are still guaranteed.

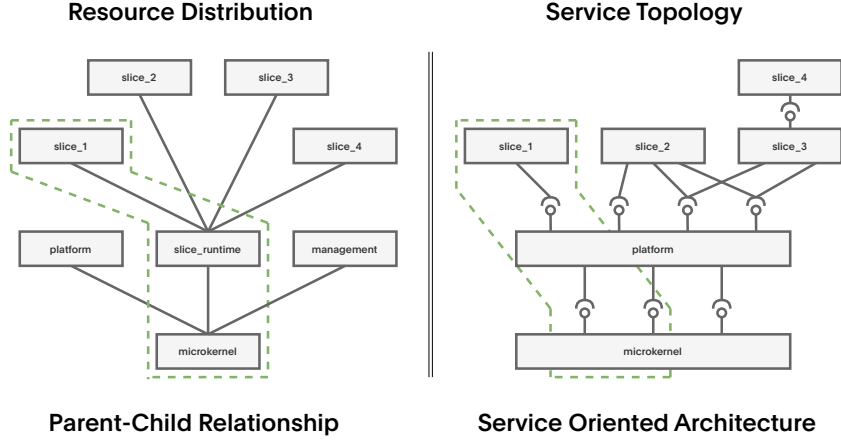


Figure 4: Resource distribution and service topology

The third type of dependency shown in Figure 5 controls the supply chain of *Software Dependencies* where a SLICE depends on binaries, libraries, or other artifacts that are part of distributable packages. When people in software engineering talk about dependencies, they often mean a form of library dependency. The transactional package management system of Gapfruit OS lets you define and verify the Software Bill of Material (SBoM) for each SLICE during build, deployment, and run-time. The declarative dependency definition solves the trade-off between deploying subsystems independently and effectively sharing common libraries. The package management system makes updates as lightweight as possible since only the delta is being deployed.

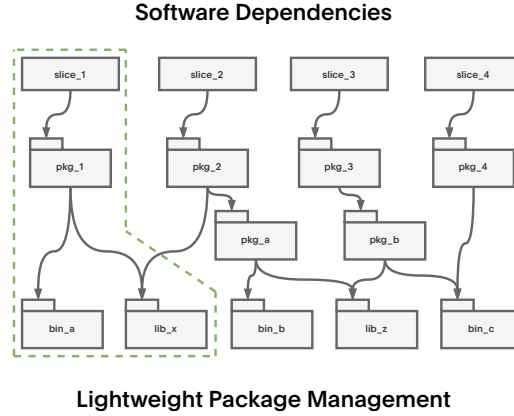


Figure 5: Software dependencies

Figures 4 and 5 show these three types of dependencies in three views of the *same* system. Having this level of control over all dependencies lets you define the components that can impact a specific feature's computation and data flow. E.g., the components highlighted in green are part of the Trusted Computing Base (TCB) of that particular feature. No other component can interfere with that specific TCB due to the isolation guarantees and the governance over the dependencies. These are very powerful properties. This separation allows you to design a system where only the components in green have to be, e.g., certified for safety-critical criteria. Any other components - such as internet-facing network components - can be rapidly updated without re-certifying the whole product.

This property solves the problem described in section Safety vs Security. With Gapfruit OS, you can now respond quickly on a disclosed vulnerability in, e.g., *OpenSSL/libcrypto* without expensive re-certification for safety.

Furthermore, the isolation of every component in the system combined with the capability-based security, protects potentially malicious code from breaking out and protecting data in use.

An analogy would be the objectives of enclaves [20] combined with what virtual machines or other sandboxing mechanisms try to achieve. (Figure 6). Hence, Gapfruit OS guarantees the isolation of components from the outside-in and inside-out. This duality of isolation is essential, as it is sometimes unclear which stakeholder considers which component of a system as trustworthy.

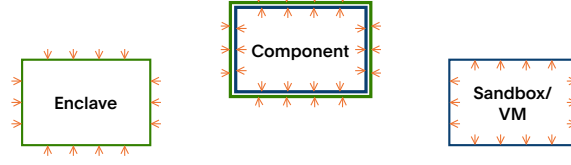


Figure 6: The isolation of a component in Gapfruit OS in contrast to enclaves and sandboxes

4.2 Comparison of Attack Surface

Section 2 describes how an exploit in the network stack on typical operating systems results in the complete compromise of the whole system. Figure 7 shows the attack surface from a compromised network driver between Linux and Gapfruit OS. E.g., when there is an exploitable vulnerability in a Linux network driver, the whole Linux system is compromised. Gapfruit OS supports running Linux drivers strongly isolated in userspace. In this case, the vulnerability is still part of the driver. However, that compromised driver is only another hop in an already untrusted network. In other words, the confidentiality and the integrity of the communication between the management agent and its cloud counterpart stay intact. When it comes to availability, the self-healing resilience system of Gapfruit OS will detect a misbehaving component and restart that component without any impact to the rest of the system (see section Resilience and Availability).

Compared to other trusted computing approaches, such as ARM Trustzone [21], which divides the world into two compartments (secure and non-secure world), Gapfruit OS offers truly dynamic compartments (as required in 7 Properties of Highly Secured Devices). Any exploit of third-party code, such as device drivers or network stacks, is isolated and only affects that particular component.

Compared with monolithic operating systems where each feature has a global access to the rest of the system, such as Linux, Gapfruit OS reduces the attack surface by more than 99%.

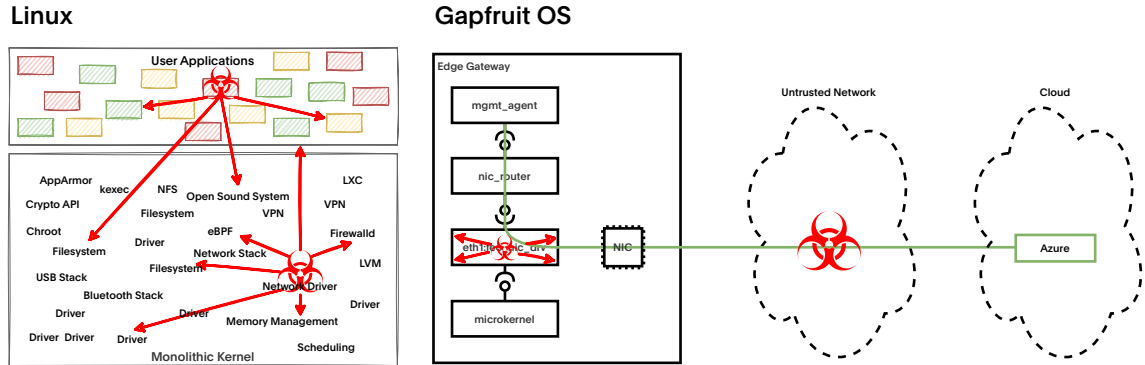


Figure 7: Reduction of attack surface by $>99\%$

4.3 Resilience and Availability

The desired state of the SLICE topology is defined using a declarative configuration interface.

Gapfruit OS supports analyzing the health of each SLICE during run-time and, depending on different criteria, restarting SLICES when required. Even device drivers are designed so they can be restarted while keeping the impact on the overall system to a bare minimum.

Via the system configuration, it is possible to pin a SLICE to one or more specific CPU cores. Product developers can use this to prevent interruption of a critical SLICE so that its functionality

is deterministic and its availability is guaranteed - a necessity for hard real-time requirements.

Each SLICE is restricted by resource usage. The microkernel enforces this limit and stops (and may restart) components that exceed the configured limit. This prevents the system from becoming unstable due to, e.g., exceeding the memory consumption of individual SLICES.

4.4 TPM and PKI

Trusted Platform Modules (TPM) [18] are a hardware root of trust with a standardized cryptographic API. Here is a list of topics for which Gapfruit uses TPM features:

- Authenticated Boot
- Measured Boot
- Attestation of Integrity
- Hybrid Secure Counters for Dynamic Disk Integrity
- Protect Secrets with Policy
- Strong Digital Identity for Authentication

An important topic in the context of zero-trust is using TPMs for authentication. For example, from an application perspective, a POSIX application may utilize the credential management system of Gapfruit OS for authentication to cloud services. The application can read the certificate and the private key for the mTLS connection using POSIX read functions. The credential management system will then issue short-lived credentials from TPM-backed certificates issued from external public-key infrastructures (PKI).

In other words, Gapfruit enables application engineers to leverage the benefits of PKI in combination with strong digital identities without requiring them to be security experts. More on that subject in section Case-Study: Industrial IoT Gateway.

4.5 Linux Device Driver Environment

Gapfruit OS leverages the Linux Device Driver Environment from Genode [2] - an absolute game-changer in the microkernel community. It enables Gapfruit to benefit from the vast Linux ecosystem regarding hardware support (see section 2.1). The Linux driver is embedded in a lightweight shim that provides the Linux kernel internal API to the driver and transforms this API to sessions as described in section 4.1.2. Note that this Linux driver also runs within a strict sandbox in userland and, therefore, is not critical for the trustworthiness of the overall system - even when the driver is compromised.

Hardware vendors struggle to keep their Linux board support package (BSP) up-to-date with upstream Linux. This often makes hardware deployed in the field obsolete after 2-4 years since critical vulnerabilities in the Linux kernel remain unpatched. Devices running Gapfruit OS have a significantly longer lifespan since the drivers can be buggy without impacting the rest of the system.

5 Case-Study: Industrial IoT Gateway

The benefits of Gapfruit OS are relevant for many industries. This section describes a solution to bring a zero-trust strategy to the industrial automation sector: An IoT Gateway that Gapfruit provides in collaboration with hardware vendors and solution providers.

5.1 Challenges in the Realm of Digitalization of the Factory Floor

The industry has hugely underestimated the challenges of digitalization on the factory floor. We can observe that the industry and its various stakeholders have positioned themselves in a deadlock. To illustrate the problem, let's assume a scenario with the following stakeholders:

- The factory which produces goods with machines from different manufacturers.
- The machine manufacturer who wants to explore new business models, such as predictive maintenance or machine-as-a-service.
- The IIoT solution builder that designs and implements the digital solution for the machine manufacturer.
- The managed service provider which maintains the infrastructure to ensure a smooth service.

From the factory's perspective, maximizing productivity is an important priority. Any incident that jeopardizes this productivity has an extreme financial impact. Cyber attacks, therefore, have severe consequences. According to Bitkom [22], cyberattacks cost the German economy €178.6 billion in the most recent reporting period. In 2023, 25.7% of all global cyberattacks targeted the manufacturing industry, making it one of the hardest-hit sectors [23]. Therefore, many factories hesitate to interconnect machines to the cloud or on-premise solutions. Additionally, updating the machines' firmware within a factory leads to downtime. Any downtime incentivizes maintenance technicians not to update the firmware of the machines. On the other hand, not updating the firmware increases the vulnerabilities of the whole factory floor. It becomes apparent how this situation becomes a negative feedback loop. While there is a shortage of maintenance technicians for the factory, an offering from the machine manufacturer regarding predictive maintenance or machine-as-a-service would be very welcome. However, the data that the machine manufacturer needs for their offering is not accessible.

From the machine manufacturers' perspective, they are experts in building the machine but often lack expertise in software engineering, IT, Linux, and security best practices. Therefore, the machine manufacturer is forced to shift the responsibility regarding security to their customers (factories). Hence, their machines don't connect to the cloud, and they don't get the much-needed telemetry and insights for their new business opportunities. However, starting in 2027, the machine manufacturer is liable regarding the security of their machines [8].

Machine manufacturers seek help from IIoT solution builders to digitalize their machines for new business opportunities. With the currently available technology, the IIoT solution builders are forced to have deep security expertise (PKI, TPM, application hardening) and deep operating systems expertise (OS Hardening, Docker [8] tooling, etc.). They are, therefore, struggling to focus on their core expertise. This fact forces them to shift the responsibility of maintaining security to either the factories or the managed service providers.

From the managed service provider perspective, constantly patching and maintaining the fleets of devices is an uphill battle (see section 2.1). Guaranteeing high availability with existing solutions results in extensive manual labor, which results in low margins. Further, the imperative deployment and provisioning of the solution with existing solutions is time-consuming and error-prone. From a security perspective, zero-day exploits have severe consequences and result in too much manual labor.

From a network security perspective, the cleanest solution is to move from a perimeter network model to a zero-trust model.

5.2 Zero-Trust in OT

Zero-trust refers to a security approach in which all devices and users are treated as untrusted and must be continuously authenticated and authorized before being granted access to resources. This approach mitigates the risk of insider threats and reduces the attack surface of OT systems, which are critical to the operations of many organizations.

However, implementing a zero-trust strategy in OT environments can be challenging for several reasons:

- **Computing power:** Many sensors, actuators, or simple IoT devices lack the computing resources for the cryptographic computations necessary for password-less authentication (see section 3).
- **Interoperability:** Many OT systems need to work with specific protocols and a mix of modern and legacy technologies, making integrating them with a zero-trust security solution challenging.
- **Visibility:** It can be difficult to obtain a complete and accurate view of the devices and users within an OT environment, which is necessary for implementing a zero-trust approach.
- **Maintenance:** OT systems often have long life cycles and may not be regularly updated or maintained, making it difficult to ensure that they are secure and compliant with zero-trust best practices.

These challenges call for a hybrid approach of moving this “last perimeter” as close as possible to devices incapable of many security mechanisms. The IoT Gateway running Gapfruit OS protects this last perimeter. The gateway shown in Figure 8 forms a first line of defense that makes a zero-trust transformation possible in OT.

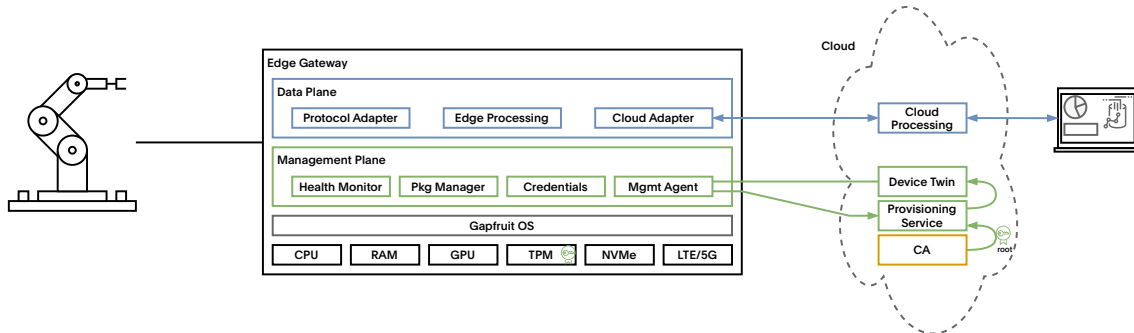


Figure 8: IoT Gateway running Gapfruit OS

We can divide the IoT Gateway into two core functionalities: The management plane and the data plane.

The data plane connects the OT world with the IT world. It is responsible for connecting and pre-processing the payload from sensors, actuators, and other devices behind the gateway. Gapfruit provides several building blocks for connecting sensors and actuators via field buses and IP-based protocols. We also provide building blocks for cloud connectivity to various cloud vendors. Gapfruit offers extensive tooling to combine and adapt these building blocks to create deployable SLICES.

The management plane is responsible for configuring and managing the gateway itself and providing access to the managed solution provider for monitoring and maintenance purposes. To manage truly scalable IoT deployments, we collaborated with OT experts such as Device Insight [25] and Axiros [26], one of the leading Telco technology providers. We’re currently offering management capabilities over Azure DPS/IoT Hub [3] and the Telco standard TR369 [28], which are easy to integrate into existing management solutions. TR369 is designed to bring the scaling of Telco equipment to the internet of things. Note that these management agents are interchangeable to other management protocols, such as NETCONF/RESTCONF.

5.3 Enable Plug and Produce with Azure DPS

Figure 8 shows a setup with a gateway that connects to Azure Device Provisioning Server (DPS) for zero-touch deployments. A precondition is the physical provisioning of the gateway. The way this works is that the Certificate Authority (CA) creates a new root certificate for a cluster, product line, customer, or any other similar group of devices. The managed solution provider creates a new enrollment group for that cluster, defines the initial desired twin, and imports the root certificate. Any device registering with an intermediate certificate is granted access, and the Azure DPS will then create a device twin on the Azure IoT Hub for each device.

On the device side, the system creates a new key pair within the TPM. The system will then issue a Certificate Sign Request (CSR) to the CA. The system attests to the CA that the private key has been generated within the TPM and that the key is non-exportable.

Later, the credential manager will issue short-lived certificates with TPM-backed long-lived keys for the management agent and data-plane SLICES.

All gateways of a cluster connect to the same Azure DPS. A precondition is to define the initial desired state for all devices. Once a gateway is installed and powered on, the management agent will connect to Azure DPS and authenticate itself with TPM-backed credentials. Azure DPS will create a device twin with the initial desired state configuration. The device twin has two main sections:

- Desired Properties: The declarative desired state configuration
- Reported Properties: The actual state of the device

Given the desired state configuration, the gateway will pull the required packages and all their dependencies.

Once all the packages are downloaded, verified, and extracted, Gapfruit OS spawns the respective SLICES and sets the configuration, ultimately automatically coming to the desired state.

From that moment on, the device is being managed via the declarative device management system CENTERSIGHT scale [27].

5.4 Benefits

In the context of the digitalization of the factory floor, we can see how we can address the challenges of the respective stakeholders described in section 5.1 using an industrial IoT gateway running Gapfruit OS:

From the factory’s perspective, we can leverage zero-trust principles in a case-by-case iterative approach. The benefits of zero-touch provisioning, zero downtime upgrades, and end-to-end product lifecycle management guarantee maximum productivity of the shopfloor while enabling a path towards outsourcing complex machine maintenance to the machine manufacturer so they can offer predictive maintenance or machine-as-a-service business models.

Due to the connectivity, the machine manufacturer can now get real-time insights and telemetry of their machine. This data enables them to improve their existing products and offer new business models. Additionally, they now have a way to separate OT and IT-specific software, which helps them to be CRA [24] compliant.

The IoT solution builder can leverage Gapfruit OS and its tools to focus on their core expertise and build a resilient, scalable, and secure solution. There is no need to harden the operating system, and although they still need to take care of some application hardening techniques, they are limited to the application’s intrinsic features. Further, there is no vendor lock-in because the custom SLICES built for Gapfruit OS can be directly exported to Docker [8] images.

The managed solution builder can scale to massive IoT deployments with minimal total cost-of-ownership (TCO). Additionally, they can guarantee high availability while keeping the fleet secure with lightweight transactional upgrades. Minimizing risk and manual labor enables them to maximize revenue with minimal costs while conforming to their service level agreements (SLAs). Last, they can significantly shorten the time to value for their deployments as they move from an imperative setup to a declarative management system.

About Gapfruit AG

Gapfruit is a deep-tech company based in Switzerland with a proven track record in systems security, product development, and software engineering. The founding team developed a military-grade operating system fulfilling the requirements set by national governments and security agencies across the world for ironclad security. With this expertise, Gapfruit brings scientifically recognized academic research to real-world products for today's and future challenges. The developers at Gapfruit have been contributing to the Genode Framework [2] for over a decade.

If you want to deliver trustworthy products yet focus on your core expertise, contact us today.
<https://gapfruit.com>

Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
App	Application
CA	Certificate Authority
CIA	Confidentiality, Integrity and Availability
DPS	Device Provisioning Service
DRTM	Dynamic Root of Trust Measurement
GDPR	General Data Protection Regulation
IoT	Internet of Things
IT	Information Technology
JVM	Java Virtual Machine
LOC	Lines of Code
MAC	Mandatory Access Control
mTLS	Mutual TLS
NIC	Network Interface Card
OT	Operational Technology
PCI DSS	Payment Card Industry Data Security Standard
PKI	Public Key Infrastructure
PLC	Programmable Logic Controller
ROM	Read-only Memory
SBoM	Software Bill of Materials
SE	Secure Element
SLA	Service Level Agreement
SLICE	Secure and Light Instance of Contained Enclave
SoC	System on Chip
SRTM	Static Root of Trust Measurement
STRIDE	A threat modeling technique
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
Telco	Telecommunication
TLS	Transport Layer Security
TOC	Time of Check
TOU	Time of Use
TPM	Trusted Platform Module

Appendix

6 Threat Models

Threat models are used to define the scope of security guarantees for products. This section uses attack trees and the STRIDE models. For these models, there are the following assets: The IoT Gateway, the cloud, the sensor(s), and the actuator(s).

6.1 Attack Trees

This section describes how an adversary would attack a door system. Figure 9 shows how they would try to open the door.

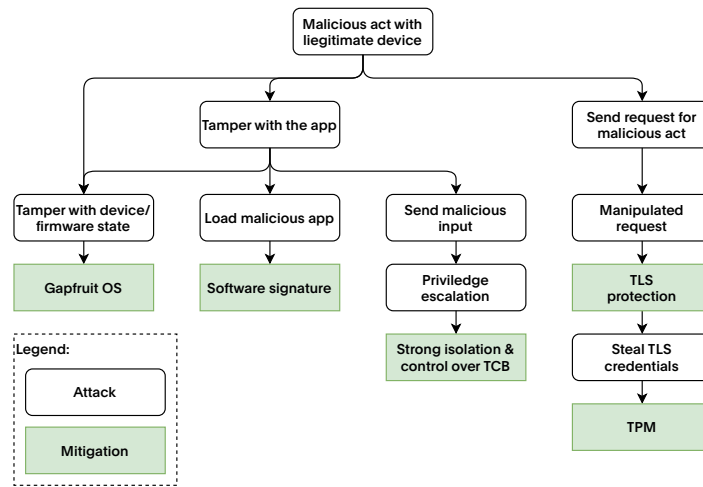


Figure 9: Attack Tree Actuator

While being able to open the door is not desired, spoofing the state of the door can also be disastrous. The analysis can be seen in Figure 10.

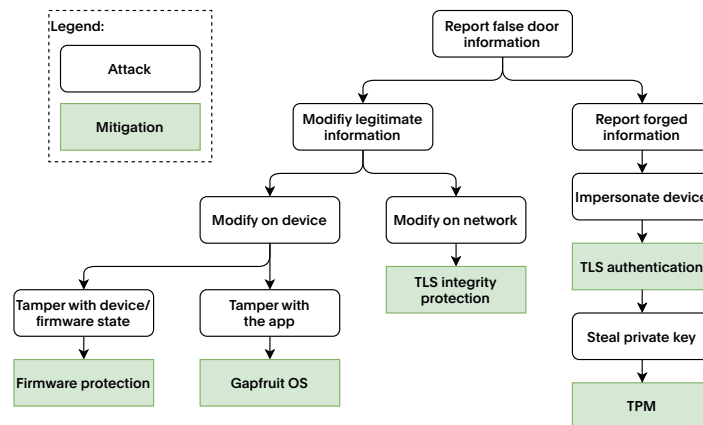


Figure 10: Attack Tree Sensor Information

Figures 9 and 10 reference mitigations that involve the protection of the firmware, shown in Figure 11.

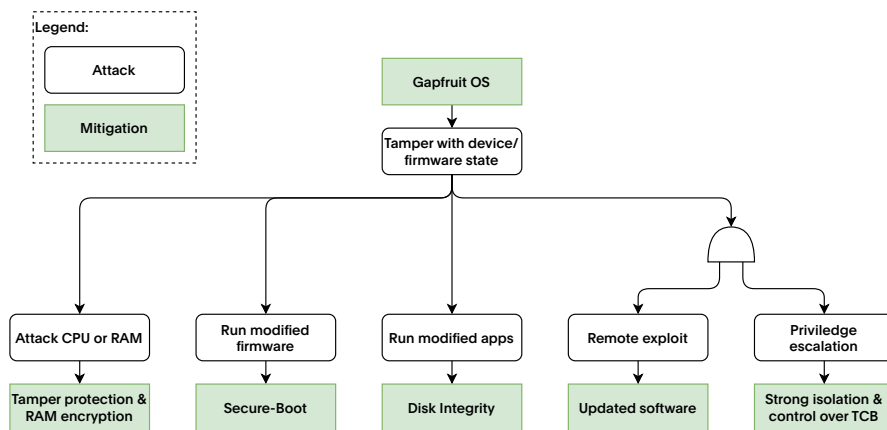


Figure 11: Attack Tree Gapfruit OS

6.2 STRIDE Model

This section describes the threat model of IoT gateway built with Gapfruit OS. The threat model here is derived from the STRIDE model.

Table 1: Threat for Asset - IoT Gateway

Threat	Example
Spoofing	S1: The attacker may impersonate the cloud, device, or an app
	S2: The attacker may spoof to or from another legitimate device or app
	S3: The attacker may replace the legitimate device or app with a forged one
	S4: The attacker may send commands to the actuator as a spoofed device
Tampering	T1: The attacker may modify the firmware or apps of the device
	T2: The attacker may modify a command sent to the device
Repudiation	R1: The attacker may prevent logging
	R2: The attacker may erase or truncate the log
Information disclosure	I1: The attacker may steal the data collected on the device
	I2: The attacker may intercept data transferred between sensors and the device
	I3: The attacker may intercept data transferred between the device and the cloud
Denial of service	D1: The attacker may overload the device or data connection
	D2: The attacker may disable the device
Elevation of privilege	E1: The attacker may gain access to other apps on the device
	E2: The attacker may gain administrative privilege on the device

Table 2: Adversary for Asset - IoT Gateway

Adversary	Example
Network attacker	N1: The attacker may eavesdrop, modify, or spoof packets on the network.
	N2: The attacker may connect to the device to exploit a vulnerability in the firmware or an app
	N3: The attacker may send an unauthorized request to the device
	N4: The attacker may overload the data connection with superfluous network traffic
Unprivileged software attacker	U1: The attacker may insert malicious code in an app to take control of the device
	U2: The attacker may send an authorized request or sensor input to exploit a vulnerability in the firmware or an app
	U3: The attacker may overload the device with a malicious app
Privileged software attacker	P1: The attacker may modify settings or apps
	P2: The attacker may steal credentials
Simple hardware attacker	H1: The attacker may modify the firmware or data on the nonvolatile storage
	H2: The attacker may copy the data from the nonvolatile storage
	H3: The attacker may turn off the device
	H4: The attacker may eavesdrop or modify the communication to the secure element
	H5: The attacker may attach a debug probe
Skilled hardware attacker	K1: The attacker reads or modifies the device's RAM
	K2: The attacker may perform side-channel attacks on the CPU or secure element

Table 3: Mitigation for Asset - IoT Gateway

Mitigation	Example
Protection	Communication with the cloud is encrypted and authenticated (S1, S2, S3, I3, N1, N3)
	Keys are stored and used in the secure-element (S1, S2, S3, P2, H1, H2)
	Nonvolatile storage is encrypted and integrity-protected (T1, I1, H1, H2)
	The device will only boot firmware signed by the manufacturer (T1, H1)
	System components are strongly isolated and governed (T1, R1, R2, I1, E1, E2, N2, U1, P1, P2)
	The casing of the device is tamper-protected (H5)
	Sensors, actors, and their communication to the IoT Gateway are physically protected (S4, I3)
Detection	The heartbeat monitor will detect non-responsive software components (D1, U3, N4)
	The cloud will detect non-responsive devices (D1, D2, N4)
	The casing of the device has tamper detection (K1, K2)
Recovery	The heartbeat monitor will restart non-responsive software components (D1, U3, N4)
	If the integrity check of the firmware fails, the device boots into recovery mode and installs an authentic firmware (P1, H1)

References

- [1] Gapfruit - A new era of operating systems
<https://www.gapfruit.com>
- [2] Genode Framework
<https://www.genode.org>
- [3] Azure DPS/IoT Hub
<https://learn.microsoft.com/en-us/azure/iot-dps/>
- [4] General Data Protection Regulation
<https://gdpr-info.eu/>
- [5] PCI Security Standards Council
<https://www.pcisecuritystandards.org/>
- [6] Role-based Access Control
https://en.wikipedia.org/wiki/Role-based_access_control
- [7] Getting to Mass IoT Deployment
https://mass-iot.com/wp-content/uploads/2022/11/BR_Getting-to-Mass-IoT-Deployment.pdf
- [8] Docker
<https://www.docker.com/>
- [9] WebAssembly (WASM)
<https://www.w3.org/TR/wasm-core-2/>
- [10] WebAssembly System Interface (WASI)
<https://wasi.dev/>
- [11] Carnegie Mellon University's CyLab Sustainable Computing Consortium
<http://www.cylab.cmu.edu/>
- [12] OpenSSL Library
<https://openssl-library.org/>
- [13] Max Weidele, Whitepaper: Der grosse Industrial Security Guide (German)
<https://www.sichere-industrie.de/ressourcen/>
- [14] Kees Cook, Google Security Blog: Linux Kernel Security Done Right
<https://security.googleblog.com/2021/08/linux-kernel-security-done-right.html>
- [15] Linux Kernel Stats
https://github.com/gregkh/kernel-history/blob/master/kernel_stats.ods
- [16] Akshay Bhat: The “upstream first” strategy is only half of the story for embedded devices
<https://www.timesys.com/security/challenges-linux-os-bsp-security-maintenance/>
- [17] Galen Hunt, George Letey, and Edmund B. Nightingale, The Seven Properties of Highly Secured Devices
<https://www.microsoft.com/en-us/research/uploads/prod/2020/11/Seven-Properties-of-Highly-Secured-Devices-2nd-Edition-R1.pdf>
- [18] Trusted Platform Module
<https://trustedcomputinggroup.org/resource/tpm-library-specification/>
- [19] Mandatory Access Control
https://en.wikipedia.org/wiki/Mandatory_access_control

- [20] Julian Stecklina, The Flawed Design of Intel TDX
<https://x86.lol/generic/2023/02/07/intel-tdx.html>
- [21] ARM TrustZone
<https://developer.arm.com/ip-products/security-ip/trustzone>
- [22] Andreas Streim, Bitkom, Angriffe auf die Wirtschaft nehmen zu
<https://www.bitkom.org/Presse/Presseinformation/Wirtschaftsschutz-2024>
- [23] Produktion, Cyberangriffe: Fertigungsunternehmen am stärksten betroffen
<https://www.produktion.de/technik/cybersecurity/cyberangriffe-fertigungsunternehmen-am-staerksten.html>
- [24] European Cyber Resilience Act
<https://www.european-cyber-resilience-act.com/>
- [25] Device Insight
<https://device-insight.com/>
- [26] Axiros, Any Device. Any Protocol. Any Service. Any Time | We manage all THINGS.
<https://www.axiros.com/>
- [27] Device Insight, CENTERSIGHT scale
<https://device-insight.com/en/centersight-scale/>
- [28] Broadband Forum, TR369: The User Services Platform
<https://usp.technology/>