Alcatraz: Data Exfiltration-Resilient Corporate Network Architecture

Daniele E. Asoni
ETH Zürich
Email: daniele.asoni@inf.ethz.ch

Takayuki Sasaki
NEC Corporation
Email: t-sasaki@fb.jp.nec.com

Adrian Perrig
ETH Zürich
Email: adrian.perrig@inf.ethz.ch

***Abstract*—In Advanced Persistent Threats (APTs), an adversary targets network components such as switches and middle boxes as well as end hosts to exfiltrate sensitive information out of the network. We propose Alcatraz, a new corporate network architecture to prevent data exfiltration. Alcatraz ensures path integrity, packet integrity, and packet confidentiality to prevent a malicious network component from extracting, altering, or maliciously forwarding any network packet.**

Alcatraz leverages Trusted Execution Environments (TEE) created by Intel SGX to protect modules providing these security properties. To achieve exfiltration resilience, our architecture ensures that sensitive information is only processed within a TEE, from the sender to the receiver and along all network nodes. Although our architecture requires many changes, it explores the design space of what level of security can be achieved today with commodity hardware. Through our software switch implementation, we demonstrate that the performance is already viable for a corporate environment with high security requirements. Our results suggest that an optimized hardware implementation could satisfy also higher performance requirements.

***Keywords*-network security; trusted computing; exfiltration resilience**

I. Introduction

Enterprise and data center networks are targets of *Advanced Persistent Threats* (APTs)—highly sophisticated and targeted attacks, which surreptitiously take control of end hosts and network nodes, and which can go undetected for years [1]. Although APTs have predominantly targeted end hosts, recently disclosed network vulnerabilities [2]–[6] suggest that adversaries will increasingly compromise network nodes because they offer an ideal vantage point to monitor traffic, to alter packets, to exfiltrate data, and to attack connected hosts. Traditional defense strategies (e.g., intrusion detection and prevention systems, firewalls) have proved insufficient to address recently disclosed attacks [1], [7]–[9].

In this paper we study how trusted computing technologies can be leveraged to build secure networks that can effectively defend data centers and enterprises against APTs, while retaining the flexibility of modern software defined networking (SDN). We assume that all end hosts and network nodes provide secure trusted execution environments (TEEs) with remote attestation capabilities, for instance by supporting Intel Software Guard eXtensions (SGX) [10], [11], a new technology that is already available on commodity Intel processors. Although a strong assumption for current networks, we expect that the next generation of network devices can be equipped with trusted computing technology. Initially, the most likely adopters of such a solution would be data centers and corporations with strong security and exfiltration-resiliency requirement, such as governmental, financial and military institutions, as well as cloud providers catering to the needs of such institutions.

We thus propose Alcatraz, a radical new approach to counter APT-based data exfiltration. Unlike previous work, Alcatraz is designed to provide security in a constructive way, rather than by trying to detect and remove intrusions. TEEs on network devices and end hosts ensure that the confidentiality and integrity of all packets is maintained even in the presence of compromised network nodes and (with some constraints) compromised end hosts by constructing a trusted logical path from end to end through each traversed TEE. At the same time, Alcatraz enables secure inspection and modification of packets within TEEs on network nodes.

APTs are complex and involve multiple aspects and layers of systems and networks, so we do not expect Alcatraz to eliminate the risk posed by them entirely. Instead, we aim to clearly identify the guarantees and limitations of our system, and we strive for a design that is sufficiently generic and flexible to allow integration with other solutions. Such a design also allows Alcatraz to be adapted to different requirements and technologies: for instance, while for our evaluation and for many examples we assume an instantiation of Alcatraz based on SGX and a standard SDN model with centralized control plane, it would be possible to adapt Alcatraz to work with another trusted computing technology, and a logically distributed control plane.

The main contributions of this paper are as follows.

- We design Alcatraz, an architecture to protect against data exfiltration by APTs. Alcatraz provides path integrity, application-bound packet integrity, and packet confidentiality. Alcatraz allows packet modifications, even for middleboxes and SDN switches.
- We implement Alcatraz using SGX, a new technology that allows the construction of lightweight and flexible TEEs. We minimize the amount of trusted software that is run within these TEEs.
- We analyze the security guarantees offered by our system, showing that the use of SGX allows us to provide strong guarantees not achieved by previous work.

II. Background

A. Intel SGX

Intel has recently proposed a new technology (a successor to the Intel Trusted eXecution Technology (TXT) [12]), which ensures runtime integrity for code and confidentiality for its data, called Intel SGX [10], [11]. This technology creates a TEE called an *enclave* on an untrusted platform. Specifically, the CPU creates an isolated memory space containing heap, stack, and program code. When the enclave is executed, SGX performs access control so that the processes outside of the enclave cannot access its contents. Moreover, SGX performs memory encryption to defend against a compromised memory bus or DRAM. Therefore, even if the BIOS, OS, and applications are compromised, the security-sensitive data (e.g., password and encryption keys) and program logic inside the enclave are protected. Since SGX runs in userspace (i.e., without kernel privileges), it can also be executed by virtual machines.

SGX also supports remote attestation. Remote attestation allows a remote verifier to ensure that the enclave is actually running on the correct CPU with the expected code and inputs. More precisely, the attestation is made by SGX by measuring the code (i.e., computing a hash over it) together with initial parameter and potentially additional data. This measurement is then cryptographically authenticated by SGX in a way that is verifiable based on an Intel certificate.

B. SDN and OpenFlow

Software Defined Networking (SDN) is a networking paradigm that enables dynamic adaptation of network configurations by software controlled by a logically centralized controller. OpenFlow [13] is a standardized protocol which realizes SDN functionalities by specifying a vendor-independent interface for network devices (switches) which can be invoked by the controller to configure and manage the devices. Generally, the first packet of a flow is forwarded by the switch to the controller with a `Packet-In` message, and the controller may then set up a suitable *flow entry* using `Flow-Mod` messages. A flow entry typically comprises a match field specifying a matching condition on incoming packets, and an action field specifying actions such as how the packets should be forwarded or modified. The OpenFlow switch has flow tables to store the flow entries, and it performs a table lookup when it receives a packet. Additionally, the controller can order a switch to send a packet using a `Packet-Out` message.

III. Problem Description

Our high-level goal is assurance of confidentiality of network traffic, as well as packet and forwarding integrity in the presence of compromised switches. We now describe more in detail what setting we assume in terms of network structure, we state our goals and requirements, and we define our threat model and limitations.

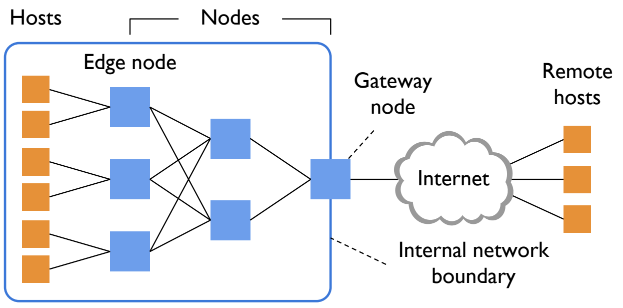


Figure 1: A possible network structure which illustrates our terminology.

A. Network setting

We consider a set of devices (programmable switches, routers, middleboxes, etc.), which we generically call *nodes*, interconnected to form a network with an arbitrary topology (however, the network graph is connected). A subset of these nodes, which we refer to as *edge nodes*, are connected to end hosts (or simply *hosts*). The network can include one or more *gateway nodes* connecting the internal corporate network to the Internet, through which *remote hosts* can be reached. This network model is shown in Figure 1. We assume that all nodes and all hosts have hardware supporting TEEs such as SGX enclaves.

This model can also capture multi-tenancy, which is common in data centers. In this case, a *host* is an end point controlled by one tenant, and can represent either a physical single-tenant end host, or a virtual machine (*VM*) assigned to a tenant, running on top of a hypervisor (SGX enclaves can be run by VMs, see Section II-A).

B. Rules

We assume that nodes forward and modify packets according to *rules*, which we intend as a generalization of OpenFlow's flow entries (see Section II-B). The rules on a node fully specify how incoming packets should be handled. In particular, when a host communicates with another host, the rules on the nodes determine the *path* that the individual packets traverse, where we define a path as the sequence made of the originating host (the *source*), all the traversed nodes, and the receiving host (the *destination*). A path may also end at an intermediate node in case a rule on that node specifies that the packet should be dropped.

As for switches in OpenFlow and for firewalls and middleboxes, we assume that an *administrator*—which in practice could be an automated set of controllers, but also more simply a human—specifies a set of rules for each node, which we call *trusted rules* of that node: as we explain in more detail in Section IV-E, these rules are authenticated by the administrator in such a way that each TEE can verify them, and can verify that they are meant for itself and not another TEE.

C. Security Goals

Our high-level security goal is exfiltration prevention. However, for reasonable definitions of the threat model there are always scenarios in which data exfiltration may still be possible even when using Alcatraz. For this reason, instead of imposing artificial constraints on the adversary's capabilities, we formulate lower level security goals of network hardening that hold under a strong adversary without unreasonable restrictions, and in our security analysis (Section V) we discuss under which circumstances exfiltration prevention can be formally guaranteed. With the assumptions stated in the sections above, these security goals are the following.

- **Packet integrity.** Packets are only modified within TEEs by approved code; TEEs only modify packets according to trusted rules.
- **Path integrity.** Packets are forwarded according to the trusted rules that apply to them, which means that they traverse the network according to the path the trusted rules determine.
- **No packet injection.** Packets can be created and sent freely only by hosts. Nodes can send packets only when authorized by their TEE, which could be either in response to a received packet (for instance for management purposes) or because the node is instructed to send a new packet by a trusted rule (see Section VI).
- **Packet confidentiality.** The contents of a packet (payload) should be accessible only at the sender, within the TEEs on the path, and at the packet's destination.

The path integrity property cannot be guaranteed unconditionally: for instance, a compromised operating system on a node may drop and/or incorrectly forward packets (see Section III-D below). Therefore our secondary goal is **fault localization**: if path integrity is violated, i.e., if a packet is dropped (or reordered[1]), the offending link can be identified by an administrator.

Fault reporting: The TEEs in Alcatraz should be able to report faults to the control plane, together with information about the faults' type. How the control plane handles these reports is outside the scope of this paper. Deciding how to react to fault reports is not a trivial problem, especially because some of them (e.g., packet dropping) could be due to non-adversarial causes. We point out, however, that even in the extreme case where faults are completely ignored Alcatraz would still provide the properties listed above, in particular packet confidentiality and integrity, and injection prevention, while malicious packet drops, modifications and re-routing would simply constitute DoS attacks.

[1]Packet dropping cannot be distinguished from reordering within a fixed amount of time, so reordering is also treated as a fault. In our implementation we adopt mechanisms to tolerate reordering within a time window.

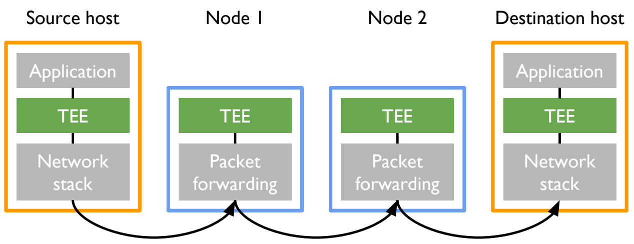


Figure 2: End-to-end architecture overview.

D. Threat Model

We consider an adversary that can compromise a set of nodes, taking full control of their operations (except for the code running within TEEs). Furthermore, we assume that the adversary can also control a set of hosts, either because of compromise or because, in a multi-tenant setting, the adversary registers as one or more tenants, controlling thus all the hosts belonging to those tenants.

However, we assume that SGX can be trusted, implying that the adversary cannot obtain the secrets contained in a TEE. For this assumption to hold the adversary cannot be allowed invasive physical access to hosts or nodes for extended periods of time, which can be achieved through physical access control and surveillance systems. The assumption also requires that the hardware implementation of SGX itself be correct, and that Intel's private signing key remain uncompromised. The issue of SGX security is orthogonal to this work (cf. Costan and Devadas [11]).

In our threat model we also assume that administrators are trusted: a malicious administrator can always install compromised keys on nodes and hosts, which would provide a trivial way to break the security goals of our system. In practice, this limitation can be easily mitigated by having multiple administrators which oversee each other's operations, and by enforcing separation of duty where possible.

IV. Alcatraz Architecture

We first provide a high level overview of Alcatraz, and afterwards we explain the details of host and node setup, of packet sending and processing, and of the reporting of detected faults.

A. Architecture overview

With our network model defined in Section III-A, we consider a source host communicating with a destination host through a sequence of nodes. As an example we consider the path for this communication as depicted in Figure 2, with two intermediate nodes. Hosts and nodes run TEEs (SGX enclaves) which store shared keys that allow secure communication with neighboring TEEs. The TEEs also store encryption keys that allow encryption and decryption of packet payloads.

When the source S wants to send a packet p, S first passes p to its local TEE, which encrypts p using a key shared among all TEEs for confidentiality, and then adds a

Message Authentication Code (MAC) using the key shared with the first node's TEE, to guarantee that the packet was indeed generated on that host. Once the enclave returns the encrypted and authenticated packet, the source sends it to the first switch. When a switch receives a packet, it looks up what rule applies to the packet, and then passes the packet and the rule to the local TEE, together with the information about which host or node the packet came from. The TEE first verifies that the packet is authentic by checking its MAC with the key shared with the previous hop (host or node). It then checks that the rule is *trusted* (see Section III-C, concretely this means verifying an authentication tag attached to the rule, generated with a trusted key by the administrator). The TEE then verifies that the rule matches the given packet, and if all checks hold then it applies the rule to the packet. Through the rule the TEE learns which host or node the packet should be forwarded to (unless the packet is to be dropped), and computes a new MAC over the packet with the key shared with the next hop. When the packet reaches the destination, it is passed to the destination's TEE which checks the packet integrity and decrypts the packet.

In addition to these steps, nodes and hosts also use counters in a way that allows detection of packet replay and dropping. We describe the details of this process in Section IV-C. If any fault is detected, e.g., a packet with an incorrect MAC is received, then depending on the network different actions can be taken, such as secure logging of the incident, or immediate communication to an SDN controller.

B. Setup and key management

To setup a node or host, the administrator establishes a TEE on it, and verifies it through attestation. After successful attestation the administrator provides the TEE with the keying material required for its operation, and with a list of neighboring devices. After the setup, the device can begin to operate.

Attestation: At creation, the TEE is given the long-term public key of the administrator K_{admin}, a short-term public key K_{setup} created by the administrator for the setup, and a new identifier ID_{dev} (unique in the network). The TEE generates a public-private key pair K_{dev}, K_{dev}^{-1}, and then produces an attestation to the administrator which includes its own public key K_{dev} and the two public keys of the administrator: $\mathtt{attest}(\mathit{enclave\_code}, K_{dev}, K_{admin}, K_{setup}, \mathrm{ID}_{dev})$.

Once the administrator has verified the attestation, she has the guarantee that an enclave is running on the device with the expected code, and that this enclave has generated the public key K_{dev}. The administrator can thus establish a secure channel to the enclave (using the enclave's public key and her own temporary private key), and through this secure channel she sends a master key MK (which is the same for all devices)[2] and a neighbor list to the enclave.

[2]This design choice helps minimizing the trusted computing base.

Key derivation: The enclave uses the master key MK as input to a key derivation function kdf to generate a set of keys. These are the *universal control-plane authentication key*, used by the administrator and the switches to make universal claims such as neighboring statements (see below), defined as $\mathsf{CAK} = \mathit{kdf}(MK, \text{“CAK”})$; the *TEE-specific control plane authentication key* relative to a device x, used by the administrator to authenticate device-specific rules, defined as $\mathsf{CAK}_x = \mathit{kdf}(MK, \text{“CAK”}, \mathrm{ID}_x)$; the *data-plane confidentiality key*, used to encrypt and decrypt the contents of data packets in end-to-end communications, defined as $\mathsf{DCK} = \mathit{kdf}(MK, \text{“DCK”})$; and the *data-plane authentication key*, used for authentication of data packets on individual links (see Section IV-C1), defined as follows (for any pair of devices x, y):

$$\mathsf{DAK}_{x,y} = \begin{cases} \mathit{kdf}(MK, \text{“DAK”}, \mathrm{ID}_x, \mathrm{ID}_y), & \text{if } \mathrm{ID}_x > \mathrm{ID}_y \\ \mathit{kdf}(MK, \text{“DAK”}, \mathrm{ID}_y, \mathrm{ID}_x), & \text{if } \mathrm{ID}_x < \mathrm{ID}_y \end{cases}$$

The latter is specific to a pair of neighboring devices x and y, and is computed using their identifiers. Note that according to the definition, $\mathsf{DAK}_{x,y} = \mathsf{DAK}_{y,x}$.

Neighbor list: The neighbor list consists of a set of *neighboring statements*, which are device identifier pairs authenticated by the administrator using the CAK key. Once set up, the device's TEE forwards each neighboring statement to the respective neighbor: this allows dynamic and automatic additions of devices to a network without the need of manually updating the neighbor list of all neighboring devices.

1) SGX-protected master key management: Given the sensitivity of the master key MK, we design an `admin TEE` to carry out the task of generating and distributing MK. In particular, the admin TEE (which should run on a single-purpose computer used by the administrator) will provide MK to TEEs on nodes and hosts only after successfully verifying their integrity through attestation.

Externally the correctness of the entire process can be verified in two steps. First the verifier requests an attestation of the admin TEE, which includes a value v derived from the master key as follows:

$$v = h(\mathit{kdf}(MK, \text{“verification”})) \qquad \text{(IV.1)}$$

Then the verifier can check any device by requesting an attestation from it including value v (which the device can generate). If both checks are verified then the device is running a TEE with the correct code and is using a master key which was securely generated inside an admin TEE (also running the correct code) and has only been shared with other correct TEEs.

C. TEE components

Every TEE, on both hosts and nodes, contains a *monitoring module*, which is responsible for the communication with neighboring TEEs, and in particular for ensuring that communications are correctly authenticated. Additionally,

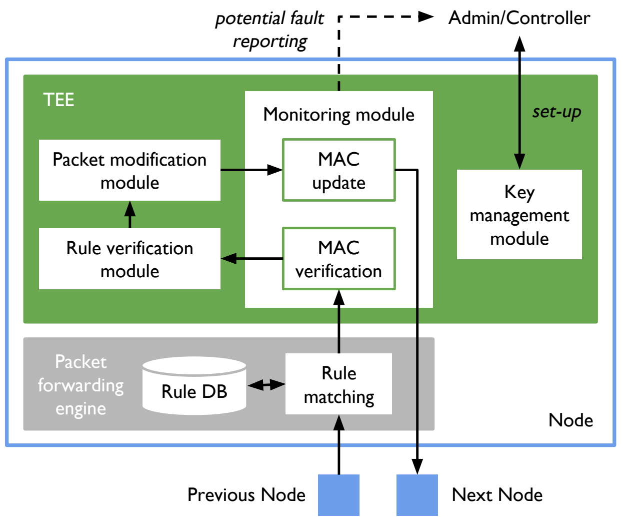


Figure 3: Node structure showing the way packets are processed sequentially by the TEE modules.

TEEs on nodes (see Figure 3) include a *rule verification module* to verify that provided rules are trusted and that they apply to the given data packet, and a *packet modification module* to modify packets according to the rules if required. All TEEs also include a *key management module*, which carries out the setup operations described in Section IV-B.

1) Monitoring module: A monitoring module stores two monotonic counters for each neighbor x: one for incoming packets from that neighbor ($\mathcal{N}_x^r$), and one for outgoing packets to that neighbor ($\mathcal{N}_x^s$). These counters are initially set to 0. When a TEE y sends a packet to a neighbor x, its monitoring module increases the current send counter for neighbor x ($\mathcal{N}_x^s := \mathcal{N}_x^s + 1$) and appends it to the packet, and computes a MAC over the packet and the counter. The receiving TEE x verifies the MAC and then checks that the counter in the packet $\mathcal{N}_{pkt}$ is greater than the current receive counter for its neighbor y ($\mathcal{N}_y^r$). If not, i.e., if $\mathcal{N}_{pkt} \leq \mathcal{N}_y^r$, then the packet is considered a replay and is dropped, and a fault is reported for that neighbor. If the counter in the packet is greater by one ($\mathcal{N}_{pkt} = \mathcal{N}_y^r + 1$), then the packet is regular and the local counter is increased by one. If the counter is greater by more than one ($\mathcal{N}_{pkt} > \mathcal{N}_y^r + 1$), a dropping or reordering fault is reported, but the packet is still considered valid and the local counter is updated to match the counter in the packet ($\mathcal{N}_y^r := \mathcal{N}_{pkt}$). We present the details of fault reporting in Section IV-E2.

2) Rule verification module: For higher performance and to minimize the trusted computing base (TCB) we keep the rule database and the rule matching functionality of nodes outside the TEE (hosts do not need these components since they do not forward nor modify packets). To ensure that packet and path integrity are guaranteed nonetheless, the TEE contains a rule verification module which checks whether the rules provided together with an incoming packet are *trusted* (authenticated with the TEE-specific authentication key CAK$_x$), and whether they actually apply to the packet.

In our design we assume that the rule database has been created to contain only non-overlapping rules (i.e., only a single rule can match a packet). While this process increases the size of the database, the size will be upper-bounded by the number of flows, which is supported by today's devices: indeed, in SDN per-flow rules are common in network applications such as load balancing, where the controller has to decide on the routing of each flow.

3) Packet modification module: A rule may require a node x to modify a packet. This modification is carried out by the TEE using the packet modification module. Once the rule verification module has checked that a given rule is trusted (i.e., correctly authenticated with key CAK$_x$) and that it applies to the given packet, if the rule requires a packet modification then the rule and the packet are passed to the packet modification module, which updates the packet and returns it to the monitoring module. The monitoring module then re-authenticates the packet, thus guaranteeing that the modification was carried out within the protected environment of the TEE, and then forwards the packet to the next hop.

4) Differences for host TEE structure: The TEEs on hosts do not strictly need the rule verification module nor the packet modification module, since they do not need to change/forward packets. However, a rule verification module might still be included to enforce certain host-based rules that, e.g., restrict sending rate or the allowed set of destinations. While these restrictions could also be enforced on the edge node to which the host is connected, applying them on the hosts reduces the computational load on the edge nodes.

Furthermore, the TEEs on hosts need a confidentiality module, which is responsible for encrypting outgoing packets and decrypting incoming packets. TEEs on hosts can also include *deep packet inspection* (DPI) functionalities, which in Alcatraz do not need to be centralized. This allows content inspection right at the source, reducing the overhead imposed on the network. With the use of TEEs, DPI becomes straightforward, and does not require complex cryptographic constructions used by previous proposals [14].

D. Data-plane operations

Packet sending: To send a packet, a source host x passes the packet to the local TEE. The TEE proceeds to encrypt the packet using key DCK and an initialization vector (nonce) constructed as the concatenation of the identifier of the sending TEE and a monotonic counter. The construction of the initialization vector guarantees that although all hosts within the network use the same key for encryption no two packets are ever encrypted with the same initialization vector. After packet encryption the host's TEE adds the packet counter corresponding to the next hop (see Section IV-C1), authenticates the packet with the key DAK$_{xy}$ shared with the edge node y to which the host is connected, and returns the result, which the host can send out to the next hop (i.e., the first node on the path).

Packet forwarding: When a node receives a packet, it looks up what rule should be applied to it, then passes the packet together with the matching rule and the information about what neighbor sent the packet to the local TEE. The TEE processes the packet as described in Section IV-C, and then returns the potentially modified packet with updated counter and MAC together with the next-hop information (unless the rule specifies that the packet is to be dropped, in which case the TEE returns a notification accordingly).

Packet receiving: When the destination host receives a packet, it passes it to its local TEE which verifies the packet's integrity in the monitoring module analogously to the processing done by the intermediate nodes. If the packet is considered valid, it is decrypted using key DCK and returned to the application.

E. *Control-plane operations*

Control-plane messages sent by the administrator (or also by the controller, in the SDN case) include a global sequence number to prevent replay attacks. During the TEE initialization (see Section IV-B) the TEE stores the current sequence number, and later with every valid control-plane message received it updates the sequence number. If the TEE receives a message with a sequence number smaller or equal to the locally stored one, it discards the message and reports a fault (see below).

When receiving control-plane messages (e.g., new neighboring statements or neighbor revocations), TEEs are expected to reply with an acknowledgment containing the hash of the original message. The sequence numbers included in the original message protect the acknowledgments to control-plane messages sent by the administrator against replay attacks. To achieve replay protection for control messages sent by TEEs on nodes and hosts, these TEEs add a nonce to the messages. Acknowledgments are authenticated with a MAC using a TEE-specific authentication key CAK_x.

One important use of key CAK_x is for the authentication of rules by the administrator, and also for the issuance of commands to revoke rules. Because of the authentication of these rules and commands through message authentication codes, the TEE x will only accept valid rules and commands, and thus always store the correct set of *trusted rules* (see Section III-B). Thanks to the acknowledgments sent by the TEE, the administrator is able to ensure that the rules are installed/removed correctly.

1) Neighbor synchronization: At regular intervals, TEEs on nodes and hosts send out synchronization request messages to their neighbors. When a TEE x receives such a message from a neighbor y, x is expected to reply with a message containing $\mathcal{N}_y^s$, the current value of the counter of packets sent from x to y. This reply message is also an acknowledgment to the synchronization request message, i.e., it contains a hash of the request (see above). The reply allows y to ensure that it has been receiving the most recent messages sent by x. Since some messages may still be in transit, the reported counter is not expected to exactly match y's local counter $\mathcal{N}_x^r$, and instead a configurable tolerance threshold α is set as the maximum allowed drift. If the drift is higher than α, a fault is reported. Otherwise, if the counters do not match exactly ($\mathcal{N}_x^r < \mathcal{N}_y^s$), then the value of the neighbor's counter, $\mathcal{N}_y^s$ is stored, and at the next synchronization node y ensures that the local counter has reached that value, or else a fault is reported.

The process of synchronization described so far hides one complication, which is the fact that in a compromised environment the TEE does not have access to reliable timing information. The solution to this problem is for a TEE to count the number of operations it performs (e.g., sending and receiving packets) and use this count as a lower bound on the amount of time passed. Another factor that is used as a lower bound for timing estimation are synchronization request messages sent by neighbors. Note that this is done *in addition* to considering the timing provided by the device on which the TEE is running, i.e., on an honest device the TEE sends out synchronization requests regularly, even if no packets are sent or received.

2) Fault reporting: When a TEE detects a fault (e.g., data packet replay, data packet dropping/reordering, missing acknowledgment for a control packet) it can handle this exceptional condition in various ways, depending on the concrete system and its requirements. The best option we envision, viable in particular for software defined networking, is that the faults are directly reported to the *controller* (see Section VI), which can take immediate action such as the rerouting of flows in order to avoid a faulty link. Another simple alternative solution is local logging, which thanks to SGX can be done in a tamper-evident manner (in an analogous way as what is done for instance by Parno et al. [15]). The administrator could regularly collect the logs from all devices and check them for reported faults: if a fault is found, it may be investigated, and the possibly compromised or defective devices reset or replaced.[3]

V. Security Analysis

We now analyze the security of our system to show that our security goals outlined in Section III-C are satisfied by our architecture. To this end we first show that the secret keying material used by the TEEs is never leaked outside the TEEs, then we investigate what properties the system achieves in terms of protection against denial of service (DoS) attacks, and finally we show, based on these properties, that our security goals are met. Finally, we show in what circumstances achieving the security goals also implies protection against data exfiltration.

[3]If it is necessary to reset the TEE on a device, the new TEE will generate a new key and thus it will have a new identifier, requiring an update to the neighbor list.

A. Keys and cryptographic operations

Once the integrity of the TEEs of devices and of the admin TEE has been verified, we have the guarantee that the master key MK and the keys derived from it are never sent out, and that they are used only in a cryptographically sound way. Two keys deserve to be analyzed with more detail. First, the data-plane confidentiality key DCK is used by multiple hosts to encrypt, so to be secure typically a stream cipher would be used with an initialization vector (nonce) that is never used to encrypt two different packets. We ensure this by constructing this nonce out of the ID of the host and a monotonic counter provided by SGX which cannot be reset. Second, the verification key ($kdf(MK, \text{“verification”})$, see Section IV-B1) may be leaked if the hash function used to compute v in Eq. (IV.1) were not secure. However, this key is never used for other operations, and assuming that the key derivation function kdf is cryptographically secure, the key can be considered independent from the master key and the other keys from it.

Other processes and physical security measures should be in place to prevent the possibility that an adversary may get physical access to the hardware of the network devices and the machine running the admin TEE, especially not for extended periods of time and outside the premises of the enterprise. This is, however, outside the scope of this paper.

B. Synchronization and DoS

We described neighbor synchronization in Section IV-E1. The following Lemma shows the main property it achieves.

Lemma V.1 (Synchronization)**.** *As long as at least one device (host or node) in the network is honest, synchronization messages are sent regularly between every pair of neighboring TEEs, or else a fault is reported.*

Proof: An honest device has correct timing information and thus sends out synchronization requests regularly to its neighbors. If any neighbor does not reply with a valid acknowledgment, the device reports a fault. If no fault is reported, all neighbors of the honest device receive regular synchronization messages, and since these are used as lower bound on the timing, it follows that all neighbors will have approximately correct timing information and will thus also send out synchronization requests regularly. By induction it follows therefore that either in the entire network (which we assume to be connected, see Section III-A) all pairs of devices exchange synchronization messages regularly, or at some point a fault is reported. ■

Note that a fully compromised network is out of the scope of our threat model, since it is impossible to resist an adversary that has full control of all end hosts and all devices. In the rest of this section we will therefore always assume that the network is not fully compromised. Synchronization also allows the detection of DoS attacks, as the following shows.

Lemma V.2 (DoS detection)**.** *If any packet sent out by a TEE is not delivered to the intended neighbor, a fault will be reported.*

Proof: For control-plane packets, acknowledgments are expected, and if they are not received within a certain time interval a fault is reported (timing is guaranteed by synchronization). A data-plane packet p sent by TEE x to neighbor y includes a counter $\mathcal{N}_p$, and y will update its local counter of packets received from x, $\mathcal{N}_x^r$, to match the one in p. If packet p is not received by y, then either a subsequent packet sent by x is received, in which case a fault for packet dropping/reordering is reported by y, or x will eventually detect through the synchronization messages that y has not received p, and will report a packet dropping fault. ■

C. Path and packet integrity

The following Lemmata show the integrity properties for packet delivery between two neighbors and for packet processing on one TEE.

Lemma V.3 (1-hop integrity)**.** *If a TEE x sends a packet p, either p arrives without modification at the TEE y on the next hop on the path, or a fault is reported.*

Proof: If packet p is not delivered to y a fault is reported, as shown in Lemma V.2. If the packet is delivered but it has been tampered with, the MAC verification on y will detect the integrity violation, and y will report a fault. ■

Lemma V.4 (Forwarding integrity)**.** *If a TEE receives a valid packet p, it will process p according to the trusted rules that apply to it, and will therefore correctly determine the next hop on the path to which p should be forwarded. If the TEE is not allowed to complete the processing, or if invalid rules are provided for the packet, then a fault is reported either by the TEE or by its neighbors.*

Proof: This follows directly from the specification of the TEE and from the code execution guarantees of SGX. In case the TEE's execution is blocked, the neighbors will detect this as a DoS attack, see Lemma V.2. ■

We can now state the main result for path and packet integrity.

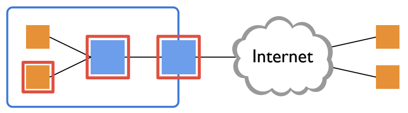
Theorem V.1 (End-to-end integrity)**.** *If a new packet on a host is provided to the host's TEE, then either that packet is correctly forwarded according to the packet integrity and path integrity properties (Section III-C), eventually reaching the correct destination, or else a fault is reported.*

Proof: Follows from inductively applying Lemma V.3 and Lemma V.4 to all TEEs on the path of the packet. ■

D. Packet confidentiality

It is straightforward to see that packet confidentiality holds if our assumptions that the source and destination hosts are not compromised holds. In particular, once the source passes a new packet to its local TEE, the packet will be encrypted and sent out by the source in this form. No TEE on the nodes on the path will ever output a packet in decrypted form (assuming that the TEE code is correct), and the TEE on the

(a)



(b)

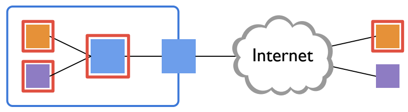


Figure 4: Two simple scenarios in which exfiltration is possible (nodes circled in red are compromised). The lower scenario shows exfiltration in a multi-tenant case: exfiltration for information belonging to the purple tenant is possible through the orange tenant.

destination will only decrypt the packet if it was forwarded correctly (see Theorem V.1). This means, in particular, that only the correct destination can decrypt the packet.

E. *Exfiltration protection*

Unfortunately the properties presented so far cannot unconditionally guarantee exfiltration prevention. For instance, in the extreme case where all devices (hosts and nodes) are compromised, and at least one of these as unrestricted access to the Internet, it is impossible to defend the system against data exfiltration. For a more precise understanding of the problem, it is useful to give a more detailed definition of the threat model. We consider nodes and hosts to be either honest (will behave according to specification) or fully compromised (the adversary completely controls the behavior).

Definition V.1. With respect to the full network graph, the set of compromised devices induces a subgraph which we call the *compromised network graph*; we refer to each connected component in this graph as *badnet*.

Any device inside a badnet has access to the information known by all other devices in the badnet. However, the adversary does not automatically learn the information acquired by a badnet: the adversary is able to access this information if and only if the badnet includes a host with the capability of communicating openly with the Internet (e.g., a gateway node or a remote host). Additionally, two badnets can exchange information if both contain compromised hosts (in case of multi-tenancy, hosts belonging to the same tenant[4]). Two examples in which exfiltration is possible are shown in Figure 4.

To have additional protection in such scenarios, additional measures have to be taken that go beyond securing networking devices. We list the main possibilities.

- **Resilient topology design.** The network could be structured in a way that requires multiple nodes to be traversed before the gateway is reached. Hosts belonging to different tenants could be connected to different nodes.
- **Restricted remote hosts.** Remote hosts should be denied free Internet access, their communications should be restricted to those to the gateway node if possible.
- **Trusted applications.** Particularly sensitive applications could be contained within special TEEs, which will only communicate with the local TEE through an encrypted channel. This option is out of the scope of this paper and subject of future work.

[4]Consider two hosts belonging to different tenants that belong to separate badnets: since the badnets are separate, on any path between them there is an honest node, which will guarantee that tenant isolation is preserved.

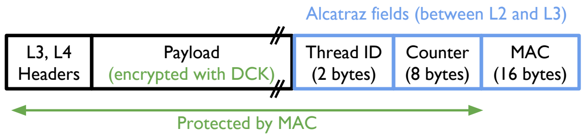


Figure 5: Alcatraz packet structure.

VI. Alcatraz SDN Implementation

Alcatraz is a generic network architecture, so for our implementation we apply Alcatraz to SDN, in particular to OpenFlow (OF). We first show Alcatraz packet structure and implementation with Open vSwitch. Then, we discuss OF extensions for key distribution and rule authentication.

A. *Alcatraz packet structure*

An Alcatraz packet requires that three fields be added compared to traditional packets (see Figure 5). These fields are hop-by-hop, and they consist in an 8-byte *Counter* field used for packet drop and reordering detection, a 2-byte *Thread ID* field for supporting multi-threaded enclaves,[5] and a 16-byte *MAC* field which protects the integrity of the entire packet. We note that, conceptually, Alcatraz can for this reason be seen as a layer 2½ protocol on the dataplane.

B. *Implementation with Open vSwitch*

We implement our prototype using Open vSwitch 2.5.0 and Intel SGX SDK for Linux 1.5 to implement the modules. Open vSwitch and an SGX application with an enclave are implemented as different processes communicating with each other using a Unix domain socket. We run Open vSwitch in user mode (using netdev data-plane) to avoid the interface between a kernel module of Open vSwitch and the enclave in user mode. In particular, the SGX enclave is only allowed to run in user mode to protect the OS from user-defined enclaves.

To pass the packet to the enclave, we modify a function of the Open vSwitch data-plane to capture packets when the OF action is `Output`. Then, using the Unix domain socket, the packet and the matched flow entry are sent to the SGX application, which calls the enclave passing the packet and flow entry to it. Finally, the SGX application receives a modified packet from the enclave and returns it to the

[5]We omit a discussion on multi-threaded enclaves in our treatment for lack of space, but in our implementation we have to use per-thread counters, to avoid the performance loss incurred by using a shared counters. This requires the Thread ID field.

Open vSwitch process using the socket. For cryptographic operations we use the functions provided by the SGX SDK. In particular, we use an AES CBC-MAC function for computing a MAC.

The Alcatraz switch is connected with Ryu [16], which is a Python-based OF controller, and we run an L2 switch application on the controller. We also deploy two LXC [17] containers corresponding to end hosts and their virtual NICs connected with Open vSwitch.

C. OpenFlow extensions

We next discuss how the OF protocol, controller, and switches are extended regarding the key distribution and the authentication of rules (flow entries).

Switch setup: We extend the OF protocol to distribute a master key MK from the OF controller to enclaves of OF switches. Specifically, after exchanging `Hello` messages, the OF controller and the OF switch exchange attestation reports generated by SGX to perform remote attestation. If the remote attestation succeeds, the controller encrypts the master key using the public key of the enclave of the switch and sends MK to the switch. Specifically, when the OF switch receives a `Set-Master-Key` message, which is a new OF message, the switch gives the master key to the enclave. The controller also sends the neighbor list to the switch.

Distribution of flow entries: As we discuss in Section IV-C2, the controller adds a MAC to flow entries so that the switch enclave can verify them. To this end, we extend the `Flow-Mod` message structure to carry the MAC of the flow entry. We also extend the flow tables of the switches to store flow entries with the MACs.

MAC verification: To perform the packet forwarding described in Section IV-D, the packet forwarding engine of the OF switch calls the enclave when the action matching an incoming packet is one of the following: `Output`, `Drop`, `Push-Tag`, `Pop-Tag`, `Set-Field`. OF supports multiple actions for a single flow entry. The packet modification module performs all actions modifying the packet, then the monitoring module performs each of the `Output` actions. Specifically, the monitoring module duplicates the packet and updates the MAC on the basis of the next switch specified by each of the `Output` actions. Alcatraz can thus support multicast.

In the enclave, a MAC of a packet is calculated from packet header (H), packet payload (P), and counter (C) using data-plane attestation key shared between switch x and y ($\mathsf{DAK}_{x,y}$): $MAC = MAC_{\mathsf{DAK}_{x,y}}(H||P||C)$.

In order to compute $\mathsf{DAK}_{x,y}$, the monitoring module needs to know the identifier of the next switch y, but in the OF protocol the flow action specifies an egress interface of the switch. Therefore, to allow the monitoring module to compute the new MAC, the monitoring module needs to be able to map the egress interface specified in the rules to an identifier. To create this mapping, we extend the neighboring statements provided by the administrator (see Section IV-B) with the interfaces of both neighbors in all neighboring statements. Note that for the identifiers defined in Section IV-B we use the data-path IDs (switch IDs) of the OF protocol.

Packet injection: The OF protocol supports a `Packet-Out` message that makes a switch send a packet. The packet sent by this message should also have a MAC so that the next switch can verify it. To allow this function, the controller is treated as a special neighbor switch so that no additional functionality is required by the enclave. In particular, the packet is encrypted by the controller using DCK and sent from to switch x via the control-plane network.

Report to the controller: We also extend the OF protocol to allow the enclaves to send a statistics report to the controller, which includes the reported faults. In existing OF, the packet forwarding engine reports the statistics. In Alcatraz the monitoring module sends the report in authenticated form to the controller so that a malicious forwarding engine cannot tamper with them. Note that the report messages are authenticated using the per-switch control-plane authentication key CAK_x.

Verification of rule cache: Open vSwitch [18], [19] performs packet forwarding in two stages to improve performance. In the first stage, the switch checks a cache that performs exact matching using a hash of the flow. For the first packet, the cache entry does not exist, thus the switch performs full-matching and sends the generic rule containing wildcards to the enclave. Then the rule verification module verifies the rule and generates a MAC of the exact match rule. The generated MAC is stored in the cache together with the cache entry. For the subsequent packets belonging to the same flow, the cache entry is found, and the switch calls the enclave with the cache entry, its MAC, and the packets.

VII. Evaluation

We evaluate Alcatraz performance to answer the following research questions:

- How fast can enclave(s) process packets? What is the bottleneck of the enclaves? We measure the enclave throughput and also conduct micro benchmark analysis (Section VII-A).
- Is an Alcatraz switch viable for a corporate network? We measure its throughput and show that the performance is sufficient for a high-security corporate network (Section VII-B).
- Is the bandwidth overhead of data-plane and control-plane acceptable? We examine the overhead for typical packet sizes (Section VII-C).

A. Enclave performance

To measure the performance, we use a machine with an Intel Core i7-6600U CPU (2.6 GHz, two physical cores with hyper threading) and 8 GB memory. First, we measure the throughput of the enclave containing the monitoring module and the rule verification module. Note that these

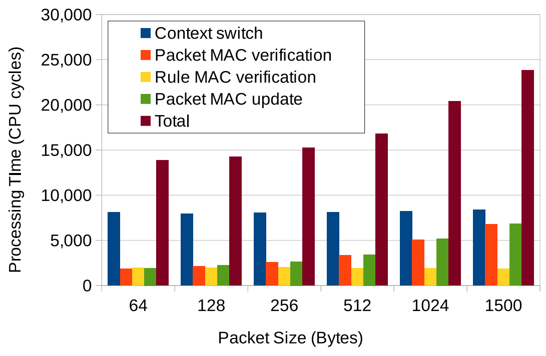


Figure 6: Micro-benchmark of an enclave.

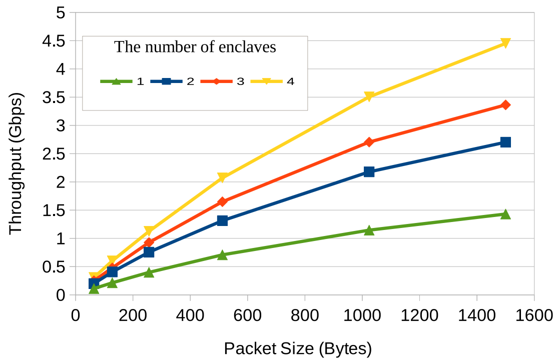


Figure 7: Throughput of enclaves.

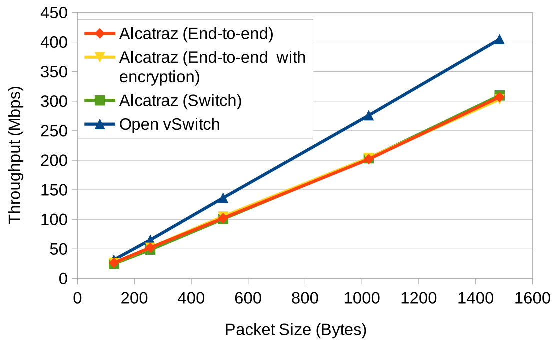


Figure 8: End-to-end throughput.

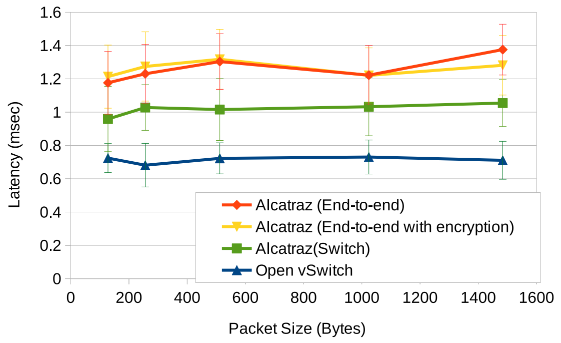


Figure 9: End-to-end latency.

measurements are only of the enclave's performance, i.e., the packet forwarding engine is not included. Figure 7 shows the throughput when varying packet size from 64 B to 1500 B and varying the number of enclaves from one to four (note that our CPU has four logical cores). Large packets get better performance and the throughput reaches a maximum of 4.5 Gbps when the packet size is 1500 B and four enclaves are running simultaneously.

We also conduct a micro-benchmark analysis to identify the bottleneck of the enclave. Figure 6 shows the breakdown of the processing time. *Context switch* shows the processing time when the enclave does nothing and *Packet MAC verification*, *Packet MAC update* and *Rule MAC verification* show additional measurements when each function is enabled. For small packets, the context switch is the heaviest operation, causing over 50% of the total processing time. Therefore, processing time is almost the same for 64 B and 128 B packets. The processing time of MAC calculation is proportional to packet size, and it becomes a significant factor for large packets.

B. Alcatraz switch and end-to-end performance

Next, we measure throughput and latency of an Alcatraz switch and end-to-end communication between the Alcatraz switch and an Alcatraz host. We deploy two SGX machines; one is an Alcatraz switch and another is an Alcatraz host corresponding to both source and destination. To measure their performance, the host sends packets to the switch, then the switch just sends packets back to the host.

Figure 8 shows the end-to-end throughput from host to host when traversing a single switch. We measure the throughput with and without encryption. Figure 8 also shows the throughput of a single Alcatraz switch (without the processing of the hosts) and the throughput of a plain Open vSwitch process (software switch) for comparison. We send UDP packets from the hosts by varying the packet size from 128 B to 1500 B and measure the average bandwidth during 120 seconds.

The throughput of the end-to-end communication reaches 300 Mbps for maximum size packets, and the comparison with the case of the single switch shows that the intermediate switch constitutes the performance bottleneck. By comparing to Open vSwitch, we see that the throughput is reduced by 26%, which is due to the packet verification overhead. This overhead is significant, but we believe that it is nonetheless viable for corporate networks and data centers requiring high security due to the sensitivity of the stored information. Since our prototype is still in the early stages of development, we expect higher performance to be easily attainable through various optimizations.

Figure 9 shows latency for the same setting as for Figure 8, i.e., latency of the end-to-end communication as well as latency of a single Alcatraz switch and of an Open vSwitch switch. The latency is mostly unaffected by the packet size. Theoretically, one would expect the latency of Alcatraz communication to depend on the packet size because the Alcatraz switch and the host need to calculate a MAC over the entire packet. However, the latency caused by the other length-independent operations dominates the delay introduced by the MAC calculation. We can thus conclude that in practice the latency caused by Alcatraz's cryptographic operations can be ignored when compared to the latency of packet transportation.

C. Bandwidth overhead

As we have shown in Figure 5, Alcatraz requires additional 16 bytes for a MAC, 8 bytes for a counter, and 2 bytes for the thread ID. Assuming an average packet size of 850B [20], the bandwidth overhead is 3%, which should be tolerable in most deployment scenarios.

Alcatraz also increases the rule size by 16 bytes due to the addition of the MAC to the rule. Assuming an OpenFlow 1.0 `flow_mod` message specifying a match conditions of source/destination Ethernet addresses and Vlan, the message size is 72 bytes, thus our extension increases the message size by 22%. However, the bandwidth overhead of the control-plane is less critical since the number of messages exchanged is limited

VIII. Related work

Trustworthy Computing in Networks: Relatively few researchers have so far studied the application of Trustworthy Computing in networking contexts, but we believe that this will change due to the emergence of SGX technology that provides much stronger security properties than the previous TPM-based technologies.

The closest related work is TrueNet [21] which performs fault localization based on TEEs executing on network nodes. TrueNet enforces forwarding integrity, and detects malicious nodes that drop or alter packets. In contrast, Alcatraz provides exfiltration resilience, which is a much stronger property, by exploring the design space of TEE-based computation on hosts and nodes. TrueNet, for instance, cannot achieve packet confidentiality nor prevent packet injection. Moreover, Alcatraz can support secure packet updates by deploying a modification module within the TEE. This module accepts trusted rules and can verifiably modify packets before forwarding them. Thanks to this mechanism, Alcatraz can support middleboxes and even software defined networking (SDN) switches which need to be able to inspect and modify packets.

Recently, Kim et al. [22] have proposed to use SGX for networking applications such as software-defined inter-domain routing and in-network TLS inspection. While there are some points in common between this proposal and ours, Alcatraz has a stricter set of goals which provide higher security and more flexibility, such that the functionalities proposed by Kim et al. can be realized as special case in Alcatraz, which is however significantly more general.

We briefly discuss other works that use trustworthy computing in networking, however, they are weakly related as they seek different properties than Alcatraz. NetQuery [23] is a network knowledge plane that relies on trustworthy computing for verifying the correctness of shared information. BIND [24] proposes an attestation approach for linking distributed computations together to achieve trustworthy distributed operation. The Assayer [25] system performs attestation on end host code to prove the benign nature of requests to networked services.

Securing SDN: Some mechanisms to enhance security of SDN have been proposed. Jacquin et al. have proposed an integrity measurement system of OpenFlow switches [26]: specifically, each switch has a TPM and performs measured boot. However, such a system has a very large computing base, and if a vulnerability is found, e.g., in the operating system, then the system cannot guarantee security. SDNsec [27] provides functionalities of path enforcement and validation for SDN. Unfortunately, SDNsec does not provide packet confidentiality, nor does it meet the requirement that packet can be modified by intermediate network nodes.

As for the control-plane security, many mechanisms to protect the controller have been proposed. FortNox [28] performs authentication of SDN application and solves conflicts of rules based on roles of the applications. PermOF [29] isolates SDN applications and enforces policies that specify operations allowed to the applications. Klaedtke et al. have proposed access control based on ownership of the flow entries [30]. Moreover, several architectures [31], [32] that can virtualize/isolate the control-plane to confine a compromised controller. Security of the controller is out of scope of Alcatraz, thus Alcatraz can be used together with these mechanisms to enhance controller security.

IX. Conclusion

We have proposed a new network architecture named Alcatraz that can prevent data exfiltration caused by malicious end hosts and network components. Alcatraz provides strong path integrity through rule verification in a TEE created by Intel SGX, which ensures that packets are forwarded and modified according to legitimate rules. Moreover, Alcatraz guarantees packet integrity and packet confidentiality by performing verification and encryption in the TEE. We have presented its design and initial implementation using Open vSwitch. The main remaining challenge for Alcatraz is the improvement of the performance, but we believe that this can be achieved with relative ease through optimization in the implementation, as well as (in the longer term) through the improvements in trusted computing technologies and the implementation in hardware on a commercial router. For the most security-critical environments the performance of our prototype may indeed already be practical.

X. Acknowledgments

We gratefully acknowledge the support from ETH Zurich, from NEC Corporation, and from the Zurich Information Security and Privacy Center (ZISC).

References

[1] GovCert.ch, “APT case RUAG,” Tech. Rep., 2016, https://www.melani.admin.ch/dam/melani/en/dokumente/2016/technical%20report%20ruag.pdf.

[2] US-CERT, “Alert TA16-250A,” https://www.us-cert.gov/ncas/alerts/TA16-250A, accessed: 2017-10-20.

[3] “Cisco confirms NSA-linked zeroday targeted its firewalls for years,” https://perma.cc/2TMS-5PQA, 2016, accessed: 2017-10-20.

[4] “SYNful Knock - A Cisco router implant,” https://www.fireeye.com/blog/threat-research/2015/09/synful_knock_-_acis.html, 2015, accessed: 2017-10-20.

[5] “Important announcement about ScreenOS,” http://forums.juniper.net/t5/Security-Incident-Response/Important-Announcement-about-ScreenOS/ba-p/285554, 2015, accessed: 2017-10-20.

[6] S. Checkoway, S. Cohney, C. Garman, M. Green, N. Heninger, J. Maskiewicz, E. Rescorla, H. Shacham, and R.-P. Weinmann, “A systematic analysis of the Juniper Dual EC incident,” in *ACM CCS*, 2016.

[7] Mandiant, “M-trends 2016,” Tech. Rep., 2016, https://www.fireeye.com/current-threats/annual-threat-report/mtrends.html.

[8] T. Beery and C. Hoch, “The remote malicious butler did it!” ser. Black Hat, 2016, https://www.blackhat.com/docs/us-16/materials/us-16-Beery-The-Remote-Malicious-Butler-Did-It-wp.pdf.

[9] A. K. Sood and R. J. Enbody, “Targeted cyberattacks: A superset of advanced persistent threats,” *IEEE Security & Privacy*, vol. 11, no. 1, pp. 54–61, 2013.

[10] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, “Innovative instructions and software model for isolated execution,” in *Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[11] V. Costan and S. Devadas, “Intel SGX Explained,” *Cryptology ePrint Archive, Report 2016/086*, 2016.

[12] “Intel Trusted Execution Technology: White Paper,” https://www-ssl.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html, accessed: 2017-10-20.

[13] “OpenFlow Switch Specification, version 1.0.0,” 2009, accessed: 2017-10-20.

[14] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, “BlindBox: Deep packet inspection over encrypted traffic,” in *ACM SIGCOMM*, 2015.

[15] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune, “Memoir: Practical state continuity for protected modules,” in *IEEE S&P*, 2011.

[16] “Ryu,” https://osrg.github.io/ryu/, accessed: 2017-10-20.

[17] “Linux Containers - LXC,” https://linuxcontainers.org/lxc/, accessed: 2017-10-20.

[18] “Open vSwitch,” http://openvswitch.org/, accessed: 2017-10-20.

[19] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, “The design and implementation of Open vSwitch,” in *NSDI*, 2015.

[20] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding data center traffic characteristics,” *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, Jan. 2010.

[21] X. Zhang, Z. Zhou, G. Hasker, A. Perrig, and V. Gligor, “Network fault localization with small TCB,” in *IEEE ICNP*, 2011.

[22] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han, “A first step towards leveraging commodity trusted execution environments for network applications,” in *ACM HotNets*, 2015.

[23] A. Shieh, E. G. Sirer, and F. B. Schneider, “NetQuery: A knowledge plane for reasoning about network properties,” in *ACM SIGCOMM*, 2011.

[24] E. Shi, A. Perrig, and L. V. Doorn, “BIND: A fine-grained attestation service for secure distributed systems,” in *IEEE S&P*, 2005.

[25] B. Parno, Z. Zhou, and A. Perrig, “Using trustworthy host-based information in the network,” in *ACM Workshop on Scalable Trusted Computing (STC)*, 2012.

[26] L. Jacquin, A. L. Shaw, and C. Dalton, “Towards trusted software-defined networks using a hardware-based integrity measurement architecture,” in *NetSoft*, 2015.

[27] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, “SDNsec: Forwarding accountability for the SDN data plane,” in *ICCCN*, 2016.

[28] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A security enforcement kernel for OpenFlow networks,” in *ACM HotSDN*.

[29] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, “Towards a secure controller platform for OpenFlow applications,” in *ACM HotSDN*, 2013.

[30] F. Klaedtke, G. O. Karame, R. Bifulco, and H. Cui, “Towards an access control scheme for accessing flows in SDN,” in *IEEE NetSoft*, 2015.

[31] R. Sherwood, G. Gibb, K.-k. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, “FlowVisor: A network virtualization layer,” Tech. Rep. OPENFLOW-TR-2009-1, 2009.

[32] T. Sasaki, D. E. Asoni, and A. Perrig, “Control-plane isolation and recovery for a secure SDN architecture,” in *NetSoft*, 2016.