Technology Innovation Institute

Ghaf Compute Platform
Virtualization on the Edge

Secure Systems
Research Center

tii.ae

Innovation for a better world

Contents

Ghaf Compute Platform

This white paper introduces the Ghaf platform developed by the Secure System Research Center (SSRC) of the Technology Innovation Institute (TII). The objective of the platform is to provide an edge device software architecture that enables key features such as modularity and scalability through virtualization, support research and development of Zero Trust Architecture (ZTA), and allow for low maintenance efforts while keeping the main code base stable and secure. The name “Ghaf” stems from the Ghaf tree, a highly resilient plant that remains green even in harsh desert environments.

The term Zero Trust has received broad attention and adoption in the cybersecurity industry after it was first introduced in the early 2000s. Since then, it has shifted the paradigm from implicit trust based on physical characteristics towards explicit verification utilizing stringent authentication, rule of least privilege, separation of duties, and other core security principles. With remote work driving the adoption of Zero Trust Architecture in the last few years across IT systems in large enterprises, the underlying principles continue to find more and more adoption in software and hardware designs.

While enterprise security posture has significantly improved with ZTA, the threat exposure remains vast due to the large Trusted Computing Base (TCB) of enterprise software, especially in heterogeneous computing environments. These challenges reduce the security of the end-to-end solution and significantly increase the cost of developing and maintaining a secure product.

The SSRC Secure Technologies team is focusing their research on enhancing Zero Trust Architecture to scale horizontally across Edge HW Platforms (Phones, Drones, Laptops, Communication modules) and vertically across SW platforms (Linux, Android, Browser, Applications). The Ghaf platform is a baseline software platform for edge devices, utilizing a virtualized architecture for research and product development aiming to achieve the following core objectives: apply the general security principles of zero trust within the software architecture, and act as an enabler for Zero Trust Architectures within organizations.

Virtualization is one of the core enablers to transform the traditionally monolithic software stack within edge devices into isolated components with minimal TCB and clearly defined functionality. The Ghaf platform utilizes a collection of virtual machines to define a system. Contrary to the traditional monolithic OS, this concept allows to define and run host services in isolated environments, which breaks up the monolithic structure and allows for a modular system definition that is customizable for a specific use case. To this end, various applications and guest operating systems can be deployed while simultaneously utilizing the platform's features.

This approach allows the application of zero trust concepts across the platform, and promises developers to choose the right building blocks to close in on the optimal balance between security and performance for their solution. Moreover, the platform promises to improve the ZTA adoption of organizations by increasing resource control and overall security within edge devices.

The vision for the Ghaf platform is to create a virtualized, scalable reference platform that enables the building of secure products leveraging trusted, re-usable, and portable software for edge devices. Ultimately, security relies by and large on the trustworthiness of the underlying hardware. Silicon vendors often reserve highest system privileges, and limit access to security relevant components with proprietary solutions. Thus, the SSRC team works not only to bring zero trust principles into software, but eventually into open-source silicon hardware platforms based on RISC-V.

Product Applications

The Ghaf platform development is focused on the transition to a modular architecture for edge devices. Products such as secure phones, drones, laptops, and other communication devices have unique challenges in their respective hardware and software ecosystems. Enabling the integration of individual technology stacks into an organizational framework can be a challenging task. The Ghaf platform is designed to ease this process and enable research to overcome a number of challenges as discussed in the following section.

Modern security hardened smartphones are, for the most part, built using commercial-off-the-shelf (COTS) components, and as secure phones are typically not mass market products, device manufacturers cannot accommodate exclusive manufacturing partnerships. The main component is the System-On-Chip (SoC), which implements the majority of the phone's functionality. This includes application processors, communications processors (Modem - 2/3/4/5G, WIFI, GPS, Bluetooth), graphics processors, media solutions (audio, video, camera), security solutions, and peripherals (USB, SIM, fingerprint sensors, memories, etc.).

The choice of the SoC also often dictates a majority of the peripheral components such as sensors, memories, power management units, and displays. This leads to a high dependency on the manufacturer (vendor-lock) that impacts large parts of the supply chain including software security updates, but also forces the manufacturers to change the secure phone HW even when only a specific sub-system change (like communication or security) is required. Additionally, any vendors' custom hardware solutions such as anti-tamper mechanisms or secure microcontrollers further complicate the changes and often require a costly re-design. Given the relatively short life-span of an SoC, the costs involved in hardware and software redesign per iteration are significant.

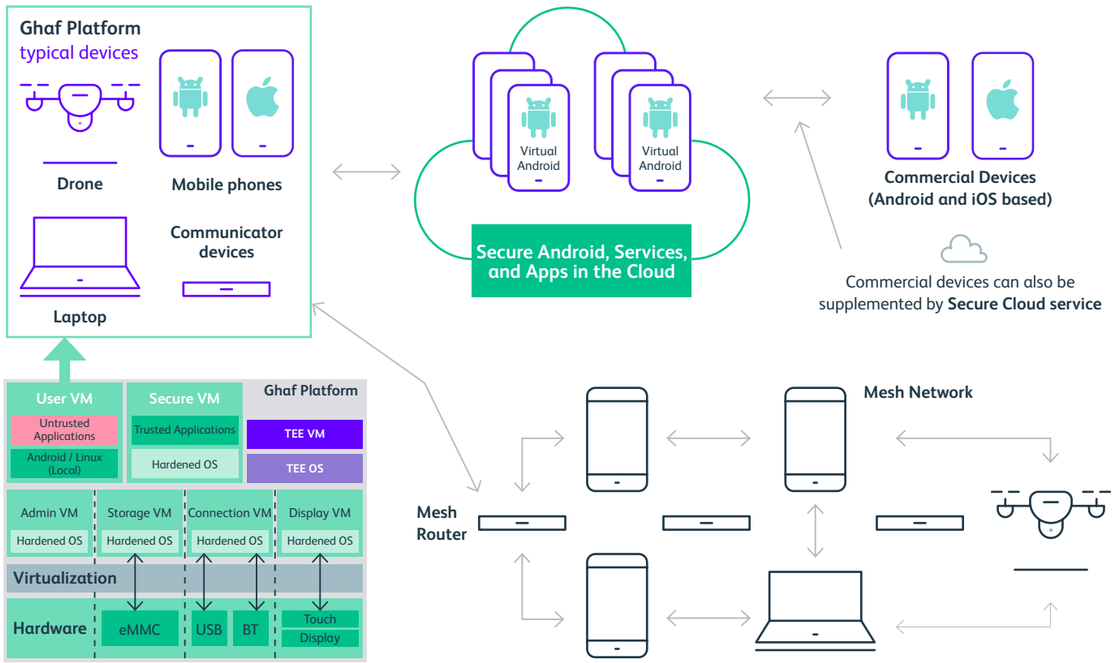


Figure 1 Typical devices and infrastructure around the Ghaf platform

On the software side, secure phone vendors deploy additional security hardening of the software stack. The fast pace of development in the Android software stack and the customer's expectation to receive the latest version often result in costly and time-consuming efforts. Compared to a vanilla Android kernel, the additional code base including drivers can take up to 50% of a hardware vendor's linux kernel [1]. As the bulk of software and intellectual property (IP) belong to the SoC vendor, device vendors typically spend a majority of their efforts on adopting their hardened software to the custom phone hardware. Secure applications are designed, via their interaction with the underlying operating systems (Android, iOS), to be reusable. In most scenarios, an application continues to work when a new version of an operating system is released. There are notable exceptions where an OS introduces a new feature that requires application changes, or a bug that breaks the application (e.g., power optimization changes).

Nonetheless, a modularized phone software architecture can mitigate a number of the challenges described above. Hereby, a key concept is to compartmentalize third-party dependencies in order to gain security control and maintainability. By breaking up the monolithic structure, one can move insecure, obscure, or untrusted functionality such as certain drivers and applications into virtual machines in user-space in order to increase control of their access to security critical parts of the system, and potentially reduce the impact of security vulnerabilities by restricting the attackers lateral movement and make privilege escalation more difficult. On the other hand, secure custom applications with high criticality and security can be equally isolated, for example, a secure messenger or conferencing app.

Moreover, even some critical custom security functionality such as silent switches to turn off camera, microphone, and communications that previously may have required hardware mechanisms could be isolated and controlled without custom hardware by separating out the respective systems controls. The curious reader may have noticed that while an attacker has to surpass an additional layer, this isolation mechanism defers the problem to the hypervisor (e.g., running in highly privileged exception level 2 (EL2)). While this is true, the hypervisor TCB is much smaller and therefore easier to harden and audit, and less prone to changes compared to a mobile phone OS (e.g., Android). Additional security measures may be applied that will further reduce the risk of compromise compared to a monolithic system.

Drones are an example of embedded devices with specific hardware requirements. Contrary to phones, drones are safety critical systems that are typically running real-time embedded operating systems, optimized to account for the time constraints of critical in- and outputs. One challenge with virtualization on such platforms is to guarantee appropriate scheduling to minimize the latency for interrupts and context switches, e.g., between threads. Thus, time critical interrupts must be executed immediately and not delayed by the virtualization overhead or scheduling.

Moreover, drones and similar devices are highly dependent on low latency and often high bandwidth communication, further increasing the systems constraints. While some of these challenges can be solved with appropriate hardware, others have to be accounted for in the software design. Utilizing zero-copy operations and binding hardware (e.g., CPU and FPGA cores) to specific software stacks are crucial to the performance and must be accounted for in the architecture. These constraints further reduce the application of dynamic policies and security monitoring within the software stack significantly, demanding further research of efficient mechanisms to bring these features to the edge.

Other product applications for the Ghaf platform include edge communication devices that, for example, are used to interconnect or control nodes in edge networks. The Ghaf platform architecture naturally supports these use cases, where the isolation principles allow it to deploy numerous security enhancement mechanisms. Traffic monitoring and inspection for identification of malicious internal traffic and detection of adversaries, device resource control, data analysis and anonymization, to name a few. One interesting example is the processing of camera data, which has seen growing attention due to privacy concerns of the collected data which is often required to be appropriately anonymized. One solution is processing the data in a trusted execution environment, e.g., to guarantee compliance with data privacy laws and regulations. Besides the security benefits of isolating data streams and storage, the architecture promises to simplify proving adherence to data privacy rules and regulations.

Design Principles

Edge Security

Edge computing exposes organizations to a variety of security risks, and these vary significantly by individual industry and use case - a one-size-fits-all security approach will not work. The main difference is the exposure of devices and networks to physical interference that may not have been part of the threat model for many organizations in the past.

The security in edge products frequently falls short for various reasons; the considered threats are unknown or too advanced to be adopted into a vendor's threat model, the necessary countermeasures are too expensive or impossible to implement, or simply disengagement between development teams. Thus, understanding the required level of security, designing an appropriate threat model and subsequently selecting the correct security controls for a specific use case is crucial.

The Ghaf security architecture under development by SSRC aims to provide an understandable yet comprehensive view of security controls in the platform so that vendors can make informed decisions and adopt the platform for their use case. The security architecture and subsequent research will be published by SSRC in aseries of technical whitepapers.

Generally, the most critical threat to any software platform are remote attacks over networks, which are often scalable and easily distributed. Even more devastating, as recent events have shown, can be the compromise of software within the supply chain. While these threats are present for all networking devices, the security requirements for edge devices extend beyond common software and networking threats.

Edge platforms deal with an extended attack surface: an edge device software platform has to anticipate a vast variety of environments. Compared to server platforms for example, edge devices must protect against numerous additional threats, particularly attacks in the physical domain. While servers are expected to be deployed in environments with some extent of physical protection, edge devices may fall into the hands of an adversary. Therefore, hardware attacks such as mis-use of debug interfaces, probing attacks, memory extraction, advanced methods such as side-channel analysis and fault injection, and many other threats must be considered in a edge products' threat model. Many of these threats cannot be solved in software alone, but require a symbiosis of hardware and software mitigations.

When considering autonomous edge systems, a number of additional threats are of particular interest that may not be as prevalent in other types of systems. Common threats are denial of service attacks such as wireless jamming, exhaustion of resources, and physical damage or disruption. Moreover, unsupervised and unguarded devices face a higher risk of physical attacks. A physically or logically compromised autonomous system can have more severe consequences due to the safety implications, and could potentially cause physical harm. This increases the need for unsupervised self-tests, self-monitoring, peer-monitoring, and self-organization within the system, requiring advanced detection and response mechanisms.

The objective for the Ghaf platform security is to anticipate and support countermeasures for as many practical use cases as possible while offering implementation flexibility. Hereby, multiple security considerations for the platform are necessary:

- Secure and reliable development, building, distribution, and provisioning
- Continuous code assessments and audits of the trusted computing base
- Patch management with fast vulnerability identification and update deployment
- Multi-layered platform security through isolation and resource control
- Continuous monitoring of the systems health and telemetry
- Adaptive security system to dynamically adjust security detection and response measures
- Modularized security components for data collection and analysis
- Integration into ZTA for organizations and their (autonomous) edge systems
- Application of fine-grained security policy propagation and enforcement

This non-comprehensive list shows a selection of security considerations for the platform. SSRC is committed to conduct research and publish case studies to assist development and improve practical solutions.

Zero Trust

The concepts of Zero Trust have been present long before the term "Zero Trust" was invented. Early descriptions include the work by the publication of the Jericho forum ("Jericho Forum Commandments")[2] motivating to move from a perimeter-based security model towards individually secured transactions. Subsequently, the evolution of the more broader concept was introduced as Zero Trust by John Kindervag at Forrester [3]. The term then became a synonym for the paradigm shift from implicit trust in traditional network security based on physical location towards individual resource validation and access control on transaction basis.

In essence, a Zero Trust Architecture enforces proven security principles by removing implicit trust boundaries. This concept does not only apply to network security, but to all areas of information technologies. In order to apply these principles of explicit resource control to software platforms, the historically monolithic structure and static policy enforcement must be re-evaluated. The idea of resource isolation through virtualization is well known and has seen wide adoption especially in cloud environments. Due to a number of factors including resource constraints and technology support, research and development is required to bring ZTA to the edge.

There are many concurrent definitions and interpretations of ZTA. According to the Zero Trust Architecture document by NIST (SP - 800-207)[4] , the basic tenets of ZTA are

1) All data sources and computing services are considered resources.
2) All communication is secured regardless of network location.
3) Access to individual enterprise resources is granted on a per-session basis.
4) Access to resources is determined by dynamic policy—including the observable state of client identity, application/service, and the requesting asset—and may include other behavioral and environmental attributes.
5) The enterprise monitors and measures the integrity and security posture of all owned and associated assets.
6) All resource authentication and authorization are dynamic and strictly enforced before access is allowed.
7) The enterprise collects as much information as possible about the current state of assets, network infrastructure and communications and uses it to improve its security posture.

The Ghaf platform aims to **apply the general security principles of zero trust** within the software architecture, and **act as an enabler for Zero Trust Architectures** for edge devices within organizations.

Applying Zero Trust Principles

The Ghaf platform applies ZTA tenets by isolating and controlling relevant system resources. A specific implementation highly depends on the hardware capabilities for virtualization and its performance impact, especially considering available process and memory isolation mechanisms.

Hereby, not only code and data in separate security domains, but all system components including peripherals are considered resources and must be isolated and controlled. This is ideally supported in hardware by virtualization support and I/O memory management units (IOMMU), and can be further enhanced by moving untrusted code such as peripheral drivers into virtualized environments. In many use cases the respective hardware support for virtualization is limited, which requires significant efforts to implement the isolation requirements, if at all possible. Similar challenges are present in process and memory isolation due to memory pipelining, cache structures, and a vast amount of possible side channels.

Each use case and implementation requires a specific security model that accounts for respective hardware and virtualization support (or lack thereof). An example is the virtualization of graphic memory, where performance is crucial for the user's experience. Lack of appropriate hardware support will inadvertently impact the security in favour of performance as complete memory clearing during context switches may not be practically feasible. Another example is hardware component authentication and attestation, which is only possible if supported by the hardware supplier, which in many cases is not achievable in real-world devices without an extensive (and expensive) network of hardware suppliers.

In order to control and monitor traffic within a virtualized platform, communication between the VMs is a crucial part of the systems design and an area of active study. Again, considerations of performance and security trade-offs play an essential role in the design. If, for example, sockets are used for inter-VM communication a number of concepts such as authenticated and protected sessions can be implemented, but usually at the cost of overhead and resulting performance loss. One objective of the platform implementation is to minimize these losses while simultaneously supporting the security functionality.

Following the tenets of ZTA, enforcement of dynamic security policies can be implemented as an internal policy engine within a specific virtual machine with a majority of policy enforcement residing in the virtual machine monitor, which is a natural location for resource control in virtualization. Moreover, security functionality such as traffic monitoring, attack detection and response mechanisms, and resource supervision can further be extradited to dedicated virtual machines. The extent of dynamic vs. static controls to administer system resources is another performance critical decision in the architecture. While strict monitoring and control has obvious security benefits and follows the ZTA tenets, it may not be efficient or possible due to constraints in the system.

Thus, resource monitoring, data collection, attack detection and response, and overall policy application with minimal overhead is an active area of research in the embedded context.

Enabling Zero Trust Architecture Implementation

The following illustration shows the Zero Trust Pillars according to the DoD's Zero Trust Reference Architecture [5].

In the following, we will briefly look into each pillar and highlight some of the impacts of a software architecture following the ZT security principles, and the potential benefits that it can contribute to an organization's adoption of the Zero Trust Architecture.

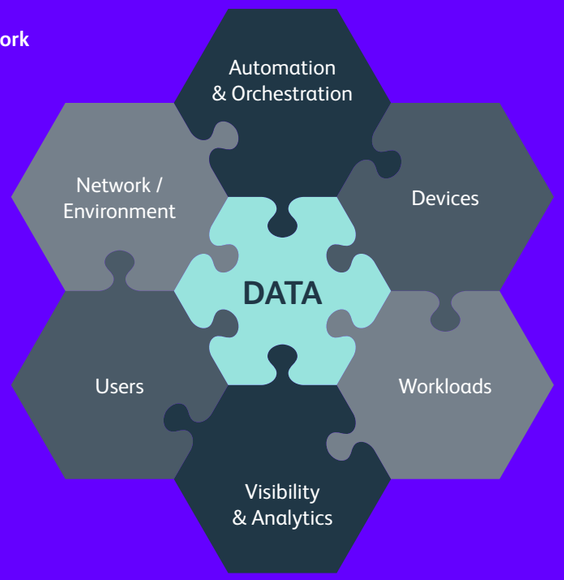
User: ***"Organizations need the ability to continuously authenticate, authorize, and monitor activity patterns to govern users' access and privileges while protecting and securing all interactions."***

The application of continuous authentication and authorization of resources is compelling. An important factor is to detect malicious behavior, not caused by a genuine user but by a malicious actor, e.g., through a compromised application.

In addition to mechanisms such as attribute-based authentication (e.g., behavioral analysis), applications' access to both internal and external resources can be controlled in the Ghaf platform through static and dynamic policies even when traditionally highly privileged parts of the system are compromised.

Device: ***"Continuous real-time authentication, inspection, assessment, and patching of devices in an enterprise are critical functions."***

Zero Trust Framework



ZT Pillars were identified by the ZT Strategy. These associate to an interlinked group of strategic resources.

Protecting data is at the center of ZT goals and is a part of all other resources.

All the resources are bound into the Zero Trust Framework.

Figure 2: Zero Trust Pillars (source: DOD Zero Trust Reference Architecture[5])

There are numerous mechanisms that already allow control of resources within edge devices, however, the partition of system and application functionality into separated compartments enables policy enforcement by design. In real-world devices, it is common and often necessary to run untrusted third-party applications (or even firmware and drivers). The virtualization approach in the Ghaf platform can allow granular levels of resource access control to user data and peripherals within the device. Moreover, the architecture allows for the implementation of a central low-level point for continuous automated inventory and telemetry data collection, which is fundamental for evaluation in the ZTA's analytics backend and policy decision point (PDP). Discrepancies between current and previous activity patterns can lead to different policies being applied to a device, and the virtualization enables dynamic and seamless service updates.

Network/Environment: ***"Segment (both logically and physically), isolate, and control the network/environment (on-premises and off-premises) with granular access and policy restrictions."***

Likewise, granular access and policies can be enforced through the isolation of any external connections through various communication channels (WiFi, Bluetooth, USB, etc.) into a separate, isolated compartment **(see Connection VM)**. This is especially useful for the implementation of security functionality that may not be natively supported by a third-party operating system. Further, the Ghaf platform architecture enables micro-segmentation of networks, which is especially interesting for an application landscape with diverse levels of trust.

Applications and Workload: ***"Securing and properly managing the application layer as well as compute containers and virtual machines is central to ZT adoption."***

A central part of the Ghaf platform architecture is to allow the separation of applications with different levels of trust, e.g., a fully audited application and an application with known privacy concerns **(see also section Cloud-hybrid Edge Computing)**. In addition, several approaches are being investigated to determine solutions leveraging an optimal balance between cloud and local resource usage. The platform aims to enable fast and efficient integration of local and cloud applications.

Data: ***"A clear understanding of an organization's DAAS is critical for a successful implementation of a ZT architecture."***

Similar to applications and workloads, data residence and protection is highly dependent on the specific use case. As briefly discussed in the section **Product Applications**, the data security model and its implementation depends on a number of factors such as connectivity, storage capacity, privacy concerns, and isolation capabilities. Contrary to monolithic designs, virtualized file systems and memory isolation capabilities allow an individual exposure of data towards guest applications, enforced by the hypervisor, enabling fine grained control as required in the ZTA. The proposed architecture aids data isolation by design, allowing to grant and remove data access permissions based on a variety of available controls through policies.

Visibility and Analytics: ***"Contextual details provide greater understanding of performance, behavior and activity baseline across other ZT Pillars."***

The isolation and resulting inter-machine monitoring capabilities of the Ghaf platform allow to capture data streams and contextual information that can be useful to identify malicious activity as well as additional insights into a device's telemetry. This enables the deployment of analytic services such as advanced intrusion detection or continuous authentication by allowing to gather contextual information at the source. Hereby, the distinguishing feature of the Ghaf platform is the possibility of unified data collection and analytics across different target use cases. While not removing the need for OS-level security features, a unified interface promises to ease integration of collection and analytics features into an organization's ZTA, independent from a specific software stack.

Automation and Orchestration: ***"Automate manual security processes to take policy-based actions across the enterprise with speed and at scale."***

Enabled by a centralized agent within the edge device, security responses (e.g., to a zero day vulnerability in an untrusted application) can be delivered fast with several measures that do not require the availability of patches and may have less interference with the user's experience.

The Ghaf platform promises to improve the automation and orchestration of edge devices by providing internal policy distribution and policy enforcement points throughout the platform without the need of several agent implementations tailored to specific use cases and target platforms.

Trusted Computing Base

A main objective of the Ghaf platform is to establish a trusted computing base to build secure and performant edge products. Moving traditionally privileged functionality into de-privileged domains such as EL 0/1 in ARM or Ring 2/3 in x86, the security of the hypervisor and related functionality becomes ever more critical. Thus, a minimal, de-bloated, and monitored code base promises to limit attack surface, increase auditability, and minimize efforts for hardening, testing, and maintenance.

The general principle for establishing the trusted Ghaf platform code base is to rely on audited software and proven security modules while carefully evaluating and integrating new concepts. The modularized platform not only simplifies integration of additional security measures, but also to facilitate integration of hardware security features. Leveraging and contributing to open source projects is not only a cornerstone for the platform components maintainability, but also for the toolchain to increase transparency and auditability. By providing a hardened code base for hypervisor and OS for the various virtual machines in the architecture, the Ghaf platform leverages security benefits across all modules.

As software supply chain security becomes more and more relevant to product security, it is necessary to provide mechanisms to assert reproducible builds, with a transparent chain from source code over build environment to the final binaries. Such a system allows faster analysis of not only software bugs, but also security vulnerabilities and their impact to a product without the need of extensive analysis. This approach further reduces the efforts required for patching, and allows mechanisms for safe fallbacks to secure states.

Last but not least, a minimal trusted modular code base promises a longer life-cycle than complex operating systems under heavy development. By removing the elevated trust privileges of operating systems and proprietary binaries such as device drivers, the impact of bugs and vulnerabilities are reduced, and the platform's most security critical components are likely to remain more stable. In addition, secure applications running on top of a minimal system are also expected to require less maintenance effort. One potential disadvantage is the effort to separate out OS functionality, which may require additional development resources for productization.

Build System and Supply Chain

Supply-chain attacks against software have become more and more prominent and the focus of attacks in recent years, with famous examples such as Solarwinds, Microsoft Exchange server, and British Airways payment gateway making headlines. In general terms supply-chain attacks are a threat where malicious or vulnerable code is inadvertently included into a legitimate product, often by infiltration of a third party supplier.

This can be achieved at any point within the product's lifecycle. The Executive Order 14028 [6], issued by the President's office of the USA has declared that a Software Bill Of Materials (SBOM) must be provided for all software that is delivered to the government. Meanwhile, industry proponents of Supply Chain Security (SCS) have long started to categorize and define standards in this regard. The cross-industry collaboration Supply-chain Levels for Software Artifacts (SLSA) framework [7], for example, aims to standardize the terminology and the methods to verify and deliver software.

One objective of the Ghaf platform architecture is to provide developers with a trusted code base for the development of edge devices while applying ZTA principles and simplify its adoption. In line with the principle to "never trust and always verify", the code base will be completely open source. This allows developers to perform security assessments in accordance with their supply chain governance requirements. To this end, SSRC has started to improve their own development process, with quality and security assurance in line with the SLSA framework and beyond; including third-party reviews, automatic vulnerability scanning, build environment monitoring, and automated SBOM generation.

One step into this direction is the move towards reproducible builds in the next platform generation. Therefore, the generation under development uses the Nix package manager and flakes [8] (an experimental feature at the time of writing), which allows pinning of packages and source code (e.g., Git or Mercurial sources) to their exact version, taking a step further into the direction of pure, reproducible builds. The Nix ecosystem further provides a basis for automated vulnerability (CVE) scanning (vulnix [9]) and continuous integration (hydra [10]). Ultimately, the ambition is to provide not only trusted images/binary caches, but also a reproducible build environment that can be used to independently build, verify, and share build results.

Platform Architecture

The main architectural concept of the Ghaf platform is breaking up the traditional monolithic structure in favour of modularized components. Figure 3 displays the components and complexity of a traditional phone architecture.

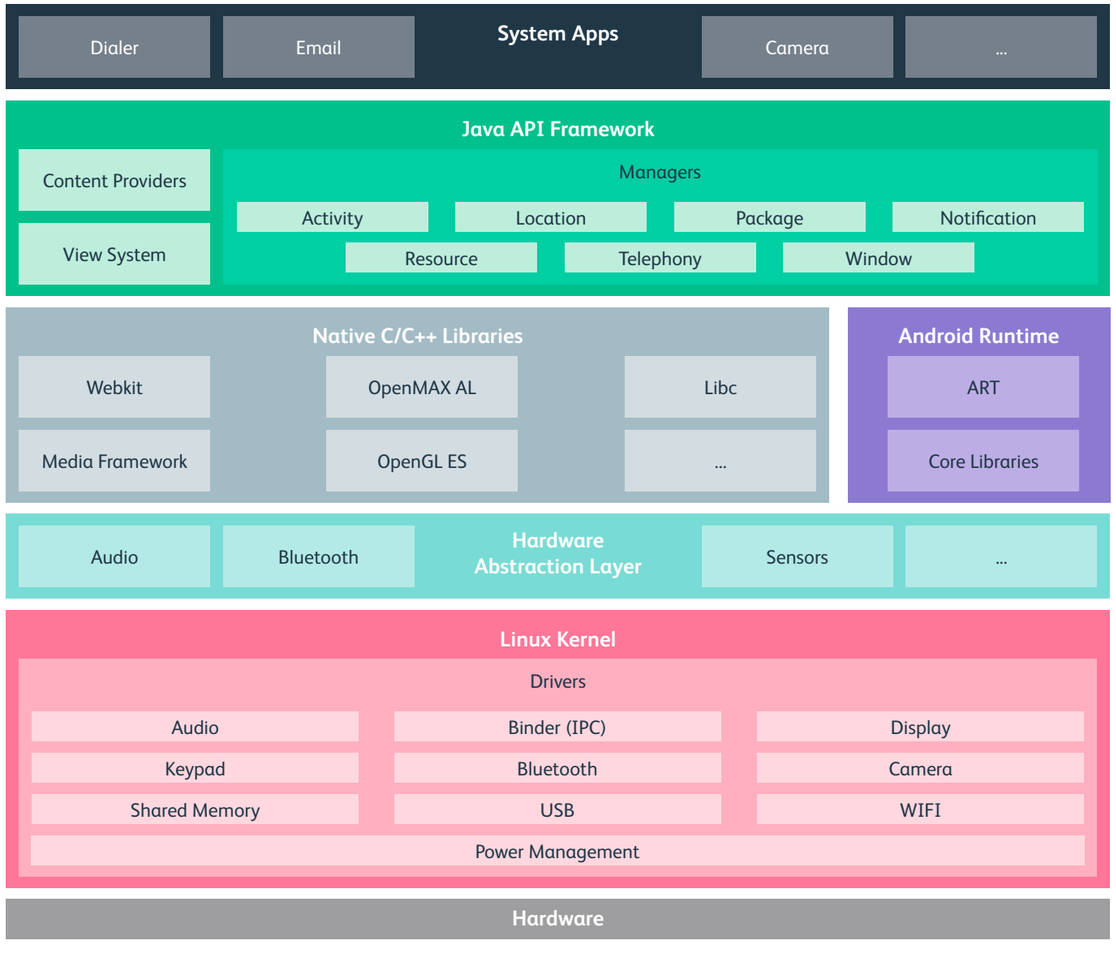
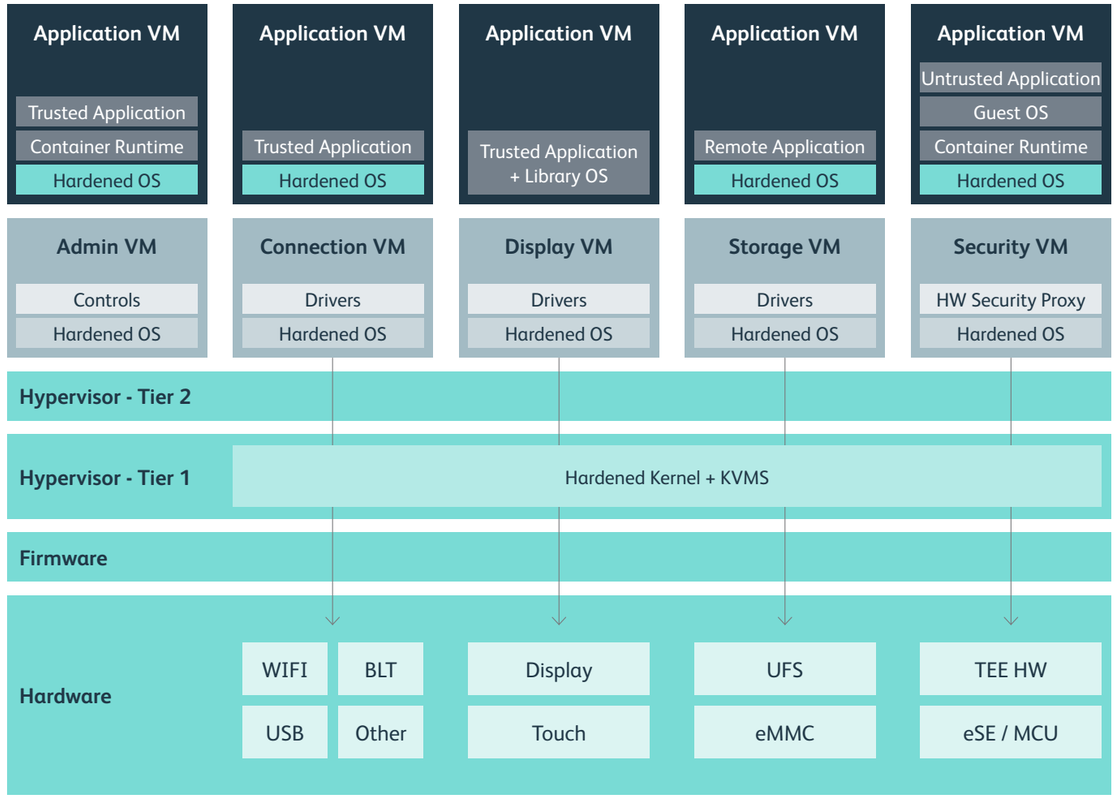


Figure 3: Traditional (Android) phone platform architecture

The Ghaf platform architecture moves traditional OS functionality into isolated (and if possible less privileged) environments; more precisely, into separate virtual machines dedicated for this purpose. These purposefully constructed virtual machines handle important system functions such as graphics, communication, storage, security functionality, and other peripherals, orchestrated and controlled by the Admin VM.



In the following sections, we will dive into the architecture and have a closer look into the different compartments and highlight some of the architectural benefits and challenges.

Figure 4: Modularized platform architecture

System Host

Hardened Kernel

The Ghaf platform project heavily utilizes the Linux kernel. Each underlying hardware supports different kernel versions and configurations, so it is generally not possible to create a singular supported master kernel for the Ghaf platform host. However, as part of the Hardened OS running in the guest virtual machines the same up-to-date kernel baseline can be used for application virtual machines, whereas the specific system virtual machines require modifications to adjust to the specific use case. Upgrading the vendors board support package (BSP) to a more recent kernel is beyond feasible for most projects. Similarly, vendor reference BSP and HW support for virtualization is paramount - with no BSP and HW support, this design building on embedded virtualization is not possible. Thus, it is necessary to adapt each kernel, and ultimately build a tool chain to support kernel hardening automation for a specific hardware platform. A baseline of kernel security modules are vital to the functionality of the platform and will be present in all versions. A variant of this design with only a hardened host, when HW support is not available, is possible but suffers from the limitations of monolithic architecture.

For each supported kernel version, a minimal kernel configuration must be determined. As different versions have different levels of security patching, a gap analysis must be performed to evaluate which patches are absent in this specific kernel version, and apply these along with a selection of additional kernel patches to increase security from various Kernel hardening programs. Individual kernel configurations are created based on the specific threat model of the use case, and further extended depending on whether the kernel is used as hypervisor or Hardened OS kernel. Kernel Self-Protection [11] mechanisms are one of the core elements for kernel security, aiming to passively eliminate classes of bugs, methods of exploitation, and actively detect and respond to malicious actions. With the application of runtime kernel protection mechanisms, several considerations besides performance impact are required. One example is the deployment of active countermeasures (such as triggering kernel panics upon attack detection) which may decrease exploitability, but also makes the system more vulnerable to denial of service attacks.

The first generation of the Ghaf platform kernels was built using buildroot, managing kernel configurations using kernel fragments, which are small configuration pieces built into flat kernel config. The second generation (Nix-based) can handle generated kernel configurations in a NixOS-specific configuration file for kernel sources with out-of-Nix config generation. The Nix configuration system cannot handle kernel fragments without porting the make files and bash-scripts to Nix-language, which is part of the on-going work. Nix provides powerful mechanisms to overlay and patch flat kernel configs including the creation of unique cryptographic identifiers for kernel binary and modules, supporting the software supply chain efforts.

Hardened Hypervisor

The hypervisor, built on the foundation of the Kernel-based Virtual Machine (KVM) and Virtual Machine Monitor (VMM) are the most critical components in the Ghaf platform, as they facilitate and enforce the virtualization and fulfil respective security related functions. The Hardened Hypervisor utilizes the Hardened Kernel and subsequently a hardened version of the KVM.

The KVM (Kernel-based Virtual Machine) is a virtualization module in the Linux kernel that allows the kernel to function as a hypervisor. As KVM is integrated in the Linux kernel, it benefits from reusing Linux memory management and CPU scheduling functionality, but suffers from the huge TCB that comes along with it. The TCB can be reduced by isolating certain functionality to guest virtual machines and using a pass-through mechanism. This way the drivers are owned by the virtual machines and reduce the TCB of the critical host system.

The KVM was originally built for x86 architecture and later ported to ARM. The KVM on ARM implementation has been split into the so-called Highvisor and Lowvisor. The Highvisor lies in ARMs kernel space (EL1) and handles most of the hypervisor functionalities. The Lowvisor resides in hypervisor mode (EL2) and is responsible for enforcing isolation, handling hypervisor traps and performing the context execution switches between guests and host. On both instruction set architectures (ISA) we focus on 64-bit systems. RISC-V virtualization support is closer to the ARM implementation with U- and S-modes mapped to ARM EL0 and EL1, and M-mode similar to ARM EL3.

There are several projects extending the KVM with additional security functionality, including Google's pKVM[12], KVMS[13], and SeKVM[14], with the latter aiming at formal verification to prove security functionality works as defined. The hardening objectives for the virtualization are mostly to guarantee proper isolation of processes and memory, and detection of configuration changes. Side channel analysis or fault injection resistance in the physical domain is typically out of scope for many of these projects, due to the high amount of effort required as these vulnerabilities need to be addressed throughout the entire software stack, from hypervisor to applications. These threats are applicable for embedded edge devices and must be considered. The focus however lies on the protection against software-based vulnerabilities (including micro-architectural attacks) as they are often remotely exploitable and more easily distributable.

The project utilizes proven (Linux) security modules to counter the various attacks towards hypervisor and operating system. One active field of research is the handling and enforcement of static vs. dynamic policies within the system **(see also: Admin VM)**. Moreover, the separation of KVMS (Lowvisor) into EL2 (on ARM architectures) allows to isolate a security relevant part of the hypervisor into a higher privileged CPU context. As one of the zero trust principles is to remove as many implicit trust boundaries as possible, a first step is taken into this direction for the hypervisor. The KVMS attempts to augment the security of the traditional KVM architecture by limiting the access of the host to guest memory. This is achieved through the "host blinding" feature and is enforced by the privileged part of KVMS (Lowvisor), which leverages multi-level address translation to restrict the access of the Highvisor to guest memory. The applicability of this solution depends on the use case as host blinding may impact the ability for data collection.

There are now multiple Virtual Machine Monitors (VMMs) utilizing the KVM interface including qemu, kvmtool, crosvm, cloud-hypervisor, and Firecracker, each focusing on specific use cases. The majority of popular VMMs are based on rust-vmm, a modular framework with contributors from Alibaba, AWS, Cloud Base, Crowdstrike, Intel, Google, Red Hat as well as individual contributors. It provides a set of virtualization components that any project can use to quickly develop virtualization solutions. Using rust-based frameworks provides more robust memory management compared to C/C++ due to its enforcement of memory safety.

Hardened OS

The Hardened OS is a minimal OS that runs in both the host system and the virtual machines. Its configuration and modules are defined based on the requirements of the use case and threat model. Generally, lightweight OS components are preferred as motivated by the design principles. In order to reduce attack surface, many of the components receive specific hardening and debloating (removal of unused functionality) where possible. This concept allows for a system that only implements the necessary code to run.

One actively researched concept is the LibraryOS approach (e.g., Unikraft [15]) to compile all dependencies into a single binary. While seemingly in contrast with the Ghaf platform philosophy, its usage can be very beneficial due to the natural de-bloating, dead code elimination, link and compile time optimizations, and overall very high performance from context-switch elimination. Additional security benefits besides the small code base is the removal of system management tools, e.g., an attacker cannot access a shell, and compiler optimizations and security features such as address-space layout randomization (ASLR) and control-flow integrity (CFI) can be applied consistently through the entire stack. The main application for it in the Ghaf platform is currently envisioned within the Application VM space.

In many use cases, the Hardened OS uses containers to implement further isolation in user space. Containers introduce more isolation to operating system process boundaries without sacrificing performance. Despite the similar name to real virtualization, containers cannot provide the same security claims. An escape from a container can compromise the operating system providing the container runtime. A hardened container approach aims to secure Hardened OS user space virtualization using “open industry standards around container formats and runtimes” - Open Container Initiative (OCI). OCI members include major industry key players and was established by Docker, synonymous with containers, in 2015.

The primary motivation to utilize container runtimes is to ease application development, deployment, and portability of solutions. It allows developers to use tools and interfaces they are familiar with, while enabling compartmentalization into self-contained environments that can be efficiently administrated.

Container hardening is mainly focused on debloating the industry leaders container runtime stack to only the open, security audited, low-level tool runc “for running and spawning” OCI compatible containers - including Docker. Ghaf platform reference hardened containers are run in non-privileged (rootless) mode. In addition to runtime security, the OCI image specification compliant containers in Hardened OS are security audited and out-of-system containers cannot be installed using the Hardened OS tools. More information and benefits of containers can be found in the section **Application Virtual Machines.**

System Virtual Machines

Ghaf is a modular platform with a collection of virtual machines that can be configured to define a system. This section introduces example systems, the platforms System VMs and their functionality. Figure 5 shows an example headless device with no local graphical user interface (GUI).

Virtualization supported hardware resources are isolated with a hardened hypervisor on the system host. The host uses the resources to launch and monitor three VMs: Connection VM, secure/trusted App VM, and an untrusted App VM. Trust levels are indicated with colors between blue (trusted) and red (untrusted).

This example system could be an autonomous robot, e.g., a drone. A user can monitor the system over secure network connection. This kind of a device with embedded virtualization is similar to a headless cloud application.

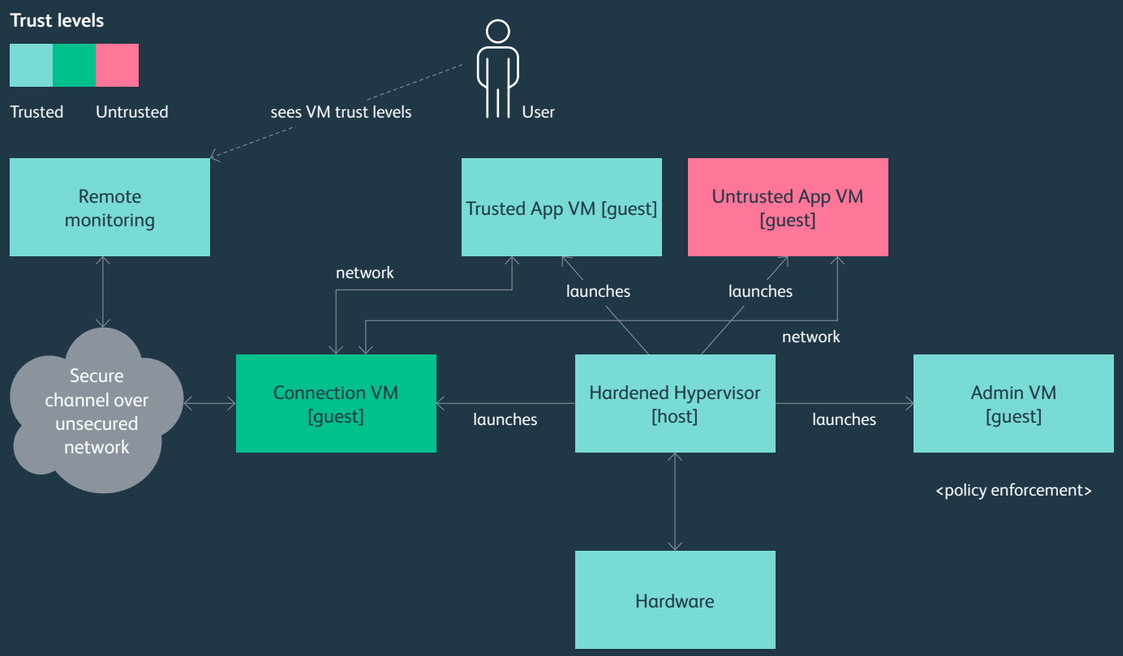


Figure 5: Simple headless edge device use case

Another example system is illustrated in Figure 6 where the user interacts with the system using a graphical user interface.

Figure 6 adds a Display VM to the system for user interaction and isolating application use cases with secure chat (trusted) and browser (untrusted) from each other. In addition to the hardened Display VM, the GUI protocol between the Display VM and Application VMs is used to isolate the VMs from each other and ensure no confidential data is leaked when the user switches between applications. This kind of architecture can be leveraged in devices with embedded virtualization such as a secure phone, a tablet, or a laptop.

For virtualization of hardware drivers, the System VMs can utilize both passthrough and paravirtualization, each implying different advantages and disadvantages.

While paravirtualized architectures provide better portability, the performance constraints are a critical factor. Specific implementations for particular hardware and use cases will be described in separate case studies to be published, showcasing the various challenges, solutions, and architectural decisions.

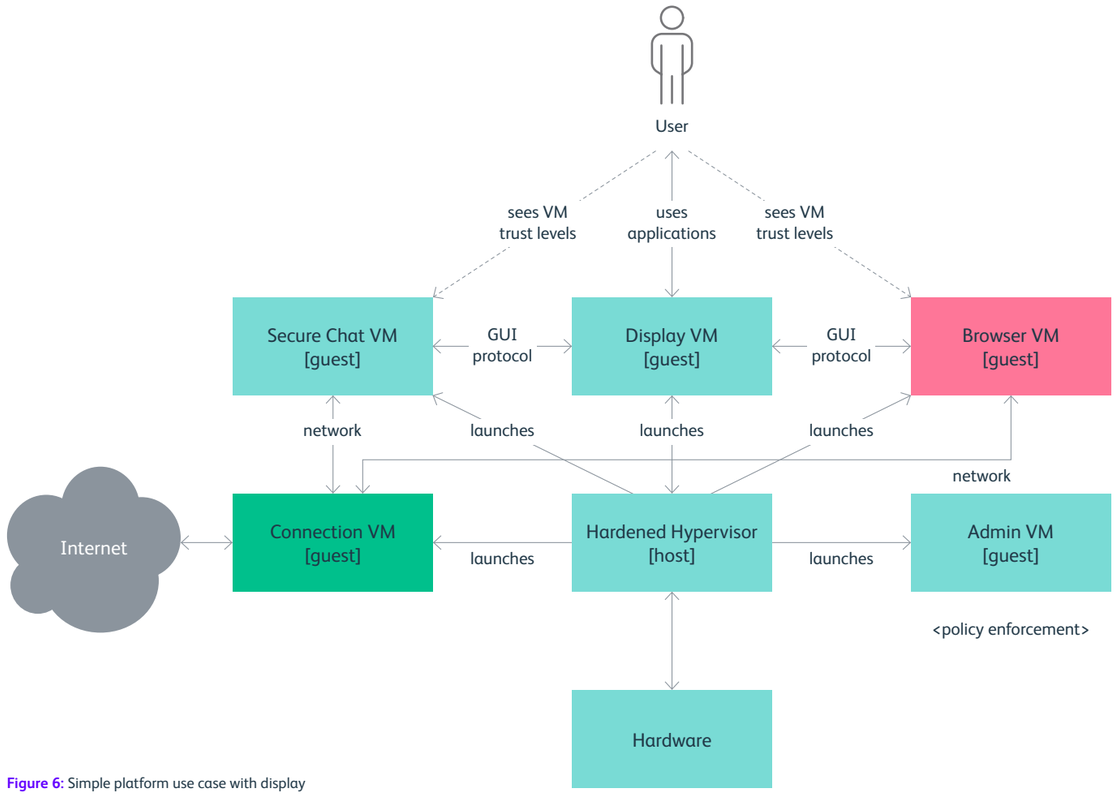


Figure 6: Simple platform use case with display

Admin VM

The Admin VM is used to monitor and control system and application virtual machines. This includes resource monitoring as well as functionality to set and update policies, thus functioning internally as policy engine and externally as policy enforcement point (PEP) and Endpoint Agent in the ZTA.

Even though the Admin VM may not directly control all resources, it can be utilized to administrate the different endpoints to enforce policies. Monitoring and controlling the traffic between guest VMs can be implemented via traditional firewalls in case TCP/IP based communication is utilized. The Admin VM is in charge of these policies, and can be extended to function as a central communication node if, for example, the security plugins require packet inspection capabilities within the internal network. Internal VPNs may be used to enable authenticated and protected channels between VMs, or can be used as a bare minimum authentication mechanism without traffic protection. The modularity also allows to deploy traffic inspection and analysis for only specific use cases, certain security status levels, or not at all and can be extradited to specifically purposed VMs.

There are several limiting factors to consider as high performance requirements for components such as GPUs often require direct memory access, where monitoring overhead for functionality besides simple memory boundary checks may not be acceptable.

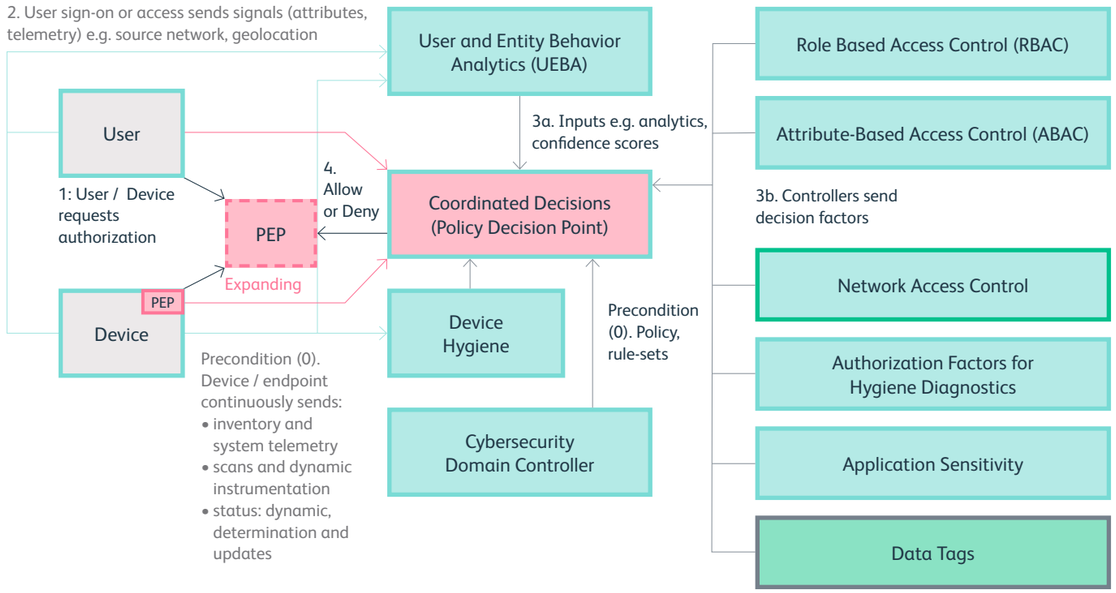


Figure 7: Device as policy enforcement point in the zero trust architecture (source: ZTRA [5])

Dynamic policy setting, attack detection and response, and dynamically scalable security measures are active research topics. While ML-based detection engines appear to yield reasonable results, scalable security is of high importance due to the performance constraints in edge devices. This affects data collection mechanisms, narrowed down by principal component analysis of data sources to limit resource usage, and extradition of analysis into the cloud (for Analytics Engines, SIEM/SAOR, and PDP) to reduce workloads on the edge device. Note that in use cases such as autonomous systems this may not be possible, and the limited hardware resources have to be utilized. Another crucial advantage is the possibility to develop unified data collection across multiple devices, potentially independent from hardware and/or software such as different client operating systems. Unified analytics for Data, Applications, Assets, and Services (DAAS) is one of the cornerstones of the ZTA.

The Admin VM can propagate security and confidence levels as well as policies throughout the platform to both system and application virtual machines. Upon detection of suspicious activity, system logs and other data collection mechanisms can be enabled, for instance, system call tracing in the guest for ML-based malware detection, which has yielded significant results. On the edge device itself, gathering data for analytics can be a key enabler to detect compromised systems, whereas the analytics itself can be implemented in the SOAR backend to determine and propagate appropriate policies.

One way for the Admin VM to respond to attacks is its ability to restart virtual machines on the fly, thus re-loading and authenticating code and data to remove non-persistent compromised code (safe fallback mechanism). This measure does not solve the underlying problem, however, an attacker will have to re-run the exploit. A dynamic security policy allows to deploy additional security measures such as cutting I/O connections, enabling more stringent code flow integrity or memory protection mechanisms, reboot or halt the VM to limit the window for and extent of data exfiltration. Vulnerable virtual machines can ultimately be patched on on-the-fly once security issues are identified and fixes available.

Connection VM

The Connection VM is responsible for controlling I/O interfaces and devices such as WiFi, Bluetooth, and USB. While each of these could be separated into their own VMs, some architectures may work better using a singular Connection VM for performance reasons.

The illustration in Figure 8 showcases an instance of a Connection VM, in this case handling networking for an Android OS. While being controlled by the Admin VM, networking can also be turned off by the user, e.g., to cut all wireless communications (airplane/secure modes). Firewall configurations for external traffic are controlled by the Admin VM and applied within the Connection VM as policy enforcement point. The isolation of network functionality further facilitates micro-segmentation of networks, e.g., to compartmentalize traffic streams from different resources both internally and externally. Alternatively, the network resources can be administrated on service level in the spirit of zero trust. While all of this functionality may be implemented in monolithic systems, the platform reinforces this concept through its architecture.

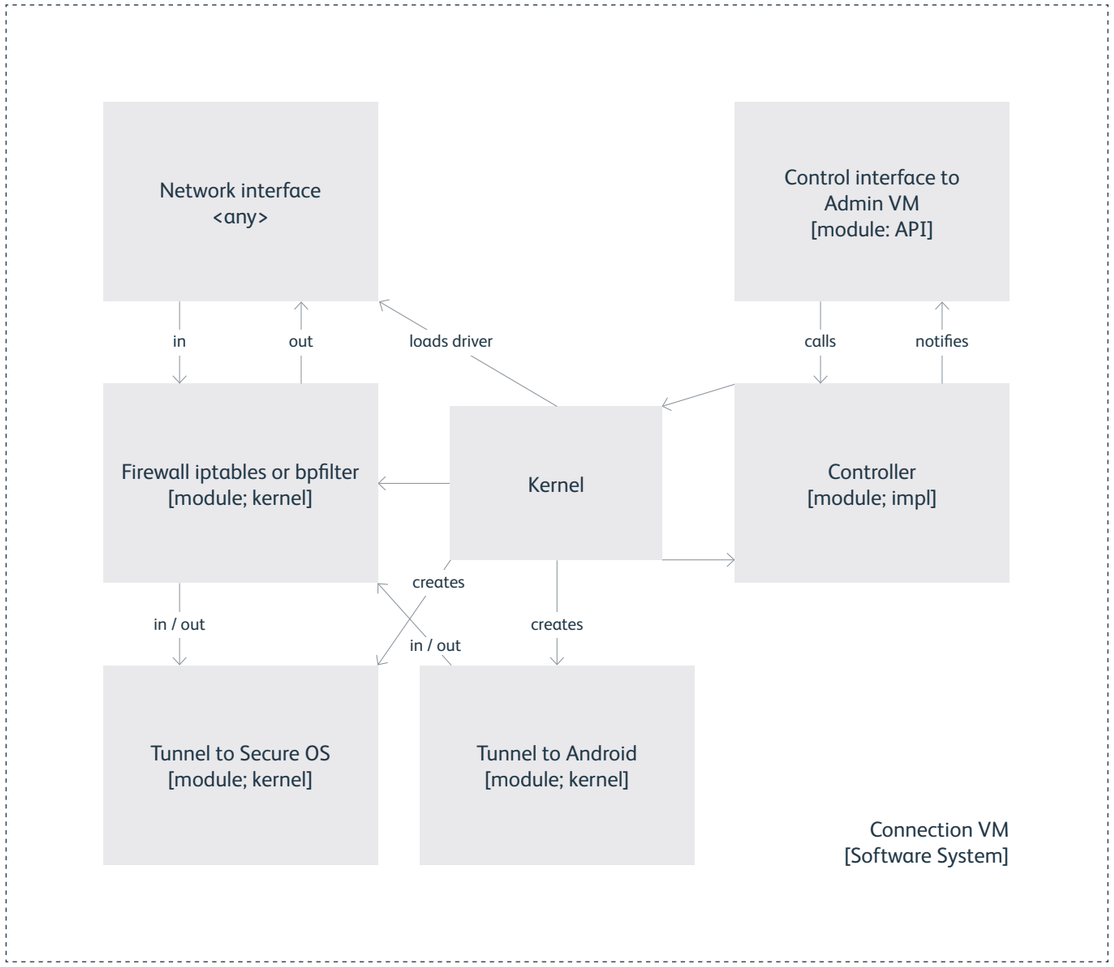


Figure 8: Internal data flow and modules of the Connection VM

Display VM

The key functionality of the Display VM is to provide graphical user interface services. GUI services are needed by secure chat applications, Cloud Android WebRTC client, and other apps. The Display VM uses hardened OS kernel as baseline and provides display services with event devices, display, display server, compositor, and window manager. The Ghaf platform reference implementation for Display VM started out with containerized and secured X11, but has since moved to the more performant Wayland[16] for streamlined memory management and a more sleek code base compared to the X system.

The complex Virtio Wayland architecture and various implementation options and their specific vulnerabilities require significant efforts for security assessment and hardening. Due to the fact that the Display VM effectively communicates with most applications, secure and insecure, the Display VM is a high risk component. Further, varying GPU virtualization support and direct memory access requirements decrease the amount of control that can be exercised efficiently.

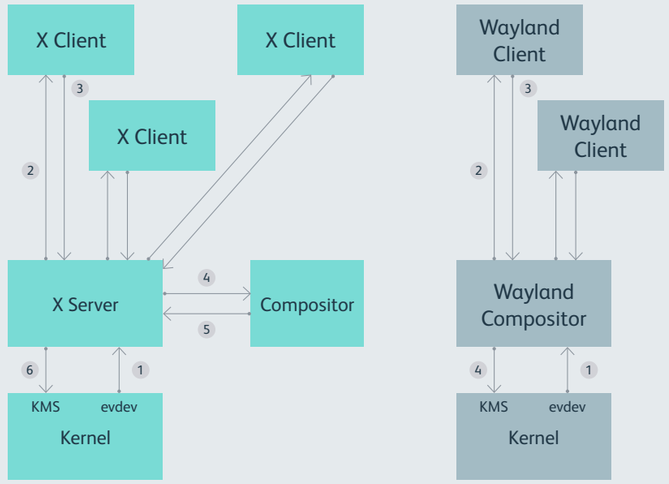


Figure 9: Simplified X-system and Wayland architecture (source: [16])

Security VM

The Security VM acts as a generic interface for hardware or software based security environments. Independent on the capabilities of the underlying platform, the Security VM provides security functionality towards virtual machines. This is beneficial as several application virtual machines require utilization of hardware backed mechanisms for key storage, attestation, and other cryptographic operations.
The Security VM implementation thus acts as a proxy, which allows it to apply fine grained resource access controls. As secure key storage and cryptographic primitives are highly critical resources, this virtual machine plays a key role in the platform. The architecture allows several different underlying implementations, and can make use of hardware based TEEs such as TrustZone and SGX, but also as proxy for secure elements and enclaves, hardware fuses, and proprietary secure microcontrollers. In case such hardware is unavailable (although highly discouraged), software implementations can be used to replace the required functionality. The ability to use such a dedicated virtual machine and respective functionality highly depends on the use case.

Storage VM

The Storage VM is responsible to isolate user and critical system data and provide data at rest protection. The key idea is to only expose data to an application/guest virtual machine that is required. Facilitated by hardware crypto engines, data is encrypted at rest, and optionally not stored on the device but loaded from a backend **(see also Cloud-hybrid Edge Computing).**

Although RAM memory is arguably the most critical resource of a Guest VM, persistent memory also poses a significant threat in case of a compromised host. Since the persistent storage is required by all VMs, in addition to the fact that many platforms (e.g., mobile phones) offer a single flash peripheral, it is practically impossible to dedicate flash to separate VMs. As such, disk I/O is handled by the host in the current architecture. Considering that disk I/O is emulated for guests, a direct guest-to-guest attack should not be possible without implementation bugs in the VMM. A compromised, blinded host may be able to indirectly compromise a guest through the emulated I/O functionality, for example, by modifying the executable when the guest attempts to map the binary in its memory. There are integrity features (e.g., dm-verity) that can detect such attacks, up to a different degree each, thus requiring guests to employ such controls. While these mechanisms could potentially be tampered with as well, it requires an attacker to bypass an additional barrier.

Application Virtual Machines

Introducing untrusted and potentially dangerous third-party applications to a security solution exposes all protected assets to new threats. User, system, and organizational data could be at risk through insecure third-party apps and services by introducing individual remote attack vectors and requiring wide access to the system's resources to function and opening the attack surface of the system with functional dependencies. All of these should be mitigated with full isolation, preferably backed by hardware, to prevent compromising the security of the system. While application level vulnerabilities have to be addressed within the application itself, the platform design attempts to mitigate lateral movement and reduce attack surface.

Application VMs (aka Guest VMs) can technically run anything - from Android OS to a simple headless offline application. As we have seen, the architecture is built to reduce the exposure of each (potentially compromised) application to the system (guest-to-host), to other applications (guest-to-guest), and to itself through the host or other guests. Further, measures have been taken to reduce exposure of the guests (host-to-guest, see Hardened Hypervisor).

Trusted applications can be built on the basis of Hardened OS and utilize containerization for further isolation. Containerized applications offer security advantages over regular user processes due to user namespaces and control groups. They provide an additional layer of isolation when running several applications in parallel. Containers help alleviate some classes of attacks that Linux applications usually suffer from, such as

- sharing of filesystem between processes leading to attacks through path handling, race conditions and abuse of file permissions,
- unconstrained sharing of system resources that can allow an application to exhaust resources of the whole system,
- access through local network (localhost), which is generally overlooked by system administrators and tends to be less restricted than access from/to external networks,
- abuse of the Linux DAC permission system, especially SUID binaries which are a very common target for unprivileged malicious applications looking to escalate privileges on the host system, and
- exploit of kernel-user interfaces like procfs, ptrace, and various system calls.

Sandboxing through namespaces gives applications an isolated view of the system, which is primarily used by container engines for the purpose of replicating runtime environments across different machines. It also has the advantageous consequence of preventing most attacks cited above: each container has a unique filesystem, network interface, PID and DAC credentials as a result of grouping processes in several user namespaces. The issue of overusing resources is largely solved by cgroups and seccomp policies preventing most dangerous system calls.

Cloud-hybrid Edge Computing

Third party services, especially social media services, introduce new threats against user identity and user privacy. There is nothing that can be done to prevent a user from compromising their privacy or identity via the social media service, for instance registering with their own name, posting publicly, or falling victim to identity theft due to a weak password. However, the application can be isolated so that it has no access (or access to fake) device identity, user history/log data, recording devices, location services, or other potentially compromising resources. These isolation and privacy countermeasures could be deployed in the device, in the back-end with remote access to the third party application, or a combination of both. Tracking and digital fingerprinting is efficiently mitigated by running the third party application in the back-end in an isolated disposable context, if so required.

The Ghaf platform aims to make integration of cloud services easy. Besides the capability to deploy security policies to edge devices, one concept that has been extensively explored is to use a containerized Android OS in the cloud (using Google's cuttlefish [17]) while running an isolated application on the phone to use the hardware. The security and maintenance benefits are rather obvious, however, this solution has limits in its usability as it requires permanent connectivity.

For this reason, cloud-hybrid solutions are being explored and tested that store data in the cloud and load on demand, or run specific applications remotely. This can encompass both trusted applications and untrusted or privacy concerning applications, e.g., for anonymization or additional security monitoring utilizing the abundant cloud resources.

Conclusion

With the Ghaf platform, SSRC Secure Technologies is building a reference architecture and platform for developing secure edge devices. With the focus on scalability and security, this platform heavily utilizes virtualization to enable isolation within the software architecture that promises fine grained security control and low(er) maintenance effort.

Besides significant engineering efforts to solidify the basic building blocks that enable fast adoption to different hardware, active research areas include efficient implementations for dynamic policy application as well as attack detection and response mechanisms to enable scalable security measures. Hereby, the architecture supports the implementation of the ZTA's software defined perimeter, and enables the unified collection of device inventory and telemetry data for analysis and policy decisions in a zero trust architecture.

The efficient resource sharing between edge device and cloud backends is actively researched, impacting both data and services. The platform architecture is designed to simplify this resource management, however, hardware and connectivity constraints require optimized solutions depending on the product use case.

As can be seen, the Ghaf platform architecture aims to bring the well known security benefits of virtualization and compartmentalization from the cloud to the edge, and facilitate the implementation of zero trust architectures. SSRC aims to contribute to this space by continuing to support open source projects as well as publishing a series of technical whitepapers about our research.

Glossary

Terminology	Description/Definition
Analytics	Information resulting from the systematic analysis of data or statistics. This analysis includes discovering, interpreting, and communicating significant patterns in data.
ASLR	Address space layout randomization
BSP	Board Support Package
CFI	Code flow integrity
COTS	Commercial off the shelf
CVE	Common Vulnerabilities and Exposures
DAAS	Data, Applications, Assets, Services
EL	Exception Level
Endpoint Agent	Client software installed on a network endpoint that communicates or is controlled by a centralized system.
FDE	Full Disk Encryption
FPGA	Field-programmable gate array
GPU	Graphics processing unit
KVM	Kernel-based Virtual Machine
ML	Machine Learning
MMU	Memory Management Unit
OCI	Open Container Initiative
Policy	Statements, rules or assertions that specify the correct or expected behavior of an entity.
PDP	Policy Decision Point Mechanism that examines requests to access resources, and compares them to the policy that applies to all requests for accessing that resource to determine whether specific access should be granted to the particular requester who issued the request under consideration.
PEP	Policy Enforcement Point This system is responsible for enabling, monitoring, and eventually terminating connections between a subject and an enterprise resource. The PEP communicates with the Policy Administrator (PA) to forward requests and/or receive policy updates from the PA. This is a single logical component in ZTA but may be broken into two different components: the client (e.g., agent on a laptop) and resource side (e.g., gateway component in front of resource that controls access) or a single portal component that acts as a gatekeeper for communication paths.
SBOM	Software Bill Of Materials
SIEM	Security Incident and Event Manager
SLOC	Source lines of code
SOAR	Security Orchestration, Automation, and Response
SSRC	Secure Systems Research Center
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPN	Virtual Private Network
ZT	Zero Trust
ZTA	Zero Trust Architecture

References

1 Google. “The Generic Kernel Image (GKI) project.” Android Open Source Project, 25 October 2022, https://source.android.com/docs/core/architecture/kernel/generic-kernel-image. Accessed 19 December 2022.

2 “Jericho Forum Commandments.” Open Group Collaboration, 2007, https://collaboration.opengroup.org/jericho/commandments_v1.2.pdf. Accessed 19 December 2022.

3 Cunningham, Chase. “Next-Generation Access and Zero Trust.” Forrester, 27 March 2018, https://go.forrester.com/blogs/next-generation-access-and-zero-trust/. Accessed 19 December 2022.

4 NIST. Zero Trust Architecture. Special Publication (NIST SP) - 800-207. NIST, 10 August 2020, https://www.nist.gov/publications/zero-trust-architecture. Accessed 19 December 2022.

5 “Zero Trust Reference Architecture.” DoD CIO, Department of Defense (DoD), July 2022, https://dodcio.defense.gov/Library/. Accessed 19 December 2022.

6 “Executive Order on Improving the Nation's Cybersecurity.” The White House, 12 May 2021, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/. Accessed 19 December 2022.

7 “SLSA.” SLSA • Supply-chain Levels for Software Artifacts, https://slsa.dev/. Accessed 19 December 2022.

8 “Flakes.” NixOS Wiki, https://nixos.wiki/wiki/Flakes. Accessed 19 December 2022.

9 Bronsky, Maksim. “Flying Circus.” Flying Circus, 27 July 2016, https://flyingcircus.io/blog/vulnix-v1-0-release/. Accessed 19 December 2022.

10 NixOS. “NixOS/hydra: Hydra, the Nix-based continuous build system.” GitHub, https://github.com/NixOS/hydra. Accessed 19 December 2022.

11 kernel.org. n.d. “Kernel Self-Protection — The Linux Kernel documentation.” The Linux Kernel Archives. Accessed December 19, 2022. https://www.kernel.org/doc/html/latest/security/self-protection.html.

12 Google. 2022. “Security.” Android Open Source Project. https://source.android.com/docs/core/virtualization/security.

13 Hyvonen, Jani. n.d. “jkrh/kvms: ARMv8 hypervisor. Custom Linux KVM variant with the guest and the host memory protection, integrity verification and encryption support.” GitHub. Accessed December 19, 2022. https://github.com/jkrh/kvms.

14 Li, Shih-Wei, Xupeng Li, Ronghui Gu, Jason Nieh, and John Z. Hui. 2021. “A Secure and Formally Verified Linux KVM Hypervisor.” CS @ Columbia. https://www.cs.columbia.edu/~nieh/pubs/ieeesp2021_kvm.pdf.

15 The Unikraft Authors. n.d. “Unikraft is a fast, secure and open-source Unikernel Development Kit.” Unikraft. Accessed December 19, 2022. https://unikraft.org/.

16 Wayland freedesktop. n.d. Wayland. Accessed December 19, 2022. https://wayland.freedesktop.org/.

17 Google. n.d. “device/google/cuttlefish - Git at Google.” android Git repositories. Accessed December 19, 2022. https://android.googlesource.com/device/google/cuttlefish/.

Technology Innovation Institute LLC
P.O. Box 9639
Abu Dhabi, UAE

tii.ae