SDN-Controlled Isolation Orchestration to Support End-User Autonomy

A Major Qualifying Project Report

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

in

Computer Science

by

Jake Backer

Lia Davis

Nour Elmaliki

December 16, 2021

APPROVED:

Professor Craig A. Shue, Project Advisor

Contents

Abstract

Numerous data breaches and ransomware attacks in recent history have highlighted the importance of data security. There is always a trade off between security and end-user autonomy. Organizations need methods of securing data without overly hindering productivity. Current systems either do not provide enough control over data usage or overly restrict users and hinder productivity.

This project designs and implements a system intended to provide fine-grained control over data while allowing end-users more freedom over their systems. Our system leverages the confidentiality benefits of virtualization to provide end-users with multiple environments to work. These environments are protected by security controls proportional to the data contained within. Users are allowed environments for high-risk activity and are confined to interact with sensitive data in low-risk environments. We built a data provenance tracking system to label, update, and transmit data provenance labels. Provenance labels will be used to determine how data is distributed among different risk environments.

We performed benchmark testing on the data provenance tracking system and determined that its overhead does not pose a threat to the usability of the systems it governs. We evaluated the mechanism that transmits provenance labels and likewise concluded that it does not impede the usability of the system or the network on which it transmits.

This paper is a snapshot of the project mid-development. One of the team members is graduating and this deliverable represents the work performed thus far.

1 Introduction

With the rise of data breaches and ransomware attacks in recent years, the importance of data security in the 21st century is clear. Data breaches and ransomware attacks can cost organizations large amounts of money and recovery time, potentially harming their day-to-day operations. In 2020, ransomware payments increased by 311% from previous years to a total amount of $350 million [1].

The COVID-19 pandemic has also sparked a movement of large numbers of people to fully remote work preventing them from being connected directly to a corporate network and potentially leaving themselves and the data they are handling vulnerable on a home network. Current solutions for this problem involve difficult-to-use security controls that can impede day-to-day work [2]. Organizations also must maintain the confidentiality, integrity, and availability of their user' data. This must be done with access controls to ensure the data is only accessed when it needs to be and by the processes and people that need to access it. Though proper access controls greatly reduces potential attack surface, they can make it difficult to work with the data.

Current access control mechanisms do not offer the flexibility and security needed to keep up with these modern issues. In this paper, we are leveraging multiple isolated environments to separate various levels of sensitive information. We are also developing a centralized organizational access control system to integrate with and broker information access and migration between environments. We measure file and network access events in each environment and send these events to the centralized access control system. This system will make decisions on whether or not the actions are permitted, which is then forwarded to the environments to enforce the decisions.

In traditional systems, access controls decisions are made and enforced on endpoints. This system will enable security administrators to influence when information can cross boundaries by combining a network based, centralized access control system along with Mandatory Access Controls to ensure policy is enforced. Additionally, the isolation of data into separate environments along with data provenance tracking helps to ensure data does not become contaminated with more sensitive data without also changing the data's provenance level.

Despite these benefits to our system, there are operational and financial costs associated with implementing it. First, all endpoints must be capable of running multiple Virtual Machines. This requires a significant amount of processing power that may not be included with a typical workstation. Each endpoint must also maintain a constant network connection to communicate with the centralized access control system. Additionally, the centralized access control system must be capable of processing large amount of incoming data and the network that supports the system must be capable of transferring the data with little latency.

The remainder of the paper is structured as follows. In Section 2, we discuss background information on access control mechanisms, virtualisation, as well as works related to this paper.

In Section 3, we discuss the design of the centralized access control system as well as a high level overview of the data provenance system that is run on each endpoint. Section 4 includes details about the proof of concept we built to implement our design. In Section 5, we discuss the tests we ran to measure behavior of our implemented system and analyzed the results. Finally, we discuss lessons learned, future work, and the implications of our work in Section 6.

Note: this paper is a work in progress. One member of the team will be graduating during the project, therefore a preliminary version of the report was produced. A final version of the report will occur in the Spring of 2022.

2 Background and Related Work

This research focuses on methods to transmit and maintain data provenance while minimizing organization overhead and maintaining user autonomy. To provide the reader with a better understanding of the landscape, we introduce the concepts of access control mechanisms and data provenance. We also report on similar works in the fields of data loss prevention, intrusion detection systems, virtualization, and software-defined networks.

Private organizations have long lacked a mechanism to securely store and regulate access to data of multiple sensitivities when stored on an individual's machine. Popular access control schemes do not offer the fine-grained control in a modern work environment (Bell-LaPadula or Role-based Access Control), or require significant overhead to maintain the system (Attribute-based Access Control)[3][4][5].

2.1 Access Control Mechanisms

Access control mechanisms provide a way to ensure data confidentiality and integrity. There exists a large variety of different mechanisms, each with different configurations. With all of these schemes, different systems can use different configurations and mechanisms for their specific needs. This paper is based around Mandatory Access Control and a variation of Attribute-Based Access Control.

Currently, Discretionary Access Control (DAC) is one of the most common schemes for access control. DAC governs the access of information based on the user's identity, authorization, the information itself, and whether the user is requesting to read, write, or execute the information [6]. Any subject in a DAC system can pass its access, or a subset of its access, to any other subject [7]. For example, the user "user" in a Linux system could grant everyone access to a file it is the owner of. Because of this, DAC is much more flexible than other mechanisms and is therefore used widely in standard operating systems such as Microsoft Windows and Linux [8]. Unfortunately, it is simple to bypass DAC, as once a user has a piece of data, there is no restriction on its usage. As such, a user could simply re-distribute the data to users who were not authorized to read the data by the original owner [7].

In contrast, Mandatory Access Control (MAC) is a high security access control scheme that allows security administrators to define a policy that is guaranteed to be enforced for all users. These security policies are arbitrated by a "reference monitor" [4], or an authoritative system that enforces policy. With MAC, users do not have the ability to override security controls. With DAC, a user could modify the controls on the SSH configuration file which could allow unauthorized users access to the user's SSH keys, creating a severe security vulnerability. MAC allows a security administrator to enforce controls, preventing users from creating insecure controls on files. With MAC, users without sufficient authorization cannot grant more access to an object, preventing a central issue with DAC.

One key implementation of MAC is the Bell-LaPadula Model (BLP). BLP was designed to be used by the US military to formalize the Department of Defense Multi-Level Security policy [9]. BLP focuses solely on data confidentiality and does not look at data integrity. There are three primary security properties that define BLP: The Simple Security Property prevents a user from reading an object at a higher security level, the Star Security Property prevents a user from writing to an object at a lower security level, and the Discretionary Security Policy allows further restriction of access using an access matrix [3]. These three rules are more simply stated as “read down, write up”, which means that a person at a specific security level can read from any lower security level and write to any higher security level. There is also a Strong Star Property which allows a subject to only write to their security level, but not a higher or lower level [10]. This property allows for a higher level of integrity than typical BLP provides.

Though DAC and MAC are both valid access control schemes on their own, it is more common to implement them using a higher level scheme such as Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC). RBAC is a method to implement MAC or DAC by assigning users to “roles” where each role has a specific set of permissions. RBAC is currently the gold standard for access control in many organizations as it is easy to implement and provides sufficient security [4]. A major drawback of RBAC is that it is not fine-grained. To provide small changes in access control, new roles must be created which can cause an unmanageable number of roles to exist. In contrast to the coarse-grained control of RBAC, ABAC provides much more fine-grained control. With ABAC, objects are controlled using an access control equation, allowing complex decisions to be made on whether a user is authorized to access the object [5]. For example, an object could be made to be accessible by all users in organization A, all users in organization B that also have a specific level of clearance, or a specific person in organization C. With RBAC, this would be very difficult to implement in an efficient manner, but ABAC allows for fine-grained control [4]. Though ABAC allows for very fine-grained control, implementing ABAC on an enterprise level can incur large development, implementation, and performance costs due to its high complexity [5].

2.2 SELinux

Security-Enhanced Linux (SELinux) allows security administrators to implement MAC policies on a Linux device. SELinux was originally developed by the NSA to implement the Department of Defense’s Multi-Level Security (MLS) system into Linux [11][9]. SELinux is commonly configured via targeted policy or MLS mode. Targeted policy contains a large selections of policies for common Linux applications. MLS allows for classification levels for users and files and implements the Bell-LaPadula model. MLS mode is typically only used my government organizations due to its complexity [11]. Everything in a system with SELinux has a label - including files, processes, and ports. Labels are comprised of an SELinux User, role, type, and level. The SELinux user is conceptually similar to a normal Linux user, SELinux maintains a mapping of SELinux user to regular Linux user. In targeted mode, type is the most important part [11]. SELinux's type enforcement applies policies that define whether a process running with a certain type can access a file labeled with a certain type [11].

2.3 Data Loss Prevention & Intrusion Detection Systems

Organizations that store important data have been increasingly targeted by cyber criminals in recent years. The Department of Health and Human Services (HHS) estimates that almost 30 million patient records were breached in 2020 [12]. In response to the increase of cyber crime, organizations have invested in Data Loss Prevention (DLP) tools. A DLP tool identifies data on a system, organizes that data into classifications, and enforces organizational policies on sensitive data [13]. Corporate data generally exists in the following locations: large centralized/distributes file at rest, moving throughout the network and with external devices, and on endpoints such as laptops and USB drives [14]. For our purposes we will mainly focus on securing data at endpoints, specifically end-user machines.

DLP products faces two main challenges: volume and accuracy. DLP tools are required to process terabytes of stored data, analyze the network traffic of the entire network, and track activity for thousands of endpoints [13]. It is essential that the identification process is efficient and scalable. These products generally use pattern matching and hashing to identify data types. The issue arises when data is reformatted or transformed in a way that avoids programmatic detection [13]. Allowing users to manually classify data is also problematic; human error poses threats to accuracy and consistency. Lastly, a DLP tool must be able to understand and enforce policy requirements as specified by the organization.

An example of such a tool is UC4Win. UC4Win is a data loss prevention solution for Microsoft Windows. Like our project, it uses system calls to provide fine-grained policy protection. UC4Win also shares goals with this project - “the enforcement of a defined policy concerning the usage of sensitive data” [15]. UC4Win accomplishes this by modifying Windows system calls to interpose itself between applications and the operating system [15]. However, our project monitors the system through Linux's logging module. One benefit of this approach is that it takes advantage of native Linux functionality and does not require modifications to the operating system. Due to the difference in mechanisms, UC4Win benefits from increased visibility into user actions and has the ability to block system calls before they happen. UC4Win utilizes Obligation Specification Language (OSL) - a mechanism to encode policy requirements into a machine readable format. One of the biggest limitations of DLP enforcement systems is that they “might not be able to withstand sophisticated attacks, and thus may not be suitable to defend against data disclosure by malicious attackers such as hackers” [15]. For that reason, organizations often employ intrusion detection systems (IDS).

There are three main methodologies for detecting intrusions: signature detection, anomaly detection, and stateful protocol analysis [16]. Different implementations of an IDS will have unique mechanisms of event gathering, storage, and processing. Signature detection, also known as “knowledge-based detection”, searches for patterns that have been recorded in previous attacks [16]. Anomaly detection creates a profile of expected behavior from monitoring normal day-to-day activity and flags abnormal behavior as potential attacks. Stateful protocol analysis, also known as “specification-based detection”, is similar to anomaly detection – it relies on knowledge of specific protocols to create rules that determine if activity may be malicious [16].

Demand for IDS tooling has increased in response to the rise of cyber crime. Accenture’s Cyber Investigations, Forensics and Response team found a 125% increase in cyber crime in the first half of 2021 [17]. CrowdStrike Falcon is a popular suite of enterprise cyber security tools. CrowdStrike installs hosts on all endpoints that monitor events, analyzes the endpoints for abnormal activity, and stores that activity in “Threat Graph”[18]. Threat Graph is a proprietary data structure that summarizes a device into a “state.” That state is stored as the graph node and events are stored as the transitions between nodes [19]. Since Crowdstrike falcon is a commercial product, detailed information of how it works is limited beyond the filed patent.

There are two relevant subsets of intrusion detection systems: host-based intrusion detection systems (HIDS) and network-based intrusion detection systems (NIDS). HIDS is a variant of IDS that generally monitors activity on an end user’s operating system. HIDS often relies on a kernel level event audit system to generate system logs, process logs, user commands, and file access logs [20]. Audit software can be resource intensive but it has high visibility into a system [20]. NIDS monitors network traffic flows on a device. NIDS consumes less resources than HIDS but only having visibility into network traffic reduces accuracy [20].

AlarmNet is a combined host-based IDS and network-based IDS that utilizes a neural network to process event data. AlarmNet is not unique in this. Machine learning is an extremely common tool to process data for an IDS [21] . AlarmNet uses “word embedding methods and convolution neural network” to transform data into an intermediate state to be consumed by the decision making neural network [22]. Once the decision making neural network is sufficiently trained, it is put into a decision making module that detects malicious activity on a system. Our system processes the same data as AlarmNet, however, we employ an algorithm to parse event data. We use that data to detect when data crosses a boundary based on a set of predetermined rules. Host event data is collected by AuditD for our system and AlarmNet [22].

AuditD is a component of the Linux auditing system that can be configured to produce logs of specific events on the system [23]. Specifically AuditD refers to the audit daemon that communicates messages from the Linux kernel to the user. The specifics of delivery can be configured from `/etc/sysconfig/auditd` and `/etc/auditd.conf` [24]. Audit rules are contained in `/etc/audit.rules`. These rules configure which messages should be delivered from the kernel. AuditD can be configured to transmit a variety of messages. We will focus on two types of rules: actions and watches.

Actions log system calls (syscall), which is how an application interacts with the operating system. Common use cases include requesting disk IO and network resources AuditD can be configured to log syscalls made by the operating system and can be filtered by: the specific syscall, the arguments of the system call, and the file system (if relevant) among other things. The alternate approach to using AuditD is to implement "watches", which logs whenever a "watched" file is accessed. Watches and actions generally accomplish the same goal. Figure 1 shows an example of how AuditD logs an application opening a file.

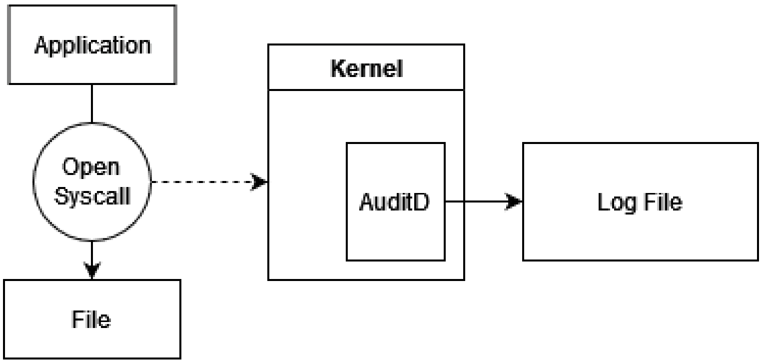

Figure 1: How AuditD Logs File Interactions

Similar to AlarmNet, our proposed system utilizes AuditD, but for a different purpose. We propose the use of AuditD to log file access syscalls, linking them to the application and targeted file. This allows us to determine when files are accessed by multiple applications and use that information to define access controls.

2.4 Data Provenance

Our project shares similar goals, complications, and mechanisms with both DLP and IDS systems: the intent to minimize data loss and the need to efficiently and accurately classify information. However our system does not process information at a data element level. Instead, we focus on the origination and purpose of the data, also known as data provenance. "In its strongest form, data provenance supports information and process integrity by documenting the entities, systems, and processes that operate on and contribute to data of interest. This serves as an unalterable historical record of the data's lifetime and its sources" [25]. Recent research has included applications in operating systems and security [26] [25].

Some of the limitations of data provenance include integrity and storage. Relying on provenance metadata can be risky if the authenticity of the metadata cannot be verified [25]. Additionally, a data set will accumulate metadata throughout its life cycle. It is risky to trim the associated metadata since it is impossible to determine what part of the history will be relevant in the future, [25]. We propose a solution to mitigate this limitation in specific situations by adopting a practice commonly used in data taint analysis and employing a variation of RBAC.

When using data provenance for access controls for data loss protection - it is essential to know what types sensitive data may be at risk within a given data set. We claim that in that situation, knowing the order in which the data set was manipulated is irrelevant - the important information is the potential sum of the data within. For example, there are three data sets: the first containing first names, the second containing last names, and the third containing social security numbers. Separately, these data sets are important but not critical. However, if these data sets were combined it would be considered personable identifiable information (PII)[27]. The order in which these data sets are combined does not matter in this context, only the fact that the data set now contains PII and must be subject to organizational security controls accordingly.

In data taint analysis: when a suspicious data element comes into contact with other elements in the system, those elements are deemed equally suspicious[28]. In a system that cannot determine with certainty whether the suspicious data element compromised another, any interactions must be assumed to be malicious. In that same vein, we propose the use of that doctrine with the interactions of data provenance.

RBAC commonly chosen over ABAC because the functional use of ABAC requires prohibitive overhead [5]. Therefore we propose a system akin to RBAC that employs the data provenance doctrine ascribed previously to dictate access controls throughout an organization. The “roles” represent the security requirements associated with that data set.For example: Payment Card Information (PCI) requires specific security controls and by definition must contain certain data elements [29]. Therefore a data set containing PCI would also carry metadata declaring as such. For the remainder of this paper, we will refer to a data set’s role as its classification. Depending on the setting, users may be required to interact with data sets that have varying classifications. It could be useful to have a system minimizes contamination between classifications but allows a user to access their entire workflow without interruption.

2.5 Virtual Machines and Containers

Virtualization refers to to the technique of creating isolated environments within a physical computer. These isolated environments are usually virtual machines or containers. The virtual machine, also known as a guest, behaves as if it is a physical machine but all the physical components are simulated by the real physical machine (also known as the host) [30]. The program that manages virtual machines is known as a “Virtual Machine monitor” or more commonly, as a “hypervisor.” There are two types of hypervisors: Type 1 hypervisors are “bare-metal” and host the virtual machine directly on physical hardware, Type 2 hypervisors run on top of an operating system. Generally Type 1 hypervisors are faster since there is no host operating system.

Containers function differently from virtual machines. Containers are isolated instances of an operating system that share a kernel [30]. This difference makes containers faster. Often containers are used in situations that require instantiating numerous virtual environments simultaneously. However, containers require a host operating system and the containers must be the same operating system, since kernels are not cross-compatible with other operating systems. Since containers share code, there is always a risk that malicious code within a container will spread to other containers or escalate privileges into the host OS [31].

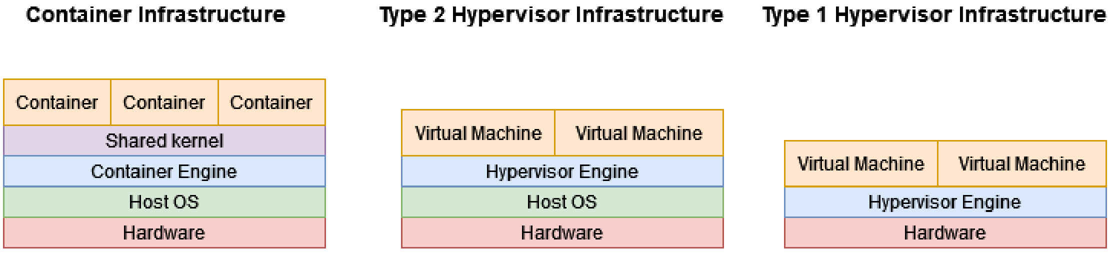

Figure 2: Virtualization Infrastructures

Virtual machines were created to partition IBM mainframes into logical segments and quickly caught on as an efficient mechanism for running multiple applications at the simultaneously [32]. In addition to performance benefits, virtualization offers isolation. In a traditional system, applications on a system can see each other and interact with shared resources. In a virtual system, every virtual instance is cut off from the other, only the virtualization engine can see and interact with the virtual systems. Security researchers leverage this feature constantly. Malware is often examined within virtual machines so that any damage is contained to the virtual machine. Additionally, relying on isolation can improve existing security policy [33].

Because virtualization relies on isolation, if an attacker can break isolation they can leverage their access and infect other virtual machines or the host operating system [32]. Often systems are configured to intentionally break isolation. For example VirtualBox provides the capability of sharing clipboards between machines, this makes it easier to work with, however an attacker could leverage that to spread malicious code or exfiltrate data [34].

Qubes OS is an operating system that leverages the isolation capabilities of a type-1 hypervisor to provide unique security capabilities to the end-user. The goal of Qubes OS is to minimize the impact of insecure applications by isolating insecure activities from critical data. Each isolated container is known as a "qube." Qubes instantiates one qube known as dom 0 that serves as the buffer between other qubes and the hypervisor and acts as a qube manager. Each qube has access to a suite of virtualized devices, each abstracted into a qube itself [35].

Qubes OS has been used as the basis for a variety of systems that can utilize isolation or improve user experience. SAFE-OS is a patented system that builds on Qubes by creating the Dom U, a variant of Dom0, designated for untrusted materials. SAFE-OS also abstracts away the isolation components from the user interface, presenting a unified view to the user [36]. However, Qubes OS is not stable enough for enterprise and only works on specific sets of hardware. The difficulties of installing and configuring Qubes OS may be prohibitive. Our proposal seeks to demonstrate security controls designed around virtualization without the use of Qubes-OS.

Bitvisor is a system developed by a multitude of universities and companies at the behest of the National Information Security Center (NISC) of Japan [37]. This system introduces a hypervisor that provides encryption/decryption for storage devices and network connections. Bitvisor aims to minimize data loss from end-user devices for government and corporate organizations [38]. Specifically, data loss that originates from unauthorized use of USB drives or physical theft of the laptop [39]. Bitvisor is similar to our project in that it leverages virtualization to minimize data ex-filtration.However, we propose a virtualization scheme that focuses on minimizing software and network level data ex-filtration and Bitvisor aims to minimize data ex-filtration through hardware.

Hysolate is a commercial product that isolates end-user machines into a non-persistent "risky zone" and a "secure zone." The "risky zone" enables employees of an organization to browse the internet, handles device IO, and exercises minimal security controls in the name of end-user autonomy. The "secure zone" stores all the organization's critical applications and data. Networking and on-device controls configured by a cloud hosted control panel. Hysolate's features are extremely limited in scope. The virtualized machine can only be a non-persistent version of Windows 10. Therefore, any data saved only the hard drive is lost on reboot and a new Windows 10 image must be requisitioned at start time. Organizations cannot configure an OS image ahead of time. Any changes to the image must occur every time an end-user installs the image. Additionally, the system only creates one boundary on the end-user system. This limits organizational policy to essentially a boolean operation of "is this secure" or "is this untrustworthy", there is no mechanism to enforce unique security controls based on the classification of data [40]. Our project explores the security benefits of using virtualization as an access control boundary, but with persistent hosts and multiple boundaries.

Shamon is a system that utilizes a Type 1 hypervisor, SELinux, and IPsec tunnels to enforce MAC controls on a group of computers on an untrusted network. The goal of this system is to extend the fine-grained control of SELinux and RBAC to a distributed system. Generally, RBAC schemes do not lend themselves well to large networks as they quickly require too many roles to be feasible. This system presents the concept of using virtual machines as a mechanism to simplify RBAC implementations [41]. Shamon allows for the assurance that MAC policies are enforced across systems and that distributed computations can be protected. Shamon assumes that the network between physical machines is untrusted but offers a trustworthy mechanism for communication between virtual machines sharing a host. We share this concept of using isolation to simplify access control models and apply it to a different use case: a system of virtual machines across a trusted network but with the added responsibility of maintaining data provenance throughout the system. Virtual machines in our system interact in the same fashion, regardless the underlying physical machine.

For this paper, we chose to use VirtualBox, a Type 2 hypervisor. We considered both Docker containers as well as Xen and QubesOS, but decided to use VirtualBox. Docker containers provide ease of use as they are small and lightweight, but do not provide the same security benefits and extensibility as a Type 1 or Type 2 hypervisor. We decided against Docker as we wanted to avoid container kernel security vulnerabilities [31]. Also, since OpenVSwitch requires kernel modules, it must be installed on the host and the Docker container must run in privileged mode [42]. Docker privileged mode allows processes inside the docker container to access all devices on the host as if it were root. This can result in security problems if the container were to become compromised [43]. Type 1 hypervisors provide the security benefits of a hypervisor without the performance costs of a Type 2 hypervisor, but they can be more difficult to work with. Since the hypervisor engine is not run on top of an operating system, it can be more difficult to configure. When investigating Type 1 hypervisors, we were working of off QubesOS. Though QubesOS provide immense security benefits over other options, it does not function on all hardware and is difficult to configure and use.

2.6 Software-Defined Networking

Software-Defined Networking (SDN) is a networking paradigm that increases flexibility in network management by abstracting the control system away from vendor-specific technology [44]. SDN provides a centralized programmable platform that can control an unlimited number of networking devices. Each networking device communicates with the SDN controller to receive configuration changes and packet flow control information. The SDN controller also receives analytical traffic data from the networking devices, enabling visibility into the traffic flowing in the network.

Software-Defined Networking is the culmination of three decades worth of research into making computer networks more programmable [45]. Motivated by the desire for fine-grained control, dynamic operation, rapid development of network services, and research experimentation at scale, so called “Active Networks” became the foundation of what we now know as SDN. Much of the promises of Active Networking still apply to SDN; primarily, unified control over networking devices across vendors and models [46].

The most common protocol used for SDN is OpenFlow [45]. The OpenFlow protocol defines a standardized mechanism by which networking devices can be controlled dynamically and programmatically. The rapid growth in popularity of OpenFlow can be attributed to the OpenFlow community’s focus on backwards-compatibility with existing protocols and technologies. OpenFlow-enabled switches store one or more tables of packet-handling rules that define how the switch should handle a packet that matches the rule. When there are no rules defined the switch requests a flow decision from the controller [47]. Upon receiving such a decision, the switch stores the rule in its table and handles the packet accordingly.

Standard SDN implementations with OpenFlow focus on controlling the networking hardware, primarily switches, that handle network traffic [48]. These implementations are simple to deploy, but only allow a coarse level of control over the network traffic. Host-based SDN extends the traditional SDN to include control over each host that is connected to the network [49]. A host-based SDN implementation allows much finer control over what network traffic is allowed or denied based on what application process is operating on the network traffic, which user executed the application, as well as which device the traffic is originating from [48]. Host-based SDN is implemented by deploying an agent to each endpoint device that will connect to the network. The endpoint-host routes all communications through the agent, enabling the agent to report on the traffic being sent and the applications or process that are sending it. Since our design relies on OpenFlow to regulate network connectivity between clients, we must be able to assume that OpenFlow is trustworthy.

One tool that assists in deploying SDN is FRESCO. FRESCO is "an OpenFlow security application development framework designed to facilitate the rapid design, and modular composition of OF-enabled detection and mitigation modules" [50]. FRESCO provides a set of simple software libraries that enable network and security administrators to build custom network protection applications. With FRESCO, administrators can build firewalls, filters, detection systems, and other network protection applications. FRESCO integrates with the popular NOX OpenFlow controller to provide a Development Environment and a Resource Controller. Since we rely on OpenFlow, we need it to be trustworthy. The existence of FRESCO as a means to create security tools helps demonstrate OpenFlow as trustworthy and reliable.

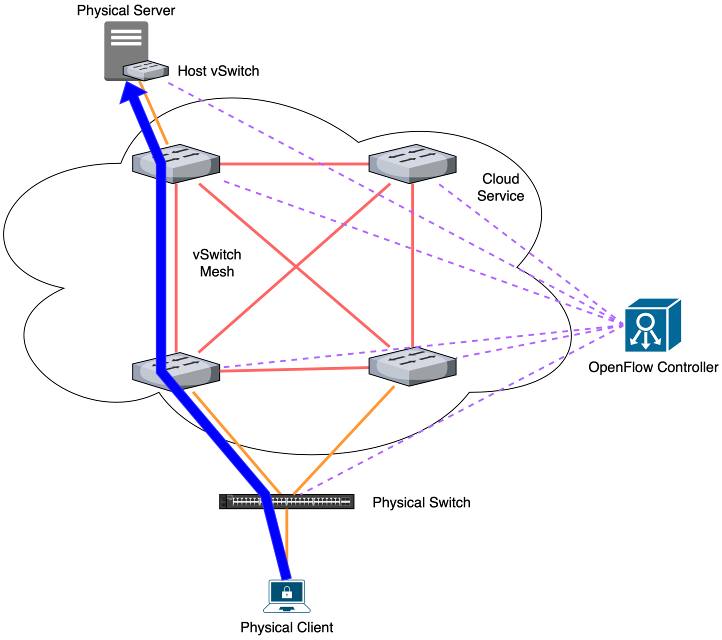

Figure 3: Architecture of Scotch overlay network. The vSwitch Mesh acts as an overflow buffer for when the physical switch's connection to the OpenFlow Controller gets saturated. Each vSwitch is also configured to overflow to one of its neighbors. vSwitches can dynamically be added and removed by the cloud service

A similar research project on SDNs is Scotch. The Scotch research team endeavored to develop a solution to dynamically scale the capacity of the control plane's path throughput [51]. They accomplish this by forwarding packets from the overloaded phyical switches to virtual switches which have non-congested control-plane paths to the SDN controller. The mesh of virtual switches acts as backup for the physical switches, allowing them to offload to the vSwitch mesh and still ensure the packets will be routed correctly.

This solution helps prevent DOS attacks which could exploit the limited bandwidth capacity of the physical switch's control plane, which typically has a smaller bandwidth than the data plane. The solution also supports more specific OpenFlow flows, which can be leveraged to provide a finer-grain access control scheme, that require more bandwidth to direct traffic. At scale, our project would likely benefit from an implementation of the Scotch project, as our control plane will likely become saturated with requests for each process.

Another research team worked to improve the Scotch project, and in the process created SDNShield. The SDNShield project provides a solution to defend against the potential DDOS vulnerabilities of the OpenFlow control plane [52]. SDNShield works by filtering traffic, to reduce congestion and remove malicious packets. By filtering packets on the control plane, SDNShield is able to prevent edge switches and the SDN controller(s) from being overwhelmed by control traffic. SDNShield utilizes a similar architecture to Scotch (shown in Figure 3), but adds a set of virtual machines running filtering algorithms connected to each virtual switch. These filtering algorithms are used to reduce the amount of malicious traffic flowing on the network. For any network that is planning to scale, SDNShield is a necessary component to safeguard the core networking infrastructure from malicious actors as well as inevitable device failure. A real life implementation of our design would require an SDNShield-like architecture to overcome vanilla OpenFlow's traditional availability concerns.

3 Design

In this Section we describe the design of our infrastructure explaining our intentions, decisions, and assumptions. We complement the information with diagrams to elucidate our infrastructure design. We created a proof of concept for our design, details for which can be found in Section 4.

Since security goals often motivate organizations to limit user autonomy on corporate devices, this paper blueprints a system that leverages data provenance to maximise user autonomy without sacrificing confidentiality, integrity, or availability. Users within an organization would have machines capable of virtualization, a suite of virtual machines specified to a purpose, and a interactive experience that abstracts away the complexities of working with multiple virtual machines simultaneously (inspired by Qubes). For the remainder of this paper, virtual machines will be referred to as clients. Each client should have a specific use case, can only store data required to fulfill that goal, and has security controls proportional to the criticality of the data within. For our purposes, it is irrelevant whether the clients share a physical system or not.

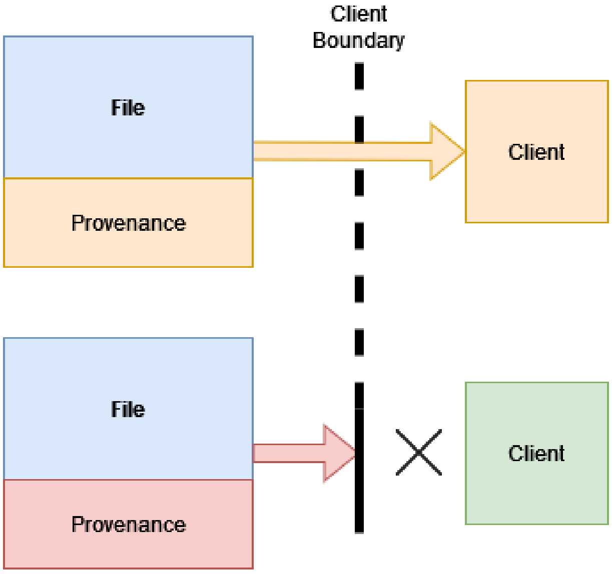

Figure 4: File crosses client boundary and is allowed or blocked dependent on provenance.

This system enforces two sets of boundaries: between applications and between clients. Each client will have a administrative service known as the Provenance Updater monitoring file access events. When data crosses the application boundary, we store data related to its data provenance. The Provenance Communicator is a similar administrative service monitors network activity and supplies relevant provenance metadata to a centralized decision-making entity when that data crosses the client boundary. The decision making entity decides if the data is allowed to cross the boundary or not as shown in Figure 4. When data is allowed to transition between clients, the appropriate provenance metadata is communicated from the decision making entity to the receiving client's Provenance Communicator to ensure consistency across clients. The client boundary is enforced by network switches completely invisible to the client as shown in Figure 13.

When data attempts to cross a boundary, its permission to do so is verified by access control policies configured by the organization. The contributing factors to the decision to pass through a client boundary is the destination client and the provenance of the data itself as demonstrated in Figure 4.

When data passes through the application boundary, its provenance may be updated to reflect the change in sensitivity for the data within the file. As shown in Figure 5, the new provenance is dependent on the which application's boundary is being crossed. We assume that any given interaction transmits all potential data available to that application during its lifetime to the file that it is interacting with.

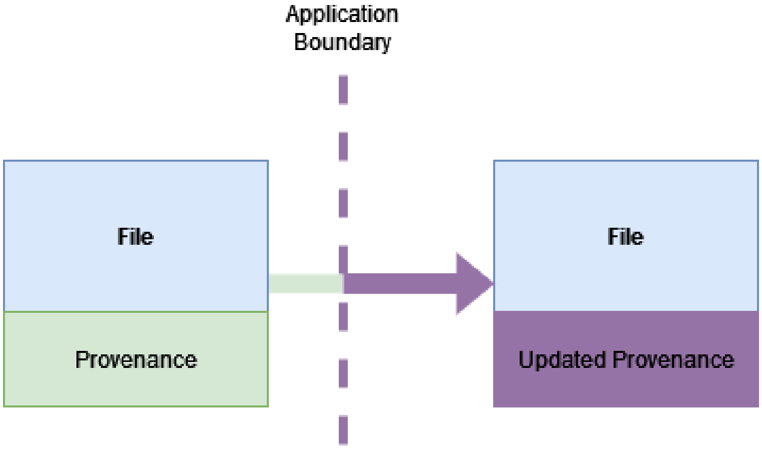

Figure 5: File crosses application boundary and its provenance is updated.

This system requires visibility into clients' in order to track provenance transitions at the application barrier. For this purpose we suggest adopting techniques common in host based intrusion detection systems and data loss prevention tools to detect when the contents of a file require an update to its provenance. We suggest the use of AuditD systems to detect boundary crossings. AuditD can provide file access events, giving us the ability to know every time a file is accessed and by which application - which is enough information to determine if and how the application boundary has been crossed. We also propose the use of Linux's Extended File Attribute system to maintain persistence of provenance metadata. It is possible that other mechanisms are better suited to the task, in Section 6.2 we discuss the benefits and limitations of our chosen tools.

To meet our confidentiality expectations we rely on the trustworthiness of the Provenance Updater and the Provenance Communicator. The principle of least privilege [53] requires that an application only access the resources that are required for its function. Following that principle: all applications with administrative access, have it because it is required for that application to function. The Provenance Updater and communicator are set administrative services on every client because they require administrative access to modify file provenance data. But they do no not have the unnecessary power of a driver or kernel module. Relying on the principle of least privilege allows us to assume that these services are trustworthy. The untrusted zone is the user space on an end user machine, which we have segmented into smaller parts using virtualization. This segmentation reduces the impact if one of the untrusted zones is compromised.

The next question is: How do we create a centrally-managed boundary between every client on any number of physical hosts? We suggest taking advantage of the benefits of programmable networks such as SDN. With SDN we have centrally managed and dynamically modifiable control over client agents (acting as firewalls) to reflect the organization's access control policy. All client agents rely on a centrally located SDN controller to configure rules that dictate how to manage communications with other clients. This system ensures that all communications are subject to organization access controls as shown in Figure 13.

The integrity and availability of the system rely upon trusted network communication between the Provenance Communicator and the SDN controller as facilitated by SDN endpoints and OpenFlow. OpenFlow is known to be susceptible to DDOS [52], but technologies such as Scotch and SDNShield that OpenFlow can be trustworthy if modified. Further, the ability to easily create security tooling through FRESCO demonstrates that the OpenFlow environment is robust enough to be trusted in a system with high confidentiality and integrity requirements.

3.1 Policy Manager

Given that we have the ability to dynamically control the network activity of all clients from a single controller, how do we decide which boundary crossings to allow and which to prevent? We propose the use of an access control matrix to dictate allowed transitions. The component that contains this matrix would be called the Policy Manager. Due to the early graduation of one of the members of the team, development of the Policy Manager is not completed. Future iterations of the paper will have integrated a Policy Manager into the SDN controller, allowing for fine-grained access control and ensured confidentiality.

In the next Section we will describe the technical details of our design and walk through how we built a proof of concept.

4 Implementation

In this Section we describe the design of our system by explaining each component in detail and expanding on how the components interact. We provide flow charts and diagrams to accompany each component.

Our goal was to design a system that enabled centralized, fine-grained access control based on data provenance. In service of that goal, we built a proof-of-concept from open source components and leveraging features of the Linux operating system.

4.1 Virtualization Scheme

This proof of concept leverages VirtualBox, a Type 2 hypervisor that supports the creation of multiple concurrent virtual machines. We configured a Debian Linux virtual machine with the appropriate settings and installed/developed the software required to maintain our proposed system. Modifications are described in the next subsection. We used VirtualBox's import/export functionality to create copies of our virtual machine. Throughout this Section, all references to "virtual machines", "containers", and "client" refer to a virtual machine that used this image.

This system relies on the ability for each user to support a suite of clients, each configured for a specific use case. The virtualization design assumes that: the clients cannot communicate except through monitored network channels, that network communication is managed by SDN flows, each clients is running security software, a configured vSwitch Additionally, we created two systems to be installed on all clients: The Provenance Updater, and the Provenance Communicator. Both of these rely on AuditD logs and we created a program to interpret those logs, turn them into event classes, and call the relevant provenance tool as shown in Figure 6.

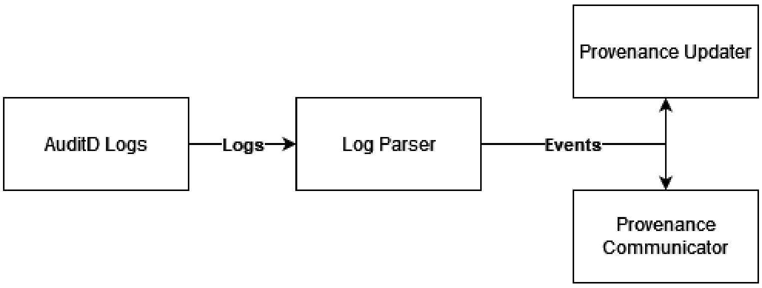

Figure 6: Provenance Updater and Provenance Communicator Flow Chart

4.2 Provenance Updater

We created the Provenance Updater to track when an file crosses the application boundary. We do this by processing AuditD logs to track when an application accessed a file. As shown in Figure 7, When a file is accessed, we update its provenance to reflect the potential changes in content. Since applications will interact with many files throughout their life cycles, application boundaries are stored in memory and are also updated on file access. When a file crosses the network boundary and, because it is being sent by a program, it will trigger the Provenance Updater. In those situations, the Provenance Updater communicates with the program handling the inter-client boundary called the Provenance Updater to ensure that any incoming data is correctly tagged.

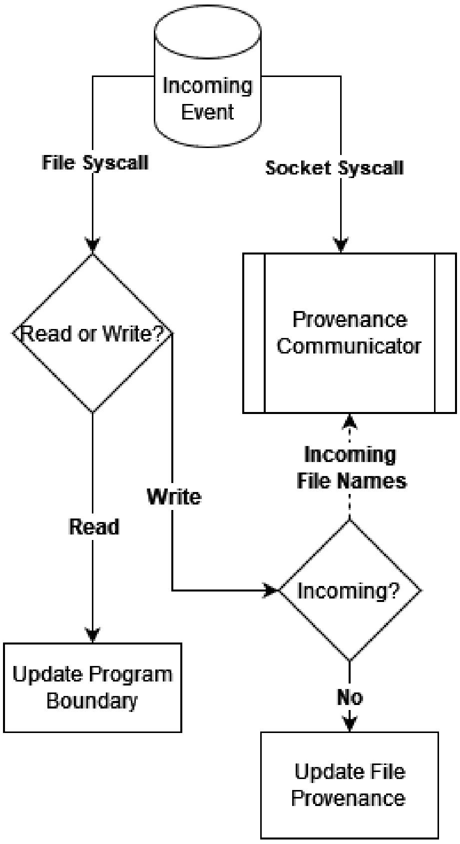

Figure 7: How Provenance Updater Handles Events

File provenance is stored using the Linux Extended File System, which allows users and applications to attach arbitrary data to files in the same manner that the operating system assigns tags such as “Date Created” and “Date Last Modified.” Once set, a file is linked to its provenance as long as it stays on that client, when a file is transmitted to another client the Provenance Communicator interacts with the recipient client’s Provenance Communicator to maintain integrity.

As shown in Figure 7, the Provenance Updater acts on incoming events. These events are instantiated from AuditD’s logs via Python script. The functionality of the Python script is visualized in Figure 8. With a fully configured AuditD, an event will generate multiple logs that are can be tied together by a unique identifier (UID). Regardless of the event, AuditD will issue a log messaged “PROCTITLE” when an event is done being logged. The log processor parses log messages, storing relevant data by UID, until it reaches the “PROCTITLE” message. When a “PROCTITLE” occurs, the event associated with that UID is sent to the appropriate next party.

It is essential to properly configure the AuditD logs to properly log events of interest without

overwhelming the system. We used auditctl, a command line tool, to configure AuditD. Action rules generate “SYSCALL” logs detailing which syscall was used and which process called it. To create a rule use: `auditctl -a always, exit`. This tells AuditD to always log syscalls on their exit. There are other options for -a but thye are not important to understand this paper. To prevent the volume of log entries from overwhelming the system, each rule has a filter that only tracks events that occur in the home directory or subdirectories.The -F parameter allows for filtering action logs by the argument. For example: `-F dir=/home/user` would limit logging to syscalls that impacted a file within the home directory or a subdirectory of home. -S allows filtering based on specific syscalls; `-S connect` only logs the ‘connect’ syscall. This system relies on the following action rule:
`auditctl -a always,exit -F dir=/home/user -F perm=wa` which logs all syscalls within the user home directory.

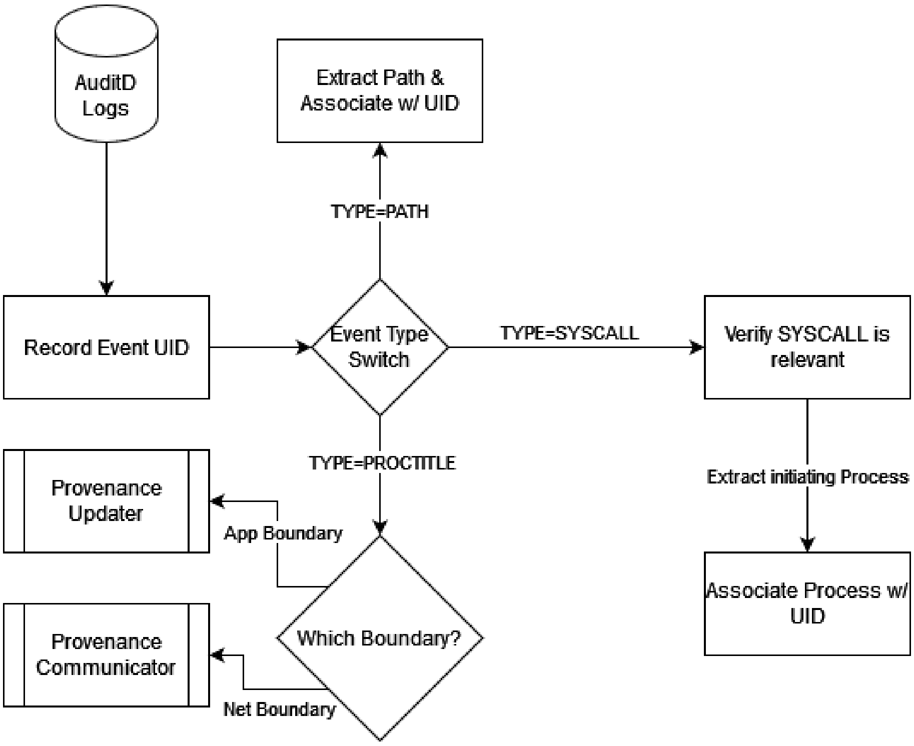

Figure 8: Processing AuditD Logs into Events

Watch rules generate “PATH” logs that communicate when and which file has been accessed with a matching Linux permission (read, write, executable, etc). Watch rules are created with auditctl’s -w option, with the directory as an argument. For example: `auditctl -w /home/user` will create a watch in the home directory. watch only takes two optional arguments: -p and -k. -k functionality is the same as in action rules. -p specifies which permissions to watch for. `-p w` sets a watch for any writes to a file. This project uses two watches:
`sudo auditctl -w /home/user -p w -k process_monitor` and
`sudo auditctl -w /home/user -p r -k process_monitor`. They are identical except one watch is for writes and the other uses `-w r` to watch for reads.

Employing watches and action rules provides enough data to the Provenance Updater to detect when a file crosses the application boundary. A drawback to using logging as the mechanisms for enforcing a boundary is that we only know about events after they happen and therefore cannot take preventative action. Further discussion on the limitations of retroactive enforcement can be seen in Section 6.2.

4.3 Provenance Communicator

We created the Provenance Communicator to transmit file provenance to other clients and the Policy Manager.

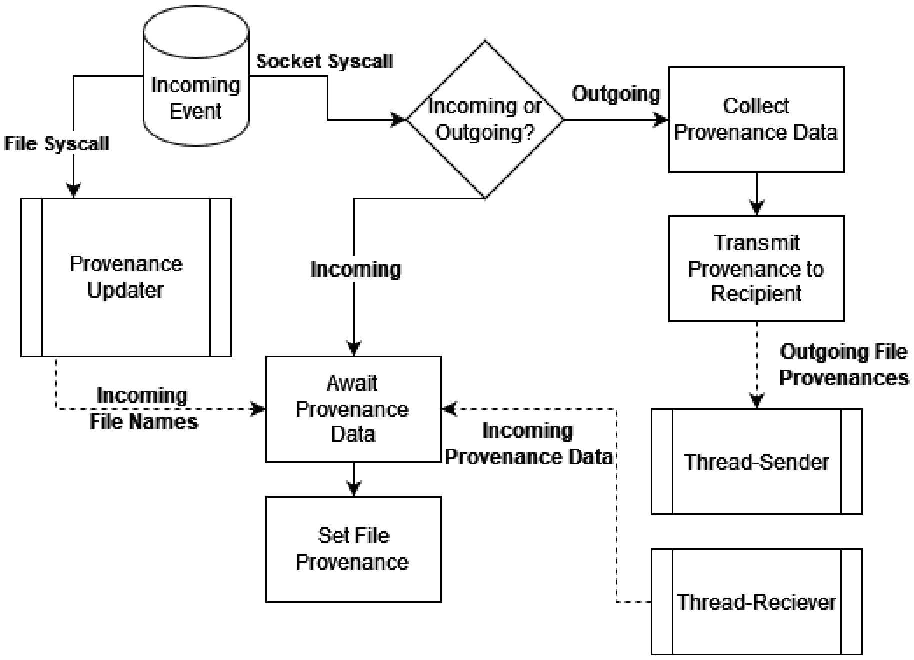

Figure 9: How the Provenance Communicator Handles Events

As shown in Figure 9 The Provenance Communicator uses AuditD to detect when a process opens a socket. AuditD is configured with the following rule: `auditctl -a always,exit -F arch=b64 -S connect -S bind -S socket -S accept -k socket_monitor`

When the socket is opened, the Provenance Communicator transmits the file provenance of all the potential files that may be sent through that connection. The SDN controller then communicates that information to the receiving party, which has their own Provenance Communicator installed. The Provenance Communicator also listens for incoming provenance metadata from the controller, and assigns the appropriate provenance to files as they are transmitted through the socket.

When the client and server are initialized, the client first forms a connection with the server on port 1338 (by default). This connection will later be used to send data to the client. The socket_monitor script parses logs from AuditD to capture when a socket is opened. The socket_monitor script also keeps track of files opened by processes. When a socket is opened by a process, the socket_monitor script sends all files the process opened, along with their provenance levels and the source and destination IP addresses and ports, to the client script. The client script uses two background threads to manipulate, parse, and send and receive data to and from the server in the background without disrupting AuditD log parsing. After data is sent to the client script, the sender thread will parse and send data to the server over port 1337 (by default).

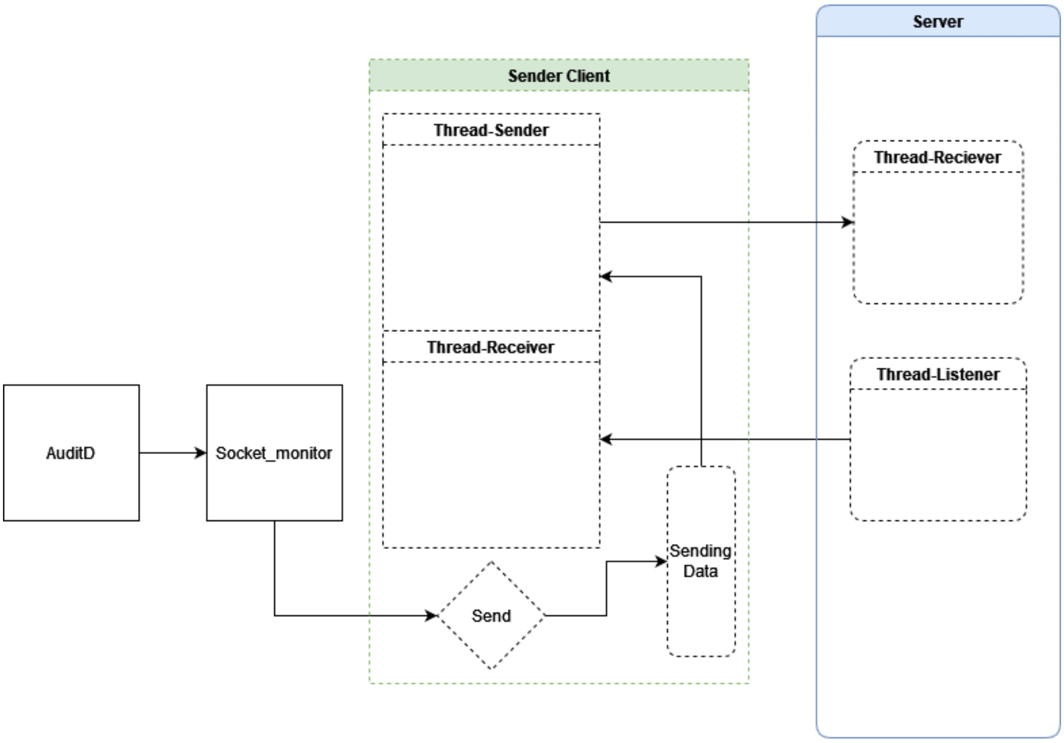

Figure 10: Sender Provenance Communicator Flow Diagram

The server contains three threads. One thread for sending data to clients, one thread from receiving data from clients, and one thread for initializing the connections on port 1338. This model allows for the fewest amount of bottlenecks while processing data. After the receiving thread receives data from the client, it will store the data into a location that the Policy Manager can access. The Policy Manager can then poll the server script and make decisions on whether or not it will accept the traffic. If it accepts the traffic, the server script will tell the sender thread to forward the data to the receiving client. If it rejects the client, the server script will simply discard the data.

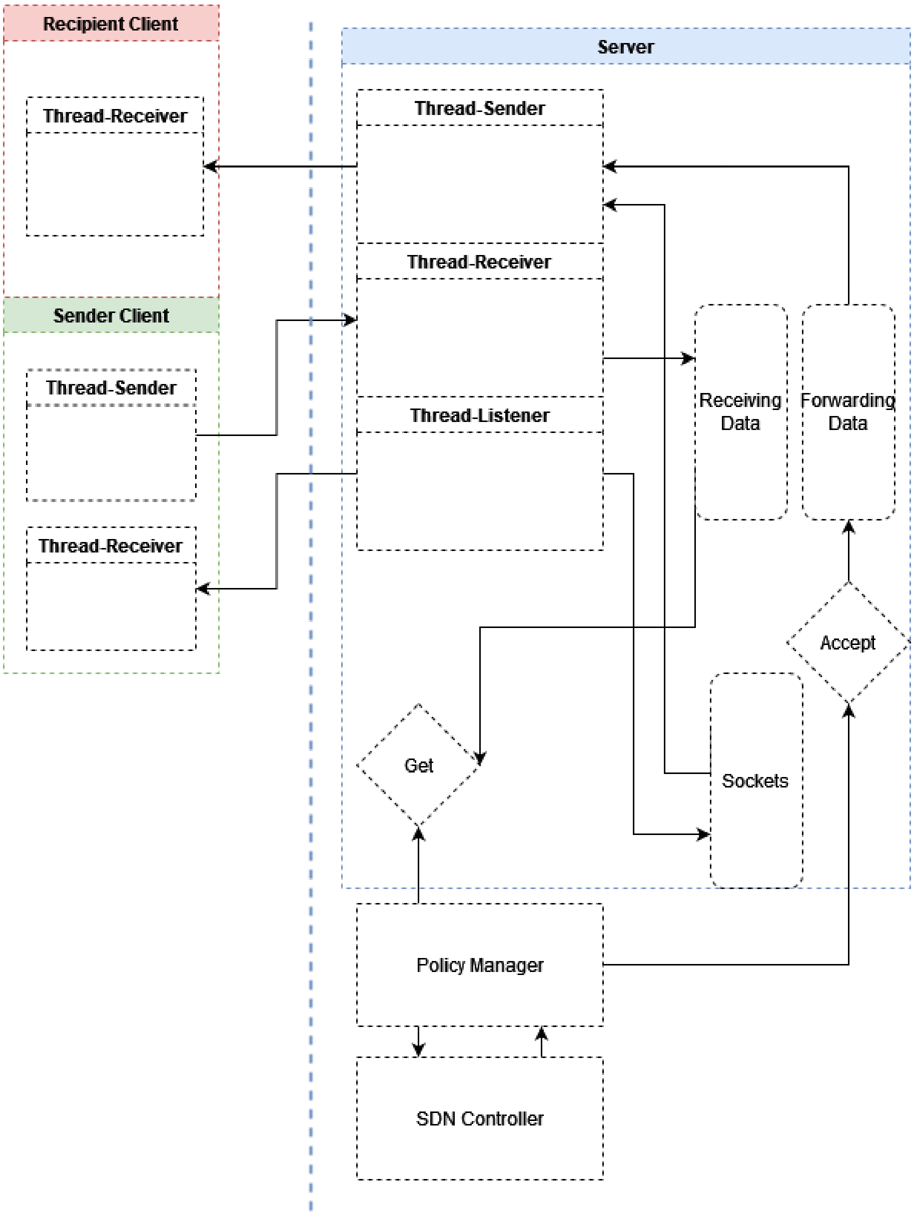

Figure 11: Server Provenance Communicator Flow Diagram

When the receiving client receives the data, it will parse it and send to the socket_manager script. This script will find the received files and edit their labels according to the sensitivity labels that were provided by the server.

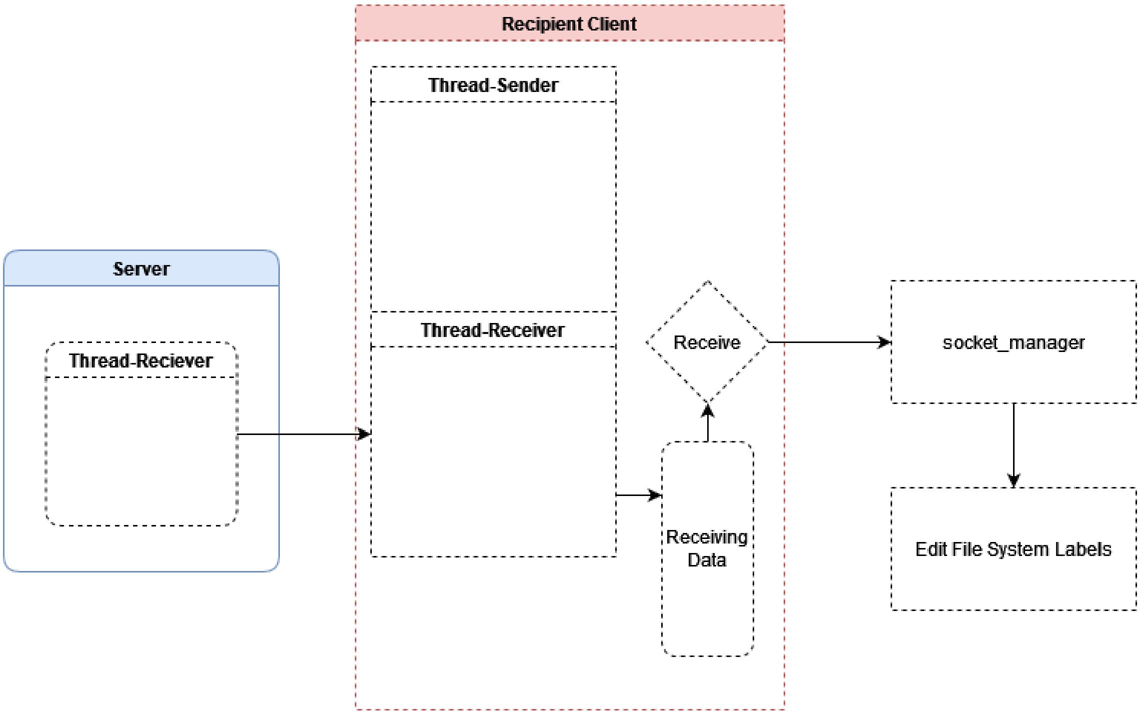

Figure 12: Recipient Provenance Communicator Flow Diagram

4.4 Networking Control and SDN

We implemented a programmatic network control system using two simple components; SDN agents, and an SDN controller.

4.4.1 SDN Agent

Every client has a virtual switch associated with it, also known as an SDN agent. This agent handles all network traffic from its corresponding machine, and enforces network controls communicated from the SDN controller. The SDN agent abstracts the network policy away from the client (see Figure 13). Our SDN agent of choice for this project was Open vSwitch, as it is the most popular OpenFlow virtual switch [54]. We configured the switch to communicate with our controller (see Section 4.4.2) to receive flow rules. Configuration is minimal, but followed these steps:

1. Install Open vSwitch (OVS):
   ```
   sudo apt-get install openvswitch-switch openvswitch-common
   ```
2. Ensure OVS is running:
   ```
   sudo /usr/share/openvswitch/scripts/ovs-ctl start
   ```
3. Add a bridge to the switch:
   ```
   sudo ovs-vsctl add-br main2
   ```

4. Add the primary NIC to the switch as a port:
   `sudo ovs-vsctl add-port main2 enp0s3`

5. Set the controller for the switch:
   `sudo ovs-vsctl set-controller main2 tcp:{ip address}:{port, typically 6653}`

6. Remove the ip address assigned to the NIC:
   `sudo ip a flush dev enp0s3`
   Note: This command will likely make Debian believe that it does not have access to the Internet. Even though an alert may pop up, the client has connection to the Internet after running the rest of the commands

7. Set the bridge as the active NIC:
   `sudo ip link set main2 up`

8. Set the DHCP client to use the bridge NIC to obtain a new IP:
   `sudo dhclient main2`

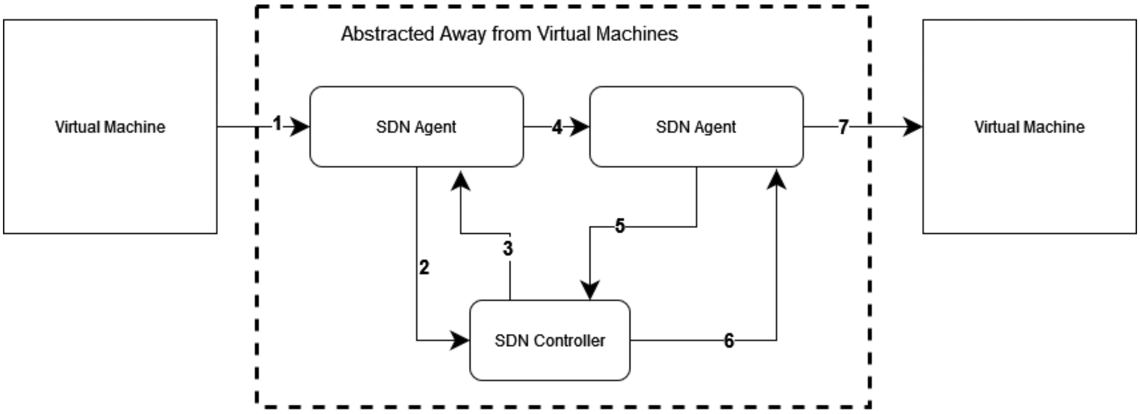

Figure 13: SDN Agent Flow

4.4.2 SDN Controller

The SDN controller manages all communications between clients and contains network policy elements that define how source and destination environments can communicate. The SDN controller can manage communication between two containers of the same compliance, different but compatible compliance, or between a clients and an unmanaged environment, i.e., the Internet. Policy is abstracted out to a separate entity known as the Policy Manager (See Section 3.1). The SDN controller we chose was RYU, due to it's friendliness for researchers as it is simple to use and modify, and it is written in Python [55]. Configuration of the RYU controller was as follows:

1. Install RYU into the user's home directory on a dedicated server

   ```
   git clone git://github.com/osrg/ryu.git
   cd ryu; Python3 ./setup.py install
   ```

2. Configure the controller. A template to modify can be found at:

   ```
   ryu/ryu/app/
   ```

   We recommend simple_switch_13.py as it supports OpenFlow v1.3

3. Navigate back to the main folder and run install again:

   ```
   cd ~/ryu; Python2 ./setup.py install
   ```

4. Run the SDN controller with the now-modified configuration:

   ```
   ryu-manager ryu.app.simple_switch_13
   ```

For the purposes of this paper, the SDN controller was left to the default configuration of simple_switch_13.py; that is, a simple L2 switch. In future works, we plan to integrate the SDN controller with the Policy Manager (see Section 3.1) to provide dynamic control over which connections are allowed or denied.

5 Results

To test the feasibility of the system, we ran several stress tests, latency tests, and modified the system to record relevant statistics. Our stress testing was performed by executing a Python script that caused files to rapidly cross the application barrier and the client barrier. During that stress test we recorded statistics and observed the behavior of the Provenance Updater, Provenance Communicator, and an unfinished version of the Policy Manager configured to allow all connections.

5.1 Stress Test

The stress test was performed by a Python script that created and modified a preset number of files in a given amount of time in order to generate many events in short periods of time. The script generated up to hundreds of events a second forcing the Provenance Updater to handle dramatic changes in the amount of events to process at once. The script also caused files to cross the client boundary interacting with the central server and other clients. We did not detect any server bottleneck. The server and SDN controller could handle transmitting provenance metadata between multiple clients.

5.2 AuditD Processing Overhead

We analyzed the latency of how long it took our system to respond to an event. We recorded the exact time of the event as reported by AuditD and recorded the exact time that we finished processing the event. We compiled that latency by how many events were occurring at the time. The goal of this analysis is to determine if there is a relationship between the frequency of events and latency. Essentially, does our system lag behind when many events happen simultaneously?

Figure 14 shows that the Provenance Updater can take up to 20 seconds to process events. There is a distinct trend line between the frequency of events and the processing time. It is evident that processing AuditD logs introduces latency, but there is not enough data to prove that the relationship is causal. We hypothesized that additional latency is introduced when many files cross the network boundary at the same time.

To investigate that hypothesis, we recorded latency in the same fashion as the last trial. The latency was measured against the number of files crossing the network boundary as a result of the event.

As seen in Figure 15, the overhead quickly increases as the number of files crossing the boundary increases, but not consistently. There appears to be a trend, but the data set is unevenly skewed. There are relatively few instances of the Provenance Communicator transmitting many bytes at once. In the future, additional trials with balanced samples could and allow for stronger conclusions.

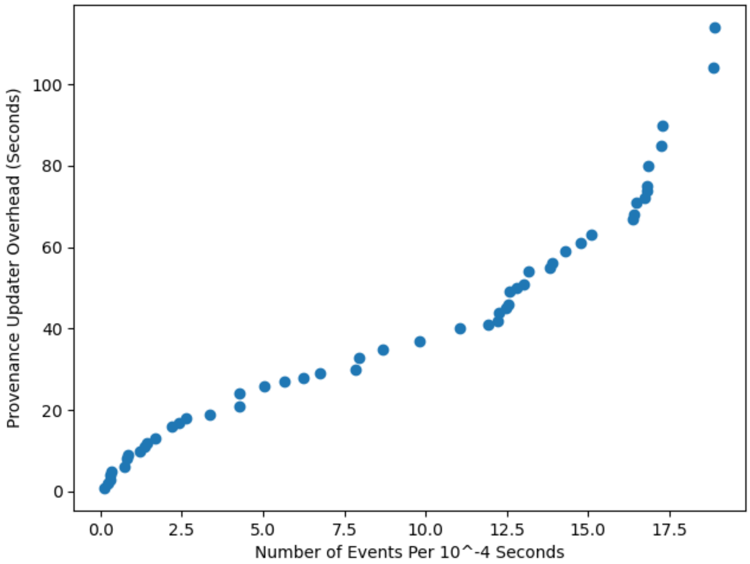

Figure 14: Overhead of processing AuditD logs

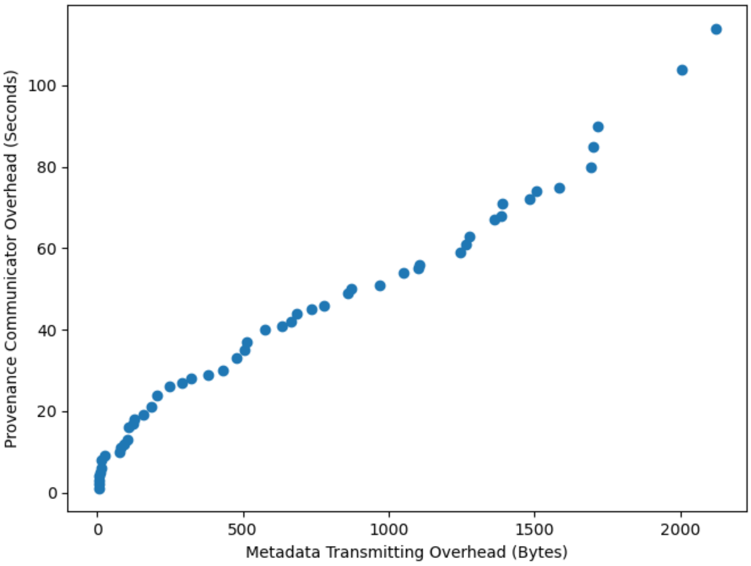

Figure 15: Internal Latency of Transmitting File Metadata

5.3 Provenance Communicator Overhead

A system that adds overhead to every network connection must be relatively lightweight since the overhead will be applied many times and may exacerbate network congestion or overload servers. We modified the system to report the amount of bytes that had to be sent to communicate file provenance data to other clients. We recorded that data in conjunction with the amount of files who's provenance metadata had to be transmitted.

As seen in Figure 16, there is a very noticeable consistent relationship between the amount of bytes transmitted and the number of files sent, with the largest transmissions approaching 2,000 bytes. As a result of our reliance on the AuditD system, the size of the transmission is directly proportional to the application sending data and how many files have crossed that application's boundary during its lifetime. As shown in Figure 17, the majority of transmissions were smaller than 750 bytes.

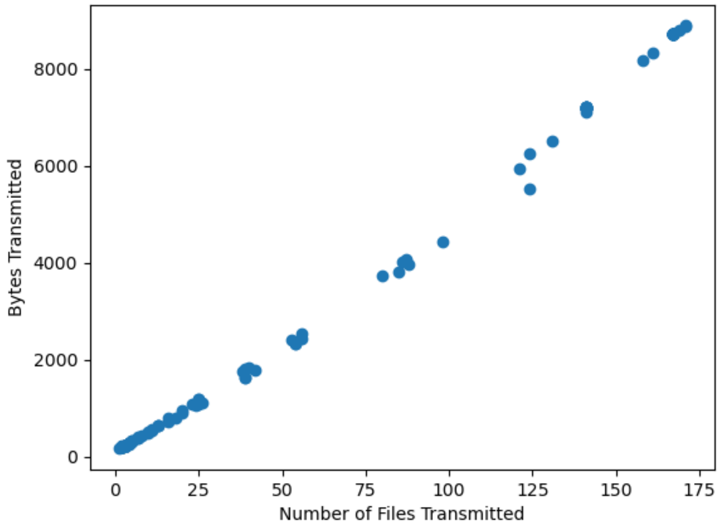

Figure 16: Network Overhead of Transmitting File Metadata

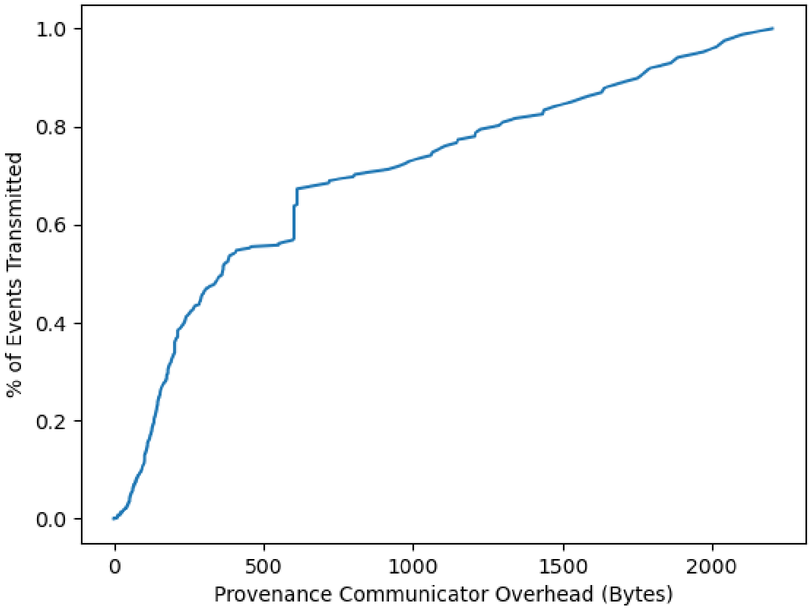

Figure 17: Network Overhead of Transmitting File Metadata

We noticed that web browsers generate the most traffic. Our system ran Firefox and we noticed that Firefox frequently makes outgoing connections and constantly interacts with operating system. Since web browsing is one of the most common use cases for a computer, we can expect that any implementation of this system would need to, at the minimum, handle the amount of events generated by a program like Firefox. Even at its worst, Firefox only required the transmission of 175 file provenances with a negligible overhead of around 2,000. Modern devices and networks can trivially handle that level of overhead, and we detected no significant performance penalties related to the transmission of file provenance.

6 Discussion

This Section is dedicated to elements of our project that are not evaluated in experiments, but nonetheless may be insightful for individuals interested in continuing this research. We discuss limitations of our work, what we learned during the development of this project that may be useful to others, and ideas on how to extend this research.

6.1 SELinux

While developing this project, we encountered many difficulties with SELinux, QubesOS, VirtualBox, and AuditD. After switching away from QubesOS when prototyping our initial ideas, we switched to Linux Mint VirtualBox VMs. At this point, we began to use SELinux as a replacement for MAC enforcement in QubesOS. This combination of technologies caused problems as SELinux is not entirely supported on Linux Mint and other Ubuntu based distributions. In particular, we discovered that Linux users could not be assigned to SELinux users. This was problematic as this is critical functionality in SELinux. After noticing this problem, we switched to Debian VMs, as Debian explicitly does support SELinux. Though our previous problem was solved, we quickly discovered that the 'se-troubleshoot' and 'se-troubleshoot-server' packages were not available to be installed on Debian, only on Red Hat Enterprise Linux (RHEL). These packages are essential to troubleshooting issues with SELinux as they provide human-readable explanations of SELinux errors. After realizing this, we finally switched to RHEL to investigate SELinux configurations.

SELinux's targeted policy mode does not easily support the type of access control needed for our system. We attempted to create an SELinux user for a Privileged Access Workstation (PAW) model as the SELinux reference policy did not have a user suitable for this purpose. This user would have extremely restricted access to the file system and the network to create a user that can absolutely perform only the tasks it needs to and nothing more. This turned out to be difficult with SELinux. Typically, SELinux policies are configured using booleans (policies built into the SELinux reference policy than can be turned on and off), file labeling, and local policies. Allowing a specific user to access only a small set of files would require significant relabeling of the file system as well as a custom policy. It is important to ensure all system critical executables and files are still accessible by the restricted user. A final important note with SELinux is that SELinux users and roles are not used by targeted policy by default. We spent a significant amount of time trying to use SELinux users to accomplish our goals while reading misleading and confusing documentation relating to them.

To implement MAC enforcement in the future, SELinux or AppArmor should be investigated more to create an explicit policy for different data protection models. SELinux can be more powerful than AppAmor, but AppArmor is simpler to create manual policies.

6.2 AuditD

Throughout the creation of our proof of concept, we wanted to determine the usefulness of AuditD in creating access control boundaries. As documented in Section 4, AuditD can be configured to provide suitable insight into a system to create application boundaries and transmit data required for client boundary transitions. However, since AuditD logged events after they happened, our implementation could only enforce access controls retroactively which allows for any manner of race condition and inconsistent behavior. Especially since, as shown in Section 5.2, relying on AuditD cannot guarantee the speed desired to reliably maintain accurate data provenance. Future implementations of this system should investigate alternate methods, we suggest modifying System calls as described in the UC4Win paper [15] to ensure timely detection and the ability to proactively enforce access controls.

6.3 Develop MAC Policy on Clients

To improve the on-device security controls, developing and implementing MAC security policies for different types of clients (such as a PAW, Payment Card Industry (PCI) compliance, and Personally Identifiable Information (PII) compliance) is required. This could be implemented with SELinux or an alternative such as AppArmor[56]. These MAC policies can ensure that data is being handled in an allowed manner on the device, as with the current system policies are only enforced when transmitting data over a network.

6.4 Integrate SDN as Access Control Enforcement

Currently, the SDN controller automatically approves all flows to the requested destination. In order to use the SDN controller as an access control enforcement mechanism, the SDN controller will have to integrate with a Policy Manager (see Section 3.1). As an initial goal, the Policy Manager will combine the request from the SDN controller with a matching request from the Policy Communicator (see Section 4.3), and determine if the communication is allowed based on the access control matrix. If the flow is allowed, the SDN will send the appropriate flow to the requesting switch, and the Policy Manager will send the file provenance information to the destination client (if that client is registered with the system). If the flow is not allowed, the SDN will send a flow to the switch denying the communication, and the Policy Manager will not send any information.

Another area in which to improve this work would be to modify the SDN agent running on each client to send the file provenance information along with the flow request to the SDN controller (by modifying the OpenFlow packet, as described in [47]). Sending the file provenance information in the OpenFlow packet removes potential blocking behavior in the Policy Manager as it awaits a second packet containing the provenance information for the flow request (see Figure 18). We expect that this modification would greatly improve the reliability, integrity, and performance of the Provenance Communicator.

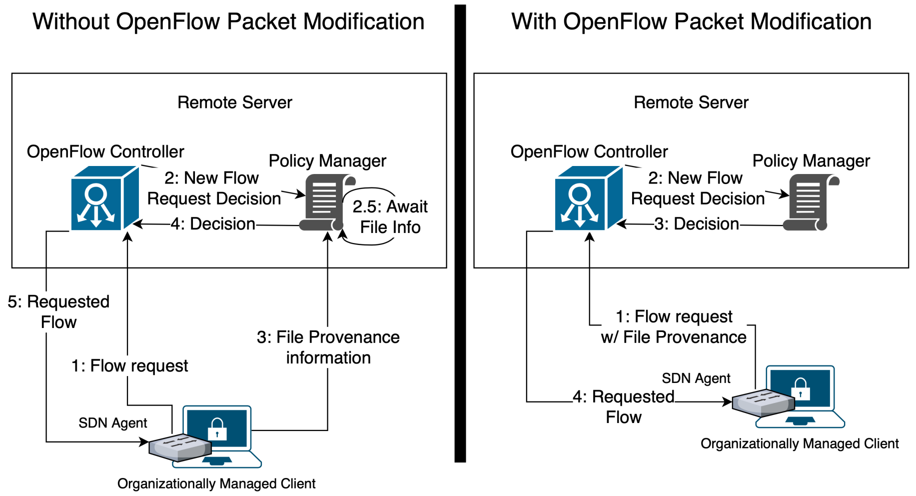

Figure 18: Modifying the SDN agent to send a modified OpenFlow packet removes a blocking step, a second packet, and likely will improve integrity, reliability, and performance

6.5 Conclusion

In this paper we created a fine-grained access control system based on data provenance and outlined how to implement such a system. We built a proof-of-concept and benchmarked the data provenance tracking systems. The provenance tracking system was accompanied by a virtualization scheme designed to isolate trusted zones connected through a software defined network.

Our network configuration provides increased insight into each software process that sends data across the network. Our system correlates file transfers across containers to the originating processes in order to accurately mark and classify files on recipient containers for appropriate data provenance tracking. We collected and analyzed test data, and concluded that the addition of basic metadata into the flow protocol did not significantly impact the performance of the network.

This paper is a snapshot of our progress thus far, as one of the members of this project is graduating two-thirds of the way through the project. The remaining members will continue developing the proof of concept to expand upon the use of the software defined network as a mechanism for enforcing access control policies. In doing this research, we observed several areas in which future work can be conducted; including process monitoring improvements, on-device policy management, policy as a means for access control, policy enforcement with the SDN controller, boundary precision, and network stack complexity. More information about these potential future research areas are outlined above.

7 References

[1] B. Stackpole, "Symantec security summary – february 2021." [Online]. Available: http://symantec-enterprise-blogs.security.com/blogs/feature-stories/symantec-security-summary-february-2021

[2] Hysolate, *How Chief Information Security Officers Are Balancing Enterprise Endpoint Security and Worker Productivity in Response to COVID-19*, Oct 2020. [Online]. Available: https://go.hysolate.com/hubfs/Content/Survey%20Report%20The%20CISOs%20Dilemma%20Oct.%202020.pdf

[3] D. Bell, "Looking back at the bell-la padula model," in *21st Annual Computer Security Applications Conference (ACSAC'05)*, Dec 2005, p. 15 pp. – 351.

[4] M. Nyanchama and S. Osborn, *Modeling Mandatory Access Control in Role-Based Security Systems*, ser. IFIP Advances in Information and Communication Technology. Springer US, 1996, p. 129–144. [Online]. Available: https://doi.org/10.1007/978-0-387-34932-9_9

[5] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. National Institute of Standards and Technology, Jan 2014. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf

[6] M. Su, F. Li, Z. Tang, Y. Yu, and B. Zhou, "An action-based fine-grained access control mechanism for structured documents and its application," *TheScientificWorldJournal*, vol. 2014, p. 232708, 2014, date completed - 2015-04-22; Date created - 2014-08-20; Date revised - 2021-05-07; Last updated - 2021-05-16. [Online]. Available: http://ezproxy.wpi.edu/login?url=https://www.proquest.com/scholarly-journals/action-based-fine-grained-access-control/docview/1554940591/se-2?accountid=29120

[7] D. D. Downs, J. R. Rub, K. C. Kung, and C. S. Jordan, "Issues in discretionary access control," in *1985 IEEE Symposium on Security and Privacy*, 1985, pp. 208–208.

[8] S. Govindavajhala and A. W. Appel, "Windows access control demystified," *Princeton university*, 2006.

[9] D. o. D. United States, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, dod-5200.28-std ed. United States, Department of Defense, Jul 1987, vol. NCSC-TG-005.

[10] E. E. Moe and M. M. S. Thwin, *Effective Security and Access Control Framework for Multilevel Organizations*. Cham: Springer International Publishing, 2019, pp. 267–288. [Online]. Available: https://doi.org/10.1007/978-3-030-30436-2_13

[11] R. Hat, "What is selinux?" Aug 2019. [Online]. Available: https://www.redhat.com/en/topics/linux/what-is-selinux

[12] H. H. S. C. Program, "2020: A retrospective look at healthcare cybersecurity," Feb 2021. [Online]. Available: https://www.hhs.gov/sites/default/files/2020-hph-cybersecurty-retrospective-tlpwhite.pdf

[13] M. Hart, P. Manadhata, and R. Johnson, “Text classification for data loss prevention,” in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, S. Fischer-Hübner and N. Hopper, Eds. Springer, 2011, p. 18–37.

[14] S. Liu and R. Kuhn, “Data loss prevention,” *IT Professional*, vol. 12, no. 2, p. 10–13, Mar 2010.

[15] T. Wüchner and A. Pretschner, “Data loss prevention based on data-driven usage control,” in *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, Nov 2012, p. 151–160.

[16] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network and Computer Applications*, vol. 36, no. 1, p. 16–24, Jan 2013.

[17] Aug 2021. [Online]. Available: https://cybernews.com/security/first-half-of-2021-sees-triple-digit-rise-in-cybercrime/

[18] Crowdstrike, “Stopping breaches with threatgraph.” [Online]. Available: https://go.crowdstrike.com/rs/281-OBQ-266/images/WhitepaperThreatGraph.pdf

[19] D. T. Martin and D. F. Diehl, “Event model for correlating system component states.” [Online]. Available: https://patents.google.com/patent/US9477835B2/en?q=ids&assignee=crowdstrike&oq=crowdstrike+ids

[20] J. Liu, K. Xiao, L. Luo, Y. Li, and L. Chen, “An intrusion detection system integrating network-level intrusion detection and host-level intrusion detection,” in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, Dec 2020, p. 122–129.

[21] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, “Machine learning and deep learning methods for cybersecurity,” *IEEE Access*, vol. 6, p. 35365–35381, 2018.

[22] J. Liu, K. Xiao, L. Luo, Y. Li, and L. Chen, “An intrusion detection system integrating network-level intrusion detection and host-level intrusion detection,” in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, Dec 2020, p. 122–129.

[23] Linux, “Audit daemon - linux man page.” [Online]. Available: https://linux.die.net/man/8/auditd

[24] L. Zeng, Y. Xiao, and H. Chen, “Linux auditing: Overhead and adaptation,” in *2015 IEEE International Conference on Communications (ICC)*, Jun 2015, p. 7168–7173.

[25] P. McDaniel, “Data provenance and security,” *IEEE Security Privacy*, vol. 9, no. 2, p. 83–85, Mar 2011.

[26] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, “Provenance-aware storage systems.” in *Usenix annual technical conference, general track*, 2006, pp. 43–56.

[27] U. S. N. R. Commission, “Personally identifiable information and privacy act responsibilities awareness course.” [Online]. Available: https://www.nrc.gov/docs/ML1530/ML15306A425.pdf

[28] S. Jana, “Taint tracking.” [Online]. Available: https://www.cs.columbia.edu/~suman/secure_sw_devel/taint_tracking.pdf

[29] J. Kagan, “Pci compliance,” Mar 2021. [Online]. Available: https://www.investopedia.com/terms/p/pci-compliance.asp

[30] T. Bui, “Analysis of docker security,” *arXiv:1501.02967 [cs]*, Jan 2015, arXiv: 1501.02967. [Online]. Available: http://arxiv.org/abs/1501.02967

[31] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, “A measurement study on linux container security: Attacks and countermeasures,” in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC ’18. Association for Computing Machinery, Dec 2018, p. 418–429. [Online]. Available: https://doi.org/10.1145/3274694.3274720

[32] J. S. Reuben, “A survey on virtual machine security,” in *Security of the End Hosts on the Internet, Seminar on Network Security Autumn 2007. Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory*, 2007.

[33] B. D. Payne, R. Sailer, R. Cáceres, R. Perez, and W. Lee, “A layered approach to simplified access control in virtualized systems,” *ACM SIGOPS Operating Systems Review*, vol. 41, no. 4, p. 12–19, Jul 2007.

[34] J. Kirch, “Virtual machine security guidelines,” Aug 2006. [Online]. Available: https://lasr.cs.ucla.edu/classes/239_1.fall10/papers/CIS_VM_Benchmark_v1.0.pdf

[35] Qubes, “Qubes intro.” [Online]. Available: https://www.qubes-os.org/intro/

[36] F. Lesueur, A. Rezmerita, T. Herault, S. Peyronnet, and S. Tixeuil, “Safe-os: A secure and usable desktop operating system,” in *2010 Fifth International Conference on Risks and Security of Internet and Systems (CRiSIS)*, 2010, pp. 1–7.

[37] S. Soomro, *Engineering the Computer Science and IT*. BoD – Books on Demand, Oct 2009, google-Books-ID: BSqhDwAAQBAJ.

[38] M. Hirano, T. Shinagawa, H. Eiraku, S. Hasegawa, K. Omote, K. Tanimoto, T. Horie, K. Kato, T. Okuda, E. Kawai, and et al., “Introducing role-based access control to a secure virtual machine monitor: Security policy enforcement mechanism for distributed computers,” in *2008 IEEE Asia-Pacific Services Computing Conference*, Dec 2008, p. 1225–1230.

[39] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, and et al., “Bitvisor: a thin hypervisor for enforcing i/o device security,” in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE ’09. Association for Computing Machinery, Mar 2009, p. 121–130. [Online]. Available: https://doi.org/10.1145/1508293.1508311

[40] Hysolate, “Hysolate,” 2021. [Online]. Available: https://www.hysolate.com/

[41] J. M. McCune, T. Jaeger, S. Berger, R. Caceres, and R. Sailer, "Shamon: A system for distributed mandatory access control," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, 2006, pp. 23–32.

[42] OpenVSwitch, "Docker hub." [Online]. Available: https://hub.docker.com/r/openvswitch/ovs

[43] T. Bui, "Analysis of docker security," *arXiv:1501.02967 [cs]*, Jan 2015, arXiv: 1501.02967. [Online]. Available: http://arxiv.org/abs/1501.02967

[44] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737

[45] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, p. 87–98, apr 2014. [Online]. Available: https://doi.org/10.1145/2602204.2602219

[46] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, pp. 80–86, 1997.

[47] M. E. Najd and C. A. Shue, "Deepcontext: An openflow-compatible, host-based sdn for enterprise networks," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, 2017, pp. 112–119.

[48] C. R. Taylor, D. C. MacFarland, D. R. Smestad, and C. A. Shue, "Contextual, flow-based access control with scalable host-based sdn techniques," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[49] J. Naous, R. Stutsman, D. Mazieres, N. McKeown, and N. Zeldovich, "Delegating network security with more information," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, ser. WREN '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 19–26. [Online]. Available: https://doi.org/10.1145/1592681.1592685

[50] S. W. Shin, P. Porras, V. Yegneswara, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in *20th Annual Network & Distributed System Security Symposium*. Ndss, 2013.

[51] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up sdn control-plane using vswitch based overlay," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 403–414. [Online]. Available: https://doi.org/10.1145/2674005.2675002

[52] K.-y. Chen, A. R. Junuthula, I. K. Siddhrau, Y. Xu, and H. J. Chao, "Sdnshield: Towards more comprehensive defense against ddos attacks on sdn control plane," in *2016 IEEE Conference on Communications and Network Security (CNS)*, 2016, pp. 28–36.

[53] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[54] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, “The design and implementation of open vswitch,” in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[55] R. S. F. Community, “Ryu sdn.” [Online]. Available: https://ryu-sdn.org

[56] AppArmor. [Online]. Available: https://apparmor.net/