How Sweet It Is: A Comparative Analysis of Remote Desktop Protocol Honeypots

Lauri Marc Ahlman

How Sweet It Is: A Comparative Analysis of Remote Desktop Protocol Honeypots

GIAC (GCIA) Gold Certification

Author: Lauri Marc Ahlman, lauri.ahlman@student.sans.edu
Advisor: *David Fletcher*

Accepted: December 27, 2020

Abstract

Remote Desktop Protocol (RDP) and other remote administrative services are consistently targeted by attackers seeking to gain access to protected systems. Honeypots are a valuable tool for network defenders to learn about attacker tools and techniques. This paper proposes an architecture for an RDP honeypot running on a Linux host. The proposed solution includes a capability to replay RDP sessions and observe attacker activity and keystrokes. Further, this paper presents a comparative analysis between this proposed solution and an RDP honeypot using the open-source project PyRDP (Gonzalez, 2020) which is represented as a Windows environment.

1. Introduction

1.1. Remote Desktop Protocol

Remote Desktop Protocol (RDP) facilitates remote access between clients and servers, including user interaction such as mouse and keyboard input and graphical output. This enables users and administrators to connect to devices in remote locations or interact with cloud resources that may not have physical interface devices at all. While this capability is a time-saving feature that provides much-needed functionality, it also introduces several risks to organizations that implement it. The global Covid-19 pandemic has only increased this risk, as organizations have rushed to enable their workforces to work remotely. Between February and April 2020, attacks against RDP surged by 330% in the United States (Atlas VPN, 2020).

Ransomware is also increasingly spread via targeted RDP attacks. Since 2017, several variants of ransomware such as Crysis have used RDP as their primary propagation method. These attacks can bypass many traditional perimeter defenses that have been relied upon to prevent the spread of ransomware in the past (Wang et al., 2018). RDP ransomware often brute-forces the login credentials of additional hosts to propagate further after initially compromising a machine. Beyond brute-forcing, older versions of RDP are often vulnerable to exploitation. The Bluekeep vulnerability (CVE-2019-0708) grants access to servers running vulnerable versions of RDP, which affected millions of hosts when discovered in 2019 ("RDP Bug in Older Windows Versions Leaves Millions Vulnerable to 'Wormable' Attack," 2019, p. 1).

Given the myriad of threats facing RPD implementations, it is more important than ever to gather intelligence regarding attacks targeting this protocol in the wild.

1.2. Honeypots

Honeypots offer network defenders one of the most direct and effective methods to gather information regarding the tools, techniques, and procedures used by malicious actors targeting their networks.

Existing research and projects regarding RDP honeypots have covered several relevant topics. Bryce (2020) described a solution using a Linux operating system to

present a realistic Windows login screen for capturing attempts to log in via RDP. While this does provide valuable information that can be used for network defense, such as blocking malicious IP addresses at the edge of a network, this solution does not actually allow a session to be created beyond the authentication attempt. Gonzalez (2020) created PyRDP, which uses a Machine-in-the-Middle (MitM) technique to allow for the capture and replay of Windows RDP sessions. This allows researchers using PyRDP to observe the actions taken by an attacker after access is gained and potentially gather malware samples and other techniques employed by the attacker.

2. Proposed Honeypot Architecture

This paper proposes a novel RDP honeypot architecture running in a Linux environment, allowing for the capture and analysis of attacker behavior targeting a Linux environment that is accessible by RDP. The solution presented here offers a unique capability to the cybersecurity community that is not otherwise available in existing projects. The architecture provided below enables researchers to expose an X Desktop Server environment to attackers and collect all attacker activity for review and use in network defense. Currently, no other honeypot offers this same capability in a Linux environment.

An existing RDP honeypot solution that offers comparable features for Windows-based targets was used as a control in the experiment: the open-source PyRDP project. A comparison of the capabilities provided by each solution is presented below in Figure 1. Although the use of PyRDP with a Linux RDP server was not addressed in the PyRDP documentation, testing was conducted that attempted to implement the XRDP MitM capability using PyRDP, and a functional configuration was not found.

	PyRDP	**XRDP Honeypot**
Target Operating System	Windows	Linux
Collection/Analysis Operating System	Windows/Linux	Linux
Collect RDP Sessions for Replay	Yes	Yes
Real-Time Session Monitoring	Yes	No
Keystroke Replay	Yes	Yes
Available as Docker Image	Yes	No

Lauri Marc AhlmanAuthor Name, email@address

Supports Network Level Authentication	No	No

Figure 1. Comparison of XRDP Honeypot and PyRDP Capabilities

Details of both solutions and their implementation for this experiment are provided in Sections 2.1 and 2.2 below.

2.1. Linux XRDP Honeypot

2.1.1. System Architecture

The proposed Linux RDP honeypot (hereafter referred to as the XRDP Honeypot) is composed of three Ubuntu virtual machines. The first virtual machine, the XRDP Host, runs the XRDP software and provides the environment for the attacker to interact with after logging in. The second virtual machine, the Passthrough Host, exposes the RDP port to the internet, collects packet capture (pcap) data from RDP connections to the XRDP Host, and relays the data from the internet to the XRDP Host. The final virtual machine, RDP Replay Host, replays RDP sessions that were captured by the passthrough host.

Figure 2, shown below, lays out the flow of information between the three components of the XRDP Honeypot in five steps. Transmission Control Protocol (TCP) port 3389, the default port for RDP traffic, is open on the Passthrough Host. Connections are permitted to the honeypot from the internet on this port (Step 1). The Passthrough Host uses a host-based firewall Network Address Translation (NAT) rule to rewrite the source address on these packets and forward them to the address of the XRDP host (Step 2). The XRDP Host allows RDP traffic only from the Passthrough Host to interact with the XRDP service. Packets sent from the XRDP Host back to the connecting client are routed through the Passthrough Host (Step 3), where a host-based firewall masquerade rule rewrites the packets to provide the appearance that they originated from the Passthrough Host. The Passthrough Host captures all this network traffic as it is routed back to the internet-based client (Step 4), which is downloaded to the RDP Replay Host (Step 5) for analysis.

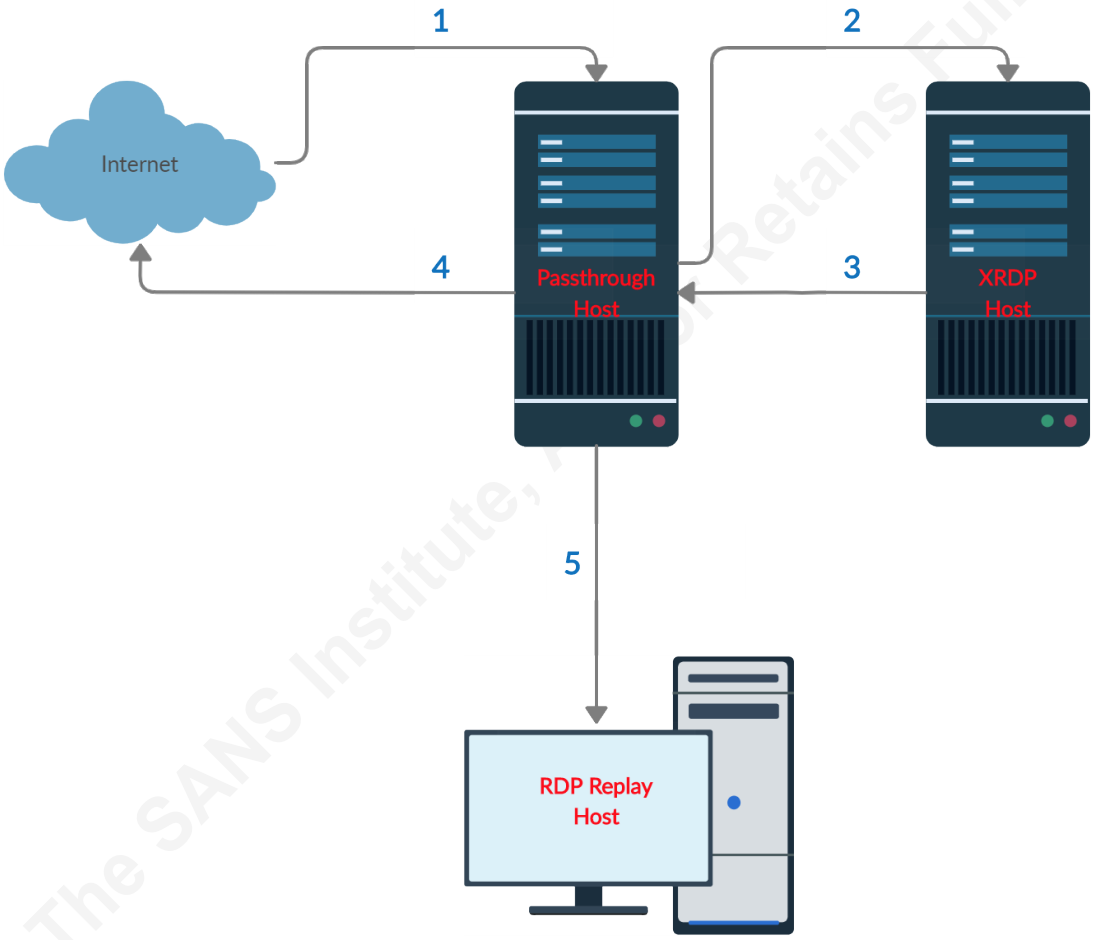

Figure 2. XRDP Honeypot Architecture

While many Linux and Unix variants would likely perform similarly in this experiment, Ubuntu Linux was used as the operating system for each virtual machine described. The specific configurations provided in this paper were tested in both Windows Hyper-V as well as Amazon Web Services (AWS) EC2 environments.

The following sections describe the configuration of each host. Ansible scripts to configure each virtual machine and supporting files for each installation are provided in Appendix A.

2.1.2. XRDP Host Configuration

The XRDP Host is built using Ubuntu Server 20.04, amd64. After installing the operating system and updates, the packages ubuntu-desktop, XRDP, and iptables-

persistent must be installed. Ubuntu-desktop provides the attacker with a Graphical User Interface (GUI) desktop environment when they connect via RDP. The XRDP package is the remote desktop software, and iptables-persistent is used for the firewall.

According to a study of RDP scanning activity over a 334-day period by Rapid7, attackers tried usernames "administrator" or "Administrator" in over 59% of attempted logins (Hodgman, 2016). Thus, a new user "Administrator" is created in order to provide a commonly attacked username, but no password is set for the user. By creating an account with no password, this account can be made vulnerable to RDP login via XRDP without exposing an additional user account to access by other means, such as SSH.

To properly configure the XRDP service for use in the honeypot, several changes must be made to the settings file xrdp.ini. The man pages for XRDP (Neutrino Labs, 2020) describe all the settings available for configuration through xrdp.ini, but there are two key settings for this honeypot. The following line prepopulates the login screen with the username "Administrator", but editing of the username is still allowed:

```
username=askAdministrator
```

More importantly, the security layer and encryption level used by XRDP must be set appropriately to ensure that sessions can be decrypted and replayed later using RDP Replay. The default `crypt_level` setting, high, is correct for this use case, but the `security_layer` setting must be changed to "rdp", which is not safe from MitM attacks (Neutrino Labs, 2020). This prevents the sessions from using more secure encryption (which is implemented through TLS as part of XRDP). The following line implements this setting:

```
security_layer=rdp
```

Xrdp-sesman is the session manager for XRDP. According to Neutrino Labs (2020), xrdp-sesman is responsible for authenticating the user and starting the proper Xserver. The following settings were changed for the XRDP Honeypot in the sesman.ini configuration file:

```
KillDisconnected=True
IdleTimeLimit=300
```

`KillDisconnected` terminates sessions 60 seconds after the client disconnects from the XRDP server. `IdleTimeLimit` specifies the period that a session will continue without user input. These two settings combined will ensure that inactive sessions do not carry on indefinitely, which shortens the analysis time for RDP Replay.

The configuration as described so far will run XRDP, but it will not allow the Administrator user to log in to the virtual machine. Setting the password for this account to a common, easily guessable value was considered, but this would still greatly reduce the chances of an attacker accessing the system for their behavior to be observed. Therefore, following in the footsteps of the PyRDP project, the XRDP Honeypot allows any password entered to successfully authenticate a session. This is accomplished through a custom PAM configuration file that replaces the default xrdp-sesman file installed in /etc/pam.d/ when XRDP is installed. The content of the custom file is as follows:

```
#%PAM-1.0
@include common-session
account    required   pam_permit.so
auth       required   pam_permit.so
password   required   pam_permit.so
```

This configuration replaces the standard account, auth, and password pam modules with pam_permit.so. According to the pam_permit man page (Linux man-pages project, 2016), pam_permit is the promiscuous module used to always grant access. The XRDP Honeypot requires that common-session still be used in order to properly launch the interactive GUI session.

The allowed_users setting in /etc/X11/Xwrapper.config also needs to be updated to a more permissive setting, or it will prevent the GUI from launching correctly:

```
allowed_users=anybody
```

The default firewall software for Ubuntu, ufw, needs to be disabled and stopped, and iptables configured. The iptables rules for this host are based upon a default drop policy, with allowed connections for port 22 to manage the server, and port 3389 filtered for connections from the Passthrough Host. The exact details of this configuration are

provided in the iptables_xrdp.sh script in Appendix A. Although an attacker accessing the Administrator account should not have the privileges necessary to change the firewall rules, this experiment also implemented restrictions on TCP port 3389 to the IP address of the Passthrough Host through a Network Security Group firewall rule on AWS.

The XRDP host is now fully configured, and the last step of setting up this host is to download /etc/xrdp/rsakeys.ini from the honeypot. This file will be used in configuring the RDP Replay host and contains the encryption keys necessary to decrypt the RDP sessions later.

2.1.3. Passthrough Host Configuration

Like the XRDP Host, the Passthrough Host is also built on Ubuntu Server 20.04, amd64. After installing the operating system and installing updates, tcpdump is installed to capture network traffic, and iptables-persistent is installed to provide the firewall.

Since this virtual machine will act as a router for RDP sessions intended for the XRDP Host, IP forwarding must be enabled by sysctl. The following line in /etc/sysctl.conf must be uncommented so that it is active:

```
net.ipv4.ip_forward=1
```

Iptables is used to implement the forwarding and masquerade rules needed to send the traffic via the Passthrough Host to the XRDP Host. The default Ubuntu firewall, ufw, must be disabled and stopped before the iptables firewall is enabled and configured. A default drop policy is implemented, with exceptions for port 22 to manage the server via SSH, and four rules for the RDP traffic forwarding:

```
iptables -t nat -A PREROUTING -p tcp —dport 3389 -j DNAT —
to-destination [XRDP Host IP address]:3389
```

This rule implements a destination Network Address Translation rule that rewrites the destination IP address for RDP packets to the IP address of the XRDP Host.

```
iptables -A FORWARD -p tcp —dport 3389 -j ACCEPT
iptables -A FORWARD -p tcp —sport 3389 -j ACCEPT
```

These rules allow for RDP packets to and from the XRDP Host to be forwarded.

```
iptables -t nat -A POSTROUTING -j MASQUERADE
```

This rule implements a source IP address masquerade, meaning that it replaces the source IP address with the Passthrough Host's IP address in each packet before it sends it out (Rupp, 2016).

Traffic is captured on the Passthrough Host by tcpdump. The following command is an example of how to capture the traffic:

```
tcpdump -I eth0 -w /home/ubuntu/Data/capture-
2020%m%d_%H_%M_%S.pcap -G 1800 tcp port 3389
```

This command will continuously write packets to disk that are sent to and from TCP port 3389. The filename will include the date and time when the capture started and begin a new file every 30 minutes.

2.1.4. RDP Replay Host Configuration

RDP Replay was developed by Steve Elliot of Context Information Security to investigate a malicious RDP session for one of his company's clients. RDP Replay can be used to decrypt and replay video of RDP sessions that are using RC4 encryption (Elliott, 2014).

Unfortunately, due to the version of openssl that it relies upon, RDP Replay cannot be installed in Ubuntu 20.04. The operating system for the RDP Replay Host is Ubuntu 16.04, amd64. After installing the operating system and installing updates, the ubuntu-desktop package and dependencies for RDP Replay must be installed. The full list of required packages can be found in the install-rdpreplay.yml Ansible script in Appendix A. Once these requirements are met, the project can be cloned from its Github repository: https://github.com/ctxis/RDP-Replay. After navigating to the directory that the project is cloned to, the executable can be built by running the `make` command.

Unlike the Passthrough Host, this virtual machine does not need to be accessible from the internet. Ubuntu 16.04 LTS will reach end of life in 2021, so it will become increasingly important to host the virtual machine in a secure manner once it stops receiving updates. Hosting the Passthrough Host locally is the best solution, as this prevents exposure of the older operating system to the internet through cloud hosting.

In order to decrypt the RDP sessions, encryption keys must be provided to RDP Replay. Elliott (2014) describes the process of extracting the relevant keys from Windows systems using Passcape LSASecretReader or mimikatz depending on the version of the operating system. A method for obtaining these keys for a Linux host is not provided by Elliott, but is presented here.

According to Elliott (2016) the private keys provided to the rdp_replay executable should be in the following format:

```
<name>,<public_key>,<private_key>
```

The name can be any string, so for this explanation, the string "ExampleKey" will be used. The remaining data can either be observed by capturing an RDP session or taken from the rsakeys.ini file downloaded from the XRDP Host during setup.

The private_key component of the format listed above is the private exponent from the rsakeys.ini file downloaded from the XRDP Host in hexadecimal with no byte separation. This is usually a 256-byte value for XRDP. As an example, the private key in an rsakeys.ini file is formatted in the following manner (shortened for brevity):

```
pri_exp=0xe1,0x61,0x04,0x71,0x01,0x3a,0x8a,0x03
```

The corresponding private key for the RDP Replay key file would be:

```
e1610471013a8a03
```

The public_key section of the key file consists of RSA metadata, including the public exponent, and the public modulus from the rsakeys.ini file. The RSA metadata can be viewed during the establishment of an RDP session in Wireshark. Figure 3 below shows a network capture of a connection established to XRDP with the packets decoded as TPKT. This metadata begins with "RSA1" indicating that the key is being used as public-only (Elliott, 2016). The next two bytes, "08 01", represent the length of the public modulus. This is little-endian hexadecimal notation, so in the example below, this indicates that the public modulus is 264 bytes long. It is important to note this length to include the appropriate padding at the end of the modulus. The last four bytes of the metadata "01 00 01 00" is the public modulus, once again in little-endian hexadecimal notation. This is also shown in the rsakeys.ini file as pub_exp. The value in the example below is decimal 65537, which is the minimum exponent value specified by the NIST

Special Publication on Computer Security SP 800-78 Rev 1 (National Institute of Standards and Technology, 2007).

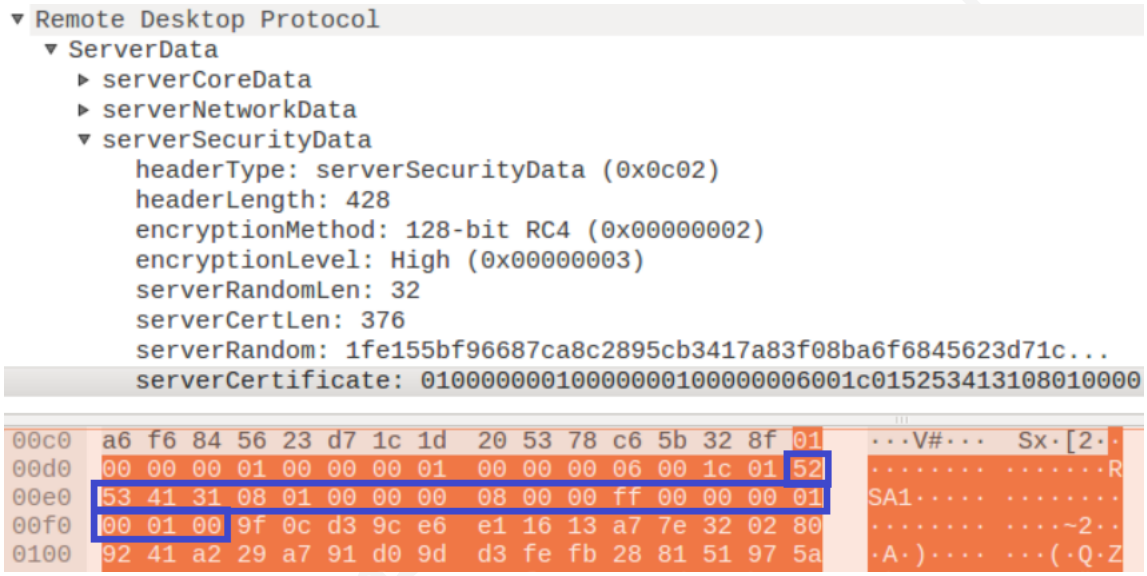

Figure 3. RSA Metadata from XRDP in Wireshark.

The public modulus is provided in the rsakeys.ini file as well. Like the private exponent, the value in the rsakeys.ini file is hexadecimal but must be stripped of separator characters. The value found in this file for a typical installation of XRDP is 256 bytes long and must be padded with "00" bytes to produce a modulus of the proper length specified in the metadata. For example, a public modulus from an rsakeys.ini file is shown below (shortened for brevity, ellipsis representing 248 additional bytes):

```
pub_mod= 0x9f,0x0c,0xd3,0x9c,0xe6,0xe1,0x16,0x13…
```

The corresponding public modulus value needed for the RDP Replay public_key with padding is:

```
9f0cd39ce6e11613…0000000000000000
```

Combined with the RSA metadata shown above, the public_key portion of the ExampleKey will look like this:

```
5253413108010000000800 00ff00000001000 1009f0cd39ce6e11613…00
00000000000000000
```

When combined with the key name, and private key, the final file format for this example is:

Lauri Marc AhlmanAuthor Name, email@address

```
ExampleKey,525341310801000000080000ff000000010001009f0cd39c
e6e11613…0000000000000000,e1610471013a8a03
```

A Python script named keyparse.py is included in Appendix A, which will generate a key file for use with RDP Replay from an XRDP rsakeys.ini file.

2.2. PyRDP Honeypot

To experimentally test the efficacy of the XRDP Honeypot, an existing RDP honeypot solution was run simultaneously as a control for comparison. PyRDP is an open-source project created by the company GoSecure. PyRDP was implemented in Python3 and can run an RDP MitM service from both Windows and Linux hosts, though it was designed to work with Windows environments as the target. The library can be used for both penetration testing purposes as well as honeypots. First released in 2018, a number of additional features including CredSSP support, Clipboard File Carving, and Dynamic Certificate Cloning were added leading up to the PyRDP 1.0 release in October 2020 (Beaulieu, 2020).

PyRDP functions as a MitM, decrypting and capturing RDP sessions for real-time monitoring or asynchronous review as they are passed between the client and the Windows server.

Figure 4 below shows the PyRDP honeypot architecture as it was implemented for this experiment. Traffic from the internet is permitted to the PyRDP Host TCP port 3389 (Step 1). PyRDP downgrades the connection's security to SSL and uses its own SSL certificate to encrypt the sessions between PyRDP and the client. The PyRDP Host initiates a connection to the Windows Host and relays the decrypted requests from the client to the Windows server (Step 2). The Windows Host routes traffic back through the PyRDP Host (Step 3), which is then encrypted and sent to the connected client.

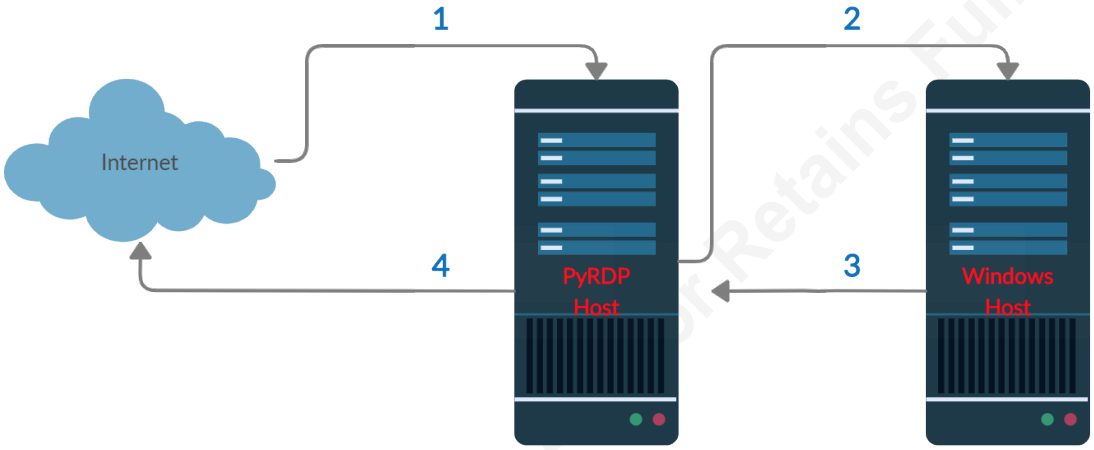

Figure 4. PyRDP Honeypot Data Flow

2.2.1. Windows Host Configuration

The following configuration was tested on Windows Server 2016 in Microsoft Hyper-V, and Windows Server 2019 in AWS EC2. The operating system should be installed with a GUI and a secure password. Once the installation is complete, RDP must be enabled. To turn on RDP, click on the Local Server in Server Manager, and then click on "Disabled" next to Remote Desktop.

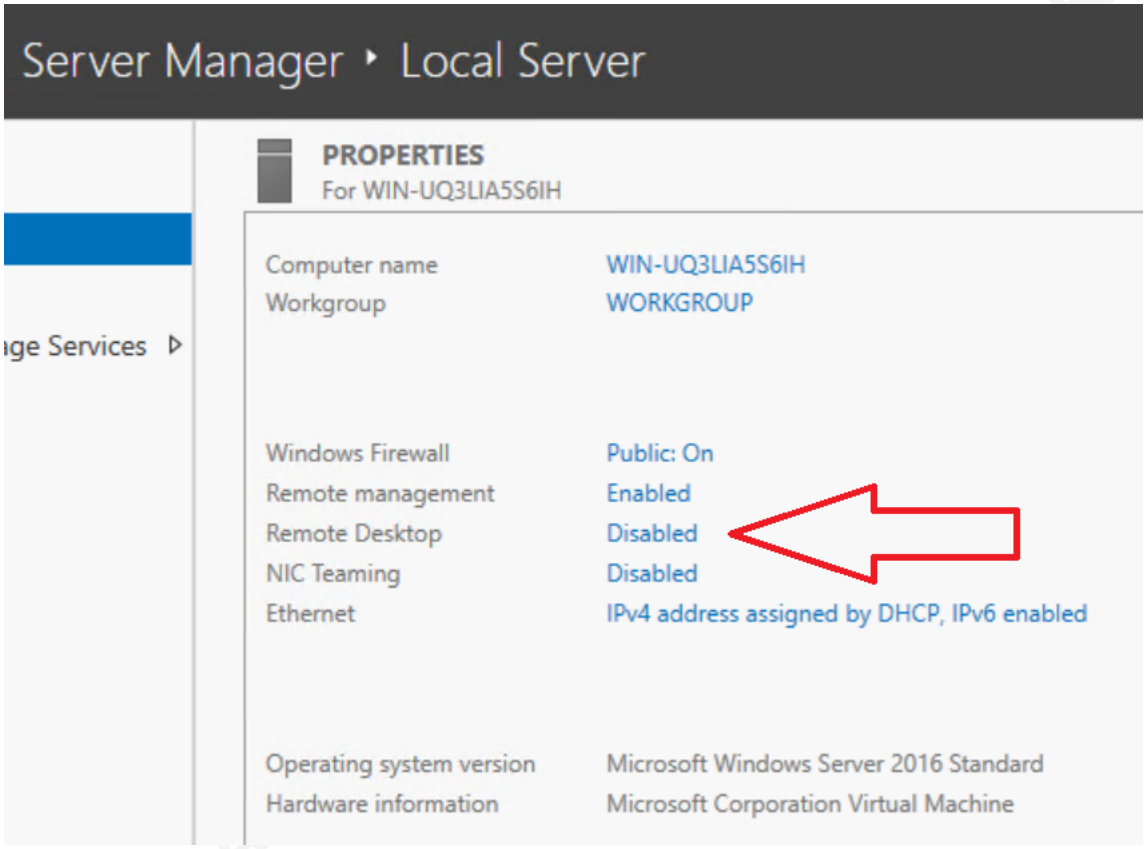

Figure 5. Enabling RDP on Windows Server 2016 Standard

When the properties window opens up, click "Allow remote desktop connections to this computer" and ensure that the checkbox for "Allow connections only from computers running Remote Desktop with Network Level Authentication (recommended)" is not checked. This feature was not compatible with PyRDP at the time of writing, and therefore was disabled. While a best practice for securing a server in a production environment, enabling this feature will prevent the proper functioning of the honeypot due to a lack of support by PyRDP (Gonzalez, 2019).

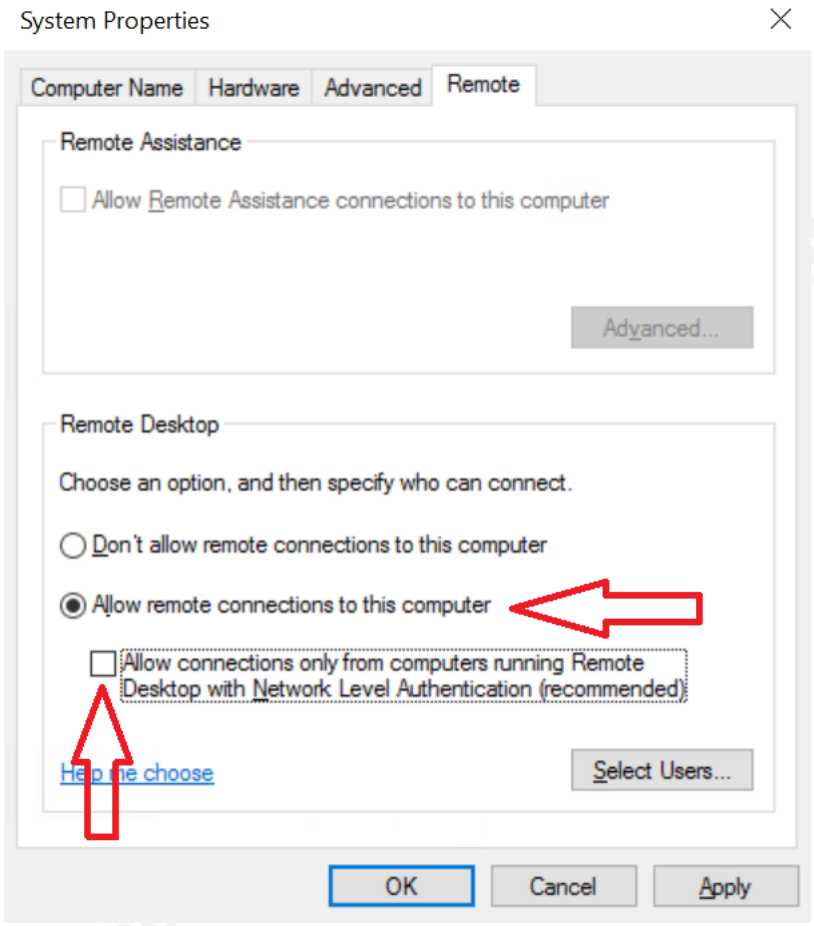

Figure 6. Remote Desktop System Properties

A firewall is required to ensure that only RDP connections from the PyRDP Host to the Windows Host are permitted. This can be configured in the Windows Firewall, but these settings could be changed by an attacker once they are logged into the VM. For this experiment, a Network Security Group was used in AWS that only allowed traffic to the Windows Host TCP port 3389 from the PyRDP Host IP address.

2.2.2. PyRDP Host Configuration

PyRDP can be installed in both Linux and Windows environments. In this experiment, Ubuntu Server 20.04 was used as the operating system. After installing the operating system, the following packages must be installed: python3, python3-pip, python3-dev, python3-setuptools, python3-venv, build-essential, git, openssl, libdbus-1-dev, libdbus-glib-1-dev, libgl1-mesa-glx, notify-osd, dbus-x11, libxkbcommon-x11-0,

and iptables-persistent. Once these are installed, the project can be cloned from github.com.

PyRDP must be installed in a virtual environment (virtualenv), so the virtualenv package must be installed via pip3. After creating a virtualenv, the project requirements must be installed in the virtualenv. These requirements are pip, setuptools, and wheel. The command to finish the installation of PyRPD in the activated Python virtual environment is:

```
pip3 install –U '.[full]'
```

To complete the installation of the honeypot, firewall rules must be configured to allow incoming traffic to TCP port 3389. Unlike the XRDP Honeypot, no forwarding rules are required as the PyRDP software does all the traffic forwarding itself. A script to configure iptables is included in Appendix A, along with an Ansible script to install and configure the PyRDP Host.

3. Experiment Design and Analysis Methodology

To test the proposed XRDP Honeypot, both honeypots configured as described above were operated simultaneously in AWS EC2 infrastructure for a period of 12 days. During this time, RDP sessions were captured by PyRDP, and PCAP data was collected for analysis from the Passthrough Host of the XRDP Honeypot.

Once the honeypots were configured, several reconnaissance tools were used to scan the two honeypots. The honeypots were then analyzed based on their appearances from an attacker's point of view.

PyRDP Honeypot sessions were reviewed using the PyRDP Player, pyrdp-player.py. PyRDP Player displays the content of the RDP sessions in a video window and the keystrokes and other information in a text box below the video. This review process could also be accomplished by using the PyRDP Conversion Tool, pyrdp-convert.py. The following command was used within the PyRDP Python virtual environment to open the results in the PyRDP Player:

```
python3 bin/pyrdp-player pyrdp_output/replays/*
```

Lauri Marc AhlmanAuthor Name, email@address

This command will open the Player GUI, seen below in Figure 6. The Player will show a tab for each replay file opened. It also has a pane where the video of the session plays, and a text pane below the video which shows the non-visual data for the session such as hostname and keys pressed.

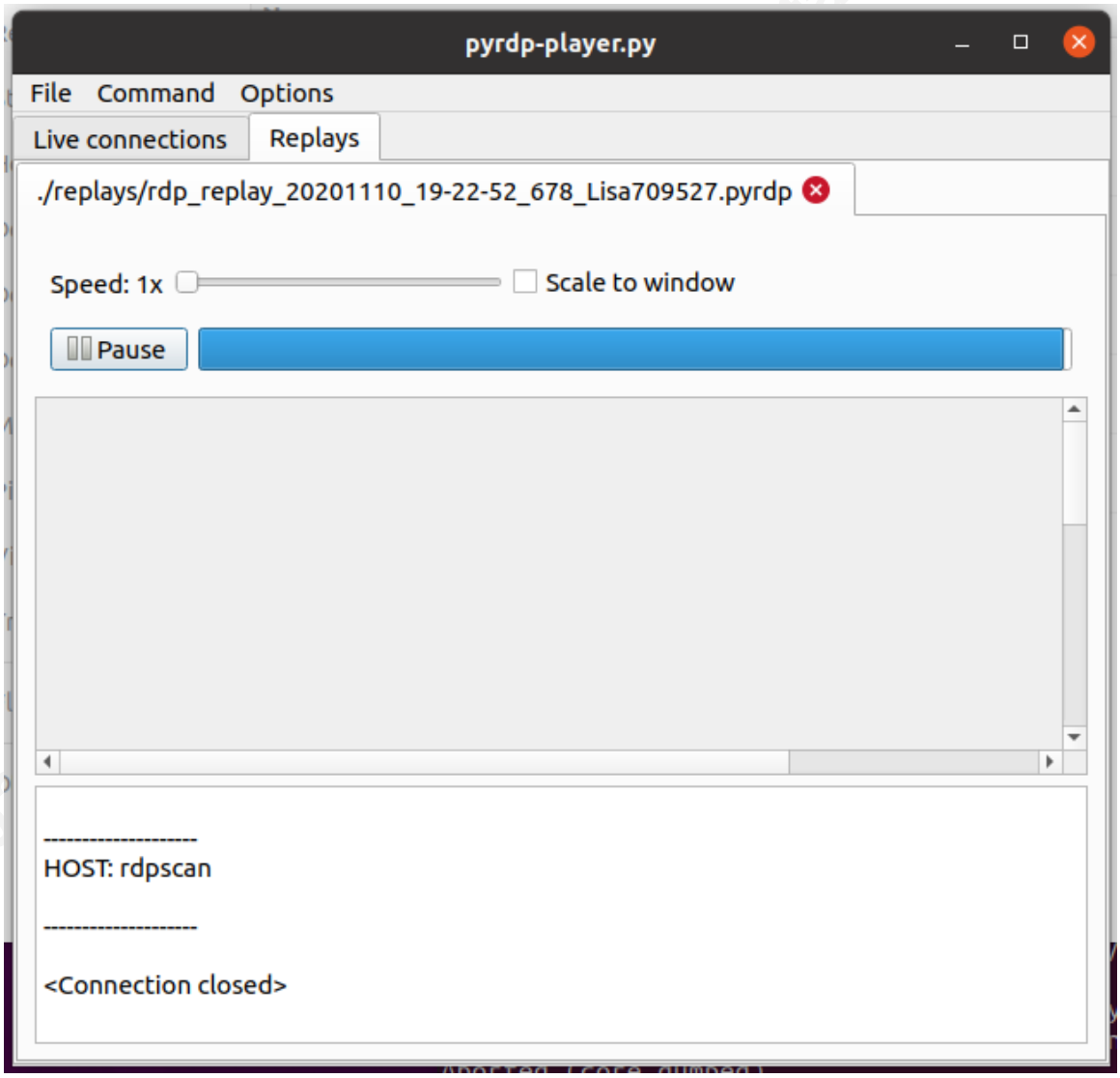

Figure 7. PyRDP Player GUI.

For the XRDP Honeypot, sessions were parsed out of the PCAP data captured by the Passthrough Host and were replayed for review using RDP Replay on the RDP Replay Host. To enable review of individual sessions with RDP Replay, the TCP stream containing the RDP session must be parsed out into an individual pcap file. This was

completed using a combination of mergecap, and tshark, two tools that are maintained by the Wireshark Foundation.

To begin, the pcap files for the review period were merged with mergecap. This is important as a preprocessing step because RDP sessions may span multiple pcap files. Merging the pcap files avoids breaking sessions that are arbitrarily separated into different files based on the time they occurred. The command used to merge the files was:

```
mergecap -w combined.pcap ./*.pcap
```

Then, individual RDP streams were parsed out of the merged pcap file using tshark. The sessions were filtered to include only those containing the RSA metadata "RSA1", indicating that the server sent encryption information to the client. This eliminates scanning sessions that do not establish the RDP encryption. The following command was used to identify streams containing this encryption exchange and then parse them out into individual pcap files:

```
tshark -Y 'frame contains RSA1' -r combined.pcap -T fields
-e tcp.stream | xargs -I {} /bin/sh -c 'tshark -r
combined.pcap -Y tcp.stream=="$0" -w
pcapsessions/"$0".pcap' {}
```

Following this stream parsing process, sessions over a certain size threshold were replayed with RDP Replay using the following command where [session] represents the TCP session number:

```
replay/rdp_replay -l key.txt —no_cksum —show_keys -r
pcapsessions/[session].pcap
```

This command will initiate the RDP Player window, where a video of the session will play. The keystrokes sent in the RDP session are output in the terminal where the command runs.

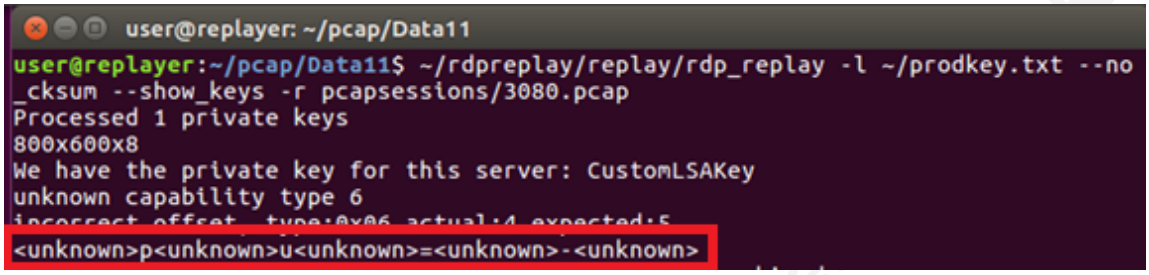

Figure 8. Keystroke Output during RDP Replay

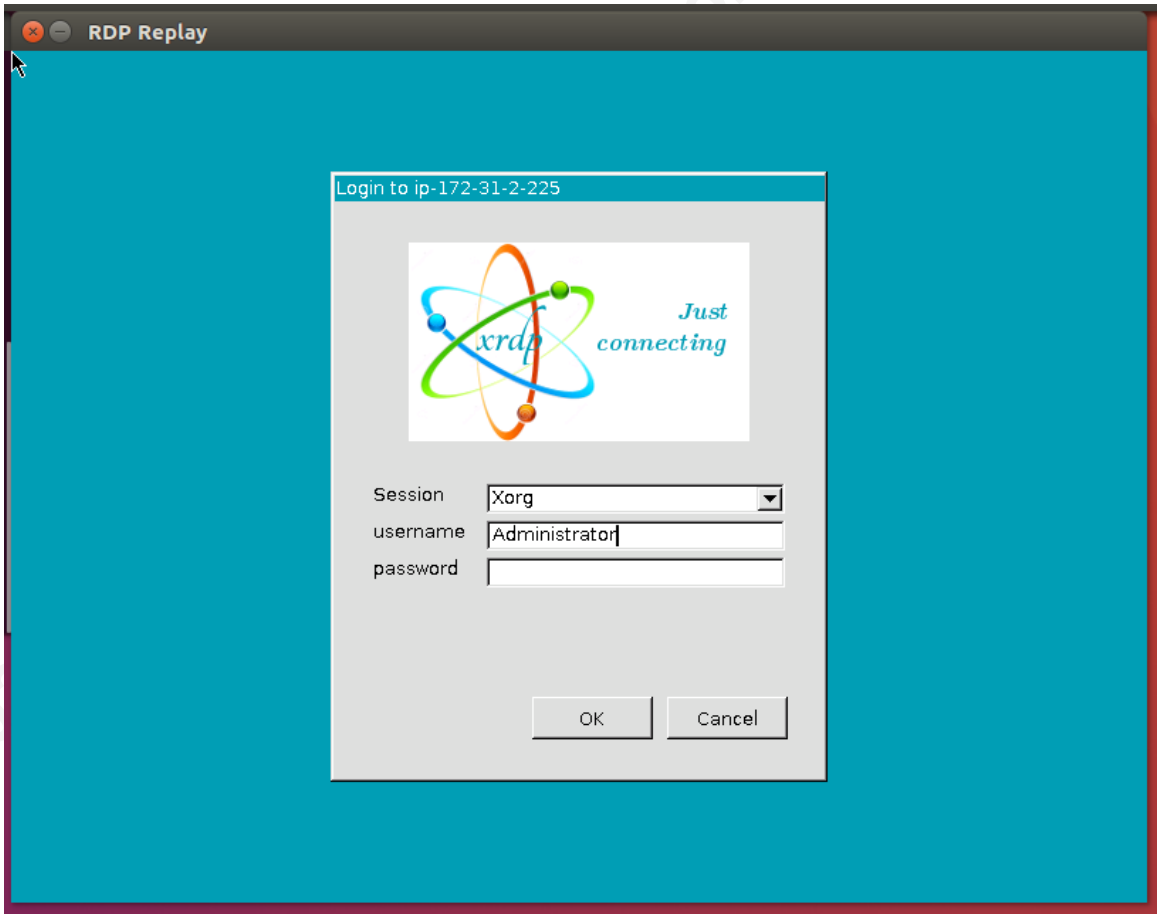

Figure 9. RDP Replay Video Playback Window.

This replay functionality was tested for accuracy in video playback and keystroke logging prior to the deployment of the experiment.

4. Findings and Discussion

4.1. Honeypot Appearance to Reconnaissance

Both honeypots were examined with reconnaissance tools that are commonly used by attackers. Nmap, the Network Mapper, is an open-source network scanning and security tool. Nmap allows the user to scan target hosts by sending novel packets and analyzing the responses to determine host and software characteristics (Lyon, 2011). Nmap is the top active network reconnaissance tool (Poston, 2019). The XRDP Honeypot was scanned with the following command (IP address removed):

```
Nmap -Pn -A XXX.XXX.XXX.XXX
```

The nmap results for the XRDP Honeypot identify the open TCP ports (22 and 3389) and correctly identifies the version of RDP software running:

```
Starting Nmap 7.70 ( https://nmap.org ) at 2020-11-06 00:12 PST
Nmap scan report for ec2-18-220-218-153.us-east-
2.compute.amazonaws.com (XXX.XXX.XXX.XXX)
Host is up (0.087s latency).
Not shown: 998 filtered ports
PORT     STATE SERVICE       VERSION
22/tcp   open  ssh           OpenSSH 8.2p1 Ubuntu 4ubuntu0.1
(Ubuntu Linux; protocol 2.0)
3389/tcp open  ms-wbt-server xrdp
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect
results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 74.52 seconds
```

The PyRDP Honeypot was scanned using the same command as above. This scan did not identify the RDP software running on the host:

```
Starting Nmap 7.70 ( https://nmap.org ) at 2020-11-06 00:07 PST
Nmap scan report for ec2-3-137-198-226.us-east-
2.compute.amazonaws.com (XXX.XXX.XXX.XXX)
Host is up (0.092s latency).
Not shown: 998 filtered ports
PORT     STATE SERVICE        VERSION
22/tcp   open  ssh            OpenSSH 8.2p1 Ubuntu 4ubuntu0.1
(Ubuntu Linux; protocol 2.0)
3389/tcp open  ms-wbt-server?
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
https://nmap.org/cgi-bin/submit.cgi?new-service :
```

```
SF-Port3389-TCP:V=7.70%I=7%D=11/6%Time=5FA50450%P=x86_64-
redhat-linux-gnu%
SF:r(TerminalServerCookie,13,"\x03\0\0\x13\x0e\xd0\0\0\x124\0\x
02\0\x08\0\
SF:x01\0\0\0");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect
results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 226.34 seconds
```

Shodan is another tool that is often used by attackers to conduct reconnaissance on a target to explore potential vulnerabilities and is one of the top passive reconnaissance tools according to Poston (2019). Shodan results for both honeypots also showed distinct differences between the proposed solution and the control. Like the nmap scan, Shodan also showed the XRDP Honeypot TCP ports 22 and 3389 as open, and RDP was identified as the service on port 3389.

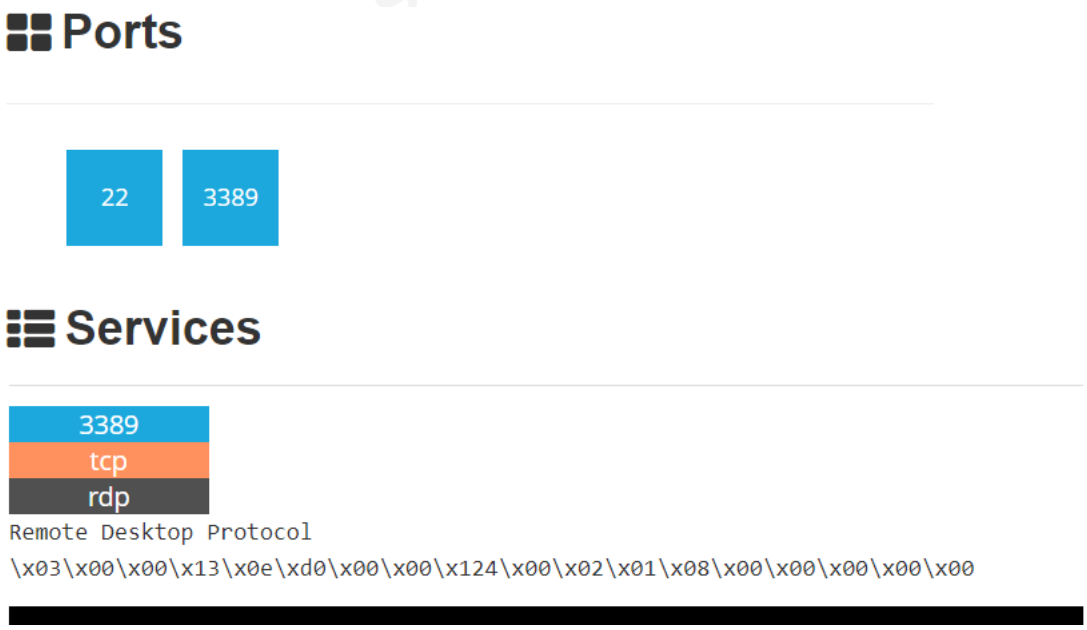

Figure 10. Shodan Results for XRDP Honeypot (OpenSSH omitted).

PyRDP, on the other hand, was not interpreted as an RDP service by Shodan. When scanned with Shodan, the PyRDP results only showed TCP port 22 open, and no service on 3389 was displayed. An on-demand Shodan scan was conducted to verify these results, and traffic from the Shodan scan was observed on the PyRDP Honeypot.

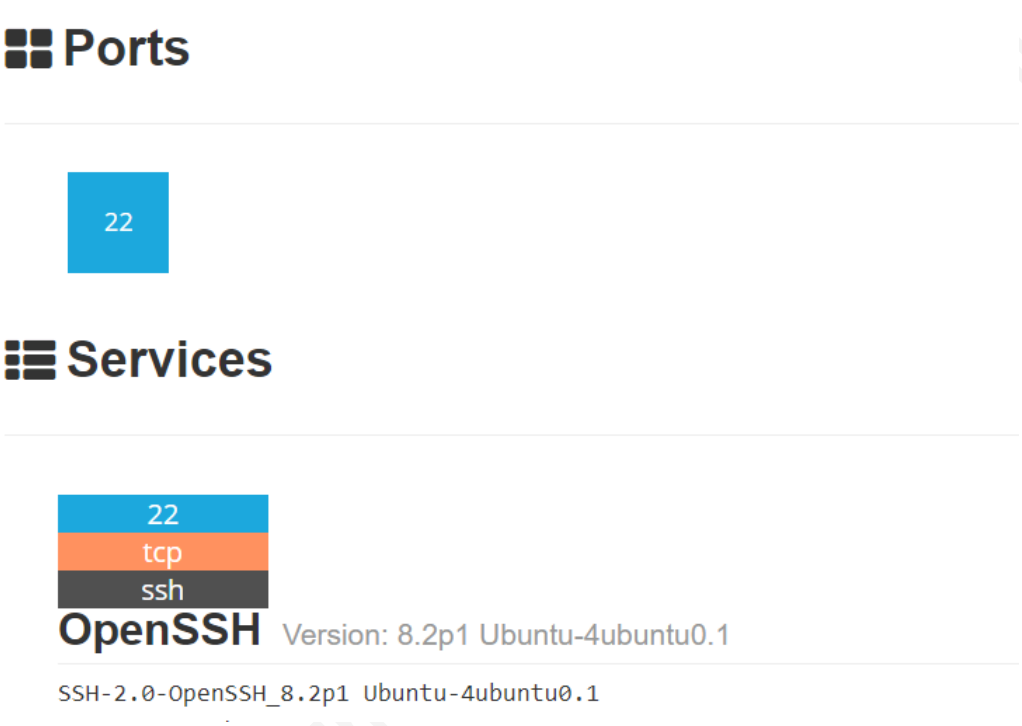

Figure 11. Shodan Results for PyRDP Honeypot

While collecting data during this experiment, a particular hostname was observed accessing both honeypots multiple times, "rdpscan". Open-source research identified this activity to likely be the open-source project rdpscan. Rdpscan is a "quick-and-dirty scanner" that was written to search for CVE-2019-0708 vulnerability in Microsoft Remote Desktop (Graham, 2019). According to Graham, in 2019 there were over 900,000 hosts publicly accessible on the internet vulnerable to exploitation via this vulnerability. The pervasive presence of this tool's use in the observed activity indicates that it is being used widely by attackers in the wild.

Based on the observed activity, rdpscan was compiled and run against the two honeypots. The XRDP Honeypot had the following results:

```
[XRDP Passthrough Host IP] - SAFE - Target appears patched
```

This result indicated that rdpscan did not find the RDP software to be vulnerable to CVE-2019-0708.

Rdpscan was also run against the PyRDP Honeypot and produced the following results:

```
[PyRDP Host IP] - UNKNOWN - SSL - network error
```

This result appeared to show that the vulnerability testing conducted by rdpscan was not compatible with the PyRDP software.

Overall, the reconnaissance tests produced favorable results for the XRDP Honeypot. All three tools showed the honeypot to be a functioning RDP solution, and nmap correctly identified the software being used. Conversely, PyRDP did not appear to be an attractive target when scanned with any of the three tools. Aside from noting that the port was open, nmap did not identify the software running on the PyRDP Honeypot as Windows RDP. Shodan did not even detect that the port was open, and rdpscan, seemingly the most likely avenue for attracting an attacker's attention, encountered an error when scanning the PyRDP Honeypot.

4.2. Activity Observed Targeting XRDP Honeypot

During the collection period, the XRDP Honeypot received 24,379 TCP connections to port 3389. Of these connections, the RDP client/server encryption information was exchanged 2,100 times, and 17 RDP reviewable sessions were established. A session was determined to be reviewable based upon the amount of data transferred during the TCP session. Sessions that were too small to have substantial data transferred were discarded from further analysis.

The vast majority of observed connections, 76.5%, were from the rdpscan tool described above and used the hostname "rdpscan". Two of the sessions (11.7%) observed were interactive sessions that appeared to originate from an interactive client that both sent and received data via RDP. Both sessions failed to complete the login process and interact with the environment. The remaining sessions did not interact with the honeypot once the session was fully established.

4.3. Activity Observed Targeting PyRDP Honeypot

The PyRDP Honeypot received 23,387 TCP connections to port 3389 during the review period. PyRDP attempted to initiate an RDP session with each connection, but the majority of connections did not result in a replay being created. Twenty five replays were created by PyRDP during the collection period.

The PyRDP Honeypot also experienced significant traffic originating from the tool rdpscan (25%), using the hostname "rdpscan". Notably, PyRDP flags the rdpscan activity as "Bluekeep (CVE-2019-0708) scan or exploit attempt detected." In addition to traffic using the hostname "rdpscan", a second hostname, "WIN-UJCESDGS3Q9", was also flagged as "Bluekeep (CVE-2019-0708) scan or exploit attempt detected." All the connections using this hostname originated from the same /31 address range. It is possible that this was also rdpscan activity using a new hostname to bypass blocking based on the rdpscan hostname widely observed in this study. None of the additional sessions interacted with the honeypot beyond establishing the initial connection. Because this honeypot was configured to log the user into the honeypot server automatically without a username or password, it is highly likely that all these sessions were automated and did not involve an interactive user.

5. Recommendations and Implications

5.1. Recommendations for Practice

Use of the XRDP Honeypot as presented here is encouraged for other researchers based upon the results of this experiment, though collection for a longer period of time would likely yield results of additional interest and value. The configurations in this paper were effective both in collecting data and appearing to be a legitimate XRDP installation when examined with various reconnaissance tools. An attacker would likely view this as a legitimate system.

Two of the sessions collected by the XRDP Honeypot appeared to be interactive based upon the keyboard input that was sent to the server before the session was terminated. If the XRDP Honeypot had restricted input to the password field alone, instead of the username field and the password field, these would have successfully authenticated to Desktop sessions. Changing the username settings in xrdp.ini to the following would enable this functionality:

```
username=Administrator
```

By removing the "ask" value, it will disable the user's ability to edit this field and will force the use of the Administrator account. Making this change could increase the number of successful interactive sessions in future honeypot operations.

Based upon the sheer volume of scanning activity searching for systems vulnerable to Bluekeep (CVE-2019-0708), it would be of value for the security community to continue assessing the scope of vulnerable systems addressable from the internet. Individual organizations should also assess their own network's exposure to this vulnerability, as it appears to be pervasively scanned for by attackers.

5.2. Implications for Future Research

After this study, questions remain unanswered regarding attacker activity in non-standard environments, but the XRDP Honeypot architecture proved effective for collecting this type of data. Longer periods of collection and analysis are likely needed to determine answers to these questions.

Further research into attracting malicious users is also needed, as this is not well explored in existing research. An analysis of actions that can be taken to draw malicious users to the honeypot would be valuable to the security community. Ideas considered during this study to increase traffic to the honeypots include registering domain names that would point to the honeypots and posting information about the "vulnerable server" in known hacking forums. These ideas and others should be investigated in future research.

PyRDP currently identifies attempts to exploit Bluekeep that are targeting the honeypot. If possible, implementing this vulnerability into the honeypot, instead of merely alerting to its attempt would yield a wealth of attacker interactions. In addition to adding functionality, it would be beneficial to the project to fix the fingerprinting/scanning issues identified in the research above. If the server does not appear to be a functioning RDP server when attackers are scanning, they may avoid interacting with the environment.

Finally, based upon the prevalence of the rdpscan tool in the observed traffic to both honeypots, a honeypot that purposefully presents CVE-2019-0708 for exploitation would likely find a large amount of success in attacker engagement.

6. Conclusion

The proposed XRDP Honeypot architecture presented in this paper is a viable solution for an RDP honeypot. It fared well when compared against existing open-source honeypot solution PyRDP using reconnaissance tools to assess both honeypots as targets. Both honeypots received similar levels of traffic and attacker engagement during the test period. The XRDP Honeypot and PyRDP offer similar features to researchers in two different target environments. The XRDP Honeypot offers a unique capability to researchers and network defenders seeking to learn more about attackers targeting Linux-based GUI environments, and it is built completely from open-source components. Future research conducted using this honeypot architecture could test several other variables related to attracting attacker interaction, such as mimicking specific vulnerabilities or publicizing the vulnerable RDP service through a variety of different methods.

References

Atlas VPN. (2020). RDP Attacks Surged by 330% in the US. *Software World*, *51*(3), 24. http://www.softwareworldpublication.com/

Hodgman, R. (2016, March 1). *The Attacker's Dictionary*. Rapid7 Blog. https://blog.rapid7.com/2016/03/01/the-attackers-dictionary/

Neutrino Labs. (2020, October 19). *xrdp.ini man page*. GitHub. https://github.com/neutrinolabs/xrdp/blob/devel/docs/man/xrdp.ini.5.in

Linux man-pages project. (2016, April 1). *pam_permit(8) - Linux manual page*. Man7.Org. https://man7.org/linux/man-pages/man8/pam_permit.8.html

Rupp, K. (2016, September 26). *NAT with Linux and iptables - Tutorial (Introduction)*. KarlRupp.net. https://www.karlrupp.net/en/computer/nat_tutorial

Elliott, S. (2014, October 30). *RDP Replay*. Context Information Security. https://www.contextis.com/en/blog/rdp-replay

Elliott, S. (2016, June 22). *RDP Replay*. Github. https://github.com/ctxis/RDP-Replay

National Institute of Standards and Technology. (2007, August). *NIST Special Publication on Computer Security SP 800-78 Rev 1*. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-78-1.pdf

Beaulieu, A. (2020, October 19). *Announcing PyRDP 1.0*. GoSecure. https://www.gosecure.net/blog/2020/10/20/announcing-pyrdp-1/

Lauri Marc AhlmanAuthor Name, email@address

Gonzalez, E. (2019, November 18). *Enhanced RDP Security - failed negotitation · Issue #168 · GoSecure/pyrdp*. GitHub. https://github.com/GoSecure/pyrdp/issues/168

Lyon, G. (2011). *Chapter 15. Nmap Reference Guide | Nmap Network Scanning*. Nmap.Org. https://nmap.org/book/man.html

Poston, H. (2019, September 17). *Top 10 network recon tools*. Infosec Resources. https://resources.infosecinstitute.com/topic/top-10-network-recon-tools/

Graham, R. (2019, June 8). *robertdavidgraham/rdpscan*. GitHub. https://github.com/robertdavidgraham/rdpscan

Wang, Z. H., Liu, C. G., Qiu, J., Tian, Z. H., Cui, X., & Su, S. (2018). Automatically Traceback RDP-Based Targeted Ransomware Attacks. *Wireless Communications and Mobile Computing*, *2018*, 1–13. https://doi.org/10.1155/2018/7943586

RDP bug in older Windows versions leaves millions vulnerable to 'wormable' attack. (2019). *Network Security*, *2019*(6), 1–2. https://doi.org/10.1016/s1353-4858(19)30065-0

Bryce, C. (2020, February 15). *3-Step RDP Honeypot: Step 1 | Honeypot Setup | Pythonic Forensics*. Medium. https://medium.com/pythonic-forensics/3-step-rdp-honeypot-part-1-honeypot-setup-b5f01af19f4d

Gonzalez, E. (2020, January 23). *RDP Man-in-the-Middle – Smile! You're on Camera*. GoSecure. https://www.gosecure.net/blog/2018/12/19/rdp-man-in-the-middle-smile-youre-on-camera/

Lauri Marc AhlmanAuthor Name, email@address

Appendix A
Code and Configurations to Deploy the Honeypots

For each file included in this section, a page break will begin the contents of the file, followed by a comment with the name of the file. Comments will begin with a "#", and the actual text of the files will be presented in `Courier New` font. The following files are provided in Appendix A:

#install_xrdp.yml – Ansible Script to Install XRDP Host, must change passthrough_ip.

```
---
- name: Prepare for xrdp - Linux Host
  hosts: all
  vars:
    passthrough_ip: 0.0.0.0
  tasks:
  - name: ping
    ping:
  - name: Update packages
    become: true
    apt:
      name: "*"
      state: latest
      update_cache: yes
  - name: Install Required Packages
    become: true
    apt:
      pkg:
      - xrdp
      - ubuntu-desktop
      - iptables-persistent
  - name: Replace pam.d/xrdp-sesman with template
    become: yes
    template:
      src: xrdp-sesman
      dest: /etc/pam.d/xrdp-sesman
      owner: root
      group: root
      mode: 0644
  - name: Add username=askAdministrator to xrdp.ini
    become: yes
    replace:
      path: /etc/xrdp/xrdp.ini
      replace: 'username=askAdministrator'
      regexp: '^username=ask'
  - name: Update /etc/xrdp/xrdp.ini
    become: yes
    lineinfile:
      path: /etc/xrdp/xrdp.ini
      line: security_layer=rdp
      regexp: 'security_layer=negotiate'
  - name: Modify xrdp/sesman.ini 1/3
    become: yes
    lineinfile:
      path: /etc/xrdp/sesman.ini
      line: 'KillDisconnected=True'
```

Lauri Marc AhlmanAuthor Name, email@address

```
    regexp: '^KillDisconnected=*'
- name: Modify xrdp/sesman.ini 2/3
  become: yes
  lineinfile:
    path: /etc/xrdp/sesman.ini
    line: 'DisconnectedTimeLimit=60'
    regexp: '^DisconnectedTimeLimit=*'
- name: Modify xrdp/sesman.ini 3/3
  become: yes
  lineinfile:
    path: /etc/xrdp/sesman.ini
    line: 'IdleTimeLimit=300'
    regex: '^IdleTimeLimit=*'
- name: Add Administrator User
  become: yes
  user:
    name: Administrator
    shell: /bin/bash
- name: Add xrdp user to sslcert group
  become: yes
  user:
    name: xrdp
    groups: xrdp,ssl-cert
- name: download rsakeys.ini for key extraction
  become: yes
  fetch:
    src: /etc/xrdp/rsakeys.ini
    dest: keyfile.txt
- name: Update /etc/X11/Xwrapper.config
  become: yes
  lineinfile:
    path: /etc/X11/Xwrapper.config
    line: allowed_users=anybody
    regexp: 'allowed_users=console'
- name: Restart xrdp
  become: yes
  service:
    name: xrdp
    enabled: yes
    state: restarted
- name: Restart xrdp-sesman
  become: yes
  service:
    name: xrdp-sesman
    enabled: yes
    state: restarted
- name: Disable ufw
```

```
  become: yes
  service:
    name: ufw
    enabled: no
    state: stopped
- name: Enable iptables
  become: yes
  service:
    name: iptables
    enabled: yes
    state: started
- name: Enable ip6tables
  become: yes
  service:
    name: ip6tables
    enabled: yes
    state: started
- name: Copy iptables script to remote host
  become: yes
  template:
    src: iptables_xrdp.sh
    dest: /usr/local/sbin/iptables.sh
    owner: root
    mode: 0700
- name: Create iptables rules
  become: yes
  command: /usr/local/sbin/iptables.sh
- name: Save iptables rules
  become: yes
  # ignore_errors: True
  shell: iptables-save > /etc/iptables/rules.v4
- name: Save ip6tables rules
  become: yes
  # ignore_errors: True
  shell: ip6tables-save > /etc/iptables/rules.v6
```

#xrdp-sesman – Pam configuration file for XRDP Host

```
#%PAM-1.0
#@include common-auth
#@include common-account
@include common-session
#@include common-password

account   required  pam_permit.so
auth required  pam_permit.so
password  required  pam_permit.so
```

#iptables_xrdp.sh – iptables configuration script for XRDP Host

```
#!/bin/bash
# iptables config script

# Flush old rules
iptables -F

# Set Default Policy to Drop
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Allow loopback traffic - drop anything else from
127.0.0.0/8
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A INPUT -s 127.0.0.0/8 -j DROP

# Allow inbound ssh connections
iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT

# Allow inbound RDP connections from xrdp-passthrough
iptables -A INPUT -s {{passthrough_ip}} -p tcp --dport 3389
-m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

# Allow inbound traffic related to connections started
locally
iptables -A INPUT -p tcp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p udp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p icmp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT

# Allow outbound traffic
iptables -A OUTPUT -p tcp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p udp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p icmp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT

# Log other traffic before dropping
iptables -A INPUT -j LOG
iptables -A OUTPUT -j LOG
```

Lauri Marc AhlmanAuthor Name, email@address

```
iptables -A FORWARD -j LOG


# ipv6
# Flush old rules
ip6tables -F

# Drop all ipv6 traffic
ip6tables -P INPUT DROP
ip6tables -P OUTPUT DROP
ip6tables -P FORWARD DROP

# Allow loopback traffic - drop anything else from
127.0.0.0/8
ip6tables -A INPUT -i lo -j ACCEPT
ip6tables -A OUTPUT -o lo -j ACCEPT
```

#Keyparse.py – Python script to parse rsakeys.ini file from XRDP Host

```
import sys

if len(sys.argv) != 2:
    print('Usage: keyparse.py [keyfile]')
    exit(0)

filename = sys.argv[1]

firstline = True
KeyName = 'CustomLSAKey'
PublicKey = ''
PrivateKey = ''
RSAMetaData = '52534131080100000080000ff00000001000100'
Padding = '0000000000000000'


with open(filename, 'r') as infile:
    for line in infile:
        if firstline:
            if line.strip() == '[keys]':
                firstline = False
            else:
                print('Improper file format.')
                exit(1)
        else:
            if line.split('=')[0] == 'pri_exp':
                privateexponent =
line.split('=')[1].strip()
                privateexponent =
privateexponent.replace('0x','').replace(',','')
                print('Private Exponent Found:
'+privateexponent)
                PrivateKey = privateexponent
            elif line.split('=')[0] == 'pub_mod':
                publicmodulus = line.split('=')[1].strip()
                publicmodulus =
publicmodulus.replace('0x','').replace(',','')
                print('Public Modulus Found:
'+publicmodulus)
                PublicKey =
RSAMetaData+publicmodulus+Padding
print('LSASecret Generated:
'+KeyName+','+PublicKey+','+PrivateKey)
print(len(PublicKey))
```

Lauri Marc AhlmanAuthor Name, email@address

```
with open('key.txt', 'w') as outfile:
    outfile.write(KeyName+','+PublicKey+','+PrivateKey)

print('Key saved to key.txt')
```

install-xrdp-passthrough.yml – Ansible Script to install XRDP Passthrough Host, must change xrdp_ip.

```
---
- name: Prepare VPS for Passthrough to xrdp - Linux Host
  hosts: all
  vars:
    xrdp_ip: 0.0.0.0
  tasks:
  - name: ping
    ping:
  - name: Update packages
    become: true
    apt:
      name: "*"
      state: latest
      update_cache: yes
  - name: Install Required Packages
    become: true
    apt:
      pkg:
      - tcpdump
      - iptables-persistent
  - name: Enable IPv4 Forwarding
    become: yes
    lineinfile:
      path: /etc/sysctl.conf
      line: net.ipv4.ip_forward=1
      regexp: \#net.ipv4._ip_forward=1
  - name: Disable ufw
    become: yes
    service:
      name: ufw
      enabled: no
      state: stopped
  - name: Enable iptables
    become: yes
    service:
      name: iptables
      enabled: yes
      state: started
  - name: Enable ip6tables
    become: yes
    service:
      name: ip6tables
      enabled: yes
      state: started
  - name: Copy iptables script to remote host
```

Lauri Marc AhlmanAuthor Name, email@address

```
    become: yes
    template:
      src: iptables_xrdp-passthrough.sh
      dest: /usr/local/sbin/iptables.sh
      owner: root
      mode: 0700
  - name: Create iptables rules
    become: yes
    command: /usr/local/sbin/iptables.sh
  - name: Save iptables rules
    become: yes
    # ignore_errors: True
    shell: iptables-save > /etc/iptables/rules.v4
  - name: Save ip6tables rules
    become: yes
    # ignore_errors: True
    shell: ip6tables-save > /etc/iptables/rules.v6
  - name: Create Data Dir
    file:
      path: /home/user/Data
      state: directory
      mode: 0755
  - name: Create capture script
    copy:
      dest: /home/user/capture.sh
      content: tcpdump -i eth0 -w /home/user/Data/dump-
2020%m%d-%H-%M-%S.pcap -G 1800 tcp port 3389
      mode: 0700
```

#iptables_xrdp-passthrough.sh – iptables configuration script for XRDP Passthrough.

```
#!/bin/bash
# iptables config script

# Flush old rules
iptables -F

# Set Default Policy to Drop
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Allow loopback traffic - drop anything else from
127.0.0.0/8
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A INPUT -s 127.0.0.0/8 -j DROP

# Allow inbound ssh connections
iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT

# Forward inbound RDP connections to xrdp
iptables -t nat -A PREROUTING -p tcp --dport 3389 -j DNAT -
-to-destination {{xrdp_ip}}:3389
iptables -A FORWARD -p tcp --dport 3389 -j ACCEPT
iptables -A FORWARD -p tcp --sport 3389 -j ACCEPT
iptables -t nat -A POSTROUTING -j MASQUERADE

# Allow inbound traffic related to connections started
locally
iptables -A INPUT -p tcp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p udp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p icmp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT

# Allow outbound traffic
iptables -A OUTPUT -p tcp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p udp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p icmp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT
```

Lauri Marc AhlmanAuthor Name, email@address

```
# Log other traffic before dropping
iptables -A INPUT -j LOG
iptables -A OUTPUT -j LOG
iptables -A FORWARD -j LOG


# ipv6
# Flush old rules
ip6tables -F

# Drop all ipv6 traffic
ip6tables -P INPUT DROP
ip6tables -P OUTPUT DROP
ip6tables -P FORWARD DROP
```

install-rdp-replay.yml – Ansible script to deploy RDP Replay Host.

```
---
- name: Prepare for Installation of RDPReplay - Linux Host
  hosts: all
  tasks:
  - name: ping
    ping:
  - name: Update packages
    become: true
    apt:
      name: "*"
      state: latest
      update_cache: yes
  - name: Install Required Packages
    become: true
    apt:
      pkg:
      - build-essential
      - git-core
      - cmake
      - libssl-dev
      - libx11-dev
      - libxext-dev
      - libxinerama-dev
      - libxcursor-dev
      - libxdamage-dev
      - libxv-dev
      - libxkbfile-dev
      - libasound2-dev
      - libcups2-dev
      - libxml2
      - libxml2-dev
      - libxrandr-dev
      - libgstreamer0.10-dev
      - libgstreamer-plugins-base0.10-dev
      - libavutil-dev
      - libavcodec-dev
      - libavformat-dev
      - libpcap-dev
      - libreadline-dev
      - iptables-services

  - name: Clone PyRDP Git Repo
    git:
      repo: https://github.com/ctxis/RDP-Replay.git
      dest: ~/rdpreplay
```

Lauri Marc AhlmanAuthor Name, email@address

```
- name: Make RDPReplay
  shell: make
  args:
    chdir: ~/rdpreplay
```

install-pyrdp-linux.yml – Ansible script to configure PyRDP Host.

```
---
- name: Prepare for Installation of PyRDP - Linux Host
  hosts: all
  tasks:
  - name: ping
    ping:
  - name: Update packages
    become: true
    apt:
      name: "*"
      state: latest
      update_cache: yes
  - name: Install Required Packages
    become: true
    apt:
      pkg:
      - python3
      - python3-pip
      - python3-dev
      - python3-setuptools
      - python3-venv
      - build-essential
      - git
      - openssl
      - libdbus-1-dev
      - libdbus-glib-1-dev
      - libgl1-mesa-glx
      - notify-osd
      - dbus-x11
      - libxkbcommon-x11-0
      - libxcb-xinerama0
      - iptables-persistent
  - name: Clone PyRDP Git Repo
    git:
      repo: https://github.com/gosecure/pyrdp.git
      dest: ~/pyrdp
  - name: Install virtualenv
    become: true
    pip:
      name: virtualenv
      executable: pip3
  - name: Install PyRDP requirements in venv
    pip:
      requirements: ~/pyrdp/requirements.txt
      virtualenv: ~/pyrdp/venv
```

Lauri Marc AhlmanAuthor Name, email@address

```
      virtualenv_python: python3
  - name: "Copy virtualenv wrapper file"
    copy:
      src: ./pyvenv
      dest: "~/pyrdp/venv/bin/pyvenv"
      owner: user
      group: user
      mode: 0755
  - name: Disable ufw
    become: yes
    service:
      name: ufw
      enabled: no
      state: stopped
  - name: Enable iptables
    become: yes
    service:
      name: iptables
      enabled: yes
      state: started
  - name: Enable ip6tables
    become: yes
    service:
      name: ip6tables
      enabled: yes
      state: started
  - name: Copy iptables script to remote host
    become: yes
    template:
      src: iptables_pyrdp.sh
      dest: /usr/local/sbin/iptables.sh
      owner: root
      mode: 0700
  - name: Create iptables rules
    become: yes
    command: /usr/local/sbin/iptables.sh
  - name: Save iptables rules
    become: yes
    # ignore_errors: True
    shell: iptables-save > /etc/iptables/rules.v4
  - name: Save ip6tables rules
    become: yes
    # ignore_errors: True
    shell: ip6tables-save > /etc/iptables/rules.v6

# Use virtualenv
- name: Finish installation in venv
```

Lauri Marc AhlmanAuthor Name, email@address

```
  hosts: all
  vars:
    ansible_python_interpreter: "~/pyrdp/venv/bin/pyvenv"
  tasks:
  - name: "Guard code, so we are more certain we are in a
virtualenv"
    shell: echo $VIRTUAL_ENV
    register: command_result
    failed_when: command_result.stdout == ""
  - name: Install requirements in venv
    shell:
      cmd: pip3 install -U pip setuptools wheel
      chdir: ~/pyrdp/
  - name: Install PyRDP in venv
    shell:
      cmd: pip3 install -U '.[full]'
      chdir: ~/pyrdp/
```

iptables_pyrdp.sh – iptables configuration script for PyRDP Host.

```
#!/bin/bash
# iptables config script

# Flush old rules
iptables -F

# Set Default Policy to Drop
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Allow loopback traffic - drop anything else from
127.0.0.0/8
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A INPUT -s 127.0.0.0/8 -j DROP

# Allow inbound ssh connections
iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT

# Forward inbound RDP connections to xrdp
iptables -A INPUT -p tcp --dport 3389 -m conntrack --
ctstate NEW,ESTABLISHED -j ACCEPT

# Allow inbound traffic related to connections started
locally
iptables -A INPUT -p tcp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p udp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p icmp -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT

# Allow outbound traffic
iptables -A OUTPUT -p tcp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p udp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -p icmp -m conntrack --ctstate
NEW,ESTABLISHED -j ACCEPT

# Log other traffic before dropping
iptables -A INPUT -j LOG
iptables -A OUTPUT -j LOG
```

Lauri Marc AhlmanAuthor Name, email@address

```
iptables -A FORWARD -j LOG


# ipv6
# Flush old rules
ip6tables -F

# Drop all ipv6 traffic
ip6tables -P INPUT DROP
ip6tables -P OUTPUT DROP
ip6tables -P FORWARD DROP
```

Example host file for Ansible Scripts. Host IP, SSH key file path, and/or passwords must be updated. Uncomment ansible_password and ansible_sudo_pass if not using SSH key.

```
xrdplinuxhost:
  hosts:
    xrdphost:
      ansible_host: 0.0.0.0
      ansible_user: ubuntu
#      ansible_password: xrdpuberpassphrase
#      ansible_sudo_pass: xrdpuberpassphrase
      ansible_ssh_private_key_file: xrdp_client.pem
      ansible_ssh_extra_args: '-o StrictHostKeyChecking=no'
```