DEIDtect: Towards Distributed Elastic Intrusion Detection

Praveen Kumar Shanmugam
spraveen@cs.utah.edu

Naveen Dasa Subramanyam
naveends@cs.utah.edu

Joe Breen
joe.breen@utah.edu

Corey Roach
corey.roach@utah.edu

Jacobus Van der Merwe
kobus@cs.utah.edu

University of Utah
Salt Lake City, UT

Abstract - We present a distributed elastic intrusion detection architecture called DEIDtect. DEIDtect exploits the increasing deployment of cloud computing and software defined networking technology in enterprise and campus environments to deal with current inflexibilities associated with compute and network resources required by security tools. We present the detailed design and implementation of DEIDtect's networking functionality and illustrate its functionality in an emulated environment.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection

Keywords

distributed intrusion detection; software defined networking; cloud computing

1. INTRODUCTION

Intrusion detection and prevention systems (IDS/IPS) are widely deployed as critical tools in the toolkit of security professionals. For example, among the open source IDS/IPS systems, Snort boasts millions of downloads and approximately 400,000 registered users, while it is estimated that as many as 10,000 organizations make use of Bro [1, 14]. Despite its widespread use, current IDS/IPS deployments are plagued by a number of practical concerns that limit their utility. First, the compute and network resources required to effectively run an IDS/IPS often present problematic cost versus functionality tradeoffs: Compute requirements for an IDS/IPS system varies over time depending on the volume of traffic and the type of analysis that security personnel are performing. For example, a developing security event might require more detailed deep packet inspection, which would demand running an IDS/IPS instance configured for this purpose. In practice this results in two undesirable options. Either, compute resources are deployed to accommodate anticipated peak requirements, leading to over provisioning during off-peak times; or more typically, compute resources are knowingly under-provisioned. Under-provisioning results in a loss of visibility during peak times, which, in instances like a DDoS attack, might be when intelligence is most needed.

Network requirements for IDS/IPS deployment involve a tap point in the network infrastructure and sufficient capacity from the tap point to the compute resources hosting the IDS/IPS. A network tap is typically realized using an optical splitter or, in smaller deployments, a switch span/monitoring port. Both approaches are highly inflexible: Once deployed, monitoring is constrained to the chosen tap location which is typically deployed at the ingress/egress point of a campus or enterprise network. The implication is that intrusions that remain within the enterprise network might go undetected.

A more fundamental concern is that current IDS/IPS deployments are typically strictly local concerns. (This remains true in practice despite various earlier efforts towards distributed intrusion detection systems [5, 7, 16].) Specifically, while security professionals at different organizations readily exchange intelligence through personal communication, there is no systematic way to follow a lead to a remote location to investigate the potential source of an attack. Further, it is typically not possible to utilize remote expertise or resources to investigate a local problem.

A final concern is the fact that managing an IDS/IPS system is quite complex. For example, setting up a small scale Snort instance is a well documented activity, typically well within reach for a competent system administrator. However, performing the same activity to scale to campus or enterprise environments quickly becomes a significant engineering challenge [10]. Further, systems like Bro provide more flexibility, customization and analysis capabilities, however, is significantly more involved to set up and requires ongoing management by domain experts.

We argue that the inflexibility associated with the compute and network resources needed for IDS/IPS is the root cause for these concerns and prevent richer cross-domain security models and investigation. In this paper we address these concerns by presenting our work on a *distributed elastic intrusion detection* architecture, called DEIDtect. DEIDtect exploits two technology trends in campus and enterprise network environments: The widespread and continued use of cloud computing to consolidate compute resources, and the increasing deployment of software defined networking (SDN) technology.

Within a specific campus or enterprise domain, DEIDtect uses a general purpose shared cloud infrastructure to elastically scale compute resources needed by IDS/IPS. DEIDtect exposes a new network abstraction through the cloud control interface to allow for IDS/IPS specific traffic distribution within the cloud platform. DEIDtect uses SDN within the campus or enterprise environment to realize flexible and safe tapping of network traffic. DEIDtect

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
DCC'14, August 18, 2014, Chicago, IL, USA.
Copyright 2014 ACM 978-1-4503-2992-7/14/08 ...$15.00.
http://dx.doi.org/10.1145/2627566.2627579.

allows for remote and distributed access to both cloud and network tapping resources. Where available, DEIDtect utilizes wide area SDN networks to realize this distributed functionality in a flexible manner.

DEIDtect involves three forms of inter-domain SDN interaction. First, interaction between the campus/enterprise SDN and the cloud SDN is required to allow tapped traffic to be delivered to the cloud environment for inspection. Second, DEIDtect allows for interaction between cooperating distributed campus or enterprise environments. This distributed functionality enables the remote monitoring of networks, or conversely, the use of remote resources to monitor a network. Finally, DEIDtect requires interaction between wide area and campus/enterprise networks to realize its distributed functionality end-to-end.

DEIDtect's flexible use of compute and network resources directly addresses the core resource related concerns described above. Variable compute resource needs are addressed by allowing general purpose cloud resources to be used for IDS/IPS functions. Flexible network tap realization and flexible routing of tap traffic allow DEIDtect to deal with network resource concerns. DEIDtect does not directly address the management complexity of IDS/IPS, but because it allows great flexibility in terms of networking monitoring and IDS/IPS placement, it does enable a number of scenarios that mitigates the problem. Specifically, the ability to run IDS/IPS remotely allows for outsourcing all, or part, of network security monitoring functionality. Similarly, because of the flexibility it provides, DEIDtect lays the foundation for systematic cross-site IDS/IPS functionality.

We make the following contributions:

- We present the DEIDtect architecture which provides a distributed elastic framework for cross-site security functions.
- We present a detailed design of the networking component of DEIDtect which involves: (i) SDN primitives for safely tapping arbitrary traffic at arbitrary locations in an enterprise network, (ii) cloud abstractions for the precise distribution of traffic in a cloud environment, (iii) a number of inter-domain SDN interactions, including security specific inter-site communication.
- We present the implementation of our design and evaluate it in an emulated SDN environment.

2. CHALLENGES & OPPORTUNITIES

A fundamental concern with current deployments of network security tools is the *inflexibility associated with network and compute resources*. In DEIDtect we address these fundamental concerns by exploiting software defined networking and cloud computing. The DEIDtect approach effectively enables a decoupling of the location of the network being monitored/protected, and the location of the security tools performing security functions. The DEIDtect approach also enables rapid scaling of security resources during a security event. This flexibility provides the opportunity to explore *new distributed network security functionality* and potentially enable better visibility and coordination among partnered organizations.

2.1 Inflexible resource usage

Network tapping: State-of-the-art network security deployments typically involve security groups installing optical tap points at strategic vantage points of the core of an enterprise or university network. Due to cost, availability and scalability issues, these optical taps cannot cover the breadth of the network. As a result, many areas are therefore unseen and unmonitored by security tools. Optical taps also require dedicated fiber and aggregate switch infrastructure, leading to increased costs and deployment complexity. Alternative solutions involve the use of SPAN/mirror ports on switches and routers. This approach can introduce additional load on network devices and under high load captured packets will be dropped in favor of forwarding "normal" data.

DEIDtect dynamic and comprehensive network tapping: DEIDtect exploits SDN functionality to allow the network and system administrators to tap at any point in the network and feed back the tapped traffic stream to security collectors and analyzers. Though limited to the bandwidth of the aggregate links, administrators can implement tap points rapidly at arbitrary points in the network. With a fully instrumented network, administrators can deploy taps anywhere at the access edge, distribution, core, or internet border. With a partially instrumented network, administrators can leverage SDN taps in a flexible manner and optical taps and SPAN/mirror ports at traditional key areas. DEIDtect also allows for fine grained tapping of specific sets of flows.

Compute resources: Network and security administrators have traditionally leveraged a handful of dedicated hardware boxes for security analysis. These boxes may be stand-alone or operated in a cluster. Each of these individual boxes have limits on the amount that they can process. Different flows must be spread across multiple machines in order to scale appropriately. Higher capacity connections or the addition of more tap ports requires more hardware, which takes time to deploy.

DEIDtect elastic security compute platform: The DEIDtect approach allows security and network administrators to leverage cloud resources to spin up virtual or 'bare metal' images of security tools. DEIDtect adds the capability to the normal cloud orchestration architecture to spin up the image and to create a network path from the border of the cloud to the specific host with the security image. By leveraging this technique, DEIDtect can balance multiple flows across multiple security images quickly in order to scale.

2.2 Distributed Network Security Functions

DEIDtect's flexible resource usage enable distributed network security functions. One possible scenario involve a cloud vendor or a large entity, such as a university, company or government with a large private internal cloud, offering elastic cloud based security functions to internal or external customers. Assuming a fully DEIDtect-enabled network and cloud, i.e., DEIDtect controlled SDN in both enterprise and cloud, internal customers can be readily served by dynamically exposing selected tap points to cloud-based security tools. Serving external customers will be possible by similarly deploying DEIDtect technology in the customer network and utilizing wide area SDN infrastructure (or at least semi-static pre-defined circuits) between the customer network and the cloud location.

Another scenario might involve a university or company with a number of remote sites, i.e., small campus sites, field stations or clinics. With the current set of security tools, security administrators rarely have the ability to deploy tap infrastructure with a dedicated feed into the central campus security tool suite. With a DEIDtect-enabled network and cloud this scenario becomes feasible.

A corollary scenario might involve a university or entity that has "sister" sites or smaller campuses with a tight business, academic, research or healthcare association. These discrete sites may wish to leverage common computational resources, security tools or expertise. This arrangement would also allow for greater visibility into emerging security concerns, thus providing a foundation for detecting more subtle attacks.

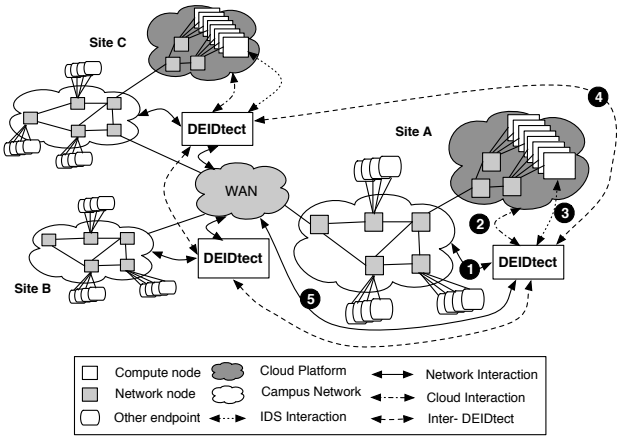

Figure 1: DEIDtect Architecture

3. ARCHITECTURE

DEIDtect exploits two current trends namely the increased use of cloud computing technologies to consolidate compute resources and the increasing deployment of software defined networking (SDN) technology in enterprise, cloud and wide area networks. DEIDtect uses cloud computing resources to flexibly and efficiently deal with the computing needs of security tools, like IDS and IPS, while SDN is utilized to flexibly and safely tap and distribute network traffic between monitored networks and IDS/IPS instances.

A high level view of the DEIDtect architecture is depicted in Figure 1. The figure shows three campus or enterprise networks, i.e., Sites A, B and C, of which sites A and C also have their own cloud computing platforms. The different sites are interconnected via a wide area network (WAN). For ease of exposition, we assume that all networks in question are SDN enabled, although hybrid deployments, e.g., with static or dynamic WAN circuits, would certainly be feasible. We note that the sites could be distributed locations of the same institution, such as a university, or they could be associated with different organizations that have a collaboration or business arrangement to work together on security functions.

Given this underlying physical infrastructure, the DEIDtect architecture involves DEIDtect systems deployed at each site. As shown in Figure 1, the DEIDtect system at each site is involved with five types of interactions. The interaction with: (1) The campus or enterprise network to realize network taps and to transfer tapped traffic towards the IDS/IPS systems in the cloud infrastructure. (2) The cloud computing platform to realize IDS/IPS instances and to manipulate the distribution of tapped traffic towards these instances. (3) The IDS/IPS instances in the cloud to control their intrusion detection and prevention functionality. (4) Remote DEIDtect systems to request and manipulate cloud, network and IDS/IPS resources at remote sites. (5) The wide area network to realize inter-site connectivity.

DEIDtect Use Cases: Different scenarios enabled by DEIDtect are depicted in Figure 2. Figure 2 (a) shows the default case that mimics current common practice. As shown in the figure, an IDS instance is assumed to be operational in the (general purpose) cloud environment. This IDS is fed by a single tap point at the network ingress/egress. A key difference between DEIDtect and conventional deployments is depicted in Figure 2 (b), where a security professional, or the system by itself, determines the need to realize another tap point inside the campus network and spin up another IDS instance (IDS2) in the cloud platform to monitor this new tap point. Finally, Figure 2 (c) depicts an inter-site scenario whereby another IDS instance (IDS3) is realized in the cloud, and in this case traffic from a network tap at a remote site is being monitored by the new IDS instance.

The setup in the last scenario accommodates several different use cases. For example, the security administrator of a remote site may wish to have traffic from its network be analyzed by a more sophisticated setup elsewhere. E.g., site B in Figure 1, which does not have its own cloud infrastructure, might routinely outsource the security functions of its network to sites A or C. Or site C might run its own Snort instance, but might have a need to perform more detailed analysis using a Bro instance administered at site A. Alternatively, the security administrator of site A in Figure 1, might want to investigate an attack originating from site B and since site B does not have a cloud platform to allow dynamic instantiation of IDS instances, the remote tap traffic is relayed back to site A.

Note that the scenarios illustrated in Figure 2 and discussed here are example configurations. A key strength of DEIDtect is its flexible use and manipulation of distributed resources related to security which enables many alternative scenarios.

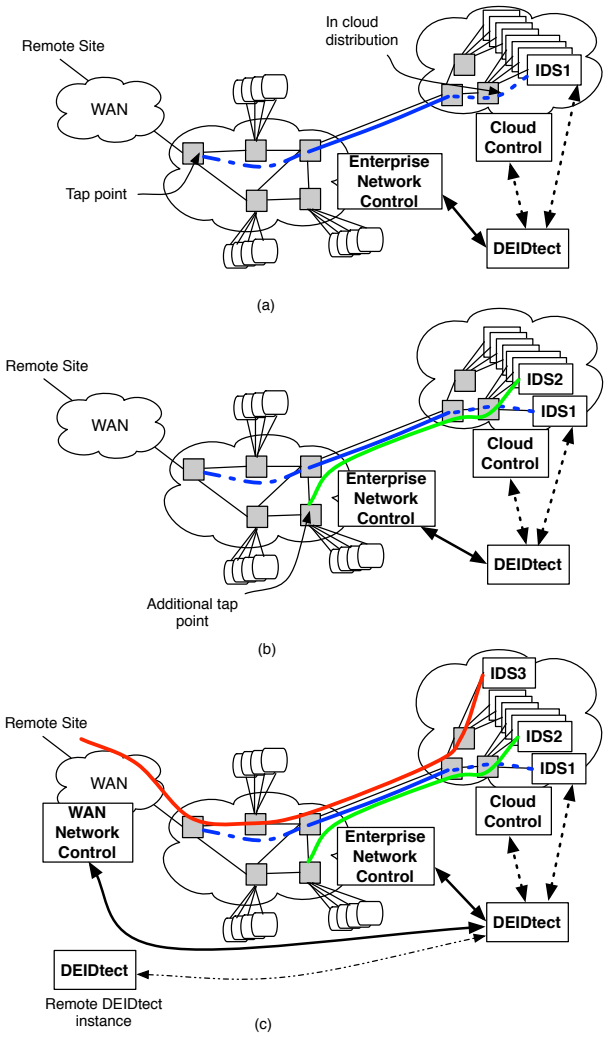

Figure 2: DEIDtect Network Functionality

DEIDtect Inter-domain SDN: From an SDN perspective the DEIDtect architecture involves several types of inter-SDN domain interactions. First, within each site the cloud platform and campus network represent two separate SDN domains. As shown in Figure 2 (a), the campus network is directed by an enterprise network controller to realize the functionality and policies associated with such an environment. The cloud network, on the other hand, is controlled by a cloud control architecture, to realize cloud specific functionality. The inter-SDN requirement here involves DEIDtect coordinating with both the network control and cloud control entities, i.e., across two different domains, to realize distributed elastic detection. Specifically this includes creating tapping resources in the campus network, creating distribution resources in the cloud network and finally orchestrating the interconnection of these resources between the two domains. This is depicted in Figure 2 (a) with the solid line interconnecting the tap resources in the campus network with the distribution resources in the cloud platform, respectively depicted as different types of dotted lines.

A similar set of inter-SDN domain interactions are involved with the inter-site DEIDtect functionality depicted in Figure 2 (c). First, the DEIDtect systems in each site need to interact to realize the required functionality. E.g., setting up a (possibly remote) network tap or instantiating a local or remote IDS instance. Following these application specific interactions, DEIDtect again needs to orchestrate the connection of these sets of resources to realize end-to-end functionality. In this case, however, the orchestration would typically involve interaction with a network controller responsible for interconnecting the distributed sites across the WAN.

DEIDtect System: A system level view of DEIDtect is depicted in Figure 3. At the center of the system is the DEIDtect Core Module which interacts with and orchestrates actions across other system components. Specifically, as shown in the figure and described earlier, the Core Module interacts with five other components in the system: (1) the Enterprise SDN Network to create tap points in the network and to deliver monitored traffic to the cloud, (2) the Cloud Computing Platform to instantiate cloud based IDS/IPS instances and to route traffic from the network to the appropriate IDS/IPS instance, (3) the instantiated IDS/IPS instances to orchestrate intrusion detection and prevention, (4) remote DEIDtect systems to enable distributed security functions and (5) the WAN SDN network to allow delivery of tapped network traffic between distributed locations.

Figure 3 also shows how DEIDtect components (shaded boxes) integrate with existing systems, specifically the cloud computing platform and the enterprise SDN network. As shown in the figure, a DEIDtect Network Module associated with the enterprise SDN network, allows DEIDtect to tap the enterprise network. Similarly, the DEIDtect Cloud Module allows DEIDtect to distribute tapped traffic to appropriate IDS/IPS instances in the cloud.

4. DESIGN & IMPLEMENTATION

In this section we present the design and implementation of the key DEIDtect components. Our current implementation involves the shaded components shown in Figure 3. I.e., our current implementation is limited to the cloud and enterprise network framework, and distributed interactions thereof.

4.1 DEIDtect Network Module

We assume that the enterprise network in question is SDN enabled and specifically supports OpenFlow (version 1.1 or higher). A key DEIDtect requirement is to be able to safely tap arbitrary network traffic at arbitrary points in the network without impacting existing traffic flows. Further, tapped traffic need to be transparently transported to the cloud platform for processing by the IDS/IPS.

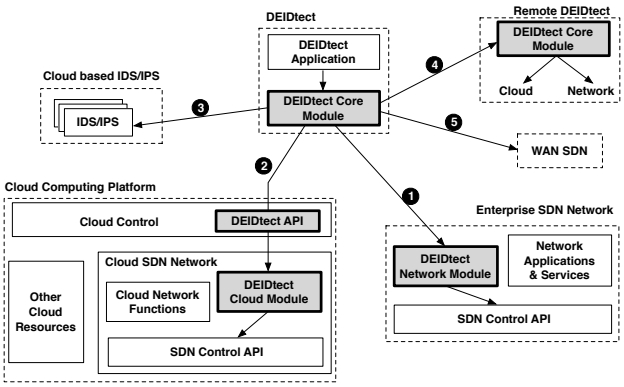

Figure 3: DEIDtect System

Transparent transportation of tapped traffic in DEIDtect is achieved by adding a tunneling tag (e.g., a VLAN tag) to tapped traffic at the tap switch, and to route the tagged traffic to the cloud platform. DEIDtect takes advantage of the multi-table functionality available in OpenFlow version 1.1 (or higher) to achieve safe tapping. Specifically, for flows to be tapped, the normal (existing) flow entry is augmented so that in addition to the existing flow actions the flows are also routed to an IDS specific flow table, which adds the tunneling tag and forwards the packet towards the cloud platform.

Figure 4 shows a single switch tapping example. The top part of the figure shows an existing flow entry that forwards packets received on port 5 out on port 3. The bottom part of the figure shows the modified existing flow entry which continues to output packets on port 3, but also copies the packet for processing to the IDS table *goto IDS_table*. The IDS table entry in turn adds a VLAN tag and sends the packet out on port 2 to complete the tapping action.

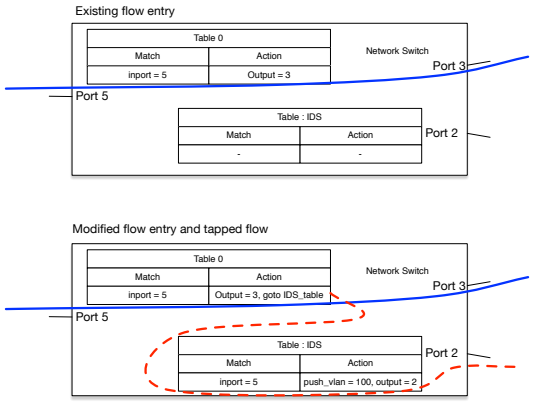

Figure 4: DEIDtect flow tables

Listing 1 shows the pseudo-code associated with flow table manipulation to realize tapping. As shown, DEIDtect allows for ports to be tapped irrespective of whether they function as ingress or egress ports in flow table entries. For ingress ports, DEIDtect looks for the port in question in table 0 and performs the appropriate table manipulation to perform tapping. For egress ports, however,

DEIDtect has to consult all flow tables to find tap port entries, and direct appropriate entries to the IDS table. Our current implementation taps all the traffic from a port in a switch. This can be readily extended to allow more specific traffic to be delivered to the IDS.

Listing 1: Tap flow table manipulation

```
TAPPING_FLOW_TABLE = X,
    and no other application must use this table X.

ingress(switch, port_to_tap, IDS_PORT){
    if(table=0 has ingress_port == port_to_tap){
        1. add(goto_table(,TAPPING_FLOW_TABLE))
           for the existing flow
        2. add flow entry table:X :
           actions:add_tag(packet),output:IDS_PORT
    }
}
egress(switch, port_to_tap, IDS_PORT){
    for(all_table)
    if(actions has output == port_to_tap){
        1. add(goto_table(,TAPPING_FLOW_TABLE))
           for the existing flow
        2. add flow entry table:X :
           actions:add_tag(packet),output:IDS_PORT
    }
}
```

The Network Module maintains state for each switch and each tap request. Tap requests can be removed on demand in which case the state is restored as it was before the tap was realized. The Network Module can also react to the removal of flows by other SDN applications by removing the associated tapping flows. The Network Module exposes the following API functions:

1. *tapFlowEntry(switch_id, port_of_interest, ids_port,vlan_id)*: This goes through the flow table and as described above modifies all the flows having the *port of interest* as the outport in the action of the flow entry. The modification is according to the pseudo-code in Listing 1. I.e., the flow is modified to copy the traffic to the IDS table, where the traffic is tunneled by adding a vlan_id and pushed to the ids_port. A FLOW_MOD_REMOVE message is set when modified flows are installed by the tapping module so that the controller module is notified if the flows are removed by other applications.

2. *clearTapEntry(switch_id, port_of_interest)*: This removes the modification flows associated with the switch and port and reverts back to the old flow entry. This is also invoked in case of a flow removal event by an external application to do the cleanup of the tunnel flows created.

Implementation: We have implemented the Network Module functionality as a module in the RYU controller. Our implementation exports the functions listed above as a RESTful API.

4.2 DEIDtect Cloud Module

The purpose of the DEIDtect Cloud module is to deliver the traffic from the cloud gateway to the appropriate IDS instance. This is done by finding the topology of the network and finding the shortest path between the cloud gateway and the IDS instance and installing the required flows. The flow installation procedure checks for existing flows and split at the lowest possible subtree. This also takes into consideration that the VM's existing traffic must not be disrupted.

As described in Section 3, the DEIDtect Cloud Module forms part of and extends the functionality of an SDN capable cloud computing platform. We abstracted the Network Module functionality into a higher level API that a cloud control architecture would expose to allow DEIDtect to orchestrate cloud network functionality:

1. *createTrafficRoute(tunnel_id, traffic_type, VM_instance)* tells the cloud controller to create an isolated tunnel flow between the cloud gateway switch and the VM instance. Traffic tagged with tunnel_id in the cloud gateway, with the particular traffic_type are delivered to the VM_instance

2. *removeTrafficRoute(tunnel_id, VM_instance)* removes all the flow entries which was installed for *createTrafficRoute* API for the particular VM_instance identified by the tunnel_id.

3. *removeAllRoutes(VM_instance)* removes all the flow entries associated with that VM_instance. In other words, removes all flows installed for *createTrafficRoute* API calls which is associated with VM_instance.

The above higher level DEIDtect cloud API maps to the lower level API of the Cloud Module which is part of the cloud SDN network as shown in Figure 3. The lower level API exposes the following functions:

1. *tapTunnelEntry(src_switch_id, dst_switch_id, dst_port, vlan_id)*: This creates a tunnel from the source switch to the destination switch with the given vlan_id. The dst_port specifies the port which is connected to the VM with respect to the cloud environment and at the destination switch the vlan_id tag is removed and the traffic is delivered as it is seen by the actual destination.

2. *tapTunnelDelEntry(src_switch_id, dst_switch_id, dst_port, vlan_id)*: This removes the tunnel created by *tapTunnelEntry*.

3. *splitTunnelEntry(src_switch_id, dst_switch_id, dst_port, vlan_id, traffic_type)*: This splits the traffic tagged with vlan_id from the source switch and pushes the specified type of traffic_type to the destination switch via the dst_port.

4. *splitTunnelDelEntry(src_switch_id, dst_switch_id, dst_port, vlan_id, traffic_type)*: This removes the changes done by *splitTunnelEntry*.

Implementation: As with the Network Module, we implemented the Cloud Module as a module in the RYU controller. With our current implementation, we have not performed the integration of the higher level DEIDtect cloud API extension with a cloud platform; The DEIDtect Core Module interacts directly with the Cloud Module.

4.3 DEIDtect Core Module

The DEIDtect Core Module performs orchestration functions by interacting with the enterprise network controller, the cloud controller and other DEIDtect Core Modules. The Core Module exposes the following API to enable orchestration across the system as a whole:

1. *CoreSet(switch_id, port_of_interest, ids_port,vlan_id):* Sends a create tap request to the Enterprise SDN Controller.

2. *CoreDelete(switch_id, port_of_interest):* Sends a remove tap request to the Enterprise SDN Controller.

3. *CloudSet(src_switch_id, dst_switch_id, dst_port, vlan_id)*: Sends a create tunnel request to the Cloud SDN Controller.

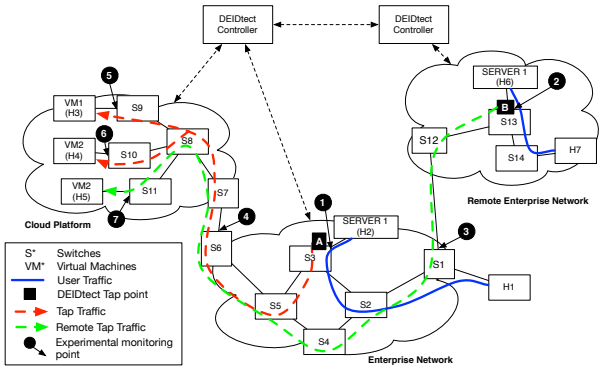

Figure 5: Experimental topology

4. *CloudSplitSet(src_switch_id, dst_switch_id, dst_port, vlan_id, traffic_type)*: Sends a create split traffic request to the Cloud SDN Controller.

5. *CloudDelete(src_switch_id, dst_switch_id, dst_port, vlan_id)*: Sends a remove tunnel request to the Cloud SDN Controller.

6. *CloudSplitDelete(src_switch_id, dst_switch_id, dst_port, vlan_id, traffic_type)*: Sends a remove split traffic request to the Cloud SDN Controller.

Implementation: We have implemented the Core Module as a Python library. We have also implemented a simple command-line-interface (CLI) DEIDtect application to use in our evaluation.

5. EVALUATION

We evaluated our DEIDtect implementation by using Mininet and setting up the topology shown in Figure 5. As shown in the figure we emulated a cloud platform and enterprise network under control of one DEIDtect instance. The topology also contains a remote enterprise network under control of a second DEIDtect instance. We emulated two sets of "normal" traffic flows in this setup using the ***sendip*** utility (shown by the blue lines in Figure 5): First, in the enterprise network between H1 and H2, representing traffic entering the enterprise network from an external network (such as the Internet). Second, between H7 and H6 in the remote enterprise network, representing traffic local to that enterprise network. We demonstrate the DEIDtect functionality by using the DEIDtect CLI application described in Section 4.3 to perform a number of DEIDtect operations. We monitor the traffic in the network topology at a number of experimental monitoring points (the numbered black circles in Figure 5), using the ***ifstat*** utility.

Figure 6 shows a number of time series corresponding to these monitoring points to show the flow of tapped traffic through the topology. We now describe each event shown in the figure: (1) Start TCP and UDP traffic from H1 to H2 and send ping traffic from H7 to H6. The effect of this event is the initial step increase in the amount of traffic shown in all the plots. (Plots 1 to 3 corresponding to the traffic between H1 and H2 and plots 4 and 5 for the H7 to H6 set.) (2) Create a DEIDtect tap for switch S3's port 2 (tap point A in Figure 5) and redirect it towards S6 (and tag it with VLAN_ID 200). The effect of this event is shown in plot 1 with the step increase in traffic associated with monitoring point 4. (3) Create a tunnel between S7 and S9 for all traffic with VLAN_ID 200. The effect of this event is shown in plot 2 where the tapped traffic is now visible at monitoring point 5. (4) Create a split tunnel between S8

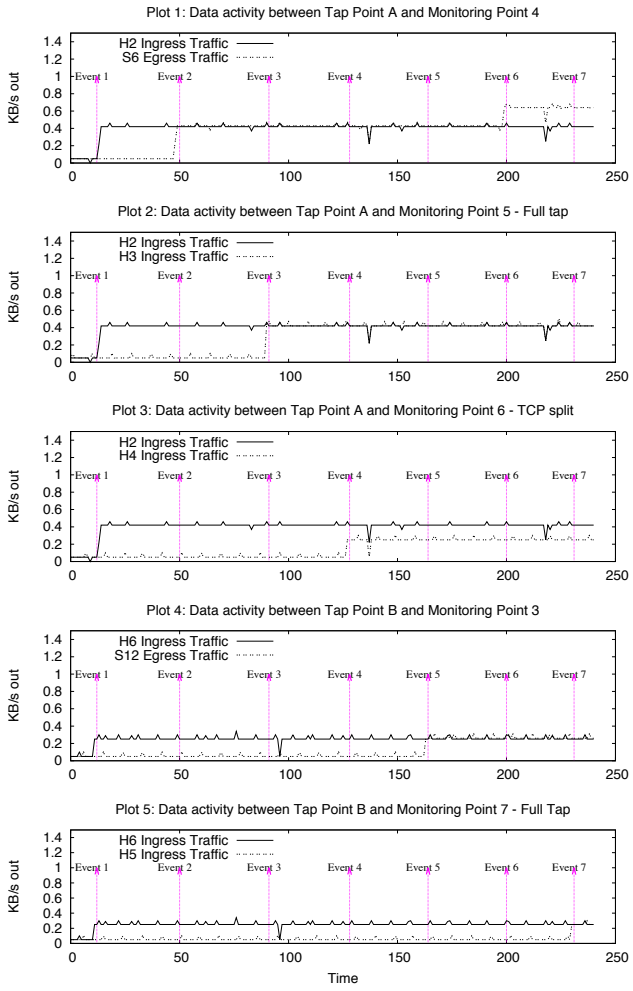

Figure 6: Experimental time series

and S10 for TCP traffic with VLAN_200, i.e., monitoring point 6. The effect of this event is shown in plot 3 where only the TCP traffic associated with the tapped traffic from the enterprise network is directed towards H4. (5) Create a remote tap between S13 and S12 with VLAN_300. (Tap point B in Figure 5.) The effect of this event is shown in plot 4 where the tapped traffic is now visible at tap point 3. (6) Create a tunnel between S1 and S6 for the remote traffic having VLAN_300. As shown in plot 1 after this event the remote traffic shows up at monitoring point 4. (7) Create a tunnel between S7 and S11 for all the remote traffic with VLAN_300. The effect of this event is shown in the traffic step increase in plot 5 associated with monitoring point 7.

6. RELATED WORK

DEIDtect combines cloud computing resources and software defined networking across a variety of domains to realize a distributed network security framework. We touch upon the most relevant related work below.

A variety of SDN cloud networking efforts exist. For example, the use of OpenFlow to develop a networking infrastructure for the cloud which can support millions of IP and MAC addresses by virtualizing layer 2 network has been proposed [12] . The Cloud Broker work [6] uses OpenFlow to connect multiple data centers via

flow based networking. Support for SDN in the popular OpenStack cloud has also been developed [4]. In contrast to these works, DEIDtect exposes an SDN cloud abstraction to allow control of the delivery of network traffic to specific virtual machine instances in the cloud. SDN has also been used in the context of enterprise security functionality. OpenFlow capabilities have been used for the distribution of traffic load from routers into multiple IDS instances [10]. This approach uses SDN in localized fashion with a static set of IDS resources, lacking DEIDtect's network-wide tapping and elastic compute capabilities. Perhaps most related to DEIDtect's enterprise and cloud interworking, the use of SDN to enable communication between enterprise and cloud platforms and to enable inter-cloud workflow is suggested in [3]. DEIDtect realizes a framework to enable a security related workflow that spans across distributed cloud and enterprise instances. To enable the wide area part of our architecture, DEIDtect also assumes the use of inter-domain "stitching" protocols, either using SDN technology [8], or more conventional dynamic circuit establishment [9].

DEIDtect also follows in the footsteps of a variety of distributed security efforts over a long period of time. Dshield [2] is part of the SANS' Internet Storm Center program, allowing firewall users to share intrusion detection information so as to analyze and make them publicly available. Snapp et al. [15] demonstrated a prototype of Distributed IDS (DIDS) that combines distributed monitoring and data reduction with centralized data analysis (through the DIDS Director). A Distributed Intrusion Prevention System (DIPS) has been proposed by Sproull et al. [16]. Our work is complementary to these approaches focusing on the flexible use of network and cloud resources across different domains to realize security functions in a distributed setting.

The scalability of security tools have been addressed by a number of earlier works. For example, IDS clusters to improve the scalability of intrusion detection have been proposed [5, 17]. In [5] a load balancing system is proposed which splits responsibilities of a node to others, replicates traffic to NIDS clusters and aggregate results to split expensive processing at the NIDS. The NIDS cluster work [17] realize highly scalable intrusion detection by running individual IDS instances in a cluster, exchanging low level information among instances. With a complete DEIDtect realization we expect to use similar approaches for the DEIDtect cloud-based IDS instances.

Finally, IDS/IPS in the cloud has been proposed to provide security for cloud tenants [11, 13]. A cooperative IDS network is proposed in [11] to prevent DDOS attacks on the cloud. The integration of an IDS into a cloud environment was demonstrated using Eucalyptus [13]. The focus of these existing approaches is on providing cloud security. In contrast DEIDtect uses the elastic properties of cloud for the security of enterprise networks.

7. CONCLUSION

In this paper we presented our work on the DEIDtect architecture. DEIDtect exploits increased cloud and software defined networking deployments to realize an elastic distributed intrusion detection framework. DEIDtect effectively decouples the location of a network being protected from the location of the security tools performing security functions. This flexibility enables DEIDtect to realize new distributed security functions between partnered organizations. We presented the detailed design and implementation of the networking component of DEIDtect and illustrated its functionality in an emulated network environment. To realize the full potential of DEIDtect our future work includes full integration with a cloud computing platform, developing a policy framework to govern inter-organization security functions, and most importantly developing security applications that can exploit the unique cross-domain functionality of DEIDtect.

Acknowledgements: This research was supported in part by NSF awards 1341034 and 1314945.

8. REFERENCES

[1] www.snort.org.

[2] Dshield - Internet Storm Center. http://www.dshield.org/howto.html.

[3] AZODOLMOLKY, S., WIEDER, P., AND YAHYAPOUR, R. Cloud computing networking: challenges and opportunities for innovations. *Communications Magazine, IEEE* (2013).

[4] CHRIS WRIGHT AND KYLE MESTERY AND ANEES SHAIKH AND STEPHAN BAUCKE. OpenDaylight: An Open Source SDN for Your OpenStack Cloud. OpenStack Summit Presentations.

[5] HEORHIADI, V., REITER, M. K., AND SEKAR, V. New opportunities for load balancing in network-wide intrusion detection systems. CoNEXT '12.

[6] HOUIDI, I., MECHTRI, M., LOUATI, W., AND ZEGHLACHE, D. Cloud service delivery across multiple cloud platforms. In *Services Computing (SCC)* (2011).

[7] JANAKIRAMAN, R., WALDVOGEL, M., AND ZHANG, Q. Indra: A peer-to-peer approach to network intrusion detection and prevention. WETICE '03.

[8] KISSEL, E., FERNANDES, G., JAFFEE, M., SWANY, M., AND ZHANG, M. Driving software defined networks with xsp. In *Communications (ICC)* (June 2012).

[9] LAKE, A., VOLLBRECHT, J., BROWN, A., ZURAWSKI, J., ROBERTSON, D., THOMPSON, M., GUOK, C., CHANIOTAKIS, E., AND LEHMAN, T. Inter-domain Controller (IDC) Protocol Specification. http://www.controlplane.net.

[10] LEHIGH, K., AND KHALFAN, A. Multi-Gigabit Intrusion Detection with OpenFlow and Commodity Clusters. www.openflowhub.org/download/attachments/3244813/SPC-Present.pdf.

[11] LO, C.-C., HUANG, C.-C., AND KU, J. A cooperative intrusion detection system framework for cloud computing networks. In *Parallel Processing Workshops (ICPPW)* (2010).

[12] MATIAS, J., JACOB, E., SANCHEZ, D., AND DEMCHENKO, Y. An openflow based network virtualization framework for the cloud. In *Cloud Computing Technology and Science (CloudCom)* (2011).

[13] MAZZARIELLO, C., BIFULCO, R., AND CANONICO, R. Integrating a network ids into an open source cloud computing environment. In *Information Assurance and Security (IAS)* (2010).

[14] MESSMER, E. Start-up morphs open-source security system for research networks into commercial platform. http://www.networkworld.com/news/2013/071613-broala-271856.html.

[15] SNAPP, S. R., SMAHA, S. E., GRANCE, T., AND TEAL, D. M. The dids (distributed intrusion detection system) prototype. In *USENIX ATC* (1992).

[16] SPROULL, T., AND LOCKWOOD, J. Distributed instrusion prevention in active and extensible networks. IWAN'04.

[17] VALLENTIN, M., SOMMER, R., LEE, J., LERES, C., PAXSON, V., AND TIERNEY, B. The nids cluster: Scalable, stateful network intrusion detection on commodity hardware. RAID'07.