PivotWall: SDN-Based Information Flow Control

TJ OConnor
North Carolina State University
tjoconno@ncsu.edu

William Enck
North Carolina State University
whenck@ncsu.edu

W. Michael Petullo
United States Military Academy
mike@flyn.org

Akash Verma
North Carolina State University
averma3@ncsu.edu

ABSTRACT

Advanced Persistent Threats (APTs) commonly use stepping stone attacks that allow the adversary to move laterally undetected within an enterprise network towards a target. Existing network security techniques provide limited protection against such attacks, because they lack intra-network mediation and the context of information semantics. We propose PivotWall, a network security defense that extends information flow tracking on each host into network-level defenses. PivotWall uses a novel combination of information-flow tracking and Software Defined Networking (SDN) to detect a wide range of attacks used by advanced adversaries, including those that abuse both application- and network-layer protocols. It further enables a variety of attack responses including traffic steering, as well as advanced mechanisms for forensic analysis. We show that PivotWall incurs minimal impact on network throughput and latency for untainted traffic and less than 58% overhead for tainted traffic. As such, we demonstrate the utility of information flow tracking as a defense against advanced network-level attacks.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; **Distributed systems security**; **Information flow control**; *Access control*; • **Networks → Security protocols**; **Programmable networks**;

KEYWORDS

Software Defined Networking, Information Flow Control

ACM Reference Format:
TJ OConnor, William Enck, W. Michael Petullo, and Akash Verma. 2018. PivotWall: SDN-Based Information Flow Control. In *SOSR '18: SOSR '18: Symposium on SDN Research, March 28–29, 2018, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3185467.3185474

1 INTRODUCTION

The state of practice in network access control has remained largely static for decades. The policies that govern access are tedious to configure and prone to errors [65]. As a result, network-security administrators spend a great deal of time writing policy and finely tuning rules. Despite this effort, the result involves low-level semantics, lacks context, and most often is enforced at the dividing perimeter between networks. This leaves controls ill equipped to defend against attacks that effectively originate from within the network. Such attacks arise from bring-your-own-device environments, insecure wireless networks, and compromised web browsers.

Advancements in Software Defined Networking (SDN) have made network access control more dynamic. Ethane [10] advanced the concept of intra-network access control and evolved into the OpenFlow standard. SDN governs at host granularity while maintaining a centralized policy [28, 31, 54, 67], and this versatility gives rise to flexible and reactive defenses that consider more than the perimeter. For example, SDN and Network Function Virtualization (NFV) can enable per-host quarantines, provide moving target defenses, and route traffic through stricter enterprise controls. Bates et al. demonstrated an SDN-based forensic system capable of identifying a number of previously unobservable attacks [4]. Hardware switches and other devices from many manufacturers now support the OpenFlow specification; examples include devices from Hewlett-Packard, Cisco, and Pica8.

Operating systems security makes a distinction between access control and information-flow control. The latter is more context-sensitive because it governs not only who can access information but also what they can do with information once it is accessed. Prior research has considered information flow tracking in a distributed setting [3, 69]. However, more practical proposals [46, 47, 59] are limited to a cloud environment where there is tighter control over hosts and their communication. Pedigree [50, 51] explored extending taint tags to an enterprise setting, but sacrificed security by probabilistically removing taint information.

We seek to extend information flow tracking out from a host and through the entire network. Similar to Pedigree [50, 51], our goal is to reduce the semantic gap between host and network access controls, leading to security policies that better map to the governed activities. Specifically, we seek to extend information flow tracking as a defense against stepping stone attacks within enterprise networks. Such attacks evade traditional network defenses by compromising a series of hosts within a network, repeatedly pivoting laterally towards the final target. Of key importance, tracking flows between and within hosts enhances both real-time defense and post-incident forensics.

In this paper, we propose PivotWall, a novel network security defense that extends information-flow tracking on each host into network-level defenses. PivotWall identifies and defends against

ACM ISBN 978-1-4503-5664-0/18/03…$15.00
https://doi.org/10.1145/3185467.3185474

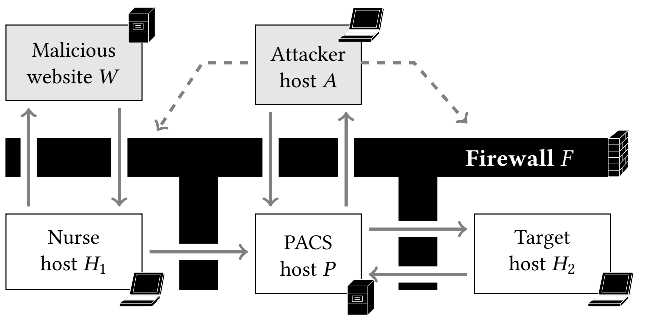


Figure 1: Scenario based on 2016 breach of hospital network

previously unobservable attacks through a novel combination of information-flow tracking and SDN's centralized management and intra-network controls. Our evaluation shows that PivotWall can detect a wide range of attacks used by advanced adversaries, including those that abuse both application- and network-layer protocols. Furthermore, we show that PivotWall provides this protection while incurring minimal impact on network throughput and latency for untainted traffic and less than 58% overhead for tainted traffic.

This paper makes the following contributions:

- *We extend information-flow control into the network using SDN.* PivotWall extends information-flow control beyond the boundary of the host.
- *We propose a policy language for practically specifying information flow control within an enterprise network.* PivotWall's policy language syntax builds on the popular Snort IDS syntax and introduces unique actions that leverage the benefit of SDN technology.
- *We prevent attacks that bypass traditional enterprise defenses.* We demonstrate that moving the information-flow reference monitor to the SDN/NFV increases the context available to defenders during stepping-stone and other elaborate attacks.

The remainder of this paper proceeds as follows: Section 2 motivates our work using a recent real-world example. Section 3 overviews the PivotWall architecture. Section 4 describes its design. Section 5 evaluates accuracy and performance overhead. Section 6 discusses limitations. Section 7 provides an overview of related work. Section 8 concludes.

2 MOTIVATION

PivotWall is motivated by advanced persistent threats (APTs) that use stepping stone attacks within an enterprise network. Such attacks compromise an initial host and then move laterally within the network, evading traditional network defenses. In this section, we provide the necessary background and intuition behind stepping stone attacks through a motivating example. The section concludes with a threat model for the PivotWall design.

2.1 Motivating Example

To illustrate the problem of stepping-stone attacks, we present a scenario based on a recent security breach in the healthcare industry. TrapX Research Labs highlighted the breach in a 2016 report [58]. The victim (a hospital) used traditional enterprise defenses including a standard firewall, a heuristic-based intrusion detection system, endpoint security, and anti-virus software. Despite these security measures, attackers stole confidential data.

Figure 1 depicts a generalization of the compromised network. The figure includes five hosts and a firewall. The lower hosts are part of the hospital network, and the upper hosts are outside of the network and controlled by the attacker. We depict traffic as directed edges; dashed edges represent traffic blocked by the firewall.

This attack exemplifies a stepping-stone attack. First, the attackers compromised the web browser on a vulnerable workstation H_1 after the user of that workstation visited a malicious website W. Next, the attacker used H_1 to compromise a picture and archive communications system (PACS) P. Ultimately, the attacker compromises workstation H_2 and gains access to confidential data which he exfiltrates via P.

The firewall, F, in the scenario was most likely configured to prevent a direct flow from the Internet to H_1 and H_2. Yet P is less protected, because it facilitates the movement of medical imagery such as X-rays throughout the hospital and its off-site offices. Thus information can freely flow between A and P

The difficulty arises because F cannot distinguish data that should flow between P and the Internet from data that should not. The context required for this decision is only available by examining a network information-flow control (NIFC) graph that spans both the activity within P and H_2 as well as messages between hosts. Only with this information could F block the confidential traffic before it flows between P and A while permitting benign interaction between P and other hosts. Put another way, information should be able to flow between H_2 and P or between an Internet host and P, but not from H_2 to a host on the Internet by way of P.

In summary, traditional approaches lack the knowledge gained from a NIFC graph containing the flow of confidential access and data throughout the network, whether between or inside hosts. This attack can be stopped by applying data labels and implementing flow control across host boundaries. We capture this conceptual approach in PivotWall, a novel enterprise security architecture that combines the logically central placement of the SDN with the context of host-based information flow tracking.

2.2 Threat Model and Assumptions

Our threat model assumes the attacker's goal is to obtain confidential information. To achieve this goal, attackers must evade intrusion detection systems and bypass network- and host-based access controls. To achieve this result, we assume the attacker will use stealthy strategies that abuse trust by laterally pivoting through blind spots in the network [67]. For example, an external attacker might pivot through an employee workstation to launch an internal attack that avoids the enterprise controls at the perimeter of the network. Similarly, an inside attacker might exfiltrate confidential data from a protected service to a globally-accessible server by routing through blind spots. An attacker might create covert or stealthy channels by abusing legitimate protocols including application layer protocols (e.g, Gmail, Slack, Twitter [2]) and network layer protocols (e.g., TCP, ICMP, DNS [44, 53, 68].) An attacker might use tools such as the Data Exfiltration Toolkit [2] or DNSCat2 [9] to create these

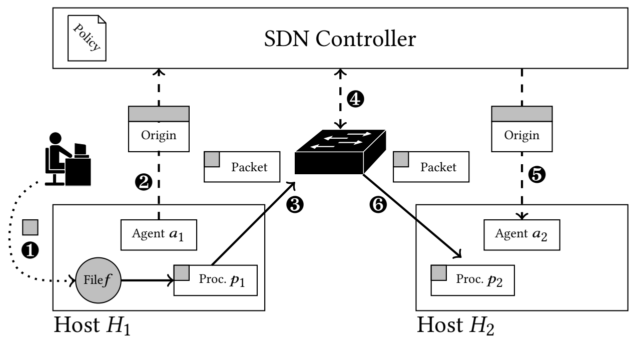


Figure 2: Overview: PivotWall Information Flow Control

channels. The goal of PivotWall is to prevent these and other stealthy attacks that abuse blind spots in defenses.

Our trusted computing base (TCB) includes the SDN security application, the network data plane devices, and each host's core operating system (i.e., kernel, core system daemons, and our host-agent). We do not protect against malicious-but-trusted administrators that can detect the host-agent, remove confidentiality labels on data and processes, or disable the host-agent packet labeling implementation. Similarly, we assume networking devices are not compromised to deactivate defense mechanisms. The TCB extends to other SDN applications running on the controller that are not segregated from the PivotWall security application. We consider the attestation of the host-agent, the security application, and enterprise networking devices as important but orthogonal problems. We leave attestation as a deployment task.

3 OVERVIEW

We designed PivotWall to extend information flow tracking across hosts in a distributed environment. PivotWall broadens the enforcement of secrecy by establishing information-flow controls at the SDN controller. This section describes PivotWall's architecture, addresses the identified challenges in centralizing information-flow controls, and discusses the key ideas of PivotWall.

3.1 Architecture Overview

The architecture of PivotWall involves three components:

Host agent: Each host on a PivotWall network is governed by a modified SimpleFlow [30] kernel. SimpleFlow tracks the flow of confidential information on a host. SimpleFlow labels packets which emanate from processes that might have read confidential information, and it taints processes that read a packet bearing a confidential label. The PivotWall host agent maintains provenance and sends control messages containing the origin to the controller as described in Section 4.2.

Network control plane and SDN controller: A lightweight *Pox* OpenFlow security application creates the necessary network flow modifications that deliver flows bearing confidential packets to the control plane for inspection. At the control plane, the security application implements the policy store, network information flow control (NIFC) graph, and reference monitor. The security application uses these components to inspect flows for a violation, and it implements eight primitive actions for handling confidential flows. These actions include unique methods for redirecting, throttling, or modifying confidential flows as described in Section 4.1.

Network data plane: The network hardware implements the OpenFlow flow modifications to deliver labeled flows to the control plane for inspection. The network hardware modifies flows based on the instructions from the SDN controller.

Figure 2 summarizes the PivotWall architecture. An administrator has labelled the file f on host H_1 as confidential (❶). When process p_1 reads from this file, SimpleFlow taints process p_1. As process p_1 establishes a network flow from $H_1 \rightarrow H_2$, the PivotWall agent on host H_1 notifies the SDN controller of the unique origin label for the flow (❷). Subsequently, process p_1 writes the confidential information to the network in the form of a labelled packet (❸). Upon receiving this packet, the switch observes that it is labeled and thus queries the SDN controller (❹). The SDN controller acts as a reference monitor and examines the confidentiality, origin, and steering labels against indexed network information flow control (NIFC) graphs. The controller governs the return, mutation, dropping, or redirection of packets based on its configured policy. Here the SDN controller notifies host H_2 of the source of the packet (❺), and it delivers the packet (❻). Finally, SimpleFlow taints process p_2.

3.2 Challenges

PivotWall controls the flow of information in a distributed network but is governed by a centralized policy. Practical information flow tracking in a distributed environment requires overcoming the following challenges:

Practical policy enforcement: Precision is a challenge for information flow tracking in a network environment. A lack of accuracy can cause label propagation to fail, violating secrecy. Conversely, coarse precision can cause false positives leading to *taint explosion.*

Label Integrity: While information flow tracking within hosts is well studied, tracking information flows across networks has been limited to statistical measures that break down under even normal operations [12, 48, 56]. Statistical correlation approaches fail when data is compressed, encrypted, or delayed at the host. Furthermore, broadcasting the mandatory protection state of data does not scale.

Attribution: Determining the origin of a policy violation is challenging in a distributed environment. An attacker can take several intermediary steps on the host and the network to conceal their origin. Thus, implementing the *intent* of the policy is challenging without understanding the origin of the data encapsulated in a flow.

3.3 Key Ideas in PivotWall

PivotWall addresses the aforementioned challenges through the following key design concepts:

Network Reference Monitor: To achieve dynamic taint analysis at the network layer, PivotWall extends the classical host-centric reference monitor to establish a network access control enforcement mechanism. Under PivotWall, each network device modifies confidential flows to first pass through the SDN controller application. There PivotWall's reference monitor determines whether or how traffic may flow through the network device. While a typical reference monitor returns a binary response, PivotWall offers a range of responses. The reference monitor only evaluates traffic marked as confidential; non-confidential traffic is not evaluated.

Practical Policy Grammar: A single policy governs the reference monitor's decisions. PivotWall provides the three necessary properties of dynamic taint tracking—namely, *taint sources*, *taint sinks* and *taint propagation rules*—with a policy grammar focused on performance, simplicity, and flexibility. The grammar consists of a series of clear and concise *Snort-like* rules.

PivotWall's policy language expresses *taint sources* as the input source where confidential data entered the network. A taint source can represent a single host, a subnet of hosts, or any hosts in the exclusion of a host or subnet. PivotWall's policy represents *taint sinks* using the same syntax and describes the location where PivotWall applies actions outlined by the rule. Finally, PivotWall defines *taint propagation* rules, describing which action must be applied when data from a particular taint source reaches a taint sink. This key contribution by PivotWall allows a trusted administrator to express practical policies for reactive, fine-grained-modification on a per-flow basis. Rules include typical intrusion detection and prevention systems actions, such as the ability to *pass*, *log*, *drop*, *reject*, or *alert* on network flows. Furthermore, the use of SDN allows the ability to *redirect*, *throttle*, and *rewrite* network flows. Section 4.1 expands upon the policy language design and the implementation for reactive actions. Reactive taint propagation rules provides a flexible means of mitigating a wide variety of attack vectors against confidential data and processes.

Persistent Labels: Information flow tracking across distributed hosts can fail when data is transformed—intentionally or unintentionally—to remove confidential labels. PivotWall overcomes this limitation by establishing distributed persistent labels that seamlessly transfer between the host and the network layer. Through distributed persistent labeling, PivotWall establishes a mandatory protection system complete with labeling, protection, and transition states that cross over the boundary of the host and network. In contrast to traditional MLS models [6, 14], PivotWall adopts an approach an approach similar to taint tracking systems such as TaintDroid [15] where the label indicates if the data contains confidential information of a specific type.

PivotWall tracks information flow within each host on its network using SimpleFlow [30], an information-flow-based access-control system built within the Linux kernel. SimpleFlow permits a trusted administrator to label system objects—such as files, sockets, or pseudo-terminals—as confidential. Processes on a SimpleFlow host that read from confidential objects become tainted, and this taint status follows the writes and reads that result in the flow of confidential data through the system. For example, a process that reads from a pipe shared with a tainted process will itself become tainted. Under SimpleFlow, the Linux kernel will mark any packet written to the network by a tainted process. Processes that read from marked network packets also become tainted. SimpleFlow covers a wide range of system calls [30, §4.1.2], and addresses some of the high-bandwidth covert channels found in Unix [30, §4.1.3].

PivotWall extends SimpleFlow so that its network filter component further labels traffic with a network taint byte.[1] The network taint byte contains information used by PivotWall's SDN controller to steer the packet through a software-defined network. Combining SimpleFlow with the SDN-based steering label allows labels to persist across in-host and network communication and thus extends information-flow tracking to the network. It also presents the opportunity to create increased expressibility in policy language as we examine in the following section.

Network Information Flow Control (NIFC) Graph: In a simple classical access enforcement mechanism, data is labeled with a single bit which represents notion such as confidentiality or trusted. The reference monitor interface is responsible for querying the policy store to authorize a request. This binary label presents a challenge when implementing a practical policy enforcement mechanism that queries characteristics such as the origin of the confidential data. Practical policy enforcement requires the ability to examine the entire path of a confidentially-labeled object. PivotWall stores the flow of a confidentially-labeled object in a directed graph $G = (V, E)$. The vertices, V, of the NIFC graph represent the union of the set of subjects and objects that have interacted with a single confidentially-labeled object. The set of edges E, represent the flow of information from data in a protection state to a new object. These edges represent network flows to a new host. Representing the confidential data flow as an NIFC graph allows PivotWall to offer heuristics about how particular confidential data violated a policy. Although a simple scenario may result in a single path, complex scenarios may include an attacker attempting multiple data ex-filtration points. Representing the flow of confidential data as an NIFC offers visibility of the extent of the attack and key articulation points that extend bridges to the network border. This insight allows an administer to apply tighter controls on the hosts discovered as articulation points. Section 4.1 expands upon the design of the NIFC graph and the heuristics for constructing the complete path of confidential data prior to the policy violation.

4 PIVOTWALL

The goal of PivotWall is to protect the secrecy of data or processes by extending information flow tracking to a distributed architecture. We focus on establishing a logically central SDN controller to manage distributed information flow control through a reference monitor, policy store, and a network information flow control (NIFC) graph. A key idea in PivotWall is to maintain a NIFC graph that maintains the provenance propagation of confidential objects. By establishing a NIFC graph, we provide the ability for a system administrator to write rules to prevent attacks that span across multiple connected flows.

4.1 Network Layer Design

SDN eases network administration by combining a centralized policy with distributed enforcement. OpenFlow-enabled hardware consults a centralized controller to determine how to handle the

[1]The original SimpleFlow design blocked tainted packets at the subnet boundary.

flows it processes. In the case of PivotWall, this controller application implements a variation of the classical reference monitor to govern network flows. Whereas a classical reference monitor can only permit or deny, PivotWall can leverage SDN technology to throttle, modify, drop, reject, or redirect network flows. An administrator defines these actions by writing a policy, and PivotWall's controller application enforces this policy.

4.1.1 Network Information Flow Control Graph. A key insight of PivotWall is its use of a provenance graph to reduce or limit false positives and taint explosion. In contrast, Pedigree [50, 51] labeled confidential information with a taint tag, which can lead to taint explosion as the network becomes strongly connected. Pedigree addressed taint explosion by probablastically removing taint bits; however, this design choice sacrifices security.

PivotWall addresses this challenge by maintaining graphs that represent each information flow with a confidential system object (e.g., file, pipe, socket) as its source. This leads to more precise tracking while revealing many sophisticated attacks.

The NIFC is a directed graph $G = (V, E)$ containing the set of vertices $V = \{v1, v2, ...\}$ and set of edges $E = \{e1, e2, ...\}$. The graph represents the flow of a confidential object throughout the network. Each vertex v represents a single host or server that has processed, accessed, or stored a confidential object. Each directed edge e represents the network flows between hosts that have carried or provided access to the confidential object. Algorithm 1 depicts how PivotWall adds vertices and edges to a NIFC. First PivotWall checks the policy to determine if the packet source, destination, and protocol are permitted by the policy (1). The algorithm records the original source and creates a unique graph for a previously unseen confidential object (2 – 4). The algorithm next checks if the policy permits the path an object has taken by comparing the original source against the packet destination (6). If the policy permits this path, then a directed edge is added from the source and destination in the packet header (7), a per-flow rule is installed (8), and the packet is forwarded to the destination (9). In the case that the policy does not permit the action, the packet is dropped (11) or modified in accordance with the policy. We depict an example graph in the top right hand corner of Figure 3.

Figure 3 overlays the NIFC graph on the scenario in Figure 1, depicting the hosts W, A, H_1, P, and H_2 as rectangles. Overlaid upon this is a graph: processes s, b, p, and c (dotted rectangles) and file f (dashed circle) are the graph's nodes, and the flow of information makes up its directed edges. Also depicted as circles are six sockets S_1–S_6 that facilitate network communication.

An attacker compromises host H_1 after browser process b visits a malicious website (❶). The payload makes a connection to host P and compromises the PACS server process p (❷). Process p then compromises the PACS client process c on host H_2 (❸). The compromised process c reads the confidential file f (❹) and transmits it over the network to host P (❺). Host P then attempts to exfiltrate the file under the cover of a DNS query to host A (❻). Sensing that this was thwarted by PivotWall's SDN controller, the illicit software transmits the confidential information back to host H_1 (❼), but the SDN controller again blocks the exfiltration attempt (❽).

The directed graphs maintained by PivotWall inform the SDN controller as it governs the network. A graph indicates a tainted

Algorithm 1: HandleTaintedPkt

```
   Input: A tainted packet (p), and the UUID (u)
1  if $PolicyPermitsPacket(p)$ then
2      if $u$ not in $taintsTable$ then
3          $taintsTable[u] \leftarrow [p.src, p.sport]$
4          $graphTable[u] \leftarrow new\ Directed\ Graph()$
5      if $PolicyPermitsPath(p, u)$ then
6          if $edge(p.src, p.dst)$ not in $graphTable[u].edges$ then
7              $graphTable[u].add\_edge(p.src, p.dst)$
8          $installFlowRule(p)$
9          $send(p)$
10         return
11 $drop(p)$
12 return
```

flow, and it provides context into how the confidential information passed through the network. The graphs exist in a hash table that is indexed by the UUID of the confidential source object (described in Section 4.2). In the example here, the solid black edges represent the graph which PivotWall would correspond with the source f. The dashed gray edges represent incidental communication. Section 4.2 describes how PivotWall bridges between host-based information-flow tracking (e.g., tracking a `read` of f) and network-based information-flow tracking.

Storing the provenance history in a directed graph allows for a more expressive rule syntax for defining network policy. The presence of the information-flow graph means that the rules that govern network flows can consider the origin of a flow. For example, the denial at ❻ could realize the confidential source and redirect the DNS request to a sinkhole which would aid in a forensic analysis. The denial at ❽ could reduce the throughput of the exfiltration channel to the point of being useless. Very similar flows which do not contain illicit information could be allowed to pass.

4.1.2 Customizable Rules. A key benefit of PivotWall is its ability to consider the provenance propagation of confidential objects when dynamically analyzing flows. To achieve this in a flexible way, PivotWall provides a syntax for specifying the rules that govern flows. Unlike traditional intrusion detection systems, PivotWall examines the entire path of confidential flows when matching a rule pattern. Upon a successful match, PivotWall can respond with the eight base actions depicted in Table 1.

The syntax that specifies the most basic rules resembles Snort and references a flow's protocol, direction, and port to determine an action [52]. For example, Listing 1 prevents confidential data originating from `HOST1` from egressing outside the local area network by dropping packets. This rule implements data loss prevention with full information flow context to prevent an internal attacker at `HOST1` from ex-filtrating confidential data using stepping stones inside the network.

Similar to Snort, PivotWall interprets option fields inside parenthesis. The rule in Listing 2 redirects confidential traffic to a honeypot based on the redirect destination option (`rdst`) specified in the rule. As seen in both examples, PivotWall offers matching

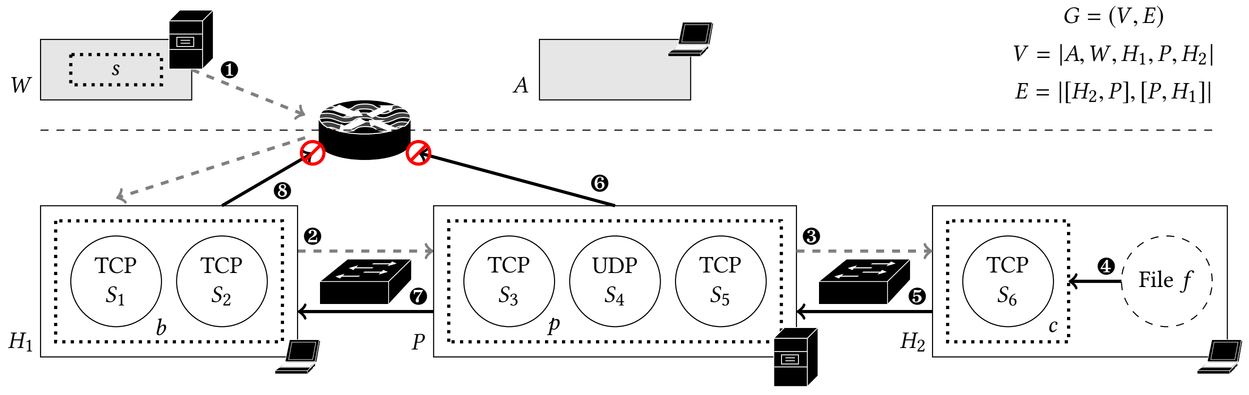


Figure 3: Example Actions that Build an NIFC Graph

Listing 1: Simple PivotWall rule preventing exfiltration of data from a specific host

```
$HOST1  = 10.1.1.1
$HOME   = 10.1.1.0/24
drop tcp $HOST1 any -> !$HOME any
```

Listing 2: Rule For Redirecting Traffic to a HoneyPot

```
$HONEY = 10.1.1.4
redirect tcp $HOST1 any -> !$HOME any
    (rdst=$HONEY)
```

Table 1: Customizable Actions for PivotWall

Action	Description
pass	Permit confidential flow; keep confidential labels intact
untag	Permit confidential flow; remove confidential labels
log	Log packets from confidential flow to pcap
drop	Drop packets from confidential flow and log
reject	Drop packets from flow, log, and send TCP Reset of ICMP unreachable
redirect	Redirect flow using alternate destination or source
slow	Rate throttle packets using TCP congestion window or queuing
modify	Permit flow but rewrite packets based on customizable script

a single IP address, host name, or CIDR notation. Variables can be used to easily construct rules. The '!' operator can be used to denote traffic outside of a particular network. Each rule begins with a unique action to handle the violation created by the flow.

Managing the defense at an SDN controller introduces several unique and different response methods. Since the controller manages the data plane of the network, the controller has the capability to communicate with all enterprise network devices in response to an attack. By sending control messages across the data plane, the SDN controller can near instantaneously communicate and propagate knowledge of a threat to all managed enterprise network devices. PivotWall offers typical basic actions for responding to violations as well as custom-tailored advanced actions.

Basic Actions: PivotWall provides five primitive actions to implement security directives to implement information flow tracking. The *log, drop,* and *reject* actions provide similar functionality to their Snort counterpart actions and are described in Table 1. Because the *pass* and *untag* options interact with the labels, they require further discussion. *Pass* permits confidential flows and leaves the network, host, and origin labels intact on the packet headers of the flow. In contrast, *untag* permits the confidential flow, removing the network, host, and origin labels from packets in the flow. Essentially, the pass action propagates labels to the next hop while the untag action removes the confidentiality labels from the flow.

Advanced Actions: To specifically address secrecy and integrity attacks, PivotWall implements three advanced action primitives for redirecting, throttling, isolating, and modifying flows. SDN-enabled traffic redirection has shown promise as a means of mitigating threats by redirecting traffic to a honeypot, sink-hole, or hiding critical resources [22, 40, 54, 55].

Slow and *modify* are actions that shape traffic and mitigate secrecy and integrity attacks. *Slow* throttles the flow by artificially queuing the flow of IP packets. When possible, the *slow* action throttles the transport layer protocol by rewriting packets to reduce the TCP Congestion Window Size. Reducing the throughput of transport layer protocols degrades the effectiveness of the attack channel thus slowing the propagation of an attack. Essentially, the *Slow* action uses the benefit of SDN technology to implement the concept of a *tarpit* [66]. By keeping the attacker channel open but unusable, an analyst can investigate an attack in real-time.

The *modify* action reduces the impact of false positives while not decreasing true positives. Consider the specific cases where an administrator may write a rule that has a high false positive rate (e.g., HTTP, DNS, ICMP traffic). The *modify* action permits the administrator to allow the flow of confidential data but eliminates optional fields that might carry the confidential data. To achieve this, the administrator defines the handling of matched packets with a custom *modify*-script.

To illustrate the utility of the *modify* action, consider a simple use case of ICMP covert traffic. Daemon9 [13] proposed using `ICMP Echo Requests` to carry confidential data in 1996 by embedding data in the an `ICMP Echo Request` payload. Several common data exfiltration tools still use this technique [27]. Listing 3 provides a

Listing 3: Modify-Script for Mitigating Covert ICMP

```
rewrite_ping_req(packet):
  ipkt=packet.find("icmp")
  if (ipkt):
    if (ipkt.type == ICMP.TYPE_ECHO_REQUEST):
      ipkt.payload = "A"*len(ipkt.payload)
      packet.payload = ipkt
  return packet
```

PivotWall *modify*-script that removes covert data from the payload of an `ICMP Echo Request` by rewriting the payload field entirely with the letter *A*.

PivotWall *modify*-scripts provide the customization to address secrecy attacks while limiting the impact on benign traffic on the protocol flows. Simply, PivotWall *modify*-scripts define how packets of a confidential flow are modified to remove covert channels.

4.2 Host Design

In order to classify packets that are sent from a host, PivotWall must track flows within that host. Intra-host flows and network flows are inherently different. Flows within a host result primarily from software that loads, generates, transforms, or stores information. In contrast, network flows have more to do with transporting information from one host to another. While information might change in the network, manipulation is generally performed merely to allow information to traverse the network.

Intra-host flows result from user activities and therefore model user intentions and the effect they have on confidential information within the host. Such flows provide links between processes, and they most commonly result from processes invoking system calls such as opens, reads, and writes. One process might concatenate two files to a third, another might read information from the network and write it to a file, and yet another might communicate with a peer process using a pipe. In any case, these flows have the potential of moving confidential information from file to file, from network socket to file, or from process to process through a pipe.

Host Flow Tracking: Whole-system provenance, or the tracking of confidential data throughout the host, is a challenging problem. Several promising solutions rely on the Linux Provenance Module (LPM) interface to gain whole-system provenance and handle side channel attacks [5, 30]. PivotWall builds upon SimpleFlow [30] to track the host information flow. As a Linux kernel modification based on the Linux Security Module (LSM) interface, SimpleFlow removes many of the race conditions present in user-space monitoring tools or tools that combine a provenance engine with a separate access-control mechanism.

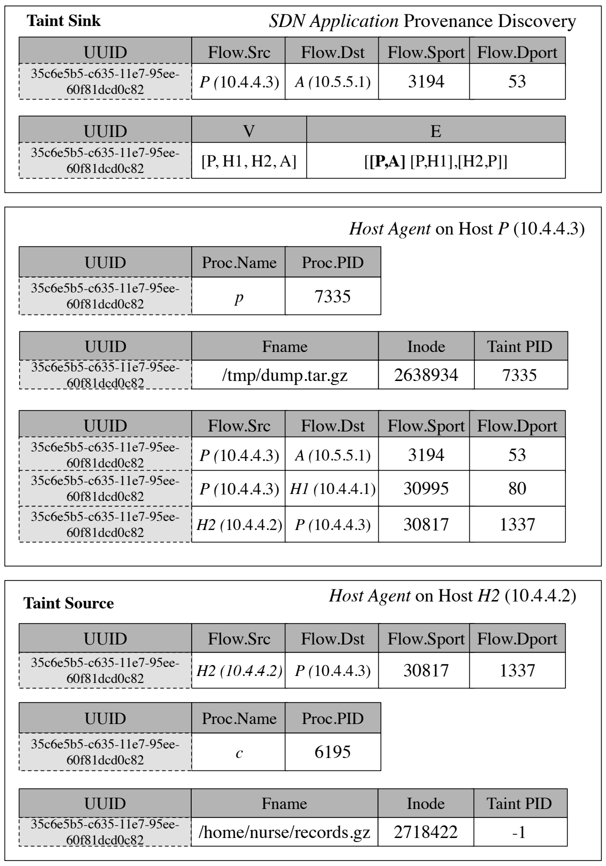
Labeling: A key component of the PivotWall design is to maintain persistent labels that extend beyond the boundary of the host. When a file is sent over the network or a process is accessed via the network, the label must propagate to the associated network flow. The controller must be able to determine the confidentiality of the traffic and the origin of the confidentiality to match the appropriate policy. SimpleFlow uses a *Netfilter* interface in the kernel to apply a confidential *host label.* This label is a binary representation that depicts if the packet contains classified data. The *host label* is created by borrowing the extra bit in the IP fragment field (identified as the evil bit in RFC 3514 [7]).A packet bearing the evil bit is handled as confidential by any SimpleFlow enabled hosts. However, to extend this label to the network, we modified the SimpleFlow source by adding a network label.

The *network label* provides the steering information which affects how the PivotWall reference monitor should handle packets. The *network label* borrows the IP Type of Service (ToS) header byte. As identified by Fayazbakhsh et al. [17], the IP ToS header provides a full byte that we use to create matching on-demand flow modifications that work with all legacy OpenFlow hardware. Packets bearing the *network label* are brought to the controller on a per-flow basis. However, to apply policy and create flow modifications, PivotWall requires the origin of the confidential packets.

The *origin label* uses a 128-bit universally unique identifier (UUID) to represent a global identifier for each classified object. In our initial design, we padded each packet with the UUID as a Commercial Security Option Type (6) IP Option. However as described in [19], IP Options are rarely well supported. Further, our initial experiment demonstrated that adding 128-bits to every classified packet introduced performance consequences. To ensure compatibility and performance, our design labels network flows by sending the 128-bit UUID in a control message encapsulated in an ICMP packet. While our current implementation has the host agent send this message directly to the controller, it would be straightforward for the OpenFlow switch to capture the control message and forward it to the controller (e.g., in the case of an out-of-band controller). *Pedigree* [50, 51] also uses a separate connection to pass provenance data to their network arbitrator.

An *origin-label* is generated when an administrator initially labels a file or process as confidential. The host agent propagates the UUID for each interaction with the confidential-labeled file. Consider the example where a process copies data from a confidential-labeled file to a new file. As a result, both the tainted process and the newly-tainted file bear the *origin-label* of the original file. To implement NIFC, these unique identifiers extend beyond the boundary of the host. PivotWall accomplishes this by sending the UUID for each tainted network-flow to the PivotWall application at the controller. The application updates the NIFC graph, indexed by the UUID. Further, PivotWall sends a control message with the UUID to the destination host. The host agent propagates the UUID from the network flow to interactions on the host.

It is possible that a single object might be associated with multiple UUIDs. Consider when two confidential files are merged into a new file. Or a remote access toolkit process may attempt to exfiltrate multiple confidential files over the same network flow. When multiple confidential objects are merged, the host agent assigns a new UUID to the combination. The host agent sends the updated UUID in a control message to the PivotWall application at the controller. Further, the controller application sends a control message containing the updated UUID to the destination host. The controller creates a new NIFC graph from the merger of the NIFC graphs from each object. Since the new NIFC graph is used to check for policy violations, PivotWall deletes the flow modifications for all edges of the merged graph. This action brings all network flows (represented as edges) back to the controller to re-examine the policy implications of the UUID combination.



Taint Sink *SDN Application* Provenance Discovery

UUID	Flow.Src	Flow.Dst	Flow.Sport	Flow.Dport
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*P* (10.4.4.3)	*A* (10.5.5.1)	3194	53

UUID	V	E
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	[P, H1, H2, A]	[**[P,A]** [P,H1],[H2,P]]

Host Agent on Host *P* (10.4.4.3)

UUID	Proc.Name	Proc.PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*p*	7335

UUID	Fname	Inode	Taint PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	/tmp/dump.tar.gz	2638934	7335

UUID	Flow.Src	Flow.Dst	Flow.Sport	Flow.Dport
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*P* (10.4.4.3)	*A* (10.5.5.1)	3194	53
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*P* (10.4.4.3)	*H1* (10.4.4.1)	30995	80
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*H2* (10.4.4.2)	*P* (10.4.4.3)	30817	1337

Taint Source *Host Agent* on Host *H2* (10.4.4.2)

UUID	Flow.Src	Flow.Dst	Flow.Sport	Flow.Dport
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*H2 (10.4.4.2)*	*P* (10.4.4.3)	30817	1337

UUID	Proc.Name	Proc.PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	*c*	6195

UUID	Fname	Inode	Taint PID
35c6e5b5-c635-11e7-95ee-60f81dcd0c82	/home/nurse/records.gz	2718422	-1

Figure 4: Provenance discovery enabled by information collected by the controller (NIFC graph) and host (process and file information).

4.3 Forensic Analysis

The PivotWall taint propagation logs are a valuable resource when performing forensics in response to an attack. Because PivotWall uses taint analysis to detect secrecy attacks, the logs within the SDN application and the host agent can be used to create a provenance graph describing how the attack progressed. PivotWall raises an alarm for the last hop of an attacker's chain. Therefore, a forensics tool can walk backwards to identify all of the hosts and resources that led to the alarm. Both the network and the host maintain provenance data about taints in SQLite databases. Therefore, tainted objects can be easily identified using SQL queries. Producing this set and aggregating the data over the network provides an incident response team and forensic analysts valuable information to contain the attack and remove the attacker from the internal network. The PivotWall heuristics provide visibility of the taint source, the intermediate hosts involved, and the series of intermediate tainted objects (processes, files, and network connections).

Figure 4 abstracts the queries necessary to offer insight into the motivating example attack in Section 2.1. The first query begins at host P, where a policy violation occurred on the flow $P \longrightarrow A$. Examining the NIFC graph, an analyst can determine this violated policy because a path exists from $H_2 \longrightarrow A$. Examining the host agent on P, provides information about the specifically tainted processes, files, and network flows. Examining these queries, we determine that the process p propagated the attack and wrote a file named *dump.tar.gz*. Following the graph back to the taint sink (H_2), we determine that process c read the source of the origin taint file *records.gz*. Ultimately, these heuristics and queries provide an administrator the necessary information to determine the anatomy of the attack and the incidental elements of the attack.

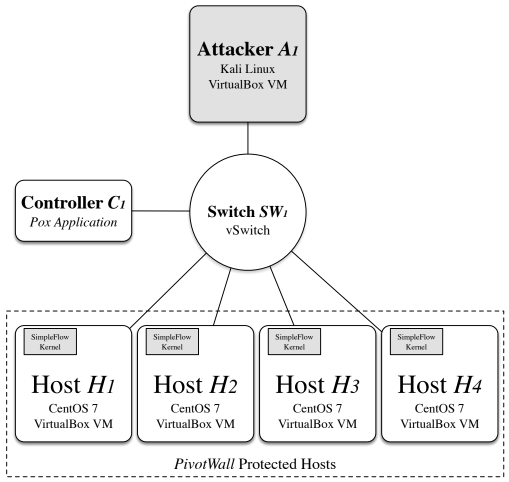


Figure 5: Diagram of Testbed Network

5 PERFORMANCE EVALUATION

PivotWall extends information flow tracking to the network by leveraging the logically central placement of the SDN controller. By extending traditional host-based information-flow tracking to the network, PivotWall prevents attacks that bypass existing enterprise defense mechanisms. In this section, we empirically evaluate the design and performance of our prototype by answering the following research questions.

RQ1 (*Stealthy Attack Detection*): Can PivotWall detect attacks that otherwise bypass traditional enterprise defense mechanisms?
RQ2 (*Attack Coverage*): Can PivotWall detect against a broad coverage of communication channels and tools?
RQ3 (*Network Scalability*): What is the scalability of the network controller application?
RQ4 (*Response Capability*): Can an administrator extend PivotWall to provide a custom-tailored, flexible response to an attack?

Experimental Setup: Figure 5 illustrates the testbed network for our experiments. We created a virtual network infrastructure environment on a 2.8 GHz Intel Core i7 CPU with 12GB of memory running Ubuntu 16.04.2 LTS, an OpenFlow control platform, Open vSwitch 2.7.90 and the Pox 0.2.0 controller running our PivotWall security application. Our hosts protected by PivotWall are VirtualBox instances with 512MB of RAM running CentOS 7, SimpleFlow 0.3.0, and our PivotWall packet relabeling agent. Each host is connected to a virtualized network, via soft SDN switches (Open vSwitch) that support OpenFlow. Our attacker host is a VirtualBox VM with 512MB of RAM running Ubuntu 16.04.2 LTS.

5.1 RQ1: Stealthy Attack Detection

Attackers employ stealthy attacks [67] to exploit blind spots in enterprise defenses to circumvent traditional defense mechanisms. We evaluated PivotWall against stealthy attack tools to demonstrate how our prototype defends against attacks that bypass traditional enterprise defenses. Our evaluation considered two specific stealthy attack scenarios. We first examined a data-loss-prevention (DLP) scenario where the attacker originated inside the network. Next, we examined a stepping-stone-attack scenario where the attacker originated external to the network to gain access to confidential data. In both scenarios, we compared PivotWall's ability to detect and mitigate attacks in comparison to traditional defenses.

Stealthy Data Loss Prevention (DLP) Attack: To understand the benefits of PivotWall, we compared it to two traditional enterprise defenses: a network intrusion detection system (*Snort*) and a host-based firewall (*iptables*). We applied the traditional defenses in a conservative manner with the goal of blocking communication from a protected host to an external network. On the IDS, we applied an IDS rule to block any TCP packet from our confidential host (10.10.1.1) to a destination outside our local network (!10.10.1.0/24). At the host firewall, we applied an *iptables* rule to block any TCP packets to a destination outside our local network.

Using the tools listed in Table 2, we modeled a stealthy attacker that passed data from the protected host to the intermediary host on the local network. Although these four tools can be used in multiple ways, we tested configurations that maximized coverage between the variables of encryption and relay model. We transferred and stored data on the intermediary host by using netcat and cryptcat before ultimately egressing the data (store-forward). In contrast, we used socat and meterpreter to forward TCP ports, actively relaying the data through the intermediate host to egress the data.

In all cases, the network intrusion detection system and the host-based firewall failed to detect the data loss. Both conservatively applied rules failed to detect the information flow through the intermediary host. To demonstrate PivotWall, we labeled a file as confidential, and applied a simple policy rule that prevented confidential information with an origin of 10.10.1.1 to flow beyond the boundary of the local area network.

Listing 4 displays the logging output on our prototype controller for the netcat case. Upon identifying the origin (taint-source) from the UUID, the reference monitor determined the flow to the destination (taint-sink) matched a rule in the policy and subsequently dropped the packets. As listed in Table 2, PivotWall successfully detected and dropped packets containing the confidential file.

Stealthy Stepping Stone Attack: We modeled a stealthy stepping stone attack scenario where an external attacker used an intermediate host to pivot and attack an internal confidential host. This experiment also confirmed that traditional access controls and enterprise defenses lack the context of information flow and fail to detect stealthy attacks.

To illustrate traditional enterprise defenses, we enabled an Apache web server and restricted access to the server to only local area network hosts using an Apache access control list. To bypass this access control list, we established a stepping stone (on the local area network) using the *socat* toolkit. This tool forwarded inbound traffic on TCP Port 4444 on the stepping stone to an external request to TCP Port 80 on the webserver. This bypassed the ACL on the webserver, since the stepping stone IP address was within the permitted ACL. However, this violated the intent of our expressed access policy since our external attacker could access information on the webserver. We repeated the experiment with a meterpreter port forwarder and determined the same result. Traditional access controls cannot determine the information flow and fail against stealthy attacks. To illustrate the information flow tracking aspect of PivotWall, we repeated the experiment but marked the Apache process as confidential. We wrote a PivotWall policy to prevent external communication with the protected webserver.

5.2 RQ2: Attack Coverage

To test the coverage range of PivotWall, we examined its ability to detect a broad range of data exfiltration attacks. The extensible Data Exfiltration Toolkit (DET) [2] provides nine different communication channels for covertly exfiltrating data out of a network, bypassing traditional tools. The tool abuses common protocols including DNS, HTTP and ICMP as well as commercial tools such as Gmail, Slack, and Twitter. To test the coverage, we established an attacker's listening post outside the local area network.

We then modeled an insider attack where an attacker used DET to egress data outside the local area network. We repeated the process for the DET covert channel plugins for DNS, Gmail, Google_Docs, HTTP, Slack, TCP, Twitter and UDP. The results were the same for all covert channels: PivotWall identified the access of the confidential data and applied the PivotWall policy preventing the egress of confidential data outside the local area network. The results are summarized in Table 3.

5.3 RQ3: Network Scalability

We evaluated the network performance of our prototype solution using the iPerf3 toolkit [60]. To realize the total achievable bandwidth of our solution, we measured the total MBytes transferred during a ten second TCP connection and recorded the impact on the congestion window. We repeated this experiment ten different times to determine the average and standard deviation for bandwidth and bytes transferred during the connection. To understand the impact of the flow modifications of our prototype solutions, we compared PivotWall against an SDN application that performed mac-layer matching. To illustrate the impact of confidential labels, we tainted the iPerf3 process on the Server and repeated the experiment. Our results are illustrated in Table 4.

As the results illustrate, both unlabeled PivotWall traffic and mac-layer traffic have similar performance. However, the labeled traffic (e.g. the tainted iPerf3 process) is limited to 53% of the achievable bandwidth. In our experiments, the most notable impact was on the TCP congestion window of labeled traffic. As described in Section 4.1, labeled traffic is brought to the controller on a per-flow basis. As the reference monitor matches a rule, the controller sends a flow modification for the remainder of the flow. Prior to the establishment of the flow modification, the TCP congestion window does not grow at the same rate as mac-layer matched traffic.

Table 2: Stealthy Data Loss Evaluation Tests

Toolkit	Encrypted	Model	Snort IDS Detected	Ipfwadm Firewall Detected	PivotWall Detected
Netcat	N	Store-Forward	N	N	Y
Cryptcat	Y	Store-Forward	N	N	Y
Socat Forward	N	Active-Relay	N	N	Y
Meterpeter Port Forward	Y	Active-Relay	N	N	Y

Listing 4: PivotWall Detection of Netcat Store-and-Forward

```
PivotWall: Received UUID=b0815bee-b82b-11e7-b120-60f81dcd0c82 for 10.1.1.2:55848->10.5.5.1:1337
PivotWall: UUID=b0815bee-b82b-11e7-b120-60f81dcd0c82 originates at 10.1.1.1
PivotWall: Detected New Labeled Flow From 10.1.1.2->10.5.5.1, 55848, 1337
PivotWall: Rule Matches: drop tcp 10.1.1.1 any -> !10.1.1.0/24 any
PivotWall: Rule Action: drop, sending flow modification for 10.1.1.2:55848,10.5.5.1:1337
```

Table 3: Coverage of DET Plugin Attacks

Plugin	Description	Transport Protocol	Pivotwall Detected
DNS	DNS Record Request	UDP/TCP	Y
Gmail	B64-encoded e-mail	TCP	Y
Google_Docs	B64-encoded parameterized URL	TCP	Y
HTTP	B64-encoded HTTP Header	TCP	Y
ICMP	ICMP Echo Requests	UDP	Y
Slack	Hex-encoded Slack chat message	TCP	Y
TCP	Hex-encoded Raw TCP socket	TCP	Y
Twitter	B64-encoded Direct Message to self	TCP	Y
UDP	Hex-encoded Raw UDP socket	UDP	Y

Table 4: iPerf Bandwidth Results

	Mac-Layer Switch	PivotWall	PivotWall (Confidential)
Transferred (MB)	833.6 ±30.5	836.3 ±22.3	445.2 ±62.7
Bandwidth (Mbits/sec)	696.7 ±25.7	699.0 ±19.0	372.9 ±52.6

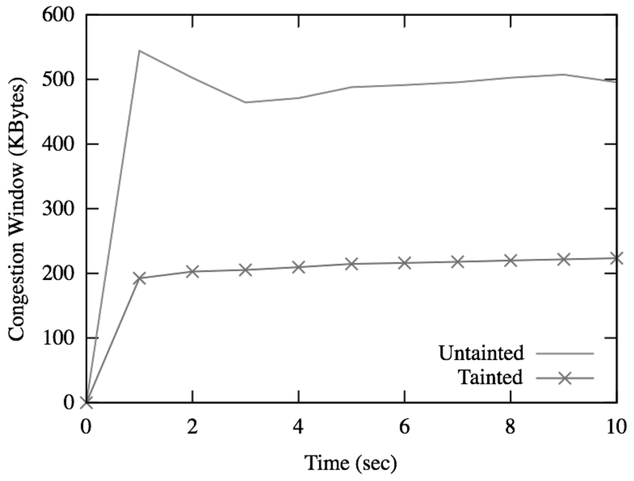


Figure 6: PivotWall Impact on TCP Congestion Window

Figure 6 depicts the average growth of the congestion window for labeled and unlabeled traffic. Labeled traffic suffers for two reasons. Labeled packets pass through a *netfilter* interface (when the label is applied) and are also inspected on a per-flow basis by the reference monitor at the controller. These actions cause the packets to be acknowledged at a slower rate, causing the congestion window to grow at a slower rate compared to packets that bypass the *netfilter* interface and the reference monitor. This host-introduced delay causes the most significant performance impact.

At the SDN layer, PivotWall 's SDN security application performance benefits from the design that only labeled packets are redirected to the controller for inspection. The implementation of PivotWall achieves this by setting the *NW_ToS* flag OFPMatch to to a higher priority over standard flow rules. Further, labeled packets are brought only a per-flow basis. After the first labeled packet is inspected, only packets requiring modification are redirected through the controller. Depending on the demands of packet modification by *Modify-Scripts*, future solutions could examine using NFV to expand the scalability of the PivotWall.

5.4 RQ4: Response Capability

We examined the ability of the PivotWall modify-script functionality to respond to a covert attack which uses the DNS protocol as its channel. Detection of covert DNS channels presents an interesting problem, with several proposed machine-learning based approaches for detecting the channel [25, 26, 29, 32, 53]. Examples of DNS-based attacker tools include OzymanDNS, dns2tcp, iodine, heyoka, element53, DeNiSe, and DNSCat.

We examined how PivotWall prevents an attacker from using DNSCat to covertly embed an attacker channel in valid DNS requests. A significant challenge associated with DNSCat is that it supports forwarding the channel through a legitimate DNS Server

Listing 5: Modify-Script To Log/Redirect DNSCat

```
rewrite_dns_query(packet):
  dns = packet.find('dns')
  ip  = packet.find('ipv4')
  for q in dns.questions:
    log.info("[+] Tainted Query: %s " % q.name)
    log.info("[+] DNS Server: %s " % ip.dstip)
  redir_dst = "10.10.10.10"
  return redir_pkt(packet,redir_dst)
```

to an attacker-controlled authoritative server. Because C2 Tunneling redirects traffic through the victim's default DNS servers, the attacker can use it to bypass egress filtering.

To demonstrate the response capability of PivotWall, we configured DNSCat to exfiltrate data from a victim host to an external domain. In our experiments, DNSCat successfully used DNS Queries to establish a covert channel. However, when the attacker tried to use the established channel to exfiltrate confidential data, the host agent propagated the taint from the file to DNSCat and ultimately to the UDP packets carrying the DNS Requests.

To address this particular attack, we created a PivotWall Modify-Script that rewrote infected queries and logged the domain name used in the attack. Our script, which successfully logged and redirected tainted DNS Queries is listed in Listing 5.

6 DISCUSSION

Limitations of the Host Agent: We recognize that an attacker can disable the host agent if the attacker fully compromises the host. From the presence of a full compromise, an attacker can un-label confidential data, intercept and modify host-agent control messages, labeled packets, and exhaust the controller resources by a denial of service attack. Attestation of the host agent is necessary but we consider an orthogonal problem for this work. The host agent does not address side channel or covert channels that exfiltrate confidential data outside network connections.

The host agent has difficulty dealing with monolithic applications that do not rely on the operating system to partition information into separate objects. For example, many database engines and web servers themselves implement access controls on information. This is exacerbated by using a single process to manage connections from a number of clients. These factors lead to a semantic gap between monolithic application and the operating system kernel which implements information-flow tracking. The researchers behind SELinux have encountered these same challenges, and they have proposed a series of changes to application software [33, 35, 43, 45]. Finally, the host agent does not address a distributed or cloud environment where processes may move between hosts. We refer to Pappas et. al, who have presented solutions for cloud-based information flow control. [46, 47]

Blind-Host Assumption Tainting: We do not address hosts that cannot participate in the defense due to resource constraints. Embedded IoT devices are often used in stepping stone attacks because they are programmed with hard-coded credentials in the firmware. While an attacker might be able to construct an SSH pivot through this device, it is highly unlikely that we would be able to modify it to participate in the active defense like a full host. We assume that when a host is unable to participate in the defense, the controller would be able to use *assumption-tainting*. Our initial results offer promising results into *assumption-tainting*. However, we reserve the details for future work.

Integrity Attacks: Not all attacks seek to exfiltrate information. An adversary may seek to, for example, modify a file in a source code repository to insert a back door or similar vulnerability. While the focus of this paper is secrecy attacks, PivotWall provides primitives that are also valuable to defend against integrity attacks. For example, if a Git server is assigned a label, PivotWall network tainting will follow the TCP ACK messages back to to the network flow returning to the attacker on the Internet. However, applying PivotWall in this way may induce significant false positives (e.g., if developers use code snippets from Stack Overflow), and therefore requires further investigation.

7 RELATED WORK

PivotWall touches on several areas of prior work. We begin by discussing prior approaches for detecting stepping stone attacks by correlating network traffic. Next we describe how SDN has been leveraged in the past to enhance network configuration and management. Finally, we discuss host-based provenance frameworks.

7.1 Detection Using Network Flow Correlation

Previous approaches to detect stepping stone attacks have relied on passive observation [64, 70] or actively watermarking packets or flows [61–63] and using the embedded watermark to correlate flows. However, these approaches mostly focus on detecting interactive stepping stone attacks where the attacker maintains a command and control channel and performs the attack steps within a *maximum tolerable delay*. These approaches do not necessarily consider the methods where an attacker evades detection using timing perturbation, introducing random long delays between packets, or adding chaff packets, flow splitting, or repacketization.

Due to concerns that embedded watermarks are observable, RAINBOW [24] proposed an invisible non-blind watermarking technique. RAINBOW records both incoming and outgoing flows and correlates flows using small delays. The technique, however, does not scale to large networks with heavy traffic. Others have proposed techniques that use intervals in different ways to correlate network flows. SWIRL [23] changes the locations of packets within selected time slots to encode watermarks. Despite significant effort, there is no robust way of correlating network flows with high accuracy. For an active network flow watermarking technique to work, the watermark must be preserved across stepping stones. However, if the attacker uses store-and-forward or split-relay techniques, the embedded watermark is lost. Store-and-forward attacks were first discussed by Coskun and Memom [12]. Along with split-relay based attacks, they necessitate a new approach for stepping stones attack detection. PivotWall aims to address these new challenges while tackling existing ones.

7.2 Network Management and Monitoring for SDN

With the increasing popularization of Software-Defined Networks [11, 18], many different controller frameworks have been proposed and adopted in practice [8, 16, 21, 36, 42]. Kim et al. [34] discuss the new possibilities for network management and configuration methods provided by the OpenFlow protocol [20, 41]. Their studies demonstrate that SDN significantly reduces the complexity of network management in a variety of network settings and for a range of network policies. We leverage the flexibility and the novel properties of SDN to enable security mechanisms that may not be feasible in a traditional network.

There have been a number of proposals that use SDN to enhance network security. Avant-Guard [55] specifically focuses on addressing the communication bottleneck between the control and data planes by identifying malicious traffic, including network scanning and denial-of-service. Network Iron Curtain [57] also focuses on detecting scanning attacks in an SDN environment by implementing a security service that responds to network scanning predefined policies and redirecting attackers to a honeynet. This confuses attackers by providing fake scanning results. Further, Abaid et al. [1] expand the SDN architecture and develop an application to allow deep packet inspection by proposing to elastically partition network traffic on demand using a broad range of detectors. Their research bridges the gap between distributed DPI and SDN-based NIDS. Bates et al. [4] studied the possibility of using SDN for network forensics. They designed an SDN-based forensic system that can be used to investigate previously unobservable attacks such as data exfiltration and collusion between compromised nodes by using SDN to maintain a global view of the network activities.

Apart from leveraging the properties of SDN for building security mechanisms, several recent papers have focused on the security aspects of software defined networks [49, 54]. FRESCO [54] proposes an application development framework for rapidly prototyping security applications for an OpenFlow controller. The authors propose a modular design that allows development through minimal coding. Porras et al. presented SE-Floodlight [49], which is a reference implementation of a security-enhanced controller and proposed a role-based hierarchical resolution strategy that could resolves conflicts, along with a Northbound API that could provide authenticated per-message credentials.

In contrast to these prior works, PivotWall enhances the SDN controller with information from host agents. It is also the first solution to use SDN to actively perform network taint analysis to detect stepping stone attacks.

7.3 Host Provenance Frameworks

Prior work [5, 37–39] uses audit logging and provenance propagation on hosts to investigate and detect attacks. The central idea of these frameworks is to track the flow of attack provenance from an initial source and so that any future actions can be attributed back to the source.

BEEP [37] provides efficient, dependence explosion-free logging for binary programs. It partitions a long running process to multiple autonomous units that handle independent input data to perform fine-grained logging. BEEP-generated logs, along with the regular audit logs, enable identifying precise causality between a root cause (i.e. an attack) and its symptoms, avoiding the dependence explosion problem with regular audit logs. In a separate work [38], the authors of BEEP presented LogGC, which is an audit logging system towards practical computer attack forensics that uses garbage collection to along with partitioning a database file into data units so that dependences can be captured at tuple level. ProTracer [39] is a lightweight provenance tracing system that alternates between system event logging and unit level taint propagation along with event processing. ProTracer leverages a lightweight kernel module and a concurrent userspace daemon. The LPM interface [5] is a framework for the development of provenance-aware systems. It creates trusted provenance-aware execution environments, imposes insignificant performance overhead on normal system operation, and responds to queries about object ancestry in tens of milliseconds.

Most the provenance tracking systems lack the network context while performing provenance propagation. They primarily focus on mitigating host data loss or integrity protection for a single host. Therefore, these systems would have difficulty identifying a stepping stone attack chain. PivotWall complements these prior approaches in that its host agent could be enhanced by incorporating their host-based provenance frameworks.

Pedigree [50, 51] is closest to PivotWall in concept. It extends packets with a taint tag derived from host-level information and uses an OpenFlow-based arbiter to make security decisions. To mitigate taint explosion, Pedigree probabalistically removes taint bits, which sacrifices security for usability. PivotWall's NIFC graphs provides an alternate design approach that puts declassification of tainted network packets more directly into the hands of network administrators.

8 CONCLUSION

This paper presented PivotWall, a new network security architecture to implement information flow control in a distributed environment. Our solution combines the logically central placement of the SDN Controller with the context of host-based information flow tracking. We implemented a prototype by implementing a host agent and network controller application. We evaluated our prototype by testing it against a broad coverage of stealthy attack tools, examined the performance impacts, and demonstrated the ability to provide unique response capabilities to real-world attacks. We demonstrate that PivotWall is a feasible approach to enable context-based response and prevention capabilities that would be difficult to realize with existing solutions. With the ability to detect stealthy attacks and implement reactive measures with acceptable impacts to performance, PivotWall is a promising solution for enhancing the security of enterprise networks.

ACKNOWLEDGEMENTS

This material is based upon work supported in whole or in part with funding from the Laboratory for Analytic Sciences (LAS) and NSF grant CNS-1330553. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the LAS and/or any agency or entity of the United States Government.

REFERENCES

[1] Zainab Abaid, Mohsen Rezvani, and Sanjay Jha. 2014. MalwareMonitor: An SDN-based Framework for Securing Large Networks. In *CoNEXT Student Workshop (CoNEXT Student Workshop '14)*. ACM, New York, NY, USA, 40–42.

[2] Paul Amar. 2016. DET (extensible) Data Exfiltration Toolkit. https://github.com/sensepost/DET. (Sep 2016).

[3] Jean Bacon, David Eyers, Thomas FJ-M Pasquier, Jatinder Singh, Ioannis Papagiannis, and Peter Pietzuch. 2014. Information flow control for secure cloud computing. *IEEE Transactions on Network and Service Management* 11, 1 (2014), 76–89.

[4] Adam Bates, Kevin Butler, Andreas Haeberlen, Micah Sherr, and Wenchao Zhou. 2014. Let SDN be your eyes: Secure forensics in data center networks. In *NDSS workshop on security of emerging network technologies (SENT)*.

[5] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. 2015. Trustworthy whole-system provenance for the Linux kernel. In *USENIX Security Symposium*. 319–334.

[6] D Elliott Bell and Leonard J LaPadula. 1973. *Secure computer systems: Mathematical foundations*. Technical Report. MITRE CORP BEDFORD MA.

[7] Steven Bellovin. 2003. RFC 3514: The Security Flag in the IPv4 Header. https://rfc-editor.org/rfc/rfc3514.txt. (1 April 2003).

[8] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and others. 2014. ONOS: towards an open, distributed SDN OS. In *ACM workshop on Hot topics in software defined networking*. 1–6.

[9] Ron Bowes. 2016. DNSCat2. https://github.com/iagox86/dnscat2. (2016).

[10] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, Vol. 37. ACM, 1–12.

[11] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, Vol. 37. 1–12.

[12] Baris Coskun and Nasir Memon. 2007. Efficient detection of delay-constrained relay nodes. In *ACM Annual Computer Security Applications Conference (ACSAC)*. 353–362.

[13] daemon9. 1996. Project Loki. http://phrack.org/issues/49/1.html. (Aug 1996).

[14] Dorothy E Denning. 1976. A lattice model of secure information flow. *Commun. ACM* 19, 5 (1976), 236–243.

[15] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 5.

[16] David Erickson. 2013. The beacon openflow controller. In *ACM SIGCOMM workshop on Hot topics in software defined networking*. 13–18.

[17] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. 2013. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 19–24.

[18] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2014. The road to SDN: an intellectual history of programmable networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 44(2). ACM, 87–98.

[19] Rodrigo Fonseca, George Manning Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. 2005. IP Options are not an option. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-24.html

[20] Open Networking Foundation. 2015. OpenFlow Switch Specification, Version 1.5.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf. (2015).

[21] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. 2008. NOX: towards an operating system for networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 38. ACM, 105–110.

[22] Victor Heorhiadi, Seyed Kaveh Fayaz, Michael K Reiter, and Vyas Sekar. 2014. SNIPS: A Software-Defined Approach for Scaling Intrusion Prevention Systems via Offloading. In *Information Systems Security (ISS)*. pp. 9–29.

[23] Amir Houmansadr and Nikita Borisov. 2011. SWIRL: A Scalable Watermark to Detect Correlated Network Flows.. In *Network and Distributed System Security Symposium (NDSS)*.

[24] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. 2009. RAINBOW: A Robust And Invisible Non-Blind Watermark for Network Flows.. In *Network and Distributed System Security Symposium (NDSS)*.

[25] Hikaru Ichise, Yong Jin, and Katsuyoshi Iida. 2015. Detection method of DNS-based botnet communication using obtained NS record history. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, Vol. 3. IEEE, 676–677.

[26] Hikaru ICHISE, Yong JIN, and Katsuyoshi IIDA. 2018. Analysis of DNS TXT Record Usage and Consideration of Botnet Communication Detection. *IEICE Transactions on Communications* (2018), 2017ITP0009.

[27] Mehmet Ince. 2016. Data Exfiltration Attacks against Corporate Network. *Pentest Blog* (2016). https://pentest.blog/data-exfiltration-tunneling-attacks-against-corporate-network/

[28] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. 2012. Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking. In *Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*. ACM, New York, NY, USA, 127–132.

[29] Yong Jin, Hikaru Ichise, and Katsuyoshi Iida. 2015. Design of detecting botnet communication by monitoring direct outbound DNS queries. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*. IEEE, 37–41.

[30] Ryan Johnson, Jessie Lass, and W. Michael Petullo. 2016. Studying Naïve Users and the Insider Threat with SimpleFlow. In *8th ACM CCS International Workshop on Managing Insider Security Threats (MIST '16)*. ACM, New York, NY, USA, 35–46. http://www.flyn.org/publications/2016-SimpleFlow.pdf

[31] Panos Kampanakis, Harry Perros, and Tsegereda Beyene. 2014. SDN-based solutions for Moving Target Defense network protection. In *IEEE World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 1–6.

[32] A Mert Kara, Hamad Binsalleeh, Mohammad Mannan, Amr Youssef, and Mourad Debbabi. 2014. Detection of malicious payload distribution channels in DNS. In *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 853–858.

[33] Doug Kilpatrick, Wayne Salamon, and Chris Vance. 2003. Securing The X Window System With SELinux. (2003).

[34] Hyojoon Kim and Nick Feamster. 2013. Improving network management with software defined networking. *IEEE Communications Magazine* 51, 2 (2013), 114–119.

[35] KaiGai Kohei. 2008. Security-Enhanced PostgreSQL: System-Wide Consistency in Access Control. (May 2008). Presentation at the 2008 PostgreSQL Conference.

[36] Open Networking Lab. 2015. Pox Controller Wiki. https://openflow.stanford.edu/display/ONL/POX+Wiki. (2015).

[37] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition.. In *Network and Distributed System Security Symposium (NDSS)*.

[38] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. Loggc: Garbage collecting audit log. In *ACM SIGSAC conference on Computer & communications security*. 1005–1016.

[39] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. 2016. ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting. In *Network and Distributed System Security Symposium (NDSS)*.

[40] Douglas C. MacFarland and Craig A. Shue. 2015. The SDN Shuffle: Creating a Moving-Target Defense Using Host-based Software-Defined Networking. In *ACM Workshop on Moving Target Defense (MTD '15)*. ACM, New York, NY, USA, 37–41.

[41] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 38. ACM, 69–74.

[42] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. 2014. Opendaylight: Towards a model-driven sdn controller architecture. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*.

[43] James Morris. 2008. Have You Driven an SELinux Lately?. In *Proceedings of the Linux Symposium (OLS '08)*, Vol. 2. 101–114.

[44] Steven J. Murdoch and Stephen Lewis. 2005. Embedding Covert Channels into TCP/IP. In *Proceedings of the 7th International Conference on Information Hiding (IH'05)*. Springer-Verlag, Berlin, Heidelberg, 247–261. DOI: https://doi.org/10.1007/11558859_19

[45] Simone Mutti, Enrico Bacis, and Stefano Paraboschi. 2015. SeSQLite: Security Enhanced SQLite: Mandatory Access Control for Android Databases. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC 2015)*. ACM, New York, NY, USA, 411–420. DOI: https://doi.org/10.1145/2818000.2818041

[46] Ioannis Papagiannis and Peter Pietzuch. 2012. CloudFilter: Practical Control of Sensitive Data Propagation to the Cloud. In *ACM Workshop on Cloud Computing Security Workshop (CCSW '12)*. ACM, New York, NY, USA, 97–102. DOI: https://doi.org/10.1145/2381913.2381931

[47] Vasilis Pappas, Vasileios P Kemerlis, Angeliki Zavou, Michalis Polychronakis, and Angelos D Keromytis. 2013. CloudFence: Data flow tracking as a cloud service. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 411–431.

[48] Pai Peng, Peng Ning, Douglas S Reeves, and Xinyuan Wang. 2005. Active timing-based correlation of perturbed traffic flows with chaff packets. In *IEEE International Conference on Distributed Computing Systems Workshops*. 107–113.

[49] Phillip A Porras, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. 2015. Securing the Software Defined Network Control Layer.. In *Network and Distributed System Security Symposium (NDSS)*.

[50] Anirudh Ramachandran, Kaushik Bhandankar, Mukarram Bin Tariq, and Nick Feamster. 2008. *Packets with provenance*. Technical Report. Georgia Institute of Technology.

[51] Anirudh Ramachandran, Yogesh Mundada, Mukarram Bin Tariq, and Nick Feamster. 2009. Securing enterprise networks using traffic tainting. *Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep. GTCS-09-15* (2009).

[52] Martin Roesch and others. 1999. Snort: Lightweight intrusion detection for networks.. In *Large Installation System Administration Conference (LISA)*, Vol. 99. 229–238.

[53] Pooja Sharma, Sanjeev Kumar, and Neeraj Sharma. 2016. BotMAD: Botnet malicious activity detector based on DNS traffic analysis. In *Next Generation Computing Technologies (NGCT), 2016 2nd International Conference on*. IEEE, 824–830.

[54] Seugwon Shin, Phillip Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, and Mabry Tyson. 2013. FRESCO: Modular Composable Security Services for Software-Defined Networks. In *Network and Distributed System Security Symposium (NDSS)*.

[55] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. 2013. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks. In *ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*. ACM, New York, NY, USA, 413–424.

[56] Robert Shullich, Jie Chu, Ping Ji, and Weifeng Chen. 2011. A survey of research in stepping-stone detection, In International Journal of Electronic Commerce Studies. *International Journal of Electronic Commerce Studies* 2 (2011), 103–126.

[57] YongJoo Song, Seungwon Shin, and Yongjin Choi. 2014. Network Iron Curtain: Hide Enterprise Networks with OpenFlow. In *Information Security Applications (ISA)*.

[58] Darlene Storm. 2015. MEDJACK: Hackers hijacking medical devices to create backdoors in hospital networks. *Computerworld* 8 (2015).

[59] Yuqiong Sun, Giuseppe Petracca, Xinyang Ge, and Trent Jaeger. 2016. Pileus: Protecting User Resources from Vulnerable Cloud Services. In *32nd Annual Conference on Computer Security Applications (ACSAC '16)*. ACM, New York, NY, USA, 52–64.

[60] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. 2005. Iperf: The TCP/UDP bandwidth measurement tool. *htt p://dast. nlanr. net/Projects* (2005).

[61] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. 2005. Tracking anonymous peer-to-peer VoIP calls on the internet. In *ACM Annual Computer Security Applications Conference (ACSAC)*. 81–91.

[62] Xinyuan Wang and Douglas S Reeves. 2003. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *ACM Annual Computer Security Applications Conference (ACSAC)*. 20–29.

[63] Xinyuan Wang, Douglas S Reeves, Peng Ning, and Fang Feng. 2005. Robust network-based attack attribution through probabilistic watermarking of packet flows.

[64] Xinyuan Wang, Douglas S Reeves, and S Felix Wu. 2002. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 244–263.

[65] A. Wool. 2004. A Quantitative Study of Firewall Configuration Errors. *Computer* 37, 6 (June 2004), 62–67.

[66] V. Yegneswaran, P. Barford, and S. Jha. 2004. Global Intrusion Detection in the DOMINO Overlay System. In *In Proceedings of Network and Distributed System Security Symposium (NDSS*.

[67] Tianlong Yu, Seyed K Fayaz, Michael Collins, Vyas Sekar, and Srinivasan Seshan. 2017. PSI: Precise Security Instrumentation for Enterprise Networks. In *Network and Distributed System Security Symposium (NDSS)*.

[68] Sebastian Zander, Grenville Armitage, and Philip Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials* 9, 3 (2007), 44–57.

[69] Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. 2008. Securing Distributed Systems with Information Flow Control. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association, Berkeley, CA, USA, 293–308.

[70] Yin Zhang and Vern Paxson. 2000. Detecting Stepping Stones.. In *USENIX Security Symposium*, Vol. 171. 184.