Security Considerations in OpenTag Network

Hamid Farhadi, Ryoji Furuhashi, Akihiro Nakao
The University of Tokyo
{farhadi, furuhashi, nakao}@nakao-lab.org

***Abstract*—In this paper, we investigate threats against our previous work (i.e., OpenTag) in terms of *confidentiality, integrity* and *availability*. OpenTag is a tag based network slicing method for wide area coordinated in-network packet processing and traffic classification with performance isolation. OpenTag is an effective mechanism for slicing network testbeds. We also propose mitigation methods regarding the threat model. We design OpenTag Anomaly Detection System (OADS) to detect anomalous activities in the OpenTag network. OADS is distributed and is able to operate at line speed. We implemented and evaluated OADS through which we illustrate the proposed method is efficient and feasible.**

I. Introduction

Wide-area coordinated packet processing has become important for aiding efficient and resilient access to cloud-based services hosted at data centers at the cross-roads of the Internet. For instance, YouTube traffic costs Google $1.65M a day [1]. Efficient cacheing mechanism within the routers is useful for eliminating traffic redundancy [2]. Moreover, to enable in-network packet processing mechanisms in wide area scale networking (e.g., large network testbeds), we have to implement a network slicing mechanism. A network slice is a set of resources allocated for processing of a specific set of packets. Network slicing can have different applications such as providing researchers an isolated set of resources to conduct experiments without interfering each other.

Isolation of resources is one of the major objectives of network slicing, in which the processing operation of packets within a slice is not affected by the other slices. There are three main requirements for wide-area coordinated packet processing: first, enforcing packet processing only on intended packets, second, performance isolation and finally, security isolation.

OpenTag [3] is a slicing mechanism that supports both performance and security isolation. In OpenTag, the user injects a slice ID tag per packet that denotes which slice the packet belongs to. In addition, a redirector is installed on conventional routers so that they can parse the tag and classify them for later slice-specific packet processing. OpenTag is implemented using OCTEON Plus[4] and Click[5]. It is tested with 10Gbps traffic and is able to handle 8000K packets per second [3]. There are some slicing mechanisms available such as PlanetLab [6], CoreLab [7], OpenFlow [8] and FlowFlex [9] in which security considerations are not taken into account. Also a number of other software-based network slicing have been proposed [10] [11] [12]. However, the difference of these solutions from OpenTag is that they are designed primarily for operators not for users as OpenTag is for, and have not taken security isolation into account that OpenTag aims at. OpenTag focuses on avoiding cross-talks, i.e., malicious or unintended injection of packets into slices.

Our contribution in this paper are two folds. First, we propose the threat model of our previous work so called OpenTag. Second, we propose different mitigation plans against threats and design OpenTag Anomaly Detection System (OADS) to detect anomalous activities in the OpenTag network. We implemented and evaluated OADS and propose evaluation results to indicate the effectivity of the method.

In this paper, first we present threats against confidentiality, integrity and availability of the OpenTag. Next, we discuss mitigation methods agains aforementioned threats in terms of encryption and OADS. Finally, we describe evaluation results following by concluding remarks.

II. Threat Model

In this section, we briefly investigate threats against OpenTag in terms of confidentiality, integrity and availability. Note that the scope of this work is limited to threats targeting directly the OpenTag related equipments and data. As instance, threats related to packet payloads or routing issues are beyond the scope of this work.

Based on *defense-in-depth* principal, we investigate each threat group independent of the others. For example, when we discuss integrity related issues, we try to solve the problem in the absence of confidentiality. As the result, the overall mitigation method would be more robust against attacks.

A. Confidentiality

OpenTag deals with a small amount of data (e.g., 128 bits) that describes the tag on each packet. In terms of confidentiality, an attacker may sniff the traffic and try to understand which slice a packet belongs to. Using this information, it is possible to extract some statistics from the network. Also, it is possible to retrieve routing configuration if sniffing is conducted extensively on the network.

B. Integrity

The main challenge which OpenTag deals with is integrity of tags on packets. In terms of integrity, there are a couple of possible attacks (with some overlaps) on the OpenTag:

- *Slice hopping*: In this attack, an intruder is a slice user (i.e., an internal intruder) who tries to modify the tag in order to gain access to another slice, possibly with better network resources (e.g., bandwidth).

978-0-9836283-9-2

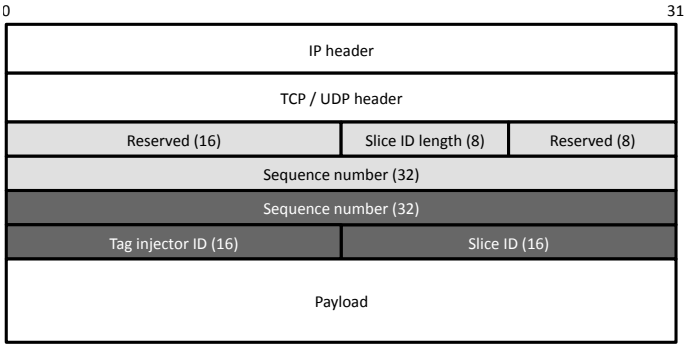

Fig. 1. Paket Layout

- *Tag replay*: This attack consists of extracting the tag from an authorized packet and putting exactly the same tag on an arbitrary packet. That is, if all the packets within a slice carry exactly the same tag, using such an attack an intruder can insert a packet to an arbitrary slice. This type of attack also includes replaying a full sniffed packet on the network that includes a valid tag plus payload
- *Man-In-The-Middle (MITM) attack*: In this type of attack, the attacker intercepts the connection between sender and receiver redirectors and injects arbitrary packets into the connection. S/He may even drop some packets.

The main difference between the replay attack and MITM attack is that the former injects packets blindly into the network. In contrast, the latter intercepts the connection between the source and destination.

C. Availability

The Denial of Service (DoS) attack family is the most common threat against system availability. In our case, the only possible attack against availability is DoS on the redirector, in order to exhaust its hardware resources. For example, an intruder may send abnormal packets to the redirector. The redirector is supposed to detect the abnormality and drop the packet in a timely manner. Due to the simplicity of the OpenTag, it is infeasible to attack on the logical process of the tags.

In the following, we explain hardening methods for confidentiality and integrity of OpenTag. As the threats to OpenTag availability are not feasible, we do not include their mitigations in this paper.

III. Tag Encryption

To provide OpenTag with confidentiality, we need to encrypt tags on every packet. As OpenTag is some sort of realtime system that requires high throughput, we choose AES, a symmetric algorithm, to be able to handle large amount of traffic. The performance evaluation (i.e. 8Mpps) of OpenTag presented in Section 1 included the overhead of full packet encryption. However, here we try to reduce the amount of encryption to improve the overhead. We need a key distribution method to generate and distribute the AES key among nodes. We assume there is a Public Key Infrastructure (PKI) and a

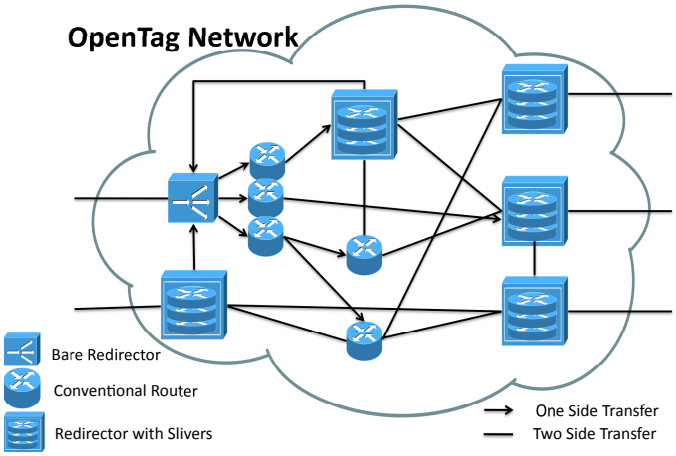

Fig. 2. Sample OpenTag Network

Certificate Authority (CA) available in the environment. All nodes are supposed to have the public key of the CA and it has their public key, as well. The CA generates a key, encrypts it with the receiver public key, sings it digitally and sends the key to the receiver. Finally, we may employ periodic key rollover to improve the overall security of the system.

Figure 1Paket Layoutfigure.1 shows a packet in which the gray section (both dark and light gray) includes OpenTag related data. The dark gray part is encrypted by the sender. The tag consists of three parts: *SenderID (i.e., Tag Injector ID), SliceID* and *Seq#*. They represent the sender redirector ID, the network slice that the packet belongs to, and the sequence number of the packet, respectively.

Once we provided with confidentiality, we can use it to prevent integrity based attacks. In order to prevent all kinds of tag reuse, we need to tie up the tag to its corresponding packet in someway so that the system can determine whether a specific tag is generated for that specific packet or it is copied from another packet. We include a plaintext sequence number in the header (see Figure 1Paket Layoutfigure.1) to check the integrity of the packet. The node that receives the packet, deciphers the tag and matches the decrypted sequence number with the plaintext one in the header. If they do not match, it means that the tag is reused and it does not belong to the current packet. Another functionality of including the sequence number in the cypher text is to keep the freshness of the cypher and makes it variant packet by packet. That is, as the sequence number changes, the resulting cypher changes as well.

IV. OpenTag Anomaly Detection System (OADS)

In previous section we focused on confidentiality threats. In this section we try to focus on integrity related threats. The OpenTag is designed simple and flexible. To preserve its flexibility, we may provide the user with different loosely coupled blocks of security features that can be extended later. Thus, the administrator is able to turn off the confidentiality mechanism for performance or any other considerations, temporarily or permanently based on the environment requirements.

A. Environment Characteristics

First of all, let us define the heterogeneous OpenTag environment characteristics and constraints. Figure 2Sample OpenTag Networkfigure.2 illustrates a possible OpenTag based network. In the following we list OpenTag flexible features to outline the environment requirements:

- A single slice may contain multiple tag injectors (which put the tag on the packet). The portion of traffic is redirected to each injector and also the total number of injectors may change dynamically. Moreover, a single slice may contain multiple tag eliminators (which remove the tag from the packet). The portion of traffic is directed to each of them and also the total number of eliminators in a slice may change dynamically. In short, all the packets sent from a single redirector may not arrive at a single next-hop redirector.
- Once a packet enters the OpenTag network, the path it goes through is unknown to the provider entity (i.e., the host network). Rather, the path is based on the routing configurations of the slice user (i.e., the guest network). Therefore, the path is subject to change. Both OpenTag and its security mechanisms have to operate transparent from the routing path.
- Media between devices are all heterogeneous with different capacities and delays. Therefore, packets may be received out of the sending order at destinations.
- All operations related to tags are transparent from the above and below layers. That is, OpenTag must be able to operate independent of specific protocols such as IP and TCP.
- There are an unknown number of devices at unknown locations in the network topology that are unaware of OpenTag technology. In the other words, there may be some devices that work independent of the tag that may affect the network in many ways. Both OpenTag redirection methods as well as security controls need to operate transparent from other devices. The topological location and operational configuration of these devices are subject to change.
- OpenTag, as a host provider for a couple of guest networks (i.e., network slices) should operate in a real-time manner on a large amount of traffic. That is, any OpenTag security mechanism should meet such constraints.
- Conventional network packets without tags can pass through the OpenTag network. Also, OpenTag packets (whit tags) can be routed by conventional routers without any change since OpenTag puts the tags at the end of the payload. So, the tags are not detectable by conventional switches. The security mechanism should be consistent with this mechanism.

B. Anomalous Activity

Before discussing the detection mechanism, we should define anomalous activity in the OpenTag environment. Our objective is to design an anomaly detection system customized to OpenTag environment. In the following we define terms that we need to design the anomaly detection system.

- *Meta Packet (MP)* : A set of 10,000 conventional network packets. Packets belong to an MP may either be in order, or out of order.
- *Meta ACK (MACK)*: Similar to TCP ACK, MACK is an acknowledgment packet that indicates the successful reception of an MP at the destination. This type of packet is sent back from the destination to the source. The MACK, only shows the total received number of packets regardless of their protocol, order etc.. In contrast to TCP, MACK is used for anomaly detection rather than error detection and correction.
- *Sender redirector (Sender)*: all redirectors except tag eliminators
- *Receiver redirector (Receiver)*: all redirectors except tag injectors
- *Anomalous Activity*: Sending one or more unauthorized MPs across the OpenTag network.
- *OADS*: An anomaly detection system customized to OpenTag needs. It is mainly aimed to detect integrity related attacks in the network.

One may argue that, why do we choose such a big number of packets (i.e. 10,000) as an anomaly, while an exploit code carrying a harmful shellcode, can be transferred using a few packets across the network? Note that OpenTag operates as a host network for a couple of guest networks. In such a scenario, we do not consider attacks targeting guest networks (slices) systems. Instead, we do care about attacks against the host network which is responsible for OpenTag operations. In this granularity, an attack would be sending large amount of data over an arbitrary slice, unauthorizedly. Conventional intrusion detection systems seek for attack data such as XSS, SQL injection, buffer overflow etc., but our detection system looks for large amount of traffic (either benign or attack data) injected to the network. Moreover, OpenTag is slicing mechanism and thud attacks against slicing mechanism are those that target extra resource usage and those that try to violate the the system resource isolation. Other sort of attacks can be mitigated from other layers of the system rather that virtualization level. That is why we defined the MP loosely since it can be highly dependent on the environment. in conclusion, OADS does not fulfill the need to have traditional IDA/IPS boxes, rather it covers attacks against another layer of the system (i.e., virtualization layer).

As a rule of thumb, OpenTag is capable of handling 10Gbps. For details refer to the evaluation section. Considering packet sizes between 80-1500 bytes, a single MP includes 800K-15M Bytes of data. This detection granularity is reasonable for our environment. The amount of data (i.e. 800K-15M Bytes) may be even considered as negligible rather than an attack. That is why we define MP size loosely, such that if we multiply it by 10 or even more, practically, it has minor affects both on detection rate as well as processing overhead.

C. OADS Design

OADS is a decentralized anomaly detection system. Due to the high bandwidth nature of our environment, we broke the processing, down to individual nodes. We detect anomaly at sender redirectors using some feedbacks from adjacent receiver redirectors. OADS detects the slice that caused the anomaly and also determines the topological location of the packets that are injected into the network.

The core of detection mechanism is to count how many packets are sent and received in a pair of hops. Let the number of sent packets at the sender S_1 be n and the number of received packets at the receiver R_1 be m. While the condition $n = m$ holds, there is no anomalous activity in the network. Once we have $n \neq m$, an alert is sent to Slice Manager machine stating that, possibly some packets are injected into the network between S_1 and R_1. Note that, S_1 and R_1 are not two hops connected with a wire so that it is only possible to inject packets via physical access to the media. Even though S_1 and R_1 are adjacent nodes (in terms of tagging), there maybe a couple of non-OpenTag devices (with any possible kind of topology and routing mechanisms) in between. In addition, it is possible for S_1 and R_1 to be adjacent in $slice_1$ and unadjacent in $slice_2$. We use an out-of-band protocol to exchange detection related data. All nodes has a point-to-point connection to exchange detection data.

Algorithm 1OADS Designalgorithm.1, illustrates the feedback mechanism on receiver redirectors. All receiver redirectors, keep a counter for number of packets that they receive. They maintain a counter per slice per sender. For example, assume the receiver redirector R_1, is adjacent to three senders S_1, S_2 and S_3, where S_1 and S_2 handle packets from $slice_1$ and $slice_2$. But, S_3 handles packets from $slice_1$, $slice_2$ and $slice_3$. In this case, R_1 maintains seven counters. Using formal notification, let $W_{R_1} = \{S_1, S_2, \cdots, S_u\}$ be the set of senders that R_1 receives packets from. In addition, let $V_{R_1}^{S_i} = \{sl_1^i, sl_2^i, \cdots, sl_w^i\}$, be the set of slices from which R_1 receives their packets via S_i. Now, the total number of counters that R_1 should maintain is $\sum_{i=1}^{u} |V_{R_1}^{S_i}|$, where $|V_{R_1}^{S_i}|$ indicates the cardinality of $V_{R_1}^{S_i}$. All receiver nodes, send back a MACK to the related sender once the corresponding counter reaches the MP size. Note that, once there is no counter corresponding to a received packet, the system initialize one for it. We may flush the counter from memory after a long idle time. The *lazy* counter setup strategy, keeps $\sum_{j=1}^{h}\sum_{i=1}^{u} |V_{R_j}^{S_i}| \ll z\frac{h(h-1)}{2}$, where:

- $\sum_{j=1}^{h}\sum_{i=1}^{u} |V_{R_j}^{S_i}|$, is the total number of counters in the network,
- z, is the total number of slices in the network,
- h, is the total number of receivers, and
- $z\frac{h(h-1)}{2}$, is the number of counters in a full mesh (i.e. the worst case).

We try to keep the total number of counters and thus the processing power, memory consumption and data transfer in a scale that it is even possible to implement OADS in a centralized manner and gather all MACKs in a single processing point. Similarly, all point-to-point connections are setup in a lazy manner. That is, once the receiver redirector, receives an MP from a sender, it sets up a connection to send back the MACK.

Algorithm 1 OpenTag Feedback

```
1: if (Packet_received) then
2:   if (counter exists for slice.injectorID) then
3:     slice.injectorID.counter++
4:     if (counter > 10,000) then
5:       if (slice.injectorID address is known) then
6:         send(MACK to slice.injectorID)
7:       else
8:         Fetch address from Slice Manager,
9:         and send MACK
10:      end if
11:    end if
12:  else
13:    Initialize a new counter for slice.injectorID
14:    slice.injectorID.counter++
15:  end if
16: end if
```

Algorithm 2OADS Designalgorithm.2, shows the sketch of OpenTag anomaly detection method. In fact, it shows two event that are called after sending an MP and receiving a MACK. As we are performing in a 10Gbps scale, it is not possible to send a TCP like, ACK per packet. So, we choose a higher level of granularity which is MP. Similar to TCP, we have an MACK for each MP. Furthermore, we need a *sliding window* which works similar to TCP. Put it other way, let S_1 and R_1 be a sender and receiver redirectors with n and m as their packet counters for a single slice, respectively. Using MACKs and a sliding window, we can find out the relation between n and m. If $n < m$, that means there are some packets injected to the network. They may be legitimate packets entered to the OpenTag network intentionally by an attacker, or unintentionally as the result of a routing problem (via non-OpenTag devices). In either case, an alert indicating an anomalous activity is fired from S_1. If $n > m$, the network may be congested, so that packets are not reaching R_1, or MACKs are not reaching S_1. A threshold can determine how bigger should n be than m to fire an alarm. Choosing the best threshold is highly dependent on the environment and hence out of the scope of this work.

The sender adds all MACKs from all receives of a single slice to a single sliding window and calculates the total number of acknowledged MPs to detect anomaly. To formalize the problem, let $R_{S_1}^{sl_i} = \{R_1^i, R_2^i, \cdots, R_k^i\}$ be the set of adjacent receivers of sender S_1 for slice sl_i. Also, let $H_{S_1} = \{sl_1, sl_2, \cdots, sl_v\}$, be the set of slices that S_1 receives packets from. Then, S_1 maintains a set of sliding windows $W_{S_1} = \{w_1, w_2, \cdots, w_v\}$. In the worst case scenario, v is the total number of slices in the network.

Algorithm 2 OpenTag Anomaly Detection

```
{After sending an MP}
if (MP_sent) then
    slice.window.end++
    if (slice.window.size > slice.threshold) then
        slice.alert("CongestionDetected")
    end if
end if

{After receiving a MACK}
if (MACK_recieved) then
    if (slice.window.begin = slice.window.end) then
        slice.alert("AnomalyDetected")
    else
        slice.window.begin++
    end if
end if
```

D. Discussion

We do not consider *redirector spoofing* as a threat to OpenTag. Redirector spoofing refers to replacing a redirector with an arbitrary redirector in the network that may behave different from the original one. Similar attack may be considered as a threat to some types of networks such as Wireless Sensor Networks (WSN). However, spoofing a redirector needs physical access to the network and even in case of potential physical access, it is not feasible. Because, once a redirector is included in the network, it should be registered and configured in the Slice Manager manually. So that, other redirectors can query its address to send MACK packets. In this way, a spoofed redirector may not be included in the network without manual registration.

Beside OADS advantages, it has some limitations as well. Assume an MITM attack in which we have a 10Gbps line and the attacker drops 1% of packets randomly. Then, s/he extracts tags from dropped traffic and plugs them into his own packets and starts injecting them into the network. In this way, s/he hijacked 100Mbps of the line speed. This type of attack can not be detected by OADS. However, this can happen only if the encryption system is disabled. If the encryption mechanism is enabled, the attacker can not send all packets through a single slice, as s/he can not extract slice IDs from encrypted tags. Sending packets blindly on different slices is useless, since the destination of packets is indeterministic. It is also not effective to send such packets as a DoS, because the total number of packets in the network does not increase. Once the attacker start adding more packets than s/he drops, the OADS can detect it at the first MP. The same holds for replaying a full packet sniffed from the network for DoS like purposes.

If the encryption mechanism is disabled, the attacker can choose packets belonging to a specific slice and drop them. In practice, the attacker needs to know the routing information of all the slivers of the victim slice to find out the destination of packets. In this case, the OADS can not detect the attack in the absence of confidentiality subsystem. However, in order to implement this attack there are two choices: First, a physical access to network cabling and installing a device without causing any down time to the network; Second, breaking into one of redirectors and changing the firmware and possibly restarting the device. In short, both methods are impractical.

OADS is an anomaly detection system which passively reports anomalies. The administrator is responsible to follow up the alerts, find the root of attack and possibly perform post detection activities. It is possible to turn OADS into an Intrusion Prevention System (IPS) that actively drops attack packets. This is possible only if we limit some flexible properties of OpenTag. However, keeping the high flexibility of OpenTag we can not actively drop attack traffic.

V. Performance Evaluation

In this section we discuss our experiments to indicate the feasibility of the proposal. Figure 3OADS performancefigure.3 illustrates our performance evaluation results. We implemented OADS on Click Modular Router over a multicore x86 environment using 10G interfaces. We used Xena packet generator connected to OADS machine NICs using SFP+ cables. The packet generator sends packets to the OADS machine via an interface and receives packets from another interface to measure the throughput. We repeated the experiment 10 times using different number of CPU cores and for each round we fed 2 million MPs and MACKs to the OADS to see if it can handle such loads. Since OADS algorithms work based on counters, famous notions in intrusion detection literature such as False Positive and True Negative does not make sense in our scenario and that is why we do not measure the detection rate. Rather, we measure the throughput of the system. In other words, the detection rate of the system always has zero False Positive unless the OADS performance is not enough to handle the traffic. The main objective of this experiment is to show how fast OADS can perform in terms of sending and receiving MPs and MACKs and managing the sliding window. The x-axes of Figure 3OADS performancefigure.3 indicates the number of CPU cores we used and the y-axis is the number of MPs and MACKs. Particularly, we can consider the y-axis as the number of packets sent over the network divided by 10,000 which is the size of an MP. For example, on y-axes, 1000 means that 10 million packets are sent over network (i.e., 1000 MPs) and in response 1000 MAKCs are sent back to the sender redirector via the out of band channel. The figure also shows the performance increases linearly using more number of cores. That is, we can achieve higher throughput using more cores.

VI. Related Work

The most similar and famous system to OpenTag is Multiprotocol Label Switching (MPLS). The details of general differences between the OpenTag and the MPLS is beyond the scope of this article. There are few research proposals regarding the MPLS security that can be considered as related work. However, there is a major difference between the MPLS

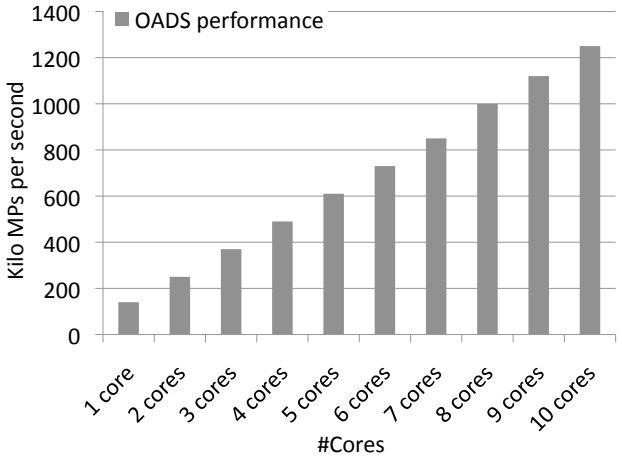

Fig. 3. OADS performance

and the OpenTag in assumptions regarding hardening and security. For example, in the OpenTag network we may have a conventional switch between the OpenTag devices which is a relaxed form of the same constraint in the MPLS case. That is, all devices in an MPLS network should support labeling. Another example is that, depending on the type of MPLS network there should be some trust assumption between provider and user [13]. Furthermore, in some cases no labeled packets are accepted by some routers from some routers (e.g. by the Autonomous System Border Routers (ASBR) from the other ASBRs in VRF-to-VRF connections at AS border routers). RFC 5920 provides security guidelines for MPLS and GMPLS networks. There are a few works focusing on security of MPLS such as [14] and [15].

VII. Conclusion

In this paper, first we explained our previous work called OpenTag and then proposed a threat model against it. Then, we designed several security mechanisms to secure OpenTag against potential threats and proposed our evaluation results. Our proposal has several advantages. First of all, the OADS and the encryption system are loosely coupled, and distributed. Second, OADS can detect anomalies that are not security related and can be used as a network management aid. Moreover, OADS detects the overall topological location of the attack (location aware detection). While we may consider negligible False Negative behavior from OADS, it has zero False Positive rate. We believe security of OpenTag (which can be used to slice networking testbeds) helps scientists to conduct research easier and can help commercial environments to operate more efficiently. Finally, as our future work we are going to extend this model to cover more generalized form of tagging and labeling networks and evaluate it on different types of tag-based systems.

References

[1] Internet Evolution, "Google Losing up to $1.65M a Day on YouTube," http://www.internetevolution.com/author.asp?section_id=715&doc_id=175123.

[2] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *ACM SIGCOMM*, 2008, pp. 219–230.

[3] R. Furuhashi and A. Nakao, "Opentag: Tag-based network slicing for wide-area coordinated in-network packet processing," in *IEEE ICC*, 2011.

[4] "OCTEON Plus MIPS64 Many-core Network Processors," http://www.cavium.com/OCTEON_MIPS64.html.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[6] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: an overlay testbed for broad-coverage services," *SIGCOMM CCR*, vol. 33, no. 3, pp. 3–12, 2003.

[7] A. Nakao, R. Ozaki, and Y. Nishida, "CoreLab: an emerging network testbed employing hosted virtual machine monitor," in *ACM CoNEXT*, 2008, pp. 1–6.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[9] Y. Mundada, R. Sherwood, and N. Feamster, "Distributed Implementation of Coordinated, Network-Wide Policies and Protocols with FlowFlex," *CC Technical Report; GT-CS-10-07, Georgia Institute of Technology*, 2010.

[10] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, "Trellis: a platform for building flexible, fast virtual networks on commodity hardware," in *ACM CoNEXT*, 2008, pp. 1–6.

[11] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Fairness issues in software virtual routers," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow (PRESTO)*, 2008, pp. 33–38.

[12] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, and F. Huici, "Towards high performance virtual routers on commodity hardware," in *ACM CoNEXT*, 2008, pp. 1–12.

[13] Behringer, Michael and Morrow,Monique , "Analyzing MPLS VPN Security. Cisco Press," http://www.ciscopress.com/articles/article.asp?p=418656.

[14] D. Grayson, D. Guernsey, J. Butts, M. Spainhower, and S. Shenoi, "Analysis of security threats to MPLS virtual private networks," *Elsevier International Journal of critical infrastructure protection*, vol. 2, no. 4, pp. 146–153, 2009.

[15] S. Raman, B. Venkat, and G. Raina, "Mitigating Some Security Attacks in MPLS-VPN Model "C"," *International Journal on Advances in Networks and Services*, vol. 5, no. 3 and 4, pp. 304–314, 2012.