Improving the Session Table Handling of Stateful Firewalls to Achieve Constant-Time Packet Filtering

Dennis Tudenhöfner
University of Applied Sciences Esslingen
detuit00@hs-esslingen.de

I. Introduction

In today's industrial networks, control systems are limited to small and isolated networks, as the communication is often time-critical and tightly clocked. With the introduction of Time-Sensitive Networking (TSN), an extension to standard Ethernet, industrial control systems can be integrated into large corporate networks. TSN enables deterministic forwarding of of packets with strict timing guarantees by minimizing the variation of latency (jitter).

With respect to the trend of growing industrial infrastructures, IT security measures are becoming increasingly essential. In recent years, companies struggled with severe production downtimes, damaged plants, and data loss due to cyber attacks. A basic and suitable measure against attacks is the separation of the network into zones. This concept is called network segmentation. Network administrators can place firewalls between the zones to analyze and filter the traffic. The mechanisms to filter packets with firewalls are implemented either in hardware (e.g., using an application-specific integrated circuit, ASIC) or in software (i.e., using a CPU). Filtering in hardware only increases the delay and jitter slightly, which makes it a suitable solution for industrial networks [1]. However, filtering in hardware has its drawbacks as well: vendor-specific implementations, less flexibility in filtering rules, and a limited total amount of rules. Filtering in software, on the other hand, strongly increases the jitter of the packet processing time, but offers more flexibility in its implementation and runs on commodity hardware. Due to the limitations of hardware-based firewalls, our goal is to improve the filtering performance of software firewalls.

So far, there are no software firewall implementations available that guarantee constant-time packet filtering in order to satisfy the latency requirements of industrial networks. To achieve constant-time packet filtering with software firewalls, we adapt existing packet filtering mechanisms. Firewalls usually rely on two packet filtering mechanisms:

1) Stateless filtering: Stateless filtering is designed to filter packets by comparing packet header fields with a set of rules. To allow or block a packet from passing through the firewall, the 5-tuples (source IP address, source port, destination IP address, destination port, protocol) of the rules are matched against the header fields of the arriving packet. Based on the result, the firewall takes an action defined in the matched rule (e.g., allow or block the packet).

2) Stateful filtering: Stateful filtering keeps track of the state of existing sessions. It uses a session table, also called connection state table, to track packets belonging to a known active session. The firewall always matches incoming packets with the 5-tuple entries of the session table. When it finds a matching entry, it allows the packet to pass through and updates the corresponding table entry with the latest session state information. If the packet does not match any entry, the firewall processes it using the stateless filtering mechanism and decides whether to allow or block the packet.

Due to the fact that stateless firewalls process rules one after another, the latency for packet processing increases linearly with the growing number of rules. Hence, an increasing number of rules can slow down the packet processing time of the firewall excessively. We intend to use stateful filtering as a basis for packet filtering with low jitter. Storing the session table as a hash table enables the firewall to perform session table lookups in constant time. The firewall extracts and hashes the 5-tuple of an arriving packet and uses it to access the corresponding table entry directly. The firewall is not required to iterate over every single field of the table entry anymore. Therefore, stateful filtering is better suited for time-critical applications than stateless filtering. However, the problem with stateful filtering is that the implementation, as it is found in software firewalls, is unsuitable for time-critical applications. For example, initial packets of a session pass through the jitter-prone stateless filtering mechanism instead of the stateful filtering mechanism.

Our approach is to modify and extend the stateful filtering mechanism in order to make it suitable for time-critical applications. Hereby, we want to use the existing high-performance software networking stack FD.io VPP [3] with the included Access Control List (ACL) plugin as a proof-of-concept (PoC) implementation.

II. Related Work

Packet filtering is implemented either in software or in hardware. One difference between both implementations is the delay of forwarding packets caused by their processing time. Wüsteney *et al.* [1] compare the filtering performance between hardware- and software-based firewalls regarding the usability in TSN networks. According to them, software-based implementations are more affected of varying CPU load which causes additional jitter. Zvabva *et al.* [4] present measurements of network packet latency, jitter and packet loss caused by the introduction of industrial firewalls when the network is segmented with the concept of zones, security levels and conduits according to the security standard IEC 62443.

Schramm [2] presents and implements three ideas to achieve low jitter and latency on software firewalls. However, he does not consider stateful sessions in his firewall design. The firewall interrupts the stateless check of the packet of a new session after a certain time limit exceeds. After the time limit exceeded, the firewall forwards the packet without complete check to limit the maximum jitter. Assuming this packet belongs to a session that will be stored in the session table, this packet passes through the stateless filtering mechanism only once. The following packets of the session only pass through the stateful filtering mechanism, assuming the first packet was checked completely and is allowed. This behavior faces a security issue, which we want to avoid in this work.

Another research area focuses on session table enhancement techniques to improve the processes to create, delete, and lookup session table entries. Chomsiri *et al.* [5] propose a session tracking system with a hash table for tree-rule firewalls that reduces memory consumption and processing time. Moreover, they show that the processing speed of their stateful firewall implementation is much faster than iptables. The tree-rule firewall is introduced by the same authors in [6]. It is a modified firewall that organizes its rules in a designated tree structure, not in lists. The work presented in [7] proposes a hybrid firewall implementation that takes advantage of both the tree-rule and stateless filtering mechanism to ensure high packet processing speed without rule conflicts. In addition, the authors added a feature that moves frequently matched rules to higher positions in the rule list automatically. They measure the firewall speed drop (in terms of packets per second, and megabytes per second) and packet loss with raising number of rules. However, they do not measure jitter or latency, which we want to consider in this work.

Rovniagin and Wool [8] revisit a classic algorithm from computational geometry and integrate it within the filtering mechanism of iptables. The algorithm, called Geometric Efficient Matching (GEM), performs packet matching in $O(d \log n)$ time and requires $O(n^d)$ space in the worst case, where n is the number of rules and d the number of fields in the packet header to match. Their optimized GEM-iptables implementation sustains a packet matching rate of over 30,000 packets per second (pps), with 80-byte packets and 10,000 rules, without packet loss on a standard PC workstation. In comparison, the unmodified Linux iptables could only sustain a rate of around 2,500 pps. However, the space complexity of the algorithm is impractical for our implementation, as we aim to achieve packet matching in $O(1)$ time.

To the best of our knowledge, there is only one research work that proposes an integrated solution that enhances stateful packet filtering and the session table architecture. Trabelsi and Zeidan [9] propose a session table architecture that invokes the hash function only once per session to reduce memory space consumption and filtering time. According to them, storing all session state information in one table entry causes additional processing time, especially for session table timeout attributes. Therefore, they separate the session table entries (session states and timeout attributes) into two different data structures to enhance the session table lookup and processing time. Nonetheless, the authors do not consider the latency impact of the firewall caused by the use of stateless filtering. We intend to avoid the firewall from using the stateless filtering mechanism by using the session table.

III. Design

To date, software firewalls implementing stateful filtering are not suitable for constant-time packet filtering. This is because the first packet of a session passes through the stateless filtering mechanism. As a result, this behavior increases the latency and jitter of the firewall.

In the context of this work, we examine different approaches to modify and improve the stateful packet filtering mechanism. Hereby, we aim to reduce the jitter to gain a constant latency of processing packets with our software-based firewall implementation. The following section describes the approaches and difficulties to improve and implement stateful filtering efficiently in order to make our firewall implementation suitable for time-critical applications.

a) Explicit use of the session table: Stateless filtering cannot guarantee a constant latency, as the duration to process a packet depends on the number of rules that need to be matched. The use of the session table will help to overcome this problem. In iptables the firewall performs the stateless check despite a match in a previous stateful check. Hereby, the firewall does not take advantage of the session table regarding the constant-time behavior. In contrast, FD.io VPP solves this problem and skips the stateless check if there is a match in the session table. However, there is one exception in VPP: When the firewall sees a session for the first time, the first packet of that session passes through the stateless check because there is no existing entry in the session table. This means that incoming packets of a new session always need to pass through the stateless check. As a result, this behavior increases the latency and jitter of the firewall for some packets, which is not suitable for industrial applications.

b) Improving the session tracking mechanism: Modifying (i.e., deleting or adding) the firewall rules in VPP leads to the deletion of all session table entries. In addition, the firewall uses timeouts to terminate and delete old sessions in the session table automatically. Whenever the session table is empty or no existing entry matches, all initial packets need to pass through the stateless check. In order to avoid the first packet from passing through the stateless check, we insert known sessions, e.g., TSN streams, statically into the session table. This guarantees that the entries are always present in the session table. Furthermore, by implementing static entries we prevent session table entries from being deleted by timeouts. However, not all information about the static entries is known at the time when inserting the entries into the session table. For example, when a client connects to a server, the destination port is usually known beforehand while the source port is unknown because it is ephemeral. However, the session table requires that all 5-tuple fields are known. We want to solve this problem by ignoring the source port of certain known sessions within the session table. We will implement this by hashing individual parts of the 5-tuple, not the entire 5-tuple.

c) Optimizing the session table maintenance: Modifying the firewall ruleset in VPP leads to the deletion of all session table entries. As a result, following packets pass through the stateless mechanism again, increasing latency and jitter. We want to prevent all session table entries from being deleted by only deleting the entries that are no longer allowed after a rule modification. Thus, we keep the untouched session table entries in the session table. However, is not easy to implement this efficiently, which is probably the reason why it is not implemented presently in VPP. We already have a basic idea to implement the optimization from above, however, we will continue to optimize the idea during the thesis:

1) Simple approach (iterate over all session table entries and firewall rules): After each ruleset modification, the firewall must iterate over all rules for each session table entry and inspect whether the session table entry is still allowed in the updated ruleset. Session table entries that are no longer allowed must be deleted because we want to keep only session table entries that are allowed. This approach has one drawback: iterating over all entries and rules is feasible, but time consuming, as the firewall must check each entry against the rules. The slow update of the session table is especially a problem when changing several rules in the ruleset successively. While this is no problem with VPP, because it allows changing multiple rules in one command or API request, this can be an issue with other firewalls. Furthermore, we need to address the scenario for new sessions that are processed by the firewall during the iteration over the session table entries. To mitigate these issues, we can make this simple approach faster by optimizing it.

2) Optimization of the simple approach to reduce the processing time (store the reference of a firewall rule in the corresponding session table entries): To optimize the simple approach, we add a new field to the session table entry. The new field stores a reference to the firewall rule that allowed the first packet of the corresponding session to pass through. This eliminates the need to iterate over the entire ruleset after a ruleset change and allows the firewall to start the iteration from the referenced rule. For example, when a referenced rule is being deleted, it is no longer necessary to iterate over all firewall rules for each session table entry. Instead, the firewall starts to iterate from the deleted rule that is referenced. If the session table entry does not match a subsequent allow-rule the firewall deletes the session table entry.

IV. Evaluation

The evaluation of this work will consist of measurements of latency, jitter, packet loss, and overhead of our implementation. We will compare the time behavior of our firewall before and after modification. Based on the result, we will evaluate the implemented stateful mechanisms with respect to its efficiency, security and limitations.

To perform the mentioned measurements we will use a low-performance industrial firewall or a comparable low-performance computer with a x86 CPU. Our proof-of-concept firewall version will be installed and deployed on a Linux-based operating system. We will use a dedicated traffic generator to put specific load on the firewall. A managed switch will be used to conduct time stamping on the packets in order to measure the latency. To reach our goal, the proof-of-concept firewall must provide:

- Lower jitter than the original firewall implementation, ideally, the jitter will be below 100 µs
- Not a significant increase in latency (due to processing overhead)
- No security degradation compared to the original firewall

V. Result

The result of this work will be a proof-of-concept firewall implementation with corresponding performance measurements. The evaluation of the implementation will face whether the firewall can achieve low jitter with constant-time behavior while maintaining the level of security.

If the implementation works and shows the expected behavior, we will upstream the optimized source code to the FD.io Git repository. Ideally, the firewall can be installed and deployed in a production environment to increase the security of industrial networks.

References

[1] L. Wüsteney, M. Menth, R. Hummen, and T. Heer, "Impact of packet filtering on time-sensitive networking traffic," in *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*, 2021, pp. 59–66.

[2] M. Schramm, "Adaptation of the vpp firewall for real-time packet processing in industrial environments," Master Thesis, Eberhard Karls Universität Tübingen, 2022.

[3] "Fdio - the universal dataplane." [Online]. Available: https://fd.io/ ,[Accessed:16-12-2022].

[4] D. Zvabva, P. Zavarsky, S. Butakov, and J. Luswata, "Evaluation of industrial firewall performance issues in automation and control networks," in *2018 29th Biennial Symposium on Communications (BSC)*, 2018, pp. 1–5.

[5] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "A stateful mechanism for the tree-rule firewall," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, 2014, pp. 122–129.

[6] X. He, T. Chomsiri, P. Nanda, and Z. Tan, "Improving cloud network security using the tree-rule firewall," *Future Generation Computer Systems*, vol. 30, pp. 116–126, 2014. [Online]. Available: https://doi.org/10.1016/j.future.2013.06.024

[7] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "Hybrid tree-rule firewall for high speed data transmission," *IEEE Transactions on Cloud Computing*, vol. 8, no. 4, pp. 1237–1249, 2020.

[8] D. Rovniagin and A. Wool, "The geometric efficient matching algorithm for firewalls," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 1, pp. 147–159, 2011.

[9] Z. Trabelsi and S. Zeidan, "Enhanced session table architecture for stateful firewalls," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–7.