Secure Deep Packet Inspection in Outsourced Middleboxes

Naveen Mukundan Ravindran, Tejaswi Palle
EECS Department,
Wichita State University
{nxravindran, txpalle}@shockers.wichita.edu

Abstract

Modern enterprise networks require middleboxes to support a wide range of advanced traffic processing techniques. While middleboxes avoid local maintenance, they still face security issues. Many middlebox processing services such as intrusion detection systems, firewalls, etc. require inspection of packet payloads. However, with the adoption of HTTPS, the functionality is limited due to end-to-end encryption. Also, packet contents and inspection rules are proprietary in nature and the sensitive information of enterprise networks could be revealed because of untrusted cloud environment. In this project, we implement a practical existing system architecture for outsourced middleboxes to perform deep packet inspection over an encrypted traffic. This approach does not reveal the packet contents and the rules performed in the inspection. We modified the design of building the high-performance encryption filter at both admin server and cloud side. The admin server will partially build the filter and send all the data to cloud for completion of filter. The Client and Server generate private keys to randomize tokens for inspection. We built the filter to support real datasets during inspection. The security is preserved by having an encrypted connection between end points using SSL. Implementation at Virtual Machines show that this approach produces less latency in each connection initialization and increase in throughput.

1 Introduction

Middleboxes are dedicated hardware which provides specific network functions like an intrusion detection system, firewall, packet forwarding, and network address translation to an enterprise. However, maintaining middleboxes within the enterprise gives rise to expensive management costs [1]. This has resulted in many enterprises calling for moving the middlebox functions to public cloud in form of virtualized services [2]. This is to mainly reduce the management costs and provide additional benefits like scalability, fault tolerance and cost effectiveness. Despite the promising advantages, outsourcing the middleboxes to cloud also presents many unaddressed security challenges. Firstly, such outsourcing of middlebox can give the cloud service provider complete access to the traffic flows. Secondly, the packet inspection rules used by enterprises are proprietary in nature because they are customized by the enterprises to meet their requirements [3][4]. These may contain highly sensitive information like intellectual properties and trade secrets. Therefore, it is important to provide protection to these enterprise rulesets when deployed in the cloud [4][5].

In case of exposed traffic flow to cloud provider, HTTPS which provides an end-to-end confidentiality can limit the middlebox capabilities [6]. The existing design BlindBox [7] which provides deep packet inspection in HTTPS is still not suitable for middleboxes outsourced to the cloud. It not only fails to protect the rulesets in the middleboxes at cloud but is expensive. These shortcomings motivated us to implement a system which allows outsourcing of middleboxes to cloud. This provides strong security to packet payloads and proprietary rulesets of the enterprises. For traffic payloads, we parse the packet payloads and encrypt into randomized tokens. We then extract key-value pairs which are the suspicious strings and their respective actions. Using these string-action pairs, we implement an encrypted rule filter which will be used for indexing these encrypted pairs. By letting the encrypted tokens pass through the encrypted rule filter, we build a much safer and efficient DPI with the use of searchable symmetric encryption and broadcast encryption schemes [8]. However, some issues are encountered in this process. Firstly, the use of any generic primitives of searchable symmetric encryption may not necessarily satisfy all the requirements for our stated model.

For this, we implement the security framework of searchable symmetric encryption with a recent high-performance hash table design [9] resulting in and encrypted rule filter. Upon using this, only if a suspicious string is encountered and matched that the corresponding action will be recovered and provoked against exfiltration and intrusion. This process not only provide privacy to the packet payloads, but also for the semantic information of rules apart from the provoked actions. Secondly, the searchable symmetric encryption alone is not sufficient for complicated inspection rules which reveal their attributes, thereby compromising the confidentiality. Some rules are multi-conditional meaning all the conditions need to be matched to reveal the actions. For such situations, we implement a customized technique to handle different type of inspection rules while simultaneously providing

security to the rules and packet payloads. This is mainly done by performing secret sharing in case of multi-conditional matching where embedded action in encrypted filter design. In situations where equal incoming tokens are generated by same strings, we duplicate the string in multiple copies to hide them. These can be later matched by other different tokens. Our procedure is independent of connection initialization, enabling us to asynchronously pre-build multiple encrypted filters.

2 Literature Review

The first practical service for outsourcing enterprise middlebox processing to the cloud was APLOMB [1]. This protocol solved some real problems faced by network administrators and outsourced over 90% of middlebox hardware to the cloud. But the problem with this protocol is, it allowed the middlebox owner to inspect or read all traffic. The situation was preferable to the status queue in the inspection phase from a client point of view. To overcome this issue, a system known as BlindBox provided both the functionality of middle-boxes and the privacy of encryption. This protocol performs deep-packet inspection directly on the encrypted traffic [7].

BlindBox currently is not ready for practical deployment due to its expensive connection setup, involving a secure two-party computation protocol between each endpoint and the middlebox. Most of the middleboxes handle HTTPS by intercepting and decrypting the encrypted traffic. Since it reveals the packet payload at middle box, it may constitute a man-in-the-middle attack [1]. A secure middlebox was designed to outsource network function securely so that the sensitive information is protected from cloud provider [10]. Even though this provides protection against traffic as well as rules, the issue with this design is that it uses heavy cryptographic tool like homomorphic encryption [10]. The proposed design in our paper provides privacy against both encrypted traffic as well as packet inspection rules of middleboxes. However, the original paper does not consider the problem of secure rule update operation [11].

Later, a secure service function chain outsourcing framework was built which encrypts each packet header and uses a label for in-cloud rule matching enabling the cloud to accomplish its functionalities with minimum header information leakage [12]. A middlebox based system called SPABox was introduced which supported keyword-based and data analysis-based deep packet inspection function over encrypted traffic [13]. But this system does not support privacy-preserving for regular expression evaluation schemes. For providing execution assurance to the outsourced middleboxes, the first practical system was developed which focuses on pattern matching based network functions and perform a probabilistic checking mechanism for assurance [14]. A system called Embark was the first ever system which enabled cloud provider to support middlebox outsourcing along with maintaining client's confidentiality [15]. However, the main drawback of Embark is it does not support real-time updates.

3 System Architecture

The architecture that we implemented in this project includes four parts. They are Initialization, Pre-processing, Inspection and Verification. At initialization, two end points A and B after establishing a connection, register at Admin Server (AS) for requesting their respective keys K_s and K_r through an encryption channel. AS also builds an encrypted filter that indexes the string-action pairs extracted from the rules and uploads it to middlebox (MB). MB now performs packet inspection for this connection using encrypted filter. In pre-processing phase, one endpoint starts sending encrypted traffic while simultaneously parsing the packet payloads into a set of strings. Key K_s is used

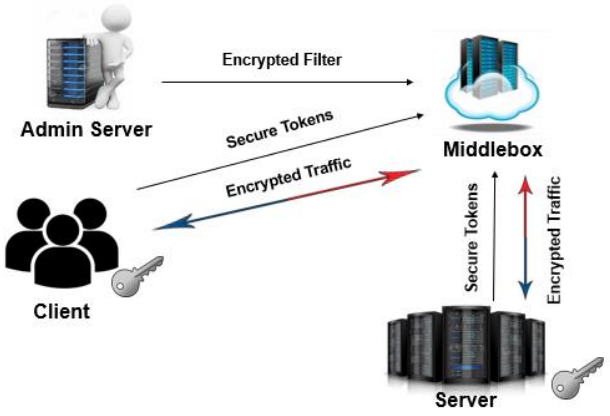

Fig 1 System Architecture

to convert into randomized tokens sent to MB. The other end parses the packet payload and uses its key K_r to generate tokens. In inspection phase, as the tokens and encrypted traffic arrive, MB executes the proposed secured DPI in the held traffic to process tokens over the encrypted filter in a streaming fashion. If a token correctly recovers an entry in encrypted of the filter, resulting action is taken. After all tokens in the packet are checked and no match is found, the packet is allowed to pass through.

4 Implementations

The encryption filter enables the middlebox to perform private and efficient deep packet inspection over encrypted traffic without revealing the packet contents and inspection rules. The high-performance encryption filter is built on the Admin Server (AS) in original paper. We modified the way of building the encryption filter partially at enterprise side and send all the data required to complete at the middlebox. The complete protocol is given below:

Detailed Protocol: A broadcast encryption scheme such as Fernet Cryptography is used for Encryption (Fernet. Enc), Decryption (Fernet. Dec), Adding Users (Fernet. Add) and Key Generation (1^k). A ruleset (Snort) R is used to match all the single tokens that are encrypted from traffic.

- AS generates three private keys K_1, K_2 and K_3 using KeyGen (1^k) in Fernet Cryptography where k is the security parameter. For our experiments, we have used

k=64 bytes.

- A high-performance encryption filter is built partially by Admin Server with parameters such as {t_1, t_2, K_2 and string-action pairs}.
- The end points Client (C) and Server (S) run the standard SSL Key exchange protocol with socket programming.
- AS computes the state information of client, middlebox and server by st ← Fernet. Enc (K_3, P, r) where P = {C, S, MB} and r represents a k-bit random string. In our system, r is 64-bit random string generated by random functions.
- AS generates KC ← Fernet. Add (K_3, C) and K_S ← Fernet. Add (K_3, S).
- {K_1, K_2, K_C, st, C} are sent to Client C. {K_1, K_2, K_C, st, S} are sent to Server S. {Partially built encrypted filter, r and state information st } are sent to MB.
- The client C has packet payloads from which the strings are tokenized as (t, s) in random. For each token, t=F_1 (K_1, str||c) where s= F_2 (K_2, str||c) and c is concatenated with each of the string to make all the strings distinct.
- Client decrypts r with K_C and st by Fernet. Dec (K_S, st) and σ ← G (r, t||s) where G is a pseudo random function keyed by r. The client sends σ to MB.
- MB decrypts σ by t || s ← G^{-1} (r, σ). The value of t and s calculated at client side is decrypted by MB. It generates t_1 and t_2 using pseudorandom functions. MB performs e ⊕ s foe every value in T_1 [t_1] and T_2 [t_2].
- This XOR operation results true if the encrypted value in hash table matches with the encrypted token. When a suspicious string is matched, the responding action act_i is recovered and an alert is sent to the other end point. If all the tokens are legitimate, the packet can pass to the other end point.
- S follows the same procedure while sending the encrypted traffic to MB for inspection.

Encryption Filter Algorithm:

Inputs: String-action pairs {(str_1, act_1), (str_2, act_2) …. (str_n, act_n)} extracted from Snort ruleset R. Private Keys: K_1, K_2. Pseudorandom Functions: F_1, F_2, P_1, P_2. d- number of entries in each bucket (2 entries in our paper), β – cuckoo threshold.

Admin Server:

- After generating string action pairs, we compute t = F_1 (K_1, str_i), t_1 = P_1 (t, 1), and t_2 = P_2 (t, 2).
- AS sends {t_1, t_2, K_2 and string-action pairs} to MB.

Middlebox:

- Two hash tables T_1 and T_2 are created with the capacity of 2000 entries in each table to hold the size of 3488 rules.
- With the received value of t_1 and t_2, the actions are inserted in either T_1 [t_1] or T_2 [t_2]. If the values already exist, cuckoo hashing is performed within β recursive trials.
- After all actions are inserted, each value is encrypted with act_i ⊕ s and reinserted into the hash tables where s represents F_2 (K_2, str_i).
- All the empty values in hash table are filled with random strings using random function.

5 Experimental Evaluation

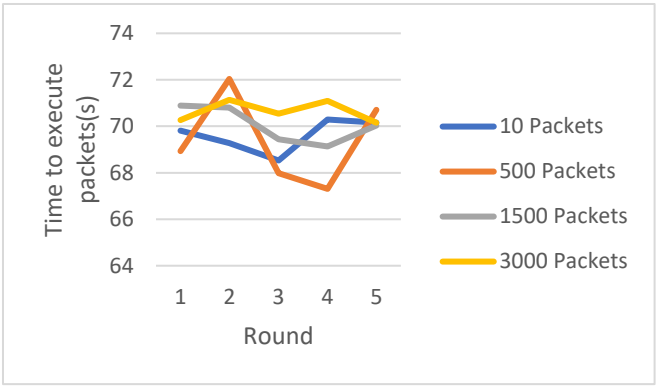

Fig 2 Time to execute different number of packets

For evaluation, we selected an open source ruleset known as Snort default ruleset and user defined packet payloads with the number of packets, such as 10,500,1500 and 3000 with all the rules. We used 3844 Snort rules for the purpose of evaluation. We implemented the middlebox module in Virtual Machine (Cloud) provided by EECS Department, Wichita State University. We installed Virtual Box to test our results with higher configuration. We used Python as programming language for all the three modules (client, server and middlebox). We installed endpoint modules on Lenovo Ideapad with Intel Core i7 CPU and 8GB RAM.

Snort

Implementation	Match Percentage
Without Encryption	100
With Encryption	99.9
Virtual Machine	99.8
Virtual Box	99.9

TABLE 1: Statistics on the percentage of matched suspicious strings across different methods and platforms.

We used different number of packets to evaluate with the set of Snort rules (3844 rules). The packet payloads are user defined and taken as examples from real encrypted traffic. We assumed to have all the packets as malicious to test the worst-case scenario to detect the accuracy of our design in such cases. As shown in the Table 1, when the number of packets is 10, the percentage of the match by our design tends to be 100% without encryption. The results

tend to give around 100% for with encryption and when executed at virtual machines and virtual box. We assumed certain number of packets as malicious and found that all the malicious packets are matched and identified by this design. This design will give the highest percentage of the match even for large packets.

System	Average Time (s)	Average Memory Usage (MB)
Local Machine	0.3169	48.028
Virtual Machine (Cloud)	0.4187	51.420

TABLE 2: Statistics on the average time and memory usage of both the systems.

Performance Evaluation: We evaluate the computational cost produced by our design in both local machines and cloud shown in Table 2. We executed the middlebox module along with endpoint module locally and the average time to execute 10 packets with 5 rules were only 0.3169 seconds. We moved the middlebox module to one of the virtual machines in the cloud and the average time to execute was 0.4187 seconds. The memory usage was around 48 MB and 51 MB in a local machine and cloud respectively. We could see that the average time and memory usage is less when executed locally. The reason is the endpoint module had a better system configuration than the virtual machine.

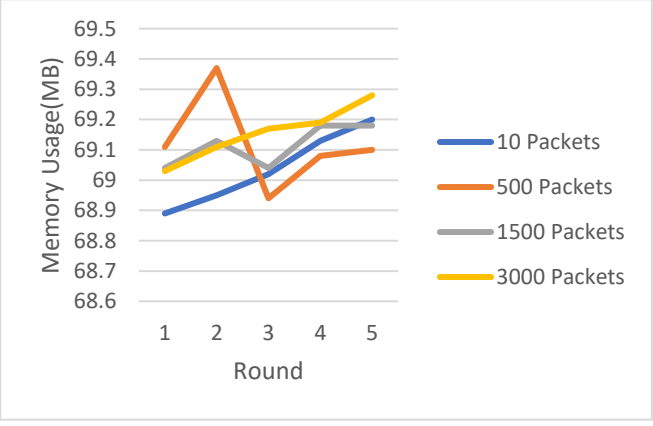

Fig 3 Memory Usage for different number of packets

In Figure 2, the time to execute a different number of packets is plotted. The Snort ruleset is considered for evaluation having different number of packets. MB executes the program with different number of packets with each of them executed 5 times. We can infer from the graph that the time to execute a different number of packets does not have a big variation in scale because once the filter is built with all the rules, the number of packets to check all the rules is minimal. The execution time will vary when the number of rules differs when building the high-performance encryption filter. All the packets will have almost equal time depending on the configuration of the middlebox. Similarly, the memory usage taken by the program to execute different number of packets are shown in Figure 3. As said earlier, the memory usage will vary depending on the rules that are used for building the encryption filter. The execution time and memory usage depend on the number of rules used for encryption filter.

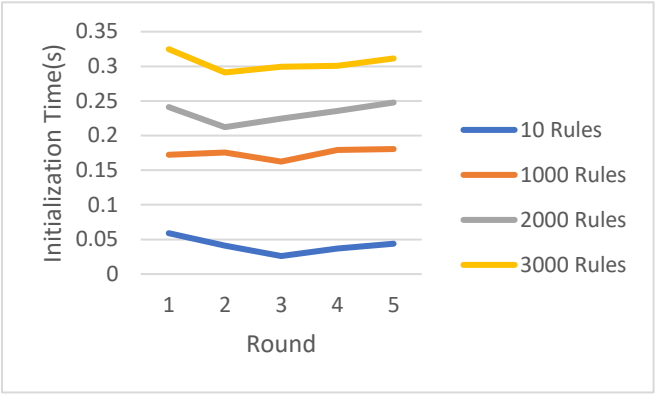

Fig 4 Initialization time for different number of rules

The initialization time taken for building encryption filter with different rules for each connection is shown in Figure 4. When the number of rules is less, the time to build the encryption filter will be less. The time to build the encryption filter will increase with the increase in the number of rules used. The memory usage also will depend on the number of rules used for building the encryption filter. We could analyze that for some number of rules between 10 and 1000, the time taken to build is more when compared to the time difference between 1000 and 3000.

6 Contributions

In the original paper, the admin server builds the encrypted filter at enterprise side and sends them to the middlebox. In this project, we modified this by partially building the encrypted filter at the enterprise's side and built the rest of the filter at the cloud in this project. Even after the modification, we preserve the privacy of both the end points by securely generating private keys at end point's side. This will ensure that the cloud will never know the private keys of both the end points at any time. The design will not reveal the packet contents at any time in the process.

References

[1] 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. In Proc. of ACM SIGCOMM, 2012

[2] H. Jamjoom, D. Williams, and U. Sharma, "Don't call them middleboxes, call them middlepipes," in Proc. of ACM HotSDN, 2014

[3] K. Skarfone and P. Mell, “Guide to intrusion detection and prevention systems,” National Institute of Standards and Technology, available at: http:// csrc.nist.gov/ publications/ nistpubs/800-94/SP800-94.pdf, 2007.

[4] J. Shi, Y. Zhang, and S. Zhong, “Privacy-preserving network functionality outsourcing,” arXiv preprint arXiv:1502.00389, 2015

[5] A. R. Khakpour and A. X. Liu, “First step toward cloud-based firewalling,” in Proc. of IEEE SRDS, 2012.

[6] Google, “HTTPS as a ranking signal,” http://googlewebmastercentral.blogspot.hk/2014/08/https-as-ranking-signal.html, 2014.

[7] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection for encrypted traffic. In Proc. of ACM SIGCOMM, 2015.

[8] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: Improved definitions and efficient constructions,” Journal of Computer Security, vol. 19, no. 5, pp. 895–934, 2011.

[9] B. Fan, D. Andersen, M. Kaminsky, and M. Mitzenmacher, “Cuckoo filter: Practically better than bloom,” in Proc. of ACM CoNEXT, 2014.

[10] L. Melis, H. J. Asghar, E. D. Cristofaro, and M. A. Kaafar, “Private processing of outsourced network functions: Feasibility and constructions,” Cryptology ePrint Archive, Report 2015/949, 2015.

[11] Guo, Y., Wang, C., Yuan, X. and Jia, X., 2018. Enabling privacy-preserving header matching for outsourced middleboxes. In *Proc. of IWQoS*.

[12] Wang, H., Li, X., Zhao, Y., Yu, Y., Yang, H. and Qian, C., 2016. SICS: Secure In-Cloud Service Function Chaining. *arXiv preprint arXiv:1606.07079*.

[13] Fan, J., Guan, C., Ren, K., Cui, Y. and Qiao, C., 2017. SPABox: safeguarding privacy during deep packet inspection at a MiddleBox. *IEEE/ACM Transactions on Networking*, *25*(6), pp.3753-3766.

[14] Yuan, X., Duan, H. and Wang, C., 2018. Assuring String Pattern Matching in Outsourced Middleboxes. *IEEE/ACM Transactions on Networking*.

[15] Lan, C., Sherry, J., Popa, R.A., Ratnasamy, S. and Liu, Z., 2016, March. Embark: Securely Outsourcing Middleboxes to the Cloud. In *NSDI* (Vol. 16, pp. 255-273).