# Final Engagement

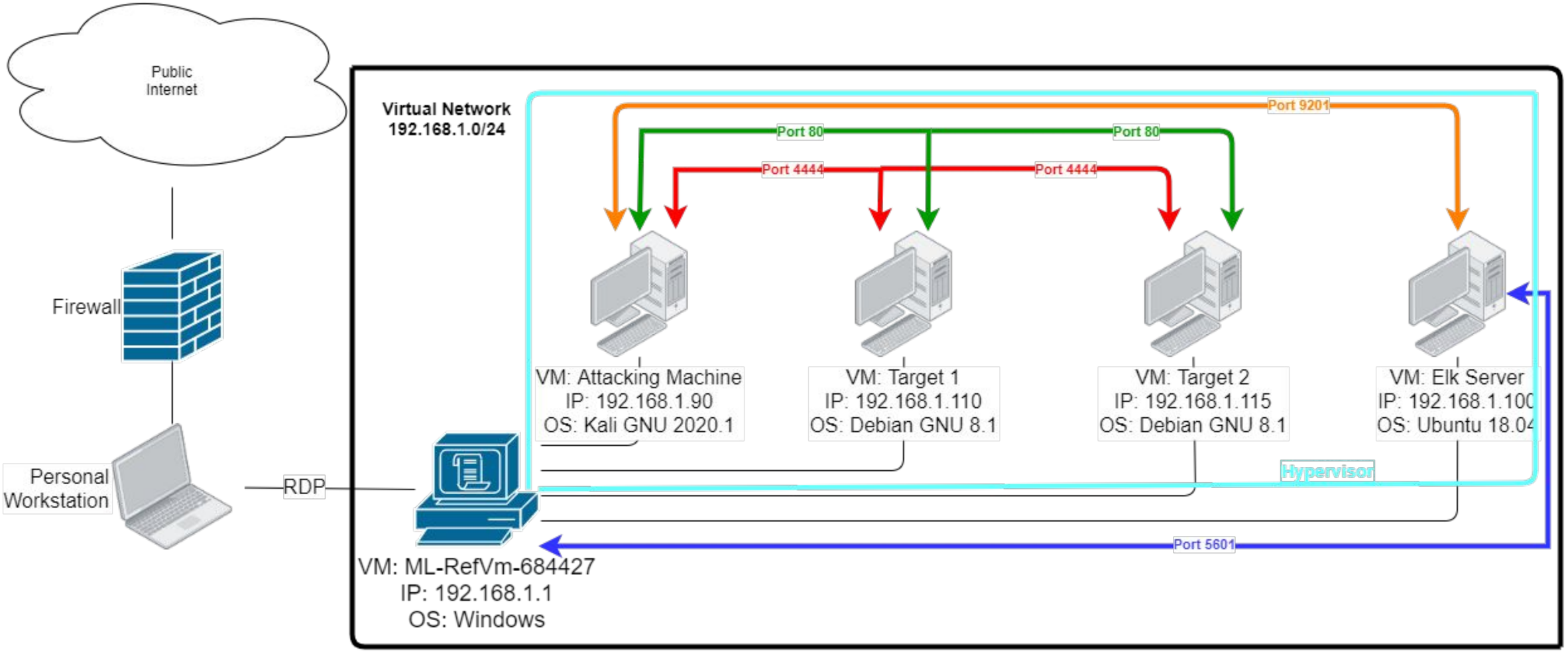
## Attack, Defense & Analysis of a Vulnerable Network

# Table of Contents

This document contains the following resources:

# Network Topology
# & Critical Vulnerabilities

# Network Topology



# Critical Vulnerabilities: Target 1

| Vulnerability | Description | Impact |
|---|---|---|
| Open Ports | Network scanning with NMAP (nmap -sV 192.168.1.110) | Found open ports and services to exploit |
| Directory Listing | Used gobuster to scan directory structure and find potential vulnerabilities. | Having open unpatched servers create vulnerabilities to the server and content |
| WordPress Vulnerable | Used ‘wp-scan’ to profile the target site and enumerate Wordpress usernames and passwords along | Using wp-scan to expose users and open the site and server up to hacking. |
| Reverse Shell | **Reverse shell** session established on a connection that is initiated from a remote machine, not from the local host | Using code injection to initiate a reverse shell to access system directories and files such as the wp-config file. |
| Direct SSH and SQL access | Having the ability to SSH into the remote system and access the database remotely. | Using SSH we were able to gain access amd find information about the users and data stored in the database. |

# Alerts Implemented

# Excessive HTTP Errors

- Which **metric** does this alert monitor?

  **This is a Packetbeat metric alert monitor**

- What is the **threshold** it fires at?

  **This threshold fires at an http.response status code above 400 within five (5) consecutive minutes**

**Reference point:  KQL string request: (Kibana)**

**WHEN count() GROUPED OVER top 5 'http.response.status_code' IS ABOVE 400 FOR THE LAST 5 minutes**

# CPU Usage Monitor

- Which **metric** does this alert monitor?

  **This is a Metricbeat metric alert monitor**

- What is the **threshold** it fires at?

  **This threshold fires at *system.process.cpu.total.pct OVER all documents* is above 0.5 for the last five (5) consecutive minutes**

**Reference point:  KQL string request: (Kibana)**

**WHEN max() OF system.process.cpu.total.pct OVER all documents IS ABOVE 0.5 FOR THE LAST 5 minutes**

# HTTP Request Size Monitor

- Which **metric** does this alert monitor?

  **This is a Packetbeat metric alert monitor**

- What is the **threshold** it fires at?

  **This threshold fires at an http.request.bytes OVER all documents above 3500 within the last immediate one (1) consecutive minute(s)**

**Reference point:  KQL string request: (Kibana)**

**WHEN sum() of http.request.bytes OVER all documents IS ABOVE 3500 FOR THE LAST 1 minute**

# Hardening

# Hardening Against Open Ports  on Target 1

**Hardening Against Open Ports**

In order to protect against OPEN PORTS vulnerabilities in target one, the best way to patch Target 1 is as follows:

**1. Update the Software**

A software update is the most effective way to successfully harden against open ports to prevent an exploit

**2. Close unused Ports**

This works to prevent port scanning of the network and reveal open ports that make be exploited

**3. Proactively Scan the Network**

This simple and preventative measure will allow the early detection and prevention of a vulnerability

# Hardening Against Directory Listing on Target 1

- **Disable directory listing**.

As a security best practice it is recommended to **disable directory listing**. You can **disable directory listing** by creating an empty **index** file (**index**. php, **index**. html or any other extension your web server is configured to parse) in the relevant **directory**.

# Hardening Against WordPress Vulnerable on Target 1

- **Hide Directory Listing in Apache with a PHP File**

A common WordPress configuration error is permitting directory listing or directory browsing as it is also known. Unless you have a specific use case where you have to have directory listing enabled, this should be disabled as it is information disclosure vulnerability.

*Command*

***sudo cp /var/www/html/wp-content/index.php /var/www/html/wp-includes***

## Hardening Against Reverse Shell on Target 1

**Reverse shell** is a remote command execution vulnerability. The reverse shell exploit is an attack that you can prevent by using:

- Require authentication to upload files
- Store uploaded files in a location not accessible from the web
- Don't eval or include uploaded data
- Scramble uploaded file names and extensions,
- Define valid types of files that the users should be allowed to upload.

A maximum possible combination of these defenses should be used according to the defense in depth principle.

# Hardening Against Direct SSH on Target 1

Secure Shell is a command line interface to access remote Linux server. SSH is based on network protocol and can use to execute various command line operations and data transfer. SSH protocol can also be used to perform scp(Secure Copy) and sftp(Secure File Transfer).

- Disable Default Port. By default **SSH** uses port 22
- Allow Users/Groups. It is necessary to limit **SSH** access to specific users as part of server **hardening**
- Block Users/Group. You can block specific users instead of allowing specific users. “DenyUsers” to SSH configuration file
- Disable Direct Root Access to enhance security
- Disable Protocol 1: SSH have two protocols. Protocol 1 and Protocol 2. The older protocol 1 have a lot of vulnerabilities and it should be disabled. Use Protocol 2 only. You can accomplish this by editing sshd_config file as shown below.

# Implementing Patches

# Implementing Patches with Ansible

## Playbook Overview

Ansible can reduce the time it takes to patch systems by running packaging modules.

Ansible can install, update, remove, or install from another location. Here is the task for updating the system:

```
 - name: update the system
   yum:
     name: "*"
     state: latest
```