

## Bridging the Gap Between Security Tools and SDN Controllers★

Li Wang and Dinghao Wu\*

College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA, USA

### Abstract

Software-Defined Networking (SDN) is a promising paradigm to improve network security protections. However, current SDN-based security solutions can hardly provide sufficient protections in a real SDN network, due to several reasons: 1) they are implemented at either the centralized SDN controllers or the decentralized network devices, which are subject to a performance limitation; 2) their designs are confined by the SDN network characteristics and can only provide limited security functions; and 3) many solutions have deployment challenges and compatibility issues. In this paper, we propose SecControl, a practical network protection framework combining the existing security tools and SDN technologies, to produce a comprehensive network security solution in an SDN environment. We implement a SecControl prototype with OpenFlow and evaluate its effectiveness and performance. Our experiment shows that SecControl can cooperate with many mainstream security tools and provide effective defense responses over SDN-supported networks.

Received on 16 November 2018; accepted on 19 December 2018; published on 21 December 2018

**Keywords:** Software-defined networking (SDN), Network Function Virtualization (NFV), OpenFlow, SDN security application, SDN controller

Copyright © 2018 Li Wang and Dinghao Wu, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

doi:10.4108/eai.10-1-2019.156242

### 1. Introduction

Software-Defined Networking (SDN) has gained much attention in both academia and industry [24]. By decoupling the control logic from the closed and pre-designed network devices, SDN enables the reprogramming capability of network devices. Previously, traditional network devices can only work as they are manufactured, and all their traffic control and forwarding functions are not changeable once produced. With SDN, the traffic control functions and traffic forwarding functions are divided as *control plane* and *data plane*. The separation of *control plane* and *data plane* provides a powerful and flexible network structure for various network applications.

A lot of network-related research has been conducted with SDN, such as network management [9, 10, 21], network QoS [15], network load balancing [18, 40], and content delivery system [41]. Similarly, researchers tried to take advantage of SDN technologies to devise new network security solutions as well. Many innovations [34–37] tried to provide better security services over software-defined networks, and they are provided either at the centralized controllers or the distributed inline network devices.

However, the existing SDN-based security solutions can hardly compete with traditional security solutions due to various reasons. First, they are designed with limitations inherently. When security functions are implemented at centralized controllers [35], the processing capabilities of controllers will become a potential bottleneck; when security functions are deployed at network devices [37], it can hardly provide a comprehensive protection over the network. Second, most of them are focusing on maximizing the control flexibility of SDN. Maximizing network control flexibility does not necessarily lead to strengthened

\*A preliminary version [39] of this article appeared in *Proceedings of the Workshop on Applications and Techniques in Cyber Security (ATCS)*, co-located with the 13th EAI International Conference on Security and Privacy in Communication Networks (SecureComm '17), Niagara Falls, Canada, October 22–25, 2017.

\*Corresponding author. Email: [dwu@ist.psu.edu](mailto:dwu@ist.psu.edu)

network protection ability. Third, the existing SDN-based security solutions are mainly on a certain aspect of network protection [20], which can hardly satisfy the general network protection requirements. Last, many of them have deployment challenges and compatibility issues.

As a result, the current SDN-based security solutions cannot provide the same protection capabilities as traditional security tools can provide over SDN networks. Actually, the key innovations brought by SDN are over network control instead of security processing capability. Network protection demands more powerful security processing capabilities, such as packet payload inspection, traffic pattern analysis, and so on. Therefore, we need a practical network security solution which can provide competitive security protection and, at the same time, can take advantage of the flexible control over SDN networks.

Traditional security tools, like firewalls and intrusion detection systems, have strong security processing capabilities in protecting traditional network infrastructures, and each type of security tool is specialized to deal with a certain type of security threat. They are composed together to form a comprehensive network security solution. However, traditional security tools can hardly be used directly in software-defined networks because of the following reasons: 1) existing security tools are designed under the traditional network infrastructure, which does not fit into SDN network structure; 2) most security tools are devised to deal with a certain type of security threat. Their exclusive designs decide they can only be used individually and cannot cooperate with each other; and 3) there is no interface on existing security tools to let them take advantage of SDN benefits.

In this paper, we propose SecControl, a new network protection framework bridging the gap between security tools and SDN technologies, to provide sufficient protection capabilities in an SDN environment. Our goal is to design a practical and comprehensive network security solution over SDN networks by leveraging existing security tools and SDN control flexibility. Unlike existing SDN-based security solutions, SecControl is designed on a new security control layer above SDN controllers, which releases SDN controllers from security processing pressure. SecControl is able to perceive the real-time security threats, generate real-time defense reactions, and adjust corresponding network behaviors dynamically. With SecControl, security engineers can easily add different security tools into the protection boundary and make use of their detection abilities to serve the entire network. Our method can be applied on mainstream SDN platforms without difficulty.

In summary, the main contributions of SecControl are as follows.

- We propose a novel network protection framework for software-defined networks, which combines the existing security tools and SDN technologies. Our framework retrofits and reuses the existing security tools in the SDN context, which avoids re-development of many security defense functionalities.
- Our method equips an SDN network with strong security processing capabilities in an economic way. Existing security tools can be used to protect SDN networks without difficulty.
- SecControl layer provides an additional layer above SDN controllers, which release controllers from security processing pressures. SecControl has a full security view of the protected network domain, which enables SecControl to offer a unified protection.
- We design a practical method to dynamically translate defense responses into SDN rules to adjust network behaviors. We provide a set of SDN primitives, namely *drop*, *forward*, *reflect*, *isolate*, and *copy*, and these primitives can be translated to OpenFlow flow rules automatically.
- SecControl separates the security processing logic from the security enforcement components. With our method, a SecControl domain can receive remote protection instructions from other SecControl domains, which enables a unified SecControl protection over different SDN networks.

The remainder of the paper is organized as follows. We introduce the challenges of SecControl in Section 2. In Section 3, we discuss SecControl's architecture and how it is designed. Section 5 describes a SecControl prototype implementing with OpenFlow. The evaluation is presented in Section 6. In Section 7, we talk about a few insights obtained from this work. We briefly summarize the related work in Section 8. Finally, a conclusion is given in Section 9.

## 2. Challenges

Our goal is to design a practical network security solution in SDN networks by employing the security processing capabilities of traditional security tools and SDN technologies. To achieve it, we need to answer several research questions.

### RQ1. How Does SDN Improve Network Security Protection?

Network security was once regarded as a subset of network management problem [10]. The key innovation of SDN is separating *control plane* and *data plane* which maximizes the network control flexibility.

However, Maximizing network control flexibility does not necessarily lead to the strengthened network protection ability. We may need to think how can we use SDN to improve security. For example, how to assign security responsibilities to *control plane* and *data plane*? How to dynamically adjust network behaviors against security threats?

### RQ2. How to Fit Traditional Security Tools into SDN Networks?

Although traditional security tools have powerful security processing capabilities, they cannot be used in an SDN environment directly. The reasons are summarized as follows: 1) traditional security tools are invented for traditional network infrastructure, which can hardly fit into the SDN structure; 2) these tools do not have interaction interfaces for using SDN features to improve security; and 3) seldom can existing security tools share threat information with each other since they are designed individually, and that is a weak point in defending SDN networks. Based on the above reasons, we need to answer how to fit existing security tools into SDN networks? For example, How do we place security tools in an SDN network? How can we collect threat information from traditional security tools?

### RQ3. How Can We Combine Them Together?

To make use of the protection capabilities of traditional security tools and maximize the SDN benefits in securing networks, we need to make them work together. Consider most security tools are designed for traditional networks instead of SDN networks, there are several practical issues when combining them together. The first issue is the current security tools are heterogeneous, and their detection results are not compatible. For instance, a host-based intrusion detection system will be mainly monitoring system behaviors, while a firewall will be interested in suspicious network activities. The log generated by the two tools can hardly join together for further security analysis. The second issue is we lack an interaction mechanism for security tools to communicate with SDN networks. We want to employ real-time threats information to adjust network behaviors dynamically. The last issue is we need a unified method to translate the semantics of threat information into SDN rules. For example, how do we extract effective threats information from heterogeneous security event information? Given a certain security threat, how do we adjust network behaviors for an effective defense? How do we distribute defense decisions in an SDN network?

## 3. System Architecture

In this section, we introduce the SecControl architecture. We first give an overview of SecControl. Then, we

illustrate the three-layer structure of SecControl as well as the main functions. Last, we explain how SecControl works.

### 3.1. Overall Architecture

SecControl seeks to make use of existing security tools to build up a practical protection framework to maximize SDN benefits in protecting production networks. Through collecting various threats information from various security tools, SecControl converges heterogeneous security alerts to one point. By using specific security analysis and detection algorithms, SecControl identifies attack evidence, accesses an overall security situation, and generate corresponding defense responses. Figure 1 shows the SecControl architecture. As can be seen from the figure, SecControl architecture is divided into three layers, Threat Collecting Layer, SecControl Layer, and SDN Controller Layer. Each layer plays a different role in the SecControl protection framework. The Threat Collecting Layer is in the upper part, which is composed of various security tools and Threat Collecting Agents. Consider SecControl aims to provide an overall protection framework, it needs to provide enough compatibility supports to various security tools. Each security tool will be attached a customized Threat Collecting Agent which is devised to cooperate with this tool. As Figure 1 shows, a security tool is represented by a blue rectangle, and the small triangle stands for the attached Threat Collecting Agent. The Threat Collecting Layer collects security events and send them to the SecControl Layer, and the SecControl Layer will transform the threat information into defense responses. After the defense responses are translated into specific SDN rules, they will be passed down to the SDN Controller Layer. The SDN Controller Layer will distribute the SDN rules to corresponding SDN devices for enforcement. The work flow will be given in next subsection.

The Threat Collecting Layer includes two parts, traditional security tools and Threat Collecting Agents, whose responsibility is gathering effective security event information from various security tools. Due to that existing SDN solutions are weak in enhancing security processing abilities of SDN networks essentially, we try to borrow the existing powerful security tools to provide SecControl stronger security processing abilities. The security tools are deployed for detecting security threats, while the Threat Collecting Agents are for collecting the latest security event information detected by these tools. The function of Threat Collecting Agents is to provide a uniform interface to fit the traditional security tools into the SecControl Layer. As Figure 1 shows, each security tool will be assigned a Threat Collecting Agent, which is equipped with customized interfaces for recording detection information on that

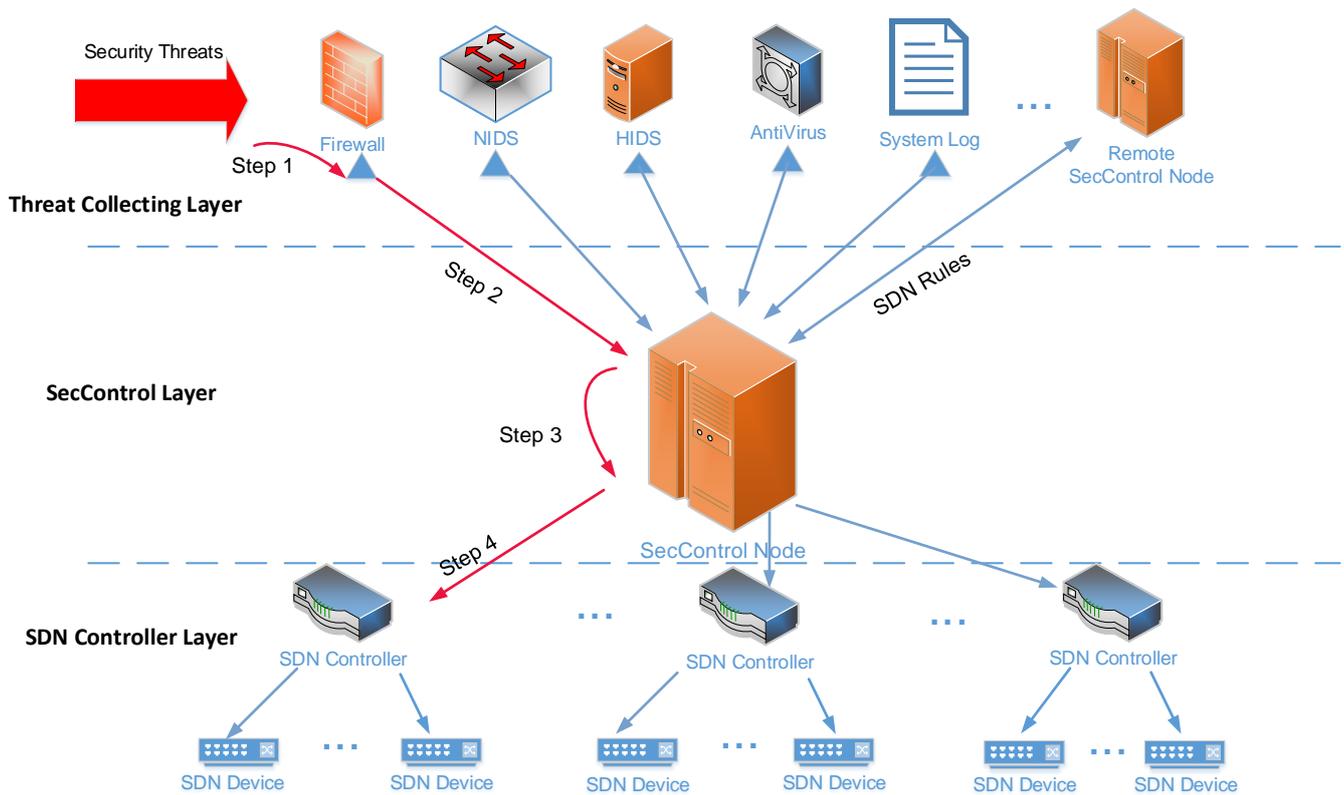


Figure 1. The SecControl Architecture

tool. The collected information will go through a pre-processing procedure which transforms heterogenous and unorganized information into a uniform format. After the threat information is preprocessed and organized, then it will be sent to the SecControl Layer for further processing.

The SecControl Layer is located in the middle of the other two layers, which contains only one component, the SecControl Node. The SecControl Node is the key component in our framework, which is designed to bridge the connection between the security tools and SDN controllers. When receiving security event records from the Threat Collecting Layer, the SecControl Node will keep running a series of standard steps. The standard steps include converging all the collected security events, correlating related alerts, analyzing alert information, and abstracting attacking evidence. After that, the SecControl Node will decide how to make a defense response to the detected security threats over SDN networks. The responses will be given in SDN rules, and these SDN rules will be distributed to the related SDN controllers for enforcement. As a result, SecControl adjusts the network behaviors based on the security threats. Our work is mainly on this layer, and there are a lot of research questions to be solved. More details about how we design the inside process of the SecControl Node is presented in the next section.

The SDN Controller Layer is responsible for enforcing the SDN rules for our protection framework, which is formed by many SDN controllers. The SDN controllers we use here are just standard SDN controllers. When receiving SDN rules from the SecControl Node, a controller will check its local device list to locate the related network devices. Then, the controller will notify the corresponding network devices and send out the latest SDN rules. The new SDN rules will be installed and enforced at SDN network devices. The revolutionary design of SDN is decoupling the functions of current network devices into control plane and data plane. The control plane is implemented as an SDN controller, while the data plane is deployed as new SDN network devices. In SDN networks, every SDN controller will be connecting at least one or more network devices, and these devices will be maintained and managed by this SDN controller. Network devices will share state information with their controllers, and controllers will collect statistic information from network devices. When there is an update request, the SDN controller will send out SDN rules to the related network devices to update data plane rules.

### 3.2. How SecControl Works

The working process of SecControl is shown in Figure 1. There are four steps: (1) The security threats are detected by security tools and the detection results are recorded and processed by Threat Collecting Agents (ThreatCA). (2) The preprocessed threat information is sent from ThreatCA to the SecControl Node. (3) The SecControl Node converges and analyzes the threat information to decide how to make a defense response over SDN networks. Based on the defense response, new SDN rules are generated. (4) The SecControl Node distributes the generated SDN rules to the corresponding SDN controllers for enforcement. After the four steps, the network devices behaviors in the SDN networks are modified based on the detected security threats.

In step one, security threat information is generated by various security tools. Consider existing security tools are divided into several types and each type of security tools is designed for a certain special security purpose, different security tools will be targeting different security threats. For example, firewalls focuses on filtering unwanted network traffic, while intrusion detection systems concentrate on detection system anomalies. It is very common to have different security event formats in different security tools. As a result, we need to customize ThreatCA for different security tools. And the security event information recorded by ThreatCA will be featured with various formats. To simplify SecControl Layer's work, ThreatCA is also responsible for preprocessing the recorded information and transform it into a uniform format in step one. It will be much easier for SecControl Layer to analyze the collected information with a unified format.

The collected threat information will be sent to SecControl Layer in step two. However, before sending threat information to the SecControl Node, ThreatCA has to extract effective information from the heterogeneous detection results, which can greatly simplify the work of the SecControl Node. To help reduce the workload of the SecControl Node, ThreatCA needs to be capable of recognizing alert information from various security tools. And it should be able to dismiss the format difference in detection results and extract effective threat characters based on the main functions of security tools. For example, a processed firewall alert could be (*firewall, network position, alert level, threat source, detection time, ...*). Actually, the preprocessing of threat information can be quite complicated. More details will be given in the design section.

Step three happens inside of the SecControl Node, where defense choices are made and SDN rules are produced. After receiving threat information from ThreatCA, SecControl Node will be analyzing security threats

information under given detection algorithms. Since different protection environments may have different sensitivities on security attacks, security engineers can equip SecControl Node with different security analysis and detection algorithms to decide whether the protected environment is facing attack threats or not. The security analysis and detection algorithms can be modified and replaced in different defense situations, which ensure the SecControl framework is able to provide a comprehensive and flexible protection on the target network environment. Based on the analysis result, SecControl can tell what defense responses can be made and how to generate the corresponding SDN rules to adjust the network behaviors. We summarize the defense reactions into several types. For each type of reaction, we transform the reaction into a set of SDN rules, which decide how we adjust network behaviors based on the threat information. The generated SDN rules will be distributed to corresponding SDN controllers in step four.

The last step is to distribute the generated SDN rules to the SDN Controller Layer. SecControl Layer will maintain a list which records the location information of all the controllers in the protected SDN networks. The SecControl Node will be aware of the controller layout in the protected SDN networks, which ensures the SDN rules can be sent to the related controllers. When the SDN rules are transmitted, the transmission process will be protected and secured. There will be a secure protocol between the SecControl Node and controllers to protect their communications.

## 4. SecControl Design

SecControl takes advantage of threat detection abilities of security tools to change network behaviors. Our design work focuses on bridging the gap between security tools and SDN controllers. Most questions raised in motivation part will be answered in this section.

### 4.1. SecControl Components

The SecControl framework is composed of four components, as shown in Figure 2. The first component is Threat Collecting Agent, which is running outside of the SecControl Node and responsible for collecting various security event information from security tools. The second one is Threat Analyser, which is in charge of converging and analyzing the collected threat information and decides corresponding defense responses. The third component is SDN Rule Engine, whose responsibility is transforming the generated defense responses to specific SDN rules. The last component, SDN Rule Distributer, is designed for distributing the SDN rules to SDN controllers (SDN

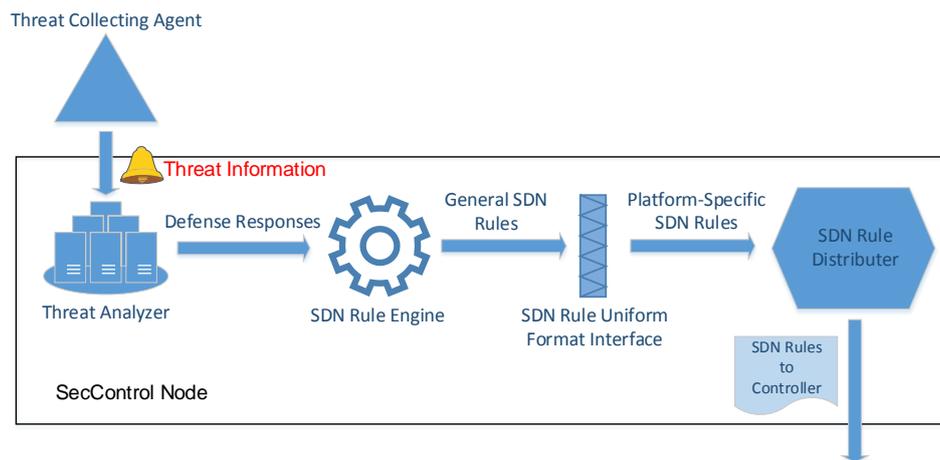


Figure 2. The SecControl Components

Rule Distributer should have a full picture of the SDN controllers).

**Threat Collecting Agent.** This is running outside of the SecControl Node. ThreatCA functions like a specially designed agent dealing with security tools. Its position should be close enough to the security tool it targets, in order to reduce the threat collection latency. The inputs of the ThreatCA are various detection results collected from different security tools, while the outputs of the ThreatCA are well-structured threat information, which can be directly used for other components. To complete the security threats SecControl can react, we need to collect as much security threats as we can through ThreatCAs. Consider existing security tools are separately targeting different threats, their detection results could be quite different. To handle different detection results, we need to provide each type of security tools at least one specialized ThreatCA, ensuring all the detection results can be handled.

The purpose of ThreatCA is to provide effective threats information to Threat Analyzer. As we mentioned in Section 3, traditional security tools are individually designed for different protection purposes, and their detection results could be very different. If ThreatCA sends raw detection results to Threat Analyzer directly, it is hard for Threat Analyzer to conduct security analysis efficiently. Threat Analyzer has to extract effective information from raw data first, then analyze the results, which will greatly reduce the performance of SecControl Node.

We design a preprocess function on ThreatCA. The preprocess function is responsible for transforming the raw detection results to a uniform format which can be used by Threat Analyzer for security analysis purpose. For each TreatCA, it is designed specially to understand the raw detection results of the security tool it is attached. To release the Threat Analyzer from

tedious format details, we present the detection results in a uniform format (the format is IDEMF [2]) so that Threat Analyzer can use the unified interface to deal with all detection results from different sources. All the ThreatCAs take different raw detection results and generate the processed detection results in the same format. Through this way, we can greatly reduce the workload for Threat Analyzer. There are several options for our choices. More details about the uniform format content will be given in Section 5.

**Threat Analyzer.** The preprocessed threat information will be sent to the Threat Analyser. The Threat Analyser will be analyzing threats, assessing security situations, and deciding defense responses. As a key component of SecControl, Threat Analyser provides strong supports for entire framework to deal with various security threats. To deal with various security threats, we are trying to design Threat Analyser a configurable, adaptable, and extendable module for different protection purposes. Security engineers are able to adjust defense strategies in Threat Analyser to practice different security analysis and detection algorithms.

In fact, analyzing threat information in a large number of detection records is quite complicated, and a lot of algorithms have been proposed [25, 30]. The analyzing results can be affected by many factors. For example, the same set of network traffic may lead to different alert information for using different defense strategies. In productive networks, people use different criterions to decide whether there is a security threat. Security engineers usually design customized security analysis algorithms for their Threat Analyzers. Consider the threat analysis algorithms depend on actual security policies, we will not provide concrete algorithms. Here, we introduce three common security analysis principles based on our experiences.

Many practical security analysis algorithms could be generated following these three principles.

**Time-based Threat Correlation:** As its name implies, time-based threat correlation analyzes detected threats according to their detection time-stamps. Time-stamp is an important attribute of a detected security event, which has an immediate relation to the happening time of the security event. Most time-driven attacks, like DDoS and network scan attacks, are identified by time intervals. Time-based threat correlation can help us detect time-related attacks happening on different targets. For example, computer worms may broadcast their propagation traffic in a very short time to reach as many nodes as they can. When a computer worm is attacking, we may experience a time period while network traffic is featured with multi-nodes to multi-nodes. We can detect these malicious activities with time-based threat correlation.

**Target-based Threat Correlation:** Target-based threat correlation studies threat records by a certain target. Here, a target could be a physical machine, a virtual machine, or even an operating system. To avoid being detected by security tools, some experienced attackers may cooperate to start an attack to a target simultaneously. For example, distributed network scans can hardly be detected by a firewall or an IDS due to the network scan is not finished by a single attacker. Target-based threat correlation can help us to dig all the detection results together to figure out potential attacks on one target.

**Experience-based Threat Correlation:** Experience-based Threat Correlation identifies security threats based on work experience of security engineers. In a real protection environment, a smart attacker may use complex strategies to compromise a target node, which can bypass the detection of most security tools. When the compromise is detected, security engineers will go to the audit information, correlate related information, and restore the attacking process reversely. The restoring process is based on the security engineers' experience. After time to time, security engineers are able to locate the suspicious events based on their experience. The experience of security engineers could be concluded as practical threat detection algorithms, which can provide more choices in real network protections.

When dealing with a real defense environment, these principles can be used together for detecting more advanced attacks. Except for the above principles, security engineers can design their own detection algorithms to detect a certain security threat. Many research has been done on correlation-based security analysis algorithms [11, 13, 38]. Security engineers can easily customize these algorithms and deploy them in SecControl Node for specified security threats. Once a security threat is identified, the

Threat Analyzer will choose a predefined defense response as a reaction against the security threat. In different protection scenarios, defense responses mean different actions. For example, on a firewall, a defense response could be blocking the threaten traffic; while on a host system, a defense response could be isolating a suspicious executable file. In SecControl, we focus on network level response, which means we adjust network behaviors through SDN technologies as defense response on potential attacks. Similarly, Threat Analyzer allows the defense responses be defined in a flexible way. Just like security analysis and detection algorithms, the defense responses should be configurable to fit different defense scenarios as well, and security engineers can dynamically map security threats to different defense responses.

In this paper, all the defense responses will be adjusting current behaviors of SDN networks in order to minimize the property loss and maximize the security benefits. How to choose a proper defense response, just like how to customize a Threat Analyzer, is also depending on security policies. We will give several examples in Section 6. Once a defense response is fixed, it will be sent to the SDN Rule Engine and then translated into basic SDN primitives.

**SDN Rule Engine.** SDN Rule Engine, as the name suggests, generates the corresponding SDN rules for the SecControl framework. Just as we mentioned previously, the defense response generated by ThreatCA will be focusing on adjusting network behaviors. Network behaviors will be adjusted through controllers over SDN network. Controllers send out SDN rules to individual network devices, where the SDN rules are enforced and the network behaviors are adjusted. In SecControl, SDN Rule Engine translates the defense responses into SDN rules. We design a systematic method to achieve the translation process through using SDN primitives, which stands for the basic network operations when dealing with security threats. We define five SDN primitives based on the network flow features. They are *Drop*, *Forward*, *Reflect*, *Isolate*, and *Copy*. The five SDN primitives can be used individually or in combination to deal with various security threats.

The five SDN primitives are as follows:

1. **Drop**, which means discarding the identified network traffic. This primitive is usually used to block unwanted network traffic.
2. **Forward**, which just tells the network devices to pass the identified traffic to its destination based on the existing SDN rules. When we do not want to do any operation on the identified network traffic for passing certain network device, we use forward.

3. **Reflect**, changes the destination of the identified network traffic both for inbound and outbound directions. For example, A wants to build up a connection with B. When A's connection traffic is reflected to C, A will be connected with C instead of B. After this, C will use B's network address and communicate with A, and A knows nothing about this. Reflect primitive can be used in deploying a shadow server or a honeypot.
4. **Isolate**, limits the identified traffic to a certain host or network area. When a node (or a node group) is identified as a source of an attack, we use this primitive to confine its network activities.
5. **Copy**, duplicates the identified packets, which is usually used for monitoring or logging use. Most current network devices have been equipped with this primitive. It could be used for real-time traffic analysis and other purposes.

The five SDN primitives can be used in combination, repeatedly, and in any sequence to form a wanted defense response. Each defense response will be translated into one or several SDN primitives. For example, a defense response may require directing the suspicious source to a honeynet, where the suspicious traffic will be recorded and analyzed. In this situation, the defense response will be translated into two SDN primitives, *reflect* and *isolate*. The suspicious traffic will be first reflected to a honeynet and then isolated in the honeynet area.

Usually, each SDN rule contains one SDN primitive, which represents the specific action of this rule. Some SDN primitives, like *drop* and *forward*, have been supported on most SDN platforms. For those SDN primitives that cannot be well supported, we may need additional translation processes to turn these primitives into corresponding SDN rules which can be executed on specific SDN platforms. For each SDN platform, we can design a set of interfaces for transforming SDN primitives to SDN rules. The generated SDN rules will be handled to the SDN Rule Distributer.

**SDN Rule Distributer.** The generated SDN rules will be sent to SDN controllers through the SDN Rule Distributer. SDN network employs a distributed structure which is formed with SDN controllers and network devices. In an SDN network, network devices are divided into groups and each group is connected and managed by a controller, and a controller cannot directly operate on a network device that is not connected to it. Since SecControl has no privileges to operate network devices directly, we need to distribute SDN rules to SDN controllers first, then have SDN controllers send SDN rules to correct network devices for execution.

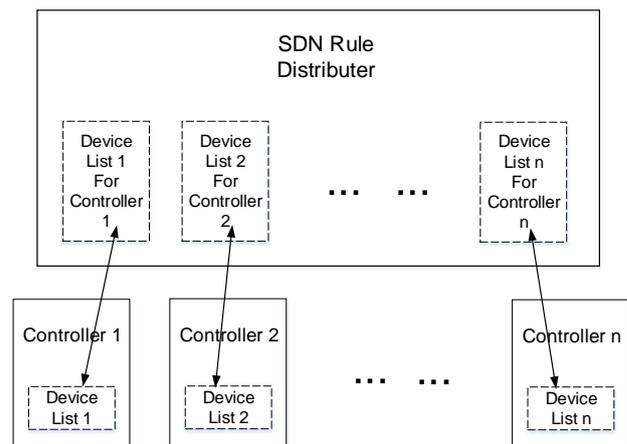


Figure 3. The SDN Rule Distributer

To ensure the SDN rules can be delivered to their destination correctly, the SDN Rule Distributer should have a full picture of the SDN networks. As can be seen in Figure 3, the SDN Rule Distributer stores a local copy of network device lists for all SDN controllers. Through the local copy of network device lists, the SDN Rule Distributer knows how to distribute SDN rules to correct SDN controllers.

Consider different network vendors may use different SDN rule formats on their SDN platforms, before sending out SDN rules, the SDN Rule Distributer may need an additional interface to transform SDN rules for a given SDN platform. In our prototype implementation, we use OpenFlow to build up SDN networks, and the defense responses are translated into OpenFlow flow rules. More details can be referred in Section 5.

We may need to consider another practical problem on SDN controller. Consider these SDN rules are generated with different criterions, there could be inconsistencies among these SDN rules. A lot of research has been done on how to update SDN rules on SDN controllers with consistencies [3, 20, 26]. Since how to consistently update SDN rules on controllers is another research problem and is not one of our contributions, we assume the inconsistency problem on SDN rules is well solved in our design.

## 4.2. Communication

To work as an integrated protection framework, the SecControl components need to cooperate and communicate with each other. We need to design inside communication mechanisms for SecControl as well.

Based on the workflow of SecControl, we need two communication mechanisms which reside in step two and step four separately. In step two, the Threat Collecting Agents need to communicate

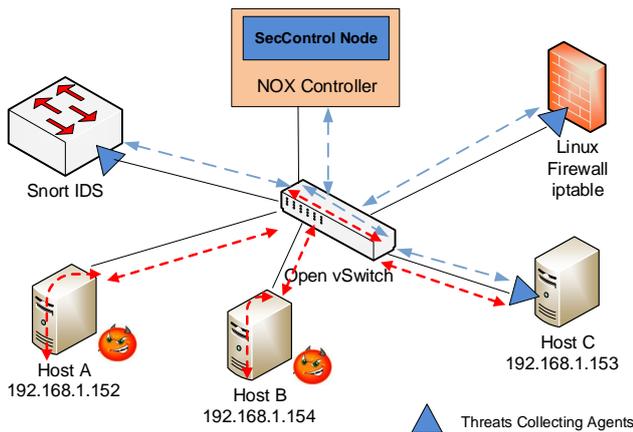


Figure 4. A SecControl Prototype

with the SecControl Node to send collected security event information, and that communication can be happening all the time. The other communication happens between the SecControl Node and SDN controllers, which serves to distribute SDN rules and maintain network devices information. Besides, we also need another communication mechanism among the SecControl Nodes, which enables the exchange of SDN rules between different SecControl Nodes.

We can achieve the step two communication like any typical network application by using TCP/IP protocols. The Threat Collecting Agents can send security events information over TCP or UDP protocol, which both can be used for network transportation purpose. The communication between the SecControl Node and SDN controllers is a little bit different. Except for distributing SDN rules, it is also used to synchronize network device information. Because it is related to device information update on SDN controller, it should be extended with existing SDN protocols. Similarly, the communication among SecControl Nodes can be implemented like any typical network application over TCP/IP. The implementation details will be given in Section 5.

### 5. A SecControl Prototype

We develop a prototype of the SecControl framework. For a proof of concept purpose, we implement both SecControl Node and SDN controller together. We chose to modify and extend an open source SDN controller, NOX [17], to finish all the related functions. Our implementation includes all the necessary functions for the SecControl components and is able to show the effectiveness of SecControl protections.

The SecControl Node is implemented on NOX version 0.9.0 with OpenFlow v1.0. NOX is an open source OpenFlow controller in C++/Python, which can be used to manage OpenFlow switches. We implemented the

```
<IDMEF-Message version="1.0">
<Alert ident="abc123456789">
  <Analyzer analyzerid="analyzer1">
    <Node category="dns">
      <location>HTTP Server</location>
      <name>host.domain.org</name>
    </Node>
  </Analyzer>

  <CreateTime ntpstamp="0xbc72b2b4.0x00000000">
    2020-05-19T15:31:00-08:00
  </CreateTime>

  <Source ident="abc01">
    <Node ident="abc01-01">
      <Address ident="abc01-02" category="ipv4-addr">
        <address>192.168.1.100</address>
      </Address>
    </Node>
  </Source>

  <Target ident="vic01">
    <Node ident="vic01-01" category="dns">
      <name>www.example.com</name>
      <Address ident="vic01-02" category="ipv4-addr">
        <address>192.168.1.50</address>
      </Address>
    </Node>
    <Service ident="vic01-03">
      <portlist>1-1024</portlist>
    </Service>
  </Target>

  <Classification origin="vendor-specific">
    <name>portscan</name>
    <url>http://www.vendor.com/portscan</url>
  </Classification>
</Alert>
</IDMEF-Message>
```

Figure 5. A Scan Detection in IDEMF

Threats Analyzer in Python and SDN Rule Engine in C++. The Threat Analyzer module is running as an OpenFlow application on NOX, while the SDN Rule Engine is inserted as an extension of NOX. We modified the built-in functions, `send_openflow_command` and `install_datapath_flow`, of NOX to implement the SDN Rule Distributer.

We pick three most used security tools for a demonstration purpose. They are Snort IDS, Linux iptables, and Linux system logs. Snort IDS is a popular open source IDS; Linux iptables is a kernel-supported firewall tool on Linux system; Linux system logs are native log system of Linux system which is often used for audit purposes. Each tool is attached a customized ThreatCA. Because the three tools use different alert formats, we implement three different ThreatCAs to collect security threat information. Besides, to simplify the protection, we categorize security events into attack events and suspicious events. The attack events should be reacted with a defense response instantly, while suspicious events need further analysis before deciding a defense response. When a ThreatCA meets an attack event, it just tags the event and sent it to Threat Analyzer to get an instant defense response. For the suspicious events, the ThreatCA extracts the critical

```

FlowAction generateOFActions(defenseResponse){
    FlowAction flowaction;
    switch (defenseResponse) {
        case drop:
            addAction(flowaction, drop);
        case forward:
            addAction(flowaction, forward);
        case reflect:
            addAction(flowaction, reflect);
        case isolate:
            addAction(flowaction, isolate);
        case copy:
            addAction(flowaction, copy);
    }
    return flowaction;
}

```

**Figure 6.** Translate Five SDN Primitives into OpenFlow Flow Actions

information of the events and put them in a unified format, Intrusion Detection Exchange Message Format (IDEMF) [2]. IDEMF provides a unified format and structure that allows the security detection results can be transferred among different parties. A scan detection involving three nodes can be demonstrated in IDEMF as shown in Figure 5.

The collected IDEMF messages are stored in a local DB for further analysis. If a defense response is determined, it will be translated into OpenFlow flow rules. In OpenFlow, each flow rule will have a set of attributes, such as *match field*, *counter*, *timeout*, *actions*, and so on, to match network flows. The *actions* field contains an action set, which indicates the operations to be executed for the matched network traffic. To enforce the SDN primitives at the OpenFlow switches, we translate the five SDN primitives into compatible OpenFlow actions. Figure 6 shows `generateOFActions()` function translating five SDN primitives to the OpenFlow flow rule actions. Finally, the new flow rules are sent to switch through function `install_datapath_flow(self, dp_id, attrs, idle_timeout, hard_timeout, actions, buffer_id, priority, inport, packet)`.

## 6. Prototype Evaluation

In this section, we evaluate the SecControl prototype with respect to effectiveness and extendibility. The evaluation testbed is deployed as shown in Figure 4. It is running on a desktop with an Intel Core i7-3370 3.4Ghz processor and 16GB RAM. We use KVM, Open vSwitch [1, 32], NOX [17], Linux firewall iptables, Snort IDS, and Linux built-in log system to construct a SecControl protected virtual network. The evaluation environment is built on a virtual network 192.168.1.0/24. The physical machine is running CentOS 6.0 with kernel 2.6.32 and qemu-kvm-0.15.1 for virtualization. The three hosts are running as guest OSes with CentOS 6.0 as well. As can be seen in

Figure 4, all the nodes are in a virtual network and connected by an Open vSwitch. We have security tools, Snort 2.9.7.5, iptables 1.4.7, and Linux Syslog systems, running at host machine. Each security tool is attached with a Threat Collecting Agent (each blue triangle in Figure 6 stands for a ThreatCA), and the ThreatCAs are communicating with the SecControl Node through the virtual network.

### 6.1. Effectiveness

We demonstrate the effectiveness of the SecControl framework with several security threats, regular scan threat, and payloads specific attacks. As Figure 4 shows, host A, and host B are attacking machines, and host C is the victim machine (for some attacking scenarios, we may deploy more attacker nodes). We use attacking machines to send out attack traffic to the victim machine.

**Regular Scan Threat.** Regular network scan is typically conducted by a single attacker to locate easy targets in an open network environment, like a public network. In our network environment, we assume an attacker owns host A 192.168.1.152, and he wants to sniff the network status of host C 192.168.1.153. We configure Snort with a scan detection rule: `alert tcp any any -> $HOME_NET any (msg: "TCP SYN"; flow: stateless; flags:S; detection_filter:track by_dst, count 100, seconds 5; sid:1000001; rev:1)`. We tag the detected scan threats as attack events and configure primitive *reflect* as default defense response to a scan threat. All the scan traffic for host C will be reflected to host B 192.168.1.154. We open port 22, 23, 25, 80, 111, and 443 on B, and 22, 25, and 111 on C. Figure 7 shows the reflecting process, from which we can see the scan results are from host B instead of host C. That is, the scan traffic is successfully reflected to B.

**An Attack with Specific Payloads.** When an attacker knows a specific vulnerability of a target machine, he can attack the target machine by sending a well-designed exploit. The attacking exploit sent through network packets is called malicious payloads. Malicious payloads can help the attacker take over the victim machine and gain an absolute control over it. We install an old Windows 2000 OS on host C 192.168.1.153 and open the vulnerable service SMB on port 445, which holds a dangerous vulnerability through which an attacker can easily obtain a remote shell with admin privileges. We configure the Snort to match the signature of the attacking payload `windows/vncinject/bind_tcp`. We choose *block* as the default defense response if any malicious payload is matched. Correspondingly, the *block* defense response is translated to primitive *drop* on the controller. We

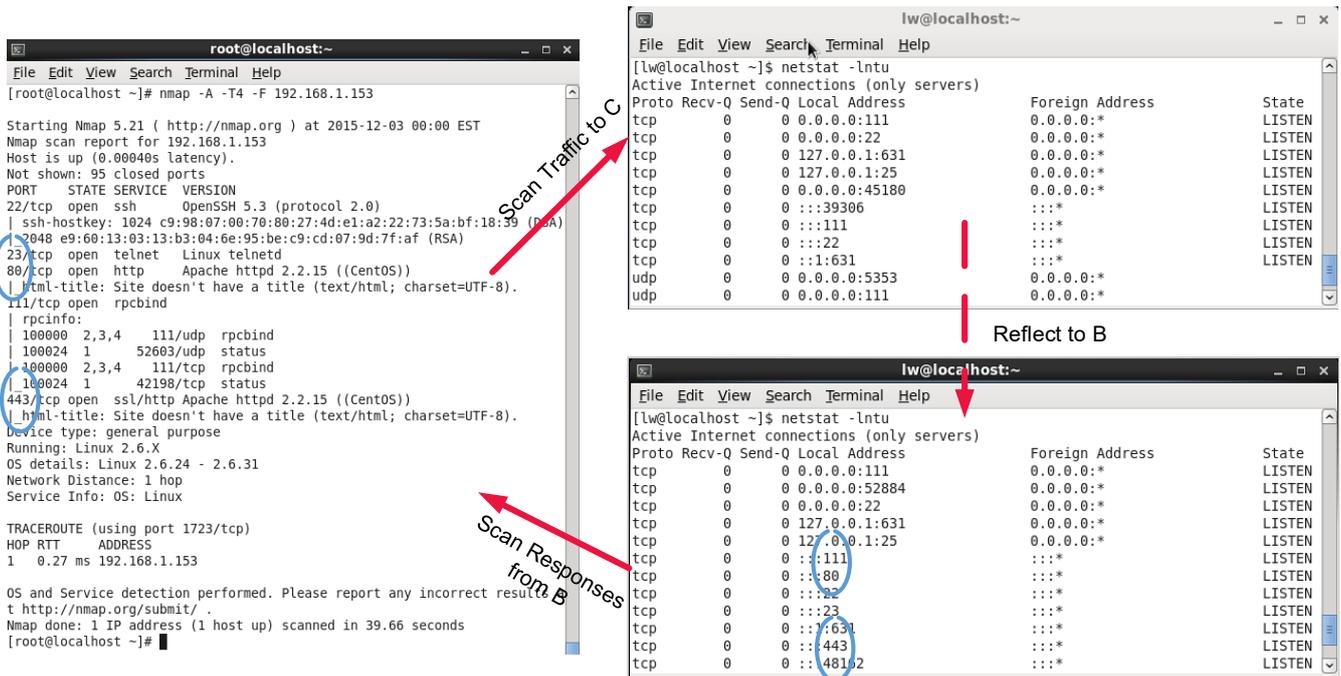


Figure 7. A Simple Network Scan

use host A 192.168.1.152 as the attacking machine. The attacking payload is sent with metasploit, a penetrating test tool. Figure 8 shows the metasploit console window. The result shows the exploit fails due to a connection timeout, which proves we successfully block the attacking traffic to host C.

### 6.2. Extensibility

We demonstrate the extensibility of the SecControl framework by using different security analysis principles. We use time-based threat correlation and target-based threat correlation to identify several advanced attacks, which usually may not be easily detected by existing security tools. And, we show the scalability of the SecControl framework by deploying multiple SecControl instances over different SDN networks, and our results show different SecControl instances can cooperate to offer protections across SDN networks.

**Distributed Scan Threat.** Distributed Scan is an advanced and hidden network scan, which is achieved by multiple scanning sources. Smart attackers can take multiple attacking sources to start a distributed scan, in order to bypass existing security tools. In this attacking scenario, we use host A and host B to start a distributed port scan on host C. Our target port range is 0-500. Host C is opening port 22, 25, and 111, while host D has port 22, 25, 80, 111, and 443 open (we add one more host D as a honeypot to communicated with the reflected scan traffic. Host D share the same configuration with host B). We choose *redirect* defense

response to deal with the distributed scan, and it is translated to *reflect* primitive. To detect the distributed scan threat, we extend the security analysis process of Threat Analyzer by following the target-based threat correlation principle. We configure Snort to record all the traffic. Figure 9 shows the results of distributed scan. From the scan result of A and B, we can see the port 80 and 443 is open, which shows D is the real scanned node and the distributed scan traffic is successfully reflected to D.

**Step-Stone Attack.** Step-stone attack is another advanced attack [7]. To reduce the risks of being detected, attackers choose to start an attack on step-stone nodes instead of his own machine. Step-stone nodes are immediate nodes taken by attackers. Through step-stone nodes, an attacker can get more accesses or conveniences in taking over the target node. Following the time-based threat correlation principle, we design a two step-stones attack detection algorithm. We use *redirect* and *block* as the defense response for the step-stone attack. In our defense, the attacker node will be blocked, and the step-stone node will be redirected to a honeypot. We use host A as the attacker's machine and host B as the step-stone to attack host C. As the attacking side on host A, we first open and login a shell remotely on host B, then we use B as a step-stone to send malicious payloads to host C. We record all the outside connections of host B, including the connection between A and B. We configure Snort to record the SSH connections between A and B. The remote login attempt is recorded

```

Terminal
File Edit View Terminal Tabs Help
RHOST          yes      The target address
RPORT 445      yes      Set the SMB service port

Payload information:
Space: 1024
Avoid: 7 characters

Description:
This module exploits a stack overflow in the LSASS service, this vulnerability was originally found by eEye. When re-exploiting a Windows XP system, you will need need to run this module twice. DCERPC request fragmentation can be performed by setting 'FragSize' parameter.

References:
http://www.securityfocus.com/bid/10108
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2003-0533
http://www.osvdb.org/5248
http://www.microsoft.com/technet/security/bulletin/MS04-011.msp
http://milw0rm.com/metasploit/36

msf exploit(ms04_011_lsass) > set RHOST 192.168.1.153
RHOST => 192.168.1.153
msf exploit(ms04_011_lsass) > exploit
[*] Started bind handler
[-] Exploit failed: The connection timed out (192.168.1.153:445).
[*] Exploit completed, but no session was created.
    
```

Figure 8. An Attack with Specific Payloads

```

root@localhost:/home/lw
File Edit View Search Terminal Help
[root@localhost lw]# nmap -p 0-250 192.168.1.153

Starting Nmap 5.21 ( http://nmap.org ) at 2016-05-21 20:40 EDT
mass dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.1.153
Host is up (0.000007s latency).
Not shown: 247 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
23/tcp    open  telnet
80/tcp    open  http
111/tcp   open  rpcbind
MAC Address: AE:23:34:45:56:68 (Unknown)

lw@localhost:/home/lw
File Edit View Search Terminal Help
[root@localhost lw]# ifconfig
eth6      Link encap:Ethernet  HWaddr AE:23:34:45:56:68
          inet addr:192.168.1.153  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::pac23:34ff:fe45:5668/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2596 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2514 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:137466 (134.2 KiB)  TX bytes:163757 (159.9 KiB)
          Interrupt:10 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:41 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4730 (4.6 KiB)  TX bytes:4730 (4.6 KiB)

root@localhost:/home/lw
File Edit View Search Terminal Help
[root@localhost lw]# nmap -p 250-500 192.168.1.153

Starting Nmap 5.21 ( http://nmap.org ) at 2016-05-21 20:40 EDT
mass dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.1.153
Host is up (0.00011s latency).
Not shown: 250 closed ports
PORT      STATE SERVICE
443/tcp   open  https
MAC Address: AE:23:34:45:56:68 (Unknown)

[root@localhost lw]# netstat -lntu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52884           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25           0.0.0.0:*               LISTEN
tcp        0      0 :::111                  :::*                    LISTEN
tcp        0      0 :::22                   :::*                    LISTEN
tcp        0      0 :::23                   :::*                    LISTEN
tcp        0      0 :::1631                  :::*                    LISTEN
tcp        0      0 :::48162                 :::*                    LISTEN
udp        0      0 0.0.0.0:5353           0.0.0.0:*
    
```

Figure 9. A Distributed Network Scan

in the system log of host B. Targeted by the detection algorithm, the SSH traffic is tagged as attack traffic. The results show SecControl detected the step-stone attack and the host B's traffic is successfully reflected to the honeypot node.

**Cooperations among SecControl Nodes.** We show the scalability of the SecControl framework with multiple SecControl deployments. We use two physical machines, and each physical machine is deployed with one SecControl instance. Two SecControl frameworks are running in two different virtual networks. We configure routing information of two virtual networks so that they

can communicate with each other. In our evaluation, we manually send a set of OpenFlow rules from one SecControl Node to the other, and the result shows the other SecControl Node can successfully receive and enforce the OpenFlow rules. However, there could be an information inconsistency problem when we have more different SecControl Nodes. In order to send SDN rules to the proper SecControl Node, every SecControl Node should have a full picture of all other SecControl Nodes' network positions and their network device lists. A lot of algorithms studied in distributed computing can be borrowed and used in this scenario. Consider this is not the focus of this paper, we will not elaborate further on this.

### 6.3. Overhead

SecControl is a practical network security solution aiming to provide a comprehensive protection for SDN networks. Since SecControl uses different strategies and algorithms to deal with different security threats, we can hardly find a unified method to evaluate its overall performance. We evaluate the time interval between a SecControl flow rule leaves NOX and the flow takes effect in the network. For the *forward* primitive, the time interval is 7.542 ms; for the *drop* primitive, the time interval is 13.152 ms; for the *reflect* primitive, the time interval is 17.684 ms. Besides, consider our evaluation testbed is deployed on one physical machine and all the involved nodes share the same set of physical resources, we should be able to shorten the time interval value if it is conducted on a more powerful machine.

## 7. Discussion

We discuss some limitations of the SecControl framework in this section. First, SecControl may have a delay reaction issue when providing defense responses. This is a common issue for many monitor-based security tools for there is always a delay between threat detection and defense reaction. Also, the network efficiency may affect the protection effect of SecControl. Consider security events are transmitted over network between the Threat Collecting Agents and the SecControl Node, the network transmission efficiency can affect SecControl's protection effect. In some protection scenarios, security engineers may require an instant response on a detected threat. A possible way to alleviate this issue is to build an exclusive network channel between the Threat Collecting Agents and SecControl Node. Further, to improve the performance of security event collecting, we may design built-in threat collecting interfaces on security tools.

Second, SecControl relies on existing security tools to gather security events and generate defense responses. We may face an accuracy issue because the accuracy of the security threat information is not exactly

guaranteed. Almost all mainstream security solutions follow a detection-based protection policy, and the protection is affected by detection accuracy. Consider the current detection algorithms are not perfect, the detection results may suffer false positive and false negative issues. Therefore, SecControl may produce inaccurate defense responses. One possible solution is to manually record the real attacks and pick up corresponding defense responses. We believe a lot of further research can be done on this issue.

Third, consider the SecControl framework relies on a distributed architecture, it may suffer all possible issues that can happen in a distributed network environment. For example, a potential issue is the single failure problem. If the SecControl Node is down, our protection will be discontinued. In fact, single failure and all other related issues have been well researched in the distributed system field. We can just take whatever comes to our protection scenarios and adopt these solutions.

## 8. Related Work

Security Incident and Event Management (SIEM) [29, 31] is a set of technologies which are used to gather, analyze and present information from network and security devices. SIEM is designed to collect security-related information from all kinds of devices and applications such as firewalls, IDS, antivirus, and so on. When an attack happens, security engineers will turn to SIEM for a complete record of that attack for security investigations and audits. SIEM mainly focuses on monitoring and tracing purposes. Compared with SecControl, although SIEM is capable of collecting and analyzing security threats, it does not provide interaction interfaces for the latest SDN networks.

OpenFlow is currently one of the most popular SDN protocols, which has been widely accepted both in academia and industry. It originates from a series of research on enterprise network management architecture, which includes SANE [10], Ethane [9], and ONIX [22]. SANE is proposed to solve enterprise network security problem by providing fine-grained network connection control. In a SANE protected enterprise network, all the network subjects by default are not allowed to visit any network services unless being authorized. The initial implementation of SANE relies on a network service directory, which maintains an access control list for all the registered network services. The restricted design and clean-slate implementation make SANE difficult to deploy and test. Ethane is a greatly improved enterprise network architecture based on SANE, which mainly extends SANE in three aspects. Ethane first recognizes network security as a subproblem of network management, and believes the security goal can be achieved by

defining better management policies; Ethane supports incremental deployment, and that makes it a more practical network management solution; Ethane has both software and hardware implementations which provide more hands-on experience for future SDN research. Following Ethane, ONIX is devised to provide a production-level network control paradigm. ONIX extends Ethane and broadens its control functions to network discovery, devices statistics collecting, and failure recovery mechanism. Different from Ethane, ONIX aims to break the stable architecture of the traditional network and build a new distributed network control plane. Furthermore, ONIX creatively suggests building network management applications over network control plane, which provides valuable references for the following SDN controllers.

SecControl can be regarded as a “controller” of the SDN controllers. It releases the security related computation logic from typical SDN controllers that should focus on managing low-level network devices. NOX [17] and POX [27] are two twin open source OpenFlow controllers implemented in C++ and Python respectively. They provide a set of APIs for upper-level network applications to dynamically change the flow tables of OpenFlow switches. However, the current OpenFlow structure is problematic and may meet some issues when deploying in a large scale network. Researchers propose different SDN controller solutions to fit existing controllers into large scale deployments, like HyperFlow [4], Pratyastha [23], DISCO [33], ElastiCon [14], and ONOS [6]. These methods enhance the existing controllers by adding more supports on scalability, device state synchronization, controller cooperation, fault tolerance, and other functions. Relying on SDN controllers, many network relevant applications have been innovated. Heller et al. [19] propose to reduce the energy consumptions by improving network infrastructures of data centers through centralized SDN controllers. Curtis et al. [12] suggest to use SDN controllers to optimize flow management to further achieve a better overall network performance.

SecControl combines traditional security tools and SDN technologies to provide a practical network security solution. For one hand, SecControl makes use of security processing abilities of existing tools; for the other hand, SecControl maximizes the security benefits of taking SDN technologies. Shin et al. propose FRESKO [35], a modular security application development framework for OpenFlow networks. FRESKO provides a fine-grained framework to implement security functions as OpenFlow applications. However, it requires security engineers to reimplement all security functions to fit FRESKO design, which brings a lot of engineering work. Besides, consider FRESKO is implemented at controller side, it is greatly confined

by the processing capabilities of the controller. As a result, the security functions requiring complicated computation and analysis can hardly be deployed with FRESKO. AVANT-GUARD [36] aims to improve the data plane performance in order to provide SDN security applications a more scalable and responsive OpenFlow infrastructure. It designs a *connection migrations* mechanism to improve OpenFlow's weak points and protect OpenFlow devices from saturation attacks. However, AVANT-GUARD does not change the fact that the SDN controller could be a potential bottleneck in security applications. Different from FRESKO and AVANT-GUARD, OpenFlow Extension Framework (OFX) [37] modifies the software system of network hardware devices to allow SDN applications dynamically load software modules. OFX achieves a good performance because it is running on switch hardware directly. However, not all security services can provide effective protections on a switch hardware. Compared with existing SDN security innovations, SecControl neither introduces heavy workload to SDN controller nor brings negative effects to existing security tools.

Except for the SDN security application frameworks, researchers also extended the individual security tools in SDN environments. FlowGuard [20] is designed to achieve a firewall running over SDN networks. FlowGuard is capable of checking suspicious network flows and verifying network-wide firewall policies. However, it just provides basic firewall functions and cannot be extended with other security functions. Similarly, some research modifies traditional intrusion detection systems to fit SDN environments. Mehdi et.al [28] suggest using SDN to solve home network security problems. They provide four prominent traffic anomaly detection algorithms to detect security threats on SDN controllers. This innovation provides an example of applying SDN technologies in home network security solution.

Some researchers also try to innovate security functions with Network Function Virtualization (NFV) [5]. Aaron et al. design OpenNF [16], a control plane architecture to enable the reallocation of flows within NF instances. Through OpenNF, network operators are able to create rich control applications, including firewall, NAT, traffic loadbalancer, and so on. OpenBox [8] is designed to decouple the control plane of middleboxes from their data planes and unify the data plane through service instances. It provides a set of interfaces and protocols to communicate with SDN controllers and middleboxes. OpenBox introduces a uniform platform for network admins to design network applications cross SDN network devices and middleboxes. Similarly, these NFV innovations focus on a universal network architecture for general network applications instead of security applications.

SecControl can be regarded as a “controller” of the SDN controllers. It releases the security related computation logic from typical SDN controllers that should focus on managing low-level network devices. NOX [17] and POX [27] are two twin open source OpenFlow controllers implemented in C++ and Python respectively. They provide a set of APIs for upper-level network applications to dynamically change the flow tables of OpenFlow switches. However, the current OpenFlow structure is problematic and may meet some issues when deploying in a large scale network. Researchers propose different SDN controller solutions to fit existing controllers into large scale deployments, like HyperFlow [4], Pratyastha [23], DISCO [33], ElastiCon [14], and ONOS [6]. These methods enhance the existing controllers by adding more supports on scalability, device state synchronization, controller cooperation, fault tolerance, and other functions. Relying on SDN controllers, many network relevant applications have been innovated. Heller et al. [19] propose to reduce the energy consumptions by improving network infrastructures of data centers through centralized SDN controllers. Curtis et al. [12] suggest using SDN controllers to optimize flow management to further achieve a better overall network performance.

## 9. Conclusion

In this paper, we propose a new network protection framework bridging the gap between existing security tools and SDN technologies, to produce a practical and comprehensive network security solution for SDN environments. SecControl integrates the capabilities of existing security tools and combines SDN controls to obtain an optimized SDN network security solution. We demonstrate the capability of SecControl by implementing a prototype with the OpenFlow protocol and evaluate its effectiveness and performance impacts with common security threats. Our experiments show that SecControl can cooperate with many mainstream security tools and provide effective defense responses over SDN-supported networks.

## References

- [1] Open vSwitch. <http://openvswitch.org/>.
- [2] RFC4765. The Intrusion Detection Exchange Message Format (IDEMF). <https://www.ietf.org/rfc/rfc4765.txt>.
- [3] E. Al-Shaer and S. Al-Haj. Flowchecker: configuration analysis and verification of federated openflow infrastructures. In *The 3rd ACM Workshop on Assurable and Usable Security Configuration, SafeConfig, Chicago, IL, USA, 2010*.
- [4] B. Balis. HyperFlow: A model of computation, programming approach and enactment engine for complex distributed workflows. *Future Comp. Syst.*, 2016.
- [5] J. Batalle, J. F. Riera, E. Escalona, and J. A. Garcia-Espin. On the implementation of nfv over an openflow infrastructure: Routing function virtualization. In *Future Networks and Services (SDN4FNS)*, pages 1–6. IEEE, 2013.
- [6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow, and G. M. Parulkar. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking, HotSDN, 2014*.
- [7] A. Blum, D. X. Song, and S. Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Recent Advances in Intrusion Detection: Proceedings of 7th International Symposium, RAID, France, 2004*.
- [8] A. Bremner-Barr, Y. Harchol, and D. Hay. Openbox: a software-defined framework for developing, deploying, and managing network functions. In *Proceedings of the 2016 conference on ACM SIGCOMM*. ACM, 2016.
- [9] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *SIGCOMM Review*, 2007.
- [10] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: a protection architecture for enterprise networks. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15, 2006*.
- [11] F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy, 2002*.
- [12] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2011*.
- [13] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection, Proceedings of 4th International Symposium, RAID Davis, CA, USA, 2001*.
- [14] A. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella. ElastiCon: an elastic distributed sdn controller. In *Proceedings of the 10th ACM/IEEE symposium on Architectures for networking and communications systems, 2014*.
- [15] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, APSIPA, 2012*.
- [16] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. Opennf: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review*, 2015.
- [17] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *Computer Comm. Rev.*, 2008.

- [18] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-Serve: Load-balancing web traffic using OpenFlow. *ACM Sigcomm Demo*, 2009.
- [19] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: Saving energy in data center networks. In *Proceedings of the 7th USENIX Symposium, NSDI*, 2010.
- [20] H. Hu, W. Han, G. Ahn, and Z. Zhao. FlowGuard: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking, HotSDN '14*, 2014.
- [21] H. Kim and N. Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 2013.
- [22] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Pou-tievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI*, 2010.
- [23] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson. Pratyastha: an efficient elastic distributed SDN control plane. In *Proceedings of the third workshop on Hot topics in software defined networking, HotSDN*, 2014.
- [24] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks. HotNets*, 2010.
- [25] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.
- [26] R. Mahajan and R. Wattenhofer. On consistent updates in software defined networks. In *Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII*, 2013.
- [27] J. Mccauley. POX: A Python-based OpenFlow controller. <http://www.noxrepo.org/pox/about-pox/>, 2014.
- [28] S. A. Mehdi, J. Khalid, and S. A. Khayam. Revisiting traffic anomaly detection using software defined networking. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection, RAID*, 2011.
- [29] D. Miller, S. Harris, A. Harper, S. VanDyke, and C. Blask. *Security information and event management (SIEM) implementation*. McGraw Hill Professional, 2010.
- [30] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *Network, IEEE*, 1994.
- [31] M. Nicolett and K. M. Kavanagh. Magic quadrant for security information and event management. *Gartner RAS Core Research Note (May 2009)*, 2011.
- [32] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending networking into the virtualization layer. In *Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII), HOTNETS*, 2009.
- [33] K. Phemius, M. Bouet, and J. Leguay. DISCO: distributed multi-domain SDN controllers. In *IEEE Network Operations and Management Symposium*, 2014.
- [34] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for OpenFlow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012.
- [35] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. FRESKO: Modular composable security services for software-defined networks. In *20th Annual Network and Distributed System Security Symposium, NDSS*, 2013.
- [36] S. Shin, V. Yegneswaran, P. A. Porras, and G. Gu. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2013.
- [37] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith. Enabling practical software-defined networking security applications with OFX. In *23th Annual Network and Distributed System Security Symposium, NDSS*, 2013.
- [38] F. Valeur, G. Vigna, C. Krügel, and R. A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Trans. Dependable Sec. Comput.*, 2004.
- [39] L. Wang and D. Wu. Seccontrol: Bridging the gap between security tools and sdn controllers. In *Security and Privacy in Communication Networks: SecureComm 2017 International Workshops, ATCS and SePrIoT, Niagara Falls, ON, Canada*. Springer, 2018.
- [40] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-based server load balancing gone wild. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE*, 2011.
- [41] H. Yin, X. Liu, G. Min, and C. Lin. Content delivery networks: a bridge between emerging applications and future IP networks. *IEEE Network*, 2010.