FlowShredder: a Protocol-Independent in-Network Security Service in the Cloud

Bin Song[1], Bin Sun[1], Qiang Fu[2], and Hao Li[3] ✉

[1] Xi'an Jiaotong University
[2] RMIT University
[3] Xi'an Jiaotong University hao.li.@xjtu.edu.cn

Abstract. Cloud services increasingly generates enormous Internet traffic. Much of it such as rich media traffic is not highly sensitive, but prefers some sort of protection. The traditional end-to-end encryption such as TLS is costly and has issues such as increased latency, while the simple anonymity solutions cannot resist traffic analysis attacks. In this paper, we propose FlowShredder, a protocol-independent and in-network service to secure such traffic in the cloud. FlowShredder aims to break the association between packets, data flow and hosts by obfuscating the packet header (some payload if needed). Without the context of flow and hosts, packets are of little value to the adversary. The operation is carried out at cloud gateways, without encrypting the payload. Its simple logic can therefore be executed within a single pipeline of the Tofino programmable switch, to ensure wire-speed performance without the scalability issue. Being protocol-independent and operating in-network at wire speed make FlowShredder a practical and generic security service to protect the cloud traffic. In addition, FlowShredder can work with end-to-end encryption such as 0-RTT TLS for enhanced protection. We implement FlowShredder in P4 switches. Experiments show that FlowShredder can effectively resist the traffic analysis attack with supervised learning techniques.

1 Introduction

In the era of 5G, IoT and edge computing, enormous information is transferred in the cloud. Much traffic is with rich media, *e.g.*, live video, online gaming and online conference streaming. One major threat over these applications is traffic analysis attack. The adversary can sniff the traffic from a compromised switch, pick the packets of hosts, and reassemble them into a complete flow to obtain the private data, *e.g.*, the video clips and the voice recordings.

Mature techniques, may overkill or are ill-suited for these scenarios, such as TLS (Transport Layer Security) and traffic anonymity [2]. TLS strongly protect individual packets, but requires extra RTTs to establish session key causing increased latency, and consumes more resources at the end systems while these scenarios often have to establish many sessions.

0-RTT TLS was proposed to minimize the latency by reusing the established sessions and session keys, but the risk of leaking the key could make the entire flow exposed to the adversary. On the other hand, without content encryption, the adversary can reassemble the flow and obtain full content. Furthermore, these solutions are usually designed for specific protocol stacks, *e.g.*, TCP [4], and cannot be easily extended to other transport protocols such as QUIC.

For those that don't require strong protection but are sensitive to latency, it is not necessary to encrypt the payload. We argue that the flow is safe as long as its packets aren't distinguishable, because exposure of single packet is valueless if the adversary cannot reassemble flow. In short, our insight is that *without the context of flow and hosts that the packets belong to, these packets are of little value to the adversary.* In such cases, strong protection at the cost of latency as TLS does is not necessary. Even if the strong protection is needed, preventing the adversary from reassembling the flow can significantly enhance the protection. The goal is therefore to break the association between packets, flow, and hosts.

With this in mind, we propose FlowShredder, which only obfuscates IP addresses and L4 header of packets, without encrypting payloads. (In some cases, payload containing identifiable metadata has to be encrypted which is left to an end-to-end scheme such as 0-RTT TLS) This makes it possible to take advantage of programmable data plane such as P4 switches. As a result, FlowShredder is a *protocol-independent* and *in-network* approach operating at *wire speed*. Note that the wire speed is ensured due to the simplicity of obfuscation operations, which can be fitted into a single pipeline of the Tofino architecture. Therefore, there is no scalability issue compared to end-host based approach such as TLS.

FlowShredder does not aim to replace TLS or 0-RTT TLS, but provides an option for applications such as rich media, requireing no strong protection. In fact, FlowShredder can work with end-host based solutions for enhanced protection. Given its nature of being protocol-independent and operating in-network at wire speed, FlowShredder is a practical and generic security service for the cloud. Our contributions are as follows.

- We propose FlowShredder for per-packet indistinguishability (Section 3), which is achieved through packet header obfuscation using a per-packet random key and the lightweight `XOR` operation. This ensures the randomness of each packet, and the random key is only valid for a single packet. We take advantage of IPv6 to route the packet correctly.
- FlowShredder is evaluated with real and synthetic configurations (Section 4). The results suggest that FlowShredder is (1) effective against the traffic analysis attack even with supervised learning; (2) transparent to the end hosts and outperforms the existing in-network and end-to-end approaches.

2 Threat Model

Figure 1 shows a typical scenario of traffic leaking. The end hosts are connected to a trusted CSP gateway (yellow switches) through a trusted CSP network (yellow cloud). The trusted networks are connected with the untrusted networks (gray cloud). All devices not in the trusted CSP networks are prone to the attacks.

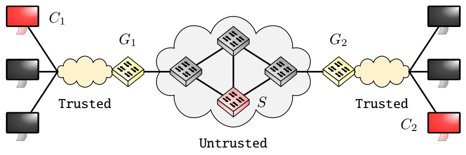

Fig. 1: Threat model. The two trusted CSP networks are connected with a large untrusted network (gray cloud and switches). Adversaries can compromise the end hosts and untrusted switches to sniff the traffic (red hosts and switches).

Adversaries can compromise some end hosts, *e.g.*, C_1 and C_2, along with the untrusted switch S, such that they can launch a chosen-plaintext attack by comparing the information sent from C_1 to C_2 with the information sniffed in S to identify obfuscation scheme. This scenario is common for public clouds, where any customers (including the adversaries) can buy virtual machines behind the trusted CSP network. The adversaries can then find a way to reassemble the data flow and discover useful information.

3 Design of FlowShredder

An intuitive approach to realizing per-packet indistinguishability is to obfuscate identifiers of connection, *e.g.*, IP addresses and TCP sequence numbers. This obfuscation is different from anonymity approach because FlowShredder breaks association not only between connections and hosts, but also between packets and connection, making *every packet* indistinguishable, achieving per-packet indistinguishability. This prevents adversary from reassembling connection with sniffed packets. However, header obfuscation messes up destination IP, leading to incorrect routing. We need to find a way to route packets correctly. The procedure to construct a packet in FlowShredder is shown in Fig. 2.

3.1 Efficient Obfuscation

As shown in Fig. 1, FlowShredder can be deployed in the gateway switch, *i.e.*, G_1 and G_2. It uses a per-packet random seed, *i.e.*, an obfuscation key, to `XOR` the header fields. In doing so, each packet is obfuscated in a different way using the lightweight `XOR` operation, and the random key is only valid for a particular packet. The leak of the key only affects a single packet. However, there are some issues. First, the obfuscation key used by G_1 must be sent along with the packets, otherwise G_2 cannot recover the packets from the obfuscation. This means that the key must be strongly encrypted, *e.g.*, with AES, which may consume significant resources [3]. Second, even if we can assume that all packets are with IP protocol, we still have to obfuscate many other fields to support the L4 protocol independence. For example, a TCP packet requires to obfuscate the TCP ports and sequence numbers, while a QUIC packet has to obfuscate the

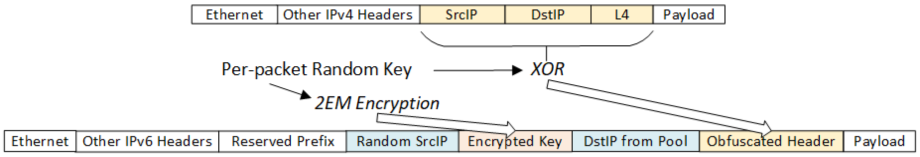

Fig. 2: The encryption of headers and the construction of packets in FlowShredder. The src and dst IP and L4 headers are obfuscated using a random per-packet key encrypted by 2EM algorithm with a rotated key set.

stream and connection IDs. There are also combined stacks like IP-in-IP that requires complex obfuscation. As a result, the bytes to be obfuscated could be too long to be fitted into a single pipeline of the P4 switch. FlowShredder addresses these challenges as follows. First, it uses a 64-bit obfuscation key to `XOR` the IP addresses and *all headers hereafter*. This ensures that all L4 protocols can be supported by FlowShredder, and `XOR` is a lightweight operation. To secure the obfuscation key, instead of AES, FlowShredder uses two-round Even-Mansour (2EM) scheme [1] to encrypt the obfuscation key for its simplicity and tight security proofs. The cipher encrypts a n-bit text M by computing:

$$E(M) = P_2(P_1(M \oplus k_0) \oplus k_1) \oplus k_2 \tag{1}$$

where k_0, k_1 and k_2 are independent encryption keys and P_0, P_1 and P_2 are independent permutations [9].

Theoretically, 2EM is secure against about $2^{\frac{2n}{3}}$ queries with chosen-plaintext attacks, *i.e.*, about 2.6M million queries for 32bit message encryption [9]. As we encrypt a 64-bit message, the number of queries will be up to 7 trillions, which should be sufficient to resist against such attacks. Besides, FlowShredder rotates k_0–k_2 from time to time, making it even harder for an adversary to break. The source and destination IP and L4 headers are obfuscated using a per-packet random key, which is encrypted by 2EM algorithm with a rotated key set. There is no need to share the random key between the trusted gateways, because the gateway can recover the random key via the 2EM key set.

3.2 Facilitating Routing Procedure

The above random obfuscation may break the end-to-end routing procedure for two reasons. First, the obfuscated destination IPs are unknown to the routers in untrusted networks, so the packets cannot be correctly routed. Second, since the encrypted obfuscation key is sent along with the packet as a header field, we need to ensure such information will not impact the correct routing.

FlowShredder works natively with IPv6, but needs to transform IPv4 packets into IPv6 format in the entrance gateway, and recover them back to IPv4 in the exit switch. The IPv6 format gives the flexibility that FlowShredder can encode the encrypted key into the 128-bit address fields. Specifically, FlowShredder constructs a new IPv6 packet, with a random source address. Its destination address is split into two parts: the reserved IPv6 prefix to store the encrypted obfuscated key (64b), and a destination address randomly selected from an address pool (32b). FlowShredder announces the prefixes of the address pool to the routers in the untrusted networks. The addresses in the pool are randomly generated, and rotated in a way to avoid the frequent route convergence in the untrusted network. The L4 headers and payload remain unchanged.

In doing so, the intermediate network can correctly route the packets from G_1 to G_2, since the destination address from the pool points to G_2. G_2 recovers the IPv4 packets by reversing the operations.

4 Evaluation

In this section, we evaluate the effectiveness and efficiency of FlwoShredder. We deploy FlowShredder in tofino P4 switches (Wedge 100BF-32X) along with the local end hosts. We build two types of testbeds, as shown below.

Traffic generator (Gen) uses a traffic generator to generate the traffic to be protected. The traffic will traverse the entrance and exit gateway sequentially to the end host. The generator and the receiver are equipped with 100Gbps NIC, in order to drain the link and test the upper bound of the throughput.

Local testbed (Local) sets a pair of end hosts connecting to the gateway switches. We use Nginx to build a simple HTTPs server, such that this testbed can test whether FlowShredder performs well on the real connections such as HTTPs sessions. The bandwidth between the client and the server is 10Gbps.

4.1 Against Traffic Analysis Attack

Traffic analysis attack (TAA) with neural networks. We leverage the classic convolution neural network (CNN) and ResNet, to emulate the chosen-plaintext attack. We assume that the adversary knows the file format of the target connection, and is able to rent the servers in the trusted network. The adversaries can then generate their own connections, ideally along with the target connections, and train the CNN and ResNet model with supervised learning.

To mimic this process, we prepare 100 different files of the same format (`.mp4`) with various sizes. We use Gen testbed, and transfer the first 90 files. Then, we train the CNN and ResNet model using the traffic captured in the middle. Finally, we use the last 10 files and the trained CNN and ResNet model to detect these target connections. We test the classification accuracy with two scenarios. In Scenario 1, the test set is not part of the training set, that is, there are no common files between the two. In Scenario 2, the test set is part of the training set, giving the adversary the advantage of knowing some properties of the target. The input of the models is the packet header with some payload. As the input length grows, more of the payload is added to the input. This is to see how the payload affects the detection accuracy.

Per-Packet Indistinguishability against TAA. Figure 3 shows the testing accuracy of the raw traffic and FlowShredder in Scenario 1, when the test set

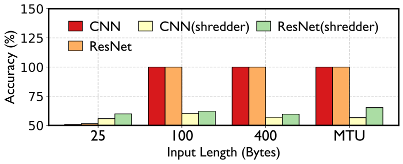

Fig. 3: CNN and ResNet can achieve a high accuracy on original packets, but fail on the traffic protected by FlowShredder, as the features to distinguish the connections are obfuscated.

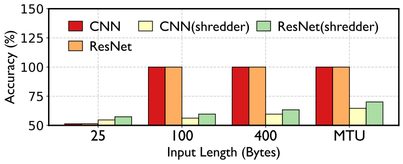

Fig. 4: If trained over the target traffic, ResNet can improve its accuracy on FlowShredder. However, it is still challenging to reassemble the connection even in this simplistic setting.

is not part of the training set. The share of the background and target traffic is 1:1. Thus, the accuracy of a random guess is 50%. Both CNN and ResNet have a high accuracy (nearly ∼100%), when dealing with the raw traffic, indicating that the traffic analysis with supervised learning is quite effective. The only exception is when the training input length is 25B, as it only covers the first two header fields in Fig. 2, which have no valid information for detection. The accuracy is essentially a random guess, ∼50%. However, for FlowShredder, the accuracy is slightly better than a random guess. This indicates that the knowledge learned from the 25B sample traffic is not quite meaningful for the testing traffic, as the headers of all packets are completely random.

Figure 4 shows the testing accuracy in Scenario 2, when the test set is part of the training set. ResNet, performs much better in this scenario with the highest accuracy of ∼70%. We reckon that this is related to the size of the training set and the diversity of the traffic. As the training set is relatively small and the test set is already a significant part of it, this makes it relatively comfortable to learn and recognise the target traffic. Nevertheless, even with this accuracy in such a simplistic setting, it is difficult to reassemble the whole connection.

4.2 In-Network Capabality

Comparison with P4-AES. P4-AES [3] is an in-network encryption approach that leverages Tofino chips to encrypt the packet payload, and can be used to offload the end hosts for encryption. We compare FlowShredder with P4-AES using Gen testbed, where the generator sends fix-sized UDP packets to the receiver. The P4 switches encrypt or decrypt every packet. For FlowShredder, S_1 obfuscates the headers. For P4-AES, S_1 encrypts the fixed-sized payload. S_2 performs the reverse operations accordingly. Fig. 5 measures the throughput of FlowShredder and P4-AES by transferring UDP packets with 16- and 32-byte payload. It can be seen that P4-AES suffers from a significant penalty as the packet size or the input rate increases. For example, it can only achieve 3.3Mpps for 16-byte payload and 1.45Mpps for 32-byte payload, at the input rate of 30Mpps. The root cause is that P4-AES has to recirculate the packets for many rounds to encrypt the payload, *i.e.*, 10 rounds for 16-byte encryption and 20

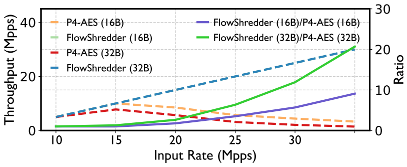

Fig. 5: Comparison with P4-AES on different packet sizes. FlowShredder's performance is irrelevant to packet size.

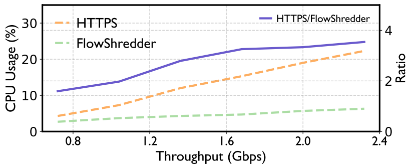

Fig. 6: Comparsion with HTTPS on CPU usage at server side. HTTPS consumes a lot more CPU cycles.

rounds for 32-byte. Larger packets lead to more recirculation rounds, and higher input rate increases the possibility to drop the packet in the recirculation. In the worst case, the packet is eventually dropped before finishing the recirculation, and the throughput may drop to zero. In contrast, the performance of FlowShredder is irrelevant to the packet size, which always catches up with the input rate. The reason is that FlowShredder only obfuscates the packet header.

Comparison with TLS. HTTPS/TLS may consume a large number of CPU cycles at the end hosts. To reveal the benefit of the in-network approach, we compare FlowShredder with HTTPS/TLS using Local testbed, where the client uses 8 processes to retrieve the 1MB file from the server at various rates. We measure the average CPU usage of the server during the transmission. Fig. 6 shows that, compared to FlowShredder when the throughput is 1.6Gbps, HTTPS/TLS consumes more than 3× of CPU cycles, and the factor is still growing as the throughput increases. This is largely because the server is busy encrypting and decrypting the payload with its CPU.

5 Related Work

Anonymity. Anonymity systems, such as LAP [6], Dovetail, HORNET and TARANET, obfuscate the IP addresses to break the association between the connection and the hosts. However, the association between the packets and the connection can still be discovered. In addition, these systems usually require host intervention. Some systems are based on the programmable data plane, *e.g.*, ONTAS, PANEL, and MIMIQ [5]. These efforts suffer the same problem of traditional anonymity approaches.

Encryption with programmable data plane. P4-AES [3] is a P4 version of TLS, which inherits its limit of requiring powerful computation capability. Moreover, the AES encryption in P4 cannot complete within a single pipeline, even for a 16B message. As a result, P4-AES cannot achieve wire-speed encryption. PINOT [9] is a DNS-privacy approach that obfuscates the IP addresses of the DNS requests. However, it does not aim to protect the content. SPINE [4] aims to protect IP addresses but avoid the high overhead of IPSec. It obfuscates the IP address and TCP sequence number for each packet. SPINE does break the association between the flow and the hosts, but relies on TLS for payload encryption. Some studies leverage the programmable data plane to obfuscate the topology [8], or adjust end-to-end encryption schemes [7]. These studies are orthogonal and complementary to FlowShredder.

6 Conclusion

We introduced FlowShredder, a protocol-independent, in-network security service to protect cloud traffic by separating packets from connection context. This achieves per-packet indistinguishability through header obfuscation. FlowShredder uses IPv6 with randomized IP addresses for accurate routing, ideal for latency-sensitive applications like rich media and real-time apps. For stronger protection, it can integrate with end-to-end encryption schemes like 0-RTT TLS to enhance security and reduce latency.

Acknowledgement. This paper is supported by the National Key Research and Development Program of China (2022YFB2901403) and NSFC (62172323).

References

1. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.X., Steinberger, J., Tischhauser, E.: Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations. In: International Conference on the Theory and Applications of Cryptographic Techniques (2012)
2. Bromberg, Y.D., Dufour, Q., Frey, D., Rivière, É.: Donar: Anonymous VoIP over tor. In: USENIX NSDI (2022)
3. Chen, X.: Implementing aes encryption on programmable switches via scrambled lookup tables. In: ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure (2020)
4. Datta, T., Feamster, N., Rexford, J., Wang, L.: SPINE: Surveillance protection in the network elements. In: USENIX Workshop on Free and Open Communications on the Internet (2019)
5. Govil, Y., Wang, L., Rexford, J.: {MIMIQ}: Masking {IPs} with migration in {QUIC}. In: USENIX Workshop on Free and Open Communications on the Internet (2020)
6. Hsiao, H.C., Kim, T.H.J., Perrig, A., Yamada, A., Nelson, S.C., Gruteser, M., Meng, W.: Lap: Lightweight anonymity and privacy. In: IEEE Symposium on Security and Privacy (2012)
7. Liu, G., Quan, W., Cheng, N., Lu, N., Zhang, H., Shen, X.: P4nis: Improving network immunity against eavesdropping with programmable data planes. In: IEEE INFOCOM Workshops (2020)
8. Meier, R., Tsankov, P., Lenders, V., Vanbever, L., Vechev, M.: {NetHide}: Secure and practical network topology obfuscation. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 693–709 (2018)
9. Wang, L., Kim, H., Mittal, P., Rexford, J.: Programmable in-network obfuscation of dns traffic. In: NDSS Workshop on DNS Privacy (2021)