2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops

Software-Defined HoneyNet: Towards Mitigating Link Flooding Attacks

Jinwoo Kim
School of Electrical Engineering, KAIST

Seungwon Shin
School of Electrical Engineering, KAIST

I. Introduction

Link Flooding Attacks (LFA) are *indistinguishable* and *undetectable* DDoS attacks, which exhaust bandwidth of bottleneck links. While traditional DDoS attacks flood end nodes, LFA aims at intermediate links. Then, it makes an extensive area entirely disconnected from the outside of networks[2]. In a LFA scenario, an attacker reconnoiters network topology via *traceroute* to locate layer-3 links, at which many flows converge. The attacker instructs bots to send low-rate traffic that is similar to benign traffic to the bottleneck links, while the attacker remains legitimate sessions with publicly accessible servers. Therefore, traditional countermeasures such as detecting spoofed IP addresses or specific signatures is inoperative.

There have been several works to detect and mitigate the LFA in both the traditional networks and SDN networks[4], [1]. However, we argue that the previous solutions are *ex-post countermeasures*. Thus, the proposed systems react after the LFA actually occur. The fundamental reason of enabling the attacker to conduct LFA is that he can build a *link map*, which contains a sequence of IP addresses for intermediate routers or end hosts via *traceroute*. With the link map, the attacker always has opportunities to launch attacks or to change target links once he locates vulnerable target link sets.

We argue that hiding real topology from an attacker's sight is a more radical solution than the previous works. However, the solution needs to be practical for detecting the real attacker; hence, simply blocking *traceroute* is not suitable. Here, we are inspired by the traditional honeypots that imitate decoy servers or systems to lure intruders. How about making *honey topology*? To achieve this, we first need to know which links can be targeted by an attacker to deploy the honey topology on a reasonable location. In the traditional network, a network operator needs to rely on out-of-band tools for discovering entire network topology. Also, the operator needs to compute link bandwidth with end-to-end measurement tests since residual link bandwidth is a major factor whether a link can be easily flooded or not.

For this reason, we leverage one of Software-defined Networking (SDN)'s benefits — Global Network Visibility. With SDN, we have in-band topology discovery for the entire network, and a capability for monitoring link statistics in real-time via OpenFlow (OF), which is a de facto standard protocol. Further, to precisely locate potentially bottleneck links, we consider both *static* and *dynamic* attributes of the bottleneck links. The static attribute is a rarely mutable property (e.g, routing paths), while the dynamic attribute is fast changeable one (e.g, utilized bandwidth).

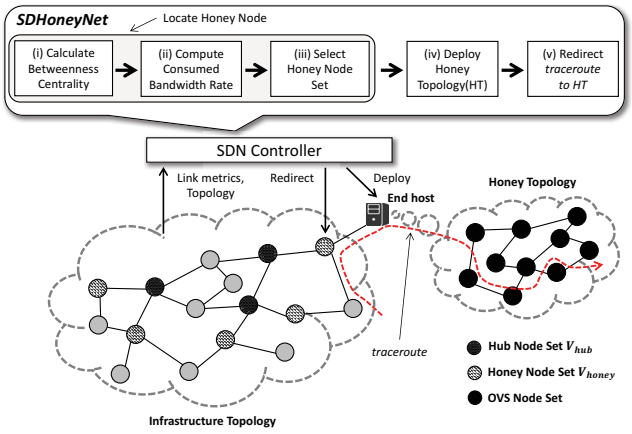

Fig. 1: Overall workflow of *SDHoneyNet*

In this paper, we present *SDHoneyNet*, a SDN based system which exposes fake topology to attackers. SDHoneyNet finds potential bottleneck links by computing the static metric, which is *Betweenness Centrality (BC)* of the network topology given by the SDN controller. It also calculates the dynamic metric, which is *Consumed Bandwidth Rate (CBR)* of each link by collecting OF's port statistics. It finds *a minimum intersection set* containing the nodes that have both high static and dynamic metrics. Then, SDHoneyNet deploys the *honey topology* where the node degree follows a power-law distribution, to mimic complex networks. We leverage software switches to easily build and deploy the honey topology. We implement the prototype of SDHoneyNet as an SDN application (2,000 lines of Java code) on the ONOS controller.

II. Proposed Method

We first define our terms, *Hub Node* and *Honey Node*. The hub node is a vertex which has both a high BC value and a high CBR value. The honey node is a vertex that is a neighbor of the hub node, and has lower BC than the hub node to hide vulnerable links from reconnaissance of an attacker. After finding the honey node set, we deploy the honey topology into end hosts of the honey nodes. Our proposed scheme is illustrated in Figure 1. The workflow consists of three logical steps.

A. Locating Honey Nodes

1) Calculating Betweenness Centrality: Our first step is to find out a hub node set V_{hub} which has possible target links

978-0-7695-6168-4/17 $31.00 © 2017 IEEE
DOI 10.1109/DSN-W.2017.10

in the network topology. We consider that AS-level routing path is determined by the policy-based routing. Thus, the BC of vertex v is computed as $C_B(v) = \sum_{s \neq t \neq v \in V} \frac{path_{st}(v)}{path_{st}}$[3], where $path_{st}(v)$ denotes the route determined by an SDN application. Here, we leverage Brandes's algorithm in order to reduce the time complexity of naively computing all routing path pairs.

2) Computing Consumed Bandwidth Rate: We leverage OF *Port Statistics* to calculate the utilization rate of a link. We regard TX bytes of a source vertex s as accumulative transmission bytes for a link $e(s,t)$. We collect the TX bytes for all ports of each switch for every *Interval*, which is a constant period that an SDN controller receives the port statistics from all switches. We denote *PreviousTXBytes(s)* that is a value for TX bytes of a vertex s collected from a previous request, and *CurrentTXBytes(s)* that is a value for TX bytes of a vertex s collected from a current request. We convert the bytes into bits per second, and then it is divided by *MaxBandwidth(e(s,t))*. As a formula, *CBR* of a link $e(s,t)$ is expressed as follows. $CBR(e(s,t)) = \frac{\frac{PreviousTXBytes(s) - CurrentTXBytes(s)}{Interval}}{MaxBandwidth(e(s,t))} \times 1.25 \times 10^9$

3) Selecting Honey Node Set: We now translate $CBR(e(s,t))$ into an *Aggregated Average CBR (AACBR)* for a destination vertex t of $e(s,t)$. Here, our intuition is that high the AACBR value destined to vertex t implies that the node has highly utilized incoming edges. To get an aggregated value, we sum up all CBR values of each vertex $v \in V$, and divide it by the number of edges that contain vertex v. Next, we construct two sets, the one is *an AACBR Set* and the other is *a BC Set*. We sort each set in descending order based on values of each set. From the highest values, we find *a minimum intersection set* between two sets. It becomes the set V_{hub} where values of the vertices have high ranks in the both sets. Lastly, we need to locate a V_{honey} set which is a set of vertices that are not only neighborhoods of V_{hub}, but also have lower BC than V_{hub}. By selecting neighborhood vertices of the hub nodes, we can prohibit that malicious traffic directly reaches the bottleneck links.

B. Deploying Honey Topology

To make the honey topology complex, we leverage *Barabsi-Albert (BA) model*, which is a random graph algorithm for generating scale-free networks. The BA model is an algorithm based on a stochastic model that determines the probability of connectivity of nodes. The model consists of two simple steps: *(1) growth:* Insert a new node with m edges on existing graph. *(2) preferential attachment:* Attach edges to the inserted node i by a probability of the degree expressed as $p_i = \frac{k_i}{\sum k_j}$ where k_i is the number of degree of node i. We build and deploy the honey topology as software switches since deploying hardware switches is time-consuming and hard to implement, given practical issues such as cabling, switch configurations, and expensive cost.

C. Deceiving Attackers

In the IP network, end hosts usually do not send a packet with a TTL value 1, unless their goal is probing. By exploiting this fact, SDHoneyNet regards the hosts who send packets that have the TTL value 1, as a scout who performs *traceroute*. If the PACKET-IN triggered by the packet is sent from one of our honey nodes, SDHoneyNet instructs the honey nodes to send the packet into the honey topology. Here, we compute a distance from the honey node to all nodes of the honey topology. We select the furthermost node from the honey node, as a final response node in order to force the attacker to fully visit the honey topology.

III. Evaluation

Our evaluations are conducted on a machine with Intel Core i5-6600K @ 3.50GHz and 16 GB RAM. We use ONOS 1.6.0 and Mininet, which emulates networks using OVS v2.3. As an experimental topology, we consider large-scale networks where the node degree follows a power-law distribution. We emulate AT&T North America topology, which is OC-48 fiber network and consists of 25 ISP-level routers from Internet Topology-zoo. Figure 2 shows the snapshot of ONOS GUI that illustrates the result of locating both a hub node and honey nodes. It also shows deployed honey topology on the honey nodes. The red colored switches denote OVSes of the generated topology by the BA model. In a performance test, the deploying time was less than 10 seconds, when the number of deployed switches were 50.

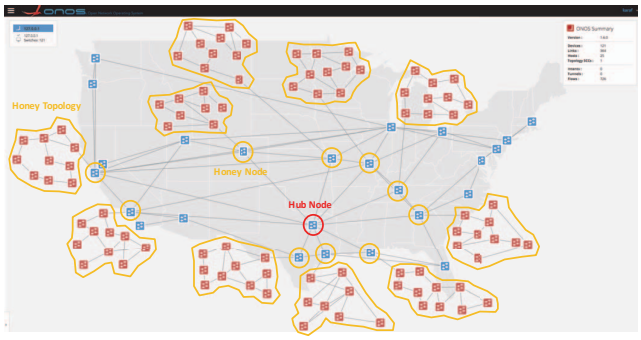

Fig. 2: Result of deploying honey topology

IV. Future Work

We have not yet considered which property is more important than others. Therefore, we will need to empirically assess weights of our attributes for accurately locating the bottleneck links in the real-world infrastructure. Also, in the case of static metric, we expect that there are additional graph metrics that can be leveraged for intelligently selecting the honey nodes.

References

[1] Min Suk Kang, Virgil D Gligor, and Vyas Sekar. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. 2016.

[2] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. The crossfire attack. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 127–141. IEEE, 2013.

[3] Max Schuchard, Abedelaziz Mohaisen, Denis Foo Kune, Nicholas Hopper, Yongdae Kim, and Eugene Y Vasserman. Losing control of the internet: using the data plane to attack the control plane. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 726–728. ACM, 2010.

[4] Lei Xue, Xiapu Luo, Edmond WW Chan, and Xian Zhan. Towards detecting target link flooding attack. In *28th Large Installation System Administration Conference (LISA14)*, pages 90–105, 2014.