Adaptive Threat Management through the Integration of IDS into Software Defined Networks

Paul Zanna, Benjamin O'Neill, Pj Radcliffe, Sepehr Hosseini, MD. Salman Ul Hoque
RMIT University

Abstract— **For many years network operators have struggled to maintain fragile, statically configured and extremely complex networks. The constant threat of viruses, malware, intruders and misconfigured devices has made the task even more difficult. The use of an Intrusion Detection System (IDS) has become a standard defense model in many networks, however they are expensive and difficult to maintain and further complicate a network. This paper introduces a novel approach that integrates a distributed Intrusion Detection System into a Software Defined Network (SDN) and in doing so provides a more scalable security and threat management solution. The core mechanisms that enable SDN to provide an IDS function have been implemented and their performance evaluated. The viability of this approach was evaluated and found to be an effective alternative to the current IDS deployment model.**

Keywords—Software Defined Network; SDN; Intrusion Detection System; IDS; Threat Management; OpenFlow

I. INTRODUCTION

Modern enterprises, both large and small, are increasingly targeted for theft or damage by cyber criminals either through direct attack methods or via data collection methods such as malware. The volume and sophistication of attack traffic has increased dramatically over the years [1]. As a result, the ability to protect a network from threats is of paramount importance to network operators. The current popular techniques for securing the network include firewalls, Access Control Lists (ACL), Anti-Virus gateways or client-side agents (AV), encryption and Intrusion Detection and Prevention systems (IDS/IPS). Many of these are implemented in the network path at a central route point, so that the majority of network traffic can be examined and filtered. Most of these systems are relatively easy to deploy and maintain in an effective state when in small deployments.

However, the IDS/IPS devices and software have struggled to reach an effective working status in many large organizations due to their complexity and pricing related capacity constraints. These devices are designed to provide alerting and auditing of malicious activity by monitoring network traffic in real-time. An IDS/IPS will be configured to examine network packets for known bad patterns and traffic pattern anomalies, in either detect mode in the case of an IDS and in a prevent mode in the case of an IPS.

Additionally, the required data throughput capacity is also of significant importance. Even in the simplest configuration, these systems are well known for flooding network operators with false positives [2], thereby degrading their effectiveness and as the requirements of users change these false positive grow in number. The primary constraint is throughput, as network operators will need to deploy an IDS/IPS that has sufficient capacity to avoid buffering in IDS mode or packet delay in IPS mode. As with most network devices, the greater the capacity the greater the cost, often to a level that strains IT budgets.

The use of Software Defined Networking (SDN) allows a level of service interaction and decentralization that has not previously been available in the traditional network model. Where most networks rely on static pre-defined configurations, a software-defined network can be a dynamic, highly customizable alternative that is designed to adapt to changing requirements.

The interest in SDN has increased significantly in recent years, not only in research but also in the commercial aspects [3]. As its popularity has increased so too has the interest in its use across a wide range of use cases, from Quality of Service (QoS) and access control to load balancing and service provisioning. Part of this success is due to the use of open standards such as OpenFlow [4] and HTTP for the REST API, this has allowed the easy integration of third party products and services. Combining the extensibility of SDN with the filtering and threat management of IDS will allow operators to provide threat management at any point in the network. In particular, the ability to cost-effectively deploy filtering at the point of ingress would significantly reduce the impact of incidents.

The key question we seek to answer in this paper is whether the SDN architecture can provide an IDS function within the SDN framework, or whether the IDS function should be provided by a traditional architecture. The long goal of this research, which will require many other research questions to be answered, is to produce an SDN/IDS solution that can actively adapt to network usage patterns, thereby providing an agile, scalable, highly responsive, device targeted security management system with far fewer false positives. Ultimately the new architecture will aim to achieve the elusive "working effectively" audit status.

The remainder of the paper is organized in the following manner; Section II provides an overview of the related work on this topic and how it influences this initiative. Section III details the solution design, its components and functional specifications. Section IV details the results of real world testing, adjustments, enhancements and recommendations. Finally, Section V provides suggestions for future work with Section VI concluding the paper.

II. Related Work

Ever since its inception, SDN has had a focus on security. Many of the very first research initiatives including SANE [5] and ETHANE [6] were about providing a secure networking environment. Research into SDN integration with IDS has been minimal with projects such as SnortFlow from Xing et al [7] investigating the use of Open vSwitch (OVS) [8] and Snort [9] for detecting intrusions on a cloud based system. SnortFlow aimed to protect a Cloud cluster using SDN and the well-known utility Snort. Snort was run on the SDN controller but each Cloud server required a Snort agent. While this architecture may suit a data center such an arrangement is not suitable to a typical corporate of campus style network where BYO devices connect and disconnect at will. The paper also uncovered throughput problems with the detection rate falling as the Snort system dropped packets as traffic loads increased. The problems started at about 2.5 MB/sec transfer rate.

Qazi et al [10] examine how SDN can best route traffic to "middle-boxes" which could include IDS systems and proxies. They propose an SDN based policy enforcement layer called SIMPLE, which can translate a logical middle-box configuration into a set of forwarding rules. Such an approach does allow SDN to integrate standard middle-boxes into a network but does miss the opportunity to incorporate such functions into an SDN controller and so eliminate or reduce the need for middle-boxes.

Flowtags [11] like SIMPLE aim to reuse existing middle-boxes but unlike SIMPLE it requires some changes to the boxes. Additionally it has only fixed policies and cannot handle dynamic changes to those policies.

III. Solution Design

After considerable evaluation it was decided that the combination of Ryu [12], a high performance Apache 2.0 licensed SDN controller and Bro [13], an exceedingly customizable, open source IDS would provide a suitable research environment. Bro has had over 15 years worth of research input and has been the IDS of choice in a number of research projects [14], [15], [16]. Bro's scripting capabilities are a significant strength, allowing easy customization, which make it extremely well suited to research. Both applications are under active development with significant community support and are extremely well documented.

The following sections describe the individual components, the interaction between them, and the reasons behind a number of design choices. Finally, the components are allocated to specific hardware devices for implementation.

A. *OpenFlow Switch*

The switch used during the development of this solution as shown in Fig. 1 was an HP 3800, 48 port gigabit, hybrid OpenFlow switch [17]. None of the features used in these tests are specific to this device therefore, any OpenFlow capable switch would have sufficed for this development. The switch was configured with a single OpenFlow aggregate instance consisting of three separate VLANs: Default, VLAN 10 and a management VLAN. The device was also configured at its maximum software flow rate of 10,000 packets per second and the hardware flow rate at left at default.

A number of flows are initially created on the switch by the SDN application at startup. The first is a catchall flow for traffic that was not considered to be of interest to the IDS. This flow used a full wild source and destination with the lowest possible priority value so it would match all traffic that had not matched the IDS flows. It also contained an action that would forward all matching traffic to port NORMAL and therefore allow the switch to process it as standard layer 2 / 3 traffic. Port *NORMAL* is a virtual port available on hybrid OpenFlow switches, which allows a packet to be handled by standard layer 2/3 switching.

A number of individual flows were also installed on the switch by the IDS; each flow was installed to create a one-to-one basis with the corresponding Bro script. The following port and protocols were included in the evaluation:

- DNS (53)
- HTTP (80)
- HTTPS (443)
- FTP (21)
- SMTP (25)
- ICMP

Each of the protocols above had a flow with two forward actions, one output to controller and the other output to port *NORMAL*. The inclusion of the second action reduces load on the controller and controller/switch bandwidth usage as the controller no longer had to generate a *Packet_Out* message back to the switch. In addition to the forward actions, the flow included a *max_len* value set to OFPCML_NO_BUFFER [18], this ensured that the entire TCP/IP payload was sent to the controller instead of the default 128 bytes.

B. *SDN Controller*

There are a number of high quality SDN controllers currently available, many of which would be suitable for a project of this type. The choice to use Ryu was based on a number of considerations including its development language, maturity and supporting documentation just to name a few.

Fig 1 depicts the SDN controller and its numerous integration points. The first is the OpenFlow connection to the networks devices that provide packet data to the controller via *Packet_In* messages. The second is the REST API that provides access the Bro for the creation and/or modification of flows. These flows are used to either direct traffic to the controller as output actions for data collection or, in the event of a threat, drop malicious packets. The final interaction point is providing event messages to the SDN application via a set of internal APIs.

C. Intrusion Detection System (IDS)

The IDS system used in the solution is the Bro IDS, originally developed by International Computer Science Institute [19] and the National Center for Supercomputing Applications [20]. Bro is an open source, commercially supported, security monitoring and network analysis application with a large install base. Of particular interest is Bro's extensibility as a significant amount of its functionality is provided by its event-driven scripting language.

The communication between Ryu and Bro is via cURL [21], an open source command line utility which allows for the transfer of data via a URL syntax. Based on the outputs of the Bro scripts, cURL is invoked to make a REST API call to the SDN controller. This call will instantiate an OFPT_FLOW_MOD request to create a flow with a match field that corresponds to the malicious traffic's source, destination and protocol. The flow that is created will have a high priority and contains no action, therefore dropping the offending traffic or if required diverting traffic to a honey pot service for later investigation.

D. SDN Application

To provide the network data required by the IDS, an SDN application module has been written for Ryu. Written in Python [22], Ryu's native language and using OpenFlow version 1.0, the application consists of two core functions. The first uses the controllers internal API to allow it to register to receive events. The OpenFlow event of interest is the *Packet_In* [18], which is triggered when a network packet is sent to the controller from one of the connected switches. As described earlier, these packets are sent to the controller by flows deployed to the switch with specific actions to forward packets of the required type to the controller. Upon receipt of a *Packet_In* notification from the controller the application retrieves the packet from the event handler and removes the OpenFlow header as it is of no interest to the IDS. These network packets, which still contain the original Ethernet headers, are then sent to a virtual network device driver as described below. The second function provides a REST API used by Bro to notify the SDN controller that an anomaly has been detected and therefore to apply the required counter-measures.

E. TUN/TAP Interface

The TUNTAP [23] device driver is designed to allow user space applications to send network packets via a virtual Ethernet interface. The driver provides two different models of operation, either TUN or TAP. The TUN device driver provides only IP packets, where as the TAP device also includes Ethernet headers in its output. To allow Bro to effectively analyze the network packets the TAP device has been used, as the original Ethernet headers are required. Using the TAP interface allows the integration of Bro with no code modifications as it is presented as a native network interface and therefore can be monitored as if it was any other network interface.

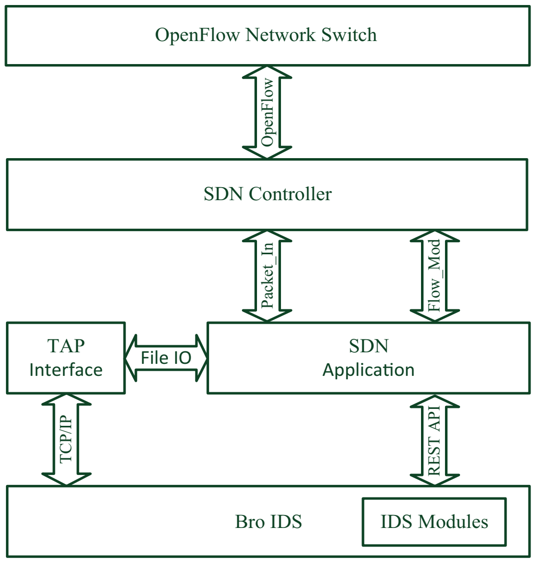

Fig. 1. Solution design detailing interaction between components

F. Allocation to Hardware

A rudimentary local network was built around a HP 3800 OpenFlow switch, the same model described in section III A. The switch was upgraded to firmware version KA.15.15.006 as earlier versions where found to not support the flow action combination of output to *Controller* and *Normal* that were required. This is believed to implementation issue in the earlier versions of the firmware and not an OpenFlow version 1.0 limitation. Three devices were connected to the switch, two test client workstations and an additional workstation, which served as the SDN Controller. This controller integrated the SDN controller, the SDN application, Bro IDS, and the TAP/TUN interface. The workstation was running Ubuntu Server 12.10 and version 3.2 of the Ryu SDN framework. The SDN application as described in section III D was installed as two separate files, one for *Packet_In* processing and the other for providing the REST API. The SDN application was split into two files to facilitate concurrent development and easier debugging.

IV. Evaluation

Initial evaluation of captured traffic was performed using tcpdump [24] and tshark [25] listening on the newly created TAP0 virtual network port. These tools were used to verify that the architecture proposed would perform as expected. As expected the traffic sent to the virtual network port was an accurate copy of the traffic being sent to the SDN controller by the Packet_In messages.

A series of tests were conducted to evaluate the impact on the network performance of having traffic mirrored to the SDN controller. The first set of assessments consisted of a series of Ping tests where blocks of ICMP bursts ranging from 1000 – 5000 echo requests with zero wait time between each ping. The time taken to complete each of the ICMP blocks was recorded and compared based on the following three configuration variations:

- OpenFlow disabled so the switch performed standard layer 2/3 switching.
- OpenFlow enabled with a single flow with a full wild card match and a single action to output to port Normal.
- OpenFlow enabled with a single flow with a full wild card match and a single action to output to port *NORMAL* and to port *CONTROLLER*.

As shown in Fig. 2, the overall time taken to complete the ICMP blocks showed little variation and therefore the latency on a per packet basis was minimal.

The second test sequence was designed to evaluate the overall throughput of network traffic and the effect of different OpenFlow configurations. Files of sizes ranging from 100MB to 500MB were transferred via FTP between the two workstations connected to the switch. Of note is that both workstations were fitted with 100mb/s network cards and therefore the maximum data throughput is 12.5MB/s. Again the three configuration variations were used yet this time there was significant difference in the result. Again, having OpenFlow disabled and having it enabled with a single flow sending to traffic to port *NORMAL* showed similar result, as illustrated in Fig.3. However, this time the use of a flow with actions sending traffic to port *NORMAL* and *CONTROLLER* showed a significant drop in throughput, in some tests up to 34% lower.

One possible explanation for such a decrease in throughput speeds may be a technology limitation in currently available hybrid switches. Many devices are still not able to process all flow types in hardware and therefore these flows use a significantly slower software model. It was found during testing that if the flow had a single action it was managed in hardware, however once the second action was added the flow processing was performed in software. If this is the case then limitation has a greater effect on larger packet sizes as the smaller ICMP packets showed little impact of software flow processing.

The final evaluation was to confirm that the Bro IDS performed correctly in the new architecture. A number of Bro scripts were created and used to confirm functionality, these included:

- A HTTP validation script used to look for certain URL patterns.
- An ICMP threshold script designed to alert to a high volume of ICMP echo requests over a given period.

Each of the Bro scripts required only standard syntax and showed that the existing library of scripts can be easily modified by replacing the *EVENT* function with an “exec::run” command to invoke the required cURL call.

The new architecture clearly demonstrated that existing IDS modules can be integrated into an SDN controller and perform as expected. The arrangement can provide either a centralized IDS using one switch at an appropriate network entry point, or a distributed IDS function where each existing SDN controller can provide an IDS function. The later option effectively gives a high flexible IDS system “for free”.

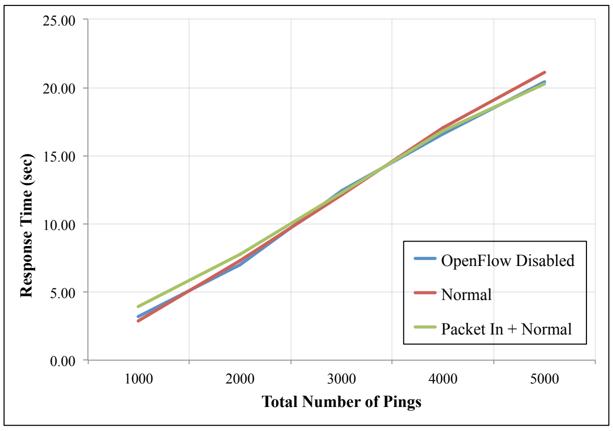

Fig. 2. Response time for varying ping burst sizes

V. Future Work

A number of additional functions can be added to the Ryu/Bro prototype to significantly increase effectiveness and usability. The current implementation is designed to instantiate counter measures in the form of an OpenFlow flow with no actions, effectively blocking access to the switch where the client is connected. There is also option of redirecting, logging or even move the device to a different VLAN to provide isolation. Other potential enhancements would include the ability to provide interval based monitoring. Example of these may include only monitoring client traffic for the first 30 minutes after connection to the network to establish trust or even random inspections. Adding or removing scans is as simple as the modification of the flows to either send traffic to the SDN controller or not.

Additional investigation in performance turning and optimization is also required. The evaluation in section IV showed that the throughput unexpectedly dropped in some circumstances. It would be useful to determine the exact reason for this degradation and whether it occurs in other switches. This information would provide further insight into network placement and usability of an IDS at the edge of the network where a solution like this is most valuable.

One of the significant benefits of integrating IDS into SDN

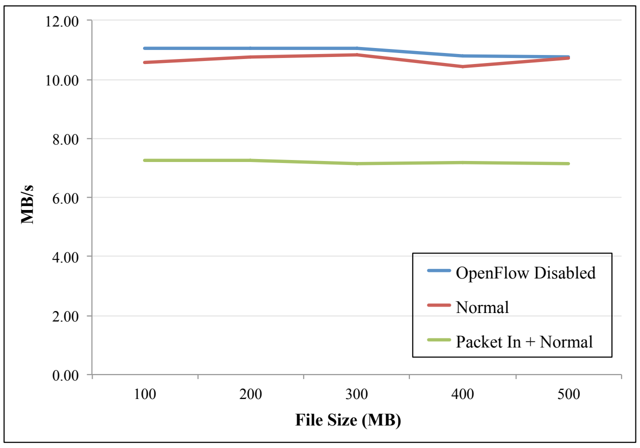

Fig. 3. Average transfer speeds for varying file sizes

is the ability to also leverage the data or functionally of other SDN applications. For example, providing threat information to a bandwidth management application could reduce the impact of Denial of Service (DoS) attacks. In addition to a reactive response to immediate threats, there is also the potential to provide traffic analysis and usage profiling to enable preemptive measures. This may be accomplished by the IDS sending notifications, instead of flow requests, to an access control application, which could implement for example a "3 strikes" policy to permanently ban devices after a series of breaches. The standardization of interactions between SDN applications presents a challenge due to the diverse nature of those applications. This area is a fertile one for further research and would greatly enhance the attractiveness of SDN as a technology.

VI. Conclusion

In conclusion, this paper describes the development of a new architecture for an action oriented Intrusion Detection System that is integrated into an SDN controller. This has been successfully implemented and has been shown to work successfully.

The benefits of implementing such a solution include highly cost-effective distribution of IDS functionality to the access edge, and the ability to take action in near real-time without delaying the initial packet stream. Having intrusion detection integrated directly into the SDN controller not only provides near real-time counter measures but also the ability to provide input into other SDN components.

The solution described in the paper has been targeted at the network edge, specifically end user devices, however a similar model would also work well in a data-center environment. In a data-center deployment, it would be possible to provide a superior level of protection without the cost and possible performance bottleneck of a traditional inline IDS deployment. The solution has the potential to monitor and protect any device connected to an OpenFlow capable switch at a significantly lower cost than traditional IDS deployment. Additionally it provides significantly better coverage by being deployed in this manner as it not only stops malicious data transgressing the core network but also protects clients from localized attacks between devices.

References

[1] "The Akamai State of the Internet Report." [Online]. Available: http://www.akamai.com/stateoftheinternet/.

[2] C.-Y. Ho, Y.-C. Lai, I.-W. Chen, F.-Y. Wang, and W.-H. Tai, "Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems," *IEEE Commun. Mag.*, vol. 50, no. 3, pp. 146–154, Mar. 2012.

[3] "Service Provider Routers and Switches." [Online]. Available: http://www.infonetics.com/pr/2014/4Q13-Service-Provider-Routers-Switches-Market-Highlights.asp.

[4] N. Mckeown, T. Anderson, L. Peterson, J. Rexford, S. Shenker, and S. Louis, "OpenFlow : Enabling Innovation in Campus Networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[5] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: a protection architecture for enterprise networks," in *15th USENIX Security Symposium*, 2006, p. Pp. 137–151.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, 2007.

[7] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "SnortFlow: A OpenFlow-Based Intrusion Prevention System in Cloud Environment," in *2013 Second GENI Research and Educational Experiment Workshop*, 2013, pp. 89–92.

[8] "Open vSwitch." [Online]. Available: http://openvswitch.org.

[9] "Snort." [Online]. Available: https://www.snort.org.

[10] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 27–27–38–38, Sep. 2013.

[11] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "FlowTags:Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, 2013, p. 19.

[12] "RYU SDN Controller." [Online]. Available: http://osrg.github.io/ryu/.

[13] Vern Paxson, "Bro: a System for Detecting Network Intruders in Real-Time," *Comput. Networks*, vol. 31, no. 23–24, pp. 2435–2463, 1999.

[14] A. S. Wu, "Active Event Correlation in Bro IDS to Detect Multi-stage Attacks," in *Fourth IEEE International Workshop on Information Assurance (IWIA '06)*, 2006, pp. 32–50.

[15] C. Kreibich and R. Sommer, "Policy-Controlled Event Management for Distributed Intrusion Detection," in *25th IEEE International Conference on Distributed Computing Systems Workshops*, 2005, pp. 385–391.

[16] T. Seppälä, T. Alapaholuoma, O. Knuuti, J. Ylinen, P. Loula, and K. Hätönen, "Implicit Malpractice and Suspicious Traffic Detection in Large Scale IP Networks," in *2010 Fifth International Conference on Internet Monitoring and Protection*, 2010, pp. 135–140.

[17] "HP 3800 Switch." [Online]. Available: http://h17007.www1.hp.com/us/en/networking/products/switches/HP_3800_Switch_Series/index.aspx.

[18] "OpenFlow Specification." [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[19] "International Computer Science Institute." [Online]. Available: https://www.icsi.berkeley.edu/icsi/.

[20] "National Center for Supercomputing Applications." [Online]. Available: http://www.ncsa.illinois.edu.

[21] "Curl." [Online]. Available: http://curl.haxx.se.

[22] "Python." [Online]. Available: https://www.python.org.

[23] "TUNTAP." [Online]. Available: https://www.kernel.org/doc/Documentation/networking/tuntap.txt.

[24] "tcpdump." [Online]. Available: http://www.tcpdump.org.

[25] "tshark." [Online]. Available: http://www.wireshark.org/docs/man-pages/tshark.html.