FTGuard: A Priority-Aware Strategy Against the Flow Table Overflow Attack in SDN*

Menghao Zhang, Jun Bi, Jiasong Bai, Zhao Dong, Yongbin Li, Zhaogeng Li
Institute for Network Sciences and Cyberspace, Tsinghua University
Department of Computer Science, Tsinghua University
Tsinghua National Laboratory for Information Science and Technology (TNList)

CCS CONCEPTS

• **Security and privacy** → *Network security*; • **Networks** → *Network architectures*;

KEYWORDS

SDN; Flow Table Overflow Attack; Eviction; Priority

ACM Reference format:
Menghao Zhang, Jun Bi, Jiasong Bai, Zhao Dong, Yongbin Li, Zhaogeng Li. 2017. FTGuard: A Priority-Aware Strategy Against the Flow Table Overflow Attack in SDN. In *Proceedings of SIGCOMM Posters and Demos '17, Los Angeles, CA, USA, August 22–24, 2017,* 3 pages.
DOI:10.1145/3123878.3132015

1 PROBLEM STATEMENT

Software-Defined Networking (SDN) allows applications to install *dynamic* and *fine-grained* flow rules in the switches. In modern hardware switches, these rules are stored in Ternary Content Addressable Memory (TCAM) to achieve wire-speed packet processing. However, due to the manufacturing cost and power consumption of TCAM, most commodity switches are equipped with relatively limited flow table space, ranging from hundreds to a few thousands. This is usually insufficient in many SDN circumstances [2].

When the flow table is full, existing SDN switches adopt an eviction mechanism and automatically eliminate some entries to make space for newer flow rules. It is implemented as the Least Recently Used (LRU) eviction scheme in most of the popular switch platforms (e.g. OpenvSwitch, Pica8, Cisco Nexus). Even so, when the number of active flows exceeds the maximum number of entries in the flow table, some active entries currently used by active flows have to be kicked out to accommodate new flows. Meanwhile, these active flows become new flows and new *flow-mod* messages have to be parsed and installed again.

This eviction mechanism could be exploited by an attacker to commit the *flow table overflow attack*. With the information (idle-timeout, hard-timeout, and even flow table capacity, flow table usage) inferred by sending probing packets [3], the attacker is able to commit this overflow attack at lower cost and risk. As the flow table is crowded under normal conditions, with only a *suitable* number of fake packets, numerous new flow rules are tricked from the controller and frequent replacements would happen in the flow table. The flow entries used by benign users are inclined to be replaced by aggressive useless fake flows from the attacker, which would result in more *table-misses* from benign users and seriously degrade the performance of the entire network system.

2 COUNTERMEASURE ANALYSIS

As all the new flows compete for the limited flow table resources in the data plane successively, if no differentiation is taken, the demand of the subsequent flow will inevitably evict the flow entries of the previous one. Especially when the increasing successive demands come from malicious users, the entire flow table is used up and the other users may suffer from flow entry starvation. A strawman solution that distinguishing suspected traffic from benign traffic and simply dropping it at the controller seems feasible, however, the judgment for suspected or not may be unreliable such that false positive or false negative is possible. As a result, some benign traffic is inevitably harmed while some attack traffic can easily pass.

To address the above problem, we propose FTGuard, a **behavior-based priority-aware** defense strategy, to cope with this overflow attack. Our basic idea is to distinguish different flows with different priorities to achieve soft-isolation for users[1] of different evaluations. The evaluation is formalized as a score, which is favorable to the benign traffic features while adverse to the malicious ones. Based on the evaluation scores, we assign different priorities to flows of different users *dynamically*. Flows of benign users are inclined to have higher priorities while those of suspected users are more likely to have lower priorities. When the flow table is full, flow entries with the lowest priority are preferred to be evicted to make room for new incoming flows. Experiments and evaluations demonstrate that FTGuard provides cost-efficient protection for the flow table and is able to mitigate the overflow attack effectively.

3 FTGUARD DESIGN

The architecture of FTGuard is shown in Figure 1. As discussed above, the idea of FTGuard seems simple conceptually. However, we encounter three major challenges in the design of FTGuard. 1) how to collect traffic features and what features to collect. 2) how to develop the evaluation criterion to distinguish flows. 3) how to implement the strategy in the SDN switch without changing its

*Supported by National Key R&D Program of China (2017YFB0801701) and the National Science Foundation of China (No.61472213). Jun Bi is the corresponding author.

SIGCOMM Posters and Demos '17, August 22–24, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5057-0/17/08…$15.00
DOI:10.1145/3123878.3132015

[1]Benign users are inclined to be associated with only one source address, and an attacker may forge a set of source addresses to commit the attack. However, the set of forged source addresses could be regarded as numerous attackers. Therefore, in this paper, we use *source address* and *user* interchangeably.

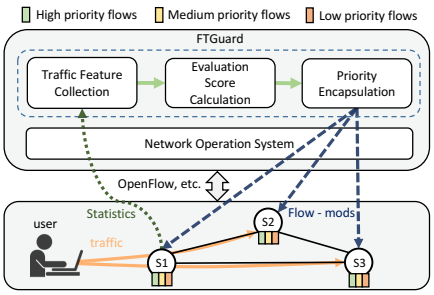
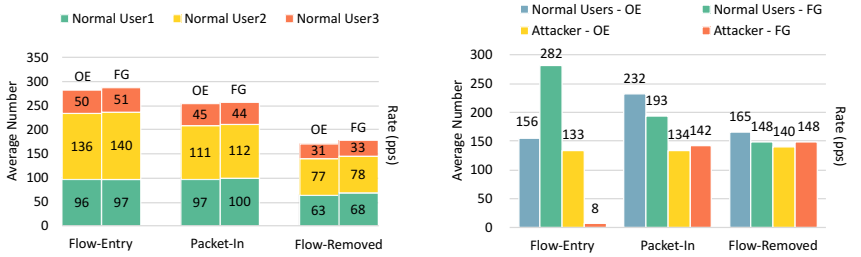


Figure 1: Architecture Overview



(a) The resource usage between an attacker and normal users under attack scenarios

(b) The resource usage between an attacker and normal users under attack scenarios

Figure 2: Evaluation Results

hardware. As shown in Figure 1, three modules are designed to deal with the corresponding challenges.

As for the first challenge, we design the *Traffic Feature Collection* module and collect traffic features from the basic service provided by the network operation system. Two fundamental differences between the normal traffic and the attack traffic are the frequency of new flows and the packet number per flow [1]. In order to reduce the attack cost, the attacker tends to commit the attack with *short-flows*, since this way he could cause a stronger attack effect with fewer packets. As each new flow would trigger a *packet-in* message to the controller, passive monitoring, classifying, and counting on the controller are sufficient to obtain the frequency of new flows. However, the number of packets per flow is not directly visible to the controller. *Statistic messages* have to be issued to the switch periodically (every T seconds) to allow the controller to query the *snapshot* information (the *counter* field of the flow entries) it needs.

For the second challenge, the *Evaluation Score Calculation* module is introduced and an evaluation criterion is developed to distinguish the behavior of different users. Considering the past period (T seconds), we denote the number of packet-in messages as pi_i, the ratio of good flow entries pulled from the switch as cr_i, where $cr_i = \frac{|\text{flow entries}|(\text{counter}>t \wedge \text{source address}==given)}{|\text{flow entries}|(\text{source address}==given)}$, t is a constant parameter used to distinguish good flow entries from bad ones (recommended to be set as 1 or 2). We denote the score associated with pi_i and cr_i as w_{in}. Obviously, w_{in} has a negative correlation with pi_i and a positive correlation with cr_i, since the attacker is inclined to attack with short frequent new flows. We simply use two linear correlations and add them up in this initial design. Considering historic information, the evaluation score w_i at this point obeys an exponential weighted moving average (EWMA): $w_i = (1-\alpha)w_i + \alpha w_{in}$, where α is a value between 0 and 1.

We address the third challenge with a probability selection algorithm in the *Priority Encapsulation* module. To map the score w_i to the priority (high, medium and low), we divide the region of the score into three parts with two *thresholds* (th_h, th_l), depending on the network policy. A Flow whose source address's score is higher than th_h is definite to be assigned to a high priority (obviously good), while a flow with a score between th_l and th_h is assigned to a medium priority with a large probability or a high priority with a small probability (ambiguously good). In order to solve the occasional case that a benign user may be evaluated with a score lower than th_l and thus suffers from starvation ever since, his flows are assigned to a high priority or a medium priority with a small probability, reserving some flow entries for him to be dug up to a higher score. With the priority obtained, we encode it into the *Importance* field of flow-mod messages, which is added in OpenFlow 1.4 as optional. The eviction process would be performed strictly in order of importance, that is, flow entries with lower importance will always be evicted before flow entries with higher importance.

4 EVALUATION

We implement FTGuard on Floodlight and OpenvSwitch, and perform the experiment in Mininet. For more details, please refer to our source code at Github[2]. Our benign traffic is collected from the Tsinghua campus network. We set the sampling period T as 3, the parameter α in EWMA as 0.5, and use a simple topology with only one switch connecting 3 hosts. In Figure 2, *OE* denotes the original strategy and *FG* refers to our priority-aware strategy.

In the no-attack scenarios, 3 hosts communicate with each other with benign traffic. As shown in Figure 2(a), flow entries of 3 hosts coexist with each other peacefully as nothing happens, which shows our strategy works with no extra effect on benign traffic when no attack happens. In the attack scenarios, 1 host becomes the attacker and sends deliberately forged packets with the same packet rate. As shown in Figure 2(b), without our strategy, the attacker occupies much more flow entries (133/8), causes more *packet-in* (232/193) and *flow-removed* (165/148) messages per second from benign users. All these results demonstrate that our strategy is able to mitigate the overflow attack effectively.

5 DISCUSSION AND FUTURE WORK

(1) **An attacker forges another user's source address to pollute the evaluation score of other users**: Method like Source Address Validation is easy to deploy in SDN and require no extra cost [4], which is able to solve this forgery problem more thoroughly. (2) **An attacker imitates the traffic feature of benign users**: The evaluation criterion and the selected features are unknowable to the attacker. Even if he could obtain these information, the cost for this overflow attack is multiplied, for he has to send multiple packets to cause the same attack effect. (3) **controller v.s. control channel v.s. switch**: Different SDN circumstances face different bottlenecks, this paper focuses on the overflow attack in the switch while the other two aspects are left to other works.

As our future work, we will consider more traffic features, develop more accurate evaluation criterion, give more suggestions on the parameters in our strategy, do more experiments to evaluate the burden on the controller, and consider more complex states.

[2]https://github.com/ZhangMenghao/FG/tree/baijiasong

REFERENCES

[1] S. Gao, Z. Peng, B. Xiao, A. Hu, and K. Ren. 2017. FloodDefender: Protecting Data and Control Plane Resources under SDN-aimed DoS Attacks. In *INFOCOM*. IEEE.
[2] N. Katta, O. Alipourfard, J. Rexford, and D. Walker. 2016. Cacheflow: Dependency-aware rule-caching for software-defined networks. In *SOSR*. ACM.
[3] J. Leng, Y. Zhou, J. Zhang, and C. Hu. 2015. An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flow table overflow in software-defined network. *arXiv preprint arXiv:1504.03095* (2015).
[4] B. Liu, J. Bi, and Y. Zhou. 2016. Source Address Validation in Software Defined Networks. In *SIGCOMM*. ACM.