A Security Enforcement Kernel for OpenFlow Networks

Phillip Porras† Seungwon Shin‡ Vinod Yegneswaran†
Martin Fong† Mabry Tyson† Guofei Gu‡
† SRI International ‡ Texas A&M University
{porras, vinod, mwfong}@csl.sri.com {swshin, guofei}@cse.tamu.edu
{mabry.tyson}@sri.com

ABSTRACT

Software-defined networks facilitate rapid and open innovation at the network control layer by providing a programmable network infrastructure for computing flow policies on demand. However, the dynamism of programmable networks also introduces new security challenges that demand innovative solutions. A critical challenge is efficient detection and reconciliation of potentially conflicting flow rules imposed by dynamic OpenFlow (OF) applications. To that end, we introduce *FortNOX*, a software extension that provides *role-based authorization* and *security constraint enforcement* for the NOX OpenFlow controller. FortNOX enables NOX to check flow rule contradictions in real time, and implements a novel analysis algorithm that is robust even in cases where an adversarial OF application attempts to strategically insert flow rules that would otherwise circumvent flow rules imposed by OF security applications. We demonstrate the utility of FortNOX through a prototype implementation and use it to examine performance and efficiency aspects of the proposed framework.

Categories and Subject Descriptors

C.2.6 [**COMPUTER-COMMUNICATION NETWORKS**]: Internetworking

General Terms

Software-Defined Networking, Security

Keywords

OpenFlow, Security, Policy Enforcement

1. INTRODUCTION

Dynamic network orchestration, driven by the benefits for elasticity of server and desktop virtualization, delivers computing resources and network services on demand, spawned and recycled in reaction to network service requests. Frameworks such as OpenFlow (OF), which embrace the paradigm of highly programmable switch infrastructures [14], compute optimal flow routing rules from remote clients to virtually spawned computing resources. Here, the question of what network security policy is embodied across a set of OF switches is entirely a function of how the current set of OF applications will react to the incoming stream of flow requests. As the state of an OF switch must be continually reprogrammed to address the current flows, the question of what policy was embodied in the switch 5 minutes prior is as elusive to discern as what the policy will be 5 minutes into the future.

Within the OpenFlow community, the need for security policy enforcement is not lost. Efforts to develop *virtual network slicing*, such as in FlowVisor [22] and in the Beacon OpenFlow controller [18], propose to enable secure network operations by segmenting, or *slicing*, network control into independent virtual machines. Each network domain is governed by a self-consistent OF application, which is architected to not interfere with OF applications that govern other network slices. In this sense, OpenFlow security is cast as a *non-interference* property. However, even within a given network slice the problem remains that a network operator may still want to instantiate network security constraints that must be enforced *within* the slice. In this paper, we assert not only that reconciliation of the needs for well-defined security policy enforcement can occur within the emerging software-defined network paradigm, but also that this paradigm offers the opportunity for radically new innovations in dynamic network defense.

The FortNOX Enforcement Kernel. We introduce a new security policy enforcement kernel (called *FortNOX*) as an extension to the open source NOX OpenFlow controller [10]. FortNOX incorporates a live rule *conflict* detection engine, which mediates all OpenFlow rule insertion requests. A *rule conflict* is said to arise when the candidate OpenFlow rule enables or disables a network flow that is otherwise inversely prohibited (or allowed) by existing rules. Rule conflict analysis is performed using a novel algorithm, which we call *alias set rule reduction*, that detects rule contradictions, even in the presence of dynamic flow tunneling using *set* and *goto* actions. When such conflicts are detected, FortNOX may choose to accept or reject the new rule, depending on whether the rule insertion requester is operating with a higher security authorization than that of the authors of the existing conflicting rules. FortNOX implements *role-based authentication* for determining the security authorization of each OF applications (rule producer), and enforces the principle of *least privilege* to ensure the integrity of the mediation process.

Contributions. In summary, our paper makes the following contributions:

- Presentation of the security enforcement challenge in Open-

HotSDN'12, August 13, 2012, Helsinki, Finland.

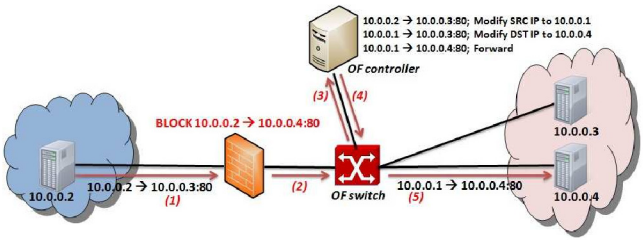

Figure 1: Dynamic Flow Tunneling Scenario

Flow networks and the need for a security enforcement kernel at the controller.

- Extension of NOX to support role-based OF application authentication through digital signatures.
- Development of a novel alias-set rule reduction algorithm for instantaneous and accurate validation of flow-rule contradictions, including those with **set** and **goto** actions.
- A prototype implementation of FortNOX and a preliminary performance evaluation of its overhead. Information regarding our reference implementation of FortNOX is available at http://www.openflowsec.org.

2. SECURITY POLICY-ENFORCEMENT CHALLENGE IN SOFTWARE-DEFINED NETWORKS

Security Policy in software-defined networks (SDNs) is a function of what connection requests are received by OF applications. OF applications may compete, contradict, override one another, incorporate vulnerabilities or possibly be written by adversaries. In the worst case an adversary can use the deterministic OF application to control the state of all OF switches in the network. The possibility of multiple (custom and third-party) OpenFlow applications running on a network controller device introduces a unique policy enforcement challenge: since different applications may insert different control policies *dynamically*, how does the OF controller guarantee that they are not in conflict with each other?

Consider a simple evasion scenario, which we call *dynamic-flow tunneling*, illustrated in Figure 1, that contains three hosts, one OF switch, and one OF controller. A firewall (which can be easily implemented as an OF security application) is used that has a rule to block network packets from the outside host 10.0.0.2 to the web service (port 80) running on the internal host 10.0.0.4. Assume now that some other OF application adds three new flow rules to the OF controller that are linked by GOTO TABLE directives [17]. The first rule modifies the source IP address of a packet to 10.0.0.1 if a packet is delivered from 10.0.0.2 to 10.0.0.3 (port 80). The second rule changes the destination IP address of a packet to 10.0.0.4 if a packet is delivered from 10.0.0.1 to 10.0.0.3 (port 80). The final rule simply allows forwarding a packet from 10.0.0.1 to 10.0.0.4 at port 80. In this case, if the host 10.0.0.2 sends a packet to port 80 of the host 10.0.0.3, this packet can bypass the firewall because it does not directly go to the host 10.0.0.4 but 10.0.0.3. However, this packet will be eventually delivered to the host 10.0.0.4 by the OF controller even if there is a firewall forbidding such traffic. It clearly shows that one can evade an existing firewall (or an OF application implementing the security enforcement) by simply adding

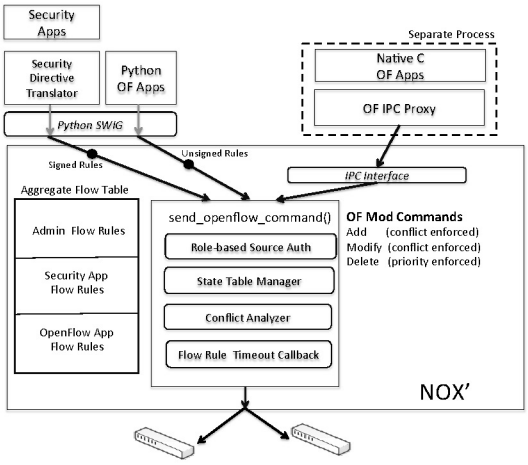

Figure 2: FortNOX Implementation

a few OF flow rules (from some OF applications). While this illustrative example is trivial, the real challenge is ensuring that all OF controller applications do not violate security policies in large real-world (enterprise/cloud) networks with many OF switches, diverse OF applications, and complex security policies. Conducting this kind of job manually is clearly error-prone and challenging.

3. WHAT IS FORTNOX

FortNOX extends the NOX OpenFlow controller by providing non-bypassable policy-based flow rule enforcement over flow rule insertion requests from OpenFlow applications. Its goal is to enhance NOX with an ability to enforce network flow constraints (expressed as flow rules) produced by OF-enabled security applications that wish to reprogram switches in response to perceived runtime operational threats. Once a flow rule is inserted to FortNOX by a security application, no peer OF application can insert flow rules into the OF network that *conflict* with these rules. Further, it enables a human administrator to define a strict network security policy that overrides the set of all dynamically derived flow rules.

By conflict, we refer to one or more candidate flow rules that are determined to enable a communication flow that is otherwise prohibited by one more existing flow rules. FortNOX's ability to prevent conflicts is substantially greater than simple overlap detection, commonly provided in switches. FortNOX comprehends conflicts among flow rules, even when the conflict involves flow rules that use `set` operations to rewrite packet headers in ways that establish virtual tunnels between two end points. FortNOX resolves conflicts in rules by deriving *authorization roles* using digitally signed flow rules (Section 3.1), where each application can sign (or not) its flow rule insertion requests, resulting in a privilege assignment for the candidate flow rule. In Section 3.3, we describe conflict resolution among the flow rules from sources with the same or different authorization roles.

Figure 2 illustrates the components that compose the FortNOX extension to NOX. At the center of NOX lays an interface called send_openflow_command(), which is responsible for relaying flow rules from an OF application to the switch. FortNOX extends this interface with four components. A Role-based Source Authentication module provides digital signature validation for each flow rule insertion request, assigning the appropriate priority to a candidate flow rule, or the lowest priority if no signature is provided (Section 3.1). The Conflict Analyzer is responsible for evaluating each

candidate flow rule against the current set of flow rules within the Aggregate Flow Table (Section 3.2). If the Conflict Analyzer determines that the candidate flow rule is consistent with the current network flow rules, the candidate rule is forwarded to the switch and stored in the aggregate flow table, maintained by the State Table Manager (Section 3.4). FortNOX adds a flow rule timeout callback interface to NOX, which updates the aggregate flow table when switches perform rule expiration.

We add two additional interfaces that enable FortNOX to provide enforced flow rule mediation. First, we introduce an IPC Proxy, which enables a legacy native C OF application to be instantiated as a separate process, and ideally operated from a separate non-privileged account. The proxy interface adds a digital signature extension, enabling these applications to sign flow rule insertion requests, which then enables FortNOX to impose role separations based on these signatures. Through process separation, we are able to enforce a *least privilege* principle in the operation of the control infrastructure. Through the proxy mechanism, OF applications may submit new flow rule insertion requests, but these requests are mediated separately and independently, by the conflict resolution service operated within the controller.

Finally, in Section 3.5, we describe a security directive translator, which enables security applications to express flow constraint policies at a higher layer of abstraction, agnostic to the OF controller, OF protocol version, or switch state. The translator receives security directives from a security application, then translates the directive into applicable flow rules, digitally signing these rules, and forwards them to FortNOX.

3.1 Role-based Source Authentication

FortNOX recognizes by default three authorization roles among those agents that produce flow rule insertion requests. These roles may be augmented with sub-roles, as needed when deployed. The first role is that of human administrators, whose rule insertion requests are assigned the highest priority within FortNOX's conflict resolution scheme, as well as the highest flow rule priority attributes sent to the switch. Second, security applications are assigned a separate authorization role. These security applications produce flow rules that may further constrain the administrator's static network security policy based on newly perceived runtime threats, such as a malicious flow, an infected internal asset, a blacklist-worthy external entity, or an emergent malicious aggregate traffic pattern. Flow insertion requests produced by security applications are assigned a flow rule priority below that of administrator-defined flow rules. Finally, non-security-related OF applications are assigned the lowest priority.

Roles are implemented through a digital signature scheme, in which FortNOX is preconfigured with the public keys of various rule insertion sources. FortNOX augments NOX's flow rule insertion interface to incorporate a digital signature per flow request. If a legacy OF application does not choose to sign its flow rules, those rules are assigned the default role and priority of a standard OpenFlow application.

3.2 Alias Set Rule Reduction

To detect a conflict between a newly inserted candidate OpenFlow rule and the existing OpenFlow rule set, the source and destination IP addresses, their ports, and wild card members we convert all rules, including the candidate rule, into a representation we call *alias reduced rules* (ARRs), and then perform our conflict analysis on these ARRs. An alias reduced rule is simply a derivation of the flow rule in which we expand the rule's match criteria to explicitly incorporate set operation transformations and wildcards. An initial *alias set* is created, containing the first rule's IP addresses, network masks, and ports (where 0 (zero) represents any port). If the rule's action causes a field substitution via a *set action*, the resultant value is added to the alias set, which is then used to replace the criteria portion of the ARR. We then conduct a pairwise analysis of the candidate ARR to the current set of ARRs that represent the active rule set. If there is an intersection between both the source and address sets, the union of the respective sets is used as the subsequent rule's alias set. For example, given the OF security rule,

$$a \rightarrow \text{b drop packet} \tag{1}$$

its source alias set is (a), while its destination alias set is (b). The derived rule is

$$(a) \rightarrow \text{(b) drop packet} \tag{2}$$

For the candidate (evasion) rule set,

$$\begin{array}{l} 1\ \text{a} \rightarrow \text{c set (a} \Rightarrow \text{a')} \\ 2\ \text{a'} \rightarrow \text{c set (c} \Rightarrow \text{b)} \\ 3\ \text{a'} \rightarrow \text{b forward packet} \end{array} \tag{3}$$

the intermediate alias sets are

$$\begin{array}{l} 1\ \text{a} \rightarrow \text{c set (a} \Rightarrow \text{a')} \quad \text{(a, a') (c)} \\ 2\ \text{a'} \rightarrow \text{c set (c} \Rightarrow \text{b)} \quad \text{(a, a') (c, b)} \\ 3\ \text{a'} \rightarrow \text{b forward packet (a, a') (c, b) forward packet} \end{array} \tag{4}$$

and the derived rule is

$$\text{(a, a')} \Rightarrow \text{(c, b) forward packet} \tag{5}$$

3.2.1 Rule Set Conflict Evaluation

FortNOX first performs alias set rule reduction on the candidate rule. These validity checks are then performed between the candidate ARR *cRule* and the set of ARRs representing the active flow rules *fRule*, as follows:

1. Skip any cRule/fRule pair with mismatched prototypes.
2. Skip any cRule/fRule pair whose actions are both either forward or drop packet.
3. If cRule's alias sets intersect those of fRule's, declare a conflict.

Thus, given the example flow description in Equation 2 and the candidate rule set in Equation 5, assuming that both rules are TCP protocol, the first candidate rule passes the first two checks. However, for the third check, because the intersection of the source and destination alias sets results in (a) and (b), respectively, the candidate rule is declared to be in conflict.

As a practical consideration, because OpenFlow rules permit both wildcard field matches and IP address network masks, determining alias set intersection involves more than simple membership equality checks. To accommodate this, we define comparison operators that determine if a field specification is (i) more encompassing ("wider"), (ii) more specific ("narrower"), (iii) equal, or (iv) unequal. Thus, an intersection occurs when the pairwise comparisons between all fields of a candidate rule are wider than, equal to, or narrower than that of the corresponding fields of the constraint table rule.

For a formalization of the above, we first define some terms: (i) S_i is the i_{th} entry of security constraints, (ii) F_i is the i_{th} entry of flow rules, (iii) $SC_{i,j}$ is the j_{th} item of the i_{th} entry of the condition part of security constraint, (iv) SA_i is the i_{th} entry of the action part of the security constraint, (v) $FC_{i,j}$ is the j_{th} item of the i_{th} condition part of a flow rule from unprivileged applications, and (vi) FA_i is the i_{th} action part of the flow rule. At this time, both $SC_{i,j}$ and $FC_{i,j}$ are sets whose elements are one of the specific value or some ranges and $j \in \{1, 2, ..., 14\}$. Rule contradiction is then formalized using the following notation:

$$\begin{aligned} &\text{if there is any } S_i, \text{satisfying } SC_{i,j} \cap FC_{i,j} \neq \oslash \text{ and} \\ &SA_i \neq FA_i, \text{ for all } j, \text{then} F_i \text{ is conflicted with } S_i \end{aligned} \quad (6)$$

3.3 Conflict Resolution

When the above alias rule reduction algorithm detects a conflict between an existing rule in the aggregate flow table and a candidate flow rule, disposition of the candidate rule is evaluated based on the authorization roles possessed by the rule insertion source. If the source of the flow rule insertion request is operating with an authorization role greater that than of the conflicted rule in the aggregate flow table, then the new candidate rule overrides the existing rule. The existing rule is purged from both the aggregate flow table and the switch, and the candidate rule is inserted into both. If the source of the insertion request is a source whose authorization role is lower than that of a conflicting rule in the aggregate flow table, then the new candidate rule is rejected, and an error is returned to the application.

If the source of the insertion requester operates with equal authorization to that of the conflicting rule in the aggregate flow table, then FortNOX enables the administrator to specify the resolution outcome. By default, the new rule will override the previous rule.

3.4 State Table Manager

The State Table Manager and Flow Rule Timeout Callback modules manage the state of all active flow rules that are enforced by FortNOX, as well as their disposition of the rule with respect to the switch's flow table and the authorization role of the rule's producer. When a flow rule is successfully inserted into the switch, its ARR is stored in the aggregate flow table. Rules are deleted through explicit timers provided through the Security Directives Translator (below), or when found in conflict with a candidate rule inserted from a producer operating at a higher authorization level.

When a rule is entered into the aggregate flow table it has the effect of prohibiting the switch from receiving subsequent flow rules that conflict with this rule. However, the switch may currently hold a rule in its flow table that conflicts with the current flow rule. As stated earlier, the aggregate flow table includes an attribute to indicate which rules are resident in the switch's flow table. When a local rule is purged from the aggregate flow table that is also found to be resident in the flow table switch, FortNOX asks the switch to delete the lower-priority conflicting rule, and it adds the new higher-priority rule to the switch.

3.5 Security Directive Translation

The Security Directive Translator is a python interface that mediates a set of high-level threat mitigation directives into flow rules, which are then digitally signed and submitted to FortNOX. The current security directive translator implements seven security directives: *block*, *deny*, *allow*, *redirect*, *quarantine*, *undo*, *constrain* and *info*. Block implements a full duplex filter between a CIDR Block and the internal network, where the primary use for this command is in blacklist enforcement. The deny, allow, undo and info directives are similar to their firewall counterparts and capable of being refined down to an individual flow. However, these two directives are implemented using the address resolution protocol, and thus enforce both directly conflicting flows, and indirect flows involving the use of `set` commands designed to establish indirect tunnels.

Redirect enables a security application to tunnel all flows between a source and given target, to a new target location of the security application's choice. The switch is directed to rewrite the packet headers of all applicable flows such that the source cannot tell that its flows have been redirected to the new target. A common application for this directive includes the redirection of a malicious scanner into a honeynet. The quarantine directive enables a security application to essentially isolate an internal host from the network. It further redirects all HTTP communications that are initiated from the quarantined machine to a proxy server that may report quarantine notifications to the end user who may be operating the HTTP browser. Finally, the constrain directive enables one to deactivate all current flow rules in the switch that are *not* set to priority N. This directive may be used in an emergency operating mode, such as a DDoS, where during the emergency only a pre-defined set of flows should be enabled (i.e., an emergency policy can be specified and activated, overriding all other flow rules until the emergency is remediated).

4. IMPLEMENTATION

FortNOX is implemented as a native C++ extension of the NOX source code in approximately 500 lines of C++ code. We modified the `send_openflow_command` function, whose main operation is to send OpenFlow commands to network switches, to capture flow rules from all OpenFlow applications, i.e., of both security (privileged) and non-security (unprivileged) applications. FortNOX intercepts flow rules in the function and stores them into the security constraints table if the rules are from privileged security applications (i.e., flow rules produced through the privileged path are considered trusted flow rules and are preserved as active network security constraints). If a flow rule is from an unprivileged application, FortNOX evaluates the rule to determine if a conflict exists within its security constraints table. If there are conflicts, an error message is returned to the OF application. Otherwise, the rule is forwarded to the network switches.

5. EVALUATION

To analyze the performance overhead of FortNOX in conducting inline OpenFlow rule conflict analysis, we deployed our FortNOX prototype into a laboratory network operating an HP ProCurve E6600 OpenFlow-enabled switch, firmware version K.15.05.5001. We compiled FortNOX into NOX version destiny 0.9.1 (full beta), and conducted our flow rule conflict analysis using the alias set rule reduction algorithm. FortNOX was hosted on an Intel Xeon 2.67 GHz E5640 CPU with 12 GB RAM, a 1 Gbps network link, and running an Ubuntu-Server v10.10. The experiment was conducted using a dedicated packet generator and a second server used for receiving flows, both linked to OF-enabled ports on the ProCurve via 1 Gbps network links. Packet generation was performed using *hping* [20] version 3.0.0 Alpha 2 operated from a dedicated Intel Xeon 3.2 GHz CPU running Ubuntu-Server v10.10 and directed to a similarly provisioned server. The server wass configured and provisioned roughly identical to the traffic generator.

In the experiment, our objective was to measure the computational impact of conducting flow rule conflict analysis between a candidate flow rule and an increasingly large corpus of active

flow rules. To baseline the experiment, we employed NOX with an optimized module_manager.py for flow rule insertion. Module_manager.py was modified to insert 4-tuple-based flow rules into the ProCurve per unique flow request. In our baseline test, we evaluate the NOX rule insertion handling logic by generating 1000 flow rule insertions at a uniform rate of 100 new UDP flows per second, where each UDP flow caused NOX to produce a new flow rule. The UDP flows were directed at an address running the Discard service so that no response packets would be generated. Since our focus was to isolate the computational overhead of flow rule conflict evaluation, this traffic profile was selected to minimize delays due to switch-local garbage collection and switch-side buffering. The same experimental traffic profile was then run using the FortNOX controller where we varied the experiment by evaluating the same 1000 candidate flow rule insertions. However, this time we pre-seeded the FortNOX aggregate flow table with 1, 10, 100, 500, and 1000, unique resident flow rules, against which the set of 1000 candidate flows were evaluated for conflicts.

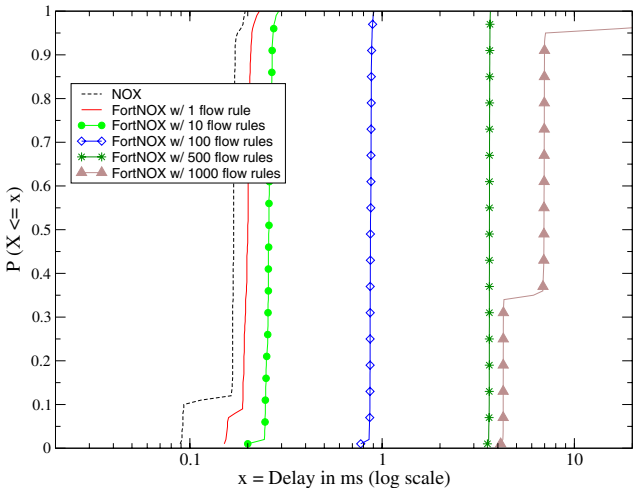

Figure 3: These CDF plots present the computational latencies imposed over UDP flows by NOX and FortNOX. Five plots illustrate FortNOX performance in conducting conflict evaluation per flow against 1, 10, 100, 500 and 1000 flow rules.

In Figure 3, we illustrate the results through a series of cumulative distribution functions representing the computational delay required to conduct full pairwise rule conflict analysis. The experiment suggests that FortNOX's conflict evaluation overhead is in the worst case linear with respect to the number of rules. Note, our experiments represent a uniform priority scenario (with no whitelisting rules) in which we force an evaluation of each incoming flow against each resident flow rule, which in practice would represent a worst case scenario. Our algorithm pre-segments active flow rules based on priorities and actions type, which in operational scenarios enables substantial prefiltering of the number of active flow rules that a candidate rule would be evaluated against (e.g., a candidate forward rule would be subject to conflict evaluation against other forward rules).

6. RELATED WORK

The FortNOX security kernel is inspired by prior research focused on testing or verifying firewall and network device configuration [7, 13, 2, 23, 1], e.g., using Firewall Decision Diagrams (FDDs) [13] or test case generators [21, 7]. The problem of routing misconfigurations has been well studied in the context of interdomain routing protocols like BGP. For example, researchers have investigated the problem of modeling network devices to conduct reachability analysis [2, 23]. The router configuration checker (rcc) uses constraint solving and static analysis to find faults in BGP configurations [8]. Their system detects faults leading to invalid routes and invisible routes. In this space, our work is perhaps most closely related to *header space analysis*, a static analysis approach to detecting network misconfigurations [11].

The OpenFlow standard has as its roots a rich body of work on control-flow separation and clean-slate design of the Internet (e.g., [5], [9]). SANE [6] and Ethane [5] propose new architectures for securing enterprise networks. The SANE [6] protection layer proposes a fork-lift (clean-slate) approach to upgrading enterprise network security that introduces a centralized server, i.e., domain controller, to authenticate all elements in the network and grant access to services in the form of capabilities that are enforced at each switch. Ethane [5] is a more practical and backwards-compatible instantiation of SANE that requires no modification to end hosts. Ethane switches reside alongside traditional network switches and communicate with the centralized controller that implements policy. Both studies may be considered as catalysts for the emergence of OpenFlow and software-defined networking. FortNOX is built over the foundations laid by these studies and shares a common objective in improving enterprise security using programmable network elements.

We build our system on NOX, which is an opensource OF controller [10]; however, our methodology could be extended to other architectures like Beacon [18], Maestro [3], and DevoFlow [15]. FlowVisor is a platform-independent OF controller that uses network slicing to separate logical network planes, allowing multiple researchers to run experiments safely and independently in the same production OpenFlow network [22]. FlowVisor cares primarily about non-interference *across* different logical planes (slices) but does not instantiate network security constraints *within* a slice. It is possible that an OF application uses packet modification functions resulting in flow rules that are applied across multiple network switches within the same slice. In such cases, we need a security enforcement kernel such as FortNOX to resolve conflicts.

The need for better policy validation and enforcement mechanisms has been touched on by prior and concurrent research efforts. The Resonance architecture enables dynamic access control and monitoring in SDN environments [16]. The FlowChecker system encodes OpenFlow flow tables into Binary Decision Diagrams (BDD) and uses model checking [1] to verify security properties. However, the evaluation of FlowChecker does not consider handling of *set* action commands, which we consider to be a significant distinguisher for OpenFlow networks. NICE provides a model-checking framework that uses symbolic execution for automating the testing of OpenFlow applications [4]. More recently, researchers have proposed developing language abstractions to guarantee consistency of flow updates in software-defined networks [19]. In contrast, our complementary work on the FortNOX security enforcement kernel is focused on detection of rule update conflicts and security policy violations. Veriflow proposes to slice the network into equivalence classes to efficiently check for invariant property violations [12]. The alias set rule reduction algorithm of FortNOX is complementary to this approach.

7. CONCLUSION

We motivated and presented the design of FortNOX, a software extension that empowers OF security applications with the ability to produce enforceable flow constraints. Our design incorpo-

rates several critical components that are necessary for enabling security applications in OF networks including role-based authorization, rule reduction, conflict evaluation, policy synchronization, and security directive translation. Our prototype implementation demonstrates the feasibility and viability of our alias-set rule reduction approach. FortNOX rule conflict analysis imposes minimal additional latency over standard NOX, with an average overhead of less than 7 ms for evaluating a candidate flow rule against 1000 existing flow rules. FortNOX is an important first step in improving the security of OF networks, but much work remains in building out a rich suite of applications that cover a wide range of security services.

8. ACKNOWLEDGMENTS

We are grateful for helpful comments from Peter Neumann and the anonymous reviewers to an earlier version of this paper. This material is based upon work supported through the U.S. Army Research Office under the Cyber-TA Research Grant No. W911NF-06-1-0316 and NSF Grant No. CNS-0954096. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Army Research Office or the National Science Foundation.

9. REFERENCES

[1] E. Al-Shaer and S. Al-Haj. FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures. In *Proceedings of the 3rd ACM SafeConfig Workshop*, 2010.

[2] E. Al-shaer, W. Marrero, A. El-atawy, and K. Elbadawi. Network Configuration in A Box: Towards End-to-End Verification of Network Reachability and Security. In *Proceedings of the IEEE International Conference on Network Protocols*, 2009.

[3] Z. Cai, A. L. Cox, and T. E. Ng. Maestro: A System for Scalable OpenFlow Control. In *Rice University Technical Report*, 2010.

[4] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. A NICE Way to Test OpenFlow Applications. In *Proceedings of the Symposium on Network Systems Design and Implementation*, 2012.

[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *Proceedings of ACM SIGCOMM*, 2007.

[6] M. Casado, T. Garfinkel, M. Freedman, A. Akella, D. Boneh, N. McKeowon, and S. Shenker. SANE: A Protection Architecture for Enterprise Networks. In *Proceedings of the Usenix Security Symposium*, 2006.

[7] A. El-atawy, T. Samak, Z. Wali, E. Al-shaer, F. Lin, C. Pham, and S. Li. An Automated Framework for Validating Firewall Policy Enforcement. Technical report, 2007.

[8] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proceedings of theUsenix Symposium on Network Systems Design and Implementation*, 2005.

[9] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. In *Proceedings of ACM Computer Communications Review*, 2005.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. In *Proceedings of ACM Computer Communications Review*, July 2008.

[11] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *Proceedings of the Symposium on Network Systems Design and Implementation*, 2012.

[12] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *Proceedings of ACM Sigcomm HotSDN Workshop*, 2012.

[13] A. Liu. Formal Verification of Firewall Policies. In *Proceedings of the International Conference on Communications (ICC)*, 2008.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. In *Proceedings of ACM Computer Communications Review*, April 2008.

[15] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee. DevoFlow: Cost-effective Flow Management for High Performance Enterprise Networks. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets)*, 2010.

[16] A. Nayak, A. Reimers, N. Feamster, and R. Clark. Resonance: Dynamic Access Control for Enterprise Networks. In *Proceedings of the 1st ACM SIGCOMM WREN Workshop*, 2009.

[17] OpenFlow. OpenFlow 1.1.0 Specification. `http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf`.

[18] OpenFlowHub. BEACON. `http://www.openflowhub.org/display/Beacon`.

[19] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent Update for Software-Defined Networks: Change You Can Believe In! In *Proceedings of the ACM Workshop on Hot Topics in Networks*, 2011.

[20] S. Sanfilippo. HPing home page. `http://www.hping.org`.

[21] D. Senn, D. Basin, and G. Caronni. Firewall Conformance Testing. In *Proceedings of the IFIP TestCom*, 2005.

[22] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the Production Network Be the Testbed. In *Proceedings of the Usenix Symposium on Operating System Design and Implementation (OSDI)*, 2010.

[23] G. Xie, J. Zhan, D. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On Static Reachability Analysis of IP Networks. In *Proceeding of IEEE INFOCOM*, 2005.