Towards an Architecture for Trusted Edge IoT Security Gateways

Matt McCormack,* Amit Vasudevan,† Guyue Liu,* Sebastián Echeverría,† Kyle O'Meara,†
Grace Lewis,† Vyas Sekar*
Carnegie Mellon University - CyLab, †Carnegie Mellon Software Engineering Institute

Abstract

Today's edge networks continue to see an increasing number of deployed IoT devices. These IoT devices aim to increase productivity and efficiency; however, they are plagued by a myriad of vulnerabilities. Industry and academia have proposed protecting these devices by deploying a "bolt-on" security gateway to these edge networks. The gateway applies security protections at the network level. While security gateways are an attractive solution, they raise a fundamental concern: *Can the bolt-on security gateway be trusted?*

This paper identifies key challenges in realizing this goal and sketches a roadmap for providing trust in bolt-on edge IoT security gateways. Specifically, we show the promise of using a micro-hypervisor driven approach for delivering practical (deployable today) trust that is catered to both end-users and gateway vendors alike in terms of cost, generality, capabilities, and performance. We describe the challenges in establishing trust on today's edge security gateways, formalize the adversary and trust properties, describe our system architecture, present preliminary results, and discuss open questions. We foresee our trusted security gateway architecture becoming a practical and extensible foundation towards realizing robust trust properties on edge security gateways.

1 Introduction

IoT devices are increasingly being deployed into edge environments, from home networks to manufacturing floors. Unfortunately, these devices are plagued by a myriad of vulnerabilities [2,73], which attackers have leveraged as stepping stones into protected networks and as launch pads for other attacks [4,36,39,46]. Consequently, these IoT devices pose a continuing threat to the security of our edge networks.

Industry and academia have proposed securing (potentially vulnerable) IoT devices on edge networks with on-site security gateways [7,9,13,24,43,54,73]. These "bolt-on" security gateways are designed to intercept all traffic to and from an IoT device and apply security protections via *middleboxes* at the *network level* (*e.g.,* a firewall). These middleboxes can be used to implement "network patches" which mitigate a device's vulnerabilities without patching the device's software.

While bolt-on security gateways are gaining popularity and are a deployable solution, they raise a fundamental question: *How do we ensure that the system providing these network level protections is trustworthy?* As an example scenario, consider a smart factory with a plethora of IoT devices protected by multiple security gateways. The factory's security gateways provide network level protections tailored to each individual IoT device's vulnerabilities. Unfortunately, security gateways form a single point of failure. They are particularly vulnerable to an adversary who can compromise the security gateway (by exploiting OS and/or application vulnerabilities), as the gateway typically runs commodity software (*e.g.,* Linux, Docker, OVS, Snort, etc.). Once compromised, an adversary can modify the gateway's protections (*e.g.,* remove a firewall rule) thereby enabling attacks against an IoT device in order to stop/alter production (à la [8,29]).

Current approaches for securing applications in untrusted cloud environments could potentially be applied to establish trust in security gateways. These approaches rely on hardware-specific capabilities (*e.g.,* SGX [37,59], MPX [76]). Unfortunately, such approaches have high performance overheads (not practical for IoT deployments) and also lack generality. They are limited to a specific processor class and only support user space applications with constrained memory allocation. Furthermore, addressing security vulnerabilities [31,51,71] requires fabricating newer revisions of the hardware.

At a high level, we envision a *trusted* IoT security gateway architecture, that provides an overarching guarantee that the correct security protections are applied to each IoT device's network traffic at all times, including when under attack (more details in §4). We use this aforementioned definition of trust throughout this paper. Our architecture aims to provide robust trust properties to a broad range of legacy hardware platforms utilizing existing software with a reasonable performance overhead. There are three challenges to realizing our vision:

- **Formalizing Adversary and Trust Properties (§4)**: To design a trusted architecture, we need to consider a *rich adversary* model, where the adversary could attack any software component and data in transit. Existing security gateway architectures often utilize a software defined network (SDN) architecture [73], where the data plane enforces network level protections, and the control plane orchestrates these protections to achieve a policy. While prior work on SDN security [14,26,69] explored some attack scenarios, they tackle a limited adversary model, only analyzing a subset of the architecture (*e.g.,* routing, application permissions). Both control and data plane elements and their communications must be protected to achieve trust.
- **Supporting Dynamic Middleboxes (§5)**: The architecture must provide trust in dynamic middleboxes that are constantly being reconfigured (*e.g.,* IoT devices frequently leaving and joining edge networks). Prior work in cloud computing proposed using secure hardware (*e.g.,* SGX, TrustZone), placing entire applications in a trusted execution environment (*e.g.,* enclave). While this approach could prevent tampering, it fails to support today's dynamic mid-

- dleboxes due to limited available memory (*e.g.,* 128MB on SGX [34]), reduced functionality (*e.g.,* inability to perform system calls [18] required for timestamps), and the high performance costs of changing enclaves (*e.g.,* reducing performance by up to 30% [52,66]). Furthermore, only placing pieces of the application in an enclave suffers severe performance costs [52]. An ideal solution provides trust to legacy software on any hardware platform in a performant manner.
- **Secure and Efficient Communication** (§6): A trusted security gateway requires secure communications, enforcing protections at a per-packet granularity both between and across the control and data planes. Existing tunneling techniques (*e.g.,* IPSec, TLS) could be used between planes, but are too expensive for protecting across a plane (*e.g.,* tunneling a packet between middleboxes on the data plane). Low performance overheads are required for latency-sensitive devices (*e.g.,* real-time, closed-loop robot controllers).

In this paper, we argue that a *micro-hypervisor* based approach is a promising architectural basis for building trust in edge security gateways. A micro-hypervisor, like a traditional hypervisor, is a software reference monitor that provides core security capabilities (*e.g.,* memory isolation, mediation, and attestation) that can be applied to effectively address the aforementioned challenges. In contrast to traditional hypervisors, these capabilities are provided with a dramatically reduced trusted computing base (TCB) and complexity (hence the *micro* prefix) which enable formal verification to rule out potential vulnerabilities [3,62,63]. Furthermore, micro-hypervisors provide an extensible foundation for realizing robust trust properties without a loss of generality and minimal performance overhead [50, 58, 61–63]. Last but not least, in contrast to approaches using specific hardware capabilities which limit applications (*e.g.,* SGX's limitations described above), micro-hypervisors can support a variety of hardware platforms (x86 [50,62], ARM [61], microcontroller [3]) running unmodified software (*e.g.,* Linux) [3,58,63]. Thus, a micro-hypervisor provides a practical and secure foundation for building security mechanisms towards realizing our vision.

Our intuition to leverage a micro-hypervisor based approach is motivated by the success micro-hypervisors have had on commodity platforms [60]. However, to the best of our knowledge, micro-hypervisors have not been used in edge IoT gateways. To this end, our contributions are: (1) a more holistic system adversary model for an edge IoT security gateway and (2) a high-level architecture based on micro-hypervisors to enable a practical and flexible solution.

2 Motivation

Traditional security solutions (*e.g.,* antivirus) fall short for IoT devices due to resource requirements and device heterogeneity [24,73]. Security gateway based approaches [7,9,13,24,43,72,73] have been proposed to secure IoT deployments.

"Bolt-on" Security Gateways: At a high level, these approaches insert a security gateway running virtualized middleboxes (*e.g.,* firewall, IDS) to protect deployed IoT devices. To achieve this, the gateway intercepts all traffic to and from the IoT device and sends the network traffic to a middlebox which imposes a security policy (*e.g.,* IoT may not SSH).

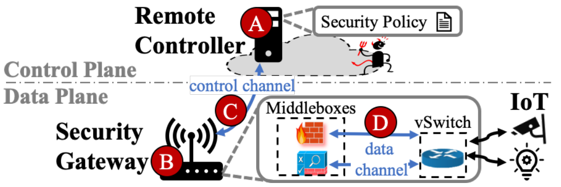

Figure 1: Attack vectors for bolt-on security architecture.

While initially these security gateways employed a single monolithic middlebox running a static configuration (*e.g.,* an IDS with a default ruleset), recent work [24, 72, 73] highlighted the need for isolated (*e.g.,* each device has its own set of middleboxes), device-specific (*e.g.,* each middlebox configured to protect a specific device's vulnerabilities) middleboxes that support dynamic security policies (*e.g.,* changing based upon context, such as other device's status). The need for these new capabilities has increased the complexity of the security gateway architectures (shown in Figure 1), adding virtual switches (vSwitch) for routing data to the appropriate middleboxes and a remote controller for dynamically configuring each gateway's protections (*e.g.,* middlebox configurations, vSwitch routes) to achieve the security policy.

These "bolt-on" gateways are promising for securing IoT deployments; however, they are currently untrusted. Under attack, these security gateways could become ineffective, or even worse, become a launchpad for new attacks.

Motivating Scenario: An attacker could launch attacks at multiple points in the architecture (shown in Figure 1). For example, an attacker could: (1) use an unpatched exploit [35, 70] to compromise the gateway itself (B in Figure 1) and (2) modify the middlebox configuration such that it allows the attacker's traffic to pass through to enable the attacker to compromise a factory's IoT device and steal proprietary data (à la [46]). Beyond modifying the software, an attacker could also tamper with network messages. For example, modifying packets on the data channel between the vSwitch and the middlebox (D in Figure 1), redirecting traffic to the wrong middlebox, evading security inspections.

A trusted security architecture needs to protect the gateway and controller's software while prohibiting tampering with network traffic. We look to prior work for potential solutions.

Strawman Solution from Prior Work: A natural strawman solution would be to run the gateway and controller software in an enclave, with a secure tunnel (*e.g.,* IPSec, TLS) protecting the control channel. This solution has been used for securing middleboxes from untrusted cloud providers [37,59].

While this solution could prevent tampering with the gateway and controller software and protects control messages, it requires specific hardware and has three key limitations. First, only limited applications are supported. Applications running inside an enclave have limited memory access (*i.e.,* 128MB for SGX) and can only perform user space actions (*e.g.,* no system calls). Second, there is a significant performance overhead for initiating communication with an enclave, which is magnified if multiple enclaves must be utilized (*e.g.,* isolating multiple middleboxes in a chain), impacting low-latency edge devices. Third, vulnerabilities identified in trusted hardware may require long timelines to patch. This approach would be sufficient for a single, static protection on the gateway; however, the need to support dynamic middleboxes which are isolated and constantly changing entails a different approach.

Ideally, we want a solution that can be deployed on a wide range of hardware platforms, including resource-constrained edge platforms. Further, it needs to support existing software applications, while adding minimal performance overhead.

3 Trusted Security Gateway Architecture

We envision a *trusted, extensible, and widely-deployable edge security gateway architecture* that addresses the security challenges of today's edge IoT deployments. When fully realized, our architecture would enable new trustworthy "security-as-a-service" offerings that providers (e.g., edge ISPs, CDNs, IoT providers) could offer to IoT consumers, ensuring the correct security protections are applied at all times. For instance, this architecture could provide a trusted mechanism for enforcing IoT security best practices (*e.g.,* access-control policies in a device's Manufacturer Usage Description specification [25]).

We can consider a strawman design space categorized along two axes. First, approaches dependent on hardware functionality (*e.g.,* [37, 59]) are limited in both the hardware platforms and software they can support. Additionally, their security properties rest on a complex and opaque implementation in microcode and silicon [12], known to have vulnerabilities [31, 51, 71]. Second, pure software approaches (*e.g.,* formal verification, secure programming languages) are limited as they require significant reimplementation and verification effort. As many commonly used software applications on edge security gateways span over 100,000 lines of C/Java this quickly becomes intractable.

We argue that it is dangerous to tie critical security features to either hardware implementations that require new hardware to address threats, or to software approaches that require significant reimplementation or formal verification effort of the entire software stack. Instead we advocate leveraging legacy hardware features in combination with a small TCB and extensible software framework to provide our fundamental trust properties and protect edge devices from evolving threats.

Consequently, we make a case for a *micro-hypervisor* based approach to enable a trusted edge security gateway architecture (Figure 2) that allows *retrofitting* security protections *to only the necessary* system components. A micro-hypervisor is in essence a software reference monitor [45], that acts as a guardian, implementing access control to system resources (*e.g.,* files, sockets) using a small TCB. These protections can be applied in a fine-grained manner, protecting a single data value (*e.g.,* secret key) or a complex set of objects (*e.g.,* virtual machine) with minimal performance overhead. Micro-hypervisors provide a strong foundation for fine-grained mediation, isolation, and attestation with a small TCB [30, 50, 62, 63], which allows for security services to be designed and implemented as extensions [63]. Due to their simplicity and small TCB, micro-hypervisors are amenable to formal verification for ruling out potential vulnerabilities within their code [3, 63]. Additionally, micro-hypervisors can potentially be supported on any hardware platform (*e.g.,* x86 [50, 62], ARM [61], microcontroller [3]).

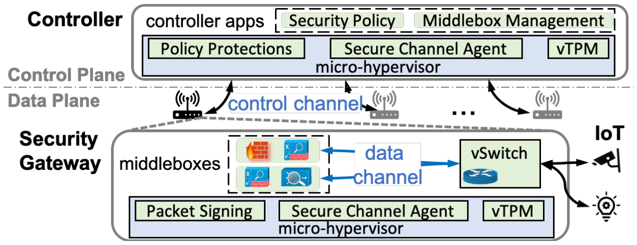

Figure 2: High-level view of a trusted, extensible, and widely-deployable edge security gateway architecture.

We build on top of the aforementioned micro-hypervisor enabled foundational capabilities to construct our trusted security gateway architecture. The controller and gateway's software run on top of a micro-hypervisor, allowing us to support any commodity OS and application stack. On the controller, we migrate critical data (*e.g.,* the security policy) into micro-hypervisor extensions to isolate it from untrusted software (*e.g.,* the OS). Further, all access is mediated by the micro-hypervisor, prohibiting an attacker from subverting the data's integrity. On the gateway, we assign a set of customized middleboxes to each device and isolate these from each other. Additionally, we periodically measure the signature of each middlebox and the vSwitch to verify their integrity. Finally, the controller and the gateway run *trusted agents*, which are micro-hypervisor extensions used to mediate communication between the control and data planes, to ensure the instantiated protections correctly reflect the security policy.

Our micro-hypervisor based architecture provides three key benefits. First, it provides fine-grained isolation and mediation which allow for precisely ensuring that the architecture enforces the correct protections while being performant. Second, it is extensible allowing for rapid growth of new security functionality and response to emerging threats. Third, it is widely-deployable, supporting a wide range of hardware platforms (*e.g.,* x86, ARM) while utilizing existing software, with a low performance overhead for providing trust.

There are three challenges our architecture must address:

- **Necessary Security Properties (§4):** Identify the security properties that ensure trust under a holistic adversary model. These properties guide the design of our architecture.
- **Support Dynamic Middleboxes (§5):** Enable protecting the dynamic middleboxes required by an IoT environment, ensuring an adversary cannot modify the protections.

Table 1: Holistic adversary capabilities, generated using the STRIDE model (referencing Fig. 1's attack vectors).

Example Threat	Vector	Violates
Modify controller software (*e.g.,* security policy)	A	P_{sw1}, P_{sw2}
Modify gateway software (*e.g.,* middlebox config)	B	
Spoof command (*e.g.,* add routing rule)	A or B	P_{sw3}
Spoof control channel message (*e.g.,* vSwitch route)	C	P_{comm4}
Tamper with data channel message (*e.g.,* skip middlebox)	D	P_{comm5}

- **Secure Communications (§6):** Provide per-packet protections with a low performance impact, guaranteeing an adversary cannot modify or spoof packets.

We now show our key building blocks that address these.

4 Adversary and Trust Properties

Our goal is to provide end-to-end protections against a holistic threat model. Within the SDN domain, this would entail protecting both the control and data planes (*e.g.,* from BGP hijacking [17,48]). However, prior works on IoT security gateways have typically only considered a narrow threat model.

To this end, we systematically define such an adversary, with a goal of inhibiting the gateway's protections (*i.e.,* enable exploiting a protected IoT device). We assume our adversary has knowledge of the security architecture as well as network access to all devices. We group our adversary capabilities into two categories: (1) ability to compromise a device's software stack (*i.e.,* software on the controller or gateway; A, B in Figure 1), and (2) ability to inject/modify network messages (*i.e.,* the control, data channels; C, D in Figure 1).

We use the STRIDE threat modeling tool [53] to generate a set of adversary capabilities (summarized in Table 1), that inhibit the architecture's ability to protecting an IoT device. While not a complete list, we use it to define the fundamental security properties needed for our trusted architecture.

Based upon our adversary model (Table 1), we posit that there are a minimum of five fundamental properties required for a trusted security gateway architecture.

- **Software Integrity** (P_{sw1}): Ability to detect code and data modifications (*e.g.,* changes in middlebox configuration).
- **Data Isolation** (P_{sw2}): Ability to isolate security critical logic (*e.g.,* keep the OS from accessing the security policy).
- **Data Mediation** (P_{sw3}): Ability to have a trusted entity mediate access to security critical data (*e.g.,* blocking an untrusted application's access to secret keys).
- **Secure Control Channel** (P_{comm1}): Ability to trust data transferred between the controller and gateway (*e.g.,* the gateway only executes commands from the controller).
- **Secure Data Channel** (P_{comm2}): Ability to that trust packets are routed through the correct middleboxes (*e.g.,* packets should not be processed by a wrong middlebox).

We envision our architecture supporting additional properties, but focus on these five as fundamental to a trusted architecture. These fundamental properties can be grouped into: (1) protecting running code (P_{sw}) and (2) protecting communications (P_{comm}). Next, we discuss our approach for providing these.

5 Supporting Dynamic Middleboxes

Ideally, the entire codebase on both the control and data planes could be robustly protected from an attacker. However, we view this as impractical as it either incurs significant performance costs (*e.g.,* multiple enclaves to process each packet) or requires significant reimplementation (*e.g.,* migrating 100,000+ lines of C/Java). Instead we look to apply fine-grained security properties to the portions of the codebase that impact the architecture's protections, thereby creating a robustness against our adversary (§4). Specifically, we apply periodic, remote attestation to guarantee the code's integrity (providing P_{sw1}). This allows the code to run with minimal performance degradation, while bounding the duration it is vulnerable to attack. Additionally, critical code (*e.g.,* security policy) can be protected with a hypervisor extension to isolate it (providing P_{sw2}) and mediate access to it (providing P_{sw3}).

Periodic, Remote Attestation: IoT security gateways rely upon a large codebase to provide device-specific protections. We look to prior work in remote attestation, such as Trusted Platform Modules (TPM) [6,11], in order to precisely guarantee that the appropriate software stack is running (providing P_{sw1}). Upon boot, the correct baseline software stack, composed of the micro-hypervisor, OS, and critical software components (*e.g.,* controller, vSwitch, middleboxes, etc.) is verified. Subsequently, new modules that will impact the provided protections (*e.g.,* a new middlebox's code prior to loading) are attested, thereby allowing the architecture to ensure that the correct protections are instantiated.

During runtime, we periodically re-attest critical modules (*e.g.,* controller, vSwitch, middleboxes) ensuring an attacker has not tampered with them. For example, the middlebox code must be run outside of the hypervisor to enable high packet throughput. To bound the potential impact of an attacker tampering with this code (*i.e.,* the protection not being applied), the controller remotely attests critical software components on the gateway at the end of every epoch, where the epoch duration can be adjusted to provide a trade-off between the vulnerable window's length and the security overhead.

We leverage a virtual trusted platform module (vTPM) on the micro-hypervisor to provide this attestation capability. A vTPM is a software implementation of a physical TPM and provides many of the same capabilities [5]. Specifically, we leverage its ability to securely store a chain of measurements, by extending a program control register (PCR), and securely providing those stored values (*i.e.,* a PCR quote). These two capabilities enable determining if a software stack on a local or a remote machine matches a known configuration. These vTPM measurements can be applied at a fine granularity, with separate storage for multiple measurements.

Protecting the Controller's Security Policy: While attestation can provide significant guarantees about a code's integrity, there are some pieces of code that merit further protection (*e.g.,* code impacting decisions about the protections provided by the security gateway). Our micro-hypervisor approach enables selectively isolating this code (P_{sw2}) and requiring that access to it be mediated by a trusted entity (P_{sw3}). Examples of such pieces of code are the secret keys used to establish a secure control channel between the planes and the security policy on the controller.

As a concrete example, consider the controller's security policy. The security policy is critical to ensuring the correct protections are implemented (*e.g.,* modifying it could result in the gateway's middleboxes not protecting the IoT devices). This code can be extracted from the controller and placed into memory isolated by the micro-hypervisor (providing P_{sw2}). Further, access to this memory is mediated by the micro-hypervisor's code white-listing (providing P_{sw3}), to ensure that only the controller's code can access the security policy. This combination prohibits an attacker in control of the OS from accessing and modifying hypervisor protected pieces of code, without requiring significant changes to existing code.

6 Secure and Efficient Communication

Our security architecture requires trust guarantees on both the control channel (between controller and gateway, P_{comm1}) and the data channel (along the gateway's packet processing path, P_{comm2}). We leverage the micro-hypervisor to provide isolation and mediation to secure these communication channels.

Secure Control Channel: It is crucial that communication between the control and data plane can be trusted as these messages often impact the security protections provided by the gateway. We look to bolster the guarantees provided by traditional tunneling (*e.g.,* IPsec/TLS) between the controller and the gateway to ensure a compromised controller or gateway cannot send spoofed messages over the tunnel (*e.g.,* malicious middlebox configuration commands). To protect these communications (P_{comm1}), we leverage a trusted agent pair running in the micro-hypervisor to mediate these communications (*e.g.,* access the secret keys required to send data over this channel). There is an agent on the controller and a corresponding agent on the gateway, which together are responsible for mediating access to the secure channel.

Secure Data Channel: On the data plane, the security protections are dependent upon each packet being processed by the correct middlebox. We can build on prior work on routing path verification (*e.g.,* control plane [27], data plane [32]), to provide per-hop guarantees with a low performance overhead. Specifically, our goal is to guarantee that packets follow the correct path and are processed by the correct sequence of middleboxes on the data plane (P_{comm2}). While traditional tunnels could be established between each middlebox and the vSwitch, this would result in significant overhead and processing delays. To protect the data channel, we propose leveraging the micro-hypervisor to enforce the correct path (*i.e.,* middlebox chain) for each packet. We achieve this by having the micro-hypervisor sign and verify each packet along its processing path, dropping packets that fail verification. Our approach differs from prior per-hop authentication proposals (*e.g.,* [27, 32]) as packets remain on a single host where a hypervisor can maintain secret keys.

These digital signatures create a connection between the raw packet data and the middlebox processing the packet, by the secret key (protected by the micro-hypervisor) shared between the vSwitch and each middlebox. Furthermore, the digital signatures can be trusted, as the secret keys are kept in isolated memory only accessible by the micro-hypervisor's mediation, stopping an attacker from forging signed packets.

7 Preliminary Implementation

Our initial results are promising towards realizing our vision. We used the uberXMHF [63] open-source micro-hypervisor that supports both x86 and ARM platforms. For our prototype, we used the Raspberry Pi 3 platform running uberXMHF, Raspbian Jessie (Linux 4.4.y) and Open Virtual Switch and Snort on the gateway (additional details in Appendix A). A preliminary policy protection extension to the micro-hypervisor (§5) resulted in a latency increase of only 1.1 ms (13%) for the controller to process a state change. Similarly, a proof of concept packet signing extension (§6) created a latency increase of 4 ms (17%) for HTTP GET requests to an IoT device. Such latency increases compare favorably with existing hardware-centric approaches (*e.g.,* systems relying on SGX) that reduce performance by up to 30% [52, 66].

8 Related Work

Trusted Computing: We leverage prior works on trusted computing to create a practical architecture for trusting edge IoT security gateways (*e.g.,* [24, 72, 73]). Hypervisors have been used to provide security primitives such as isolation, mediation, and attestation [28, 30, 41, 57, 63, 64]. A primary use of TPMs is providing remote attestation [6], leading to multiple software implementations [42, 63]. Secure routing proposals have used signatures to verify packet paths [27, 32].

SDN Security: Researchers have focused on mediating controller applications, adding permissions [20,47,49,67], and by ensuring consistency of routing rules [19,38]. Our work looks to support these controllers and provide the ability to provide increased trust in their operations. Others have looked to ensure consistency between the control and data planes with respect to packet routing, creating tools for identifying forwarding anomalies [1, 10, 15, 16, 21–23, 40, 55, 56, 74, 75] and SDN-specific attacks [14, 26, 44, 69]. Unfortunately, none of these provide runtime protections against our threat model.

9 Conclusions

In this paper, we described our overarching vision for enabling a trusted IoT security gateway architecture that is practical and deployable on today's edge networks. We argued that a micro-hypervisor based approach provides robust trust properties while remaining performant and preserving platform generality. Our preliminary implementation on a Raspberry Pi 3 has provided encouraging results with acceptable operational latency. We are currently working on a full end-to-end implementation and evaluation of our security architecture.

10 Discussion Topics

Desired Feedback and Discussion Type: We envision this paper generating discussion about "bolt-on" security architectures and recent activity within industry towards employing such an approach for securing edge networks. On this theme, we anticipate discussion about adversary capabilities that a security gateway will need to defend against as well as the trust properties it must provide. Are the necessary foundational security properties described in our approach sufficient? Are there some security properties that could be traded for increased performance?

Controversial Points:

- With the push towards end-to-end encryption, are security gateways practical if they cannot decrypt network data? Would our *trusted* gateway architecture enable adoption of recent proposals where middleboxes intercept and decrypt TLS packets (*e.g.,* TLS-RaR [68], mbTLS [33])?
- Providing isolated, device-specific middleboxes for each IoT device creates a scalability challenge. Is the isolation worth the increased resource utilization and system complexity? Are there scenarios where this is not going to work and cannot be deployed?
- While users want secure IoT devices, would users go to the trouble of deploying/managing a security gateway? Is it too complex for home users? Can it be integrated with existing enterprise network security practices?
- While micro-hypervisors have been demonstrated on a variety of platforms, does our architecture provide sufficient coverage for edge security gateway platforms? Could it be deployed on existing, off-the-shelf home routers?

Open Issues:

- How many IoT devices and gateways can our architecture support? What types of security policies would the controller implement? What are the scalability bottlenecks?
- How does our architecture provide security protections to devices sending/receiving encrypted data? Is there a limitation on the types of edge devices the gateway could protect?
- While the architecture is general enough to support a spectrum of users from home to enterprise, what actions are needed for it to be usable?
- What actions are required by the end-users? How do they mitigate false positives or overly protective policies?
- Many IoT security gateways protect IP-based protocols. How can gateways be extended to support other protocols used by IoT devices (*e.g.,* BLE, Zigbee)?

Circumstances When the Idea Might Fall Apart:

- If the security gateway hardware and/or the micro-hypervisor is compromised by the attacker, it undermines the root-of-trust our approach relies on for providing isolation, mediation, and attestation.
- If an adversary modifies a packet's path (*e.g.,* WiFi spoofing), they could prevent packets from reaching the gateway.
- If the edge device requires extremely low latency, it might be infeasible to integrate and protect via our approach.

Acknowledgments

We thank our shepherd, Ziming Zhao, for his help with the final version of this paper, as well as the anonymous reviewers for their detailed comments. This work was supported in part by NSF award CNS-1564009. This work was also supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. This research was also supported by the Carnegie Mellon University (CMU) Manufacturing Futures Initiative, made possible by the Richard King Mellon Foundation. Additionally, we would like to thank the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense, for support and guidance.

References

[1] Ehab Al-Shaer and Saeed Al-Haj. FlowChecker: Configuration Analysis and Verification of Federated OpenFlow Infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, 2010.

[2] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE S&P*.

[3] Mahmoud Ammar, Bruno Crispo, Bart Jacobs, Danny Hughes, and Wilfried Daniels. Sμv - the security microvisor: A formally-verified software-based security architecture for the internet of things. *IEEE Trans. Dependable Sec. Comput.*, 16(5):885–901, 2019.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, 2017. USENIX Association.

[5] Will Arthur and David Challener. *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress, 2015.

[6] Will Arthur, David Challener, and Kenneth Goldman. *History of the TPM*. Apress, Berkeley, CA, 2015.

[7] David Barrera, Ian Molloy, and Heqing Huang. Standardizing iot network security policy enforcement. In *DISS 2018*.

[8] Sofia Belikovetsky, Mark Yampolskiy, Jinghui Toh, Jacob Gatlin, and Yuval Elovici. dr0wned – cyber-physical attack with additive manufacturing. In *11th USENIX Workshop on Offensive Technology (WOOT 17)*, Vancouver, BC, 2017. USENIX Association.

[9] Bit defender box 2. https://www.bitdefender.com/box/, 2018.

[10] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo, and Chin-Laung Lei. How to Detect a Compromised SDN Switch. In *NetSoft*. IEEE, 2015.

[11] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10(2), 2011.

[12] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086), 2016.

[13] Cujo. https://www.getcujo.com, 2018. Accessed: 2018-03-23.

[14] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *Ndss*, volume 15, 2015.

[15] Mihai Dobrescu and Katerina Argyraki. Software dataplane verification. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014.

[16] Alex Horn, Ali Kheradmand, and Mukul Prasad. Deltanet: Real-time network verification using atoms. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 735–749, 2017.

[17] X. Hu and Z. M. Mao. Accurate real-time identification of ip prefix hijacking. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 3–17, 2007.

[18] Intel. Intel software guard extensions: Developer guide. https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf, 2016.

[19] Xin Jin, Jennifer Gossels, Jennifer Rexford, and David Walker. Covisor: A compositional hypervisor for software-defined networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015.

[20] Heedo Kang, Seungwon Shin, Vinod Yegneswaran, Shalini Ghosh, and Phillip Porras. Aegis: An automated permission generation and verification system for sdns. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges*, 2018.

[21] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real Time Network Policy Checking Using Header Space Analysis. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX, 2013.

[22] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P Brighten Godfrey. Veriflow: Verifying network-wide Invariants in Real Time. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX, 2013.

[23] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. Kinetic: Verifiable Dynamic Network Control. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX, 2015.

[24] Ronny Ko and James Mickens. Deadbolt: Securing iot deployments. In *Proceedings of the Applied Networking Research Workshop*, pages 50–57, 2018.

[25] Eliot Lear, Ralph Droms, and Dan Romascanu. Manufacturer usage description specification. *IETF draft*, 2017.

[26] Seungsoo Lee, Changhoon Yoon, Chanhee Lee, Seungwon Shin, Vinod Yegneswaran, and Phillip A Porras. Delta: A security assessment framework for software-defined networks. In *NDSS*, 2017.

[27] Matt Lepinski and Kotikalapudi Sriram. Bgpsec protocol specification. *Draft-ietf-sidr-bgpsecprotocol*, 2013.

[28] Lionel Litty, H Andrés Lagar-Cavilla, and David Lie. Hypervisor support for identifying covertly executing binaries. In *USENIX Security Symposium*, 2008.

[29] Lee Mathews. Boeing is the latest wannacry ransomware victim. https://www.forbes.com/sites/leemathews/2018/03/30/boeing-is-the-latest-wannacry-ransomware-victim/#9b1382d66344, 2018. Accessed: 2018-11-19.

[30] Jonathan M McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. Trustvisor: Efficient TCB reduction and attestation. In *2010 IEEE Symposium on Security and Privacy*, 2010.

[31] MITRE. Cve-2017-5691. https://nvd.nist.gov/vuln/detail/CVE-2017-5691, 2017.

[32] Jad Naous, Michael Walfish, Antonio Nicolosi, David Mazières, Michael Miller, and Arun Seehra. Verifying and enforcing network paths with icing. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[33] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. And then there were more: Secure communication for more than two parties. In *Proceedings of the 13th International Conference on*

emerging Networking EXperiments and Technologies. ACM, 2017.

[34] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. Everything you should know about intel sgx performance on virtualized systems. *POMACS*, 3(1), 2019.

[35] Phil Oester. Linux kernel memory subsystem copy on write mechanism contains a race condition vulnerability. https://www.kb.cert.org/vuls/id/243144/, 2016. Accessed: 2020-02-14.

[36] Patrick Howell O'Neill. Russian hackers are infiltrating companies via the office printer. https://www.technologyreview.com/f/614062/russian-hackers-fancy-bear-strontium-infiltrate-iot-networks-microsoft-report/, 2019.

[37] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Safebricks: Shielding network functions in the cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[38] Phillip A Porras, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. Securing the software defined network control layer. In *NDSS*, 2015.

[39] J.M. PORUP. How hacking team got hacked. https://arstechnica.com/information-technology/2016/04/how-hacking-team-got-hacked-phineas-phisher/, 2016. Accessed: 2019.

[40] Santhosh Prabhu, Gohar Irfan Chaudhry, Brighten Godfrey, and Matthew Caesar. High-coverage testing of softwarized networks. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges*, pages 46–52, 2018.

[41] Daniel Quist, Lorie Liebrock, and Joshua Neil. Improving antivirus accuracy with hypervisor assisted analysis. *Journal in computer virology*, 7(2):121–131, 2011.

[42] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, Magnus Nystrom, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. ftpm: A software-only implementation of a TPM chip. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 841–856, Austin, TX, August 2016. USENIX Association.

[43] Rattrap. https://www.myrattrap.com, 2018. Accessed: 2018-03-23.

[44] Christian Röpke and Thosten Holz. Preventing malicious sdn applications from hiding adverse network manipulations. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges*, pages 40–45, 2018.

[45] J. M. Rushby and B. Randell. A distributed secure system. In *1983 IEEE Symposium on Security and Privacy*, pages 127–127, April 1983.

[46] Alex Schiffer. How a fish tank helped hack a casino. https://www.washingtonpost.com/news/innovations/wp/2017/07/21/how-a-fish-tank-helped-hack-a-casino/, 2017. Accessed: 2019.

[47] Sandra Scott-Hayward. Design and Deployment of Secure, Robust, and Resilient SDN Controllers. In *NetSoft*. IEEE, 2015.

[48] Xingang Shi, Yang Xiang, Zhiliang Wang, Xia Yin, and Jianping Wu. Detecting prefix hijackings in the internet with argus. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, page 15–28, New York, NY, USA, 2012. Association for Computing Machinery.

[49] Seungwon Shin, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 78–89, 2014.

[50] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, et al. Bitvisor: a thin hypervisor for enforcing i/o device security. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 121–130, 2009.

[51] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016.

[52] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. Panoply: Low-tcb linux applications with sgx enclaves. In *NDSS*, 2017.

[53] Adam Shostack. Experiences threat modeling at microsoft. *Workshop on modeling security (ModSec)*, 2008.

[54] Anna Kornfeld Simpson, Franziska Roesner, and Tadayoshi Kohno. Securing vulnerable home iot devices

with an in-hub security manager. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*.

[55] Sooel Son, Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Model Checking Invariant Security Properties in OpenFlow. In *ICC*. IEEE, 2013.

[56] Aisha Syed, Bilal Anwer, Vijay Gopalakrishnan, and Jacobus Van der Merwe. Depo: A platform for safe deployment of policy in a software defined infrastructure. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 98–111, 2019.

[57] Richard Ta-Min, Lionel Litty, and David Lie. Splitting interfaces: Making trust between applications and operating systems configurable. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 279–292, 2006.

[58] Hendrik Tews, Tjark Weber, Marcus Völp, Erik Poll, MCJD van Eekelen, and PJB van Rossum. Nova micro–hypervisor verification. 2008.

[59] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. Shieldbox: Secure middleboxes using shielded execution. In *Proceedings of the Symposium on SDN Research*, 2018.

[60] Amit Vasudevan. *Practical Security Properties on Commodity Computing Platforms - The uber eXtensible Micro-Hypervisor Framework*. Springer Briefs in Computer Science. Springer, 2019.

[61] Amit Vasudevan and Sagar Chaki. Have your pi and eat it too: Practical security on a low-cost ubiquitous computing platform. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018.

[62] Amit Vasudevan, Sagar Chaki, Limin Jia, Jonathan McCune, James Newsome, and Anupam Datta. Design, implementation and verification of an extensible and modular hypervisor framework. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013.

[63] Amit Vasudevan, Sagar Chaki, Petros Maniatis, Limin Jia, and Anupam Datta. überspark: Enforcing verifiable object abstractions for automated compositional security analysis of a hypervisor. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 87–104, 2016.

[64] Amit Vasudevan, Bryan Parno, Ning Qu, Virgil D Gligor, and Adrian Perrig. Lockdown: Towards a safe and practical architecture for security applications on commodity platforms. In *International Conference on Trust and Trustworthy Computing*, pages 34–54. Springer, 2012.

[65] Sébastien Vaucher, Rafael Pires, Pascal Felber, Marcelo Pasin, Valerio Schiavoni, and Christof Fetzer. Sgx-aware container orchestration for heterogeneous clusters. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

[66] Nico Weichbrodt, Pierre-Louis Aublin, and Rüdiger Kapitza. sgx-perf: A performance analysis tool for intel sgx enclaves. In *Proceedings of the 19th International Middleware Conference*, pages 201–213, 2018.

[67] Xitao Wen, Yan Chen, Chengchen Hu, Chao Shi, and Yi Wang. Towards a Secure Controller Platform for OpenFlow Applications. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013.

[68] Judson Wilson, Riad S Wahby, Henry Corrigan-Gibbs, Dan Boneh, Philip Levis, and Keith Winstein. Trust but verify: Auditing the secure internet of things. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017.

[69] Yang Wu, Ang Chen, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. Automated bug removal for software-defined networks. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 719–733, 2017.

[70] Wen Xu, Juanru Li, Junliang Shu, Wenbo Yang, Tianyi Xie, Yuanyuan Zhang, and Dawu Gu. From collision to exploitation: Unleashing use-after-free vulnerabilities in linux kernel. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[71] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015.

[72] Tianlong Yu, Seyed Kaveh Fayaz, Michael P Collins, Vyas Sekar, and Srinivasan Seshan. Psi: Precise security instrumentation for enterprise networks. In *NDSS*, 2017.

[73] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, 2015.

[74] Peng Zhang, Hao Li, Chengchen Hu, Liujia Hu, Lei Xiong, Ruilong Wang, and Yuemei Zhang. Mind the gap: Monitoring the control-data plane consistency in software defined networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 19–33, 2016.

[75] Peng Zhang, Shimin Xu, Zuoru Yang, Hao Li, Qi Li, Huanzhao Wang, and Chengchen Hu. Foces: Detecting forwarding anomalies in software defined networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

[76] Wei Zhang, Abhigyan Sharma, Kaustubh Joshi, and Timothy Wood. Hardware-assisted isolation in a multi-tenant function-based dataplane. In *Proceedings of the Symposium on SDN Research*, pages 1–7, 2018.

A Additional Implementation Details

A high-level overview of the software stack of our premilinary implementation is shown in Figure 3 where the hypervisor extension is depicted as a green rectangle (labelled 'Packet Signing'). The blue ovals represent hypercalls to the hypervisor extension, added to the commodity software to perform a hypervisor protected operation (*e.g.,* creating/verifying a digital signature for a packet).

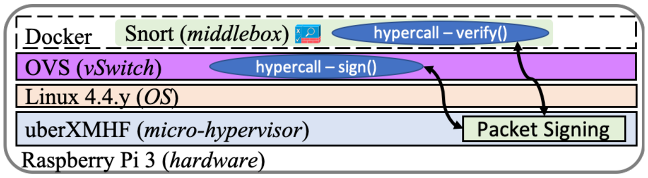

Figure 3: High-level view of our architecture's stack on a proof of concept data plane implementation, where the 'Packet Signing' rectangle is a hypervisor extension and the arrows are hypercalls to the hypervisor extension.

Virtual Trusted Platform Module (vTPM): Our architecture looks to leverage a hypervisor-enabled vTPM in order to provide periodic, remote attestation. A key motivation in the development of Trusted Platform Modules (TPM) was to provide the ability to remotely attest the health of a system's boot sequence [6, 11] We leverage a subset of the features provided by the TPM standard [5] to provide our trust properties, specifically PCR extend and PCR quote. In addition to a vTPM not requiring additional hardware, it provides two key benefits over a physical TPM, increased storage and increased performance. The measurement storage (*i.e.,* PCRs) on a vTPM is based upon the platform's memory region protected by the micro-hypervisor, allowing for a large number of separate measurements (*e.g.,* >200 PCRs on a single 4 KB page). Furthermore, these measurements are not limited by the data rate of a system bus (*e.g.,* SPI on a Raspberry Pi 3), reducing access latency by over 20x. A challenge for software-based TPMs is preserving secrets across reboots. Our vTPM can leverage existing platform non-volatile memory in order to preserve secrets across boots (*e.g.,* the Raspberry Pi 3 has a boot NVRAM that can act as a long-term secure storage [61]).

Qualitative Comparison to Trusted Hardware: As an alternative to our approach, others have looked to trusted hardware. On cloud data planes, hardware enclaves have been used to protect middleboxes (some bolstered by programming language guarantees) [37, 59, 65]. These approaches require specific hardware (*e.g.,* SGX), subjecting running applications to memory size limitations and requiring the middleboxes be reimplemented in advanced programming languages to achieve mediation. This contrasts with our vision that is broadly deployable on multiple hardware platforms running commodity software potentially without modification.