Attacking VMware NSX

Jan Harrie

Matthias Luft

jharrie@ernw.de

matthias.luft@rational-security.io

Outline

- Introduction
- Attack Surface
- Security Controls
- Attack Surface Evaluation
- Analysis Methodology & Tools

#whoami - Jan

- Security Consultant @ERNW GmbH
- Former Security Analyst/Pentester/WebApp-Monkey
- M.Sc. IT-Security TU Darmstadt

- Interests:
 - Bug Bounty/Red Teaming/Social Engineering
  - Camping/Music Festivals

#whoami - Matthias

o Freelance IT Security Researcher/Consultant/Trainer

o 10+ years in IT Security

o Technical background:

   o Pentesting

   o Virtualization/Hypervisor/Network Security

   o Nowadays: Cloud/Container/Agile Security

Scope of this Work

- Explaining VMware NSX for Hackers
  - In particular its attack surface
  - Not: Security design using NSX/Security features of NSX
- Modern technologies are often incredibly complex
  - Complex as in many large intransparent components not as in np-hard
  - Killing security research before it can take place due to complex/long/difficult/expensive setup phase
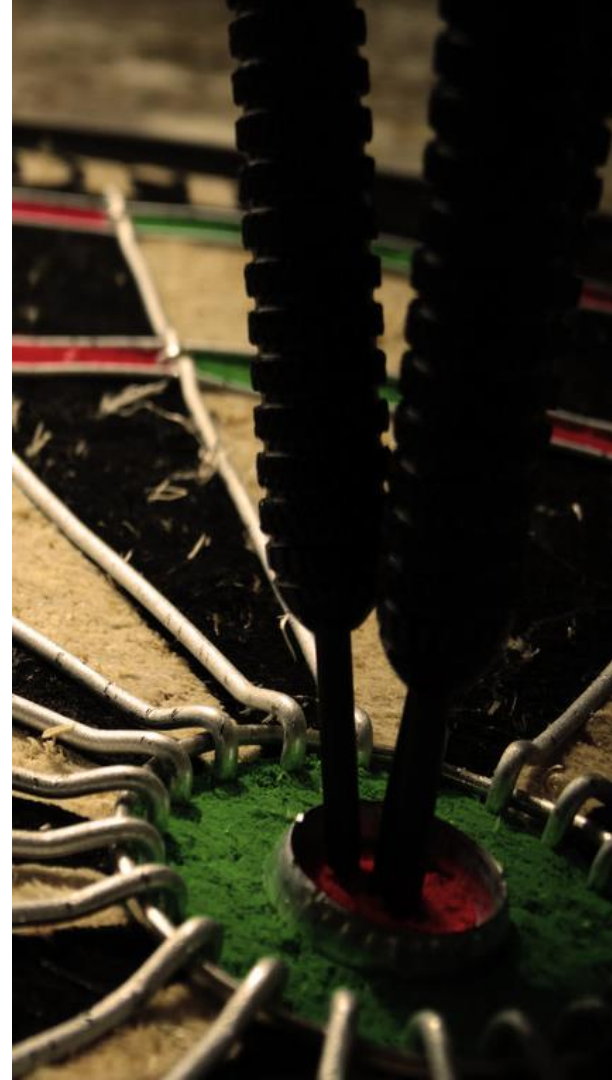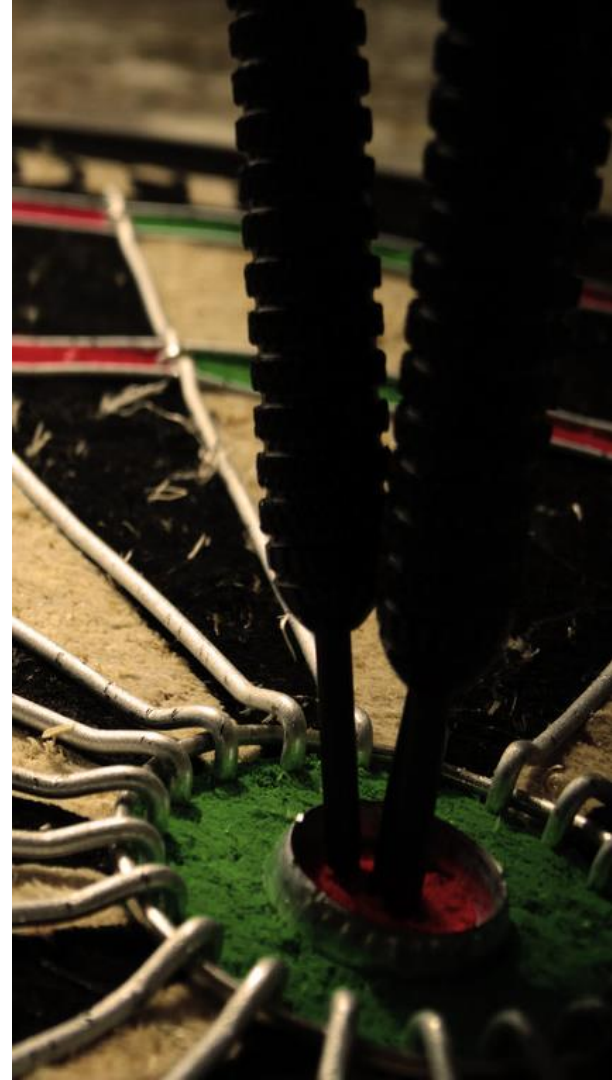
Scope of this Work

o Provide a basis for offensive research on NSX

  o E.g. 10+ person days spent so far to build lab architecture and familiarize with basic concepts.

    o Not even full scope yet, refer to future work.

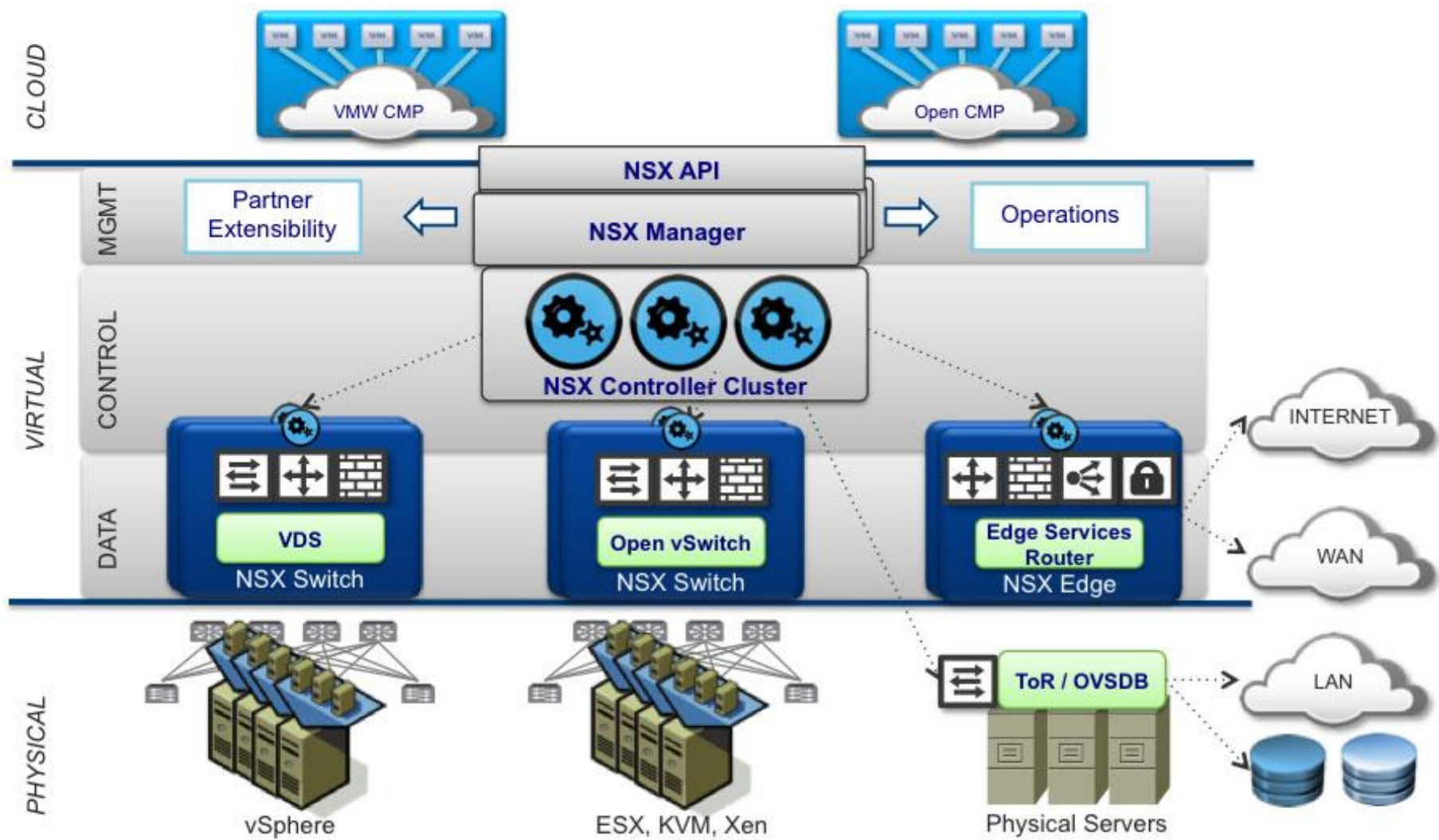o Give clear instructions on how to build a security validation lab for fellow security researchers.

halvarflake
@halvarflake

The most expensive security appliance is the most secure security appliance.

Technical Overview

Source: https://www.sdxcentral.com/vmware/definitions/what-is-vmware-nsx/

CLOUD

VMW CMP

Open CMP

MGMT

NSX API

Partner Extensibility

NSX Manager

Operations

VIRTUAL

CONTROL

NSX Controller Cluster

INTERNET

DATA

VDS
NSX Switch

Open vSwitch
NSX Switch

Edge Services Router
NSX Edge

WAN

PHYSICAL

vSphere

ESX, KVM, Xen

ToR / OVSDB

LAN

Physical Servers

VMware NSX

- o VMware NSX-T 2.1
- o Network Virtualization & Security Platform
  - o Overlay Networking for Virtualization Platforms
  - o Micro-Segmentation/-Filtering
- o De-coupling network functions from (physical) IP fabric
- o API-driven
- o NSX-V (vSphere) vs. NSX-T (Multi-Hypervisor)

NSX Manager

o Frontend (GUI and REST API) for creating, configuring, and monitoring all NSX infrastructure

o One Manager per NSX-T Setup

o Communicates with each NSX node via Management Plane Agent

NSX Controller

o Distributed state management system

o Center piece of the Central Control Plane (CCP)

o Responsible for deployment of virtual networks

o Controls virtual networks, overlay transport tunnels, and forwarding

o Traffic does not pass through controller

NSX Edge & Transport Zone

o Edge
   o Provides connectivity to external network
      o External = non-overlay
   o Typically one or more edge VMs per transport zone
o Transport Zone
   o Transport Nodes in the same Transport Zone can communicate
   o Communication over Virtual Tunnel Endpoints (VTEPs)

Logical Switches & Routers

o Logical Switches
  o L2 Broadcast-Domains
  o Flexible L2 segregation
o Logical Routers
  o North-South/East-West Connectivity
  o Aggregation of functionality that was previously performed by multiple devices
  o 2-Tier Routing Architecture
  o Control plane provides routing information
  o Management plane control routers

Planes

Source:
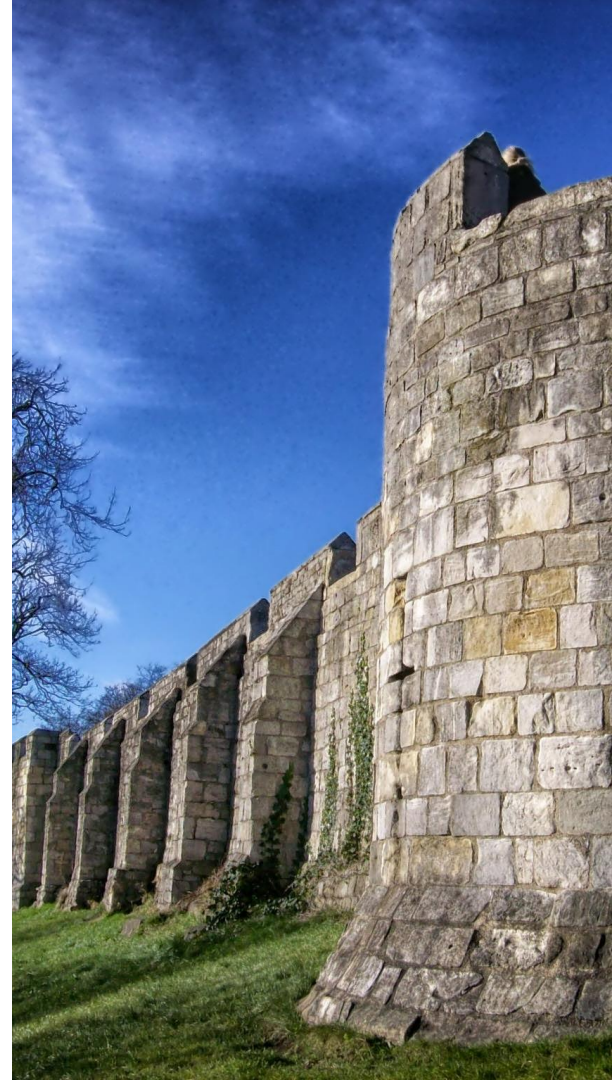VMware NSXT Install Guide
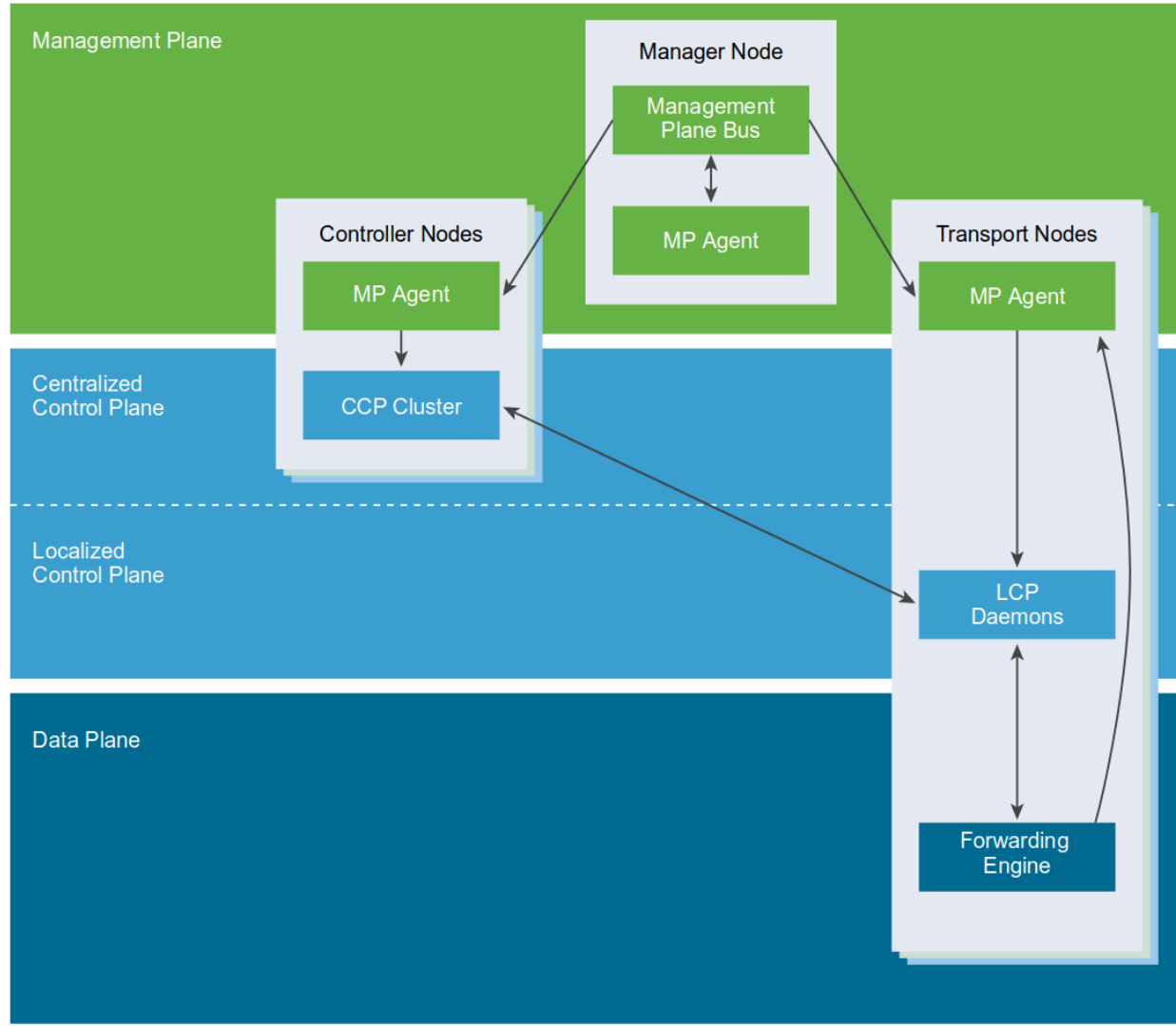
Attack Surface

Under the Hood

o Analysis typically relying on jail break
o "Jailbreak" with VMs usually easy:
  o Boot live CD/Mount virtual hard drive
  o Edit memory dump of suspended VM
  o Use debug port (see latter slides)
  o Find breakout via command line tool
    o Helpful: gtfobins
o Manager & Controller:
  o Ubuntu VMs with (large) Java services
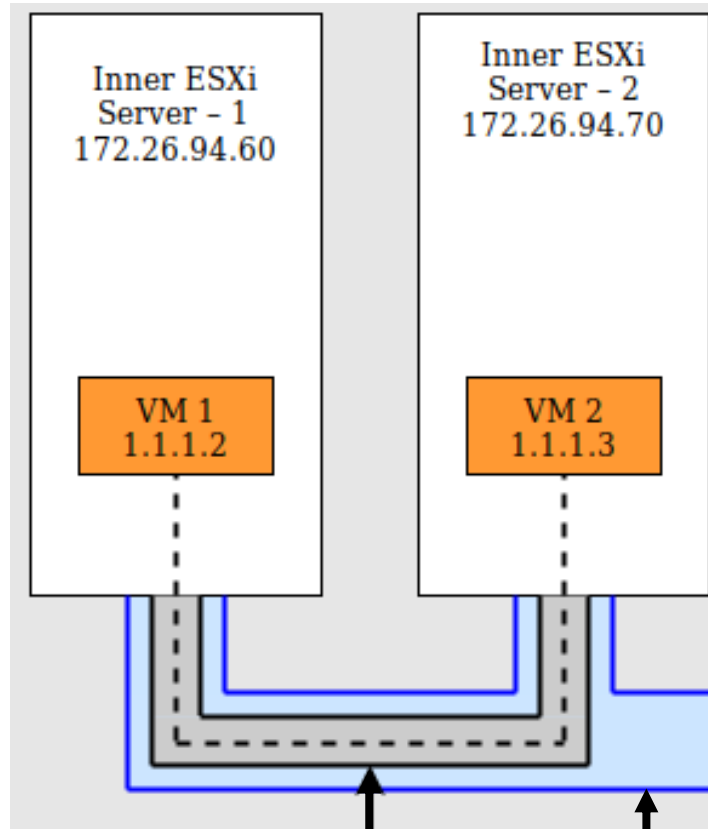 o Having `root` maintenance account with shell

Attack Surface of VMware NSX

o Data Plane
o Management Plane
o Control Plane
o "SDN Services"

o Traditional VMware/vSphere attack surface out of scope.

Data Plane

Inner ESXi
Server – 1
172.26.94.60

Inner ESXi
Server – 2
172.26.94.70
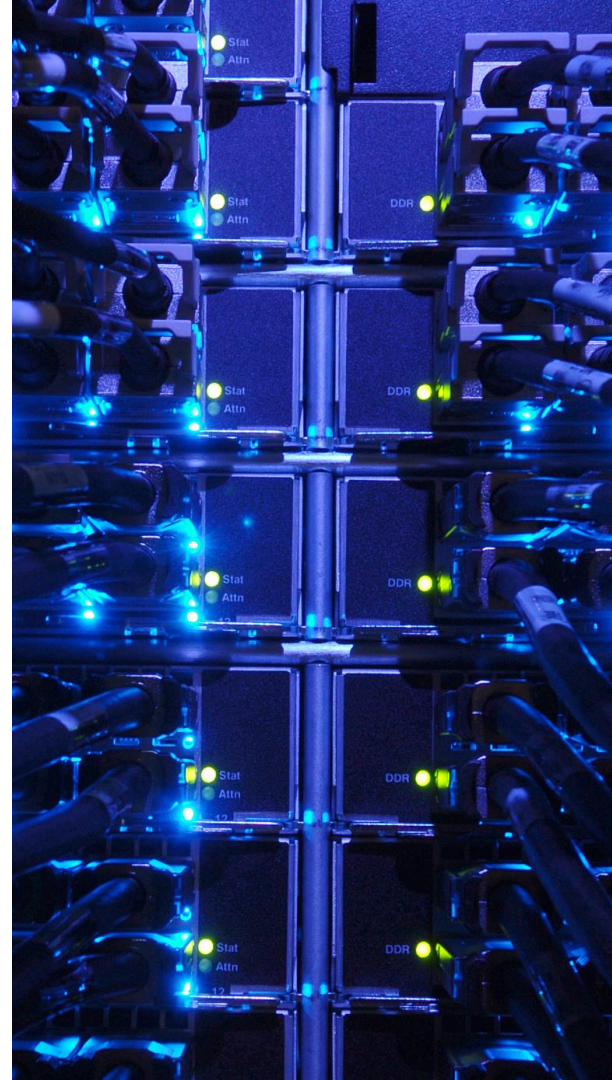
VM 1
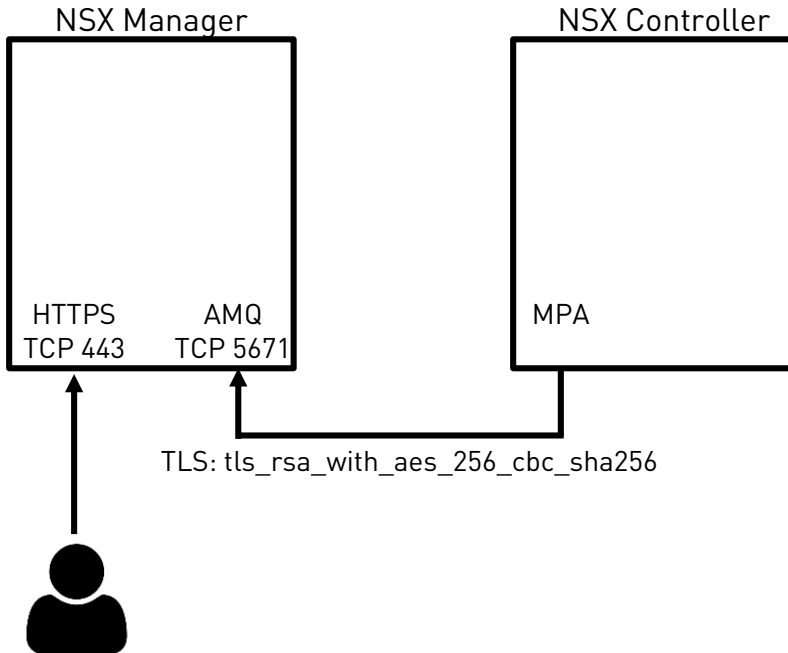1.1.1.2

VM 2
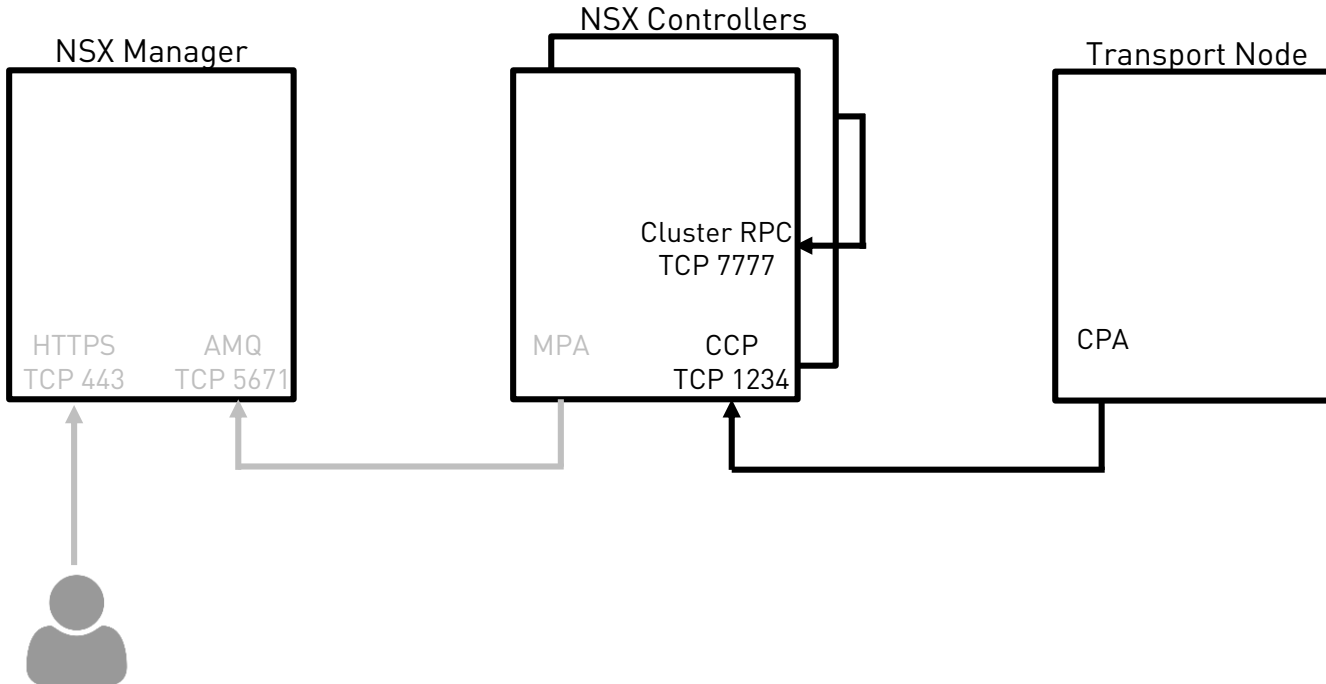1.1.1.3

Data Plane

Generic IP Transport

Data Plane

- Using Geneve as tunnel protocol
  - RFC Draft
- Virtual tunnel endpoint implemented in ESXi kernel module
  - `/usr/lib/vmware/vmkmod/nsx-vdl2`
- VTEP listening on UDP 6081

Management Plane

NSX Manager

NSX Controller

HTTPS
TCP 443

AMQ
TCP 5671

MPA

TLS: tls_rsa_with_aes_256_cbc_sha256

Control Plane

Control Plane

NSX Controllers

NSX Manager

Transport Node

Cluster RPC
TCP 7777

HTTPS
TCP 443

HTTP
TCP 8080

MPA

CCP
TCP 1234

CPA

Software Download (e.g. drivers for initial install)
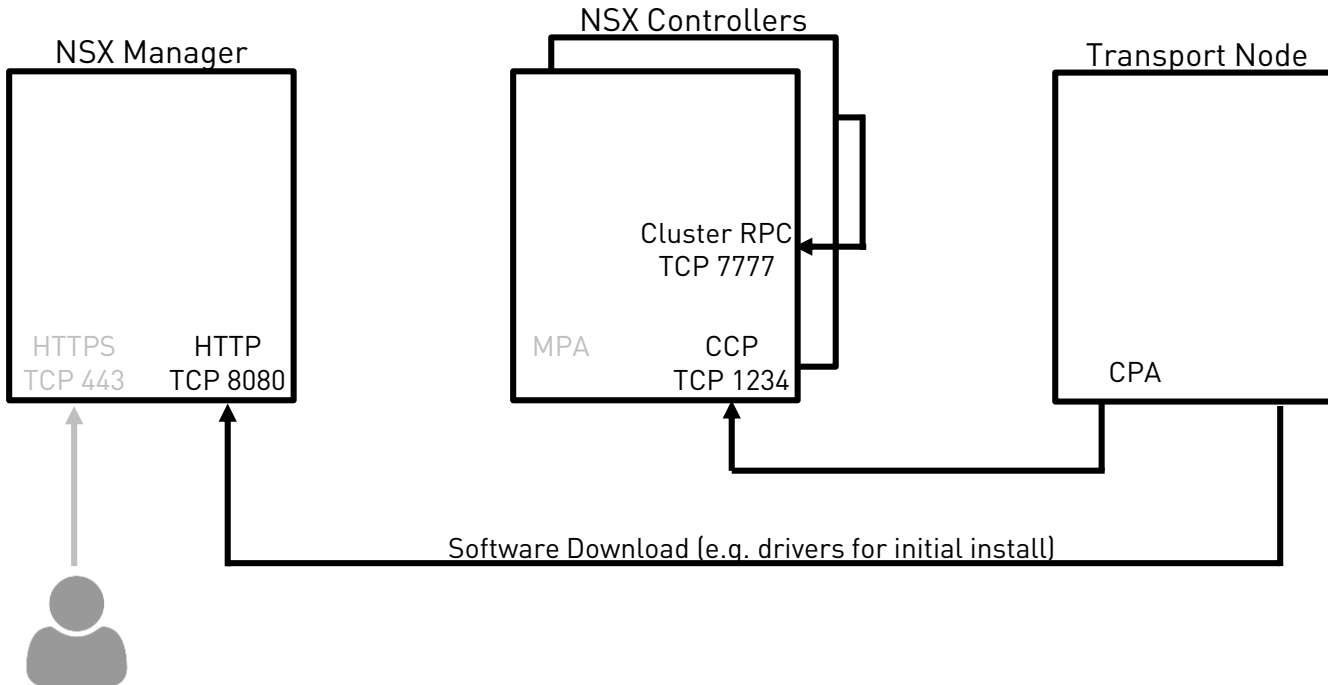
"SDN Services"

- NSX offers various networking services on the hypervisor/per-VM level such as
  - Firewalling
  - Flow Tracing
  - Routing
  - Load Balancing
- This functionality is implemented in various kernel modules
  - 3rd party vendors can supply additional modules

`/usr/lib/vmware/vmkmod/`

- nsx-vswitch & nsx-stub-vswitch
- nsx-vsip
- nsx-vdrb
- nsx-vdl2
- nsx-traceflow
- nsx-switch-security
- nsx-kcp
- nsx-ipfix
- nsx-fc
- nsx-dne
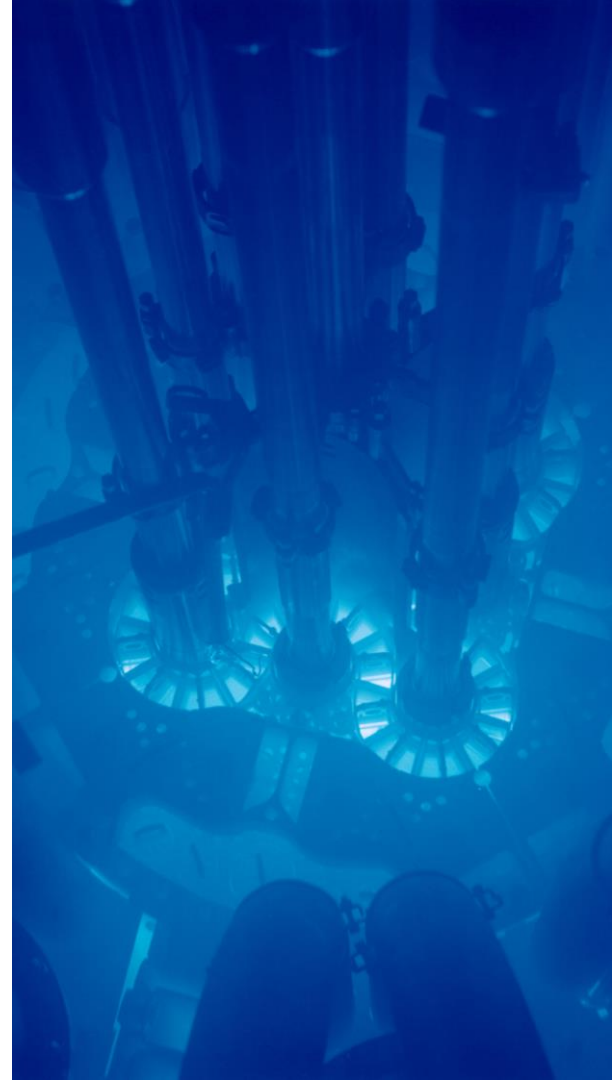- nsx-bfd

(28K-1,1M)
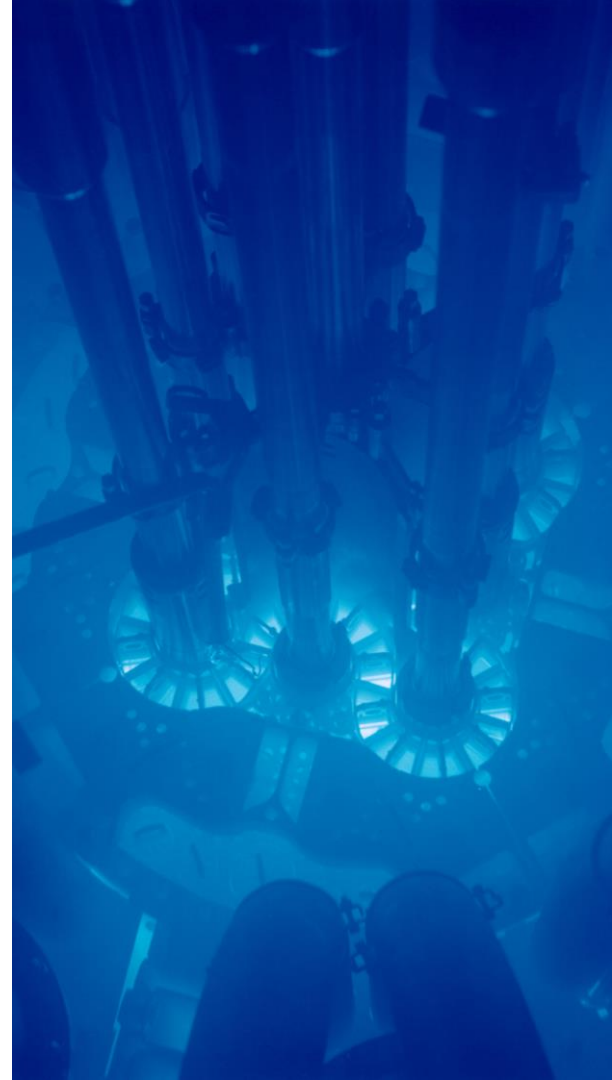
`/usr/lib/vmware/vmkmod/`

- nsx-vswitch & nsx-stub-vswitch
- nsx-vsip
- nsx-vdrb
- nsx-vdl2
- nsx-traceflow
- nsx-switch-security
- nsx-kcp
- nsx-ipfix
- nsx-fc
- nsx-dne
- nsx-bfd

Attack Surface by Component

- NSX Controller
  - Cluster RPC (TCP 7777)
  - CCP (TCP 1234)
- NSX Manager
  - UI & API (TCP 443)
  - MP/AMQ (TCP 5671)
  - Update Repository (TCP 8080)

Attack Surface by Component

o Transport Node
  o VTEP (UDP 6081)
  o Various kernel modules inspecting network traffic
    o In some cases (e.g. firewalling) *all* traffic

Security Controls & Attack Surface Evaluation

NSX Security Controls

o Design guide clearly recommends the separation of the control plane networks from production/VM networks.

    o Thus reducing overall exposure of the NSX services.

    o We strongly agree ;-)

o Control plane is implemented via TLS and authenticated.

o Local packet filter in place on the NSX components to only expose necessary services

    o For example, Zookeeper running on controller cluster

Attack Surface Evaluation

o ESXi/Transport Node kernel modules provide most interesting attack surface
  - o Very high exposure
    - o Need to inspect all network traffic going to the VMs (some of which are by design very exposed)
  - o High complexity
    - o Layer 7 inspection capabilities.
  - o High privileges
    - o Successful exploitation results in hypervisor kernel.
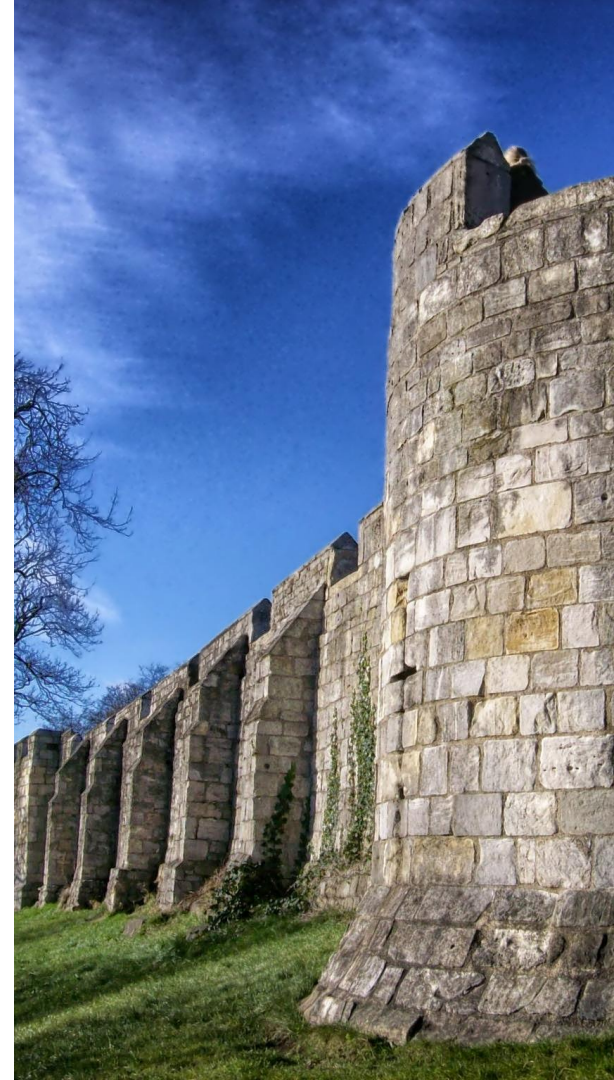
Analysis Methodology & Tools

Test Setup

- HP DL360 G5
  - 32 GB RAM, 2x Intel Xeon @2,5Ghz 4 Core, ~190 GB HDD
- ESXi server (6.5/6.7) installed on HW running:
  - 2 nested inner ESXi (6.5/6.7) servers running:
    - 2 Inner Linux VMs
  - NSX Manager
  - NSX Controller

ERNW
providing security.

Outer ESXi
Server
172.26.94.10

Inner ESXi
Server – 1
172.26.94.60

Inner ESXi
Server – 2
172.26.94.70

NSX-T
Manager
172.26.94.20

NSX-T
Controller
172.26.94.30

VM 1
1.1.1.2

VM 2
1.1.1.3

GENEVE Tunnel
(Overlay)

ESXi vSwitch
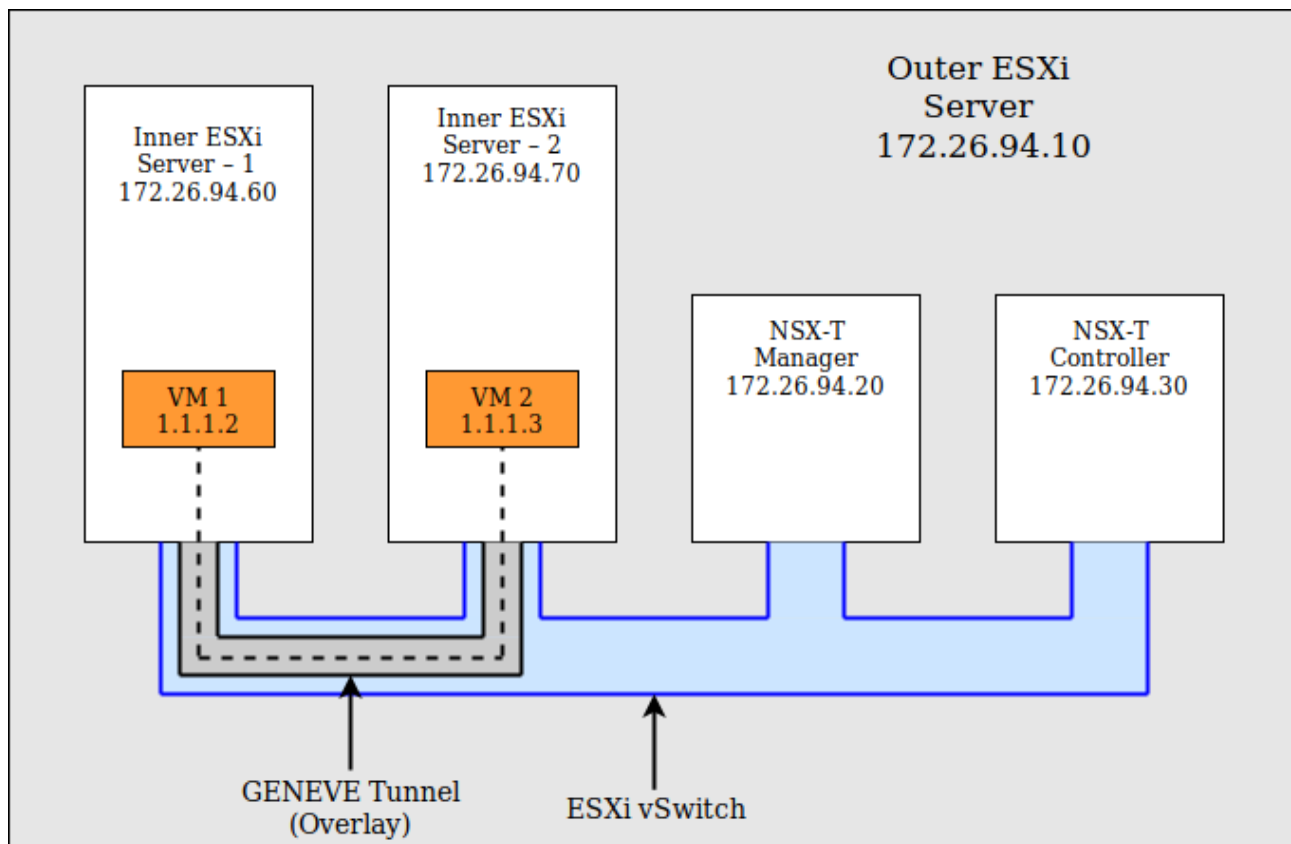
Prepare ESXi

- Modify .vmx of inner ESXi Server
  ```
  [...]
  debugStub.listen.guest64 = "TRUE"
  debugStub.listen.guest64.remote = "TRUE"
  debugStub.port.guest64 = "42040"
  [...]
  ```

- Disable Firewall on ESXi
  ```
  esxcli network firewall set --enabled false
  ```

- Disable K-ASLR
  ```
  $ cat /bootbank/boot.cfg
  [...]
  kernelopt=installerDiskDumpSlotSize=2560 no-auto-partition vmbASLR=FALSE
  [...]
  ```

Prepare Debug Setup – Get Symbols

o Download Eclipse Plugin VMware Workbench

o Open the Dashboard

o Authenticate with VMware user credentials

o Goto menu point "Debug Symbols for VMKernel Updates"

o Search for the Debug Symbols for your build version

o Extract files symbols from RPM package

```
rpm2cpio vmware-esx-devtools-bridge-6.5.0-2.50.8294253.i386.rpm | cpio -idmv
```

Prepare Debug Setup

o Getting address of kernel modules

```
$ esxcfg-info | less
[...]
\==+Module :
            |----Name....................nsx-vswitch
            |----File Name...............nsx-vswitch
            |----File Path...............usr/lib/vmware/vmkmod/nsx-vswitch
            |----Module Id...............62
            |----ReadOnly Load Address...0x0000418000df9000
[...]
```

o Load Kernel Symbols into gdb, connect to remote gdb and start debugging

```
(gdb) file /[...]/vmware/ddk-6.5.0-8294253/debug/release/vmkernel-visor
(gdb) target remote 172.26.94.10:42040
```

Prepare Debug Setup

o Load Symbols for `nsx-vswitch` to Base Addr
  `0x0000418000df9000`

```
(gdb) add-symbol-file /[...]/nsx-t/modules/nsx-vswitch 0x0000418000df9000
add symbol table from file "/[...]/nsx-t/modules/nsx-vswitch" at .text_addr = 0x418000df9000
(y or n) y
Reading symbols from /[...]/nsx-t/modules/nsx-vswitch...(no debugging symbols found)...done.
(gdb) x/10i CharDevPortValidCB
 0x418000df9000 <CharDevPortValidCB>: cmp     %esi,(%rdx)
   0x418000df9002 <CharDevPortValidCB+2>:        mov     $0xbad0005,%eax
   0x418000df9007 <CharDevPortValidCB+7>:        mov     $0x0,%edx
   0x418000df900c <CharDevPortValidCB+12>:       cmovne %edx,%eax
   0x418000df900f <CharDevPortValidCB+15>:       retq
[...]
```

Fuzzing with Dizzy

o Fuzzing on L2 ARP – WTF? – Because We Can! 🤔

o Dizzy is a network protocol fuzzer by Daniel Mende
  o Fuzzing arbitrary network protocols
  o Extensible and flexible

Fuzzing with Dizzy

```
objects = [
    Field(name="htype", default=b"\x01"                                        , size=2*8, fuzz="none"),
    Field(name="ptype", default=0x0806                                         , size=2*8, fuzz="std"),
    Field(name="hlen",  default=6                                              , size=1*8, fuzz="std"),
    Field(name="plen",  default=b"\x04"                                        , size=1*8, fuzz="std"),
    Field(name="op",    default=b"\x01"                                        , size=2*8, fuzz="std"),
    Field(name="sha",   default=mac_to_bytes(config_value("arp_src_mac")), size=6*8, fuzz="none"),
    Field(name="spa",   default=ip_to_bytes(config_value("arp_src_ip"))  , size=4*8, fuzz="none"),
    Field(name="tha",   default=mac_to_bytes(config_value("arp_dst_mac")), size=6*8, fuzz="std"),
    Field(name="tpa",   default=ip_to_bytes(config_value("arp_dst_ip"))  , size=4*8, fuzz="std"),
]
```

Source:

Fuzzing with Dizzy

```
$ 
```

GENEVE Injection

○ Generic Network Virtualization Encapsulation (GENEVE)

> 6.3.   Authentication of NVE peers
>
> A rogue network device or a compromised NVE in a data center
> environment might be able to spoof Geneve packets as if it came from
> a legitimate NVE.   In order to mitigate such a risk, an operator
> SHOULD use an Authentication mechanism, such as IPsec to ensure that
> the Geneve packet originated from the intended NVE peer, in
> environments where the operator determines spoofing or rogue devices
> is a potential threat.   Other simpler source checks such as ingress
> filtering for VLAN/MAC/IP address, reverse path forwarding checks
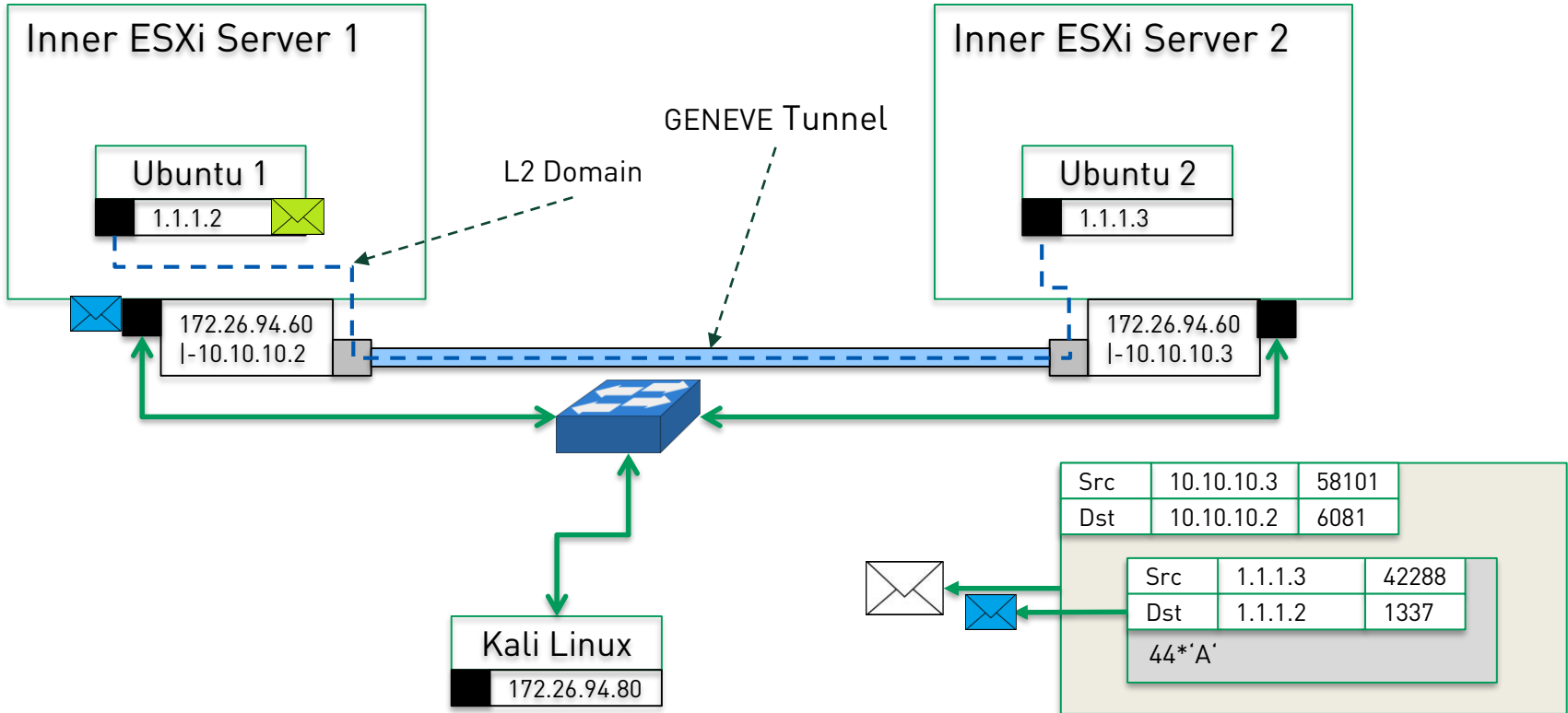
```
esxi-kali-vm [~]# python spoof.py
```

```
root@UbuntuServer01:~# sudo tcpdump -X -i eth1 net 1.1.1.0/24
```

```
root@UbuntuServer02:~# sudo tcpdump -X -i eth1 net 1.1.1.0/24
```

Conclusions

State of Work

o Features not configured/analyzed yet:
  - o Clustering
  - o Edge
  - o Container Support
  - o Internal attack surface (e.g. for privilege escalation)

State of Work - Done

o Analyze attack surface as described here
o Spot analysis of kernel modules
o Building analysis methodology/tooling
o Check network protocols for encryption and authentication
o Check on de-serialization calls in Java applications
o Fuzzing setup & some dumb fuzzing for testing

Future Work - Kernel Modules

o Main priority

o Much tooling work to be done

   o Code/state coverage of closed kernel modules

      o Current approach: Export state for afl-unicorn

      o Other ideas:

         o Internal research on control flow tracing using lib-vmi

         o Hack.Lu 2018: Hypervisor-level debugger: benefits and challenges

Future Work

o Analyze software distribution mechanism from NSX Manager update repository

o Extend lab to be able to analyze all functionality

    o E.g. filtering/encrypting kernel modules

Conclusions

- No "low hanging fruits" or inherent architectural flaws identified
- Good attack surface reduction if recommended control plane isolation is implemented
- Hopefully:
  - Provided the community a basis for more offensive research
  - Gave insight into methodology for attack surface analysis of complex closed systems

Questions & Discussion?

Thank you for your attention!

✉ jharrie@ernw.de
matthias.luft@rational-security.io

🐦 @NodyTweet
@uchi_mata

www.ernw.de

www.insinuator.net

50

Appendix

Installation/Hints

o Promiscuous Mode must be enabled on the outer ESXi host to allow network connectivity
  - o Not pretty but works
  - o Allows for simple network sniffing as well
  - o More comprehensive setup would use VLAN-capable switch + SPAN port
o Use Autostart/Shutdown feature of ESXi to start/stop the complete setup in one step

Files & Paths

- ESXi Kernel Modules:
  - `/usr/lib/vmware/vmkmod/`
- TLS certificates/keys:
  - Manager:
    - `/home/secureall/secureall/.store/`
  - Controller:
    - `/opt/vmware`
- Application data:
  - `/opt/vmware`