TrustedGateway: TEE-Assisted Routing and Firewall Enforcement Using ARM TrustZone

Fabian Schwarz
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
schwarz.fabianfrank@gmail.com

ABSTRACT

Gateway routers are at the heart of every network infrastructure, interconnecting subnetworks and enforcing access control policies using firewalls. However, their central position makes them high-value targets for network compromises. Typically, gateways are erroneously assumed to be hardened against software vulnerabilities ("*bastion host*"). In fact, though, they inherit the attack surface of their underlying commodity OSes which together with the wealth of *auxiliary* services available on both consumer and enterprise gateways—web and VoIP, file sharing, remote logins, monitoring, etc.—undermines this belief. This is underlined by a plethora of recent CVEs for commodity OSes and services of popular routers which resulted in authentication bypass or remote code execution thus enabling attackers full control over their security policies.

We present TrustedGateway (TruGW), a new gateway architecture, which isolates "core" networking features—routing and firewall—from error-prone auxiliary services and gateway OSes. TruGW leverages a TEE-assisted design to protect the network path and policies while staying compatible with commodity gateway platforms. TruGW uses ARM TrustZone to protect the NIC and traffic processing from a fully-compromised gateway and permits policy updates only by trusted remote administrators. That way, TruGW can readily guarantee the secure enforcement of trusted policies on commodity gateways. TruGW's small attack surface is a key enabler to regain trust in core network infrastructures.

CCS CONCEPTS

• **Security and privacy** → **Firewalls**; **Trusted computing**; • **Networks** → **Routers**.

KEYWORDS

TrustZone, Firewall, Isolation, TEE, NIC, Virtio, Router, Gateway

1 INTRODUCTION

Gateway routers interconnect networks and govern their communication using firewall policies. Therefore, gateways are attractive targets for adversaries seeking to abuse their central position for network infiltration and information leakage. While gateways were assumed to be hardened ("*bastion host*"), a series of recent CVEs has raised serious concerns over their security (Table 6, Appendix). These vulnerabilities typically arise out of non-hardened auxiliary services that execute on gateways. These services easily add up to a large, complex code base which is hard to audit (Table 4, Appendix). Therefore, many serious vulnerabilities lurk in these auxiliary services, which enable attackers to remotely compromise the gateways. Once compromised, they threaten also core services (routing and firewalling), because gateways nowadays build on commodity OSes for which several vulnerabilities and privilege escalation attacks have been revealed (Table 2 + 3, Appendix). Consequently, remote attackers can chain service to system exploits to gain full control over gateways and their policies—putting the entire network infrastructure at serious risk. As we will discuss in Section 2, the root cause indeed seems to be the increased threat surface due to *auxiliary* services and commodity systems, because, in fact, the core gateway tasks represent just a small fraction of the entire software stack and attack surface on gateways. Yet vendors keep adding a plethora of auxiliary gateway services (e.g., VoIP, file sharing, web proxies, printing, IoT hubs, content caching) to increase system utility and gain marketing advantages—at the cost of security.

Researchers and large companies have realised the need for more secure network gateways and try to re-establish trust by isolating their critical core functionalities. However, existing approaches fail to protect commodity gateways—leaving millions of home and smaller enterprise networks vulnerable (cf. Section 3). Commodity gateways relying on VMs or OS containers for service isolation [10, 27] suffer from their huge attack surface (Table 3, Appendix), while datacenter SmartNICs, which perform routing and filtering isolated from the host system, are too expensive, bulky, and complex for commodity devices. Research proposals using secure containers based on Intel SGX for routing and firewall protection [2, 17, 50] rely on future hardware support and cannot guarantee policy enforcement on stand-alone gateways due to SGX's missing hardware control over NICs, which enables a full policy bypass.

To foster widespread protection of network infrastructures of consumers and smaller enterprises, we require a design that (i) guarantees secure enforcement of a gateway's routing and firewall policies even under a system-level attacker while having (ii) a small trusted computing base (TCB) and (iii) compatibility with commodity hardware and software. However, the complexity of network subsystems, including NIC I/O and multiple layers of system software, makes it particularly challenging to come up with a design that balances *security*, *performance*, and *compatibility*. For example, a fully isolated network stack provides high protection, but at the cost of a bloated TCB and potential incompatibilities with separated commodity services, while a low TCB solution might face security limitations or high performance penalties on calls into the protected submodules. In addition, compatibility with consumer gateways is often in conflict with new efficient security technologies (e.g., SmartNICs) and might require TCB-increasing extra frameworks.

In this paper, we present TrustedGateway (TruGW), a system architecture for *commodity* gateway routers, which aims to tackle this design challenge. TruGW builds on ARM TrustZone-assisted trusted execution environments (TEEs) which provide HW-enforced memory and I/O isolation, can be easily combined with existing OS and hypervisor-based designs, and are widely available in millions of edge devices [49]. TruGW provides a new trusted networking core with a low TCB, which provides secure network I/O and traffic processing isolated from system-level attackers. TruGW leverages TrustZone (TZ) to protect the core's memory and grant it exclusive NIC access. That way, TruGW's network core has full control over the gateway's ingress and egress path and can guarantee the enforcement of trusted network policies. In particular, TruGW shows how to solve several technical challenges: (i) enable fast, trusted network I/O in spite of TZ's high context switching overhead, (ii) after NIC isolation, re-establish network access for commodity services *without* breaking security or compatibility, and (iii) allow for trusted policy configuration—all while preserving a low TCB.

Technically, TruGW implements a minimal NIC I/O framework in TZ's secure world, which provides essential network and link layer abstractions, and realises trusted routing and firewalling on top of it. Trusted policies are configured by authenticated remote administrators via a new trusted configuration service. TruGW's framework enables to incorporate only the essential I/O parts of physical NIC drivers into TZ, which preserves a low TCB. To overcome TZ's slow context-switches, TruGW designs a trusted, lightweight notifier and worker system for efficiently scheduling trusted NIC I/O, while keeping the system scheduler and threading in the untrusted world for a better compatibility and TCB. For supporting commodity services, TruGW implements a Virtio-based network device in TZ, which exposes a virtual NIC to the untrusted system for shared network access. However, to prevent network attacks by untrusted services (e.g. ARP spoofing), TruGW tightly controls and filters their traffic.

We realise an open-source prototype of TruGW[1] by extending an existing TEE with ≈10.5 k lines of TruGW-specific code. We evaluated this prototype on the Nitrogen6X dev board [16], which nicely resembles the hardware configuration of small commodity gateways. Our proof-of-concept illustrates how TruGW efficiently enforces trusted routing and firewall policies even under a system-level compromise, and thus re-establishes trust in commodity gateways and their millions of consumer and enterprise networks.

In summary, we make the following contributions:

- We raise awareness of the serious risk of remote system compromises of commodity network routers, how they undermine firewall policies, and why existing defenses fall short of efficiently protecting consumer and SME routers.
- We design TrustedGateway (TruGW), an architecture which efficiently enforces trusted routing and firewall policies under a system compromise on stand-alone commodity gateways. TruGW is tailored to balance *security*, *performance*, and *compatibility* for seamless consumer and SME deployment.
- TruGW provides a TEE-tailored networking framework, and implements a TEE-located Virtio-net device to support controlled network access by untrusted auxiliary or OS services.
- TruGW provides a low TCB, trusted web service for remote policy management with a secure admin enrollment process.
- We implement a TruGW prototype[1] and evaluate its attack surface, network performance, and secure memory overhead.

[1]Prototype available at: https://github.com/trugw

2 MOTIVATION

Gateway routers play a critical role for the security of consumer and enterprise networks. They isolate and interconnect internal client and server subnetworks, and their network firewalls serve as central gatekeepers for all ingress and egress network traffic. The gateways' central role makes them attractive targets for a network infiltration putting intruders in an ideal position for attacks. While gateways are widely assumed to be trusted, their number of services has drastically increased over the years and so did their attack surface. In fact, gateways nowadays fulfill a plethora of auxiliary functionalities beyond secure traffic control, including proxies to cloud services, edge computing, and typical consumer services such as file sharing, VoIP, streaming, or network monitoring. Table 2 (see Appendix) shows that popular gateway platforms therefore derive from large commodity OSes, typically Linux, to easily integrate such services.

This software stack composition opens up a huge attack surface. Table 6 (see Appendix) presents recent CVEs of popular network devices, that enable remote attackers control over a gateway's network policies or even the whole system—bypassing any kind of system-level defense. In fact, all these vulnerabilities lurk in auxiliary services and system software unrelated to the security-critical core networking components (e.g., firewalls). For instance, Table 4 shows 12 popular auxiliary network services on DD-WRT [18] routers, which together already include a large, error-prone code base of ≈4517 k LOCs. In addition, Table 3 shows that the widely-used Linux kernel (Table 2) has faced thousands of CVEs of which ≈10 % directly result in malicious code execution (CE)—with new ones getting steadily discovered [55, 56, 64]. In contrast, less than 100 CVEs have been reported for the Linux kernel firewall and Ethernet NIC drivers *together* with merely ≈3 direct CEs. However, the plethora of kernel and *remote* service vulnerabilities enable attackers to *fully* compromise gateways, and thus undermine also their security-critical components and policies.

Figure 1 shows exemplary consequences of such an insecure gateway in a small enterprise network. The central gateway interconnects an isolated guest, client, and multiple server subnetworks. The gateway firewall permits guests and clients to access external networks only through a traffic-filtering proxy. Furthermore, clients can access only servers of their work department, and the firewall heavily filters external ingress traffic. However, any vulnerable service on the gateway undermines these policies. Attackers can compromise the gateway using a remote code execution (RCE) against a service and perform a privilege escalation (e.g., kernel exploit) to gain access to the firewall policies. (1) A malicious guest could manipulate the firewall policies to bypass the proxy for direct external network access (e.g., to launch a spam campaign). (2) A

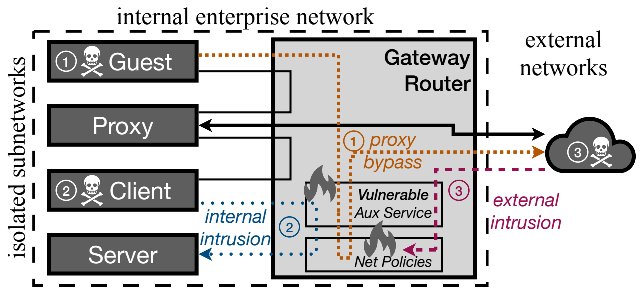

Figure 1: Three critical attacks enabled by vulnerable auxiliary services that undermine a gateway's network policies.

malicious or malware-infected client can bypass the server isolation to sabotage or steal internal secrets. Lastly, (3) if vulnerable gateway services are exposed to the public (e.g., file sharing), they enable external attackers to infiltrate the enterprise network.

Our goal is to re-establish trust in gateways by designing TruGW, a new architecture for commodity network gateways, which enforces authenticated routing and firewall policies even when the auxiliary services or system software are compromised. That way, our envisioned gateway significantly hardens the security of enterprise networks by eliminating the discussed threats. Furthermore, TruGW strengthens millions of home networks by hardening consumer routers, which include a plethora of auxiliary features (e.g., media, IoT), by providing secure traffic isolation and filtering.

2.1 Threat Model

TruGW relaxes the strong bastion host assumption for all gateway router software except of the "core networking" features—routing and firewall. We build on the common threat model which assumes a gateway located at the network perimeter and a set of internal (*int.*) and external (*ext.*) network clients trying to circumvent the gateway's security policies (cf. Figure 1). Motivated by the discussed plethora of gateway CVEs, we extend this model for TruGW in that we tolerate a system-level attacker (Sys_{gw}) which has gained major control over a gateway's software stack, including all auxiliary services, the OS, and, if available, the hypervisor (cf. Section 3). After a compromise, Sys_{gw} will attempt to leverage their central position to perform man-in-the-middle attacks, tamper with routing rules, and bypass firewall policies for full network access. We only trust verified admins which remotely manage the networking policies via secure configuration requests from trusted devices.

TruGW will root its security guarantees in hardware by leveraging CPU-provided secure containers (a.k.a. TEEs) for protecting the network traffic and policy enforcement. We therefore trust the gateway's CPU and all hardware bound to the TEE. Furthermore, we trust the software in the TEE—our trusted computing base (TCB)—and assume it to be free of vulnerabilities. While we regard Sys_{gw} in control of all non-TEE software, we exclude side-channel, denial-of-service (DoS), and all forms of physical attacks.

3 TOWARDS SECURE NETWORK GATEWAYS

We will now outline TruGW's design goals and requirements and discuss in how far alternative solutions fall short of fulfilling them.

3.1 Goals and Requirements

The goal of TruGW is the protection and guaranteed enforcement of a gateway's traffic routing and firewalling even under a full system compromise. In addition, we want TruGW to be easily integrable into commodity gateways without extra costs for wide adoption in home and (small) enterprise networks. TruGW therefore must build only on commodity hardware features and refrain from changes to a gateway's system software. At the same time, the interplay with existing gateway OSes and auxiliary services has to be efficient, and the architecture itself feature a small TCB that can be easily audited. We derive the following seven security (*SR*) and four auxiliary (*AR*) requirements that TruGW's design will fulfill:

SR1 Secure Network Setup. The setup phase must prevent unauthenticated network communication until the firewall has initialized a restrictive or restored a trusted state.

SR2 Routing and Firewall Isolation. The integrity of the routing and firewall components must be guaranteed.

SR3 Mandatory Policy Enforcment. The enforcement of the routing and firewall policies must be guaranteed.

SR4 Traffic Protection. The untrusted system must not be able to access (*confidentiality*) or tamper with traffic (*integrity*) not explicitly destined to it. This includes all forward traffic.

SR5 Spoofing Prevention. The untrusted system must not be able to spoof network addresses (e.g., MAC, IP).

SR6 Trusted Policy Changes. Only authenticated remote admins must be able to perform trusted policy changes.

SR7 Attack Surface. The trusted computing base (TCB) and exposed attack surface must be small.

AR1 Commodity hardware. The design must build only on cost-efficient commodity hardware applicable to network routers.

AR2 Service Compatiblity. The design must support existing untrusted gateway OSes and auxiliary (network) services.

AR3 Minimal Changes. The design must require only minimal changes to the untrusted commodity system software.

AR4 Network Overhead. The design must only introduce reasonably small network performance overhead to stay attractive to consumers and enterprises.

To achieve these goals, TruGW's idea is to leverage ARM TrustZone (TZ)—a widely available commodity TEE [49]—to isolate the network I/O path from the compromised system, and design new, trusted networking components. That way, even system-level attackers (Sys_{gw}) can neither tamper with network traffic or policies, nor bypass them. However, it is particularly challenging to come up with a design that fulfills multiple, partially conflicting goals, especially considering the complexity of network subsystems. For example, backwards compatibility (*AR1-3*) is often in conflict with new efficient security technologies (*AR4*) and might require additional, TCB-increasing frameworks (*SR7*), while a small TCB might limit the performance (*AR4*) or functionality (e.g. *SR3*). TruGW's main contribution is therefore to solve this design challenge and several additional challenges resulting from it (cf. Section 4 and 5).

3.2 Design Tradeoffs and their Shortcomings

Several related attempts follow similar objectives than our envisioned trusted gateway, yet fall short of fulfilling important security guarantees and/or deployment requirements. We now discuss these approaches and their shortcomings w.r.t. TruGW's properties, and motivate TruGW's decision in favour of a TrustZone-based design.

Dedicated Devices. Moving core networking services to dedicated devices could be seen as an intuitive solution to our depicted problem. While such a physical separation removes potentially vulnerable auxiliary services from the core networking devices, even dedicated routers/firewalls still have a high attack surface, including a full commodity OS (~~*SR7*~~). In addition, the extra devices introduce additional prime, energy, and maintenance costs (~~*AR1*~~). Furthermore, the declined usability (lack of auxiliary services) and the resulting need for multiple devices destroys a core marketing argument of feature-rich routers (*related to* ~~*AR2*~~).

SmartNICs and P4. In-network firewalls have been proposed for scalable enforcement isolated from vulnerable gateway systems. FlowBlaze [51] enables stateful network functions on SmartNICs for high scalability, whereas Kang et al. [28] introduce context-aware policy enforcement on P4-programmable SDN switches. While these solutions promise great scalability and security, they are too expensive, complex (*related to* ~~*AR2/3*~~), and "bulky" (form factor) for consumers and smaller enterprises (~~*AR1*~~). In contrast, TruGW focuses on protecting exactly these millions of users by providing them with an affordable gateway design for commodity hardware.

Intel SGX. Gateway designs based on Intel's commodity, hardware-isolated user space containers—so-called Intel SGX enclaves—suffer from their missing hardware control [14]. They cannot guarantee secure network policy enforcement on a stand-alone gateway, because they can neither directly access the NICs nor prevent attackers from doing so (~~*SR1/3/5*~~). Alcatraz [2] enforces firewall rules and traffic protection, but requires SGX support on every enterprise middlebox, switch, and host for per-hop tunnels (~~*AR1*~~). SafeBricks [50] and LightBox [17] securely offload middleboxes to an untrusted cloud provider using SGX, but must assume a trusted enterprise gateway to tunnel traffic to them (~~Sys_{gw}~~). SENG [57] uses SGX on the client-side to enforce trusted per-application firewall policies on the gateway, but assumes the gateway as trusted (~~Sys_{gw}~~). TruGW's focus is on *providing* such a secure design for *stand-alone* gateways, i.e., we close a gap of existing orthogonal designs.

Virtualization. Hypervisors enable a secure containment of compromised OSes and support secure I/O paths. Advanced gateway platforms by Cisco [10] and Juniper [27] already support VMs for running third-party user space services. However, for our envisioned gateway, hypervisors face two main limitations: a high attack surface (~~*SR7*~~), and compatibility issues (~~*AR3/4*~~). Following ideas of VMwall [59], a gateway could use a hypervisor to protect the network processing inside the host VM ("dom0") against a compromised gateway OS. However, Table 3 (see Appendix) shows that commodity hypervisors like Xen [20] or QEMU/KVM face a high attack surface, which is even further increased by the dom0 OS—by default a full-blown Linux. Even when splitting core services into multiple VMs (similar to QubesOS [34, 52]), the TCB stays large (~~*SR7*~~). Minimal, so-called *micro-hypervisors* have a low TCB but are by design functionally limited, e.g., to a single VM without isolated I/O, which makes efficient secure I/O difficult (~~*AR4*~~) [8]. Furthermore, the use of security micro-hypervisors is in conflict with

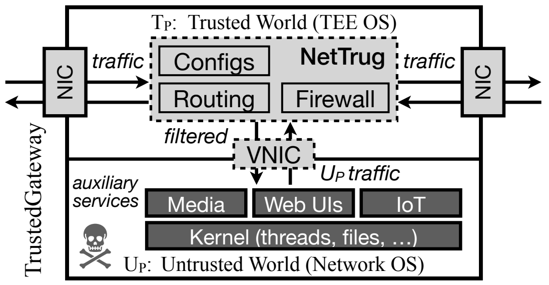

Figure 2: Design overview of TruGW with the new (dashed) trusted NetTrug and VNIC (dark: untrusted, light: trusted).

deployed commodity gateway hypervisors, and therefore either (a) requires slow, complex nested virtualization (~~*SR7,AR4*~~), (b) deep integration with gateway hypervisors (~~*AR2/3*~~), or (c) can only support gateways without hypervisors. McCormack et al. [45] have proposed such a micro-hypervisor-based secure gateway, however their concept fails to guarantee traffic protection and policy enforcement against system-level attackers (~~*SR3/4*~~). Zhou et al. [67, 68] used micro-hypervisors to build minimal TCB, trusted I/O paths from applications to specific device classes, but have not focused on NICs or network policies (~~*SR1-6*~~). TruGW's minimal TCB efficiently enforces and protects secure networking against Sys_{gw} even if they control a gateway hypervisor (cf. Section 2.1).

ARM TrustZone. TruGW builds on ARM TrustZone (TZ)^2, because TZ makes an ideal candidate for a secure network gateway due to its hardware-enforced memory and I/O isolation, and its widespread availability [49]. TZ provides hardware primitives for ARM-based TEEs, i.e., secure containers for hosting code and data isolated from all system software. Unlike Intel SGX, TZ is a system-level TEE and additionally features device isolation. TZ extends all system resources—including CPU, memory and devices—with a security state and supports HW-enforced access control rules based on the states [48, 49]. TZ's features enable stand-alone security architectures with trusted I/O similar to hypervisors, but with a potentially very small TCB (cf. OP-TEE in Table 3, Appendix) and without being in conflict with deployed gateway hypervisors. In fact, TrustZone has been used for many domains like trusted user I/O [35, 66], trusted peripheral access [31, 41], and secure stream processing [48]. However, none of these approaches explore the protection of a gateway's network path and policy enforcement (~~*SR1-6*~~). Even though StreamBox-TZ [48] proposes exclusive NIC access by trusted components for stream processing performance, it simply assumes trusted networking stacks and NIC isolation as an available black box. In fact, StreamBox-TZ neither provides details about networking, nor considers network policies, nor access by untrusted services (~~*SR1-6, AR2-4*~~). To the best of our knowledge, *there is no such* trusted networking support fulfilling all requirements for secure gateways. Therefore, TruGW designs new trusted networking components as part of its secure gateway architecture.

4 TRUGW'S DESIGN

We now describe TruGW's gateway design and mention challenges it had to solve. To highlight how TruGW fulfills the requirements outlined in Section 3.1, we refer to them at relevant passages. We provide additional details of TruGW's architecture in Section 5.

TruGW's main idea is to isolate network I/O and critical "core" gateway functionalities from a gateway's error-prone auxiliary services and system software. That way, TruGW's network "core" keeps full control over the network traffic and can guarantee secure policy enforcement even on a service or system compromise. As shown in Figure 2, TruGW uses a TrustZone-assisted TEE to divide the gateway architecture into an untrusted (U_P) and a memory-isolated trusted (T_P) partition. The untrusted U_P runs a gateway OS—in the following called *network operating system (NOS)*—which hosts the auxiliary services and commodity kernel. The trusted T_P hosts the TEE OS and all core components of TruGW, i.e., our TCB. TZ-assisted TEEs [40, 42] have a minimal TCB (*SR7*) with few CVEs (Table 3, OP-TEE), however, at the cost of a very limited secure runtime dedicated to small, U_P-exposed RPC services, e.g., trusted key storage [44]. Current TEE OSes are *not* designed for fast, I/O-intense tasks and thus *neither* support trusted network I/O *nor* traffic processing. Therefore, TruGW designs a new TZ-tailored networking core in T_P called *NetTrug*. NetTrug includes new modules for trusted network I/O, routing, and firewalling isolated from U_P attackers (cf. Figure 2). Policies are remotely configured via a new trusted interface (§4.3). To preserve compatibility with U_P services under an isolated network path, TruGW implements a new trusted virtual network device called *VNIC*, which together with NetTrug provides U_P with tightly-controlled network access (*AR2*).

We will now present how TruGW tackled the following major challenges: (i) achieve fast, trusted networking in spite of TZ's high context-switch overhead, (ii) securely share network access with U_P services *without* breaking security or compatibility, and (iii) provide trusted policy configuration—all while preserving a low TCB.

4.1 Trusted Networking

In a commodity gateway, the network I/O and processing is performed by drivers and services typically located in the NOS kernel. The NIC drivers form the I/O interface to the NICs while the services perform essential tasks, e.g., routing. However, their location makes them fully controllable by U_P system-level attackers enabling them to tamper with all traffic and bypass any security policy. To guarantee secure traffic and policy processing (*SR1-4*), NetTrug therefore revokes U_P's NIC access and provides trusted networking in T_P.

I/O and Scheduling. First, NetTrug must enable trusted NIC I/O paths. NetTrug therefore protects the NICs against U_P and supports trusted NIC drivers in T_P. NetTrug protects a NIC's I/O interfaces in T_P: memory-mapped device registers, shared I/O rings, and interrupts. Device registers enable drivers to interact with a NIC and especially configure the memory location of the I/O descriptor rings. These rings contain information about processable network buffers and by default reside in unprotected system memory together with their buffers. Descriptor changes are signaled via NIC interrupts and device registers [13]. If these interfaces stay unprotected, U_P attackers can tamper with network traffic inside the I/O buffers or directly interact with the NICs and thus bypass any policy (~~*SR1-4*~~). Therefore, NetTrug leverages TZ's Protection Controller (TZPC) [49] to bind all NICs exclusively to the trusted kernel space (T_P^k) from boot on (*SR1*). That way, TZ blocks all U_P access attempts to the NIC registers and securely redirects all NIC interrupts to T_P^k.

To protect the I/O rings and enable trusted I/O operation, NetTrug requires trusted NIC drivers inside T_P^k. However, current TZ-assisted TEEs [40, 42] have *no* support for network I/O. Naïvely, we could try to port existing drivers to T_P, but this raises several technical challenges: a full port would massively bloat the TCB (~~*SR7*~~), because drivers heavily depend on large, kernel-integrated driver frameworks and include many management functions beyond I/O. Furthermore, driver frameworks assume *fast* interrupt and threading support, which is either (i) not available in T_P due to TZ TEEs [40] relying on U_P for scheduling, which suffers from costly TEE context-switches and limitations in interrupt contexts (cf. §5.1) (~~*AR4*~~), or (ii) requires secure hardware timers [42] and respective U_P system-level changes for a TZ-tailored system scheduler—violating TruGW's goal of a *low TCB* design for *commodity* gateways (~~*SR7,AR1+3*~~).

Instead, NetTrug designs two new trusted kernel frameworks in T_P: a NIC I/O framework with partial driver integration, and a notifier and worker framework for efficient I/O scheduling. To keep the TCB small (*SR7*), NetTrug's I/O framework implements only the essential network and link layer abstractions required for I/O operations of NIC drivers, e.g., packet queues and NIC device interfaces. Furthermore, NetTrug splits each NIC driver in two parts: a trusted I/O part (T_{drv}) and an untrusted auxiliary part (U_{drv}). As shown in Figure 3, NetTrug integrates only the trusted part T_{drv} into T_P^k, but keeps U_{drv} in the untrusted network OS. T_{drv} protects the NIC I/O descriptor rings in T_P memory, handles NIC interrupts, and securely performs I/O isolated from U_P attackers (*SR4*). U_{drv} has no NIC access and only handles uncritical tasks on behalf of T_{drv} (split details in §5.2.1). To enable fast but compatible, low TCB I/O paths (*SR7,AR1+3-4*), NetTrug keeps the system scheduler and threading in the U_P NOS and instead designs new trusted NIC I/O workers. These workers build on lightweight, U_P-scheduled TEE OS threads, but are designed to minimize costly TZ context switches to U_P and be notifiable by trusted interrupt handlers. They are scheduled via a new trusted notifier on packet events, and run all NetTrug network tasks, incl. T_{drv} (details: §5.1). Combined, these frameworks ensure that NetTrug has exclusive control over the gateway's ingress and egress network paths and can efficiently perform secure NIC I/O even under a full U_P system compromise (*SR1+4,AR4*).

Routing and Firewall. For secure networking, NetTrug additionally requires trusted traffic routing and filtering—features entirely missing in current TEE OSes. However, we cannot directly port existing network stacks into TZ. Similar to driver frameworks, they consist of large modules which would bloat the TCB (~~*SR7*~~) and heavily rely on threading and synchronization primitives not efficiently available in compatible TEE OSes (~~*AR1+3-4*~~). In addition, these stacks are not designed to defend against U_P system-level attackers (~~*SR1-5*~~). Therefore, NetTrug designs new TZ-aware networking modules on top of its I/O and worker frameworks. In

[2] Our current focus is mainly on ARM TrustZone for Cortex-A (TZ-A).

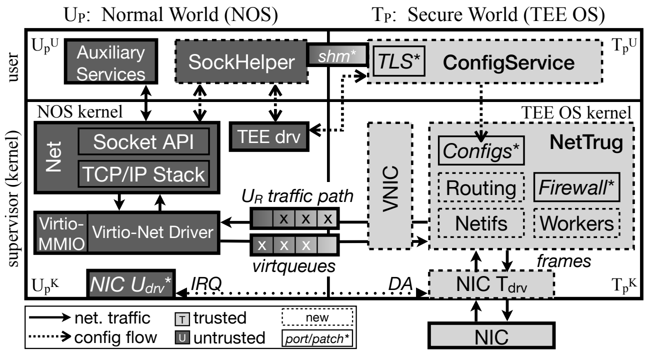

Figure 3: The TruGW architecture with untrusted (dark) and trusted (light) components. New components are marked with dashed lines; heavily ported or patched ones with stars.

contrast to existing stacks, NetTrug focuses on the security-critical "core services"—routing and firewall—and explicitly excludes client protocol and socket stacks (e.g., TCP/IP) from T_P to minimize the TCB [69] (*SR7*), as shown in Figure 3. NetTrug's exclusive NIC control guarantees secure traffic processing by its networking modules (*SR3-4*). NetTrug introduces trusted and untrusted network interfaces on which its workers enforce trusted routing and firewall policies. NetTrug maps all physical NICs to trusted interfaces by default, and can enforce extra routing and filter rules on untrusted interfaces. In §4.2, we will explain how NetTrug and its virtual VNIC device securely enable tightly-controlled network access to U_P services via an untrusted network interface. For traffic filtering, NetTrug incorporates a network firewall into its trusted networking frameworks. NetTrug assumes at least a stateful L3/L4 firewall for secure, efficient traffic protection, but is conceptually oblivious to the concrete firewall capabilities (cf. §5.1). In contrast to commodity gateways, NetTrug securely manages trusted routing and firewall policies in T_P and guarantees mandatory policy enforcement. U_P attackers can neither bypass nor tamper with these policies (*SR1-3*).

4.2 Securely Sharing Network Access

As NetTrug isolates all NICs and processes their traffic securely in T_P^k, any direct network access by U_P attackers is blocked (*SR4*). However, as NetTrug focuses on trusted network I/O and security-critical services (*SR7*), all remaining gateway services stay in U_P and become unreachable. To resolve this compatibility issue, TruGW designs VNIC, a new virtual network device that performs secure traffic forwarding between U_P and T_P. In contrast to commodity virtual network devices [47], VNIC is tailored to TZ and integrated into NetTrug's networking frameworks. Together with VNIC, NetTrug enables tightly-controlled network access for U_P services (*AR2*).

From U_P's point of view, VNIC exposes a memory mapped Virtio-net device, i.e., a virtual ethernet card with a low TCB memory interface (Virtio-mmio) following the Virtio standard [47] (*SR7*). That way, the U_P NOS can use its builtin Virtio drivers (*AR3*) to initialize a network interface to VNIC, which serves as U_P's default interface for all network I/O (cf. §5.2.2). The interface is configured with all IP addresses of the gateway. As a result, U_P services need not be modified and can use the standard socket API and TCP/IP stack of the NOS for their network communication (*AR2*).

From T_P's point of view, VNIC is a special trusted NIC driver associated with an untrusted NetTrug network interface. VNIC performs the buffer I/O between the untrusted U_P Virtio-net driver and NetTrug. VNIC's virtual I/O rings are located in untrusted memory shared with the U_P driver (virtqueues, Figure 3). Therefore, VNIC must securely copy network buffers between the rings and T_P and check that untrusted buffers never overlap with trusted T_P memory. That way, VNIC prevents traffic tampering and memory attacks by U_P system-level attackers and enables NetTrug to securely process the network buffers in protected T_P memory (*SR3, SR4*).

VNIC provides an explicit, secure path to U_P services and thus enables NetTrug to make them reachable again. However, NetTrug must enforce additional security measures on the VNIC-associated untrusted network interface to protect forward traffic against U_P (*SR4*) and prevent address spoofing attacks by U_P-located attackers (*SR5*). By default, NetTrug routes traffic only to U_P if it is explicitly destined to one of TruGW's IPs. That way, the forward traffic stays isolated from U_P (*SR4*) and avoids additional I/O overhead (*AR4*). To prevent spoofing by U_P (*SR5*), NetTrug replaces the source MACs of all egress traffic with those of the output interfaces, drops packets from U_P with spoofed source IPs, and handles U_P's host discovery messages locally in T_P (§5.3.1). In addition, TruGW enables trusted admins to define firewall policies directly on the VNIC interface to tightly control network access from and to U_P, as discussed next.

4.3 Trusted Policy Configuration

Administrators are used to manage network policies using a NOS-provided U_P web application. However, any configuration service inside U_P gives system-level attackers full control over a gateway's policies (~~*SR3+6*~~). Naïvely, we could isolate a configuration service in T_P and make it remotely reachable directly via NetTrug. Yet this would require a full network and web stack in NetTrug (incl. a full-fledged web server, TCP/IP stack, and socket API), leading to a stark increase in its TCB size and attack surface (~~*SR7*~~).

Instead, TruGW offers a web-based configuration that does not require these complex software stacks. To this end, we introduce *ConfigService*, a new tiny T_P userspace service for secure remote configuration of NetTrug's trusted network policies (cf. Figure 3). ConfigService provides authenticated admins with a trusted web application for policy management (*SR6*) while offloading web resources and the connection handling securely to U_P for a low TCB (*SR7*). ConfigService includes a new minimal (~2.1k LOCs, plus TLS library) HTTPS endpoint to handle TLS sessions with admins and ship a web interface to their browsers. To minimize the TCB (*SR7*), ConfigService securely offloads the TCP socket management to U_P; a new untrusted userspace service (SockHelper) handles the TCP sockets for ConfigService and forwards the protected TLS records between the U_P network stack and ConfigService (Figure 3). That way, ConfigService requires no TCP/IP or socket stack inside T_P.

SockHelper makes ConfigService remotely reachable via VNIC. However, U_P attackers become strong on-path MITM attackers as they control the shared U_P TCP/IP stack. While TLS provides end-to-end protection between admins and ConfigService, ConfigService must additionally prevent impersonation and web attacks by U_P

(*SR6*). Therefore, TruGW introduces a dedicated trusted web address (domain or IP) for ConfigService and supports a secure enrollment process for establishing credentials for mutual TLS authentication. The trusted address guarantees a different web origin than U_P services even though the TCP/IP stack is shared and thus enables ConfigService to prevent web attacks by U_P (cf. Section 5.4). During the enrollment process, TruGW generates a TLS server certificate with ConfigService's trusted address and registers a TLS client certificate for a master admin. Admins then submit their own TLS client certificates to ConfigService and get them approved by the master admin. By leveraging TLS client certificates, TruGW avoids password-related security issues [21], reduces the risk of phishing attacks, and can benefit from TPM-based storage back-ends [43].

Policy Translation. TruGW avoids inventing new policy languages to ease adoption. ConfigService uses a standard routing syntax (similar to ip-route [30]) and the vanilla firewall syntax (cf. §5.1) for configuration. In addition, ConfigService enables the reconfiguration of TruGW's IPs (cf. §5.3.2). Admins can reuse existing policies and further restrict services running on the gateway by defining new routing and firewall rules for the VNIC interface. The VNIC interface enables admins to explicitly control traffic from and to untrusted U_P services and ConfigService. Comparing to a commodity firewall configuration tool like iptables, firewall rules on NetTrug's physical NIC interfaces roughly translate to iptable's pre-/postrouting and forward chains, while rules on the VNIC interface roughly translate to iptable's input and output chains.

5 TRUGW DETAILS AND IMPLEMENTATION

We will now present the details of our TruGW architecture. We picked Linux as the U_P OS, given that many commodity gateway NOSes are derivatives of Linux (cf. Table 2, Appendix). For the TEE, we chose OP-TEE [40] as it is a well-known, open-source TEE for TZ with upstream Linux support, and a low TCB (*SR7*, cf. Table 3). Our implementation targets an i.MX6 SoC [58], which features a TZ-compatible Central Security Unit (CSU) for device isolation and a TZ Address Space Controller (TZASC) [49] for the memory partitioning. Without sacrificing generality and for ease of discussion, we assume an Ethernet-based router that operates in an IPv4 network.

5.1 TEE Integration and Networking

TruGW's security is rooted in the integrity of its T_P components and boot process. Therefore, TruGW leverages secure boot to guarantee that only trusted bootloader and TEE images are loaded (cf. Appendix A). TruGW's trusted kernel (T_P^k) components (cf. Figure 3) extend OP-TEE's kernel and are therefore verified as part of the TEE images (SR2). The trusted bootloader includes a device tree (DT) blob [39] which describes all hardware components of the system. On TEE boot, NetTrug parses the DT to bind all NICs to T_P by configuring them as secure in i.MX6's CSU[3] [33] and thus protect them against U_P (cf. §4.1). To prevent early boot attacks by U_P, TruGW transfers control to the U_P bootloader only after all protections have been successfully set up (*SR1*).

[3]a TZPC or an other SoC-specific technology can replace the CSU

Trusted Networking. NetTrug is TruGW's central extension to the trusted TEE kernel. NetTrug mediates all gateway traffic and securely performs trusted network I/O and policy enforcement in T_P^k (cf. §4.1). On TEE boot, NetTrug initializes one trusted network interface for each NIC and one untrusted interface for VNIC and allocates an egress queue, ARP cache (cf. §5.3.1), I/O workers, and a configurable, static IP address (cf. §5.3.2) to each of them. NetTrug tags untrusted interfaces, s.t. its routing and firewall modules can enforce special restrictions on them, e.g., to isolate trusted forward (and broadcast) traffic and prevent spoofing attacks by U_P (*SR4-5*; cf. §4.2, 5.3.1). For packet filtering, NetTrug incorporates the stateful, BPF-based L3/L4 firewall NPF [54]. To this end, we ported NPF to OP-TEE and NetTrug's worker framework, and integrated it as a callable firewall module into NetTrug's networking loop, where NPF enforces trusted filter rules on given IP packets. We picked NPF as it is well-known (NetBSD's firewall) and feature-rich. However, conceptually, NetTrug could adopt additional firewall modules (e.g., application level) as trusted kernel or user modules.

NetTrug's new I/O workers perform the actual traffic processing for each interface securely in T_P^k using a polling-based I/O model. On setup, NIC drivers (incl. VNIC) request I/O workers for their interfaces and allocate device-specific I/O callbacks to them. On a packet event (e.g., signaled by an interrupt handler), workers poll and process all current RX (or TX) packets of their assigned NIC, before reentering a sleep state. They perform a typical I/O loop: (i) Ethernet RX via driver, (ii) link layer processing, (iii) ingress filtering and IP routing, (iv) egress filtering and ARP resolution, (v) egress enqueueing, and (vi) packet transmission via driver.

NetTrug's Workers. For TruGW to be practical, it is crucial that TruGW's trusted networking causes only a small performance penalty compared to commodity gateways (*AR4*). While TruGW and OP-TEE both follow the idea of keeping full scheduling and threading stacks in U_P to preserve compatibility and a low TCB (cf. §4.1), OP-TEE's approach is not suitable for efficient NIC I/O. OP-TEE relies on U_P threads to call into the TEE for service and assigns them lightweight TEE tasks (a.k.a. threads) on entry. This design causes high overhead on thread switches and synchronization—both omnipresent in networking cores—due to costly context switches between T_P and U_P. In addition, it is *not* possible to schedule TEE tasks from trusted interrupt handlers as required for NIC I/O, because the U_P APIs are context-switching and thus not callable from interrupt contexts [29] (details on OP-TEE's design are given in Appendix B).

NetTrug's trusted workers build on lightweight (OP-)TEE threads, but overcome their limitations. NetTrug exposes a new, minimal worker registration interface to U_P, which a helper service uses to provide a pool of U_P threads. One thread registers as NetTrug's notifier and the others as workers. NetTrug's networking modules (e.g., drivers) can request scheduling of a worker using a new dedicated T_P^k API (similar to NAPI [13]). The API directly flags a worker without any context switch and is thus also callable from trusted NIC interrupt handlers, e.g., on a packet event. NetTrug's notifier periodically checks for flagged workers and if sleeping, wakes up their associated threads using U_P's scheduler. As the worker's sleep and wake-up operations fall back to costly context switches to U_P, NetTrug minimizes their number using several optimizations, e.g., I/O batch processing, notification coalescing on multiple packets or full queues, and a grace period of idling before putting worker threads to sleep. That way, NetTrug keeps the performance penalty low (cf. §7.3) while preserving a U_P-compatible, low TCB design.

5.2 Trusted Network Device I/O

5.2.1 Split NIC Driver Operation. NetTrug's network I/O and worker frameworks provide the essential support required for secure and efficient NIC driver I/O in T_P. As full NIC drivers would bloat the TCB (*SR7*), we split them and port only the critical, I/O relevant driver parts to OP-TEE and NetTrug while keeping the uncritical rest in U_P (cf. §4.1). On T_P boot, the secure subdriver T_{drv} registers a trusted network interface and I/O workers on NetTrug for the NIC and securely allocates the NIC I/O descriptor rings in T_P. Combined with the NIC's T_P-binding established by NetTrug (cf. §5.1), the NIC is in a clean and protected state before the untrusted NOS starts booting (*SR1*). On U_P boot, the untrusted subdriver U_{drv} is responsible for performing uncritical configuration tasks (e.g., power management) [68] and starting the physical Ethernet device of the NIC (PHY).[4] However, the NIC protection blocks any access attempts by U_{drv} to a NIC, s.t. they result in a data abort (DA). Therefore, T_{drv} registers a secure DA handler. That way, if an uncritical U_{drv} task requires a one-time NIC access (e.g., PHY startup), T_{drv} can trap the access fault in T_P, decode it [33], and securely perform the access on behalf of U_{drv}. After boot, the trusted NIC workers securely perform the NIC I/O and the packet forwarding between the NICs and NetTrug. T_{drv} securely handles the NIC's I/O interrupts in T_P and forwards uncritical ones to U_{drv} if required. *U_{drv} is not involved in the I/O phase*, which enables a secure, low overhead operation (*SR2-4, AR4*).

5.2.2 VNIC Device I/O. We designed VNIC's U_P-interface based on Virtio-net and Virtio-mmio [47] to make it compatible with commodity NOSes and drivers (*AR3*) while having a small TCB (*SR7*). On T_P boot, VNIC registers an untrusted network interface on NetTrug and extends the device tree [39] (cf. §5.1) to expose itself as a simple (*SR7*), memory-mapped device to U_P (Virtio-mmio). On U_P boot, Linux detects the VNIC device and uses its Virtio default drivers to set up a network interface for U_P. To enable U_P interaction, VNIC exposes virtual device registers to U_P using a dedicated memory region. VNIC protects the region from U_P via the TZASC (cf. §5), s.t. access attempts by U_P trap as data abort exceptions into T_P. On a trap, VNIC decodes the respective physical target address [33] and maps it to its virtual device registers. That way, VNIC can transparently detect and handle configuration requests and I/O ring notifications by U_P. On network I/O, VNIC's NetTrug worker receives Ethernet frames from U_P or NetTrug, securely processes and routes them, and forwards traffic between T_P and U_P (cf. §4.2 and §5.1).

5.3 Address Resolution and Assignment

5.3.1 ARP. TruGW must guarantee secure MAC address resolution to prevent redirection and spoofing attacks by attackers in U_P (*SR5*). Therefore, NetTrug includes a trusted ARP stack inside T_P and performs extra checks on U_P traffic. For the physical NIC interfaces, the ARP stack handles MAC address resolution and ARP requests securely in T_P^k. For the untrusted VNIC interface, NetTrug performs special steps to prevent ARP spoofing attacks by U_P: (a) U_P's ARP requests are directly answered by NetTrug with a virtual MAC and (b) U_P's ARP replies are dropped. That way, NetTrug transparently handles U_P's ARP resolution and prevents U_P from poisoning the ARP caches of any NIC interface or of any internal or external host (*SR5*). When forwarding traffic to U_P, NetTrug knows VNIC's U_P-exposed MAC and can directly use it as the destination MAC.

5.3.2 DHCP and DNS. By default, TruGW does not assign IP addresses or handle DNS queries to keep its TCB small (*SR7*). TruGW has a set of static, preconfigured (yet configurable) IP addresses (cf. §5.1). We assume that network admins reconfigure these to fit their setup and operate a dedicated DHCP server to assign addresses to clients. Conceptually, NetTrug could incorporate a *basic* DHCP stack for smaller networks, e.g., providing gateway, client, and DNS server IPs. However, a full DHCP server would require a UDP/IP and socket stack inside T_P, which significantly increases TruGW's TCB (cf. §4.1). Regarding DNS, the current design of TruGW assumes DNS to be outside of the gateway, such as a dedicated DNS resolver or an external DNS resolver (e.g., provided by ISPs or other entities such as Google). Either way, NetTrug protects the confidentiality and integrity of DNS and DHCP communication against U_P system-level attackers using its restrictive routing and anti-spoofing measures on the VNIC interface (*SR4-5*; cf. §4.2, §5.3.1).

5.4 Trusted Policy Management

TruGW must prevent unauthenticated network communication by U_P and network attackers until a trusted policy has been provided. On startup, NetTrug therefore sets up a "restrictive boot policy". This policy only allows local HTTPS connections *to* TruGW's configuration ports, but neither outgoing U_P connections nor traffic forwarding across network clients. That way, NetTrug restricts network traffic to local configuration sessions until a policy gets configured via ConfigService or securely restored from disk (*SR1*).

ConfigService is implemented as an OP-TEE trusted user application. Its binary is signed, integrity checked by OP-TEE on load, and protected against version rollbacks [40]. On an admin connection, ConfigService ships only an initial tiny, integrity-checked root HTML file. All other web resources are loaded from an untrusted U_P Apache server. That way, ConfigService can keep its latency and memory footprint low (cf. §7.3+7.4) and does not depend on external resources which are blocked on startup (*SR1+7,AR4*). ConfigService uses subresource integrity (SRI) [1] to guarantee the integrity of the U_P-offloaded resources (*SR6*). Furthermore, it verifies custom HTTP request headers to protect against cross-site request forgery (CSRF) [3]; attacker-induced requests from different origins, e.g., by rogue untrusted services (cf. §4.3), cannot add such custom headers. To support NPF's policy language, we ported NPF's client tool to WebAssembly [12]. It parses the NPF policies inside the trusted admin browsers and sends BPF filters via ConfigService to NetTrug, where they are securely parsed, compiled, and enforced.

ConfigService's server and client authentication is based on TLS server and client certificates, respectively. On initial boot, NetTrug

[4] a potential splitting of the PHY drivers is left as future work

securely issues a self-signed TLS server certificate C_{cnf} for ConfigService's trusted web address (cf. §4.3) and stores it on rollback-protected storage. For initial enrollment, the master admin then connects via an exclusive physical network access to ConfigService and uploads a securely generated TLS client certificate C_{mst}. In addition, the master admin distributes C_{cnf} to all admins for certificate pinning (cf. §5.5) to prevent phishing and CSRF attacks against ConfigService's trusted address, especially by U_P attackers. The master can trust the initial C_{cnf} on first use (TOFU) as the secure boot (cf. §5.1), factory state of U_P, and exclusive network access rule out any device or network attacker. On completion, ConfigService securely stores C_{mst} and starts enforcing access control based on the TLS client certificates of the HTTPS client connections. Clients without a registered TLS client certificate can only upload a TLS client certificate C_{adm} to request admin access, which then has to be explicitly granted by the master. Only admins and the master have access to the trusted routing and firewall policies. The master can additionally revoke admin certificates or request server key rollovers, e.g., on a key breach. An explicit trusted factory reset (e.g., via button) can wipe *all* certificates for a full re-enrollment.

5.5 Deployment

TruGW has been designed with the goal to be compatible with commodity ARM gateway routers (*AR1*). TruGW currently requires ARM TrustZone with memory and device isolation (TZASC, TZPC) and support for rollback-protected storage (e.g., eMMC with RPMB). T_P's secure memory demands are about 16–32 MB and therefore easily met by many router platforms (cf. Section 7.4). Regarding software, TruGW is compatible with commodity Linux and its upstream OP-TEE and Virtio drivers (*AR3*). The untrusted NIC drivers (U_{drv}) are slightly adapted versions of the Linux drivers. Manufacturers can easily deploy TruGW, because its T_P^k components are direct extensions of the OP-TEE image(s) and its ConfigService TA and U_P services can be packed into OP-TEE's Linux software package. TruGW is non-intrusive in that its TEE extension does not affect other applications (*AR2*) and its U_P helper services (e.g., SockHelper) do not require any special permissions.

Manufacturers can update TruGW using standard methods. The untrusted and trusted userspace components (incl. ConfigService) can be updated via regular Linux package updates. Attackers cannot manipulate the trusted components as OP-TEE only accepts vendor-signed TAs (cf. §5.4). TruGW's trusted kernel components (e.g., NetTrug) require an update of the TEE image using existing (or device-specific) methods for firmware updates [38]. TruGW does not affect the way commodity U_P software is updated.

Admins can follow common best practices for managing TruGW's TLS server and client certificates. The master admin distributes ConfigService's server certificate C_{cnf} to all admins for certificate pinning (e.g., via group policies). U_P manages TLS server certificates of untrusted U_P services. Admins must vet these U_P certificates to *not include* the trusted web address of ConfigService before distributing them to guarantee distinct web origins (cf. §4.3). To ease the U_P vetting, TruGW could integrate a T_P certificate authority restricted to untrusted addresses (cf. `RFC5280`), whose certificate could then be distributed instead. Key breaches and rollovers are securely handled by master or via a full re-enrollment (cf. §5.4).

6 SECURITY ANALYSIS

We now analyze TruGW's security design by discussing its countermeasures against critical attacks and assessing how it contains real world vulnerabilities of commodity gateways.

6.1 Attacks and their Countermeasures

We now summarize attacks against TruGW. Many of them are directly related to the requirements defined in Section 3.1.

Adversary Types. Following our defined threat model (cf. §2.1), TruGW's main focus is on system-level attackers (Sys_{gw}) which gained full control over U_P via a remote service and system exploit. Furthermore, we assume malicious network clients located in internal (*int.*) or external (*ext.*) networks with the goal of bypassing access restrictions. Beyond our threat model, we assume that adversaries might control a web page visited by an admin (*web*). Finally, while we regard admins and their systems as trusted, we also discuss the implications of a system-level attacker on the systems of the admins (Sys_{adm}) or master (Sys_{mst}). Based on these attacker roles, we now discuss how TruGW protects against 14 security-critical attacks shown in Table 1 (see next page).

*A01: Image/Binary Tampering (*SR2*).* The integrity of TruGW's T_P images (e.g., NetTrug) and device tree are guaranteed by secure boot (cf. §5.1). Tampering with ConfigService's binary is prevented as OP-TEE verifies TA binaries on load and prevents rollbacks.

*A02: Code/Data Tampering (*SR2*).* Sys_{gw} cannot tamper with TruGW's T_P components using memory writes or direct memory access. From boot on, TruGW protects T_P memory and NICs from Sys_{gw} using TrustZone and securely allocates all data in T_P (cf. §5.1).

*A03: Policy Enforcement Bypass (*SR1/3*).* Sys_{gw} cannot bypass NetTrug's trusted policies, because NetTrug has full control over the NIC I/O paths from TEE boot (cf. A02) and can therefore guarantee their enforcement. *int.* and *ext.* attackers cannot bypass TruGW's policies due to TruGW's deployment at the perimeter.

*A04: Direct MAC/IP Spoofing (*SR5*).* TruGW prevents Sys_{gw} from sending traffic with spoofed source MAC or IP address by replacing the source MAC with the MAC of the resp. output NIC and by dropping U_P packets with spoofed source IP on VNIC (cf. §4.2). To defend against *int.* adversaries, TruGW can securely enforce port-based MAC pinning schemes and subnetwork isolation in T_P.

*A05: ARP Poisoning/Spoofing (*SR5*).* TruGW performs ARP request and response handling securely in NetTrug. To prevent ARP poisoning and spoofing by Sys_{gw}, NetTrug isolates U_P ARP messages by directly replying to U_P ARP requests and not forwarding U_P ARP replies (cf. §5.3). For *int.* attackers, NetTrug can securely enforce static routes or other common schemes (cf. A04).

*A06: Traffic Tampering/Sniffing (*SR4*).* Sys_{gw} can neither read nor manipulate any forward traffic or any network packet stored in TruGW's trusted I/O buffers. NetTrug and its secure NIC drivers (T_{drv}) protect the NIC I/O paths (incl. I/O rings) in T_P (cf. §4.1). In addition, NetTrug routes only U_P-destined traffic to U_P (cf. §4.2).

*A07: Policy Tampering (*SR6*).* Sys_{gw} cannot directly tamper with trusted policies in memory or on disk. NetTrug isolates the policies

Table 1: Overview of TruGW's defense measures against security-critical attacks by the adversaries defined in Section 6.1.

Target / Goal	Attack	Adversaries	TruGW's Defense Mechanisms	Sec?
Comp. Integrity	A01: Image/Binary Tampering	Sys_{gw}	secure boot + signed TAs	✓
	A02: Code/Data Tampering	Sys_{gw}	TZ mem/NIC protect. via NetTrug (+T_{drv})	✓
Policy Enforcement	A03: Policy Enforcement Bypass	Sys_{gw}, *int.*, *ext.*	NetTrug's NIC I/O and policies + perimeter	✓
Address Spoofing	A04: Direct MAC/IP Spoofing	Sys_{gw}, *int.*	NetTrug's filtering (+ port pinning, subnets)	✓
	A05: ARP Poisoning/Spoofing	Sys_{gw}, *int.*	NetTrug's trusted ARP handling (+ cf. A04)	✓
Traffic Protection	A06: Traffic Tampering/Sniffing	Sys_{gw}	T_{drv}+NetTrug's NIC I/O + restrictive routing	✓
Trusted Policy Configuration	A07: Policy Tampering	Sys_{gw}	TZASC + NetTrug + secure storage	✓
	A08: Policy Change via Auth. Bypass	Sys_{gw}, *int.*(, *ext.*)	ConfigService's enrollment + cert. management	✓
	A09: Config Connection Tampering	Sys_{gw}, *int.*	ConfigService's protected TLS endpoint	✓
	A10: ConfigService Spoofing	Sys_{gw}, *int.*, *web*	C_{cnf} pinning + trusted domain/IP vetting	✓
	A11: CSRF against ConfigService	*web* (Sys_{gw})	custom request header + trusted domain/IP	✓
	A12: ConfigService File Tampering	Sys_{gw}	SRI + hashing (+ signed TAs)	✓
Admins / Master	A13: Admin/Master Compromise	Sys_{adm}, Sys_{mst}	TPM + TEE browser + secure user I/O	(✓)
Leakage	A14: Covert Channel (Hdrs,Time)	*int.*, *ext.* (Sys_{gw})	filters + traffic tunnels + time masking	(✓)

in T_P memory and allows changes only by ConfigService. Disk backups are protected via OP-TEE's secure storage API.

*A08: Policy Change via Auth. Bypass (*SR6*).* Sys_{gw} and *int.* cannot modify trusted policies (or IPs) via ConfigService, because only master and admins have access. In addition, the initial master enrollment is secure, because the gateway (incl. U_P) is in a secure boot state and the master has exclusive device access (cf. §5.4). Afterwards, master grants only trusted admins access to ConfigService and blocks any malicious requests by Sys_{gw} or *int.* TruGW restricts access to ConfigService to internal clients, which blocks *ext.*

*A09 Tampering with Config Session (*SR6*).* Sys_{gw} and *int.* cannot tamper with connections between trusted admins and ConfigService, because they are TLS-protected and end in T_P.

*A10: ConfigService Spoofing (*SR6*).* Neither Sys_{gw}, nor *int.*, nor *web* can impersonate ConfigService, because admins securely pin its server certificate (C_{cnf}) for the trusted web address (cf. §5.4). Furthermore, admins distribute U_P service certificates only for untrusted addresses (cf. §5.5).

*A11: CSRF against ConfigService (*SR6*).* TruGW prevents *web* attackers from launching CSRF attacks against admins of ConfigService by requiring custom HTTP request headers [3] (cf. §5.4) which are only settable from the same web origin. As ConfigService has a trusted web domain (or IP) and thus different origin than U_P services (cf. §4.3), Sys_{gw} cannot launch CSRF either.

*A12: ConfigService Resource Tampering (*SR6*).* Sys_{gw} cannot tamper with ConfigService's root HTML or U_P-hosted web resources, because ConfigService uses secure hashing and subresource integrity (SRI) to check their integrity on load (cf. §5.4).

*A13: Admin/Master Compromise (*SR6*).* While we assume the master, admins, and their systems as trusted (cf. §2.1), we now discuss the implications of a full compromise of their systems. Sys_{adm} cannot steal the admin private key K_{adm}^{-1} if it has been securely generated and stored in a TPM. However, Sys_{adm} can use the admin credentials to maliciously reconfigure TruGW's trusted policies via ConfigService. If such a breach is detected, the master must immediately revoke C_{adm}. To prevent such an attack, TruGW can deploy orthogonal solutions on the admin-side, which establish a secure I/O channel between the admin and a TEE-protected browser [19, 24] and enforce their use for ConfigService access [57]. That way, Sys_{adm} can neither steal K_{adm}^{-1} nor use it. The situation is similar for Sys_{mst}, however, on a master key breach K_{mst}^{-1}, a full enrollment reset is required (cf. §5.4). On re-enrollment, the master requires a clean system to prevent hijacking attempts by Sys_{mst}.

A14: Covert Channels (Headers, Timing). TruGW's current focus is *not* on an active prevention of covert channels. However, TruGW could adopt existing techniques to contain or prevent covert channels. For instance, TruGW could heavily filter all packet headers (incl. U_P's) to remove storage channels [65], deploy client-side solutions for protected traffic tunnels to TruGW to entirely strip untrusted headers by *int.* [57], or adopt time masking schemes to prevent timing channels by *int.* and *ext.* [6]. As TruGW currently relies on U_P for scheduling (cf. Section 5.1), Sys_{gw} controls the scheduler and can exploit it for additional timing channels. TruGW could prevent them by switching to a T_P-controlled scheduler [42].

6.2 Real World Vulnerabilities

Recent CVEs in network gateways have raised serious security concerns and motivated our design of TruGW. We now asses how TruGW addresses these real world vulnerabilities. As discussed in Section 2, critical CVEs of network gateways mainly lurk in auxiliary services, e.g., SNMP or web interfaces, and system components unrelated to core network functionalities (Table 3 + 6, Appendix). They enable remote attackers full control over the system, i.e., attackers effectively gain the privileges of Sys_{gw}, e.g., via a remote code execution. By design, TruGW contains exactly these types of services and system components in U_P and securely isolates the core network functionalities (incl. firewall) in T_P against Sys_{gw}. Therefore, TruGW successfully protects gateways against recent attacks.

TruGW's security can only be undermined if vulnerabilities lurk in the remaining attack surface within T_P services themselves. However, TruGW only includes core network services in T_P (e.g., firewall, NIC drivers), which have faced very few CVEs, especially compared to commodity OSes (cf. Section 2). Furthermore, our current TruGW T_P^k prototype (cf. §7.1) only has ≈110 kLOCs, whereas 12 popular auxiliary services of DD-WRT routers already include an attack surface which is one order of magnitude larger (≈4517 kLOCs, cf. Table 4 in Appendix). Commodity U_P OSes are even larger and have faced 2-3 orders of magnitudes more CVEs than OP-TEE OS, which faced only ≈10 CVEs (cf. Table 3, Appendix). Therefore, TruGW drastically decreases the TCB size of commodity routers and thus risk of critical vulnerabilities.

7 EVALUATION

We now describe our prototype implementation and evaluate it in terms of code size (TCB), performance, and memory overhead.

7.1 Open-source Prototype

We implemented an open-source TruGW prototype[5] on a Nitrogen6X development board [16] with an i.MX6Q ARM CPU (32bit, 4 cores), 2 GB RAM, and the Gbps Freescale Fast Ethernet Controller (FEC) as our secure NIC. FEC is known to be technically limited to ≈470 Mbps maximum (cf. errata 004512), and indeed showed only ≈400 Mbps for ingoing/outgoing traffic in a vanilla setting in our experiments. However, we nevertheless chose this board due to its Ethernet support and as its TrustZone support was well documented and successfully used in research projects by others [33]. We run Debian 10 with a 4.14 Linux kernel[6] as untrusted U_P OS and OP-TEE 3.8.0 [40] as the secure T_P OS. We use U-Boot 2018.07 as the (trusted) bootloader[6].

To implement NetTrug's ARP, routing, and NPF integration, we ported NPF-Router [53] to OP-TEE and significantly extended it with trusted workers (incl. notifier), device driver callbacks, packet buffers and queues, and VNIC support. NetTrug's worker registration interface (cf. Section 5.1) is exposed to U_P via OP-TEE's TA client API. We have implemented VNIC mostly from scratch, but use the vqueue implementation of Trusty OS [42] for the I/O rings. T_{drv} follows the Linux FEC driver and registers a separate Rx and Tx worker on NetTrug for increased performance. We integrated T_{drv} into NetTrug's driver framework and enabled interrupt sharing with U_{drv} (cf. §5.2.1). For trapping and decoding U_P NIC and VNIC access faults, we have extended and integrated parts of SeCloak [33] into OP-TEE and NetTrug.

ConfigService is implemented as an OP-TEE trusted application (TA). We use the tiny picohttpparser [46] for HTTP parsing and a small subset of mbedTLS for TLS. The untrusted SockHelper is a small C program which handles the TCP server sockets and calls into ConfigService via OP-TEE's TA API. SockHelper exchanges TLS records with ConfigService using a new ringbuffer based on OP-TEE's shared memory API. ConfigService's web application is a simple web page which communicates via GET and POST XMLHttpRequests with ConfigService and uses our Wasm port of NPF's client tool for the firewall policy parsing (cf. Section 5.4).

[5]Prototype available at: https://github.com/trugw

[6]we use the imx6 forks provided by the board vendor (Boundary Devices)

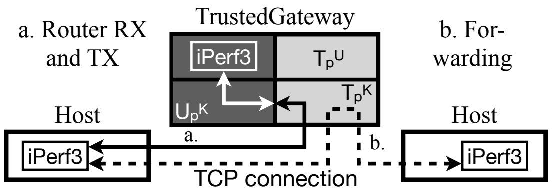

Figure 4: Throughput evaluation setup. a. Throughput between an untrusted gateway router service and an internal host (both directions). b. Forwarding throughput.

7.2 Code Size Analysis

We now analyse to what extent TruGW's components increase the TCB size compared to vanilla OP-TEE. We measured the LOCs of OP-TEE's and TruGW's trusted kernel components using cloc [15]. As OP-TEE and NPF choose files depending on the platform configuration, we only counted the actually included source/header files. OP-TEE's core has ≈52 k LOCs plus ≈24, 4 k for its crypto library LibTomCrypt [36]. TruGW adds ≈8.4 k LOCs for NetTrug, VNIC, T_{drv}, and device access trapping combined plus ≈25.1 k for the NPF firewall with all its libraries. That means, TruGW's core increases OP-TEE's core only by ≈16% and the addition of NPF is roughly on a par with OP-TEE's crypto library (*SR7*). Moreover, NPF makes up ≈75% of the current TruGW code base, and there is a substantial shrinking potential as NPF's design is not tailored to TrustZone. ConfigService adds only ≈2.1 k LOCs to T_P^u, plus a subset of mbedTLS. Altogether, TruGW has a reasonably small impact on OP-TEE's TCB size and significantly decreases the overall attack surface compared to commodity U_P OSes and services (cf. §6.2).

7.3 Performance Evaluation

We now report on the network performance of TruGW. We evaluate (i) the network throughput of TruGW, (ii) the overhead of its firewall, (iii) TruGW's impact on network latency, and (iv) the page load time of ConfigService. To this end, we interconnected TruGW with two client hosts using a 5-port gigabit switch. The first host ($Host_M$) is a Macbook Pro (Mac) with an Apple Thunderbolt-to-Gigabit Ethernet Adapter. The second host ($Host_L$) is an HP Z1 workstation with an Intel I219-LM NIC running Ubuntu. Each host is in a separate IP subnetwork and configured to use TruGW as its default gateway, s.t. all traffic is forwarded through TruGW.

7.3.1 Network Throughput. We use iPerf3 [22] to evaluate the TCP network throughput of TruGW in three ways: (i) the downlink of an untrusted router service ("Router RX", e.g., file upload to a local file server on the gateway) and (ii) uplink throughput of an untrusted router service ("Router TX", e.g., file download from the gateway's file server), and (iii) the client throughput when routing all traffic through TruGW ("Forwarding"). iPerf3 sends TCP traffic via a single connection to another iPerf3 instance and measures the resulting throughput performance over 10 s. Figure 4 illustrates our test cases and corresponding network flows. For (i) and (ii), iPerf3 runs on one client host and as an untrusted router service in TruGW's U_P. TruGW serves as the (i) receiver and (ii) sender respectively. For (iii), we run iPerf3 on both client hosts and consider both sending directions. We compare TruGW to the plain Linux

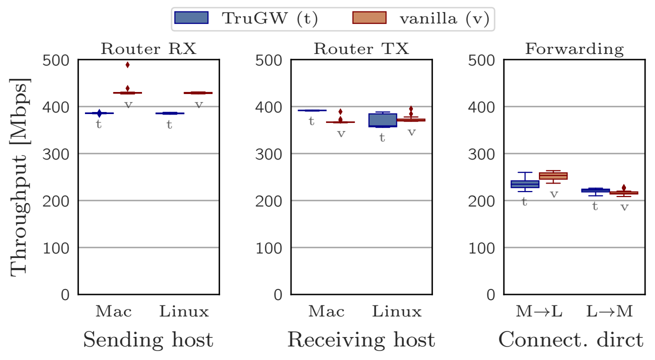

Figure 5: iPerf3 TCP throughput when the TruGW gateway router is used as a receiver (left), sender (middle), and forwarder (right); each for two clients (Mac/Linux).

setup of the Nitrogen6X board without TrustZone as the baseline ("vanilla"). We disable NIC offloading features as our current driver implementation does not yet support them and map device registers uncached due to OP-TEE's limited mapping support [33]. For both setups, we perform 20 iterations for each test case.

Figure 5 shows the performance results for all six tests. (i) TruGW reaches a receive throughput of about 385 Mbps, which is about 90% of the vanilla throughput. The observed overhead is likely caused by the current implementation of VNIC's U_P interface (Virtio-mmio). VNIC currently uses (legacy) interrupts for buffer notifications to U_P (and access traps in the opposite direction) which can frequently interrupt the U_P iPerf3 thread. We could further improve the results by refining VNIC's batch processing, but we regard the performance hit as acceptable for rare bulk uploads to router services. (ii) TruGW reaches a transmission throughput of 392 Mbps for $Host_M$, which is about 6.5% higher than that of vanilla (368 Mbps avg.). For $Host_L$, TruGW reaches 99% of vanilla's average throughput (369 Mbps). While TruGW's throughput to $Host_M$ is currently higher than vanilla, we have observed comparable maximum throughput values for $Host_L$ and vanilla, too. VNIC currently performs aggressive packet forwarding retries on a NIC congestion, which seem to benefit from $Host_M$'s ACK sending behaviour. (iii) Lastly, TruGW's forwarding performance reaches 92.6–93.8% (236 Mbps avg.) of the vanilla throughput when $Host_M$ is the sender and 101.9–103.5% (221 Mbps avg.) when $Host_M$ is the receiver. In summary, TruGW shows an overall high throughput ≥ 90% (*AR4*) and performs similar to the vanilla system (92.6–103.5%) when forwarding.

7.3.2 *Firewall Overhead.*

We now measure if adding NPF firewall rules causes overhead. We repeated the three iPerf3 measurements with $Host_M$ while applying filter rules, i.e., the "Mac Router RX/TX" and the "$M \longrightarrow L$" benchmarks of Figure 5. For RX/TX, we defined rules on the VNIC interface, which check for TCP connections to TruGW's IP on iPerf3's port. For forwarding, we defined analogous rules on the NIC interface to match iPerf3's connections to $Host_L$. We performed 20 iterations of each test with (a) stateful rules (connection tracking) and (b) bidirectional, stateless rules.

We observed a small overhead of about 0.5 to 1% in each test. This is not surprising, because NPF enforces rules using just-in-time compiled BPF code and has a fast path for connection tracking, which enables efficient allowlisting policies. While the overhead will naturally increase with large rulesets, the observed overhead comes from NPF's static code. As long as stateful policies capture most of the traffic—which is the norm for most networks—the overhead is thus marginal.

7.3.3 *Latency Overhead.*

We now evaluate how TruGW affects latency during web browsing and on a per-packet basis.

Web browsing. To follow a typical user scenario, we measure the client-side load times of web pages. We selected the ten stable pages from the top 13 of the Tranco list [32] for the evaluation. We excluded "tmall.com" and "qq.com" as they blocked the page load or faced a high baseline variance (multiple seconds) and "windowsupdate.com" as Chrome refused to load it. For each page, we measured the average load times over 10 iterations from $Host_M$ using a Chrome extension [62]. We kept all DNS entries cached, but cleared the web caches after each page load. We compare the baseline without TruGW (using a home router as $Host_M$'s direct gateway to a ≈60 Mbps line) to a setup with TruGW as an additional intermediate router between them.

TruGW incurs an average load time overhead of ≈3.4% reaching from 0.07% to 4.95% peak. The latency is low when most packets arrive while TruGW's I/O workers are still polling, and is slightly higher when TruGW's notifier must wake them up (cf. §5.1). The workers partially compensate this by having an idle grace period before entering the sleep state. We regard the observed average overhead as reasonably small. Most of the overheads translate to page load delays of about 30 ms (cf. Table 5, Appendix), which is not noticeable by average users.

Packet Latency. To gauge how latency-critical applications (e.g., gaming) are affected by TruGW, we also evaluate the per-packet latency using `ping`. We measure the average round trip time (RTT) from $Host_M$ to an external server for 1000 packets over 10 iterations. We use the same baseline as in the page load test. TruGW shows an average RTT of 14.22 ms, which is a tiny per-packet slowdown of ≈0.37 ms (≈2.67 %) compared to the average baseline of 13.85 ms.

7.3.4 *Trusted ConfigService Load.*

We now briefly report on the page load time of ConfigService's master admin page. We follow the approach of the previous section (§7.3.3). The load time includes the server *and client* TLS authentication and the fetching of all ConfigService web resources. We have observed an average load time of about 1385 ms, which fulfills current user expectations of 1–2 seconds [63]. We can further optimize ConfigService if required.

7.4 Secure Memory Overhead

Since routers are usually resource-constrained devices, we now discuss the secure memory overhead of TruGW. TruGW currently shares OP-TEE's default configuration and claims 30 MB of the system RAM exclusively for the trusted partition T_P and 2 MB for shared memory. VNIC additionally claims 268 B for its virtual device registers [47]. These memory requirements are easily met by commodity router platforms. For instance, OpenWrt [5] recommends ≥ 128 MB of RAM for routers, which is fulfilled by the majority of its supported ARM devices.[7] In addition, the TruGW

[7]https://openwrt.org/toh/views/toh_available_16128

prototype currently leaves ≈20 MB of the 32 MB for trusted user apps, such that we could further reduce TruGW's memory requirements, likely to ≈16 MB. The exact memory demand depends on the number of NICs and their I/O buffer sizes. For instance, the FEC NIC uses two rings à 512 entries for Ethernet frames (≈1.5 MB). However, TruGW relocates NIC I/O buffers, egress queues, and firewall states from U_P to T_P, i.e., they do not increase the overall system demands.

ConfigService has a small memory footprint inside the TA memory. The demands are defined by the per-client TLS ringbuffers (≈4 kB), HTTP buffers (≈4 kB), and internal TLS buffers (≈3.6–32 kB). The TLS admin certificates and the HTML file are securely stored on disk. Other web resources are offloaded to U_P (cf. §5.4).

8 CONCLUSION

The increasing attack surface introduced by commodity gateway OSes and auxiliary services enables remote attackers to easily compromise gateway routers and bypass their security-critical network policies. Unfortunately, network infrastructures still widely rely on the assumption that gateways are trusted ("bastion hosts"). Existing ad-hoc protection attempts result in large attack surfaces or are not suitable for the protection of stand-alone consumer and SME gateways. TruGW bridges this important gap by guaranteeing trusted policy enforcement with a small attack surface even on a fully compromised gateway. TruGW's design builds on widely-available hardware and software features (e.g., TrustZone, Virtio) to enable an affordable and readily deployable secure gateway architecture. TruGW thus restores the trust in the security of gateway routers.

REFERENCES

[1] Devdatta Akhawe, Joel Weinberger, Frederik Braun, and Francois Marier. 2016. *Subresource Integrity.* W3C Recommendation. W3C. https://www.w3.org/TR/2016/REC-SRI-20160623/.

[2] Daniele Enrico Asoni, Takayuki Sasaki, and Adrian Perrig. 2018. Alcatraz: Data Exfiltration-Resilient Corporate Network Architecture. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC).* 176–187. https://doi.org/10.1109/CIC.2018.00033

[3] Adam Barth, Collin Jackson, and John C. Mitchell. 2008. Robust Defenses for Cross-Site Request Forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '08).* Association for Computing Machinery, New York, NY, USA, 75–88. https://doi.org/10.1145/1455770.1455782

[4] Paul Beesley, Sumit Garg, and Sandrine Bailleux. 2020. *Trusted Board Boot.* Retrieved June 29, 2022 from https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/design/trusted-board-boot.rst

[5] Rich Brown. 2022. *Welcome to the OpenWrt Project.* https://openwrt.org/

[6] Serdar Cabuk, Carla E. Brodley, and Clay Shields. 2004. IP Covert Timing Channels: Design and Detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security* (Washington DC, USA) *(CCS '04).* Association for Computing Machinery, New York, NY, USA, 178–187. https://doi.org/10.1145/1030083.1030108

[7] Matteo Carlini. 2017. Secure Boot on Arm systems. Retrieved June 29, 2022 from https://www.slideshare.net/linaroorg/secure-boot-on-arm-systems-building-a-complete-chain-of-trust-upon-existing-industry-standards-using-opensource-firmware-sfo17201

[8] Yueqiang Cheng and Xuhua Ding. 2013. Guardian: Hypervisor as Security Foothold for Personal Computers. In *Trust and Trustworthy Computing*, Michael Huth, N. Asokan, Srdjan Čapkun, Ivan Flechais, and Lizzie Coles-Kemp (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 19–36.

[9] Inc. Cisco Systems. 2015. *Cisco IOS XR Software Release 6.0 Operational Enhancements Data Sheet.* Retrieved March 2, 2022 from https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-xr-software/datasheet-c78-736154.html

[10] Inc. Cisco Systems. 2020. *KVM App Hosting on a Cisco Router.* Retrieved June 27, 2022 from https://www.cisco.com/c/en/us/products/collateral/routers/4000-series-integrated-services-routers-isr/at-a-glance-c45-737753.html

[11] Inc. Cisco Systems. 2021. *Troubleshoot High CPU Usage in Catalyst Switch Platforms Running IOS-XE 16.x.* Retrieved March 2, 2022 from https://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-xe-16/213549-troubleshoot-high-cpu-usage-in-catalyst.html

[12] Emscripten Contributors. 2022. *Emscripten documentation.* https://emscripten.org

[13] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. 2005. *Linux Device Drivers, 3rd Edition.* O'Reilly Media, Inc.

[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086

[15] Al Danial. 2022. *cloc: Count Lines of Code.* https://github.com/AlDanial/cloc

[16] Boundary Devices. 2022. *i.MX6 Embedded Single Board Computer (Nitrogen6X).* Retrieved June 29, 2022 from https://boundarydevices.com/product/nitrogen6x/

[17] Huayi Duan, Cong Wang, Xingliang Yuan, Yajin Zhou, Qian Wang, and Kui Ren. 2019. LightBox: Full-Stack Protected Stateful Middlebox at Lightning Speed. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19).* Association for Computing Machinery, New York, NY, USA, 2351–2367. https://doi.org/10.1145/3319535.3339814

[18] embeDD GmbH. 2022. *DD-WRT.* https://dd-wrt.com/

[19] Saba Eskandarian, Jonathan Cogan, Sawyer Birnbaum, Peh Chang Wei Brandon, Dillon Franke, Forest Fraser, Gaspar Garcia, Eric Gong, Hung T. Nguyen, Taresh K. Sethi, Vishal Subbiah, Michael Backes, Giancarlo Pellegrino, and Dan Boneh. 2019. Fidelius: Protecting User Secrets from Compromised Browsers. In *2019 IEEE Symposium on Security and Privacy (SP).* 264–280. https://doi.org/10.1109/SP.2019.00036

[20] The Linux Foundation. 2022. *Xen Project.* https://xenproject.org/

[21] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. 2020. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication. In *2020 IEEE Symposium on Security and Privacy (SP).* 268–285. https://doi.org/10.1109/SP40000.2020.00047

[22] Vivien Gueant. 2022. *iPerf - The TCP, UDP and SCTP network bandwidth measurement tool.* https://iperf.fr/

[23] Global Platform Inc. 2010. TEE Client API Specification v1.0. Retrieved June 29, 2022 from https://globalplatform.org/specs-library/tee-client-api-specification/

[24] Yeongjin Jang. 2017. *Building trust in the user I/O in computer systems.* Ph. D. Dissertation. Atlanta, GA, USA.

[25] Inc. Juniper Networks. 2022. *Junos OS Evolved Overview.* Retrieved June 27, 2022 from https://www.juniper.net/documentation/us/en/software/junos/overview-evo/topics/concept/evo-overview.html

[26] Inc. Juniper Networks. 2022. *Junos OS Overview.* Retrieved June 27, 2022 from https://www.juniper.net/documentation/us/en/software/junos/junos-install-upgrade/topics/topic-map/junos-os-overview.html

[27] Inc. Juniper Networks. 2022. *VM Host Overview (Junos OS).* Retrieved June 27, 2022 from https://www.juniper.net/documentation/us/en/software/junos/junos-install-upgrade/topics/topic-map/vm-host-overview.html

[28] Qiao Kang, Lei Xue, Adam Morrison, Yuxin Tang, Ang Chen, and Xiapu Luo. 2020. Programmable In-Network Security for Context-aware BYOD Policies. In *29th USENIX Security Symposium (USENIX Security 20).* USENIX Association, 595–612. https://www.usenix.org/conference/usenixsecurity20/presentation/kang

[29] The kernel development community. 2021. *Interrupts – The Linux Kernel documentation.* Retrieved April 4, 2022 from https://linux-kernel-labs.github.io/refs/heads/master/lectures/interrupts.html#interrupt-context

[30] Michael Kerrisk. 2012. *ip-route(8) - Linux manual page.* https://man7.org/linux/man-pages/man8/ip-route.8.html

[31] Se Won Kim, Chiyoung Lee, MooWoong Jeon, Hae Young Kwon, Hyun Woo Lee, and Chuck Yoo. 2013. Secure device access for automotive software. In *2013 International Conference on Connected Vehicles and Expo (ICCVE).* 177–181. https://doi.org/10.1109/ICCVE.2013.6799789

[32] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019).* https://doi.org/10.14722/ndss.2019.23386

[33] Matthew Lentz, Rijurekha Sen, Peter Druschel, and Bobby Bhattacharjee. 2018. SeCloak: ARM Trustzone-Based Mobile Peripheral Control. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (Munich, Germany) *(MobiSys '18).* Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3210240.3210334

[34] Thomas Leonard. 2022. *QubesOS Mirage Firewall.* Retrieved June 29, 2022 from https://github.com/mirage/qubes-mirage-firewall/

[35] Wenhao Li, Mingyang Ma, Jinchen Han, Yubin Xia, Binyu Zang, Cheng-Kang Chu, and Tieyan Li. 2014. Building Trusted Path on Untrusted Device Drivers for Mobile Devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems* (Beijing, China) *(APSys '14).* Association for Computing Machinery, New York, NY, USA, Article 8, 7 pages. https://doi.org/10.1145/2637166.2637225

[36] Team libtom. 2022. *LibTomCrypt.* https://github.com/libtom/libtomcrypt

[37] Arm Limited. 2018. Trusted Board Boot Requirements CLIENT (TBBR-CLIENT) Armv8-A. Retrieved April 4, 2022 from https://developer.arm.com/documentation/den0006/latest

[38] Arm Limited. 2019. Arm Platform Security Architecture Trusted Boot and Firmware Update 1.0. Retrieved June 29, 2022 from https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/PSA/DEN0072-PSA_TBFU_1-0-REL.pdf

[39] Linaro Limited. 2022. *DeviceTree.* https://www.devicetree.org

[40] Linaro Limited. 2022. *Open Portable Trusted Execution Environment - OP-TEE.* https://www.op-tee.org/

[41] Renju Liu and Mani Srivastava. 2017. PROTC: PROTeCting Drone's Peripherals through ARM TrustZone. In *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications* (Niagara Falls, New York, USA) *(DroNet '17)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3086439.3086443

[42] Google LLC. 2020. *Trusty TEE | Android Open Source Project.* Retrieved June 29, 2022 from https://source.android.com/security/trusty

[43] Google LLC. 2022. *chrome.enterprise.platformKeys – Chrome Developers.* Retrieved June 29, 2022 from https://developer.chrome.com/docs/extensions/reference/enterprise_platformKeys/

[44] Google LLC. 2022. *Hardware-backed Keystore | Android Open Source Project.* Retrieved June 29, 2022 from https://source.android.com/security/keystore

[45] Matt McCormack, Amit Vasudevan, Guyue Liu, Sebastián Echeverría, Kyle O'Meara, Grace Lewis, and Vyas Sekar. 2020. Towards an Architecture for Trusted Edge IoT Security Gateways. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association. https://www.usenix.org/conference/hotedge20/presentation/mccormack

[46] Kazuho Oku, Tokuhiro Matsuno, Daisuke Murase, and Shigeo Mitsunari. 2021. *PicoHTTPParser.* https://github.com/h2o/picohttpparser

[47] OASIS Open. 2019. Virtual I/O Device (VIRTIO) Version 1.1, Michael S. Tsirkin and Cornelia Huck (Eds.). OASIS Committee. Retrieved June 29, 2022 from https://docs.oasis-open.org/virtio/virtio/v1.1/virtio-v1.1.html

[48] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. 2019. StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 537–554. https://www.usenix.org/conference/atc19/presentation/park-heejin

[49] Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* 51, 6, Article 130 (jan 2019), 36 pages. https://doi.org/10.1145/3291047

[50] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2018. SafeBricks: Shielding Network Functions in the Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 201–216. https://www.usenix.org/conference/nsdi18/presentation/poddar

[51] Salvatore Pontarelli, Roberto Bifulco, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, and Giuseppe Siracusano. 2019. FlowBlaze: Stateful Packet Processing in Hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 531–548. https://www.usenix.org/conference/nsdi19/presentation/pontarelli

[52] The Qubes OS Project et al. 2022. *Firewall | Qubes OS.* Retrieved June 29, 2022 from https://www.qubes-os.org/doc/firewall/

[53] Mindaugas Rasiukevicius. 2021. *NPF-Router: a demo NPF+DPDK application.* https://github.com/rmind/npf/tree/master/app

[54] Mindaugas Rasiukevicius. 2021. *NPF: stateful packet filter supporting NAT, IP sets, etc.* https://github.com/rmind/npf

[55] Sergej Schumilo, Cornelius Aschermann, Ali Abbasi, Simon Wör-ner, and Thorsten Holz. 2021. Nyx: Greybox Hypervisor Fuzzing using Fast Snapshots and Affine Types. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2597–2614. https://www.usenix.org/conference/usenixsecurity21/presentation/schumilo

[56] Sergej Schumilo, Cornelius Aschermann, Robert Gawlik, Sebastian Schinzel, and Thorsten Holz. 2017. kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 167–182. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schumilo

[57] Fabian Schwarz and Christian Rossow. 2020. SENG, the SGX-Enforcing Network Gateway: Authorizing Communication from Shielded Clients. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 753–770. https://www.usenix.org/conference/usenixsecurity20/presentation/schwarz

[58] NXP Semiconductors. 2018. *i.MX 6Dual/6Quad Applications Processors for Consumer Products.* Retrieved July 4, 2022 from https://www.nxp.com/docs/en/data-sheet/IMX6DQCEC.pdf

[59] Abhinav Srivastava and Jonathon Giffin. 2008. Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections. In *Recent Advances in Intrusion Detection*, Richard Lippmann, Engin Kirda, and Ari Trachtenberg (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 39–58.

[60] Stefan Luber and Andreas Donner. 2019. *Was ist eine Fritzbox?* Retrieved March 2, 2022 from https://www.ip-insider.de/was-ist-eine-fritzbox-a-883753/

[61] SEG / DrayTek UK. 2015. *O/S versions on Vigor 2760 Series Routers (DrayOS/Linux).* Retrieved March 2, 2022 from https://www.draytek.co.uk/support/guides/os-versions-on-vigor-2760-series-routers

[62] Alexander Vykhodtsev. 2021. *page-load-time.* https://github.com/alex-vv/page-load-time

[63] Anna Wilson. 2020. *Website Load Time Statistics.* Retrieved June 27, 2022 from https://www.top10-websitehosting.co.uk/website-load-time-statistics/

[64] Wei Wu, Yueqi Chen, Xinyu Xing, and Wei Zou. 2019. KEPLER: Facilitating Control-flow Hijacking Primitive Evaluation for Linux Kernel Vulnerabilities. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1187–1204. https://www.usenix.org/conference/usenixsecurity19/presentation/wu-wei

[65] Jiarong Xing, Qiao Kang, and Ang Chen. 2020. NetWarden: Mitigating Network Covert Channels while Preserving Performance. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2039–2056. https://www.usenix.org/conference/usenixsecurity20/presentation/xing

[66] Kailiang Ying, Amit Ahlawat, Bilal Alsharifi, Yuexin Jiang, Priyank Thavai, and Wenliang Du. 2018. TruZ-Droid: Integrating TrustZone with Mobile Operating System. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (Munich, Germany) *(MobiSys '18)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3210240.3210338

[67] Zongwei Zhou, Virgil D. Gligor, James Newsome, and Jonathan M. McCune. 2012. Building Verifiable Trusted Path on Commodity x86 Computers. In *2012 IEEE Symposium on Security and Privacy.* 616–630. https://doi.org/10.1109/SP.2012.42

[68] Zongwei Zhou, Miao Yu, and Virgil D. Gligor. 2014. Dancing with Giants: Wimpy Kernels for On-Demand Isolated I/O. In *2014 IEEE Symposium on Security and Privacy.* 308–323. https://doi.org/10.1109/SP.2014.27

[69] Yong-Hao Zou, Jia-Ju Bai, Jielong Zhou, Jianfeng Tan, Chenggang Qin, and Shi-Min Hu. 2021. TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 489–502. https://www.usenix.org/conference/atc21/presentation/zou

A ARM SECURE AND TRUSTED BOOT

ARM secure boot provides mechanisms to verify that only trusted images are loaded during system boot. To realize this, the boot images are signed and each loader verifies the image of the next stage before transferring control to it. That way, secure boot establishes a chain of trust which is rooted in a trusted root signing key. While the details are implementation-specific, the concepts of secure boot are well known [4, 37]. They typically include: (i) a trusted root key k_{rot} stored (typically by the manufacturer) in tamper-proof non-volatile storage (e.g., OTP) inaccessible by system software, (ii) a trusted boot ROM which uses k_{rot} to verify the signature of the first stage bootloader image, and (iii) a set of public key hashes used to verify subsequent boot images (e.g., TEE image, U_P bootloader) [4, 37]. The TrustZone-specific trusted board boot [37] follows these principles to enable verification of the TEE (T_P) image(s) and the U_P bootloader. As TruGW's trusted kernel components (e.g., NetTrug) are direct extensions of OP-TEE's TEE image(s) (cf. §5.1 and §5.5), their integrity is securely verified on boot and therefore guaranteed to be in a trusted state. Optionally, TruGW can include all U_P OS images in the secure boot chain, e.g., by combining trusted boot with UEFI secure boot [7].

B OP-TEE'S THREAD SCHEDULING

OP-TEE [40] is not in control of the CPU scheduling and relies on the Linux scheduler for running secure tasks. Linux applications must explicitly call into OP-TEE for service. This design suites OP-TEE's service-oriented design and principle of least privilege and contributes to OP-TEE's small TCB (*SR7*). OP-TEE implements the GlobalPlatform TEE Client API [23] which enables applications

Table 2: A sample of router network operating systems (NOS) and the respective commodity OSes they derive from. Many popular router NOSes are based on commodity OSes and therefore inherit their security vulnerabilities. Note that many "old", hardware-specific NOSes nowadays run as userspace services or VMs on recent platforms.

Vendor	Network OS	Underlying Commodity OS
AVM	Fritz!OS	Linux [60]
Cisco	IOS XR	Linux (Wind River), old: QNX [9]
Cisco	IOS XE	Linux [11]
DrayTek	—	Linux (now: DrayOS) [61]
Juniper	Junos OS	FreeBSD [26]
Juniper	Junos OS Evolved	Linux [25]
—	DD-/OpenWRT	Linux [5, 18]

Table 3: Number of CVE entries (2.3.22) for OSes, hypervisors, and Linux networking components based on categories of `cvedetails.com` and keyword searches on `cve.mitre.org`. The CVEs show that (i) security kernels (e.g. OP-TEE) face a way smaller risk of exploitation than commodity OSes used by routers, and (ii) the Linux firewall and NIC drivers add only minimally to the risk of code execution (CE) vulnerabilities compared to the full kernel or hypervisors. Note that Xen and KVM require an additional host OS (e.g. Linux).

Product	Total CVEs	CE	Search Keywords
Linux kernel	2763	263	—
Win. 10 OS	2590	538	—
FreeBSD OS	455	54	—
OP-TEE OS	10	3	—
Xen	378	20	—
QEMU (KVM)	355	77	—
Linux Firewall	67	2	linux netfilter
Eth. NIC Drv.	25	1	linux drivers net ethernet
Wireless Drv.	39	1	linux drivers net wireless

to create a session with an OP-TEE trusted application (TA) and then invoke TA-exposed RPC interfaces. OP-TEE's Linux driver and secure kernel handle the resulting thread context switches between Linux (in U_P) and OP-TEE's TAs (in T_P) based on TrustZone's SMC CPU instruction [49]. OP-TEE's kernel stores the execution contexts of the Linux threads in trusted TEE thread structures while they perform TEE tasks. In Figure 3, the resulting control flow is shown for TruGW's SockHelper and ConfigService TA.

However, a design like OP-TEE's causes high performance overhead and limitations. Keeping the thread scheduler in U_P significantly increases the required number of expensive context switches between T_P and U_P on common tasks, e.g., thread synchronisation. Furthermore, trusted T_P interrupt handlers cannot schedule TEE tasks as the U_P scheduling APIs used for TEE-associated threads are context-switching and therefore incompatible with interrupt contexts [29]. For that reason, such scheduling designs (incl. OP-TEE's) are unsuitable for fast, trusted network I/O as required by TruGW. This motivated us to design TruGW's new worker framework which overcomes these limitations and enables efficient NIC I/O in T_P while keeping the scheduler in U_P (cf. §5.1).

Table 4: Code sizes of auxiliary network services on DD-WRT routers (rev. 47201) in thousands of lines of code.

		Lines of Code [kLOC]		
Service	Short Description	Total	C/ASM	Hdrs
asterisk	VoIP server	766.6	673.7	92.9
dropbear	Sys Utils (incl. sshd)	95.3	87.5	7.8
freeradius3	Authentication service	116.1	109.6	6.5
krb5	Authentication service	308.3	256.9	51.4
lighttpd	Web server	82.9	71.5	11.4
minidlna	Streaming server	691.0	553.3	137.7
nginx	Web server	140.8	132.3	8.5
proftpd	FTP daemon	227.7	220.9	6.8
samba4	File sharing server	1515.3	1431.8	83.5
snmp	Sys/Net monitoring	288.2	257.9	30.3
squid	Web proxy	51.3	15.0	36.3
zabbix	Sys/Net monitoring	233.5	221.1	12.4
SUM		4517.0	4031.5	485.5

Table 5: Overview of Chrome page load times and overhead when routing through TruGW as an intermediate router.

	avg. load [ms]		
Web Page	Baseline	TruGW	Overhead
instagram.com	1298.5	1362.8	4.95%
linkedin.com	654.0	685.8	4.86%
google.com	563.0	590.1	4.81%
youtube.com	560.8	587.2	4.71%
microsoft.com	823.1	856.3	4.03%
baidu.com	6642.9	6895.1	3.80%
facebook.com	813.2	843.6	3.74%
apple.com	963.8	993.0	3.03%
wikipedia.org	701.5	704.5	0.43%
twitter.com	1125.7	1126.5	0.07%

Table 6: A sample of recent security critical vulnerabilities in auxiliary services, OS kernels, and hypervisors (VMMs) used by popular network devices. The CVEs show that remote attackers can fully compromise such devices by chaining remote code execution exploits to OS and (if required) VMM exploits. Thus, attackers gain full control over a device's routing and firewall.

	CVE	Device	Target Component / Vulnerability	Attack Effect
Userspace Network Services	2019-16028	Cisco Firepower Firewall	LDAP Bypass (via HTTP)	remote admin access
	2019-17621	D-Link DIR-859 Wi-Fi router	UPnP service (via HTTP)	remote code execution (LAN)
	2019-19494	Broadcom-based cable modems	buffer overflow (via JS)	remote kernel code execution
	2020-3115	Cisco SD-WAN	vManage (input validation error)	local privilege escalation (root)
	2020-11503	Sophos XG Firewall	awarrensmtp (heap overflow)	remote code execution
	2020-15635	Netgear WLAN Router R6700	acsd service (buffer overflow)	remote code execution
	2020-27600	D-Link Router DIR-846	HNAP service	remote command execution
	2021-0254	Juniper ACX/MX routers	overlayd (buffer overflow)	remote code execution
	2021-0260	Juniper net. devices (Junos OS)	snmpd (improper authorization)	remote SNMP read/write access
	2021-1287	Cisco Wireless VPN routers	web mngt. interface	remote code execution (root)
	2021-1539	Cisco ASR-5000 routers	TACACS auth. bypass (via SSH)	remote command execution
	2021-1602	Cisco RV160/260 Routers	web mngt. interface	remote code execution (root)
OS	2020-7460	FreeBSD-based Routers (Table 2)	FreeBSD kernel	local kernel code execution
	2021-31440	Linux-based Routers (Table 2)	Linux kernel 5.11.15	local kernel code execution
VMM	2020-7467	FreeBSD-based Routers (Table 2)	bhyve (FreeBSD hypervisor)	VM escape (host code exec.)
	2020-14364	Linux-based Routers (Table 2)	KVM-QEMU (Linux hypervisor)	VM escape (host code exec.)