ENDBOX: Scalable Middlebox Functions Using Client-Side Trusted Execution

David Goltzsche*, Signe Rüsch*, Manuel Nieke*, Sébastien Vaucher†, Nico Weichbrodt*, Valerio Schiavoni†, Pierre-Louis Aublin‡, Paolo Costa§, Christof Fetzer¶, Pascal Felber†, Peter Pietzuch‡ and Rüdiger Kapitza*

*TU Braunschweig, Germany, goltzsche@ibr.cs.tu-bs.de, rrkapitz@ibr.cs.tu-bs.de
†University of Neuchâtel, Switzerland, pascal.felber@unine.ch
‡Imperial College London, United Kingdom, prp@imperial.ac.uk
§Microsoft Research, United Kingdom, paolo.costa@microsoft.com
¶TU Dresden, Germany, christof.fetzer@tu-dresden.de

***Abstract*—Many organisations enhance the performance, security, and functionality of their managed networks by deploying *middleboxes* centrally as part of their core network. While this simplifies maintenance, it also increases cost because middlebox hardware must scale with the number of clients. A promising alternative is to outsource middlebox functions to the clients themselves, thus leveraging their CPU resources. Such an approach, however, raises security challenges for critical middlebox functions such as firewalls and intrusion detection systems.**

We describe ENDBOX, a system that securely executes middlebox functions on client machines at the network edge. Its design combines a virtual private network (VPN) with middlebox functions that are hardware-protected by a trusted execution environment (TEE), as offered by Intel's Software Guard Extensions (SGX). By maintaining VPN connection endpoints inside SGX enclaves, ENDBOX ensures that all client traffic, including encrypted communication, is processed by the middlebox. Despite its decentralised model, ENDBOX's middlebox functions remain maintainable: they are centrally controlled and can be updated efficiently. We demonstrate ENDBOX with two scenarios involving (i) a large company; and (ii) an Internet service provider that both need to protect their network and connected clients. We evaluate ENDBOX by comparing it to centralised deployments of common middlebox functions, such as load balancing, intrusion detection, firewalling, and DDoS prevention. We show that ENDBOX achieves up to 3.8× higher throughput and scales linearly with the number of clients.

I. INTRODUCTION

Middleboxes are part of backbones of large networks that are managed by organisations (*managed networks*), and implement diverse sets of functions related to security (e.g. firewalls and intrusion detection), and performance (e.g. caching and load balancing). At the same time, they must handle growing network traffic [1] and ever-increasing network-based attacks [2], [3], all while remaining efficiently manageable and cost-effective.

The current best practice is to deploy middleboxes centrally as part of a network, despite high infrastructure and management costs [4]. Recent research proposals, instead, investigate the benefits of outsourcing middleboxes to cloud infrastructures [4], [5]. While this reduces maintenance effort and, in turn, cost, deploying critical network functions externally and redirecting sensitive network traffic off-site introduces potential security risks and may be illegal.

To address these limitations, we propose a new *decentralised* deployment approach in which middlebox functions are placed on client machines at the network edge. Thus, middlebox functions can exploit the potentially idle resources of client machines for processing client traffic. This approach is especially efficient as client traffic constitutes a large fraction of traffic in managed networks [6], [7].

A decentralised deployment model for middleboxes raises two new challenges: (i) it requires clients to be trusted to execute middlebox functions faithfully, and (ii) network administrators must retain control over middlebox functions [6], which is more challenging with distributed middleboxes. While this is achievable for tightly administered machines such as servers, it is in contrast with today's IT management practices in which employees working as developers retain administrative privileges on their machines. Due to missing patches, improper configuration, careless users, or rogue insiders, client machines are more vulnerable to malicious software which try to circumvent client-side middlebox functions.

Many organisations would therefore consider essential middlebox functions such as firewalls or intrusion detection systems as too critical to be entrusted to client machines not under their control. For example, Internet service providers (ISPs) would typically be reluctant to deploy intrusion detection and prevention systems (IDPSs) at customers' client machines to prevent malware from spreading; companies would refrain from performing data leak prevention (DLP) on employee machines but rather install it at a centralised gateway. Thus far, research proposals have mostly considered host-based deployments of network functions for trusted server machines [4]–[6].

We describe **ENDBOX**, a new system for the trusted execution of middlebox functions on client machines. The design of ENDBOX is based on a virtual private network (VPN), namely OpenVPN [8], which is used to access a managed network from an untrusted one. We enhance the VPN client with support for the execution of trusted middlebox functions through the Click software router [9]. ENDBOX intercepts all traffic between the client and the network and ensures that it is processed by middlebox functions executing on the client machine. The functions are guarded by trusted hardware features available in modern CPUs—ENDBOX uses Intel's Software Guard Extensions (SGX) to enforce their use when clients communicate with a managed network and to protect their integrity.

 Presented in the 48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '18). The final version of this paper is available under DOI: 10.1109/DSN.2018.00048.

To support also widely prevalent encrypted network traffic [10], [11], ENDBOX leverages its trusted execution model for encrypted traffic analysis. In contrast to man-in-the-middle (MITM) proxies, which may compromise encrypted sessions, ENDBOX shields encryption keys locally on the clients, thereby enabling decryption without weakening overall security.

Despite its decentralised deployment model, middlebox functions executed by ENDBOX can be reconfigured securely, rapidly, and seamlessly. ENDBOX uses user-defined in-band VPN control messages that are periodically exchanged between a control server in the managed network and all ENDBOX clients. Control messages announce configuration updates to middlebox functions, forcing clients to always use the latest configuration version. The overhead of exchanging control messages and applying new middlebox configurations is low because ENDBOX clients retrieve new configurations asynchronously.

The remainder of the paper is organised around its main contributions:

§II introduces two scenarios for ENDBOX and discusses the problem statement as well as the threat model that we consider when outsourcing middlebox functions to untrusted clients;

§III describes the ENDBOX design, thereby explaining how it secures middlebox functions using Intel SGX, maintains VPN connection endpoints inside SGX enclaves, and securely processes encrypted network traffic without compromising end-to-end security;

§IV describes implementation details on how ENDBOX integrates with the VPN client and the Click software router. We also detail our approach to reduce the number of SGX enclave transitions, enable use case specific traffic protection, and optimise communication between ENDBOX clients;

§V evaluates ENDBOX and shows that it is immune to many attacks such as rollback, replay, denial-of-service (DoS) or cipher downgrade attacks. We show ENDBOX scales linearly with the number of connected clients and achieves 2.6× to 3.8× higher throughput compared to a centralised middlebox deployment. Finally, we show that ENDBOX has a low performance overhead of 16%.

II. TOWARDS SECURE CLIENT-SIDE MIDDLEBOXES

We first present explicit scenarios that benefit from the deployment of secure client-side middleboxes (§ II-A). We then describe how middleboxes are deployed in today's managed networks and why state-of-the-art solutions are not suitable for implementing the aforementioned scenarios (§ II-B). Finally, we explain the Intel Software Guard Extensions (SGX) as an enabling technology for our solution (§ II-C) and discuss our assumed threat model with respect to untrusted clients (§ II-D).

A. Scenarios

We describe two representative scenarios that benefit from secure client-side middleboxes as provided by ENDBOX.

Scenario 1: Enterprise network. A large company seeks to protect their network using middleboxes. Due to the increasing cost of centralised hardware middleboxes, the company decides to offload middlebox functions. It is decided to let client machines execute middlebox functions using ENDBOX. In line with employees working from remote locations, clients can be connected either to the internal network or join the network remotely using a VPN client.

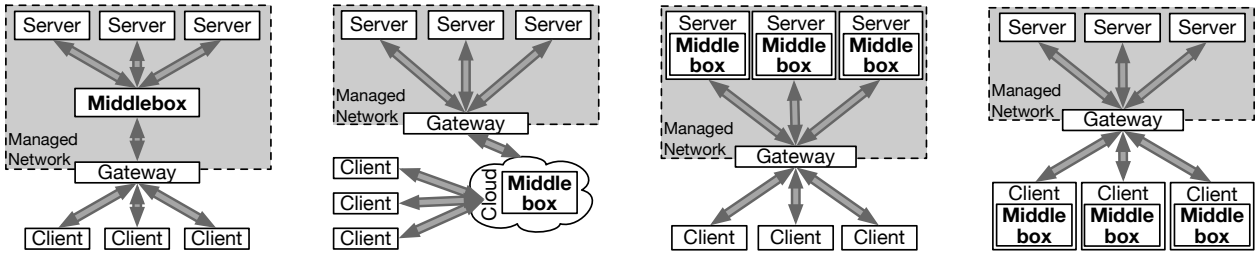
Scenario 2: ISP network. An Internet service provider (ISP) with hundreds of thousands of customers wants to offer additional protection by performing deep packet inspection (DPI) on network packets. The goal is to protect customers' client machines as well as the ISP's network components from malware, such as ransomware. However, it is challenging for the provider to implement such a system because (i) they need to access encrypted traffic payload, which is impossible without creating security vulnerabilities [12], changing well-established protocols [13], [14] , or inflicting a non-justifiable performance overhead [15]; and (ii) the acquisition of middleboxes being capable of performing extensive analysis on considerable amounts of traffic is too costly [16]. The product portfolio of the ISP is extended by a data plan that deploys ENDBOX for network traffic analysis on the client machines of customers. The plan includes a discount to compensate for the allocation of client-side resources.

B. Middleboxes today

Middleboxes play a central role in analysing, filtering, and manipulating network traffic. Typical examples are firewalls and IDPSs for improved security; or caches and load balancers for better performance. We observe three fundamentally different approaches of deploying middleboxes: (i) centralised deployment as part of the managed network; (ii) cloud-based deployment; and (iii) deployment as part of end-hosts.

Centralised middlebox deployments. This is the most common type of deployment in managed networks in which the middleboxes are placed between servers and the gateway to the Internet (see Fig. 1a). As middleboxes are diverse and often complex, there is a trend to replace costly specialised hardware appliances by software-based solutions running on top of commodity hardware [17]. With ever-growing network traffic and enterprise links offering capacities reaching 100 Gbps, this requires scalable software solutions. Since middleboxes are often stateful (e.g. during intrusion detection), it is challenging to perform simple horizontal packet-based scaling, as each network flow must be assigned to an individual middlebox instance [7]. Centralised middlebox deployments are non-trivial to scale with the number of client machines, resource-intensive and consequently costly [4].

Cloud-based middlebox deployments. In line with the trend of network functions virtualization (NFV) [18], middleboxes are outsourced to public clouds operated by a third party [4] or private *telco clouds* operated by ISPs [19] (see Fig. 1b). Although using public clouds relieves network administrators from the management of middleboxes, it comes with several downsides: (i) in order to be processed in a cloud infrastructure, traffic must be redirected thereby incurring additional latency; (ii) public clouds are external, untrusted infrastructure, i.e.



(a) centralised (b) cloud-based (c) server-side (d) client-side

Fig. 1: Different middlebox deployment models: (a) traditional centralised hardware middleboxes, (b) outsourcing of software middleboxes into public or private cloud environments, and (c, d) software middleboxes at end-hosts

critical functions are moved off-site; and (iii) traffic redirected to clouds may be filtered or manipulated outside the network.

Offloading middlebox functions to private telco clouds may incur less latency and the infrastructure can be regarded as more trustworthy. However, it still needs substantial investment by ISPs. In summary, cloud-based middleboxes are convenient to manage, but potentially reduce the reliability of managed networks. They are often discarded because of concerns regarding security, latency, and legality.

Middleboxes at end-hosts. Finally, middlebox functions may also be placed at end-hosts, be it servers in a data centre (see Fig. 1c) [6] or clients inside an enterprise environment (see Fig. 1d) [20]. These approaches benefit from network traffic being processed directly at its source or destination, improving scalability as each host handles its own traffic. However, fully untrusted end-hosts have not been considered, which is the key challenge introduced by the scenarios described in § II-A. In contrast, ETTM [20] does consider untrusted end-hosts, but its approach is limited: Contrary to ENDBOX, ETTM (i) provides lower security guarantees; e.g. it cannot withstand physical attacks; (ii) relies on traffic being correctly forwarded by physical switches, thus, extending the trusted computing base (TCB) of the whole system; and (iii) builds on an expensive distributed consensus algorithm (see §VI).

In this paper, our goal is to explore a deployment model that targets entirely untrusted clients and network hardware in order to reap the following benefits: (i) network traffic can be filtered or processed at the source or destination; (ii) processing encrypted traffic does not create vulnerabilities and is practical; (iii) central network devices in a managed network are relieved from having to provide middlebox functions; and (iv) deployments can be made to scale because middlebox functions are executed by potentially under-utilised client machines.

C. Intel SGX

Recent Intel CPUs include support for trusted execution environments (TEEs), in the form of Software Guard Extensions (SGX). SGX enables the protection of data and code through safe compartments called *enclaves*. Computations performed inside an enclave are isolated from potentially malicious software, including the operating system.

SGX uses special x86 instructions to create and manage enclaves. Enclaves occupy an isolated logical memory range inside the address space of a process. SGX protects the integrity as well as the confidentiality of this range with checksums and memory encryption. Enclave memory is stored in a system-reserved memory range called enclave page cache (EPC) that is transparently encrypted [21].

The Intel SGX software development kit (SDK) offers functionality to help with enclave software development, such as life cycle management or support for function calls across the enclave boundary. Function calls that cross from the untrusted to the trusted environment are called *ecalls*, while *ocalls* perform the opposite.

In addition to protecting code and data, SGX can authenticate enclaves through local or remote *attestation*: local attestation provides a way for two enclaves on the same machine to authenticate each other based on *measurements*, which basically are hashes of the enclaves. Attestation depends on messages called *reports* that can contain user-defined data, e.g. for binding data to an enclave instance. Remote attestation is based on keys fused into the CPU during manufacturing and extends attestation to a remote machine [22]. The process involves data structures called *quotes* that are generated by a special enclave called Quoting Enclave (QE). Using the web-based Intel Attestation Service (IAS), quotes can be remotely verified to originate from a genuine SGX CPU.

The use of SGX involves some restrictions. Since enclave code must be isolated from the untrusted environment, it is not possible to make system calls to the OS. Prior work has addressed this issue by embedding system support inside the enclave [23]–[25] while increasing the TCB size. The EPC size in the current version of SGX is limited to 128 MB per machine. It is possible to create larger enclaves by swapping EPC pages to regular memory, but this results in a substantial performance penalty [23], [26]. While SGX is vulnerable to side-channel attacks [27]–[29], research exists on mitigation techniques [30], [31].

D. Threat model

Client machines are typically untrusted, as they elude from the control of network owners. In companies, not all enterprise machines are managed by a central IT department, i.e. developers or administrators typically possess administrative rights for their own and others' machines. In the case of the ISP scenario, the client machines of customers are and should be totally out of control of the provider. Also, client machines

may lack essential security patches or be misconfigured, and, thus, are vulnerable to attacks which could circumvent any security critical middlebox functions.

We therefore assume that client machines are *not* trustworthy, and an adversary may have full control over a client machine, including its operating system, hypervisor, and hardware. They can make it send any traffic, and they have access to inbound traffic, i.e. they can drop or modify packet contents. In addition, they have full control over the OS networking stack, and can bypass or modify any of its functionality. With physical access to the client machines, the adversary can read from or write to any memory address.

The adversary can also mount DoS attacks against the enclave, i.e. not starting or entering it. However, we ignore distributed denial-of-service (DDoS) attacks on the server infrastructure: while malicious clients can collude and send spurious traffic to servers, existing mitigation approaches can be applied [32].

In line with typical assumptions about managed networks, we consider all servers to be under central administrative control and thus trustworthy. Client machines are not allowed unrestricted access to the network because they can be subject to the above attacks and act maliciously.

In contrast, we assume that users put trust in the provider of middlebox functions (e.g. the company or the ISP). Note that this assumption is also valid for traditional approaches involving middleboxes. However, this assumption could be weakened or removed by enabling users to enforce policies during runtime on SGX enclaves [33].

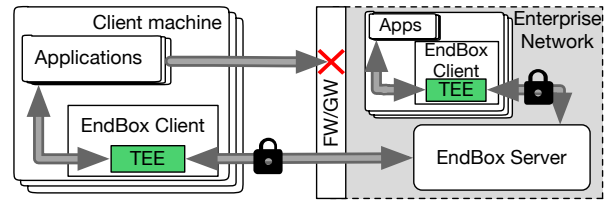
III. Design

We describe EndBox, a system that securely executes middleboxes at client machines. In accordance with a deployment scenario as part of untrusted clients (see § II-D), EndBox must satisfy the following requirements:

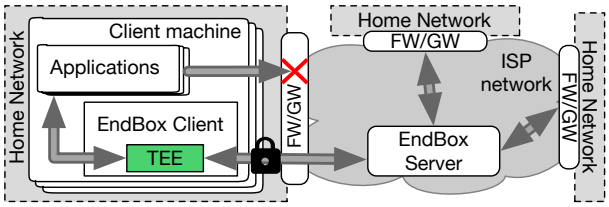
R1: *Flexibility.* EndBox should support flexible development for tailored middlebox functions for a wide range of use cases.
R2: *Enforcement.* EndBox should ensure that all traffic between the client and the managed network is processed by middlebox functions.
R3: *Integrity and privacy.* EndBox should protect the integrity of middlebox functions and the privacy of client traffic.
R4: *Manageability.* Despite middleboxes being distributed, it should remain easy for network administrators to rapidly and seamlessly manage middlebox functions, such as updating their configurations.
R5: *Low overhead and good scalability.* To be practical, EndBox should introduce only a low performance overhead compared to existing solutions and scale linearly with the number of clients, in order to support fluctuating client numbers and prevent idle middleboxes at the same time.

A. EndBox *in a nutshell*

Fig. 2 details the deployment of EndBox for our two representative scenarios introduced in § II-A. In both scenarios, a number of EndBox *clients* connect to an EndBox *server*. The EndBox client allows applications on client machines



(a) enterprise network scenario



(b) ISP network scenario

Fig. 2: EndBox system deployment of two scenarios (a) and (b), FW/GW is firewall/gateway

to access the managed network. Clients execute middlebox functions in a TEE (SGX enclaves in our prototype, denoted in **green** throughout the paper). The TEE guards the secure endpoint of the VPN communication and secures the necessary encryption keys. The keys are injected inside the TEE as part of a secure bootstrapping process, so neither the user of the machine nor software outside the TEE are granted access to them (see § III-C). Packet en- and decryption, as well as arbitrary processing, happen within the TEE. This enables the implementation of a wide range of middlebox functions (*R1*), including caching, malware detection, licensing controls, and functions such as compression that all cannot operate on encrypted packets (see § III-D). Moreover, this enables organisations to adapt middlebox functions executed with EndBox to their specific use case.

In the case of the enterprise network scenario (Fig. 2a), clients are allowed to be inside the network or to connect remotely (e.g. employees in home office). In contrast, in the ISP network scenario (Fig. 2b), clients are private machines that connect to the ISP network.

In both scenarios, the use of EndBox is enforced when accessing a managed network because the EndBox server is the only entry point: it only accepts traffic encrypted with the key owned by a correct EndBox client. This ensures that all traffic is processed by EndBox and prevents users from bypassing the middlebox functionality, because bypassed traffic is either blocked or encrypted, thus not readable (*R2*).

In addition, EndBox uses the SGX attestation support to guarantee that (i) the enclave is initialised with the correct code and data; and (ii) the encryption and decryption of network packets can only occur within the enclave (*R3*).

The EndBox server provides a management interface that enables administrators to deploy middlebox configuration changes (*R4*), e.g. to issue updates for middlebox functions. The updates are disseminated to all (connected and reconnecting) clients, which are responsible for fetching and applying the configuration changes. After a configurable grace period, the

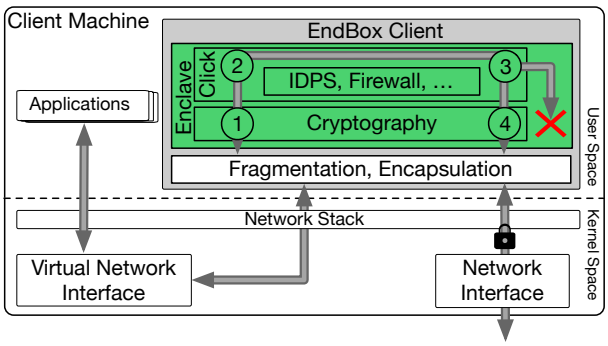


Fig. 3: Architecture of the ENDBOX client

update is enforced by accepting only traffic from ENDBOX clients with the latest middlebox configuration (see § III-E)

ENDBOX is designed to induce a low performance overhead, and scale with the number of connected clients (*R5*). This is achieved by (i) reducing the number of enclave mode transitions; and (ii) by moving middlebox functions to clients, thus removing load from centralised middleboxes as part of the managed network.

B. Architecture of the ENDBOX client

The ENDBOX client architecture shown in Fig. 3 consists of two components: a VPN client and a set of middlebox functions. The VPN client is based on OpenVPN [8] and is partitioned; security-sensitive parts (such as cryptographic functions and encryption keys) are moved into the enclave to prevent an attacker from gaining knowledge about the secret. Other parts that are not important for security (such as packet encapsulation and fragmentation) are executed outside of the enclave. ENDBOX implements middlebox functions using the Click modular router [9], which can be used to implement a diverse set of middlebox functions (*R1*). ENDBOX routes all traffic through middlebox functions: before encrypting egress or decrypting ingress traffic, OpenVPN hands all packets to Click for processing.

To ensure that all network traffic is intercepted by ENDBOX (*R2*), a client can only connect to the network through the VPN. The VPN client processes each IP packet individually in four steps: after the packet is copied inside the enclave ①, it is processed by one or more middlebox functions according to the system configuration ②. Depending on the specific function, the packet header or payload may be modified or the whole packet marked to be discarded (e.g. due to firewall or IDPS rules). After the execution of the middlebox functions, the packet is either accepted or rejected ③. Finally, the packet is signed and encrypted and then copied outside of the enclave, where it is passed back to the VPN client running in untrusted space ④ for transmission over the network.

Each packet arriving from the network is processed in the opposite order: it is first copied into the enclave, where its signature is checked and its content decrypted. It is then processed by middlebox functions, accepted or discarded, and finally copied outside the enclave and passed to the application.

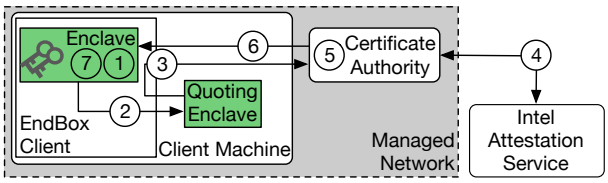


Fig. 4: ENDBOX remote attestation and key management

C. Attestation and key management

To achieve our desired level of security (*R3*), ENDBOX leverages the Intel SGX enclave attestation facilities [22] described in § II-C and. Fig. 4 shows the steps executed to attest the correctness of the enclave as well as to protect and sign the VPN keys. ENDBOX's key management is based on a certificate authority (CA) operated by network owners. The public key of the CA is pre-deployed into enclave binaries during system compilation to prevent MITM attacks. An asymmetric key pair is generated in the enclave ①, the private key never leaves the enclave. Next, the VPN client creates a report containing the public key of the aforementioned key pair and passes it to the QE to obtain a quote ② (see § II-C). This is forwarded to the CA ③, which relays it to the IAS and receives a reply ④. If that reply is positive and the quote contains a known measurement, the CA signs the public key, creating a certificate ⑤. The certificate and a symmetric shared key encrypted with the enclave's public key are provisioned to the enclave ⑥. Finally, after checking the received certificate with the CA's public key, the enclave persistently stores the generated key pair as well as the certificate using the SGX sealing feature ⑦. The client can now use that certificate to connect to the VPN server. Consequently, an enclave only has to be attested once and unattested clients cannot establish connections because of missing certificates. The symmetric shared key is used for decrypting configuration files as described in § III-E.

D. Processing encrypted network traffic

Although half of today's network traffic is encrypted [10], [11], many middlebox functions, e.g. for deep packet inspection or caching, need to access the payload of packets, i.e. cannot operate on encrypted traffic. This problem became particularly visible when the Internet Engineering Task Force (IETF) heavily discussed whether the key exchange in TLS 1.3 should be degraded to allow network monitoring [34].

There are different state-of-the-art solutions for this problem: (i) middleboxes performing a MITM attack on users; (ii) applying modifications to the TLS protocol to allow middleboxes to intercept traffic [13], [14]; and (iii) searchable or homomorphic encryption schemes [15]. These solutions solve the problem, but have drastic disadvantages: they break end-to-end security, are incompatible with technologies like HTTP public key pinning (HPKP), are impractical or are notoriously slow. Therefore, ENDBOX implements a new approach of decrypting network traffic. We assume that a client application, such as a web browser, is linked against a custom untrusted TLS library. This library forwards all negotiated session keys to the trusted Click instance, running inside the ENDBOX VPN client. The keys

are used to decrypt the packets inside a special Click element. For our prototype implementation, we modify OpenSSL by adding a *single* call to a custom function, which forwards negotiated keys via the OpenVPN management interface. Using this approach, ENDBOX can decrypt traffic transparently to the client. The client neither needs to trust a custom certificate authority nor does it see different certificates than those offered by the accessed services. Also, we do not have to change the TLS protocol or rely on special encryption schemes. Note that transferring the keys to the ENDBOX enclave is not a security risk for the client: the keys are generated by the untrusted TLS library and are therefore also stored in untrusted memory.

Our approach to analyse encrypted traffic also works with the upcoming TLS version 1.3, which today's middleboxes cannot handle correctly [35]. Additionally, our solution is applicable in our targeted scenarios: In an enterprise network, employees trust their employer to some extent and should refrain from handling private matters using company networks either way. In the ISP scenario, we assume customers opt-in for traffic analysis by the ISP to improve security, i.e. they are aware of it and consent.

E. Configuration updates

To improve manageability (*R4*), ENDBOX supports updates of Click configuration files at runtime. Network administrators can define the importance of updates by specifying a *grace period* of $n \geq 0$ seconds. During the grace period, the ENDBOX server allows both old and new configurations to be active. After its expiry, the server blocks traffic from clients that are not applying the new configuration.

We use in-band ping messages from OpenVPN to notify ENDBOX clients about configuration updates and to enforce them. These ping messages are sent periodically by both the VPN client and the server to keep the connection alive. We extend the message format with two extra fields: the version number of the latest configuration file and its grace period. To prevent malicious clients from sending crafted ping messages, the authenticity of all packets is validated inside the enclave.

The CA's public key and the pre-shared key (see § III-C) are used to sign and optionally encrypt configuration files to, for example, hide IDPS rules from employees in the enterprise scenario. In the ISP scenario, the configuration files are not encrypted to allow customers to inspect rules. The files are stored on a trusted server located in the managed network that is publicly accessible to ensure that clients can always obtain up-to-date configurations before connecting. When network administrators create an updated configuration file, they sign and optionally encrypt it, upload it to the configuration server, and instruct the VPN server to send out a ping message with the new version number. When the ENDBOX client notices that a new configuration file is available, it fetches the configuration file, decrypts it inside the enclave and applies it. To prevent clients from replaying old configuration files, the version number of the update is incorporated inside the update itself. Version numbers increase monotonically with each update.

The whole update process is shown in Fig. 5. To start it, the network administrator uploads the configuration file to the configuration server ① and triggers a configuration update at the ENDBOX VPN server ②. The VPN server starts a timer that, when expired, blocks clients with old configurations ③. With the next periodic ping message, the VPN server sends the new version number to all clients ④. When a client receives a ping message, it checks whether an update is necessary ⑤. If this is the case, it fetches the new configuration from the configuration server ⑥-⑦, decrypts it, and replaces its current configuration ⑧. Finally, the client sends a ping message with the new version number to prove its successful update ⑨.

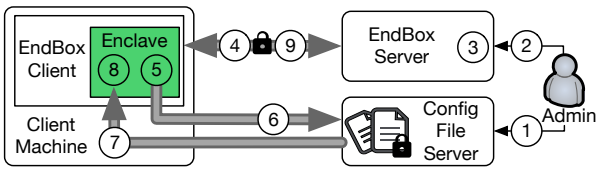


Fig. 5: Updating configuration files within ENDBOX

IV. IMPLEMENTATION

The implementation of ENDBOX is based on OpenVPN v2.4.0 [8], the Intel SGX SDK v1.9 [36], the TaLoS library for terminating TLS connections inside SGX enclaves [37], and the latest version of the Click software router [9]. We use OpenVPN as the basis for the ENDBOX client because it (i) is open-source; (ii) has relatively few dependencies; (iii) is implemented in user-space; and (iv) is widely used. This allows us to port parts of its implementation to an SGX enclave, especially given that OpenVPN is entirely executed in user-space. TaLoS is based on LibreSSL and acts as a drop-in replacement running in SGX enclaves for existing applications.

ENDBOX uses the Intel SGX SDK to define *ecalls* and *ocalls* as well as to handle the life cycle of the enclave. In addition, it uses the SDK's trusted (but functionally limited) C library implementation and extends it with further functions used by OpenVPN and Click. The ENDBOX implementation also utilises the SDK support for trusted time in order to implement traffic shaping (see § V-B). Additionally, the SDK offers a *simulation mode* that allows the execution of SGX applications on unsupported hardware without security guarantees but similar runtime behaviour.

ENDBOX relies on Click to implement middlebox functions. To configure Click, so-called *elements* are interconnected. An element can fetch packets from or forward packets to other elements, and process packets. We choose Click because it (i) is widely used; (ii) has many existing elements to realise various middlebox functions; (iii) provides a configuration hot-swapping mechanism; and (iv) is easily extensible. ENDBOX uses Click's configuration hot-swapping mechanism to efficiently update the middlebox configuration. It uses elements shipped with Click to implement middlebox functions and extends Click by adding custom elements for an IDPS function, to decrypt application-level traffic, and to perform traffic shaping using a trusted time source provided by SGX.

Changes to Click and OpenVPN. ENDBOX requires minor changes to Click: (i) the `ToDevice` element is modified to signal OpenVPN when a packet was accepted or rejected. There are

also changes to Click core: (ii) we disable signal handling for state clean-up and control sockets for communication with specific elements, as signals are not supported inside enclaves; and (iii) we adapt the hot-swapping mechanism to work with configuration files stored in memory. OpenVPN is linked against the TaLoS library, which results in all cryptographic operations being executed inside the enclave. Additionally, we compile Click as a library and link it against the enclave code to allow fast interaction.

TCB size. The total number of lines of code (LOC) within the enclave is an important factor for the TCB size. The trusted part of ENDBOX comprises 320 kLOC: 219 kLOC for TaLoS, 80 kLOC for Click, 20 kLOC for the SGX SDK, and 1 kLOC for the sensitive parts of OpenVPN. The number of lines of code for TaLoS should be regarded as an upper bound: while TaLoS provides the same API and functionalities as LibreSSL, ENDBOX only uses a small subset.

A. Optimisations

ENDBOX implements several optimisations to improve its performance and security: (i) reduce the number of enclave transitions; (ii) enable use case specific traffic protection; and (iii) optimise client-to-client communication. These optimisations are detailed in the following and evaluated in § V-G.

Enclave transitions. The performance of SGX enclaves is negatively impacted by transitions between trusted and untrusted code. Previous work [23], [26] has shown that an enclave transition is more costly than a system call. To reduce this cost, ENDBOX shifts parts of the OpenVPN encryption logic into the enclave to reduce the number of enclave transitions per processed packet: ENDBOX performs only one *ecall* per sent or received packet. As described in § V-G, this optimisation drastically improves the overall throughput of ENDBOX.

Scenario-specific traffic protection. Depending on the scenario in which ENDBOX is used, weaker traffic protection can be applied. In the ISP scenario, `AES-128-CBC` packet encryption is optional, because the trust relationship is different from the enterprise use case: users decided to let the ISP apply ENDBOX, therefore the fact that traffic is routed through Click does not have to be enforced by encrypting it. However, the fact that *egress* traffic is analysed by Click needs to be ensured by the ISP by applying integrity protection. This optimisation only targets the ISP scenario and improves the overall throughput of ENDBOX, as described in § V-G.

Client-to-client communication. In the case of client-to-client connections, our approach would lead to packets being processed multiple times, once on every client. This is not reasonable for most use cases, for example IDPSs. Therefore, ENDBOX clients flag outgoing packets after they have been processed by Click, enabling other ENDBOX clients to bypass Click. We implemented the flagging mechanism by setting the Quality of Service (QoS) byte in the IP header to `0xeb`. In order to prevent external attackers from sending IP packets containing this byte, the ENDBOX server removes the QoS byte if it is set to `0xeb`. Finally, as all packets are integrity-protected by OpenVPN, flagged packets cannot be forged. This optimisation rather targets the enterprise scenario, but can also be applied to the ISP network and improves the latency between ENDBOX clients, as described in § V-G.

B. Secure Enclave Interface

The enclave interface of ENDBOX consists of 90 calls: 70 *ecalls* and 20 *ocalls*. Most of the *ecalls* are called only during initialisation of OpenVPN and Click. ENDBOX defines only 4 *ecalls* that are executed during normal operation: (i) packet en- and decryption; and (ii) message authentication code (MAC) generation and verification. While (i) are triggered by normal traffic, (ii) are used for integrity protection of the OpenVPN control channel. With the exception of the ENDBOX-specific en- and decryption and Click initialisation *ecalls*, all *ecalls* match the TaLos/LibreSSL library calls, which perform security checks. The *ocalls* perform different tasks, among them managing untrusted memory and accessing (encrypted) configuration files. Note that they could be omitted by using in-enclave configuration files and exitless enclave services [38].

To ensure a secure interface, we closely examined all *ecalls* and *ocalls* and augmented them with sanity checks on input (resp. return) values of *ecalls* (resp. *ocalls*), and bound checking of pointers either passed to *ecalls* or returned from *ocalls* to guarantee that they point to enclave memory.

V. EVALUATION

We evaluate the security and performance of ENDBOX by discussing different attacks on ENDBOX and performing different measurements. Our results show that: (i) ENDBOX is secure against a wide range of attacks (§ V-A); (ii) it only affects network latency in a minimal way (§ V-C); (iii) it induces an acceptable best-case performance overhead of 16% (§ V-D); (iv) it scales linearly with the number of clients; (v) clients can achieve a 2.6×–3.8× higher throughput than a traditional centralised middlebox (§ V-E); (vi) our runtime reconfiguration mechanism has a 30% lower latency than the original Click implementation (§ V-F); and lastly, (vii) our optimisations described in § IV-A actually improve ENDBOX's impact on latency or throughput (§ V-G).

A. Security evaluation

Following an exhaustive evaluation of our threat model, we discuss typical attacks against ENDBOX and state how it can defend against these or why they are not applicable.

Bypassing middlebox functions. A malicious client may try to access the network without using ENDBOX. We assume that the network is guarded by a static firewall limiting traffic to VPN usage: without a properly configured ENDBOX client establishing a valid VPN connection, it is not possible for an attacker to send valid traffic that would bypass the middlebox functions in ENDBOX. Instead, the traffic will be dropped by the firewall. For incoming traffic, clients are *indirectly* forced to route their traffic through the ENDBOX client if they want to access the encrypted payload. ENDBOX ensures the authenticity of connections using remote attestation (§ III-C).

Using old or invalid middlebox configurations. An attacker may rollback configuration updates, or use unauthorised configurations. Once an adjustable grace period for an update

has passed, the server only accepts ENDBOX clients that use the currently valid configuration, as described in § III-E. The ENDBOX client and server periodically exchange ping messages containing configuration information to prevent clients from using stale configurations.

Replaying traffic. If a malicious client replays traffic, e.g. in order to establish a connection without a genuine enclave, the ENDBOX server detects this, due to OpenVPN's implementation of packet replay protection.

Denial-of-service attacks. Malicious clients can prevent enclaves from starting or being entered, as the enclave life cycle is managed by untrusted code. However, this would result in the inability of the client to communicate with the network. In contrast, a denial-of-service attack on the ENDBOX server would have the same effect as a traditional centralised middlebox deployment, thus, the attack can be mitigated using classical techniques [32].

Downgrade attacks. Attackers could try to force the usage of a weaker TLS version or cipher. However, OpenVPN implements server-side checks that ensure the minimal TLS version to be used. On the client-side, the corresponding check happens within the enclave during connection establishment and therefore cannot be circumvented.

Interface attacks. A client may try to break into the enclave by manipulating the parameters at the enclave interface similar to Iago attacks [39]. To mitigate such attacks, every *ecall* and *ocall* has been augmented with checks on input parameters and return values (see § IV-B). In addition, ENDBOX exposes a limited interface with a restricted attack surface.

Failure of a middlebox. If a middlebox fails, only the client running this middlebox is impacted; other clients and the managed network remain unaffected. This differs from the behaviour of traditional centralised middlebox set-ups in which a failure would affect many clients or even whole networks. In contrast, the failure of the ENDBOX server managing all VPN connections is equivalent to a failure of traditional centralised middleboxes, resulting in network outages.

B. Experimental set-up and use cases

We evaluate the performance of ENDBOX on a cluster of seven machines of two classes. Class Ⓐ consists of five machines, equipped with SGX-capable 4-core Xeon v5 CPUs with 32 GB of memory, while class Ⓑ are two machines with non-SGX 4-core Xeon v2 CPUs and 16 GB of memory. All machines are configured with hyper-threading and are connected to a 10 Gbps switch via two 10 Gbps network interfaces per machine. The maximum transmission unit (MTU) of the network links is configured to 9000 bytes. We conduct the throughput measurements using `iperf`, while for latency measurements we rely on ICMP pings. Throughout this section, we evaluate multiple set-ups, including these reoccurring ones: (i) *vanilla OpenVPN*, an unmodified OpenVPN v2.4.0; (ii) *OpenVPN+Click*, the same OpenVPN version, but traffic is processed by server-side Click instances; (iii) ENDBOX *in simulation mode* to show the overhead of partitioning the VPN client; and (iv) ENDBOX *in hardware mode* to show the overhead of using SGX instructions. Throughout this section, we report average values of 10 consecutive runs; the variance of results is omitted if the reported error is negligible.

In the following, we describe five middlebox functions we implemented for the evaluation. They are either based on standard or custom Click elements.

Forwarding (`NOP`). The first middlebox function we consider provides a baseline for our measurements. It forwards packets without accessing or modifying any headers or payloads.

Load balancing (`LB`). The `RoundRobinSwitch` Click element allows us to balance IP packets or TCP flows across several machines, thus balancing load.

IP firewall (`FW`). A firewall accesses packet headers and controls traffic based on a set of rules. We use the `IPFilter` Click element without any code modifications. For our evaluation we use a set of 16 rules that do not match any packet.

Intrusion detection and prevention system (`IDPS`). An IDPS monitors network traffic for unauthorised accesses and policy violations. We support Snort [40] rule sets and execute its string matching algorithm [41] using a library from [42]. The IDPS is implemented as a custom Click element called `IDSMatcher`. For the evaluation, we use a subset of 377 rules of the Snort community rule set. Again, the rules do not match packets generated for our evaluation.

DDoS prevention (`DDoS`). Distributed denial-of-service attacks can generally be mitigated by throttling or dropping packets that occur repeatedly or if source address spoofing is detected. We implement this middlebox function by rate limiting identical packets using our custom Click elements `IDSMatcher` and `TrustedSplitter`. The latter allows the shaping of traffic to a given bandwidth in a trusted way: to reduce expensive calls to obtain trusted time, the `TrustedSplitter` samples timestamps by issuing calls after a certain configurable number of packets has been processed. This number is set to 500,000 for our measurements. For *OpenVPN+Click*, we use a similar Click element called `UntrustedSplitter` which obtains timestamps using system calls. This use case is well-suited for the ISP scenario, as it enables the provider to detect malware or bot nets directly on client-side.

C. Latency

In the following, we evaluate the latency impact of ENDBOX, as this has a notable influence on user experience. We use the forwarding middlebox function (`NOP`) and perform local experiments using class Ⓐ machines. For cloud-based measurements, we rely on Amazon Web Services (AWS) Elastic Compute Cloud (EC2) and use `m3.medium` instances with 1 virtual CPU and 3.75 GB RAM in different regions.

HTTP request handling. ENDBOX's impact on latency can be observed in Fig. 6, which plots the cumulative distribution function (CDF) for HTTP page load times of 1,000 popular websites provided by Alexa [43]. Results show that the time needed to load these websites is very similar when using

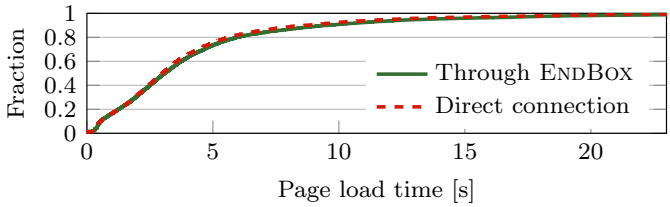


Fig. 6: CDF of HTTP page load times for Alexa top 1,000 sites with and without EndBox

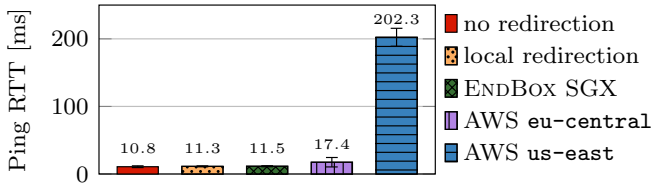


Fig. 7: Average ping RTT for different redirection methods

EndBox or a direct connection, and hence that the latency overhead of EndBox is negligible.

Traffic redirection. By further exploring EndBox's impact on latency, we want to show that offloading middlebox functions to cloud environments has more disadvantages than only losing control and trust. Inspired by [4], we create a setup of software middleboxes executing in AWS EC2 and measure the ping round-trip times (RTTs) to a fixed location. Fig. 7 shows the average ping RTT for different redirection methods: (i) no redirection with no middlebox or VPN; (ii) local redirection through a VPN and server-side middlebox using *OpenVPN+Click*; (iii) redirection trough EndBox; and (iv) redirection through middleboxes deployed on EC2 instances in different AWS regions, also using *OpenVPN+Click*. The results show that depending on the location a cloud provider chooses, the latency overhead ranges between 61% and 1773% and that EndBox's latency overhead is only 6%.

Handling of encrypted traffic. As mentioned in III-D, EndBox is able to transparently decrypt TLS traffic. We measure the overhead of this functionality by letting an HTTPS client fetch static web pages of different sizes from a web server. This client is using one configuration among: (i) EndBox with custom OpenSSL and traffic decryption inside Click; (ii) EndBox with custom OpenSSL but without traffic decryption; or (iii) EndBox with system OpenSSL and without traffic decryption. We measure the HTTPS `GET` request latency, and report the results in Table I. They show that the overhead introduced by our custom OpenSSL and traffic decryption is less than 8%. The two sources of overhead are EndBox's custom OpenSSL forwarding of keys to the enclave, and the actual decryption.

	EndBox OpenSSL		vanilla OpenSSL
Resp. size	w/ dec	w/o dec	w/o dec
4 KB	1.08 ms	1.04 ms	1.00 ms
16 KB	1.34 ms	1.29 ms	1.26 ms
32 KB	1.78 ms	1.75 ms	1.70 ms

TABLE I: HTTPS `GET` request latency for different response sizes and configurations

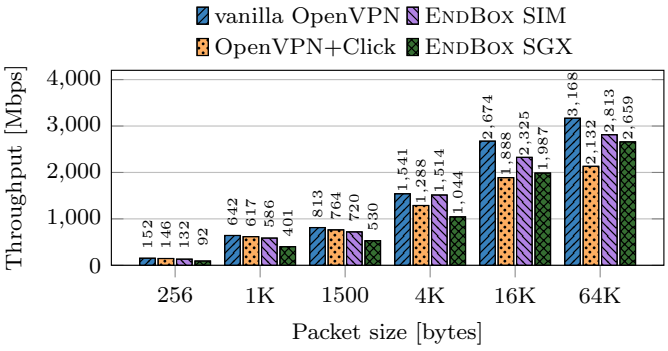


Fig. 8: Average maximum throughput of different set-ups for packet sizes 256 bytes to 64 kilobytes

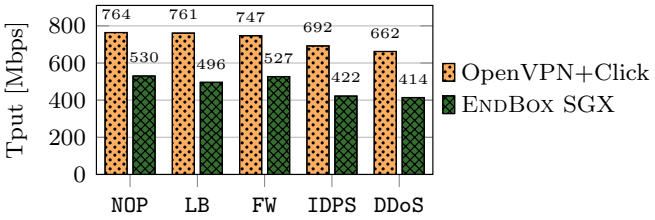


Fig. 9: Average maximum throughput of `NOP`, `FW`, `LB`, `IDPS` and `DDoS` use cases for OpenVPN+Click and EndBox with a packet size of 1500 bytes

D. Throughput

Besides network latency, throughput performance is also an important parameter impacting user experience. All these measurements are performed on two class Ⓐ machines.

Packet size. In this experiment, we measure the maximum throughput reached in different configurations for various packet sizes from 256 bytes to 64 kB. We compare four set-ups: (i) *vanilla OpenVPN*, (ii) *OpenVPN+Click*: the same OpenVPN version with an attached server-side Click instance; (iii) EndBox *in simulation mode*; and (iv) EndBox *in hardware mode*. The results are represented in Fig. 8. As expected, the throughput increases for all configurations as the payload size increases. Moreover, we see that EndBox has an acceptable performance overhead: It varies between 2% and 13% for EndBox in simulation mode. Using actual SGX instructions (hardware mode) adds overhead, resulting in a worst-case overhead of 39% for small packets, but in a best-case overhead of only 16% for large packets. This is due to the fact that larger packets allow higher throughput with less enclave transitions. We also see that a server-side Click instance has an average performance penalty of 26%; values range between 5% and 29% depending on packet size. Finally, we observe that, for large packets, a server-side Click instance achieves a throughput almost one third lower than vanilla OpenVPN due to the Click instance's packet fetching.

Middlebox functions. Fig. 9 shows the average maximum throughput achieved by a traditional middlebox set-up with VPN compared to EndBox. We evaluate all middlebox functions presented in § V-B using one client machine and a medium packet size of 1500 bytes.

With NOP representing a baseline, we first observe that the impact of a Click configuration on OpenVPN+Click is rather low: in the worst case, for the DDoS prevention use case, the throughput drops by 13%, from 764 Mbps to 662 Mbps. Second, ENDBOX incurs around 30% overhead for the use cases NOP, LB, and FW. The more computation intensive use cases IDPS and DDoS have an overhead of 39%. Note that this overhead is lower for larger packets, as shown in Fig. 8.

Summary. Results show that ENDBOX introduces an acceptable throughput overhead of only 16% for large packets in the NOP use case. For medium sized packets, the overhead is 30% regarding lightweight middlebox functions and 39% for specific heavy-duty use cases. As expected, we observe that the throughput of ENDBOX increases with the size of packets. Furthermore, ENDBOX does not have any user-perceivable impact on the latency of HTTP page load times. As a result, from a performance perspective, ENDBOX is a viable alternative to existing middlebox deployments.

E. Scalability

After evaluating user-facing properties of ENDBOX like latency and throughput, we evaluate the scalability of ENDBOX, which is important to its operators. Therefore, we measure the throughput and CPU usage on server side. The throughput is aggregated over all virtual interfaces set up by the OpenVPN servers, which is one per client. The CPU usage applies for all cores, i.e. 100% represents all cores being fully utilised. The scalability measurements use five class Ⓐ machines to execute multiple ENDBOX clients and two class Ⓑ machines, each running the ENDBOX server or `iperf` servers. For the measurement, we compare four set-ups: (i) *vanilla OpenVPN* without middlebox function as baseline; (ii) ENDBOX *in hardware mode*; (iii) *vanilla Click* on server side without encryption; and (iv) *OpenVPN+Click*: multiple server-side vanilla Click instances attached to OpenVPN servers. In these experiments, each client generates a workload of 200 Mbps. For (i), (iii) and (iv), we use one OpenVPN server instance per client, as OpenVPN does not support multithreading.

First, we evaluate the scalability using a forwarder as middlebox function (NOP). The results in Fig. 10a show that vanilla OpenVPN and ENDBOX achieve the same throughput of 6.5 Gbps at an almost identical CPU usage. This shows that client-side execution of middleboxes has no impact on throughput or CPU usage on server side. For OpenVPN+Click, the bottleneck is the CPU, which is fully utilised earlier than with ENDBOX, because Click requires a substantial amount of cycles. In contrast, the throughput of vanilla Click is limited to 5.5 Gbps by the Click process which cannot handle more packets. Finally, our measurements report an even lower throughput for OpenVPN+Click of 2.5 Gbps, which continuously decreases with a growing number of clients, as OpenVPN+Click is limited by the servers' CPUs.

Use case evaluation. We conduct the same measurement for our five use cases presented in § V-B. In Fig. 10b we use the results for OpenVPN+Click and ENDBOX from the previous measurements as baselines and show how ENDBOX scales with the number of clients when different middlebox configurations are applied. When network traffic en- and decryption fully utilises the VPN server (at 40 clients with our machine) it becomes the bottleneck of ENDBOX: we observe a maximum throughput of 6.5 Gbps for all use cases. Due to the server-side execution of middlebox functions, OpenVPN+Click reaches this limit earlier at 30 clients with a maximum throughput of 2.5 Gbps FW and LB use cases. The computation intensive IDPS and DDoS middlebox functions only achieve 1.7 Gbps.

Our evaluation shows that ENDBOX scales linearly with the number of clients. Additionally, for 60 clients, ENDBOX achieves a 2.6× higher throughput across all use cases, and 3.8× for computation intensive workloads induced by IDPS and DDoS. This is not a general limitation of ENDBOX—it is due to our evaluation setup and caused by the computation-intensive nature of pattern matching on network packets, which overloads central middleboxes faster. Thus, we show that ENDBOX performs especially well for CPU intensive middlebox functions.

Summary. Results indicate that ENDBOX scales linearly with the number of clients until the VPN server is fully utilised. They also show that by executing middlebox functions on the client side, ENDBOX can achieve 2.6× to 3.8× higher throughput than centrally deployed middleboxes, depending on the use case.

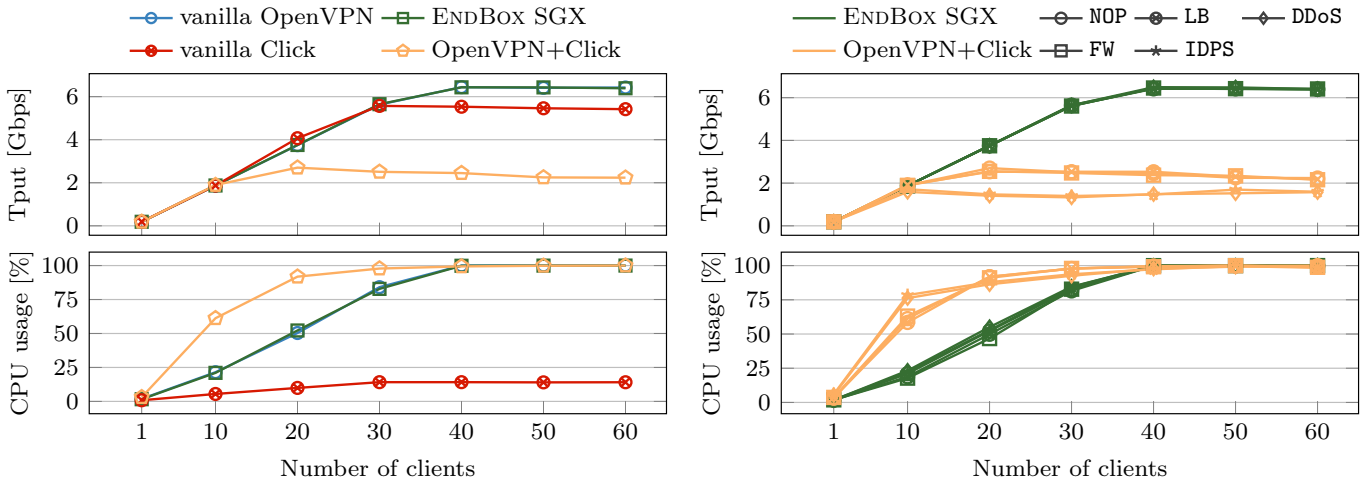
F. Reconfiguration overhead

One advantage of deploying middleboxes centrally is a simple configuration update mechanism. For ENDBOX, this is far more challenging, as middleboxes are distributed across untrusted client machines. Therefore, ENDBOX implements mechanisms to apply configuration updates across all client-side middleboxes in a secure way and enables administrators to verify that correct configurations are applied, as described in § III-E.

Phase	vanilla Click	ENDBOX
fetch	-	0.86 ms
decryption	-	0.07 ms
hotswap	2.4 ms	0.74 ms
Total	2.4 ms	1.67 ms

TABLE II: Timings of different phases of vanilla Click and ENDBOX configuration updates

Breakdown of an update operation. Table II shows the different phases of configuration updates performed by vanilla Click and ENDBOX. We use a minimal configuration file with a size of 42 and 59 bytes, respectively. Since vanilla Click does not need to fetch and decrypt the configuration file, the only operation is hotswapping the configuration, which takes 2.4 ms in average. In contrast, ENDBOX spends in average 0.86 ms for fetching the new configuration and 0.07 ms for decrypting the new configuration. However, both operations do not influence the traffic filtering of ENDBOX and are performed in the background. Finally, it takes 0.74 ms for hotswapping the configuration. Thus, ENDBOX requires only 30% of the time for the actual reconfiguration compared to vanilla Click. This is due to the fact that vanilla Click needs to set up file



(a) NOP use case applied to different middlebox deployments

(b) Five middlebox functions for OpenVPN+Click and EndBox

Fig. 10: Server-side aggregated throughput and CPU usage of (a) different middlebox deployments and (b) specific use cases

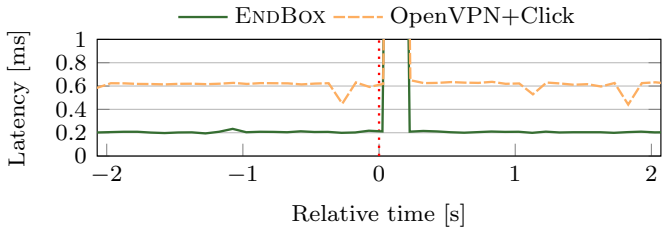


Fig. 11: Impact of configuration updates on ping latency shown for FW use case, time of reconfiguration at 0 seconds

descriptors for the ToDevice and FromDevice elements, which is not necessary for EndBox because OpenVPN took care of this task earlier.

Latency impact of updates. Additionally, we compare the impact of configuration updates with respect to latency of two set-ups: EndBox and OpenVPN+Click, both applying our firewall use case. In our experiment, a single client sends periodic pings at a rate of 10 requests per second and we measure the round trip time. As shown in Fig. 11, we observe that both OpenVPN+Click and EndBox lose one single ping packet during reconfiguration. This shows that the overhead of *distributed* compared to *local* reconfiguration is negligible if implemented correctly.

G. Evaluation of Optimisations

Finally, we evaluate impact of the optimisations described in § IV-A either on throughput or latency. Reducing the number of enclave transitions per packet results in a substantially higher throughput of 342%, while refraining from packet encryption in the ISP scenario leads to a 11% higher throughput. In contrast, optimising the client-to-client communication has no effect on throughput, but decreases the latency between clients by up to 13% for the IDPS use case.

VI. Related Work

We are not the first to advocate the benefits of moving middleboxes to end hosts, e.g. [6], [7], [20], [44]. However, the vast majority of these solutions assumes trusted end hosts and, hence, they are not suitable for a client-side deployment like the ones targeted in this paper because users have full physical access to the machine and cannot be trusted.

One notable exception is ETTM [20], which relies on a trusted platform module (TPM). This approach is inflexible because it only supports attestation at bootstrap time and lacks integrity checks during execution. Most importantly, it does not protect against malicious users with physical access to the machine as EndBox does. Further, ETTM is impractical because it requires the entire hypervisor to be part of the TCB; and physical network hardware to correctly forward traffic. While assuming that network hardware is trusted may be conceivable for enterprise settings, it is infeasible in the ISP scenario. Finally, the design of ETTM follows a distributed approach that does not involve trusted configuration servers as EndBox does. Therefore, ETTM applies Paxos [45] for consensus, but Paxos does not scale well [46], induces high latencies, and is not applicable when mobile nodes with an unstable connection are involved, as discussed in our enterprise scenario. Other proposals such as Eden [6] rely on specialised hardware on end hosts to implement middlebox functionality. While these solutions can achieve higher performance than EndBox, their hardware exceed the specifications of today's laptops and average desktops, and, hence, do not meet the requirements of our scenarios.

Middlebox functionality can be entirely moved to the cloud [4], [5], [47]. This solution avoids the risk of users mounting physical attacks and can provide great scalability. These benefits, however, come at the cost of increased expenses and higher latency due to traffic redirection (see § V-C). Further, outsourcing traffic processing entails security risks as well as privacy and legal issues.

Executing middlebox functions inside SGX enclaves has been proposed [48]–[51]. Contrary to EndBox, these systems are not designed to be deployed on clients. Instead, they execute entire middleboxes or specific functions in the cloud to guarantee integrity and confidentiality of network traffic.

As detailed in § III-D, ENDBOX is able to execute middlebox functions on encrypted traffic. The following four proposals also target this problem. BlindBox [15] presents an encryption scheme to perform a limited set of computations on encrypted traffic, but at a much lower cost than traditional homomorphic encryption. In mcTLS [13] and mbTLS [14] packets are encrypted in a way such that middleboxes that require access can decrypt them. SGX-Box [52] utilises SGX on centralised middleboxes to enable DPI on encrypted network traffic. Similarly to ENDBOX, TLS session keys are securely shared with the enclave.

VII. CONCLUSION

In this paper, we presented ENDBOX, a scalable system that enables the secure deployment and execution of middlebox functions on untrusted client machines. For typical middlebox functions, it scales linearly with the number of clients, thereby achieving a 2.6× to 3.8× higher throughput than a traditional deployment at the core of a managed network. Despite being distributed, configuration changes to ENDBOX-based middlebox services are centrally controlled and enforced. Finally, encrypted application traffic can be efficiently and securely decrypted and filtered using ENDBOX, due to its location at the client side.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable feedback. This work has received funding from the EU's Horizon 2020 research and innovation programme under grant agreements 645011 (SERECA) and 690111 (SecureCloud).

REFERENCES

[1] Cisco Visual Networking Index, "The zettabyte era–trends and analysis," *Cisco white paper*, 2013.
[2] Kaspersky Lab, "Global IT Security Risks Survey 2014 – Distributed Denial of Service (DDoS) Attacks," https://goo.gl/dbg3wZ.
[3] Verisign Blog, "Verisign Q1 2016 DDos Trends: Attack Activity Increases 111 Percent Year Over Year," https://goo.gl/Srm3cW.
[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy *et al.*, "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service," in *ACM SIGCOMM'12*.
[5] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely Outsourcing Middleboxes to the Cloud," in *USENIX NSDI'16*.
[6] H. Ballani, P. Costa, C. Gkantsidis, M. P. Grosvenor *et al.*, "Enabling End-Host Network Functions," in *ACM SIGCOMM'15*.
[7] W. Zhang, G. Liu, A. Mohammadkhan, J. Hwang *et al.*, "SDNFV: Flexible and Dynamic Software Defined Control of an Application- and Flow-Aware Data Plane," in *Middleware'16*.
[8] M. Feilner, *OpenVPN: Building and integrating virtual private networks*. Packt Publishing Ltd, 2006.
[9] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, 2000.
[10] EFF, "We're Halfway to Encrypting the Entire Web," https://goo.gl/VdUj5b, 2017.
[11] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger *et al.*, "The cost of the S in HTTPS," in *ACM CoNEXT'14*.
[12] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *IEEE S&P 2014*.
[13] D. Naylor *et al.*, "Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS," in *ACM SIGCOMM'15*.
[14] D. Naylor *et al.*, "And then there were more: Secure communication for more than two parties," in *CoNEXT'17*.
[15] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep Packet Inspection over Encrypted Traffic," in *ACM SIGCOMM'15*.
[16] 1&1 Internet Ltd., "IP Spoofing: Simple manipulation of data packets by attackers," https://goo.gl/Dn1CaV, 2017.
[17] D. Kreutz *et al.*, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, 2015.
[18] B. Han *et al.*, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, 2015.
[19] J. Soares *et al.*, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, 2015.
[20] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon *et al.*, "ETTM: a scalable fault tolerant network manager," in *USENIX NSDI'11*.
[21] S. Gueron, "A Memory Encryption Engine Suitable for General Purpose Processors." *IACR Cryptology ePrint Archive*, 2016.
[22] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *HASP'13*.
[23] S. Arnautov, B. Trach, F. Gregor, T. Knauth *et al.*, "SCONE: Secure Linux Containers with Intel SGX," in *USENIX OSDI'16*.
[24] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX," in *USENIX ATC'17*.
[25] S. Shinde, D. L. Tien, S. Tople, and P. Saxena, "PANOPLY: Low-TCB Linux Applications With SGX Enclaves," in *NDSS'17*.
[26] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt *et al.*, "SecureKeeper: Confidential ZooKeeper using Intel SGX," in *Middleware'16*.
[27] Y. Xu, W. Cui, and M. Peinado, "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems," in *IEEE SP'15*.
[28] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves," in *ESORICS'16*.
[29] J. Van Bulck, F. Piessens, and R. Strackx, "SGX-Step: A practical attack framework for precise enclave execution control," 2017.
[30] J. Seo, B. Lee, S. Kim, M.-W. Shih *et al.*, "SGX-Shield: Enabling address space layout randomization for SGX programs," in *NDSS'17*.
[31] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating controlled-channel attacks against enclave programs," in *NDSS'17*.
[32] A. Garg and A. N. Reddy, "Mitigation of DoS attacks through QoS regulation," *Microprocessors and Microsystems*, 2004.
[33] H. Nguyen and V. Ganapathy, "EnGarde: Mutually-Trusted Inspection of SGX Enclaves," in *IEEE ICDCS'17*.
[34] M. Green, R. Droms, R. Housley, P. Turner, S. Fenter, "Data Center use of Static Diffie-Hellman in TLS 1.3," https://goo.gl/95FaWD.
[35] E. Rescorla, "Update on TLS 1.3 Middlebox Issues," https://goo.gl/zCUuRG, 2017.
[36] Intel Corp, "Intel Software Guard Extensions for Linux OS (Intel SGX) SDK," https://01.org/intel-software-guard-extensions, 2017.
[37] P.-L. Aublin, F. Kelbert, D. O'Keeffe, D. Muthukumaran *et al.*, "TaLoS: Secure and Transparent TLS Termination inside SGX Enclaves," Imperial College London, Tech. Rep. 2017/5, Mar. 2017.
[38] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein, "Eleos: ExitLess OS Services for SGX Enclaves," in *EuroSys '17*.
[39] S. Checkoway and H. Shacham, "Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface," in *ASPLOS'13*.
[40] M. Roesch, "Snort: Lightweight Intrusion Detection for Networks." in *USENIX LISA'99*.
[41] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, 1975.
[42] W. Sun and R. Ricci, "Fast and flexible: Parallel packet processing with GPUs and Click," in *ACM/IEEE ANCS'13*.
[43] Alexa., http://www.alexa.com/, 2017.
[44] T. Karagiannis *et al.*, "Network Exception Handlers: Host-network Control in Enterprise Networks," in *ACM SIGCOMM'08*.
[45] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, 2001.
[46] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *iNetSec'15*.
[47] X. Yuan, X. Wang, J. Lin, and C. Wang, "Privacy-preserving deep packet inspection in outsourced middleboxes," in *IEEE INFOCOM'16*.
[48] H. Duan, X. Yuan, and C. Wang, "LightBox: SGX-assisted Secure Network Functions at Near-native Speed," *arXiv:1706.06261*, 2017.
[49] M. Coughlin, E. Keller, and E. Wustrow, "Trusted Click: Overcoming Security issues of NFV in the Cloud," in *ACM SDN-NFV Security'17*.
[50] D. Kuvaiskii, S. Chakrabarti, and M. Vij, "Snort Intrusion Detection System with Intel Software Guard Extension," *arXiv:1802.00508*, 2018.
[51] B. Trach *et al.*, "ShieldBox: Secure Middleboxes using Shielded Execution," in *ACM SOSR'18*, 2018.
[52] J. Han, S. Kim, J. Ha, and D. Han, "SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module," in *ACM APNet'17*.