Riaz, Maria; Tilli, Juha-Matti; Kantola, Raimo

Sec-ALG

Published in:
Proceedings of the 29th International Conference on Computer Communications and Networks, ICCCN 2020

DOI:
10.1109/ICCCN49398.2020.9209718

Published: 01/08/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Riaz, M., Tilli, J.-M., & Kantola, R. (2020). Sec-ALG: An Open-source Application Layer Gateway for Secure Access to Private Networks. In *Proceedings of the 29th International Conference on Computer Communications and Networks, ICCCN 2020* Article 9209718 (Proceedings : International Conference on Computer Communications and Networks). IEEE. https://doi.org/10.1109/ICCCN49398.2020.9209718

Sec-ALG: An Open-source Application Layer Gateway for Secure Access to Private Networks

Maria Riaz, Juha-Matti Tilli, Raimo Kantola
Department of Communication and Networking
Aalto University
Espoo, Finland
e-mail: firstname.lastname@aalto.fi

***Abstract*—Middleboxes such as Network Address Translators (NATs), proxy servers or Application Layer Gateways (ALGs) provide remote access to end-hosts in the private address space. The middleboxes offer proprietary solutions and encrypted traffic poses a challenge when middleboxes employ packet payload inspection techniques for connection establishment. Session key sharing and decryption followed by re-encryption of the traffic, for correctly routing to the private host, increases the connection latency and also poses a higher threat in case of traffic interception by a malicious third-party.**

In this paper, we present a novel open-source ALG, called Sec-ALG, for providing secure end-to-end communication to the web servers situated in the private address space. Sec-ALG relies on the technique of light Deep Packet Inspection (DPI) for protocol detection and session establishment using a novel parser-lexer generator called YaLe. The proposed approach offers increased security by maintaining end-to-end encryption for an HTTPS connection. Our experimental analysis demonstrates that Sec-ALG reduces the HTTPS connection latency in comparison to the NGINX reverse proxy using a 24-core host machine. Moreover, Sec-ALG handles requests at a three-fold increased rate than NGINX proxy when tested with 100 concurrent connections. The ALG can be used either as a standalone solution or a component of the Realm Gateway, that is a generic interworking solution between public and private networks. The presented work is part of an extensive ongoing research at Aalto University focusing on embedding policy based trust into the network.

I. Introduction

With the increase in complexity and scaling of the network, the Internet invented more than 50 years ago required middleboxes to cater various network challenges related to security, e.g., intrusion detection/prevention, optimization of performance, e.g., load balancing and scaling to ever increasing number of end devices. A Network Address Translator (NAT) [1] middlebox was proposed to solve the issue of scalability to the growing number of connected devices with the limited IPv4 (Internet Protocol version 4) address space. However, the traditional NAT causes a reachability problem whereby the host in the public domain is unable to initiate a connection towards a private network host.

Several NAT traversal mechanisms have been suggested to provide access to private networks over the years but no particular solution has been ubiquitously adopted. The existing NAT traversal solutions either require a separate Application Layer Gateway (ALG) for handling application layer traffic through NAT, bypass the firewall/NAT functionality for NAT unfriendly protocols, or require polling of the network on application layer making them undesirable. After careful analysis of the existing NAT traversal mechanisms we developed a custom NAT solution called Realm Gateway (RGW) [2] [3] for establishing end-to-end connectivity between the hosts in public address realm and private address realm. RGW acts as a traditional Source NAT (SNAT) for the outbound connections and behaves as a Destination NAT (DNAT) for inbound connections. It also operates as an authoritative Domain Name System (DNS) server for all the hosts it serves. Using a circular pool of publicly reachable IP addresses, RGW creates a dynamic binding between the hosts in the public domain and private domain in response to a DNS query.

Our custom NAT gateway is compatible with most of the protocols, however, an additional network component is needed for processing web based communication. The web browsers can initiate multiple parallel requests to retrieve the embedded content for a web page and that might result in stalling of connections if only one DNS query is sent by the web browser's DNS server for the multiple HTTP/S requests. Integrating third-party middleboxes with RGW for traversal of most popular application layer protocols for data communication over the web namely, Hyper Text Transfer Protocol (HTTP) and its encrypted counterpart, Hyper Text Transfer Protocol Secure (HTTPS) is one approach. However, these proprietary middleboxes employ the technique of Deep Packet Inspection (DPI) and packet modification for forwarding the traffic to the correct private host behind a NAT thus incurring high management and infrastructure costs [4]. There are also recent proposals to outsource the middleboxes to run in the cloud [4] [5]. Using cloud computing for middlebox services offers performance benefits but trusting third party providers with user-sensitive data leads to potential security vulnerabilities. The paradigm shift towards encrypted network traffic imposes additional challenges in processing the packet payload in the middleboxes. As a result, either the private keys of the host need to be shared with the middlebox for payload inspection, or the functionality of middlebox has to be disabled for encrypted communication.

To address the issue of handling encrypted traffic by the middleboxes a number of different solutions have

been proposed. These proposals range from adopting new protocol and encryption schemes while maintaining end-to-end encryption such as Blindbox [6], modifying the existing protocols like Transport Layer Security (TLS) to make them compatible with middlebox functionality such as mcTLS [7], or sharing of session keys using a key-sharing protocol [8]. We argue that the adoption of a new protocol and encryption scheme as discussed in Blindbox is not a viable solution because the packet inspection is done by comparing the keywords in the encrypted payload against a limited proprietary rule set not accessible to everyone. Modifying the existing security protocols like TLS as proposed in mCTLS which have been widely deployed on most web servers requires acceptance as a standard by Internet Engineering Task Force (IETF) before it can be widely adopted. Furthermore, sharing of session keys by the user devices through a secure channel as in [8] increases the connection latency and also poses the question of trusting the middlebox with the session keys and gives them access to plaintext network traffic.

In this paper, we present Sec-ALG, a novel open-source ALG for providing connectivity to devices located behind a NAT while maintaining end-to-end encryption. Our solution has the ingenuity where the private end host is not required to exchange certificates or private keys with Sec-ALG acting as the middlebox in case of encrypted traffic. Moreover, our solution only requires changes at the NAT middlebox and no end host protocol stack modification is required. To solve the problem, Sec-ALG uses the technique of light DPI for the detection of application layer protocol and hostname of the destined web server during the session establishment phase. We use our novel parser-lexer generator called YaLe [9] for detecting the domain name of the private web server from the packet payload. The Server Name Indication (SNI) extension [10] exchanged at the start of TLS handshake includes the plaintext name of the server requested by the client. Since TLS is the underlying protocol for HTTPS, a YaLe created parser identifies the destined server's hostname using the grammar we wrote for TLS. All the popular desktop browsers of today, namely Internet Explorer, Mozilla Firefox, Google Chrome, Opera and Safari support TLS version 1.2 with the SNI extension in their latest versions which makes it easy to adopt. We ensure middlebox transparency by logging all event details for a configurable time aiding the auditing process in case an anomaly or attack is detected later.

To improve the flexibility and allow customization for administrative purposes, the policies of Sec-ALG are stored in and retrieved from a policy management system called Security Policy Management (SPM) [11]. We observe the effects of the proposed extensions on RGW's performance by using various benchmark tools. We validate our design by comparing the performance with a well-known web proxy server called NGINX.

The main contributions of this paper are:

- Proposing a mechanism for TLS middleboxes to achieve end-to-end connectivity while maintaining encryption.
- Developing a prototype of Sec-ALG for interworking with a custom NAT solution called Realm Gateway.
- Improving the usability of Sec-ALG and RGW by integrating them with a policy management system.

The paper is organized as follows: Section II describes related work in the context of middleboxes and encrypted traffic. Section III gives a brief overview of RGW. The proposed architecture of Sec-ALG is presented in Section IV. We present a potential use case for our proposed system in Section V. In Section VI, we discuss our experimental setup and performance evaluation of Sec-ALG after integration with RGW. After a discussion in Section VII, we conclude the paper in Section VIII.

II. Related work

A. NAT Middleboxes

Multiple approaches are discussed in [2] that allow devices behind NATs to become reachable for hosts in the public domain. An overview of the existing NAT traversal solutions including Session Traversal Utilities for NAT (STUN), Traversal Using Relays around NAT (TURN) and User Datagram Protocol (UDP) hole punching is presented in [12]. Many of the existing solutions either require changes at the end hosts, deployment of additional servers in order to establish end-to-end connectivity, or both. Furthermore, forwarding in NAT can be susceptible to spoofed incoming flows or flooding from the botnets which require additional components to filter the incoming flows. In [2] [3], we proposed our NAT traversal solution called RGW that admits incoming flows based on security policies customizable by the users based on the traffic they wish to accept. However, RGW requires an additional component when dealing with web traffic as discussed in Section I. In this paper, we propose Sec-ALG that handles the HTTP/S traffic at the RGW. It can also be used as a stand-alone component by the edge nodes for forwarding the traffic to the back-end web servers.

B. Middlebox Solutions for Encrypted Traffic

Middleboxes have adopted different methods for inspecting traffic sent over HTTPS, the de facto protocol for encrypted traffic over the web. Outsourcing the middlebox functionality to the cloud for achieving higher performance has been proposed in [4] [13] but it has a number of security implications. Using a transparent SSL/TLS proxy between the client and server by breaking down the session into two segments is the most commonly used approach but it poses a number of problems as discussed in [14] [15]. The splitting of TLS session requires first decryption of the traffic using the web servers' private keys before it can be inspected, re-encrypted and forwarded to the upstream web server, which makes it unviable. Research carried out in [16] [17] indicate that TLS proxies are being used for injecting malicious code when they get access to decrypted user traffic. These violations raise security concerns among the users about the handling of their data by the service providers [18].

There are some proposals that allow network traffic processing while maintaining encryption such as Blindbox [6]. These solutions use the technique of DPI and use a limited proprietary rule set for finding matching patterns not freely accessible. Moreover, the adoption of a new encryption protocol as proposed in Blindbox requires modification in the application protocol stack of all the host systems.

III. Realm Gateway

We discussed server end NAT traversal using RGW in [2] for connecting hosts located in the public realm with the private realm. In contrast to other proposals discussed in Section II-A, RGW is deployable one network at a time and does not require any host stack modifications in kernel or user space. To provide compatibility with Internet Protocol version 6 (IPv6), the design of RGW was further enhanced and presented in [19]. In addition to acting as a traditional SNAT for outbound connections from private hosts, RGW uses a pool of globally accessible public IP addresses for establishing inbound connections from the public network hosts to the servers located behind RGW.

Fig. 1 illustrates the system architecture of RGW where the data plane functionality can be handled by a full Python-based implementation or Open Virtual Switch (OVS). RGW acts as a default gateway for accessing the Internet and uses the Circular Pool Address (CPA) algorithm for inbound traffic. Furthermore, it acts as an authoritative name server for the designated DNS zone. A dynamic binding is established using an available IP address from the circular pool in response to the DNS query sent by the public host. The DNS query contains the Fully Qualified Domain Name (FQDN) of the destined private host. We introduced the concept of Service Fully Qualified Domain Name (SFQDN) in [19] when the DNS query also includes the service to be requested on the domain name. For example, a web service running on the host can be represented as `www.host.gwa.demo` or alternatively as `tcp80.host.gwa.demo` and `tcp443.host.gwa.demo`.

A connection state is maintained by RGW against each public and private host pair and a timeout is associated with each connection attempt and with each established connection. The address given in response to DNS query is released back to the circular pool for next allocation after a certain time called the holding timeout, or upon arrival of a legitimate TCP flow having a non-spoofed source IP address. Spoofed source IP addresses are filtered by a SYN Proxy.

The operation of RGW requires configuration of different policies; host policies, circular pool allocation policies and firewall policies. RGW advocates embedding trust into the network by maintaining a reputation for all the network entities involved in communication. The reputation of a network entity can be decreased for example, if it tries to deplete the resources of RGW by sending multiple DNS queries without making any further connection attempt. The evidence is used in the future for not admitting flows from malicious users under heavy load

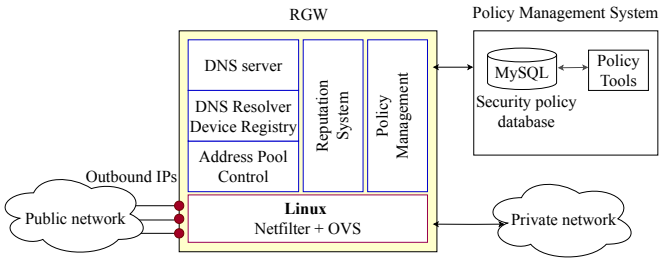

Fig. 1: System Architecture of Realm Gateway

conditions. The reputation system together with the policies allow fine grain control for the traffic flows.

IV. Overview of Sec-ALG

Sec-ALG is a proposal for dealing with web protocols particularly, HTTP and HTTPS for seamless connectivity without compromising on security or user privacy. In this section, we first discuss the design requirements followed by the proposed system architecture of Sec-ALG. We then describe our system implementation and the integration of Sec-ALG to RGW and SPM.

A. System Requirements

While designing Sec-ALG the underlying premise is that it must provide transparency to the users without requiring any modifications at the hosts. The design of the ALG adheres to the following goals:

- Sec-ALG must not interfere with the operation of existing network components. In this context, Sec-ALG must not modify the existing communication protocols.
- Sec-ALG must be able to interwork with NAT solutions particularly RGW and support their objectives.
- The proposed solution should not be complex and resource intensive; instead it should be easier to update without making exhaustive changes in the original solution.

B. System Architecture

At a high-level, Sec-ALG acts as an intermediary between two network realms; the clients in the public network requesting the services and the web servers situated in the private network behind the NAT/firewall.

1) System Operation: Our system relies on three main modules for its operation. Fig. 2 indicates the modules of Sec-ALG involved for transparent connection establishment between the public domain and private domain hosts. First, it has a module for storing information in the form of policies for the domains served by the web servers. The second module is a custom parser-lexer for identifying the hostname after the application layer protocol detection. The log management module is another component used by Sec-ALG for the purpose of auditing the connection information and detecting malicious behaviour by the entities involved.

The module with the domain information stores the internal IP address and port number corresponding to each domain name

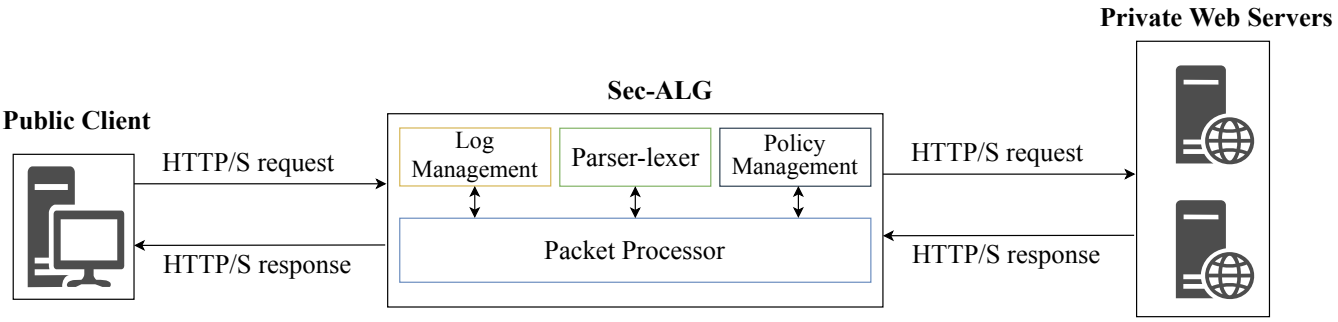

Fig. 2: Architecture of Sec-ALG

as Sec-ALG's policies, used for forwarding the connection. We use a novel parser-lexer generator called YaLe [9] for extracting the domain name from the first HTTP or HTTPS request sent by the client. Sec-ALG leverages the Linux kernel stack for the maintenance of the existing Transmission Control Protocol (TCP) connections. The connection state information is written to a file by our log management module in real time where setting different verbosity levels can log information with varied detail. The connection is forwarded to the destined web server after the protocol and hostname have been successfully detected.

Prior to any connection, Sec-ALG must have the correct domain name to internal IP address and port mapping for forwarding the connection correctly. An end-to-end connection between a client and a server is divided into two segments by Sec-ALG referred as connection halves. The communication in each direction, upstream or downstream, is handled in a separate Operating System (OS) process for simplicity.

First, the client sends the TCP SYN packet which is handled by the OS's kernel. On receiving the first data packet, the kernel forwards the packet to the user space Sec-ALG. The ALG detects the application layer protocol from the HTTP/S request and then validates the domain name using the custom parser-lexer. The connection is established with the upstream web server if a valid hostname is detected by Sec-ALG.

2) Custom Parser-lexer: Analyzing the application data is common for network monitoring tools and they employ the technique of lexical analysis and syntax analysis commonly known as parsing, in one way or another. A lexer converts the input characters into meaningful tokens while a parser is used for finding the relationship between the tokens based on the pre-defined grammar rules. Our proposal involves parsing the application payload only for detecting the protocol used for communication and the hostname of the web server and then terminating the parser.

When the HTTP/S request is received by Sec-ALG, it makes an API call to YaLe-generated parser-lexer. The generated parser-lexer uses an event-driven approach for parsing the network protocol payloads in a non-blocking manner based on a pre-defined set of grammar rules. It has a Deterministic Finite State Machine (DFSM) architecture where the operation of lexer is dependent on the parser state. YaLe uses the common lexical analysis approach called longest-match lexing which involves constructing the tokens from the maximum input characters matching against the pre-defined grammar rules. The longest match lexing is implemented using a bounded statically allocated backtrack buffer. The parser tool of YaLe uses a table-driven LL(1) grammar for matching the input into terminal/non-terminal symbols using callback functions and uses as statically allocated stack for the generation of the output.

The parser-lexer parses the HTTP request for finding the information on the destined web server contained in the Host header field. The HTTP request is broken down into lexical tokens and the HTTP grammar rule handlers consult the callback function to return the hostname. Similarly, the TLS message is parsed to find the hostname from the SNI extension of the TLS protocol [10] during the initial handshake phase. The detection is done using YaLe's TLS grammar. We use the parser-lexer module only for the detection of hostname

C. System Implementation

We implemented a prototype for our proposed Sec-ALG solution in Python that is available as an open-source software [20]. Sec-ALG exploits the multi-processing approach for maintaining a high number of concurrent connections. As discussed in Section IV-B1, each connection is broken into two connection halves. The master process is responsible for accepting new connections and spawns two child processes for dealing with one end-to-end connection. All the associated operations for web server's policies are handled by the master process. By using semaphores, it ensures that no conflict occurs when policies are updated in the master process.

Rate-limiting functionality is based on the token bucket algorithm often used in traffic engineering for accepting the incoming packets based on their arrival time [21]. The token bucket algorithm has an inherent disadvantage that on reaching the maximum connection count, defined before running Sec-ALG, all new connections are dropped until a new token becomes available. When a new TCP connection is initiated by the client, it remains open for a duration we refer to as connection timeout. If no application layer data is received by Sec-ALG within the pre-set connection timeout, the connection is closed.

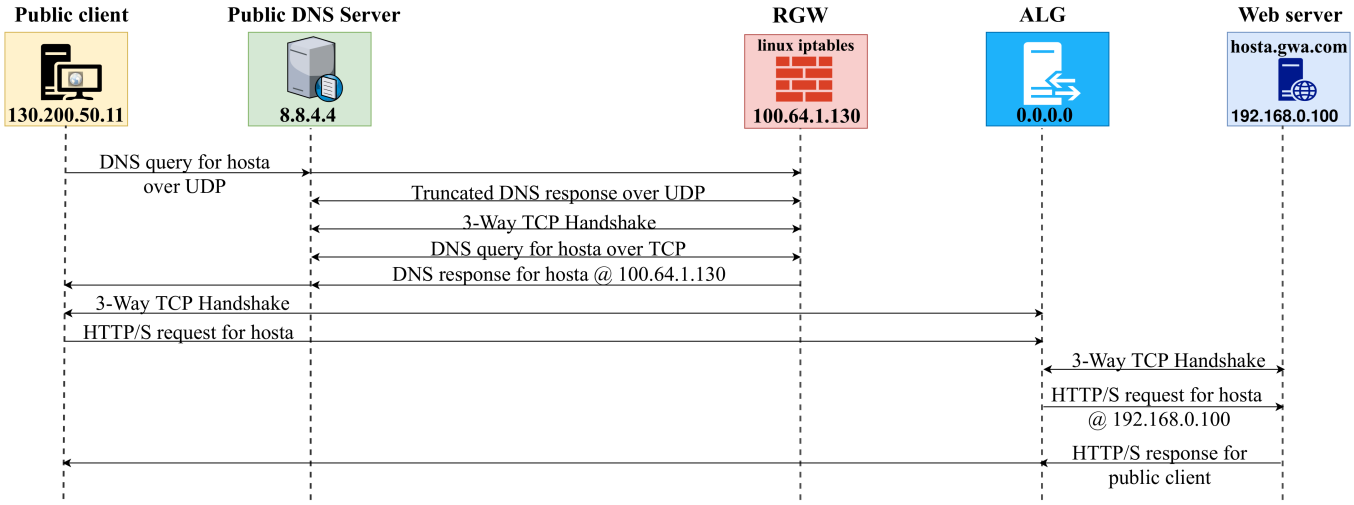

Fig. 3: Connection Establishment using Sec-ALG and RGW

The Python socket module is used for establishing a TCP connection. The socket timeout function is used to achieve the non-blocking socket behaviour by setting a minimum timeout value. The keep-alive mechanism of TCP sockets is used by Sec-ALG to send keep-alive messages to the connection in the waiting state. In the event of no response, Sec-ALG closes the connection.

When the first HTTP/S request is received from the client, ALG parses the request to determine the application layer protocol based on the stored pre-defined metadata. The protocol detection is used for sending the request to the correct YaLe-generated parser-lexer, imported as a shared loadable module in Sec-ALG. The network socket programming employed in Sec-ALG is OS-specific and it is based on Linux system. However, the logical implementation of our software can be used for other computing platforms as well.

1) Integration with RGW: We configured additional policies in RGW to forward the web traffic to Sec-ALG. The policies are enforced in the OS kernel's NAT table and filter table that are accessed using the iptables utility in the user space. After integration, Sec-ALG runs as a separate service on the RGW system. By default, Sec-ALG is listening on the wildcard IP address and thus binds to all the IP addresses used by RGW. Consequently, increasing the size of RGW's circular pool or changing the addresses in the pool would not affect Sec-ALG's operation.

As we discussed in Section III, the DNS query sent by the client can contain information about the requested service in which case the circular pool in RGW is not used for connection establishment. The flows exchanged for one HTTP/S session establishment are shown in Fig. 3. The DNS queries are handled by RGW acting as the authoritative name server for the served hosts. In the start of the communication, if the DNS query is sent by a public client over UDP, RGW responds with a truncated DNS message. The truncated DNS response is an indicator for the querier to send the DNS request using TCP. On receiving the DNS request over TCP, RGW informs the public client to access the web server using its public IP address. The HTTP/S connection is established by Sec-ALG using RGW's public IP address and bypasses the circular pool algorithm. However, if the web service is not specified in the DNS query, the connection is established using an available circular pool IP address. On receiving the HTTP/S request by the client, RGW updates the connection table with the connection state information and releases the circular pool IP address for the next connection allocation. Sec-ALG directly handles further traffic for the client session. Furthermore, to protect against TCP SYN flooding, the traffic can first pass through a SYN Proxy before it reaches the RGW and Sec-ALG setup further discussed in [22].

2) Integration with SPM: We developed SPM to allow the end users and administrators to manage security policies for improving their network communication [23]. SPM consists of a Policy-API, a security policy database and a REST based server. The security policies are stored and fetched from the policy database using the Policy-API. Other functions including modification, creation or removal of policies are also supported by SPM. The users can update the user policies remotely through a web interface. The user input generates an HTTP request for the REST server in SPM which sends it to the Policy-API after extracting the query parameters. The Policy-API handles GET, PUT, POST and DELETE policy requests. All the policies in SPM are stored in JSON format.

The policies involved in the operation of ALG and RGW, discussed in Section III and Section IV, are stored in the security policy database of SPM. The policies are retrieved using a separate HTTP REST client which polls the SPM after a pre-defined time and stores them in a local file accessed by Sec-ALG when it starts. The policies related to the web server's domain information can be updated and the changes are reflected in the local file by the HTTP client.

The updated policies become effective when new connections are established by Sec-ALG. It is also possible to register new web servers after Sec-ALG is operational. In a use case scenario, the administrator can modify the internal IP address, port number of the web server or add new web servers to modify the Sec-ALG behaviour in real time.

The architecture of RGW is based on Python's asynchronous framework, and it retrieves the policies from the SPM using an asynchronous HTTP client. A system call is generated to retrieve the policies at the start of RGW's operation. The policies concerning the network communication of the end hosts are updated after a specified time using asynchronous co-routines in case of modification in the policy database. Each host's FQDN acts as the key for managing their policies. The updated policies are reinitialized in RGW and are reflected in any further communication.

V. Application in Secure Banking

In this section, we present a use case of our proposed Sec-ALG. Banking industries often employ in-network components for making their servers resilient against attacks. They also like their customers to use the bank's own mobile applications. A detailed survey conducted in [24] indicate that most banks across the globe use SSL/TLS protocol for providing online banking services. Our Sec-ALG and RGW solution can be used by the bank's Internet Service Provider (ISP) for example, when a user connects to the bank's server for accessing internet banking using HTTPS. To improve resilience against Distributed Denial of Service (DDoS) attacks, each connection would be established using a unique FQDN by the user. The FQDN would be shared with the user by the bank after the user authentication has taken place. Different methods can be used by the banks for online user authentication [24]. To counter TCP SYN attacks, the RGW and Sec-ALG are placed behind a SYN Proxy [25]. As a result, a potential attack we consider in this paper will always use non-spoofed source IP addresses.

The Sec-ALG and RGW software running at the ISP, e.g. on a cloud platform, is responsible for routing the connection correctly to the bank's web server. RGW and Sec-ALG would admit the traffic flows based on policies defined and configured by the bank using our security policy management system [11]. To further improve connection security and resilience against DDoS, the bank can run the web service on multiple domains and the Sec-ALG will be responsible for forwarding the connection to the correct internal port and IP address of the requested web server. The use of unique FQDN for each user session provides an opportunity to collect evidence of suspicious behavior by potential attackers requesting the IP address for the domain names that are no longer available.

The bank and ISP can agree on the details of exchanging DNS information for establishing connectivity or the bank can run its own DNS server with TCP, or TLS over TCP rather than DNS over UDP for access by the ISP's DNS servers. If the RGW runs on a cloud platform, under a heavy DDoS attack, its globally unique outbound addresses can be re-orchestrated as long as the DNS zone for the RGW is also updated.

Separating the identification and location functionality of a network node can improve the security of the system in critical applications e.g., banking. Unique identifier can improve accountability in case of attack situations, and therefore enhance the trust in the network [26]. Sec-ALG does not decrypt the traffic, and thus the user confidentiality and privacy are preserved.

VI. Evaluation

A. Experimental Setup

We evaluated Sec-ALG in a virtualized environment using Linux Containers (LXC). Our testbed comprised of an orchestrated environment where different LXC containers represented the public clients, public DNS server, a host machine for the integrated setup of Sec-ALG and RGW and the private web servers. Sec-ALG was listening on standard HTTP/S port 80 and 443 and on non-standard ports 8080 and 8443.

The orchestration environment was tested on three different host machines with different hardware specifications. Host machine 1 was a workstation equipped with Linux kernel 4.15, Intel i5-6300U quad-core processor having 8GB of RAM and 2.4 GHz clock frequency. Host machine 2 and host machine 3 were running Intel Xeon processors with Linux kernel 4.13. Host machine 2 was Intel Xeon E5-2630 (2.3 GHz, 24 cores) with 32 GB RAM while the specifications of host machine 3 include an Intel Xeon E5-2630L v5 processor (2.4 GHz, 24 cores) with 64 GB RAM. All the host machines were running a 64-bit, Ubuntu 16.04 Long Term Support (LTS) OS. Most powerful web servers today run on specialized server-grade hardware and the optimum performance of Sec-ALG will be achieved on a host machine with server-grade hardware specifications.

For performance evaluation, we choose the metrics of latency, scalability and availability. First, we measure the latency of HTTPS connection to estimate the overhead due to encryption. We also evaluate the scalability of our system by subjecting it to increasing load conditions for both HTTP and HTTPS connections. To observe the availability, Sec-ALG is subjected to HTTP Denial of Service (DoS) attack and the performance is observed. In addition to our test scripts, we also used existing open-source benchmark tools for HTTP and HTTPS. We use NGINX, a high performance, well-known reverse proxy server as the baseline for evaluation as it was integrated with the original version of RGW [2]. The effect of integrating policy database on the performance of Sec-ALG and RGW is also observed.

B. Latency

To quantify the overhead added by encryption, we evaluate the time taken by a public client to retrieve a 626 bytes web page from the web server. The request is sent using the HTTP GET method and OpenSSL library is used for encrypting the request with SSL wrapper. The latency of

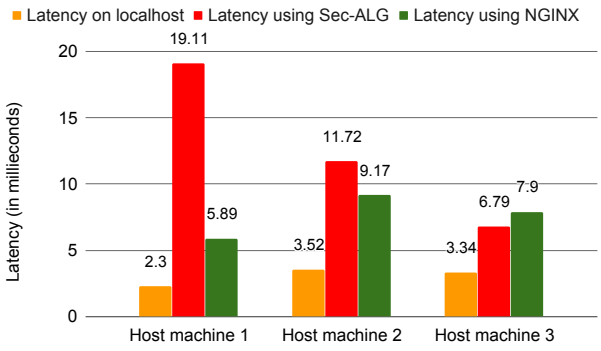

Fig. 4: Measured latency for HTTPS connection

the localhost running the back-end NGINX web server is measured as a baseline for different host machines. Each value shown in Fig. 4 is an average of 1000 measurements. The latency measurement includes the process time for maintaining consistency in different measurement setups.

The execution time of a test case is measured using Python's time module. We compare the results with those of NGINX reverse proxy. We observe that the latency of Sec-ALG was 3 times higher than NGINX on host machine 1. The reason for the higher latency in host machine 1 is attributed to the processor's limited hardware specification. However, using host machine 3, Sec-ALG outperforms NGINX by taking 1.13 milliseconds less in serving one HTTPS request. Sec-ALG utilizes the multi-core structure of the host machine whereas NGINX handles a connection using a single worker process.

Sec-ALG is written in python which introduces memory overhead in comparison to NGINX written in C programming language. Multiple processes consume memory but using server-grade hardware reduces the impact of the overhead added by Sec-ALG. We believe that Sec-ALG can perform even better when the processor of the host machine has a more advanced specification than host machine 3. The improved performance would be due to lower process creation (forking) time in different host machines. This is evident from the time taken for forking 1000 processes in host machine 1 being 7.4 seconds and a significantly lower value of 0.76 seconds in host machine 3.

C. Scalability

We next evaluate Sec-ALG's scalability by increasing the number of concurrent clients initiating multiple connections. We use weighttp [27], a multi-threaded, open-source benchmark HTTP tool for observing the number of requests handled by Sec-ALG and NGINX reverse proxy. For the HTTPS stress testing, Siege [28] benchmark tool is used as weighttp does not support HTTPS. The scalability tests are performed on host machine 3 with the hardware specification explained in Section VI-A.

1) HTTP Stress testing: The number of requests is increased gradually from 1000 to 100,000 using different concurrency levels. Each client retrieves a static web page of 323 bytes. The tests help in evaluating the time it takes

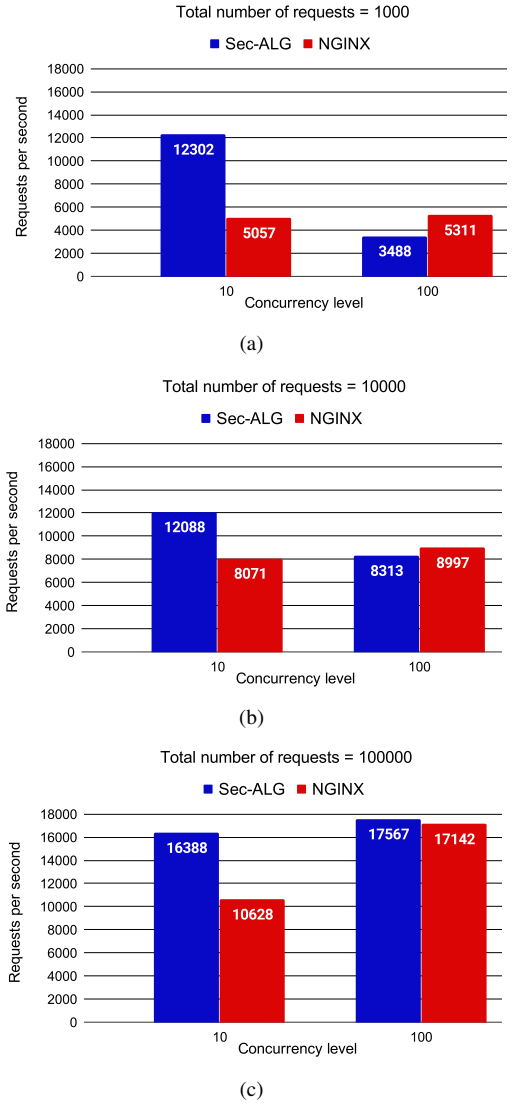

Fig. 5: Scalability testing for HTTP connections on host machine 3

to serve multiple clients. To get an accurate comparison, the configuration parameters of the NGINX reverse proxy are tuned so that each worker process in NGINX can handle up to 10000 simultaneous connections. The scalability tests are performed only on host machine 3.

Fig. 5 indicates the benchmark testing results obtained using weighttp. We can see that the performance of Sec-ALG is significantly better than the performance of NGINX reverse proxy for smaller concurrency level as shown in Fig. 5. For higher number of HTTP requests sent, the performance of NGINX reverse proxy is comparable to Sec-ALG indicated by Fig. 5b and Fig. 5c, with the ALG handling requests at a slightly increased rate as indicated by Fig. 5c.

It was observed that increasing the number of total requests sent by the clients resulted in an increased rate of served requests in Sec-ALG or NGINX reverse proxy. However, this is valid only until the maximum concurrency level is attained

by the web server. Increasing the number of requests with 1000 parallel clients resulted in huge processing delays with failed requests, discussed in detail in ([22], Table. 9).

Further investigation revealed that the cause for failed requests was due to the PHP-FastCGI module of the backend NGINX web server. Sec-ALG uses multi-processing ideal for handling small concurrency levels while NGINX is based on a non-blocking I/O multiplexing hardware better suited for higher concurrency levels. However, the results obtained with a concurrency of 100 for varied number of total HTTP requests sent indicate that performance of Sec-ALG is comparable to NGINX proxy server for HTTP connections.

2) HTTPS Stress testing: We investigate the performance of Sec-ALG when subjected to multiple parallel HTTPS requests using Siege [28] on host machine 3. Siege is our chosen benchmark tool as it supports sending SNI extension during the initial TLS handshake. The tool has an inherent limitation which resulted in a few failed requests when a large number of requests are sent in parallel. Similar to Section VI-C1, the configuration parameters of the NGINX reverse proxy were tuned so that each worker process can handle up to 10000 connections. We observed the impact of concurrent clients by increasing the total number of requests. The backend web server was unable to handle requests from 1000 simultaneous clients and the requests started failing, further discussed in ([22], Table. 14).

We illustrate the results of our scalability testing for HTTPS in Fig. 6, and it can be seen that the performance of Sec-ALG is notably better than NGINX reverse proxy. The reason behind the slow served request rate of NGINX reverse proxy is the computational overhead due to decryption and then re-encryption of traffic for forwarding. In contrast, Sec-ALG only performs light DPI to detect the hostname without decrypting the traffic, and stops the DPI after hostname has been successfully extracted.

By comparing Fig. 5 and Fig. 6, we observed that a higher request rate is attained for HTTP requests by Sec-ALG and NGINX reverse proxy than HTTPS requests because of the additional overhead associated with an HTTPS connection. Since Sec-ALG does not decrypt the traffic, the difference between HTTP and HTTPS connection request rate is lower in Sec-ALG than NGINX reverse proxy. With the increase in encryption of network traffic, the performance improvement offered by Sec-ALG becomes more significant.

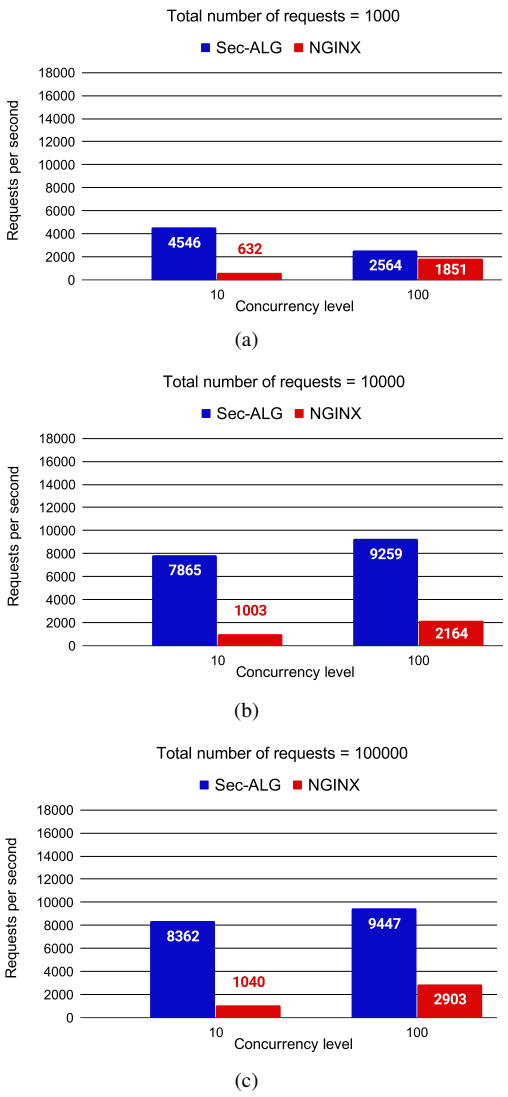

Fig. 6: Scalability testing for HTTPS connections on host machine 3

D. Availability

We evaluate the availability of the back-end web server under attack conditions. For this purpose we conduct a low-bandwidth HTTP DoS attack against the protected web server, also known as slowloris attack. The design of HTTP requires the web server to wait for the complete request before it can be processed. A malicious user exploits this design criteria of HTTP by initiating multiple HTTP connections towards the backend web server. The user sends HTTP headers periodically but never fully completes the request. The purpose is to exhaust the resources of the backend web server.

We use an application layer DoS simulator SlowHTTPTest [29] for evaluation. The connection count is gradually increased to 20000 at a a rate of 1000 connections per second. The maximum window size advertised by the client is 24 bytes and a new HTTP header is sent after every 10 seconds. The request is never fully completed by the client. We also send traffic from 100 concurrent legitimate users towards the web server. The legitimate users request a 1 MB file from the web server. Fig. 7 indicates that Sec-ALG's rate limiting functionality did not allow opening of 20000 connections when the maximum allowed connections are 10000. Moreover, the pending connection count is never more than one illustrating that Sec-ALG handles all the incoming connections. In the test presented in Fig. 7, the web server closed the connection half after 60 seconds when the complete HTTP request was not received and resultingly, Sec-ALG closed the other connection half towards the client end.

Test parameters	
Test type	SLOW HEADERS
Number of connections	20000
Verb	GET
Content-Length header value	4096
Extra data max length	52
Interval between follow up data	10 seconds
Connections per seconds	1000
Timeout for probe connection	2
Target test duration	240 seconds
Using proxy	no proxy

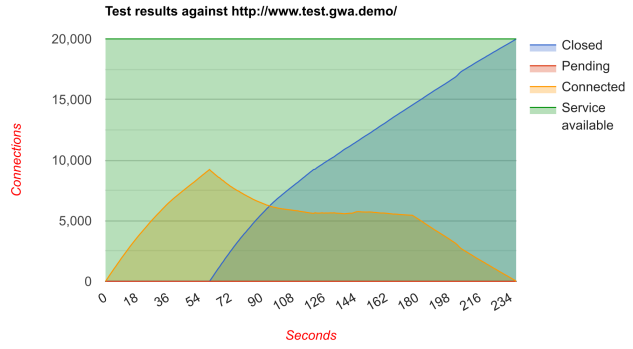

Fig. 7: Results of the low-bandwidth HTTP DoS attack

The results indicate that the legitimate users were able to retrieve the requested file without any disruption. However, without the backend web server's protection against slowloris attack, Sec-ALG alone does not have functionality for mitigating low-bandwidth HTTP DoS attack. The scalability testing in Section VI-C1 indicates that HTTP flooding using GET requests does not exhaust Sec-ALG's system resources. Moreover, the clients were able to access the web page requested within a reasonable time.

Next, we evaluate how Sec-ALG responds to HTTP requests that do not include the hostname. Sec-ALG uses a multi-process architecture and for N TCP connections initiated by the client, $2N$ processes are created by Sec-ALG. A user might exhaust Sec-ALG's system resources by initiating multiple HTTP connections and depleting the available resources of the host machine. To protect Sec-ALG against a resource depletion attack, it has a connection timeout. When Sec-ALG cannot detect the hostname or no application layer data is sent within the connection timeout, Sec-ALG closes the associated process to prevent exhaustion of system resources. In this case, the connection half towards the backend server is never initiated by the Sec-ALG. The attack scenario is simulated by initiating 40000 connections using 100 clients. The connections are closed by Sec-ALG within the specified timeout of 2 seconds and the process count never exceeded 2500. The test validates that Sec-ALG closes the malicious connections quickly and it is also verified from the log records.

E. Policy database

The performance of SPM has been analyzed in [30] and [23]. Our aim is to observe the impact of the integration of SPM on Sec-ALG and RGW's performance. The policies of Sec-ALG are grouped into one policy database entry. The policies for the web servers are updated every 10 seconds if the local file containing the policies is modified since it was last accessed. RGW has an extensive set of policies as explained in Section III. We observe the time for retrieving policies for 16 different hosts (host policies) and 463 policies for the firewall functionality. Additionally, 81 policies (CP policies) are configured for the circular pool operation of RGW. The policies are added in the MySQL database of SPM and the time taken for policy retrieval is measured.

TABLE I: Time taken to fetch policies from SPM

Setup	All policies ALG [ms]	Host policies RGW [ms]	CP policies RGW [ms]	All policies RGW [ms]
Stand-alone	9.85	102.08	23.47	126.44
RGW+ALG	10.63	146.09	25.94	173.55

RGW has an asynchronous system architecture and an asynchronous system call is used to fetch the policies. Table I shows the time taken to fetch different policies of RGW and Sec-ALG. The time is first measured when Sec-ALG and RGW act as stand-alone components. To see if the integration of RGW with Sec-ALG and SPM affects its performance, the policy retrieval time is measured again after integration. We measure the connection establishment time for one client when SPM is integrated with Sec-ALG and RGW. The time to serve the HTTP request when the policies are retrieved locally is 5.13 ms, whereas almost same response time of 5.18 ms is observed after integration with SPM. These results demonstrate that the integration of SPM with Sec-ALG, or RGW does not degrade their performance.

VII. Discussion

Different in-network components facilitate the access to end hosts in private network space. Our Sec-ALG is compatible with HTTP/S used by web servers to provide various services. It can also work with any application protocol that has TLS with SNI support as an underlying protocol.

We argue that maintaining the data privacy in an encrypted end-to-end connection is crucial. The security community acknowledges the occurrence of TLS interception and the vulnerabilities it poses [31] but no consensus has been achieved on a standard solution to deal with encrypted traffic at the middlebox. We advocate that only the end devices should have access to unencrypted traffic and the middlebox should not perform the operation of decryption. The recent shift towards cloud-based technology and its data privacy issues further validate the argument.

We believe that disruption in critical web services such as e-governance, e-health or banking can lead to social unrest. The services offered by the private network hosts should be available for the public users at all times. In Section V, we presented a use case for our system to enhance the security in internet banking. Taking a bank in United States as an example, which has 35 logins to their banking application per second and that go up to 100 logins per second during peak hours [32], we subjected Sec-ALG to legitimate and malicious traffic in the test conducted in Section VI-D. We observed

that the services for the legitimate users were not disrupted even under a low bandwidth HTTP DoS attack. Our Sec-ALG and RGW solution can be used in addition to the security mechanisms employed by banks for client authentication [24] to provide more fine-grain control for example, for admitting maximum connections based on load conditions, by allowing configuration of security policies remotely by the banks.

Our work focuses on promoting trust based networking [26] where in the event of an attack, an entity can be held responsible. RGW currently has a reputation system that is used for collecting evidence of malicious behaviour and allocating resources under varied load conditions. Future work would involve integrating the reputation system of RGW with Sec-ALG.

Although the idea of TLS SNI extraction for connection establishment has been deployed in commercial firewalls, it is still not widely available in open-source software. Our Sec-ALG relies on TLS grammar for detecting the plaintext hostname from the TLS handshake. However, there is active discussion on replacing TCP with a new encrypted transport layer protocol called QUIC. The adoption of QUIC, if followed by encrypting the SNI extension in TLS version 1.3 would require some mechanism where Sec-ALG can access the plaintext hostname in the beginning of the session. However, the replacement of TCP with QUIC as the default transport protocol in the protocol stack first requires consensus and standardization. A possible approach to handle fully encrypted inbound packet transport is to offer the RGW with Sec-ALG and SPM as Software as a Service (SaaS) to the bank on the telco cloud. Under this model, the bank would maintain the end-to-end encryption and prevent the telcos and any 3^{rd} party from intercepting user traffic by using a highly scalable and resilient network (cloud) based service.

VIII. Conclusion

In this paper, we presented the design, implementation and evaluation of a novel application layer gateway called Sec-ALG, which aims at providing accessibility to private network hosts while improving end-to-end security. Sec-ALG uses a multi-process approach and it relies on the technique of light DPI for establishing HTTP and HTTPS connections without requiring the private keys of the served hosts. We proposed an open-source solution that uses a novel parser-lexer for detection of hostname and protocol from the packet payload without decrypting the traffic. By using domain names to establish connections, Sec-ALG also alleviates the issue of IPv4 address scarcity and improves the HTTPS connection latency in comparison to other middlebox solutions. Sec-ALG is inter-operable with IPv6 networks as well.

We proposed an integrated system which comprised of Sec-ALG, NAT traversal solution called RGW and policy management system, SPM, to improve the overall security of the private hosts. Our solution has the benefit of supporting incremental deployment and it does not require any changes on the end hosts. To improve security further, we used optional changes on host applications in the banking use case discussed in this paper. The multi-process architecture of Sec-ALG utilizes the resources of the system for increasing scalability. Our solution does not introduce any overhead due to decryption and then re-encryption of HTTPS traffic, and thus the request handling capabilities are improved significantly compared to the state-of-the-art solutions.

Acknowledgment

This work was supported by the Finnish public funding agency for research, Business Finland under the project "5G Finnish Open Research Collaboration Ecosystem (5G-FORCE)" which is part of 5G Test Network Finland (5GTNF).

References

[1] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional (NAT)," RFC 3022, IETF, Jan. 2001. [Online]. Available: https://tools.ietf.org/html/rfc3022.

[2] J. L. Santos, R. Kantola, N. Beijar, and P. Leppäaho, "Implementing NAT traversal with Private Realm Gateway," in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 3581–3586.

[3] H. Kabir, J. L. Santos, and R. Kantola, "Securing the Private Realm Gateway," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, May 2016, pp. 243–251.

[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[5] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 255–273.

[6] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep Packet Inspection over Encrypted traffic," *ACM SIGCOMM Computer communication review*, vol. 45, no. 4, pp. 213–226, 2015.

[7] H. Lee, Z. Smith, J. Lim, G. Choi, S. Chun, T. Chung, and T. T. Kwon, "matls: How to Make TLS middlebox-aware?" in *NDSS*, 2019.

[8] C. Liu, Y. Cui, K. Tan, Q. Fan, K. Ren, and J. Wu, "Building generic scalable middlebox services over encrypted protocols," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2195–2203.

[9] "Yale," Software, 2019. [Online]. Available: https://github.com/Aalto5G/yale.

[10] D. Eastlake, "Transport Layer Security (TLS) Extensions: Extension Definitions," RFC 6066, IETF, Jan. 2011. [Online]. Available: https://tools.ietf.org/html/rfc6066.

[11] "Security policy management," Software, 2019. [Online]. Available: https://github.com/Aalto5G/SecurityPolicyManagement.

[12] O. Novo, "Making constrained things reachable: A secure ip-agnostic nat traversal approach for iot," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, pp. 1–21, 2018.

[13] A. Rao, J. Sherry, A. Legout, A. Krishnamurthy, W. Dabbous, and D. Choffnes, "Meddle: middleboxes for increased transparency and control of mobile traffic," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, 2012, pp. 65–66.

[14] J. Jarmoc and D. Unit, "Ssl/tls interception proxies and transitive trust," *Black Hat Europe*, 2012.

[15] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged ssl certificates in the wild," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 83–97.

[16] G. Tsirantonakis, P. Ilia, S. Ioannidis, E. Athanasopoulos, and M. Polychronakis, "A large-scale analysis of content modification by open http proxies." in *NDSS*, 2018.

[17] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala, "Tls proxies: Friend or foe?" in *Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 551–557.

[18] S. Ruoti, M. O'Neill, D. Zappala, and K. Seamons, "User attitudes toward the inspection of encrypted traffic," in *Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016)*, 2016, pp. 131–146.

[19] J. L. Santos and R. Kantola, "Transition to ipv6 with realm gateway 64," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 5614–5620.
[20] "Applicationlayergateway," Software, 2019. [Online]. Available: https://github.com/Aalto5G/ApplicationLayerGateway.
[21] D. Medhi and K. Ramasamy, "Chapter 18 - traffic conditioning," in *Network Routing (Second Edition)*, second edition ed., ser. The Morgan Kaufmann Series in Networking, D. Medhi and K. Ramasamy, Eds. Boston: Morgan Kaufmann, 2018, pp. 626 – 644.
[22] M. Riaz, "Extending the functionality of the realm gateway," G2 Pro gradu, diplomityö, 2019-10-21. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-201910275869
[23] H. Kabir, M. H. B. Mohsin, and R. Kantola, "Implementing a security policy management for 5g customer edge nodes," in *Accepted in 2020 IEEE/IFIP Network Operations and Management Symposium'*. IEEE, 2020.
[24] S. Kiljan, K. Simoens, D. D. Cock, M. V. Eekelen, and H. Vranken, "A survey of authentication and communications security in online banking," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–35, 2016.
[25] "nmsynproxy." [Online]. Available: https://github.com/Aalto5G/nmsynproxy.
[26] R. Kantola, "6g network needs to support embedded trust," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1–5.
[27] "weighttp," Software. [Online]. Available: https://github.com/lighttpd/weighttp.
[28] "Siege," Software, 2019. [Online]. Available: https://github.com/JoeDog/Siege.
[29] "Slowhttptest," Software. [Online]. Available: https://github.com/shekyan/slowhttptest.
[30] M. Mohsin, "Security policy management for a cooperative firewall," G2 Pro gradu, diplomityö, 2018-10-08. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-201810175456
[31] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of https interception." in *NDSS*, 2017.
[32] K. Silvestri and C. McFee, "Keybank chooses anthos to develop personalized banking solutions for its customers," Google Cloud Blog, Aug. 2019. [Online]. Available: https://cloud.google.com/blog/topics/hybrid-cloud/keybank-chooses-anthos-to-develop-personalized-banking-solutions-for-its-customers.