University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

PHD THESIS

SUMMARY

Enhancing application awareness through distributed firewalling

Scientific Adviser:
Prof. Dr. Ing. Nicolae Țăpuș

Author:
Ing. Radu-Alexandru Mantu

Bucharest, 2025

Chapter 1

Introduction

During the early days of the Internet, external network threats were easy to counteract due to the unambiguous nature of clients, as well as the limited availability of web applications. The association between server port numbers and the listening applications could be drawn more reliably. In turn, this simplified assessing the purpose behind each new connection. In 2003 however, Gartner Research defined what they considered to be four sound approaches to gateway security, thus coining the term ”Next Generation Firewall” (NGFW). With the purpose of opposing emerging network security threats, NGFWs integrated Intrusion Detection and Prevention Systems (IDS/IPS) and application awareness into conventional firewalls. Even at that time, it could be stated that endpoint applications were by and large detached from the meaning that the port they were bound to conferred in years prior. Thus, techniques such as JA3 or Markov Chain Fingerprinting have since been developed for the purpose of active threat mitigation. However, their efficacy (as well as that of IDS/IPS) is significantly diminished when applied to encrypted traffic. Despite ongoing efforts towards encrypted traffic classification, TLS traffic decryption unequivocally remains the most preponderant solution to this problem, levying a compromise between user privacy and general network security.

In spite of this shortcoming, NGFWs are more relevant than ever due to the ongoing adoption of Zero Trust network architectures. As enterprises seek to implement fine-grained, least-privilege resource access policies, the utilization of micro-segmentation has become increasingly common, as shown in a recent report on Alphabet’s transition to the Zero Trust paradigm, i.e. Google BeyondCorp. We argue that current access decision systems which are commonly based on user authentication are insufficient for preventing attacks that are facilitated by vulnerable versions of common tools or shared libraries, e.g. the recent `liblzma` backdoor or software supply chain attacks. This problem is further exacerbated by the increasing fluidization of network demarcations, a phenomenon determined by the necessity of free movement between enterprise-owned and public networks. We believe that meeting these challenges should be aided by end-user devices.

1.1 Thesis Contributions

In this thesis, we provide a comprehensive solution to application awareness in the form of a distributed firewall. This firewall has been implemented from scratch and has two additional variants that are more limited in scope but serve as proof of concepts for future efforts. Additionally, we propose a packet annotation scheme that allows our firewall to

convey proof of compliance for emitted traffic to other instances of the same firewall or Intrusion Detection and Prevention Systems or existing software firewalls.

We claim the following contributions:

1. **Evaluation of protocol option acceptance in the Internet:** By implementing a traffic annotation tool with IP, TCP and UDP options, we perform reachability tests between 21 sites across the globe, and four cloud providers as well as our university.
2. **Implementation of an application-aware distributed firewall:** We offer multiple variants of this firewall. The primary version is developed for Linux and runs strictly in userspace. A kernel-based alternative offers fewer features but demonstrates singificant performance increase. The third variant represents a proof of concept implementation for Windows systems.
3. **Evaluation in realistic environments:** We test our Linux-based firewall implementations in both datacenter environments, on servers with 10Gbps NICs, as well as desktop environments, on Intel NUC systems with 1Gbps NICs. We offer the user the option of tuning the firewall and select what security compromises are tolerable for the purpose of increasing performance.
4. **Packet annotation scheme and compliance verification:** We propose a packet annotation scheme based on protocol options. Although we evenutally decided on IP options as a conduit for the proof of compliance attached to each packet, we discuss the current limitations of TCP and UDP options that prevented us from implementing a Transport Layer alternative, as well as the current efforts of IETF working groups to remedy them.
5. **Network integration solutions:** In order to integrate our packet annotation scheme in existing networks without the need to deploy our firewall on more hosts than necessary, we have implemented `iptables` and `snort3` modules that are able to verify the proof of compliance attached to each packet.
6. **Overview of the state of the art in Network Fuzzing:** We explore the possibility of employing fuzzing as an additional measure for ensuring the security of the applications that we whitelist in our firewall policies.

Chapter 2

Evaluation of IP/TCP/UDP options over public networks

As the Internet evolves, so do the expectations for new network protocols. Whether said protocols pertain to traffic encryption, packet source authentication or routing optimization, all are faced with the same initial hurdle: the uncertainty of their compliance with the myriad filtering policies employed by virtually just as many middleboxes. This incertitude has only been exacerbated by the continual ossification of the foundational internet protocols. An immediately apparent indicator of this suggested entrenchment is the maladroit adoption of new standards and acceptance of implementations conforming to well-established specifications. We draw attention to RFC 7126, offering recommendations on filtering packets with IPv4 options due to widespread misuse. Its necessity after upwards of thirty years since this IP extension mechanism was provisioned in RFC 791 is epitomic of the issue at hand. Regardless of the underlying causes that lead to this predicament, the objective is not necessarily to remedy but to circumvent the obstacle that it poses.

In this chapter we try to address both of the aforementioned problems. In reponse to the former proposition, we perform an investigation of protocol options acceptance in the Internet and establish their viability as conduits for the purpose of packet tagging. The latter of the two problems we address by means of a state of the art survey. This survey is meant to inform the reader on the challenges of emplying automated testing for establishing trust in non-deterministic builds of network applications.

2.1 Problem statement

In this chapter, we try to address the issue of protocol ossification. As such, we perform an investigation of protocol options acceptance in the Internet and establish their viability as conduits for the purpose of packet tagging. To this end, we propose the following research questions:

RQ1: Is the current state of affairs in the wider Internet adequate for already standardized protocol options? Extant protocols had been designed under the assumption that the existing fixed-length headers would at some point become insufficient for encapsulating the information needed by supplementary mechanisms. Consequently, most of them included a variable-length options feature allowing their incorporation as the need arises. The benefits are most evident when observing how TCP is reinforced by the Window

Scaling option (increases the maximum window size from 64Kb to 1Gb) or the Selective Acknowledgement option (allows specifying a finite range of lost packets). In their absence, TCP would cease to be viable in high-latency, high-bandwidth environments such as WANs. While options were initially overlooked in lieu of redefining existing fields to better fit specific purposes of certain users, the approach has been thoroughly discouraged by the open standards organizations and the community at large. Despite the growing demand for extending current protocols, Autonomous Systems (AS) proved resistant to this change. Although some options have been standardized, their acceptance remains highly dependant on middlebox manufacturers.

RQ2: Can new protocol options be attached to a packet without compromising its conformity to the filtering schemes? When the aforementioned protocol extension systems were first defined, a number of option identifiers (i.e.: codepoints, kinds) were designated to features immediately necessary at the time (e.g.: the IP Security option or the UDP Authentication and Encryption option). In order to properly administer the growing number of independent projects and prevent the unauthorized usage of unassigned codepoints from becoming praxis, IANA had reserved certain ranges for this purpose. This, in turn, lead to the introduction of experimental IDs as a method of sharing the yet limited assigned ranges, when testing in common environments. Nonetheless, there have been known cases of unpermitted use of certain codepoints, some overlapping with established options (e.g.: TCP User Timeout). Seeing how these unregistered and experimental options are practically unknown to most middleboxes, an understandable concern is whether they will be accepted without any further verification or simply dropped.

RQ3: Does the annotation scheme under consideration interfere with the base functionality of any protocol or application? Assuming that a modified packet is able to traverse the network unthwarted, the need to ensure normal operation of applications whose traffic falls within the purview of the scheme under test is paramount. Ideally, the packet alteration is handled outside the scope of the application itself, either in kernel (e.g.: DCCP's Explicit Congestion Notification) or by deference to another userspace process (e.g.: tcpcrypt's packet manipulation using NetfilterQueue). However, this is not always the case. Employing kernel bypasses can be rightfully motivated by practical limitations, such as IRQ storms during DDoS attacks (most CPU time is used to receive packets, not process them). During such attacks, `iptables` reaches a state of saturation at approximately 1M packets per second (pps). While solutions based on eBPF are known to be able to drop up to 15Mpps and quickly recover from a state of FIFO tail dropping, the technology is still new and imposes stringent constraints on the developer. Meanwhile, solutions based on frameworks such as Intel DPDK or PF_RING are more likely to be incompatible with any new annotation scheme. Consequently, a large-scale testing environment is required to validate any new addition to well-established protocols.

We claim the following contributions:

- We offer a tool[1] capable of intercepting specific packets and annotating them with

[1] Code available at https://github.com/RaduMantu/ops-inject

user specified, per-protocol option types. The options are generated in their entirety by a decoder that allows for easy integration of new option types, as well as protocols.

- We propose a framework for assessing an annotation system's compliance with firewall policies and provide configuration scripts and templates for cloud experimentation under the primary available providers (i.e.: Google Cloud, AWS, Microsoft Azure, Digital Ocean).
- We evaluate current Layer 3 and Layer 4 protocol extension mechanisms and deliberate their suitability as a basis for developing new extensions.

2.2 Architecture

2.2.1 Annotation tool

In testing different features of Layer 3 and Layer 4 protocols, a usual approach consists of generating synthetic traffic and bypassing the network stack of the originating host. We decided to forgo this method in favor of a more practical alternative: modifying real traffic. The primary advantage of the latter scheme over the former is the ability to verify not only that the annotated packets can be successfully transmitted over the network, but also that their alteration does not impact higher level protocols in the OSI stack. Additionally, the flexibility in assessing compliance with stateful firewalls alleviates the effort that would have otherwise been required to simulate sufficiently credible sessions.

These aspects gave impetus to the development of a method to insert multiple protocol specific options in outgoing traffic. One of the essential characteristics of the resulting tool is the capacity to easily implement new options for existing protocols or to register new protocols altogether. Limited by the extent of our needs for this experiment, session tracking falls outside its purview. As a result, options such as MultiPath TCP are not available for testing. Instead, all options should either be static in nature (e.g., NOP) or depend entirely on the information available in a single packet's header and payload (e.g., alternative checksums). Otherwise, its use may be limited to determining only whether the initial packet was able to traverse the network (e.g., TCP timestamps).

Next, we present a summary overview of the tool's operation. Initially, the user adds one or more `iptables` rules with an `NFQUEUE` target. All packets that match one of these rules are redirected into userspace for our tool to evaluate and modify as it sees fit. During the initial invocation of the tool, the user may specify a sequence of bytes that represent option codepoints. For each packet, the codepoints will be expanded to actual TLV entries and inserted into a (potentially new) options section. The NetfilterQueue callback that fulfils this purpose takes three steps towards successful annotation. Next, we describe these steps in greater depth:

Options Decoding: Having access to the packet, as provided by the Netfilter kernel hook, the first step is generating the contents of the options section for a pre-established protocol (i.e., IP, TCP, UDP). Using the user-specified sequence of `option-type` octets, specific

decoder functions are called upon to expand them, thus populating the options buffer. This expansion phase is by and large an arbitrary process. For example, an IP Timestamp Option (`0x44`) will be translated to a 12 octet buffer containing a single timestamp associated to the source IP address. The generated flags will denote pre-specified address fields as to discourage addition of new timestamps from compliant middleboxes. This inhibition of user agency is a conscious decision intended to simplify the tool's usage. Nonetheless, the decoder functions can be easily modified on a case-by-case basis to accept more information and conform to finer-grained requirements. Note that the options will appear in the order in which they were initially specified by the user. However, the order in which they are generated may vary. For example, a UDP Options Checksum (`0x02`) is supposed to be placed as close to the beginning of the section as possible, while at the same time requiring all other options to have already been calculated. Consequently, it's decoding is delayed based on a discretionary option priority assignment.

Packet Reassembly: This step modifies the initial packet in order to include the newly generated options. Any related fields (e.g., `IP.total_length`) with the exception of the checksum are adjusted to account for the payload offset. Based on user preference, existing options can either be completely replaced, or preserved. In our tests, we employed the former approach in order to exert full control over the content of the options section. Alternatively, existing options would take precedence. In turn, this could lead to overflowing user-specified options to be discarded due to space constraints.

Checksum Recalculation: The reason why this step is separate from the previous is that changes made to a header corresponding to a lower layer can impact the checksum of a higher layer protocol. For example, while IP options are intended to extend only the IP header, the total packet length increases nonetheless. As a result, TCP's or UDP's pseudo-header changes and the Layer 4 checksum does so along with it.

An advantage of relying on the Netfilter framework is that it integrates well with `ip-xfrm` (kernel-side module that implements the IPsec protocol suite). Usually, VPN implementations are split across both userspace and kernelspace. The former carries out the negotiation required as per the IKE protocol, which establishes up to four Security Associations (SA) and creates a Security Policy (SP). The latter utilizes the SAs for packet encapsulation and encryption but more importantly, it uses the SP to determine which packets are to be passed to the IPsec stack. This decision is taken by consulting an SP database after the packet has traversed the `OUTPUT` and `POSTROUTING` chains. If appropriate, the packet is modified and reinserted prior to entering the `OUTPUT` chain. The `iptables` rule that identifies the packets which require annotations must be well-defined. Otherwise, the user risks modifying a packet that is protected by the ESP protocol. We consider this approach preferable to excluding packets containing ESP headers. Due to the potential usage of NAT-traversal encapsulation, it would be inefficient to identify the presence of the ESP header. Moreover, allowing UDP packets with destination port 4500 to pass may exempt IKE configuration packets from annotation, contrary to the user's intention.

2.2.2 Testing framework

Irrespective of the type of protocol that may benefit from these extensions, they will in all likelihood be required to interact with cloud-hosted solutions. As a result, we focus our efforts on verifying if and how cloud providers allow the passage of annotated packets through their networks. To this end, we perform a series of inter-regional tests. The structure of the testing environment can be seen in Fig. 2.1. Initially, the management instance (i.e., localhost) will install the annotation tool, as well as any traffic generation tools under test on each virtual machine. Additionally it will also start `tcpdump` processes and insert specific `iptables` rules with the purpose capturing relevant traffic for annotation and later analysis. When testing a certain instance (e.g., the one in me-south-1 region in AWS), a server will be started (1) by issuing a command over a ssh channel. Immediately after, all other instances will be issued similar commands, to start a client (2) and send requests to the server. The resulting traffic will be annotated in both directions. After all communication (3) will have ceased, another instance will be selected as the server. Finally, all `tcpdump` processes will be killed and the generated packet captures as well as any logs will be downloaded for analysis.

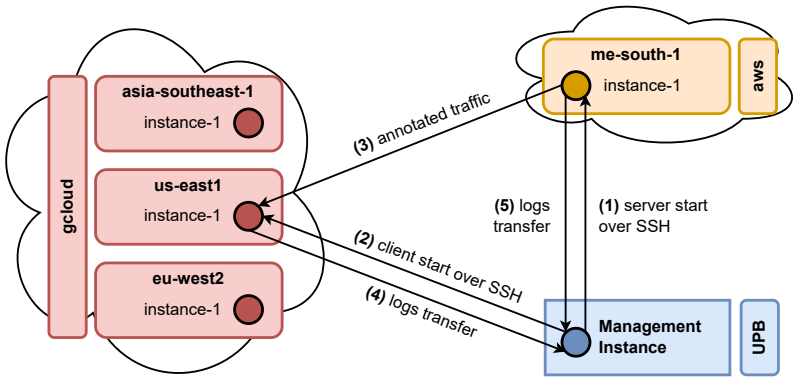

Figure 2.1: Architecture of the cloud testing framework.

The framework is comprised mostly of bash scripts using each provider's CLI management tool. A minimal setup is required: configuring default projects and public keys, enabling billing, and relaxing the firewall rules of their private networks to the extent that they are permitted. The scripts can be classified as management scripts, experiment scripts or analysis scripts, depending on their purpose. Management scripts allow the automatic creation or deletion of VM instances in accordance with a pre-specified list of regions. This list can be freely modified in accordance with the available selection of each cloud provider. Additionally, they administer the configuration of VPNs or that of the annotation tool. Experiment scripts install missing dependencies and generate traffic between any two instances. The commands executed on remote machines are issued over a secure SSH channel. All relevant inbound and outbound traffic is recorded on each instance and eventually imported on the managing host. Any analysis of packet reception and option removal or alteration can be done offline. To this end, we offer a subset of the analysis

scripts that we employed in interpreting the network captures, along with samples of said captures. We include only general-purpose scripts that, for example, report overall instance reachability or retrieve AS numbers and names of recorded IP addresses.

Note that this framework is not specifically aimed at evaluating cloud providers. Their preponderance in our tests was meant to verify the compliance to the protocol extension standards in the wider Internet. Given that the credentials are correctly configured on all machines, the experiments can be executed with either a statically defined set of public and private IPs, or via a user-provided means of dynamically generating said set.

Chapter 3

Application identity based firewalling

In 2003, Gartner Research marked the transition from classic firewalls to what they coined as Next Generation Firewalls (NGFW). Prior to this, firewalls used to analyze just the network and transport layer headers. This was sufficient due to a strong correlation between port numbers and services. However, the rapid diversification of said services and web applications gave rise to a new class of more potent adversaries that the previous generation of stateful firewalls could not contend with. As a result, NGFWs were developed for the purpose of achieving application awareness through deep packet inspection.

Although current firewalling technologies need to contend with new challenges such as micro-segmentation or the advancing dissolution of network demarcations, the core tenets established by Gartner back in 2003 still hold true today. However, a concerted effort dating back over ten years (e.g., Google site ranking, Let's Encrypt, etc.) rapidly lead to the widespread adoption of HTTPS in the Internet. While undoubtedly a change for the better, traffic encryption has the downside of rendering deep packet inspection ineffective.

Unfortunately, efforts towards obtaining an encrypted traffic classifier have not yet yielded a sufficiently accurate solution. In lieu of better alternatives, firewall manufacturers decided to adopt SSL/TLS decryption, thus sacrificing user privacy in exchange for overall network security. For outgoing connections, this is achieved via a forward proxy. Each new TLS connection to an external server can be intercepted by the firewall. Next, the firewall would emit a certificate with the Distinguished Name of the external server but signed by the Certification Authority (CA) of the local organization. Since this CA is configured a priori on all internal network hosts, the TLS connection succeeds and the firewall can access the plaintext application data, then forward it to the intended endpoint via a separate TLS connection of its own. For incoming connections, the firewall is assumed to have access to the private key used by the internal server for decryption.

Although there are heuristic-based approaches such as Cisco's Encrypted Traffic Analytics (ETA) or Palo Alto's App-ID that rely on metrics such as packet length, packet source or transmission rate, these are usually applied when encountering proprietary encryption protocols (i.e., protocols other than SSL/TLS or SSH). Furthermore, application identification on plaintext traffic is usually achieved via traffic fingerprinting. Both Cisco's Network-Based Application Recognition (NBAR) and App-ID compare the payload against a database of application-specific features called signatures.

3.1 Problem statement

In this chapter we present DAF our user space firewall. Although we also implemented a kernel space variant that improves the overall performance, this chapter only describes the former, since the underlying concepts are similar. Thus, we consider the following research questions:

RQ1: Reliable endpoint data source for NGFWs? Micro-segmentation is a foundational component of zero-trust architectures. In contrast to traditional methods such as VLANs, micro-segmentation ensures network compartmentalization based on application layer visibility. However, most solutions today rely on the application classification features of NGFWs. We propose that one method of alleviating the over-reliance on TLS traffic decryption is by providing a pertinent, alternative data source that can uniquely identify endpoint applications and correctly associate this information with intercepted packets.

RQ2: Compatibility with loosely defined networks? In keeping with initiatives such as BeyondCorp, we take into account the fact that users could not only transition between internal networks, but also access resources from beyond the conventional network perimeter. As a result, we adopt a stateless Layer 3 traffic annotation mechanism and discuss possible improvements with regard to public network traversal.

RQ3: Impact on network integrity and performance? Ideally, a solution to the previous two questions would have minimal impact on both the network and individual hosts. Although certain concessions can ofttimes be made in favor of better security guarantees, we acknowledge the necessity for unobtrusive integration with pre-existing software stacks, as well as marginal downgrades in throughput. To this end, we aim to provide a prototype that is easily tunable to fit the unique requirements of each user.

In order to address these questions we propose DAF, a distributed application firewall prototype capable of filtering traffic based on application identity. Its main characteristics are as follows:

Originating process identification: In contrast to existing solutions, we adopt a more discerning stance on how we distinguish applications from one another. We do not simply rely on the content extracted from processed packets to infer the identity of the originator. Instead, we determine all processes that had access to the socket that was used to emit a certain packet (a common occurrence in highly complex applications). Moreover, we are able to account for any object that was mapped in the virtual address space of said processes and use them as rule matching criteria. This approach is motivated by recent advances in attaining reproducible builds for both open and closed source software. One application of this feature is the ability to filter traffic that originated from processes that use a known vulnerable version of a certain library (e.g. `libssl` for traffic encryption). Alternatively, whitelists can be defined based on sets of binaries that comprise specific applications.

Packet annotation: In order to communicate that a packet had passed through DAF on the originating endpoint, we attach a Message Authentication Code (MAC). The MAC is

embedded into the packet as an IP option, making our system connectionless. This allows the destination endpoint or any middleboxes that are aware of this annotation scheme to verify the authenticity of the received traffic. Alternatively, we attach an aggregate hash of all memory-mapped objects that comprised the originating process. This latter approach has a number of practical uses, e.g. allowing a system administrator to dictate the exact application and version one process is allowed to communicate with. Another function that could be served is that of a more general and reliable replacement of user-agent identification.

Compatibility with container-based isolation: Due to the widespread adoption of microservice architectures, we deemed it necessary to ensure that `DAF` could function across namespace boundaries. As a result, we demonstrate that our firewall can apply fine-grained filtration rules in Docker-based environments. This approach has the added benefit of isolating the firewall process from potentially malicious processes, to a certain degree.

We claim the following contributions:

- The implementation of `DAF` [1], a network firewall capable of filtering and authenticating traffic based on the identity of the endpoint processes, regardless of namespace isolation.
- The integration of `DAF` with pre-existing firewalls (`iptables`) and Intrusion Prevention Systems (`snort3`), illustrating its potential for interoperability with existing traffic filtering technologies.
- An analysis of the inherent limitations of `Netfilter Queue` and the optimization techniques necessary for achieving satisfying throughputs for both consumer grade hardware as well as datacenter environments.

3.2 Architecture

In this section we describe the architecture of `DAF`. A high-level overview of our solution can be seen in Figure 3.1. In order to operate our firewall, the first step would be for an administrator to create a whitelist of all the applications and their particular type of traffic that is allowed to be emitted. `DAF` allows the creation of classic firewall rules, such as permitting communication to/from specific IPs or ports, but extends this capability in one significant way: allowing applications to only accept traffic from other *specific* applications. Moreover, `DAF` can prevent attacks such as DNS rebinding by limiting the pool of accessible networked resources on a per-application basis. Once started, `DAF` begins to collect system-level event data, create a local system-state model, and process packets on behalf of the built-in firewall.

In order to correlate a process with its specific network traffic, we have identified multiple possibilities:

[1] Code available at https://github.com/RaduMantu/DAF

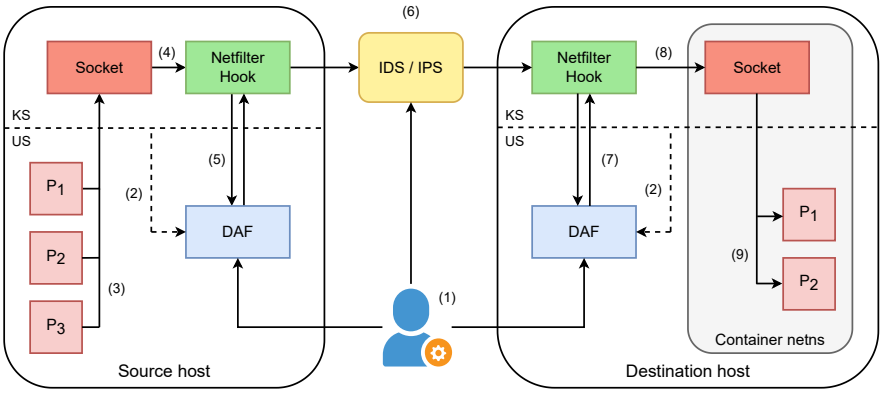

Figure 3.1: Tagged packet data path. Both endpoints utilize DAF.

- When considering a threat model where the attacker has limited user-level privileges, DAF can be implemented as an extension to the current firewall mechanisms (e.g., iptables), with no intervention on the kernel. This is our main implementation detailed in this paper on Linux.
- If we assume an attacker with elevated privileges, DAF requires a kernel level implementation, similar with existing antivirus solutions. We have implemented this model on Windows as a proof of concept.
- Alternatively, if we regard the attacker as capable of gaining control of his immediate environment but unable to break container isolation, DAF can safely run in userspace, in a superior namespace.
- If we assume that the attacker has kernel-level privileges and there is no support for virtualization-based solutions, DAF can either be implemented on an external hardware token or in a Trusted OS. The former implies that network traffic can no longer be attributed to a certain process due to the lack to OS-level information, but can still be attributed to a certain host and prevent spoofing. Meanwhile, the latter would require hardware support for separating trusted and untrusted protection domains (e.g., ARM TrustZone). These use-cases are outside the scope of this paper.

3.3 Evaluation

3.3.1 Experimental setup

The experiments described in this section have been carried out on the following hardware:

- **Intel NUC:** Intel Core i7-7567U CPU @3.50GHz, Intel I219-V 1-Gigabit Ethernet Controller, 8GB DDR4 memory @2400MT/s.
- **IBM System x3550 M4:** Intel Xeon E5-2650Lv2 CPU @1.70GHz, Intel 82599ES 10-Gigabit SFP+ Ethernet Controller, 32GB DDR3 memory @1600MT/s.

The reported throughput was derived from the `iperf3` output and was based on the application data transfer, excluding any headers. Both systems are running a minimal installation of Arch Linux with kernel version 6.7.0. We note that the *rootfs* of the IBM server was network mounted. However, due to file buffering and file system syncs being transmitted on a different 1-Gigabit interface, this fact does not impact our experiments.

Optimizations and trade-offs

During the development of `DAF`, in an effort to achieve a satisfying performance, we were confronted with a number of potential trade-offs. Although some had the capacity to improve throughput by up to an order of magnitude, each would incur a reduction in our security guarantees. Consequently, we decided to implement these optimizations as either build-time or run-time user options. Following is a list of said optimizations, as well as their associated CLI invocation flag.

- **Rescan prevention (R):** During the analysis of each packet, avoid scanning the virtual address space of each potential originator process, except during its initial identification. Subsequent queries will utilize the process state as it was ascertained during its first packet emission or reception. Dynamically loaded objects will not be accounted for.
- **Skip namespace switches (S):** Prevent switching network namespaces on consecutive rule evaluations that reference the same netns symlink. The symlink in question belonging to *procfs* would not be an issue in and of itself. However, external references to said symlink are often used in order to prevent the kernel from deleting the namespace after all processes belonging to it have terminated. This principle can be found implemented in the `netns` subcommand of `iproute2`.
- **Disable cached object ordering (build-time; U):** In order to match packets using an aggregate hash that is representative for a certain program, the comprising objects must be arranged in a deterministic manner. To this end, we decided to sort them alphabetically based on their absolute paths. In our implementation, we cache the hashes of all objects belonging to a process as an unordered set, where operations have an average constant-time complexity. When retrieved by the Process Pool Analysis component, an ordered set (logarithmic complexity) is initialized from the contents of the unordered set, relying on its key comparison implementation to order the elements. During our experiments we discovered that we have initially underestimated the cost of this operation. As a result, we offer the choice of working directly with the unordered set in exchange for the ability to match whole programs, not just individual objects.
- **Batch verdict communication (b, B):** During our experiments we concluded that the packet data transfer between kernel space and user space is not a costly operation in and of itself. On the other hand, reporting the verdict for a given packet back to the `Netfilter Queue` module would block the process for what would at times amount to approx. 30% of the total execution time. As a result, we decided to implement a

verdict batching mechanism. Although capable of significantly improving the performance of DAF, we note that this optimization is not as straightforward to utilize as the previously described ones, its performance being contingent on the type of processed traffic. Thus, we define user-adjustable upper limits for both the number of batched packets, and the amount of time that the verdict of any packet is allowed to be delayed (see Subsection **??**). Albeit difficult to suggest generic values that would guarantee an adequate performance increase regardless of the type of workload, we found it prudent to carry out forced verdict transmissions whenever a TCP SYN or TCP PSH is encountered. The reason behind the former is to avoid delaying the establishment of new TCP connections. As for the latter, postponing the transfer of what is expected to be a complete message to the endpoint process could provoke unwanted transmission delays at the application level.

3.3.2 Netfilter Queue tuning

When selecting the parameters for the **batched verdict communication optimization (b,B)**, we first determine the lower bounds for the maximum batch size and transmission timeout. Choosing the values for either of these parameters while falling short of this threshold can detract from the performance improvement of the optimization, to the point of completely negating it (e.g. size=1).

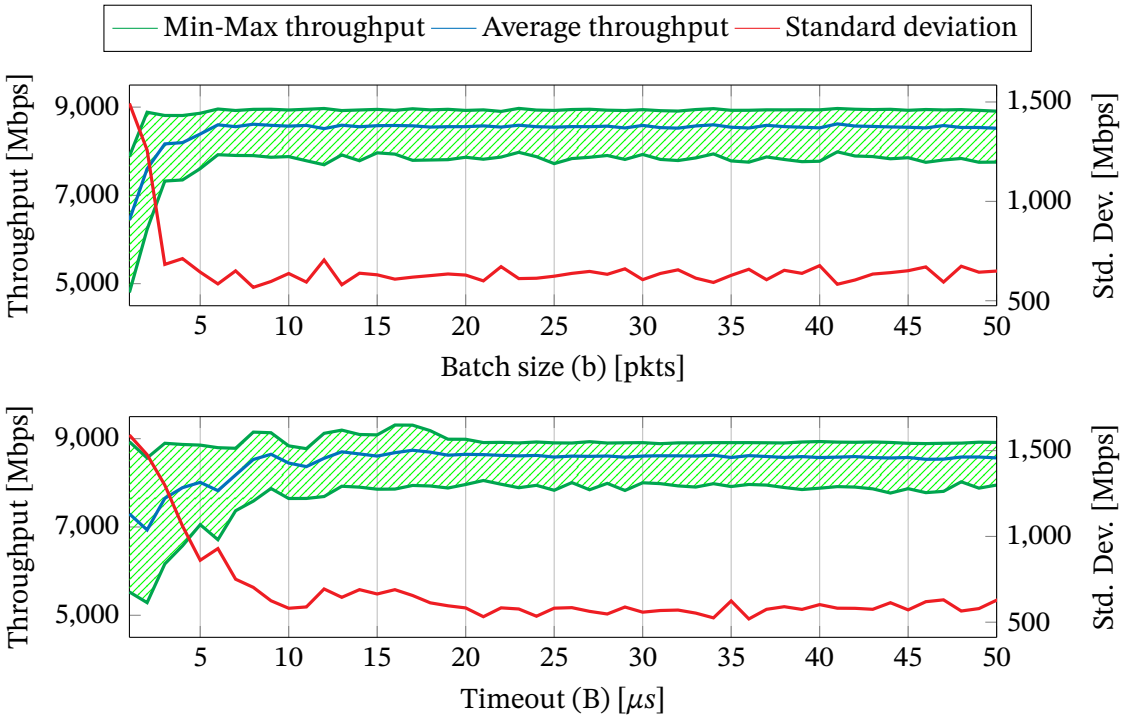

Figure 3.2: Influence of verdict batch size & timeout on throughput.

In order to empirically determine the aforementioned thresholds, we performed six rounds of experiments, varying the maximum batch size by 1 packet and the timeout by $1\mu s$ in the

1-100 range (thus obtaining a dataset of 60,000 throughput values). Figure 3.2 illustrates the impact of each of the two tuning parameters via the average throughput on a 10Gbps link with constant MTU. In order to exemplify the effect of one tuning parameter independently from the other, the dimensions corresponding to the second parameter as well as the experiment round had to be collapsed. Consequently, each average throughput data point was calculated over a subset of 600 values representing the splice obtained from fixing the value on the abscissa.

For the minimum-maximum intervals we decided to fix both the experiment round and one of the tuning parameters, in order to then average the per-experiment minimums and maximums. The motive behind this was to de-emphasize outliers. To compensate for this and correctly represent the throughput variations, we included the standard deviation obtained from the same data slice that was used to calculate the average throughput. Here, we mention that the value range for both plots was truncated to 1-50 in order to better illustrate the impact of suboptimal parameter choices. The subsets used in the calculation of the average throughputs have not been proportionately truncated.

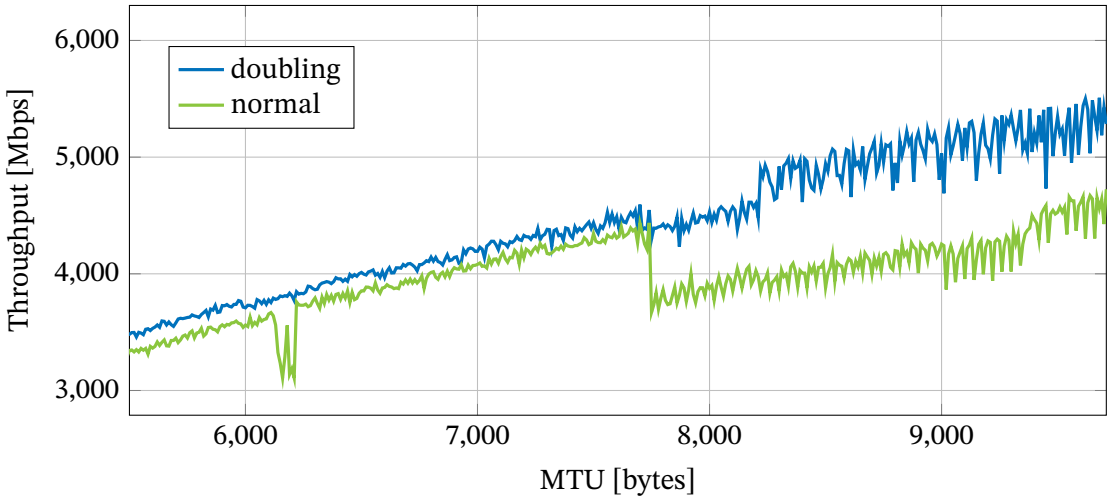

Figure 3.3: Effect of introducing dynamic resizing of Netlink socket buffer.

One other value that we concluded needed to be fine tuned is the **Netlink socket receive buffer size**. While this socket is used for communication between userspace and the `Netfilter Queue` kernel driver, its receive buffer is also used to store packets that are pending a verdict, only to later be reintroduced in the network stack. In the eventuality that this buffer fills (a common occurrence when the userspace application is slow to reach a verdict) new packets are discarded and the following `read()` operation fails with the `ENOBUFS` error code, as an informative measure. Ignoring this problem can lead to aliasing issues that normally manifest as illustrated in Figure 3.3. Insufficient TCP socket buffer sizes may also lead to similar performance downgrades. In response to these events, we assess the current Netlink socket buffer size and double it. This action is performed using the `SO_RCVBUFFORCE` socket option as to override the system maximum socket size. We note that after doubling the buffer size from userspace, the kernel doubles it once more in

order to allow for bookkeeping overhead.

Performance analysis

Our performance analysis of DAF is primarily based on `iperf3` throughput tests between a client running our firewall and a server on another identical host. The two hosts are directly linked, with no additional equipment serving as middleboxes. We note that the values we report represent the average throughput over 10s sessions, averaged once more over 10 and 5 rounds of experiments on the IBM server and NUC system respectively. On the 10-Gigabit servers, we decided to manually increase the default and maximum TCP socket buffer sizes, a common practice in such cases. Although in our final reports these are set to 32MB, we did not perceive any noticeable performance gain for values higher than 8MB. The NUC desktop computers utilize the default maximums of 4MB for the send buffer and 6MB for the receive buffer.

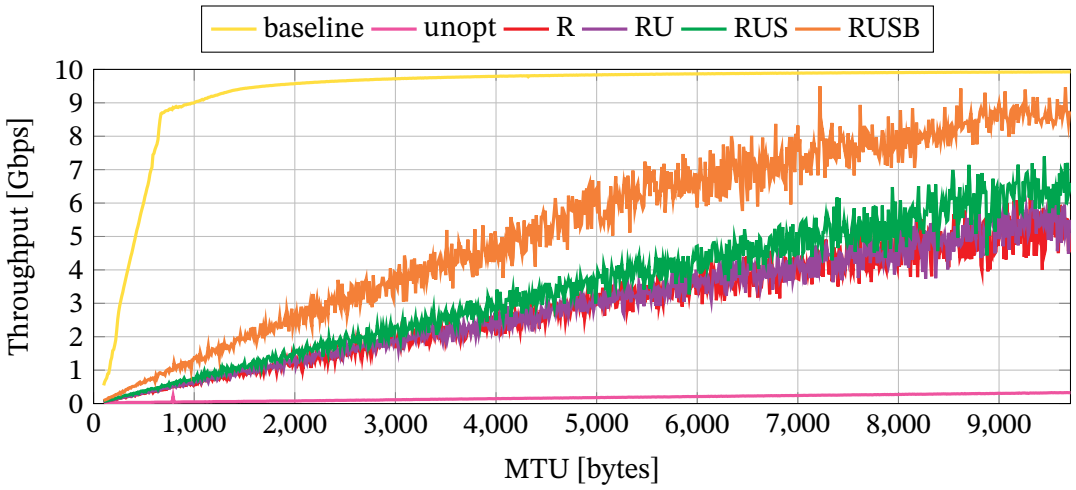

Figure 3.4: DAF throughput on IBM system (`io_uring` version).

Figure 3.4 illustrates the performance gain of each optimization. Here, we note the exceedingly high variations. We attribute these to the use of the `io_uring` interface for asynchronous I/O operations. In our implementation, we decided to launch a kernel thread on a different CPU core that continuously polls the system call submission queue in DAF. Prior to this revision, we employed an `epoll` hierarchization system to permit us the prioritization of certain events (e.g., system state model updates) over other. However, profiling sessions revealed that our firewall spent over 50% of the time waiting for `epoll` events, limiting the average throughput with full optimizations to approx. 5.4Gbps. Our experimental results using this version can be found in Figure 3.5. In it there is an additional optimization, namely *Uniform Prioritization (u)*. This optimization would circumvent the `epoll` hierarchy in order to give equal chances to both packets and system events to be processed, at the cost of potentially using an outdated system state when reaching a singular packet verdict. Once we transitioned to the `io_uring` implementation, this optimization became redundant and was elliminated. As a reulst, bynot only offloading the majority of I/O op-

erations (consisting mostly of packet reads from the netlink socket) to a secondary core but also drastically limiting the amount of context switches between protection domains, we have increased the average throughput to approx. 8.8Gbps. We believe that the gap between this and the baseline throughput can be bridged by extending `libnetfilter_queue` to allow asynchronous batched verdict transmission.

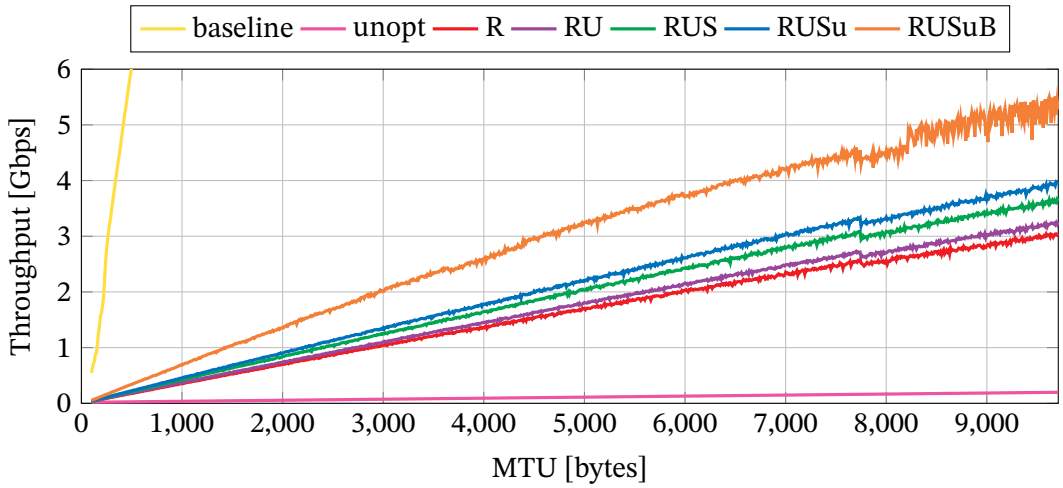

Figure 3.5: `DAF` throughput on IBM system (`epoll` version).

In the same figure we also depict the throughput of `DAF` on the 10-Gigabit server while also calculating and attaching SHA256-HMACs of the Layer 4 payload as IP options. Notice how we forgo the use of verdict batching since it would not permit us to re-inject the modified packet into the kernel, strictly due to a `Netfilter Queue` API limitation. Once again, we believe that this limitation could be alleviated by adding a simultaneous packet override through an `iov` interface. However, this would also require changes not only to `libnetfilter_queue` but also the `Netfilter Queue` driver, both of which we conside outside the scope of this paper. During this experiment we manually enforced a smaller Maximum Segment Size (MSS) for `iperf3` in order to ensure that the newly introduced 36-byte IP options section would not exceed the MTU at any given time. Note however, that `iperf3` clamps the MSS values within the 88-9216 range, thus leading to severe performance downgrades due to fragmentation. Under these conditions, we register a peak average throughput of approx. 835Mbps.

Chapter 4

Conclusion

In support of this thesis, we claim a number of contributions that address emerging issues with firewall technologies. These are as follows:

1. **Evaluation of protocol option acceptance in the Internet:** We implemented `ops-inject`, a network traffic annotator based on Netfilter Queue that can attach arbitrary interpretations of IP, TCP or UDP options to outgoing packets. We then utilized this tool to perform all-to-all reachability tests between 21 hosts with a diverse geographic distribution, from moultiple cloud providers. This study helped clarify the current state of options acceptance in the Internet and informed our choice of conduit for network traffic tagging.

2. **Implementation of an application-aware distributed firewall:** We implemented `DAF` and `AppScout`, two state of the art firewalls capable of filtering traffic based on the identity of the endpoint application without performing deep packet inspection or performing traffic decryption. Through these two alternatives, one bound to user space and the other designed as a kernel module, we attain near linereate throughputs while also accounting for microservice architecutres by supporting Linux namespace transparency.

3. **Evaluation in realistic environments:** We tested our two Linux variants on both desktop systems with 1Gbps NICs, as well as datacenter environments with 10Gbps NICs. We performed an in-depth inspection of Netfilter Queue, an entrenched kernel mechanism that allows IDS and IPS to perform deep packet inspection in userspace. Based on our observations and experience researching this topic, we propose a number of improvements that could further increase its performance and bridge the gap between a kernel-only solution and a userspace firewall. Additionally, we identify potential security risks that this technology may unknowingly introduce.

4. **Packet annotation scheme and compliance verification:** We propose a packet annotation scheme based primarily on IP options. This traffic tagging mechanism is fully implemented in `DAF` and serves to enable middleboxes to verify the compliance of individual pakets with the firewall policies of its originating system. We also provide a Windows prototype that serves to demonstrate the portability of our solution and the possibility of utilizing TCP options as a conduit in restrictive networks where IP options are blocked.

5. **Network integration solutions:** In order to facilitate easier adoption, we designed

DAF with as few deployment requirements as possible. This includes a lack of kernel components, making it an ideal choice for systems employing the lockdown LSM. Additionally, we provide a `snort3` plugin for matching and verifying HMAC annotations as IP options, as well as `iptables` plugins serving the same purpose for both the IP option conduit, and for the TCP option conduit. These extensions allow users to deploy our firewall on a small subset of hosts and perform on the path compliance verification using conventional tools.

6. **Overview of the state of the art in Network Fuzzing:** By applying our previous experience in fuzzer development, we explore the current challenges of employing network fuzzing as a method of reinforcing the trust placed in a whitelisted application. This serves as our recommendation for adopters of DAF that need to deploy their own proprietary solutions or custom builds of OSS without deterministic compilation.