P4ID: P4 Enhanced Intrusion Detection

Benjamin Lewis
Computing and Communications
Lancaster University
Lancaster, UK
b.lewis@lancaster.ac.uk

Matthew Broadbent
Computing and Communications
Lancaster University
Lancaster, UK
m.broadbent@lancaster.ac.uk

Nicholas Race
Computing and Communications
Lancaster University
Lancaster, UK
n.race@lancaster.ac.uk

***Abstract*—The growth in scale and capacity of networks in recent years leads to challenges of positioning and scalability of Intrusion Detection Systems (IDS). With the flexibility afforded by programmable dataplanes, it is now possible to perform a new level of intrusion detection in switches themselves. We present P4ID, combining a rule parser, stateless and stateful packet processing using P4, and evaluate it using publicly available datasets. We show that using this technique, we can achieve a significant reduction in traffic being processed by an IDS.**

I. Introduction

With the advent of Software-Defined Networking, both new opportunities and challenges arise. Since the initial publication of OpenFlow [1] in 2008, a number of supporting projects have emerged, eventually leading to P4 [2], a language and set of associated specifications for programming the behaviour of network devices. Conceptually similar to the Active Networks [3] of the late 1990s, P4 brings a new level of flexibility into the packet forwarding process, by programmatically exposing the data-plane.

In this paper, we present P4ID, an implementation in P4 to reduce the processing power required by an IDS for a given size of network. To achieve this, we combine rule-sets designed for traditional Intrusion Detection Systems, such as Snort [4] and apply pre-filtering in the dataplane. This technique allows for the handling of packets in the network itself, without the direct involvement of the IDS. Our results show that we achieve up to a 75% reduction in the amount of traffic being processed by the IDS.

II. Background and Related Work

A. Intrusion Detection

Intrusion Detection is a long-studied field [4] [5], with appliances traditionally situated at ingress into a network. As networks continue to grow in scale, the amount of traffic an IDS has to process increases proportionally, as they typically consume a mirrored copy of network traffic.

Intrusion Detection is usually categorised as using either signature-based or anomaly-based detection. A signature-based IDS consumes a list of rules describing various packet characteristics. These characteristics include IP addresses, ports, protocols, flow-establishment characteristics, as well as packet payload contents. Pre-assembled rule sets are often available, either under a public license or commercially. Anomaly based detection is used by platforms such as Zeek [5]. As an alternative to searching against a list of signatures (as used in a signature-based IDS), anomalies are detected by comparing traffic statistics to those of previously seen activities. However, this behaviour would be challenging to implement in a packet-forwarding environment such as P4.

Intrusion Detection Systems can also act as Intrusion Prevention Systems (IPS). In these instances, traffic is forwarded via, instead of mirrored to, the IDS. Packets are then inspected and either forwarded to their original destination, modified in some way or dropped completely. When acting as an IPS, there is an inherent overhead in terms of latency, as packets pass via the IPS to be processed before being forwarded on to their original destination. P4ID looks to bring some of these benefits without introducing the same overheads.

B. P4 and the Stateful Dataplane

Whilst OpenFlow provides a means for exposing the control plane of a network, it focuses on fixed-function data-planes. As a result, there are a number of versions of OpenFlow, with functionality differing between versions. This can harm interoperability of controllers and applications using OpenFlow.

P4 takes these concepts further by providing a language and set of specifications for defining the packet forwarding behaviour of network devices. The language is centred on the combination of programmable parsers and match-action tables, with both the structure of the tables and the behaviour of their actions being programmer-defined. A key part of the language is that, as used by our implementation, stateful packet processing can be achieved in the data-plane. In OpenFlow, stateful processing without controller intervention is minimal. By using state at this level, it is possible to change forwarding behaviour without controller intervention, providing a low latency response and less load on the controller itself.

C. SDN-based Security

Previous work, including OpenSAFE [6] and FRESCO [7], has aimed to facilitate building new monitoring frameworks, whilst work including PSI [8] and TENNISON [9] focuses on providing a security framework and policy management based around OpenFlow networks. Whilst both offer flexibility and customisable policy, they are still bound by the capabilities of OpenFlow switches, relying on a combination with virtualised network services to achieve the goals of their implementation.

As P4 offers more flexibility in the dataplane, permitting a rudimentary level of computation, research has also examined how it can be used from a security perspective. For example, P4 has been demonstrated as an effective firewall and filter [10], with the ability to keep track of flows [11]; P4Guard [12] describes work in replacing virtualised network firewall functions with those implemented in P4.

Work combining P4 and intrusion detection includes work such as that proposed by Ndonda et al. [13]. The two-layer IDS focuses on white-listing industrial control system messages, by means of rules inserted by a controller system running alongside. Statefit [14] proposes a solution which synchronises state between a number of switches, and uses this to remediate and respond to attacks.

III. Implementation of P4ID

A. Architecture

To realise the benefits of a programmable network for intrusion detection, we propose P4ID, an architecture to reduce the processing load on traditional IDS, by placing additional pre-filtering into network switches. This filtering allows traffic to bypass an IDS where possible, whilst maintaining adequate levels of detection. P4ID is comprised of two key components: the *Rule Parser* and the *P4 implementation*. The Rule Parser consumes rules made for the popular Snort IDS and produces table entries than can be installed into the P4 pipeline.

Within the P4 implementation, there are two distinguishing components, namely the stateless and stateful pipeline stages. Our work and analysis of results from the *stateless stage* led us to develop the *stateful stage*.

B. Rule Parser

The Rule Parser, written in Golang [15], takes rules in the Snort format and translates them into table entries to be installed into P4 switches. This is necessary to allow them to be mapped into match-action tables for the P4 *v1model* architecture.

The parser follows a number of stages: the first stage is searching the rule-set for any rules which cannot be easily represented in P4 match-action tables, notably those using logical *NOT* match criteria. *NOTs* add processing complexity, which does not readily map into a P4 pipeline.

The second stage generates a list of individual intermediate rules: these are generated based on the use of environment variables, used to represent local networks, service ports (HTTP) etc. These rules contain fields that are then stripped out, such as alert messages and payload signatures. These fields build on the traditional 5-tuple by also taking into account the state of a flow and TCP flags.

We use this intermediate stage to search for duplicate rules, as these can cause a table entry conflict when trying to install the rules into a switch. For the purposes of this work, we consider a duplicate rule as any rule matching the same 7-tuple criteria.

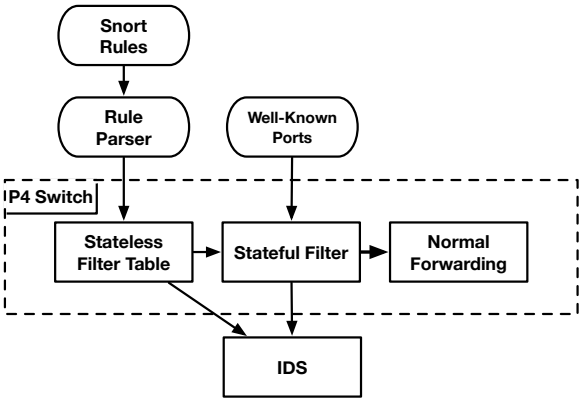

Fig. 1. Architecture of P4ID

The final stage of rule generation takes the intermediate representation and produces individual table entries to be installed.

C. P4 Pipeline

1) Stateless Stage: The stateless stage of the P4 pipeline formed the basis for our initial implementation. This influenced the subsequent stateful stage, which is discussed in Section IV. We first implemented a layer 2/3 hybrid switch in P4, capable of parsing as far as the transport layer. Our parser stores source and destination TCP/UDP ports in metadata fields, allowing one table to cover both protocols. We then add an additional table, the *rule* table. We use guarding logic to ensure that this table is only applied if the packet is TCP, ICMP or UDP. This table contains the following fields: Source IP, Destination IP, Source Port, Destination Port, Protocol and where relevant, TCP flags. We use ternary matching with this table, which better maps to switch hardware and brings other benefits, including allowing a single rule to cover multiple protocols, specifying IP subnets, ranges of ports and individual TCP flags. If a packet matches against a rule in this table, it can either be dropped, forwarded normally or forwarded to the IDS.

2) Stateful Stage: Our work with the stateless stage highlighted some key challenges, which are discussed in detail in Section IV. One of these challenges was that a significant proportion of traffic involves well-known or system ports. This can reduce the efficacy of pre-filtering. We propose that combining a stateful stage (for *well-known* ports) with our stateless stage (for traffic from other ports), allows us to exploit the benefits of both approaches. To achieve this, we instead forward the first N packets of new flows to the IDS. N can be defined on a per-port basis, as well as the timeout that is used. Three tables are used in order to implement this, along with two registers.

The first two tables match on source and destination ports respectively. Each table is populated by the controller with a list of *ports of interest*. We use two tables, as the P4 behavioral model does not support applying the same table twice, and

we wish to capture flows bidirectionally. The third table is applied if either of the previous matched. It contains the packet threshold, flow timeout and egress port for packets destined for the IDS. This third table applies the *conditional forward* action. This action is composed of 3 stages: Firstly, we hash the 5-tuple to generate the *flow identifier*. This is then used to read from two registers: the packet counter, and the last seen register. The latter is used to calculate flow timeouts and is updated by each packet.

By pushing the handling of flow tracking and timeouts to the switch itself, we eliminate the need for the controller to manage idle timeout notifications, which serves to reduce controller overheads and reduce latency by removing the need to push traffic via the controller.

IV. Evaluation

A. Evaluation Architecture

Our evaluation environment is comprised of the P4 behavioral model version *1.12.0-7fd3b395*, running on an Ubuntu 18.04 host. To evaluate P4ID, we combine the CICS2017 [16] dataset with the Emerging Threats [17] ruleset. We chose this combination as we can then demonstrate our solution beyond synthetic benchmarks.

We connect the switch to 3 virtual interfaces using *veth* pairs (traffic in, default traffic out and IDS output) and then install the forwarding, filtering and threshold rules. We use the TCPReplay tool to replay traffic through the switch, via the traffic in interface. The switch will then apply the installed rules to the traffic, producing two output streams: packets destined for the IDS, and packets that have effectively been white-listed, and thus allowed to bypass intrusion detection.

We then capture the traffic destined for the IDS using TCPDump. At this stage, we pass the packet capture file through a custom written pre-processor. The pre-processor parses the packet captures and restores the temporal spacing of the traffic, so that we can more accurately compare against sending all traffic to the IDS.

Finally, we feed this captured traffic into a virtual machine running SecurityOnion [18], using the included *so-import-pcap* tool which supports importing captured traffic, rather than capturing and analysing in real time. Once this is completed, we can use SecurityOnion's own interface to view the logs created.

We then gather figures in terms of the number of alerts generated, categorised into severity (low, medium and high), along with the number of unique signatures detected. The number of signatures detected being particularly important, as it demonstrates the range of attacks being detected.

We use the results from running SecurityOnion against the original CICS2017 captures to form a baseline from which we can compare P4ID.

B. Results

1) Rule Parser: Table I shows the variation between the original rulesets and the rules generated for insertion into a P4 switch. Each rule from the ruleset can unfold into multiple table entries, as they can list a number of IP addresses, ports or other criteria. This contrast is best expressed when comparing the Snort community ruleset [19] to the Emerging Threats [17] set. In the case of the latter, it is possible to generate over 45,000 rules from an original rule-set of 12449. This can occur due to the range of ports covered by variables such as HTTP_PORTS, which, in our configuration, comprises of ports 80, 443 and 8080. When the stateful approach is in use, this reduces significantly, as any rules concerning ports 80 or 443 are monitored on a per-flow basis.

Ruleset	Rules	Un-parseable	Table Entries
Community	971	63	337
Emerging Threats	12449	5655	1162
Security Onion	14454	5659	45024

TABLE I
RULE GENERATOR RESULTS

As described previously, the rules described as un-parseable are those containing logical *NOT* criteria, or those covering an excessive range of ports. We deem an excessive range of ports to be over 512 in this case. Reasons for this are discussed in the next section.

2) Stateless Filtering: Fig. 2 describes some of our results when filtering using the stateless packet approach. Datasets A to E represent the 5 datasets (Monday-Friday) provided by CICS2017. In the best cases, we achieve up to 99% of alerts detected with only 70% of the original traffic. However, in other cases, we ran into some limitations. The limitations from this stage also guided modifications to the rule parser.

In some cases, rules can be very broad, as they depend on being combined with Snort's ability to match on packet payloads. Some of these rules can cover ranges greater than 60,000 ports. Broad rules such as these result in a significant proportion of traffic being pushed to the IDS, reducing the effectiveness of P4ID. To counter this, we have adapted the rule parser to exclude rules covering larger ranges of ports.

Another issue, causing a similar redirection of a large proportion of traffic stems from *well-known* or system ports. As would be expected from the datasets we are using, rules covering these well-known ports include a large proportion of the traffic, again reducing the effectiveness of our stateless filtering. We propose that we instead implement a mechanism to send the initial packets of a flow via the IDS, before passing the rest of the flow through normally.

3) Stateful Filtering: Our results shown by Fig. 3 were gathered by using both the stateful and stateless filtering in conjunction. Datasets A to E represent the 5 datasets provided by CICS2017.

We are applying stateful filtering to the first 100 packets of each flow on ports 22, 53, 80, 443 and 8080. Our experimentation shows that if the timeout is too low, excessive traffic is redirected to the IDS, without increasing detection rates, whereas, if it is too high, detection rate begins to fall again.

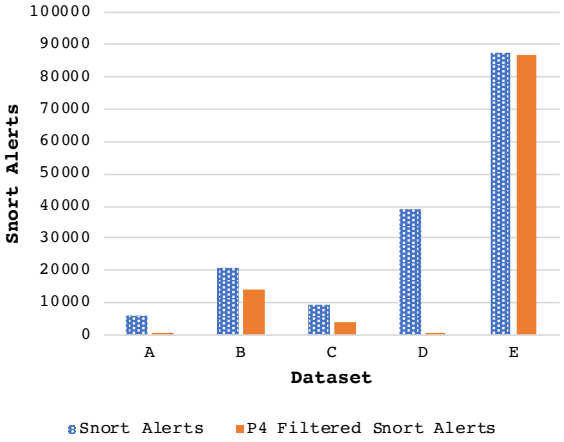

Fig. 2. Filtered vs unfiltered: Snort Alerts

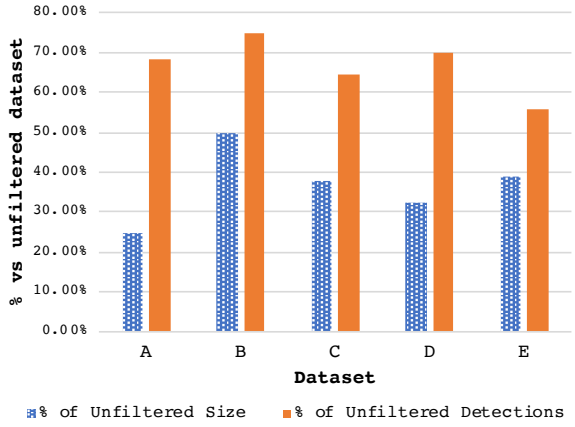

Fig. 3. Stateful filtering percentage of original traffic and detections

For the results shown, we use a timeout of 100ms per flow, with the first 100 packets being sent to the IDS.

In the best case, with stateful filtering applied, we detect 75% of attack signatures, whilst only requiring 50% of traffic to be forwarded. In other cases, the minimum we achieve is a better than 1:1 ratio of detections to traffic.

V. Future Work and Conclusions

In this work, we present a new approach in filtering for intrusion detection by using a P4 stateful dataplane to proactively determine whether traffic should be forwarded to an intrusion detection system. We highlight the relative strengths of our approach in terms of traffic reduction, whilst preserving fidelity in terms of signatures detected.

Future work includes combining this approach with feedback from the IDS. That is, parsing Snort logs to generate table entries to either allow a high-speed bypass, or to force traffic in a given flow to continue going via the IDS. This would allow us to tune the response and flow timeouts based on network conditions. Another element of future work will be to rate limiting traffic to the IDS. With the stateful processing afforded, there are also meters that can be used. In this case, these could be used to actively prevent overloading of the IDS in high-traffic situations.

References

[1] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008, ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746.

[2] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014, ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890.

[3] K. Psounis, "Active networks: Applications, security, safety, and architectures," *IEEE Communications Surveys & Tutorials*, vol. 2, 1999, ISSN: 1553-877x. DOI: 10.1109/comst.1999.5340509.

[4] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks.," in *Lisa*, vol. 99, 1999, pp. 229–238.

[5] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[6] J. R. Ballard *et al.*, "Extensible and scalable network monitoring using opensafe.," in *Inm/wren*, 2010.

[7] S. W. Shin *et al.*, "Fresco: Modular composable security services for software-defined networks," in *20th Annual Network & Distributed System Security Symposium*, Ndss, 2013.

[8] T. Yu *et al.*, "PSI: Precise Security Instrumentation for Enterprise Networks," 2017. DOI: 10.14722/ndss.2017.23200.

[9] L. Fawcett *et al.*, "Tennison: A Distributed SDN Framework for Scalable Network Security," *IEEE Journal on Selected Areas in Communications*, vol. 36, pp. 2805–2818, 2018, ISSN: 0733-8716. DOI: 10.1109/jsac.2018.2871313.

[10] P. Vörös *et al.*, "Security middleware programming using p4," in *International Conference on Human Aspects of Information Security, Privacy, and Trust*, Springer, 2016, pp. 277–287.

[11] C.-H. He *et al.*, "A zero flow entry expiration timeout p4 switch," in *Proceedings of the Symposium on SDN Research*, ACM, 2018, p. 19.

[12] R. Datta *et al.*, "P4guard: Designing p4 based firewall," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, IEEE, 2018, pp. 1–6.

[13] G. K. Ndonda *et al.*, "A two-level intrusion detection system for industrial control system networks using p4," in *Proceedings of the 5th International Symposium for ICS & SCADA Cyber Security Research*, 2018, pp. 31–40.

[14] R.-H. Hwang *et al.*, pp. 168–173, 2018. DOI: 10.1109/I-SPAN.2018.00035. [Online]. Available: https://doi.org/10.1109/I-SPAN.2018.00035.

[15] J. Meyerson, "The go programming language," *IEEE software*, vol. 31, no. 5, pp. 104–104, 2014.

[16] I. Sharafaldin *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," in *ICISSP*, 2018, pp. 108–116.

[17] *Emerging threats ruleset faq*. [Online]. Available: https://docs.emergingthreats.net/bin/view/Main/EmergingFAQ.

[18] S. Onion, *Security onion*. [Online]. Available: https://securityonion.net/.

[19] *Snort rules download*. [Online]. Available: https://www.snort.org/downloads/#rule-downloads.