U.P.B. Sci. Bull., Series C, Vol. 86, Iss. 3, 2024 ISSN 2286-3540

LABEL-BASED FIREWALLING ON WINDOWS

David Gherghiță[1], Radu Mantu[2], Mihai Chiroiu[3], Nicolae Țăpuș[4]

In this paper we present a network traffic labeling and filtering system. This system is based primarily on the Windows Filter Engine and can provide proof of origin and payload integrity to all TCP traffic. We accomplish this by annotating traffic using TCP options, a widely accepted framework for providing protocol extensions, that is also commonly routable in the Internet, and guarantees compatibility with network stacks that do not implem4nt support for our annotation scheme. Furthermore, we supply a companion Xtables module and `iptables` *plugin for Linux-based distributions that can perform matches on correctly labeled traffic. Finally, we explore the alternative of filtering traffic on egress based solely on the identity of the originating Windows application.*

Keywords: Windows Filter Engine, traffic labeling, firewalling

1. Introduction

Application fingerprinting based on network traffic is a crucial requirement for correctly managing computer networks and enforcing security policies within them. In the early days of the internet, the Layer 4 ports were generally indicative of the application that was bound to them. However, the rapid maturation of these technologies has created uncertainty and lead to new, telemetry-based solutions [3] to emerge. Being a long-term goal of companies such as Cisco that have had a vested interested for well over a decade [16], machine learning techniques have recently become more prevalent in the identification of endpoint applications [5, 15]. Nonetheless, these technologies are yet to see large scale adoption in firewalling solutions due to inherent limitations, stemming in large part from the constantly accelerating diversification of web applications.

In this paper we propose a different approach to ensuring network security: the distribution of responsibility [7] to multiple hosts in the network.

[1]MSc., Faculty of Automatic Control and Computers, University "Politehnica" of Bucharest, Romania, e-mail: `david.gherghita@upb.ro`

[2]Research Assistant, Faculty of Automatic Control and Computers, University "Politehnica" of Bucharest, Romania, e-mail: `radu.mantu@upb.ro`

[3]Assistant Professor, Faculty of Automatic Control and Computers, University "Politehnica" of Bucharest, Romania, email: `mihai.chiroiu@upb.ro`

[4]Professor, Faculty of Automatic Control and Computers, University "Politehnica" of Bucharest, Romania, email: `nicolae.tapus@upb.ro`

Instead of centralizing the firewalling mechanism and compensating with increasingly costly and potentially inaccurate fingerprinting techniques, we rely on the operating system of each host to capitalize on its extensive control over the endpoint processes in order to correctly identify the afferent application. Additionally, we attach a proof of compliance to each emitted packet for other devices to verify.

This proof would demonstrate to other networked entities that each packet that originated from a certain host had been analyzed by an instance of our firewall. Moreover, we do not make the assumption that all potential endhosts and middlebox devices are aware of our annotation scheme. While some prior solutions [10, 17] have appended their own version of a proof of compliance to the payload, we argue that this would most assuredly break the application-level communication between incompatible devices. We emphasize this problem due to recent developments in corporate network infrastructure [9, 14] where devices tend to seamlessly transition between networks. Without a guarantee that all guest devices would at least have knowledge of our annotation scheme, we decided to ere on the side of caution and only use the formal protocol extension mechanisms that were already in place (i.e., protocol options). This approach would allow unknown protocol options to be quietly ignored by the network stacks, so long as they correctly implement the afferent standards.

In turn, this raises the question of what protocol to integrate our annotation with. We have identified three choices. Under ideal circumstances, the IP protocol [1] would be the most beneficial since acting at Layer 3 would abstract the complexities that are inherent to higher layer protocols. Nonetheless, it has been previously proven that IP options cannot be reliably routed through the Internet [4]. Since this fact has remained unchanged for the past 20 years, even more recent architectures [20] strip these options before routing traffic towards external networks. Taking this information into account, the only viable remaining options are UDP and TCP. We decided to forego implementing support for UDP options [13] due to the fact that they are a recent proposition that is still pending standardisation by the IETF and that there are known inconsistent behaviours in checksum calculation [19] when this extension is encountered. TCP options on the other hand have been known to function reliably [6] for the past decade, mostly due to their necessity in ensuring adequate performance in Wide Area Networks. The downside of using TCP options is that the size of our annotation is effectively limited to 36 bytes due to design considerations dating back to 1981 [2].

We claim the following contributions and make the relevant source code publicly available[1] for any interested party.

- The design and implementation of a network traffic labeling system as a Windows 10 Filter Engine kernel module.

[1]https://github.com/david-gherghita/label-based-firewall-for-windows

- The implementation of a companion Xtables module and `iptables` plugin to bridge the gap in compatibility between Windows and Linux-based distributions.
- The assessment of the viability of using the originating process identity as a criteria for packet labeling and filtration.

The reminder of this paper is structured as follows. Section 2 describes the implementation details of each component comprising the Windows kernel module. In Section 3 we present our testing methodology, functional evaluation and performance considerations. Section 4 presents a comparison between our solution and existing works. Section 5 concludes this paper.

2. Implementation

2.1. Interaction with the Windows Filter Engine

One of the core elements that comprise our solution revolves around correctly identifying the process that generated an outgoing packet. Since this information is usually associated to the socket that performed the `send()` operation, it follows that it would be readily available at the Application Layer Enforcement (ALE) stage of the WFP Filtering Engine (see Figure 1). More specifically, a callout from the `FWPM_LAYER_ALE_AUTH_CONNECT_V4` layer is capable of blocking both outgoing TCP connections, and non-TCP traffic based on the first emitted packet.

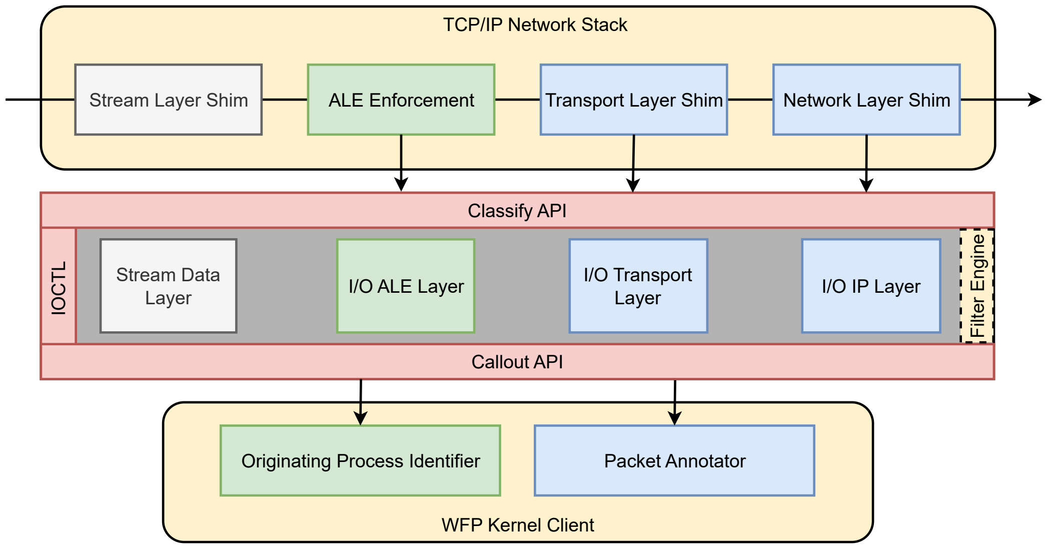

FIGURE 1. Kernel module integration with the Windows Filtering Platform.

Nonetheless, at this layer it is impossible to accomplish the other desired function of our firewall, namely packet annotation. The reason for this is that at the moment when the ALE hook is reached, the packet had only traversed the Stream Layer Shim of the TCP/IP network stack. If our goal was to modify the application data, doing so at the Stream / Datagram Data Layer would have been sufficient. However, our annotation scheme is reliant on the existence of network and transport layer protocol headers. Since these headers are constructed after the ALE connection management stage, we considered the `FWPM_LAYER_OUTBOUND_IPPACKET_V4` and `FWPM_LAYER_OUTBOUND_TRANSPORT_V4` layers for manipulating the IP and TCP options sections, respectively. Although these layers offer the possibility of affecting the construction of protocol headers, we note that the information relating to the originating process is no longer available at this stage.

In order to address this issue, we decided to register our Kernel Client both with the ALE layer and with the IP or Transport layers, as dictated by the user. The former callout would collect the relevant process information and attach it to the data flow as a context, thus making it available to future callouts. The latter callout would access said process information via the associated context, compute the hash-based message authentication code and attach it as an experimental option.

2.2. Identification of the originating process

One similarity of particular note that we identified between the Windows and Linux networks stacks consists of how network traffic is attributed to a certain process. In both cases, the most accurate association can be made during the execution of the `send()` system call (or any equivalent) by assessing the PID of the calling process. However, the data that is transferred from user space during this operation can not be equated to the payload of a single packet, mainly due to mechanisms such as the Generic Segmentation Offload (i.e.: software enabled packet fragmentation). Once this data enters the network stack, its processing may be deferred to a kernel thread. Consequently, the relation of a given packet to a userspace process can be ascertained only via its backing socket structure.

Herein lies the problem. Both the `Winsock` and `sk_buff` structures only identify one owning process: that which first instantiated the resource. However, access to said resource is granted to userspace application through file handles / descriptors, which by default are inherited by children on `exec()`. We note that aside from this, there are also other methods of sharing file descriptors between processes (e.g.: ancillary sockets that transfer file descriptors, etc.)

Our solution takes advantage of the fact that part of the processing done by the TCP/IP stack is performed by the same thread that originally invoked the `send()` operation. As a result, we are able to extract the real PID of the originating process anywhere in Stream Layer Shim or the ALE, as well as their associated Filter Engine hooks and callouts.

2.3. Packet annotation

Following the identification of the originating process, our system was intended to generate proof of authenticity and integrity of the payload. To this end, we utilized the BCrypt family of cryptographic primitives to calculate a SHA256 HMAC for each packet. We selected this variant instead of NCrypt due to the fact that the HMAC shared secret could be considered to have already been loaded in memory as a parameter to the kernel module. Nonetheless, we regard NCrypt as a viable alternative if the requirement of integrating a Key Storage Provider ever arises. Regardless, the digest of the Message Authentication code was calculated based on the TCP payload as well as the Sequence Number, so as to prevent replay attacks.

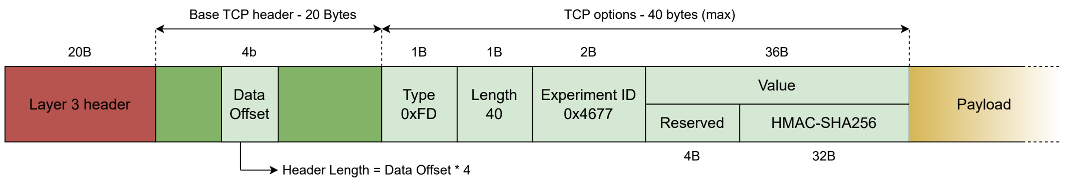

FIGURE 2. Experimental TCP option format used in annotation.

To conclude the annotation process, we would attach the 32-byte digest to the TCP header, as a TCP option (see structure in Figure 2). Each TCP option is a Type-Length-Value (TLV) tuple that is appended to the base header and any other pre-existing options. Due to it being an experimental option (i.e., not previously registered with the Internet Assigned Numbers Association), we assigned ours the RFC3692-style [8] codepoint `0xFD`. At the same time, we set the length field to the maximum value of 40 bytes. This decision was motivated by the severe space requirements of the digest, combined with additional protocol-enforced fields, and relative to the potentially variable length of other common options such as Window Scaling or Selective Acknowledgement. Because we could not guarantee that our option would be able to coexist with all the other (e.g., the aforementioned) we deemed it necessary to erase all other options present in the initial handshake for both incoming and outgoing connections. This would prevent the successful negotiation of employing any protocol extensions (aside from ours) by the operating system from that point onwards. In compliance with the Shared Use of Experiemntal TCP Options RFC [12], we assigned our experiment the ID `0x4677`. Aside from the 32-byte HMAC digest value, the Value element also consists of a 4-byte field reserved for future use. We opted for this approach instead of padding the options section with NOPs because middleboxes may remove them if encountered in groups of more than three. This truncation of the options section usually occurs since NOPs are usually employed to offset options that must be 32-bit aligned, and having a 32-bit padding sequence is not only redundant but can also be considered a type of Denial Of Service attack meant to waste system resources with TCP option decoding. Ensuring that the TCP header length is consistently 60 bytes guarantees that no other TCP options are injected on the path and simplifies the payload offset calculation process.

3. **Evaluation**

3.1. **Experimental setup**

Our solution was developed on a bare-metal Windows 11 Pro 22H2 and cross compiled with MSVC v143, to be tested on a Virtual Machine running Windows 10 Enterprise LTSC Build 17763, with a 1Gbps emulated network controller. The decoupling of development and testing environments was necessary for allowing the insertion of self-signed kernel modules and to minimize the risk of development setbacks.

A second Virtual Machine running Ubuntu Bionic was used to test the compliance of our annotation scheme with Linux-based distributions (see Subsection 3.**2**).

3.**2**. **Label verification and filtration**

In order to demonstrate the efficacy of our annotation scheme, we implemented an Xtables module for the Netfilter system in Linux. This system comprises a set of hooks placed within the network stack of the Linux kernel, and can be used to attach traffic filters and mutators. In other words, the Netfilter system represents the basis on which `iptables` was built. Correspondingly, Xtables is an aggregate name for the callback subsystem that unifies the IPv4, IPv6, ARP and Ethernet Bridge toolset backends, collectively now known as `nftables`.

Our extension also consists of a user space component (i.e., an `iptables` plugin library) that expands the parsing capabilities of the base binary to include custom match criteria and auxiliary information, such as the pre-shared secret used in the HMAC operation. This plugin marshalls the aforementioned parameters and sends them to our Xtables module, thus creating a new evaluation context, specific to a singular rule. When the `match()` callback is invoked within a certain context, our module re-calculates the SHA256-HMAC of the TCP payload and Sequence Number, compares it to that which is stored in the experimental TCP option, and finally reports the result to the caller. Since our Xtables module only implements the `match()` callback, it can be paired with any other match criteria and jump targets available to `iptables`, be they related to filtering, logging, address translation, etc.

For the purpose of testing our Linux-based filtration mechanism, we have implemented two HTTP and SSH Python servers based on `flask` and `paramiko` respectively. While the former exposed multiple routes, each providing an arbitrary implementation for different HTTP request types (e.g.: GET, POST, DELETE, etc.) the latter acted as an echo serer over a secure communication channel. We concluded that neither type of traffic was impeded by the addition of the HMAC annotation. Additionally, we also assessed the viability of Windows-side filtration of these protocols, as well as ICMP Echo Requests / Replies and raw data over UDP but without annotating outgoing traffic.

3.3. Performance considerations

Despite the security assurances that our system provides, a decline in throughput is to be expected. This can be observed in Figure 3, by comparing the average data throughput of ten second `iperf3` TCP transmissions, with and without annotations. These measurements have further been averaged across five different rounds of testing. Although the process identification feature is not particularly costly from a computational standpoint, calculating the SHA256 digest and replacing the contents of the send buffer by injecting the experimental TCP option can be detrimental. Based on our experiments we estimate that our kernel module imposes an upper bound of approx. 400Mbps on outgoing network transmissions. Note however that this limit may vary depending on the characteristics of the underlying testing environment (e.g., hardware implementation of cryptographic primitives, CPU frequency, etc.)

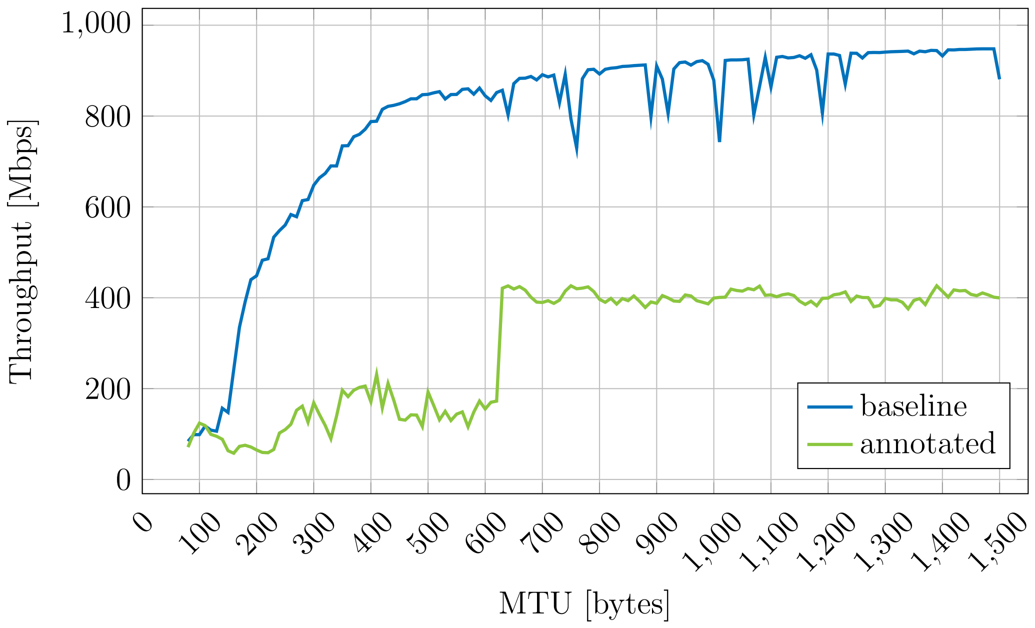

Figure 3. iperf3 data throughput with packet annotation.

4. **Related Work**

`VMwall` [11] is a Xen-based application-level firewall proposed by Srivastava et al. in 2008. Their implementation performs application identification during the connection setup phase by performing Virtual Machine Introspection, thus associating the socket used for network communication with the name of the initiating process. Our solution differs in that we dynamically identify the originating process, thus accounting for situations where socket handles are shared by the connection initiator with other entities. Similarly to `VMwall`, we leverage protection domains to isolate our module from potentially malicious applications. However, our solution operates at ring 0, while theirs runs in hypervisor space.

`Assayer` [10] is a solution dating from 2012 by Parno et al. that implemented packet annotation with application-specific counters. These counters range from the average size of mails sent by an application, to the number of host said application connects to, etc. These metrics are gathered by privileged (i.e., hypervisor space) modules to a minimal hypervisor that provides hardware attestation, and attached to outgoing packets by appending them to the end of the payload. While our solution is more specialized, aiming to provide a set of security guarantees, the annotation scheme that we utilize is more flexible. While `Assayer` requires all endpoints to be aware of the annotation scheme, our implementation is more flexible and does not obstruct the communication if one party lacks support for our system.

In 2012, Zhao et al. [18] noticed a lack of labeled network traffic captures and devised an annotation scheme for constructing such a dataset. Their solution intercepted outgoing traffic via the Network Driver Interface Specification (NDIS) hooking mechanism and attached an arbitrarily defined single-byte application identifier in the Differentiated Services (DS) field of the IP header that would be recorded at the destination gateway. The difference of our choice of filtering framework (i.e. WFP) is motivated by the fact that our annotator operates at Layer 4 while theirs requires access to the Layer 3 header. Moreover, while this Layer 3 annotation scheme encompasses a larger category of traffic and not just TCP, redefining of the DS field can interfere with classification conventions within DS domains.

`BorderPatrol` [20] is another instance of traffic labeling system based on application identity. Aimed specifically at Android devices, this solution leverages the Dalvik virtual machine and the Xposed Framework to dynamically instrument method and constructor calls. All instances of socket creation are monitored by a Context Manager process that either blocks the attempt based on classpath information or attaches a known identifier of the calling method, together with the MD5 digest of the APK to each packet as an IP option. Similarly to our solution, `BorderPatrol` is able to obstruct data transmission based on application information. Although able to apply finer grained (i.e., method-level) match criteria, `BorderPatrol` is designed to operate in a mostly trusted local network, where the traffic filtration is almost guaranteed to occur on the user device and no bad actors are liable to modify the annotated data in transit. Additionally, their architecture requires that all IP options are dropped when exiting the local network. Meanwhile, all TCP traffic that is labeled by our solution is almost guaranteed to be successfully routed in the Internet.

5. **Conclusion**

In this paper we assessed the viability of leveraging the kernel-level knowledge of endpoint hosts to aid in classifying network traffic based on its originating application. To this end, we implemented a Windows Filter Engine kernel module capable of identifying all processes that accessed the `Winsock` object, and not just its creator.

Furthermore, we devised an annotation scheme that ensures payload integrity and authentication for TCP traffic. Our design takes into consideration backwards compatibility with kernel stacks that are unaware of our labeling mechanism, preventing our protocol extension from disrupting application-level communication while still allowing middleboxes to perform their own verification.

Finally, we implement an Xtables module that permits verifying the attached proof of compliance. This module serves as a singular, independent component in the packet matching logic employed by userspace tools such as `iptables` or `nftables`.

REFERENCES

[1] Internet Protocol. RFC 791, September 1981.

[2] Transmission Control Protocol. RFC 793, September 1981.

[3] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, 2006.

[4] Rodrigo Fonseca, George Porter, Randy Katz, Scott Shenker, and Ion Stoica. Ip options are not an option. Technical report, Technical report, EECS Department, University of California, Berkeley, 2005.

[5] Jorge Luis Guerra, Carlos Catania, and Eduardo Veas. Datasets are not enough: Challenges in labeling network traffic. *Computers & Security*, 120:102810, 2022.

[6] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 181–194, 2011.

[7] Sotiris Ioannidis, Angelos D Keromytis, Steve M Bellovin, and Jonathan M Smith. Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199, 2000.

[8] Dr. Thomas Narten. Assigning Experimental and Testing Numbers Considered Useful. RFC 3692, January 2004.

[9] Barclay Osborn, Justin McWilliams, Betsy Beyer, and Max Saltonstall. Beyondcorp: Design to deployment at google, 2016.

[10] Bryan Parno, Zongwei Zhou, and Adrian Perrig. Using trustworthy host-based information in the network. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pages 33–44, 2012.

[11] Abhinav Srivastava and Jonathon Giffin. Tamper-resistant, application-aware blocking of malicious network connections. In *International Workshop on Recent Advances in Intrusion Detection*, pages 39–58. Springer, 2008.

[12] Dr. Joseph D. Touch. Shared Use of Experimental TCP Options. RFC 6994, August 2013.

[13] Dr. Joseph D. Touch. Transport Options for UDP. Internet-Draft draft-ietf-tsvwg-udp-options-23, Internet Engineering Task Force, September 2023. Work in Progress.

[14] Rory Ward and Betsy Beyer. Beyondcorp: A new approach to enterprise security, 2014.

[15] Baris Yamansavascilar, M Amac Guvensan, A Gokhan Yavuz, and M Elif Karsligil. Application identification via network traffic classification. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 843–848. IEEE, 2017.

[16] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*, pages 250–257. IEEE, 2005.

[17] Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazieres. Securing distributed systems with information flow control. In *NSDI*, volume 8, pages 293–308, 2008.

[18] Caiyun Zhao, Lizhi Peng, Bo Yang, and Zhenxiang Chen. Labeling the network traffic with accurate application information. In *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4. IEEE, 2012.

[19] Raffaele Zullo, Tom Jones, and Gorry Fairhurst. Overcoming the sorrows of the young udp options. In *TMA*, 2020.

[20] Onur Zungur, Guillermo Suarez-Tangil, Gianluca Stringhini, and Manuel Egele. Borderpatrol: Securing byod using fine-grained contextual information. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 460–472. IEEE, 2019.