

School of Design and Informatics
Abertay University
DUNDEE, DD1 1HG, UK
Ethical Hacking, BSc (Hons) 2021/22

**Evaluating and Identifying the Most Effective Detection Methods
for Purple Team Engagements Against RDP Honeypots**

Christopher Di-Nozzi

Word Count: 25001

Supervised by Jamie O'Hare

Abstract

Security experts partaking in offensive engagements need to be able to detect whether the system they are attacking is legitimate or a honeypot. Various methods need to be identified and evaluated to determine which of these is the most suitable. With the rise of popularity amongst Remote Desktop Systems, being able to properly research and detect attacks against RDP is more important now than ever before. There is little research into RDP honeypots, and those honeypots that do exist are lacking sophistication. Furthermore, no research exists into detecting RDP honeypots, thus proving that any that do exist are likely unhardened through thorough research. The aim of this paper is to evaluate and identify the most effective detection methods for purple team engagements against contemporary RDP honeypots.

Due to a lack of research within this field, a methodology was adapted from related research. A literature review was performed and highlighted several different techniques that could be tested against both RDP honeypots and legitimate services to find the most effective method of fingerprinting RDP honeypots, that is the method that is the most accurate while being quick and quiet from a network level. Three main experiments were carried out, testing network fingerprinting, latency, and the behaviour of the systems, and the results were captured and presented. The main finding of the results were key differences between the network fingerprint of honeypots and legitimate systems, as well as some latency timing differences for TCP packets for some of the honeypots. The behavioural results were lacking but it was found the one of the honeypots exhibited noticeable honeypot behaviour by redeploying itself after a connection was ended.

These results were then further analysed to determine which of these techniques would be most effective for a purple team to use in an offensive engagement. It was found that fingerprinting, particularly with Xprobe2's default settings, returned with most accurate assessment of a systems underlying operating system, a key distinguisher between real and fake RDP services that are often expected to be running on Windows architecture. It was also the quickest and one of the quietest of the scans. Nmap was also found to be very effective, providing more in-depth details about systems but took longer to complete and generated much more noise, especially with the intense scans. Latency testing was found to be ineffective at identifying fake systems from real ones, and the behavioural analyse fell short due to the lack of a real-world enterprise environment encompassing it, however, behavioural analysis in a real-world scenario may still be effective. These results show a lack of care when developing RDP honeypots. Both Windows based honeypots, AHA and rdppot, were easily distinguishable from their legitimate counterparts. This research could be continued further by automating the techniques outlined in this paper into a tool to be used by offensive engagers, as well as continuing to research these methods in more depth and explore new ones like virtual machine and logging system detection.

Keywords: Remote Desktop Protocol, RDP, honeypots, detection, fingerprinting, purple team, offensive engagement, network analysis, open source

Acknowledgements

I would like to express my deepest appreciations to my supervisor, Jamie O'Hare, who has provided guidance, advice, and encouragement since being assigned to supervise this project. I would also like to extend my sincere thanks to Xavier Bellekens who took time to discuss topics surrounding this research and helped aid in the overall direction of this project. Thanks also to my fiancé, Ruth, who has, and continues to, support me in everything that I do. A final thanks to my family and friends who have supported me in many ways throughout this year.

Table of Contents

1. Introduction.....	10
1.1 Background and Context.....	10
1.2 Aim, Research Questions, and Objectives	11
1.3 Scope.....	12
1.4 Structure	12
2. Literature Review	14
2.1 Introduction	14
2.2 Overview of Honeypots and Recent Developments.....	14
2.3 Remote Desktop Protocol (RDP)	16
2.4 Existing Honeypot Detection Techniques	19
2.4.1 Overview of current research.....	19
2.4.1 Fingerprinting.....	20
2.4.2 Behaviour	23
2.4.3 Virtualisation.....	24
2.4.4 Latency.....	26
2.4.5 Default Configurations.....	28
2.5 Honeypot detection countermeasures.....	28
2.6 Conclusion.....	30
3. Methodology.....	32
3.1 Introduction	32
3.2 Research	32
3.3 Design	32
3.3.1 Honeypots	32
3.3.2 Legitimate Services.....	33
3.3.3 Virtual Machines.....	33
3.3.4 Network.....	34
3.4 Implementation.....	34
3.4.1 Windows 10 RDP.....	35
3.4.2 XRDP	35
3.4.3 Windows XP RDP.....	36
3.4.4 ad-honeypot-autodeploy.....	37
3.4.5 rdppot	39
3.4.6 Octopus	42
3.5 Experimentation.....	43
3.5.1 Fingerprinting.....	43
3.5.2 Latency.....	44
3.5.3 Behaviour	44
3.6 Evaluation	45
3.7 Conclusion.....	45
4. Results.....	46
4.1 Introduction	46
4.2 Experiment 1 - Fingerprinting.....	46

4.2.1	NMAP	46
4.2.2	Xprobe2.....	50
4.3	Experiment 2 - Latency.....	52
4.3.1	ICMP	52
4.3.2	TCP	54
4.4	Experiment 3 - Behaviour.....	56
4.4.1	File Persistence.....	56
4.4.2	Further Attacks.....	59
5.	Discussion.....	60
5.1	Existing Detection Methods.....	60
5.1.1	Fingerprinting.....	60
5.1.2	Behaviour	61
5.1.3	Latency.....	62
5.1.4	Default Configurations.....	62
5.1.5	Conclusion.....	62
5.2	Most effective RDP honeypot detection method.....	63
5.2.1	Fingerprinting.....	63
5.2.2	Latency.....	64
5.2.3	Behaviour	64
5.2.4	Conclusion.....	65
5.3	Technique, Honeypot, and Network Improvements	65
5.3.1	Detection Method.....	65
5.3.2	RDP Honeypot	66
5.3.3	Networking	66
5.4	Future Work	67
6.	Conclusions	69
7.	Bibliography.....	71
8.	Appendix.....	76
8.1	Appendix A - Risk Analysis.....	76
8.2	Appendix B - Ethics Report.....	76
8.3	Appendix C - TCP and ICMP latency scripts	77
8.3.1	TCP	77
8.3.2	ICMP.....	78
8.4	Appendix E - PCAPNG Cleaning Script.....	78
8.5	Appendix F - NMAP Fingerprinting Results.....	79
8.5.1	Rdppot vs Windows XP - Quick Scan.....	79
8.5.2	Octopus vs XRDP - Quick Scan	79
8.5.3	AHA vs Windows 10 - Quick Scan	80
8.5.4	Rdppot vs Windows XP - Intense Scan	81
8.5.5	Octopus vs XRDP - Intense Scan.....	82
8.5.6	AHA vs Windows 10 - Intense Scan.....	82
8.6	Appendix G - Xprobe2 Fingerprinting Results	83
8.6.1	Rdppot vs Windows XP - Normal	84
8.6.2	Octopus vs XRDP- Normal.....	86
8.6.3	AHA vs Windows 10- Normal.....	88
8.6.4	Rdppot vs Windows XP - Port.....	90
8.6.5	Octopus vs XRDP - Port	92
8.6.6	AHA vs Windows 10 - Port.....	94

8.7	Appendix H - ICMP Latency	95
8.8	Appendix I - TCP Latency.....	98
8.9	Appendix J - File Persistence Results.....	100
8.9.1	Rdppot.....	101
8.9.2	Octopus	102
8.9.3	AHA.....	103
8.10	Appendix K - Further Attack Results	104
8.10.1	Rdppot	104
8.10.2	Octopus.....	105
8.10.3	AHA	105

Table of Figures

Figure 1: Weekly average traffic to the Remote Desktop category on Trust Radius, a software review service.	10
Figure 2: An explanation of the key roles in Red, Purple, and Blue teaming.	11
Figure 3: The Lockheed Martin Cyber Kill Chain, (Lockheed Martin, 2022).	16
Figure 4: A graph of different operating systems (OS) running RDP facing the internet.	17
Figure 5: The divide of malicious and benign RDP traffic shown by GreyNoise.	17
Figure 6: A bar graph showing the login attempts per day over the 30-day period.	18
Figure 7: The vulnerabilities found in each honeypot, along with remediations that could be made.	20
Figure 8: A table of NMAP scans used to scan honeypots, (Mohammadzadeh, et al., 2013). 21	
Figure 9: The command used to run KProcCheck and detect Sebek on Windows, (Holz & Raynal, 2005).	22
Figure 10: The proposed framework laid out visually, including both branches.	23
Figure 11: The difference in the stack between a non-emulated and emulated Windows XP, (Issa, 2012).	25
Figure 12: The virtual network used to test round trip times against the honeynet.	26
Figure 13: A visualisation of the classifier used by the research to determine if a link was real or virtual.	27
Figure 14: Different NMAP commands used to test the fuzzing technique.	29
Figure 15: The new honeypot detection counter measure developed by the researchers.	29
Figure 16: The accuracy of the D-FRI tool against different scanning tools.	30
Figure 17: Virtual Network design that will be used during research.	34
Figure 18: Windows 10 RDP connected to via the Attack machine.	35
Figure 19: XRDP connected to via the Attack machine.	36
Figure 20: Windows XP RDP connected to via the Attack machine.	37
Figure 21: AHA connected to via the Attack machine.	39
Figure 22: Modifications made to the main.py file on lines 23-25 and line 38.	40
Figure 23: Setting the memory and CPU specifications on the VM.	40
Figure 24: Setting the hard disk size on the VM.	41
Figure 25: Setting the network adapter on the VM.	41
Figure 26: rdppot connected to via the Attack machine.	42
Figure 27: Octopus connected to via the Attack machine.	43
Figure 28: Nmap quick scan results for rdppot.	46
Figure 29: Nmap quick scan results for Windows XP.	46
Figure 30: Nmap intense scan results for rdppot.	47
Figure 31: Nmap intense scan results for Windows XP.	47
Figure 32: Nmap intense scan for ad-honeypot-autodeploy (AHA).	48
Figure 33: Nmap intense scan for Windows 10.	48
Figure 34: NMAP intense scan for Octopus.	49
Figure 35: NMAP intense scan against XRDP.	49
Figure 36: A bar chart of the time to complete each Nmap scan against each service.	50
Figure 37: A bar chart of the number of network packets transferred during each scan against each service.	51
Figure 38: A bar chart showing the time to complete each Xprobe2 scan against each service.	52
Figure 39: A box plot of the round-trip time for ICMP pings against each service.	53
Figure 40: Time to complete ICMP ping test against each service.	53

Figure 41: A boxplot of the round-trip time for TCP pings to each service.	54
Figure 42: Number of TCP packets sent to each device during experiments	55
Figure 43: Time to complete TCP ping test against each service.	56
Figure 44: File "file.txt" created on the device and checked using the 'dir' command.	57
Figure 45: The same directory after reconnecting, showing the honeypot has been tampered with.	58
Figure 46: The file being created inside the AHA honeypot.	58
Figure 47: The file persisting on AHA after 10 minutes.	59
Figure 48: Result of quick scan against rdppot.	79
Figure 49: Result of quick scan against Windows XP.	79
Figure 50: NMAP quick scan against Octopus.	79
Figure 51: NMAP quick scan against XRDP.	80
Figure 52: NMAP quick scan results against AHA.	80
Figure 53: NMAP quick scan results against Windows 10.	81
Figure 54: Results of intense Nmap scan against rdppot.	81
Figure 55: Results of intense Nmap scan against Windows XP.	81
Figure 56: Results of the intense Nmap scan against Octopus.	82
Figure 57: Results of the intense Nmap scan against XRDP.	82
Figure 58: Results of the intense Nmap scan against AHA.	83
Figure 59: Results of the intense Nmap scan against Windows 10.	83
Figure 60: Result of Xprobe2 normal scan against rdppot.	84
Figure 61: Result of Xprobe2 normal scan against Windows 10.	85
Figure 62: Result of Xprobe2 normal scan against Octopus.	86
Figure 63: Result of Xprobe2 normal scan against XRDP.	87
Figure 64: Result of Xprobe2 normal scan against AHA.	88
Figure 65: Result of Xprobe2 normal scan against Windows 10.	89
Figure 66: Result of Xprobe2 port scan against rdppot.	90
Figure 67: Result of Xprobe2 port scan against Windows XP.	91
Figure 68: Result of Xprobe2 port scan against Octopus.	92
Figure 69: Result of Xprobe2 port scan against XRDP.	93
Figure 70: Result of Xprobe2 port scan against AHA.	94
Figure 71: Result of Xprobe2 port scan against Windows 10.	95
Figure 72: Creating the file on rdppot to test for file persistence.	101
Figure 73: The file no longer being found after reconnecting.	101
Figure 74: Creating the file on Octopus to test for file persistence.	102
Figure 75: The file is still present after reconnecting.	102
Figure 76: Creating the file on AHA to test for file persistence.	103
Figure 77: The file is still present after reconnecting.	103
Figure 78: Result of attempting a further attack from the rdppot honeypot, successfully accessing another machine via FTP.	104
Figure 79: Result of attempting a further attack from the Octopus honeypot, successfully accessing another machine via SSH.	105
Figure 80: Result of attempting a further attack from the AHA honeypot, successfully accessing another machine via SSH.	106

Table of Tables

Table 1: A summary of the three main types of honeypots.	14
Table 2: The three honeypot services used for testing.	33
Table 3: The legitimate and honeypot RDP server that will be used for research.	33
Table 4: The two different types of NMAP scans and their respective flags.	44
Table 5: A table noting if differences were seen in the scans against the pairings of honeypot and legitimate system.	49
Table 6: The number of network packets generated by each NMAP scan against each service.	49
Table 7: Time to complete both NMAP scans against each service, in seconds.	50
Table 8: Results of the normal Xprobe2 scan along with the real OS and probability.	51
Table 9: Time to complete in seconds for each Xprobe2 scan against each service.	51
Table 10: Results of the two different behaviour tests (F = fail, P= pass).	59
Table 11: The results of ICMP latency testing against each service in milliseconds.	95
Table 12: The results of TCP latency testing against each service in milliseconds.	98

1. Introduction

1.1 Background and Context

The Remote Desktop Protocol (RDP) is a fundamental aspect of remote administration of computer systems around the globe and allows IT professionals to access and administrate systems from anywhere with an internet connection. It is, therefore, of great interest to cybercriminals due to the level of access it can provide. Recently, there has been an increased interest in RDP software due to the COVID-19 pandemic forcing many to work from home. TrustRadius, a software review platform, saw an increase of 1,587% in average traffic to their remote desktop category pages, as seen in Figure 1, (Sullivan-Hasson, 2020). This trend is backed up by Cybersecurity and Infrastructure Security Agency who have reported a 127% increase in the number of exposed RDP services since the beginning of the pandemic, (CISA, 2020).

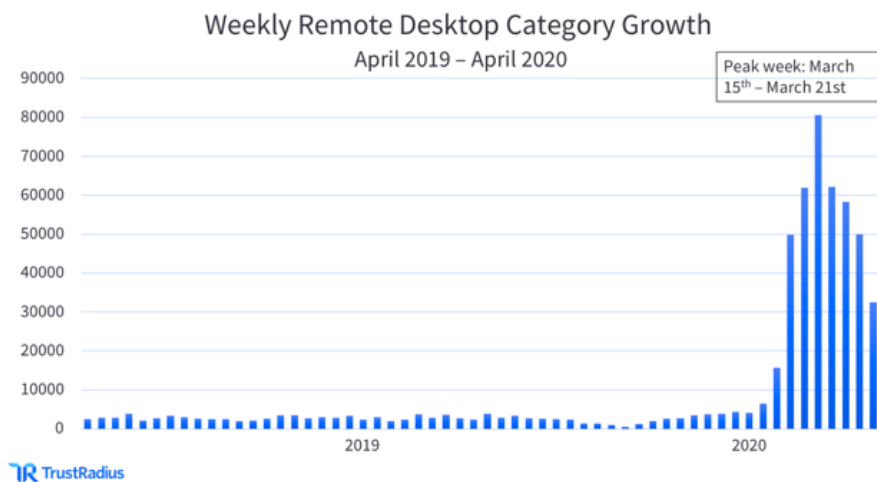


Figure 1: Weekly average traffic to the Remote Desktop category on Trust Radius, a software review service.

However, RDP has seen a reasonable amount of security vulnerabilities, including the infamous remote code execution vulnerability, BlueKeep (CVE-2019-0708 (NVD, 2021)), that was used to spread malware across the globe, (Greenberg, 2019). RDP is also a regular victim of password brute force attacks. A report by Sophos, in which 10 RDP honeypots were deployed, found that an internet-facing RDP service would receive an average of 600 brute force attempts per hour, compared to 2 per hour from a similar study conducted in 2012 (Boddy, et al., 2019).

With the rise of RDP as a target for cybercriminals there is an inevitable rise in the need for research and monitoring. Honeypots fill this role well as they can be used both to learn about current attack trends and techniques, as well as alert an organisation of potential attacks. Honeypots are applications that mimic an application or network protocol, in this case an RDP service. They are most popularly seen mimicking basic protocols such as SSH, TELNET and HTTP. These honeypots are often highly interactive, allowing deep knowledge to be gathered about attackers by faithfully imitating real systems. By convincing an attacker that they are a real system, a honeypot can gather information in much more depth than a firewall or intrusion detection system. Honeypots can monitor what an attacker intends to do once inside a system, including files they try to access or next steps they try to take, allowing an analyst to distinguish between the likes of a fluke success or a targeted attack, (Rapid7, 2022).

Being able to distinguish between a real system and a honeypot is essential, otherwise the honeypot is not fit for purpose. A honeypot serves no real value if it can be detected as one.

Knowing how to effectively fingerprint honeypots can improve both the honeypot itself and the work of those participating in ethical offensive engagements, for instance, purple teams. Purple teams are a combination of red and blue teams, as illustrated in Figure 2. They conduct penetration testing against a system or network and then, using the knowledge gained from said test, attempt to increase the security of the system, (Morrison & van den Brekel, 2020).

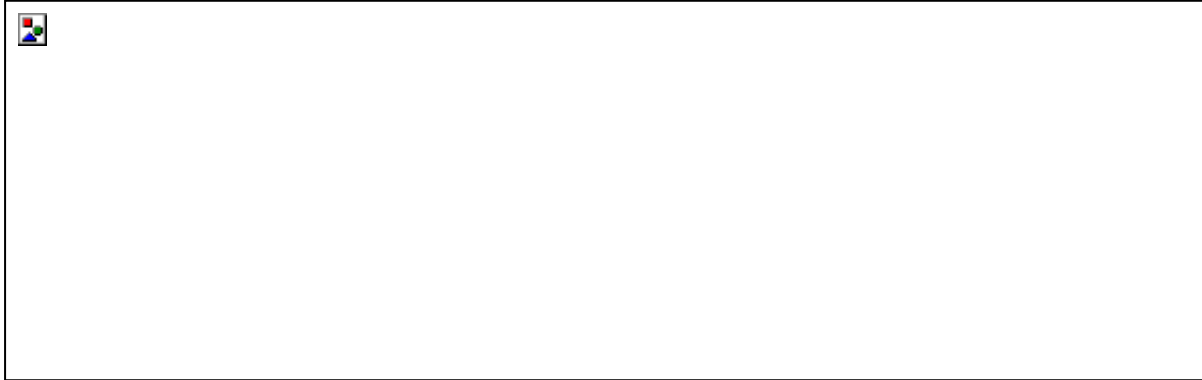


Figure 2: An explanation of the key roles in Red, Purple, and Blue teaming.

The advantage of combining the two teams is more efficient communication between a traditional red and blue team engagement, and a combination of efforts and knowledge from both sides, (Miessler, 2022). When playing the role of aggressor, these teams need to be able to distinguish between real systems and honeypots. Failure to do so is a waste of both time and money, therefore, it is important to explore effective techniques to detect and label honeypots, While some research has been carried out on honeypot fingerprinting, there has been no research into fingerprinting RDP honeypots specifically or evaluating the most effective methodology for fingerprinting honeypots in a long-term, offensive engagement.

1.2 Aim, Research Questions, and Objectives

This project aims to identify and evaluate the most effective detection methods for purple team engagements against contemporary RDP honeypots. This aim can be broken down into three different research questions:

1. What are the existing honeypot detection methods, and how do they compare?
2. Which detection technique is most effective in a purple team engagement?
3. How can these detection methods, honeypots, and the broader network be refined to improve effectiveness?

The aim and research questions will be answered by carrying out the following objectives:

1. A literature review will be carried out that will critically examine prior research into honeypots, remote desktop protocol, existing honeypot detection techniques, and honeypot detection countermeasures.

2. Using the knowledge gained from the literature review, a methodology will be designed to test various techniques against existing RDP honeypots alongside legitimate RDP clients as a means of finding notable differences.
3. This methodology will then be executed, and the results recorded.
4. Finally, the results will be analysed and discussed based on the effectiveness to distinguish real and fake RDP systems on the fly. Countermeasures to these techniques will also be discussed, which will highlight improvements that could be made by honeypot developers.

1.3 Scope

The majority of research into honeypots and fingerprinting revolves around the most popular honeypots for the most popular protocols, notable SSH, TELNET and HTTP. There is little to no published research into RDP honeypot detection. This, combined with the increased relevancy of RDP, means that this paper will focus entirely on RDP and the few dedicated RDP honeypots that exist. This will produce results that are unique to the challenges and characteristics posed by RDP. The latest versions of these honeypots will be used along with the latest RDP clients available.

1.4 Structure

Following the introduction section that outlines the background, research questions, objective, and scope of this project, the rest of the paper will be structured as follows:

- Literature Review
 - This section will present and critically analyse preceding research into honeypot detection and fingerprinting methods, as well as honeypots and RDP in general, to set a base level understanding of these two concepts.
- Methodology
 - This section will discuss the research that forms the groundwork for the methodology. A design for the methodology, along with an explanation of how the said design was implemented will also be outlined. This design will then be executed under the 'Experimentation' subheading, with each experiment being separated. Finally, the methodology will be evaluated, and a conclusion will present the key highlights.
- Results
 - This section will present the findings produced by experimentation and the analysis done on them, as well as highlighting the most notable results.
- Discussion
 - This section will discuss and evaluate the entire project, focusing mainly on the methodology and conclusion, and comparing it to other academic research. It will also touch on how the findings can be applied to both the offensive and

defensive sides of purple teaming. This section will also discuss how this particular research could be furthered.

- Conclusion
 - This final section will summarise the entire project, including any key findings and evaluating the strengths and weaknesses of the project.

2. Literature Review

2.1 Introduction

This section will critically examine the current research and literature that has been published around the relevant areas of this report. It has been separated into four main sections:

1. An overview of honeypots and their most recent developments
2. Remote Desktop Protocol
3. Existing Honeypot Detection Techniques
4. Honeypot Detection Countermeasures

There is an abundance of literature published around all four of these topics, but the most focus will be given to existing techniques that have been researched. There are many different techniques that have been discussed but no research has focused on techniques specifically against the Remote Desktop Protocol, therefore, all the research examined will be focused on more popular honeypot protocols like SSH and HTTP. However, the section will conclude by extracting all the detection techniques that could theoretically be used to effectively fingerprint RDP honeypots.

2.2 Overview of Honeypots and Recent Developments

Honeypots are devices placed onto networks to lure cybercriminals into attacking them. The actions of the attacker can then be monitored and researched. According to a 2021 survey, honeypots are mostly classified under two labels: production and research. Production honeypots are most commonly used to detect attackers inside an environment, while research honeypots are often used to examine new exploits or the actions of attackers once they are inside a network, (Zhang & Thing, 2021). They exist for many different technologies but are most seen impersonating SSH, HTTP and TELNET protocols, all with varying levels of interaction. They can be classed into three different levels as shown in Table 1, (Livshitz, 2019).

Table 1: A summary of the three main types of honeypots.

Interaction Level	Characteristics
Low	The most basic form, usually only emulating the service fingerprint and a login ability. They are the easiest to deploy and configure but offer the least amount of information.
Medium	Responds to specific set criteria but do not fully emulate the service, e.g., a vulnerable file path on a web server being served to bait a particular exploit, (Spitzner, 2002).
High	Emulates the service and an entire system to back it. This could include a filesystem, desktop environment or an entire network of honeypots. They are the most time and resource consuming to deploy but offer the most amount of data.

Research shows that honeypots are most commonly low interaction, due to the ease of development, deployment, and maintenance (Nawrocki, et al., 2016). However, due to the

advanced capabilities of RDP, medium- and high-interaction honeypots are significantly more helpful due to the extensive behavioural data they can provide. Since medium- and high-interaction honeypots are the only appropriate type for RDP, few honeypot solutions exist. Honeypots have been used defensively to disrupt the cyber kill chain, as shown in Figure 3, (Lockheed Martin, 2022), according to Zhang et al. in their 2021 survey, (Zhang & Thing, 2021). Firstly, honeypots can disrupt the reconnaissance phase by confusing any scanning activity. This use for honeypots can be seen in research by Tom Liston who developed sticky honeypots that take over unused IPs in a network to attract worms and attackers, (Liston, 2001). This research was bettered by Leslie Shing in 2016 who improved the connection parameters in an attempt to make the traffic seem more realistic. Referring to them as tarpits throughout her research, she demonstrated a better sticky IP honeypot than Liston's which, while less 'sticky', increased the honeypots capabilities and modularity, (Shing, 2016).

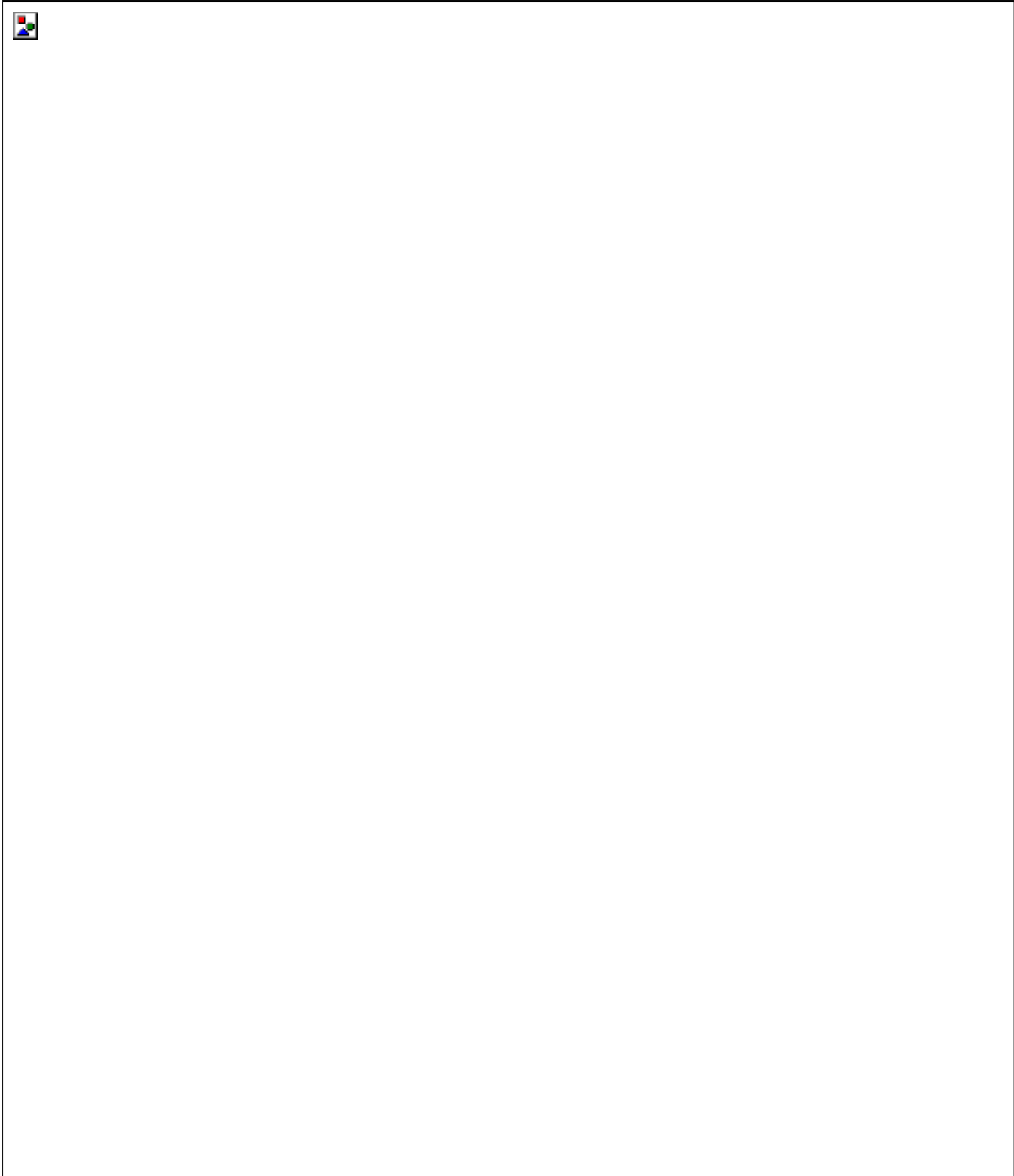


Figure 3: The Lockheed Martin Cyber Kill Chain, (Lockheed Martin, 2022).

Honeypots can also disrupt the attack phase of the kill chain, including exploitation, installation, command and control, and actions on objectives. By tricking an attacker into accessing a high interaction honeypot, all of these steps could be monitored and contained, allowing tactic, technique and procedure information to be obtained.

2.3 Remote Desktop Protocol (RDP)

The Remote Desktop Protocol is a network protocol developed by Microsoft to allow users to connect to their Windows systems over a network using a graphical interface. By default, an RDP server runs on port 3389, (Liang, et al., 2021). RDP offers the whole desktop experience,

which makes it a valuable technology for system administrators and remote workers. It is also very popular, as can be seen via internet wide scanning services like Shodan. As of March 2022, there are over 3.5 million different RDP services that are running facing the internet, with a considerable number of these running on systems that have reached their end of support, like Windows Server 2008 and Windows 7, as can be seen in Figure 4, (Shodan, 2022).



Figure 4: A graph of different operating systems (OS) running RDP facing the internet.

Open RDP services are also constantly scanned for by attackers. Examining internet wide scanning using GreyNoise showed almost 7,000 malicious RDP scans occurring as of the end of March 2022, as can be seen in Figure 5, (GreyNoise, 2022).

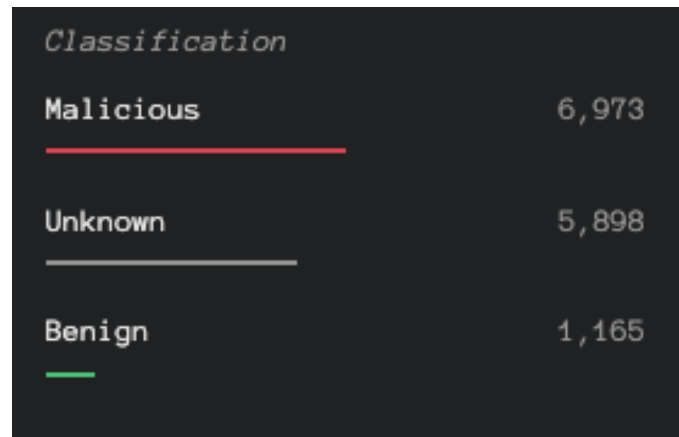


Figure 5: The divide of malicious and benign RDP traffic shown by GreyNoise.

Due to its widespread use and the power it can wield, RDP has been a popular target for hackers over the years. If a vulnerability in RDP can be discovered, it can give an attacker a massive reach into a network. The most infamous vulnerability in recent years was BlueKeep, (NVD, 2021). First disclosed in 2019, the vulnerability allowed attackers to perform remote code execution against a vulnerable system, including Windows 7 and Windows Server 2008, two popular offerings at the time. According to a report from the CISA, the attack involved sending a specially crafted packet and allowed an attacker to add new accounts with full rights, modify

and view data, or install programs. The attack required no interaction from the victim and was even considered wormable, allow a sophisticated attack to take over entire networks using it, (CISA, 2019).

RDP is also commonly a victim of brute force attacks, as outlined in a report by Sophos, (Boddy, et al., 2019). The researchers set up RDP honeypots on Amazon EC2 instances across 10 different geographic locations with accounts that could never be brute forced, allowing them to examine all the different username and password combinations attempted by attackers. They left the machines up for a period of 30 days before examining their results. Over this time, the ten different honeypots logged a total of 4,298,513 failed login attempts, which came out to 600 attempts per hour per honeypot. As a comparison, similar research undertaken in 2012 showed just 2 login attempts per hour. The login attempts got progressively higher as the time went on, as show in Figure 6.

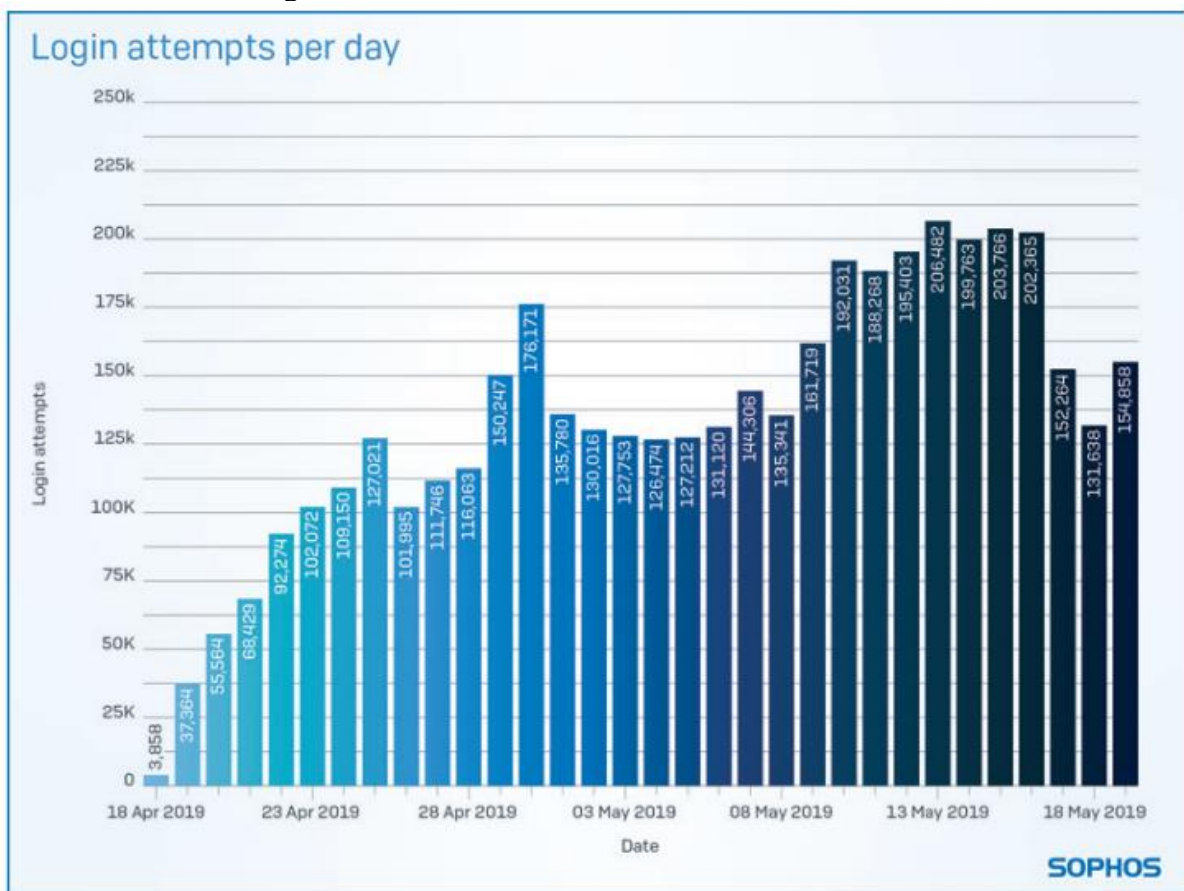


Figure 6: A bar graph showing the login attempts per day over the 30-day period.

This research demonstrated how common brute force attacks were against RDP systems, highlighting the importance of understanding these types of attacks and the need for systems like honeypots in both the detection and research of these things. However, it would be beneficial to repeat this research because, while not old, the rise in work from home over the past few years may create an interesting comparison in results between now and then.

There is little research around RDP honeypots that is relevant to this project. One paper that is relevant is a short report detailing how effectively RDP and VNC honeypots could be used as an intrusion detection system, and how much information can be gained about an attacker by using them, (Danchenko, et al., 2017). This aim of this study was to determine if RDP/VNC honeypots could be used as an IDS in place of a traditionally IDS or firewall solution. The idea

is that honeypots are significantly cheaper to deploy and maintain than these other solutions, therefore, if they can effectively be used to detect threats they could be used as an alternative. In this short paper, no experiments were carried out. Ideas were discussed by the authors and a simple explanation was proposed: a honeypot would be placed on the network and would be the first point of contact for external users. The honeypot would determine if the user was authorised to access the server and if they were, would allow the user to connect. If the honeypot determined the user was not authorised, the user would be connected to a honeypot where their intentions could be further examined. The authors concluded that this type of IDS would help organisations better gather data on their advisories, when compared to traditional security solutions. This data could be used to further develop the organisations existing security architecture. This research proposes an interesting idea that has since been developed into a real-world solution by third parties, however, it fails to dig into the feasibility of this proposal on its own. In its entirety, the research is simply a novel, but potentially effective, new way to detect, monitor, and learn from bad actors, without demonstrating whether or not it is really possible. Being written in 2017, it would have been feasible to demonstrate this type of technology as it is not overly complex.

RDPs level of interaction and complexity poses a challenge for creating high-interaction honeypots to replicate the service. The typical approach to this issue is using virtual machines that run legitimate copies of Windows software with RDP enabled. These machines are then rebooted once attacked, or a different machine from a pool of machines is given out to different attackers.

2.4 Existing Honeypot Detection Techniques

2.4.1 Overview of current research

There has been little research done into honeypot detection methods and no research into RDP honeypots particularly. This is most likely due to how much detection techniques can vary in the service they mimic and their level of interaction, making it hard to compile general methods. However, information can be gathered from some related research.

Fingerprinting comes in many different forms, but the main ones that will be focused on in this section are network, application, and system. Network fingerprinting is the most common technique and can be a reliable way of non-intrusively fingerprinting a system. Application fingerprinting can often be more intrusive than network fingerprinting but yield more verbose results. Finally, system fingerprinting is usually the most complex and intrusive, but when done correctly can provide the most accurate assessment of the nature of a system like a honeypot.

A fingerprinting framework will also be discussed as a mean of methodising different fingerprinting techniques.

Behaviour is another interesting detection vector. Honeypots behave differently from normal systems but try to hide any of these anomalies. Three different types of behaviour differences have been identified and will be discussed in detail as to how they may be used as a detection method.

Since many honeypots are run virtually, being able to identify a system as being virtualised when it is expected to be run on bare metal, for instance a standard enterprise user desktop, can be an accurate way of detecting honeypots. This section will mainly focus on the research surrounding virtual machine detection.

Measuring network latency has been a popular honeypot detection method due to its reliability over the years. This section will review the literature surrounding this technique and how it can be performed with the hopes of replicating it later in this report.

Finally, default configurations have been a reliable way of detecting honeypots since they first became popular. Most honeypots have a static, default configuration that can be turned into a fingerprint and used to scan for similar systems across networks, even being deployed in internet wide scanning.

2.4.1 Fingerprinting

The term ‘fingerprinting’ covers many different techniques which can prove useful when identifying systems or carrying out reconnaissance. A survey published in 2017 by Joni Uitto et al. outlined several anti-honeypot methods including three different fingerprinting categories: network, application, and system, (Uitto, et al., 2017).

2.4.1.1 Network

Mostly applicable for low and medium interaction honeypots, network fingerprinting can provide a consistent way of categorising honeypots. Uitto points to research conducted by Fu et al. which is further discussed in section 2.4.4 as it focuses solely on latency , (Fu, et al., 2006).

Network fingerprinting can also be done using network scanning tools like Nmap. Researchers from the Swiss German University used the tool against four different honeypots (HoneyD, Dionaea, Kippo, and Glastopf) to identify characteristics that could be used to identify the services as honeypots, (Dahbul, et al., 2017). They set up the honeypots alongside the legitimate services they mimicked, and used network scanning tools to grab banners, scan ports, and examine protocols. The researchers found issues with how each honeypot had implemented the services it was trying to mimic, as can be seen in Figure 7. If an attacker identified these issues on the system, it would serve as a red flag towards the nature of the device.

Honeypot	Layer	Focus	Vulnerabilities	Enhancements
	4	Open Ports	Suspicious open ports	Opening & Configuring essential ports
HoneyD	7	HTTP	Invalid replies	Configuring scripts
	7	IIS	Suspicious timestamps	Fixing timestamps
	4	Open Ports	Suspicious open ports	Opening & Configuring essential ports
Dionaea	7	FTP	Detected by port scan	Configuring scripts
	7	MySQL	Detected by port scan	Configuring scripts
	7	SMBD	Detected by port scan	Configuring scripts
Kippo	7	SSH	Commands not working properly	Configuring scripts
Glastopf	7	HTTP	Invalid error reply	Configuring scripts

Figure 7: The vulnerabilities found in each honeypot, along with remediations that could be made.

This research highlighted some accurate ways of detecting honeypots, specifically how weak Dionaea was to detection via network fingerprinting. However, some of the techniques provide less concrete results around simply suspicious characteristics, which on their own, would not be enough to confidently identify a honeypot.

Research by Hamid Mohammadzadeh and others from Victoria University of Wellington used Nmap to perform active network scanning against a dynamic honeypot system they proposed in their research, (Mohammadzadeh, et al., 2013). While not the focus of their research, they outline the types of Nmap scans they performed to test their honeypot system, ranging from quick scans to full comprehensive scans. The scans that were used in their tests can be seen in the below figure.

Nmap	Included Detection Switches	Time to Scan (Seconds)	Amount of Traffic Generated/Received
Intense Scan	Includes: -sV Version detection, SMB OS Discovery script Added: -O (OS Detection)	242	(562.47KB) Rcvd: (474KB)
Intense Scan plus All UDP ports	Includes: -sV Version detection, SMB OS Detection Added: -O (OS Detection)	2504	(847.76KB) Rcvd: (1.073MB)
Intense Scan plus All TCP ports	Includes: -sV Version detection, SMB OS Discovery script Added: -O (OS Detection)	529	(31.969MB) Rcvd: (28.866MB)
Quick Scan	Added: -O (OS Detection)	157	Rcvd: (125.34KB)
Quick Scan Plus	Includes: -O (OS Detection) Added: -sV Version detection, SMB OS Discovery script	267	(256KB) Rcvd: (897KB)
Comprehensive Scan	Added: -O (OS Detection) Includes: -sV Version detection, SMB OS Discovery script	2322	(927KB) Rcvd: (1.121MB)
smb-os-discovery Script Scan, ports 139/445	None	7.2	(13KB) Rcvd: (19KB)

Figure 8: A table of NMAP scans used to scan honeypots, (Mohammadzadeh, et al., 2013).

There scans can be borrowed for later use in the methodology of this research. This paper by Mohammadzadeh et al. while thorough, only uses Nmap and neglects the use of other popular network scanning tools like Xprobe2 or SinFP3 that were used in similar research done into fuzzing by Naik et al., (Naik, et al., 2021). Using more tools would have improved the quality of this papers results.

Network fingerprinting has also been performed more manually by sending TCP packets and examining different characteristics of the network traffic. Research from New Mexico Tech published in 2007 concluded that while a valuable technique, the characteristics of a benign system could change vastly based on legitimate differences like operating system or network topology, making the technique only really viable when used alongside a reliable benchmark, (Mukkamala, et al., 2007). While the nature of this research is interesting, that manual and complex nature makes this technique unviable in a purple team engagement.

More novel detection techniques have also been proposed. Alexander Vetterl presented a fingerprinting technique that involved analysing the implementation of network architecture within off-the-shelf honeypots, (Vetterl & Clayton, 2018). This technique involved examining how honeypots implemented network protocols and comparing them to the services they mimic. Although the differences were slight, they were enough to allow distinctions to be made between the honeypots and authentic services. The technique involved generating many packets and a moderate amount of low-level research. For these reasons, it is not an excellent “on-the-fly” method, however, with enough time and the right conditions, it could be implemented into a bespoke tool to be used in offensive engagements.

2.4.1.2 Application

Uitto also suggests examining the applications and services that a device offers to determine if they are in line with the expected behaviour. If a server is offering service A and B but would be expected to also offer C and D, this could be deemed as suspicious and tip off an attacker causing them to doubt the validity of the system. In the case of RDP, it would be odd to find a server with only port 3389 open and nothing else since RDP is most often used to administrate

servers that offer other services like SMB or HTTP. While this could be an indication of suspicious activity, it is not enough to determine the nature of a system. This could be used alongside another fingerprinting technique to build up a case against the system, however, on its own would unlikely ever be reliable, especially in a real-world engagement where systems are often configured imperfectly.

2.4.1.3 System

Finally, system fingerprinting is focused on high interaction honeypots. To avoid rewriting entire systems, most high interaction honeypots are the real system, usually deployed virtually. Since these honeypots use the original system, they are identical in that way, however, still require monitoring or logging software to be installed directly onto them or hooked in. Both methods leave traces that an experienced attacker may be able to find. Research has been done more deeply into this technique, specifically at how a data capture tool, Sebek, could be detected on a honeynet, (Holz & Raynal, 2005). The authors outlined how Sebek could be detected on both Linux, Windows, and OpenBSD. Detecting it on Linux was as simple as running a tool known as 'module_hunter.c' which looked for patterns in the kernel's address space. Similarly, on Windows it could be detected by searching the 'PSLoadedModuleList' and looking for the hidden module using the 'KProcCheck' tool, as shown in the figure below.

```
C:\>kproccheck -d KProcCheck Version 0.1 Proof-of-Concept by SIG^2 (www.security.org.sg)
80400000 - \WINNT\System32\ntoskrnl.exe 80062000 - \WINNT\System32\hal.dll F7410000 - \WI
NNT\System32\B00TVID.DLL [...] F7298000 - SEBEK.sys [...]
```

Figure 9: The command used to run KProcCheck and detect Sebek on Windows, (Holz & Raynal, 2005).

Their research shows it can be trivial for an attacker of a certain level to detect a honeypot based on the monitoring tools that have been injected into the system, however, with the research being published in 2005, it is unclear how relevant a practice this is today. It also relies on the attacker already having access to the system. While this task could be expected to be trivial given the nature of honeypots, an attacker may have already had to waste time scanning and accessing the system before they can determine its nature, not only wasting precious time but also giving away their presence on a network.

2.4.1.4 Fingerprinting Framework

A paper by Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis laid out an in-depth framework for fingerprinting any type of honeypot, (Srinivasa, et al., 2021). This paper presented a framework for fingerprinting honeypots by compiling research that has been taken place in the field over the last few years. This was a first-of-its-kind framework that could be used as a basis for any further honeypot research against any type of honeypots. The framework was composed in two parts, a probed-based fingerprinting method paired alongside a metascan-based method. The first consisted of a port scan, banner check, HTTP status response analysis, SSL/TLS certificate check, protocol handshake analysis, library dependency check and static command response. The second consisted of a Cenesys or Shodan analysis, a keyword search, ISP/AS check and a cloud hosting check. Both methods finished with a FQDN check. All this data was used to determine if a device was a honeypot. The framework was designed with the 9 most popular honeypots, in their default configuration, as the target. The framework was illustrated by the authors, as can be seen in Figure 10.

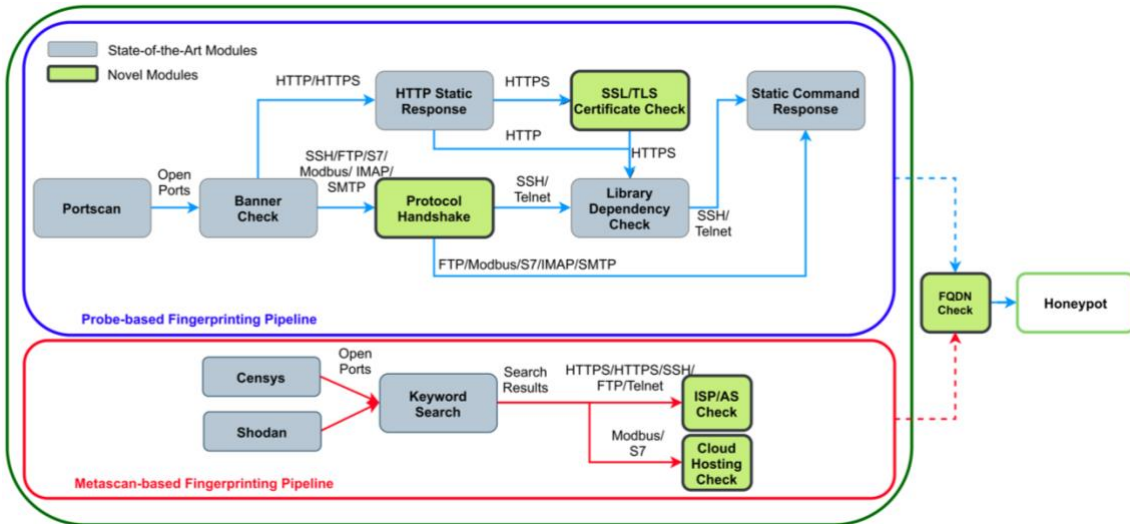


Figure 10: The proposed framework laid out visually, including both branches.

Data was collected by scanning the entire internet using ZMap, (Adrian, et al., 2014), totalling over 2.9 billion devices. This data was fed through the framework created by the authors to determine if the device was a honeypot. This was repeated three times. The authors identified 21,855 devices as honeypots. 14,246 of these were identified by the probed-based scanning and the other 7,410 were identified by the meta-based scanning. Not all these honeypots were running their default configuration. Due to the nature of honeypots, it is difficult to know how accurate the results of the study are but due to the extensive nature of the framework, the likelihood of false positives is low. The figure that cannot be provided is how many honeypots were missed by this framework. This research was quite extensive, looking at multiple honeypots and proposing a thorough framework which successfully identified a large number of honeypots, not all of which were running their default configuration. The test was also repeated three times, with consistent results. The research could be taken further by developing the framework into an automated tool or developing the framework further to target more honeypots. This framework cannot be fully used in a research environment due to the lack of internet deployment, however, what can be used from it will form a basis for a methodology later in this research.

2.4.2 Behaviour

From an attacker's perspective, using a honeypot system can seem different to using a real system based on how accurately the system replicates its counterpart. Due to this, the way a honeypot behaves can give away its nature. A paper by Srinivas Mukkamala outlined some key characteristics that honeypots possess that are not seen in regular systems, (Mukkamala, et al., 2007). These included:

1. A lack of network activity on the honeypot
2. All interactions being logged
3. Restricted bandwidth to prevent further attacks from said honeypot

The author concluded the first attribute is impossible to use reliably due to the need for long term monitoring and the high possibility of false positives. The other two characteristics, however, are exclusive to honeypot systems and would not be found in production environments.

The following subsections will focus on three different characteristics found within honeypots and not real systems that may be used to identify them as such.

2.4.2.1 Automatic Redeployment

Automatic redeployment is a technique used by honeypots to reconfigure their system to a default state after being attacked, allowing for another attacker to be trapped by it. This technique was discussed in a paper published in 2018 which outlined a number of different techniques, (Tsikerdekis, et al., 2018). Automatic Redeployment can be an obvious tell. For example, if an attacker connects to a honeypot, creates a file, and then disconnects, they expect to find the file on the system if they ever reconnect. If the honeypot rebuilds itself immediately after the attacker disconnects, the attacker could use the removal of their file as proof the machine has been tampered with by reconnecting within a short time frame.

2.4.2.2 Dynamic Intelligence

Another honeypot characteristics noted by Tsikerdekis was Dynamic intelligence. Dynamic intelligence is a method honeypots use to attempt to be more helpful. It involves using AI to adapt to the user and lure them in by doing things such as changing file names to different languages to determine ethnic background. While this technique is intended to make honeypots harder to detect, an attacker with a keen eye could notice the change in file names. RDP would be a sensible service to implement dynamic intelligence due to its high level of interaction, therefore, this detection technique is also likely applicable to either existing or future RDP honeypots.

2.4.2.3 Furthering an attack (Operation Analysis)

Honeypots are designed to be attacked, therefore, it is expected that a malicious actor will have access to the system in some form at one point in time. For this reason, steps must be taken to ensure the honeypot is not used to launch further attacks. Honeypots should never provide lateral movement into non-honey parts of a network, and they should never become part of botnet. This characteristic proves a downfall in honeypots hiding their nature. Uitto et al. raised ‘operational analysis’ as a method of detecting a honeypot in their survey, (Uitto, et al., 2017). Uitto proposes an attacker could simply attempt to add the suspected honeypot to a botnet. If it is unsuccessful for no clear reason, it is likely to be a honeypot.

2.4.3 Virtualisation

It is common for honeypots to be run as virtual machines to isolate them from production systems but more importantly allow for simple deployment and redeployment after an attack. However, the popularity of virtualisation can also be the downfall of a honeypot because if a system can be detected as virtual it can often be a giveaway to its true nature. A paper by Michail Tsikerdekis examined various improvements that could be made, both practical and theoretical, to honeypots behaviour to decrease their chance of being detected, (Tsikerdekis, et al., 2018). The third technique, hardware, would examine if the service is running on bare metal or a virtual machine (VM). This information is helpful; however, it cannot be used on its own to determine if a machine is a honeypot or not. While it is common for honeypots to be run inside VMs, due to the flexibility and ease they offer, it is also common for legitimate infrastructure to be run in VMs for the same reason.

A paper by Thorsten Holz and Frederic Raynal published in 2005 also touches on virtualisation as a means of detecting honeypots, specifically looking at VMWare, (Holz & Raynal, 2005). VMWare is software that allows for virtual machines to be deployed and managed and is a common way to deploy honeypots even to this day. Holz outlined a few different red flags including the MAC address of the network interface and a backdoor within the I/O

functionality, however, these two identifiers are no longer relevant to modern builds of VMWare.

2.4.3.1 User-Mode Linux

Another technique examined and Holz and Raynal was User-mode Linux (UML). While not strictly virtualisation, it possesses many of the same shortcomings of virtualised honeypots and has therefore been grouped accordingly. UML is a version of the Linux kernel that runs in user space and talks to the original kernel, referred to as the host kernel, like a proxy, (User Mode Linux Core Team, 2020). While its intention was as a debugging tool for programming Linux kernels without crashing the entire system, it has seen some limited use as a honeypot. Holz outlines a few techniques of detecting this, mainly revolving around examining processes and ‘dmesg’ logs, as well as the ‘/proc/cpuinfo’ file that will outright identify the system as UML. Using UML as a honeypot type system is essentially a forgotten technique due to the popularisation and easy-of-use that virtual machine software has provided. Therefore, this research is not applicable to most modern research.

2.4.3.2 Detecting Virtualisation

Virtualisation has become a popular method of deploying infrastructure, therefore, being able to tell a system is virtualised is not an immediate red flag that the system is a honeypot. However, it can be a giveaway if the system is not normalised virtualised, like a desktop environment. While Desktop as a Service has grown in popularity in recent years, it still is not the norm, (Gartner, 2020). Therefore, being able to detect if a system is being run virtually can be a helpful sign of its nature. Research by Anoirel Issa outlined methods of detecting a wide range of virtual machines, including VMWare and virtual machines running certain Windows operating systems like XP, (Issa, 2012). Unfortunately, the author did not address QEMU/KVM devices, which are used by two of the honeypots examined in this research. However, their findings are still of interest. The authors illustrates that the stack address on a Windows XP virtual machine can be predicted to prove a machine is virtualised. The other shows that the ECX and ESP is usually around 0x120000 on a virtualised XP environment, and around 0x18000 for Windows 7. The author also demonstrated how DLLs could be used as a red flag as well. In a debugging menu, instead of the system pointing towards ‘ntdll.dll’ when called, it instead points towards ‘debug’ on a Windows XP virtual machine run on Virtual Box, as can be seen in Figure 11.

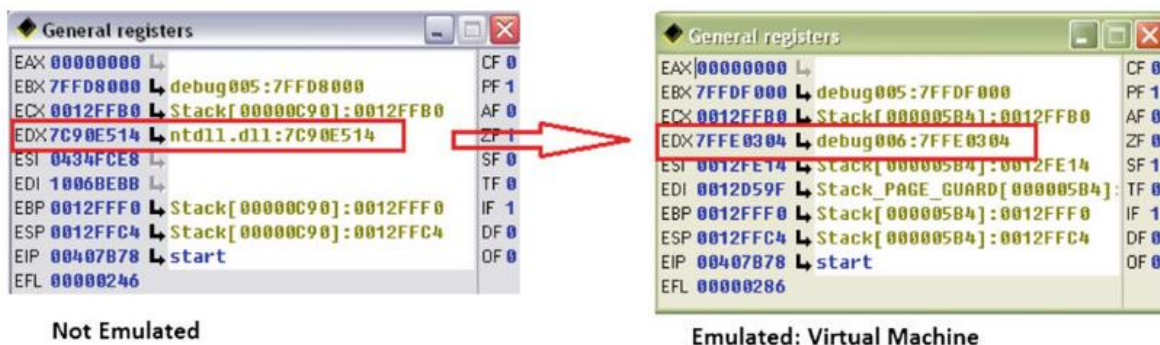


Figure 11: The difference in the stack between a non-emulated and emulated Windows XP, (Issa, 2012).

Issa also outlined techniques for detecting specific virtual machines, however, none of these virtual machines are used within the methodology of this research, therefore, this information is not within scope.

2.4.4 Latency

As mentioned in section 2.4.3, it is common for honeypots to be run as virtual machines, allowing them to be easily backed up and deployed and redeployed after an attack. Honeypots often also have built in logging mechanisms to record information about their attacker. Both additional factors add extra latency to network traffic to and from the honeypots when compared to the systems they are trying to impersonate. This issue was highlighted early on by Holz and Raynal, being one of the techniques they wrote on that is still relevant to today (Holz & Raynal, 2005). When honeypots log information or are hosted within virtual machines, extra processing time is required when compared to an ordinary system. While unlikely to be noticed by an individual, networking tools can be used to time round trip times and response times from systems, however, while promising on paper, specific circumstances are required. A baseline first must be taken against a similar system on the same network that is confirmed to be legitimate, preferably the same distance away. This can then be used to compare to other systems. A paper by Tsikerdekis in 2018 also reports on latency within honeypots, both with the network and the functionality of the system. Honeypots often take slightly longer to finalise a connection due to extra information being logged by the server that the service would not ordinarily log. This method is general enough that it could be applied to honeypots of any type, including RDP.

Research conducted in 2006 delved further into this idea, (Fu, et al., 2006). This paper looked at how attackers can detect honeypots using discrepancies in network latency, and the countermeasures that can be put in place to camouflage honeypots, in this case Honeyd. Honeypots, and more particularly honeynets, often need to emulate the time it takes for certain network events to occur. Depending on the underlying architecture of the system, this is limited by the operating systems timer interrupts, which are also dependent on the timing hardware on the system. This gets even more convoluted when using virtual machines which emulate said hardware, relying on the hardware they are being deployed on. To experiment on this, the researchers deployed a virtual network consisting of three machines connected to a router. This router was connected to a second router, and that router to a third, as can be seen in Figure 12.

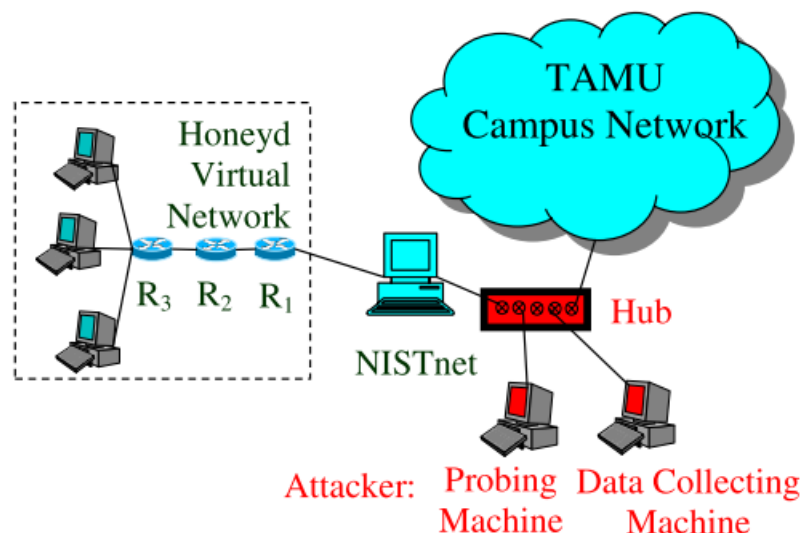


Figure 12: The virtual network used to test round trip times against the honeynet.

The research's used round-trip time (RTT) as a measurement to look for anomalies. To get the RTT they sent one packet to R2 and another to R3. They then used the following formulate to obtain the two-way link latency:

$$\begin{aligned} \text{two-way link latency} &= 2 \times LLR_{2,R3} \\ &= RTT_{pkt2} - RTT_{pkt1} \end{aligned}$$

They measured using three different techniques:

1. Ping Based: This was the most straight forward, using ICMP echo request messages and receiving an ICMP echo reply.
2. TCP Based: This was done in two ways. The first, if the victim had a well-known TCP service, like Telnet, running, the attacker could exploit said service to get an RTT by starting a three-way handshake and using the ACK/SYN response after sending an initial SYN packet. The second method relied on using TCP reset (RST) packets by sending malformed TCP requests and receiving RST packets in response.
3. UDP Based: Similar to TCP with both known services and malformed requests. By sending a UDP request with an unused destination port, the system would reply with an ICMP packet.

Once the data has been gathered, the researchers outlined how a decision could be made around the systems nature. This first involved a phase of offline training in which RTT data was collected about known real and virtual networks, keeping the data in two categories of real and virtual. This data could then be used within the Neyman-Pearson decision theory. The researchers took the detection rate (PD) and the false-alarm rate (PF) and constructed a classifier where $f_0(x)$ was the link latency of a real network, and $f_1(x)$ was the link latency of an emulated link by Honeyd. This has been visualised in Figure 13.

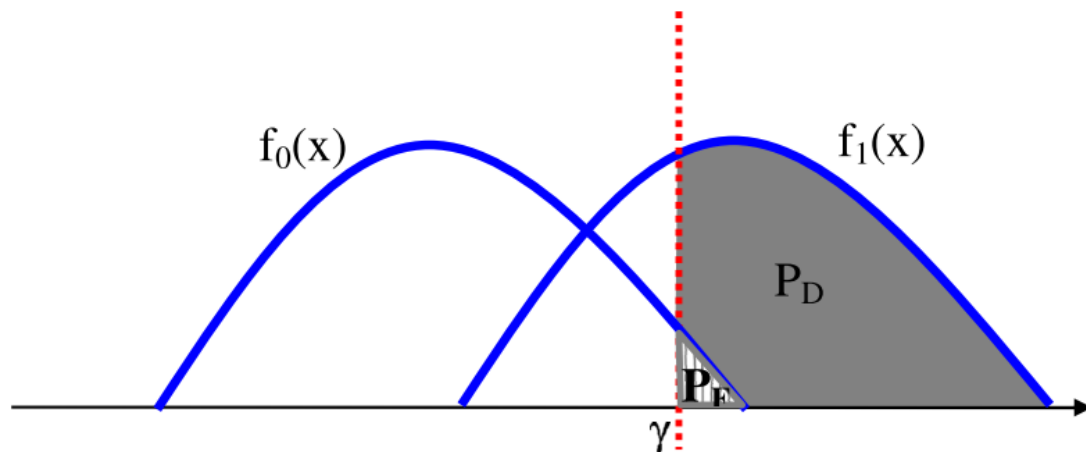


Figure 13: A visualisation of the classifier used by the research to determine if a link was real or virtual.

If the value (x) fell before Y then the link was categorised as real, if it was after, it was virtual. By conducting their tests using the model they had created, the researchers found it to be a reliable way to detect honeypots, due to the fact that Honeyd could only have its emulated latency tuned to 1ms or 20ms and could not accurately disguise itself as a real network. To fix this, the Linux kernel the honeypots were deployed on had to be tuned to increase timing accuracy to better disguise the fake system.

While this research was published in 2006, it remains relevant in 2022. It is still possible to determine RTT in modern systems and can be a quiet operation to perform.

2.4.5 Default Configurations

When a honeypot is deployed, it is often deployed in its default configuration, unless the deployer goes out of their way to make changes before deployment. To further investigate this, research was undertaken by Shun Morishita, Takuya Hoizumi, Wataru Ueno, and others, in 2019, (Morishita, et al., 2019). This research investigated different open-source honeypots running close to their default configuration and how challenging it was to detect them. It was undertaken to demonstrate how trivial it can be to evaluate a honeypot based off its configuration. The researchers used 14 different open-source honeypots, mainly focusing on SSH, TELNET, and HTTP. 20 different signatures were then crafted that could profile the honeypot based off behaviour or its configuration. The researchers then performed an internet wide scan and used their signatures to try and detect honeypots. The researchers found over 19,000 honeypots using this technique, mainly in research networks, with a high concentration being found in Taiwan. One of this paper's greatest strengths was the number of honeypots that were used. This gave the research a wider scope than many of its peers making the data it provides more useful. This project did a thorough job of showing how trivial it is to detect default configured honeypots and how effective this technique can be. While it is out with the scope of this paper to perform a similar experiment, this is a potential area of future work.

2.5 Honeypot detection countermeasures

Honeypot detection countermeasures are a popular area of research. They consist of different techniques or approaches to make honeypots more camouflaged and harder to distinguish from real systems. A popular approach in recent years has been making use of fuzzing techniques to make honeypots harder to detect.

Research published in 2018 proposed a new fuzzy technique that could be used by honeypots to identify and prevent fingerprinting attackers being carried out against it, (Naik, et al., 2018). The research was undertaken to examine how honeypots could be improved to make them harder to fingerprint, in particularly against popular finger printing techniques which are very effective against low level honeypots like KFSensor which was used in the study. Finger printing attacks were executed against a KFSensor honeypot using NMAP to simulate the attack with 5 different fingerprinting scripts, shown in Figure 14, and this data was analysed to identify various finger printing indicators.

No.	Nmap Fingerprinting Attack Script	Fingerprinting Attack Probability for 10 Run		
		Low	Medium	High
1	<code>nmap -O 192.168.0.175</code>	0	3	7
2	<code>nmap -A 192.168.0.175</code>	0	0	10
3	<code>nmap -O -fuzzy - -osscan -guess 192.168.0.175</code>	0	3	7
4	<code>nmap -sV --version - intensity 9 192.168.0.175</code>	3	7	0
5	<code>nmap -O --max -OS - tries 5 192.168.0.175</code>	0	3	7

Figure 14: Different NMAP commands used to test the fuzzing technique.

Once developed, the technique was implemented, and the same finger printing attacks were run. Each finger printing technique was run against the new fuzzing technique 10 times. The data from the scans was collected using Wireshark to examine the conversation between the honeypot and the simulated attacker. This data was then used to develop a fuzzy technique to detect when these finger printing attackers were being carried out. Data about the success of the finger printing technique was gathered by the researchers using the tool they had developed. The technique was successfully developed and used to detect finger printing attacks. A figure showing the main steps of the procedure can be seen below.

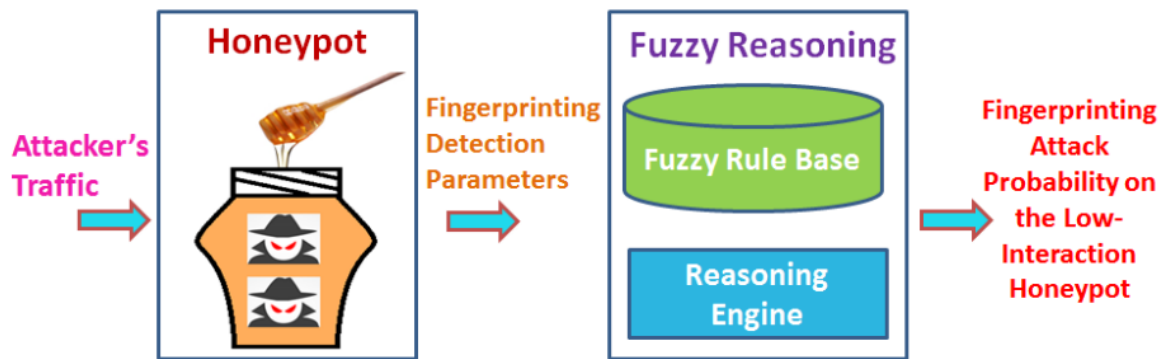


Figure 15: The new honeycomb detection counter measure developed by the researchers.

Many of the attacks were detected as being highly likely to be an attack, with only one attack being as detected as low chance 30% of the time. The full results are visible in Figure 14. This paper thoroughly examined multiple common finger printing techniques and proposed a new fuzzy technique that was successful at detecting these attacks. A large sample size was used to evaluate their solution. The methodology was well thought through, using multiple types of common finger printing attacks and accounting for all of them. This line of research could be further researched by examining a similar technique for medium- and high-level honeypots. Three years later, Naik followed up this research with a similar paper which used more complex dynamic fuzzy rule interpolation (D-FRI) instead of standard fuzzing to detect active fingerprinting attempts carried out by an attacker intending to identify a honeypot, (Naik, et

al., 2021). This study was undertaken to improve honeypots detection avoidance systems in an effect to combat the common honeypot detection method of active fingerprinting. Since honeypots are only effective when they are convincing, it is important that every effect is taken to make these devices look as genuine as possible. Testing was done against the researcher’s honeypot using the same techniques used by attackers: fingerprinting and OS fingerprinting attacks using Nmap, Xprobe2, NetScanTools Pro, SinFP3 and Nessus. The data was collected by deploying the D-FRI honeypot and running fingerprinting attacks against it. Each finger printing tool was run 50 times against the honeypot. The research concluded the D-FRI honeypot was effective at detecting active finger printing attempts, both TCP/IP based and OS based. It was found that the prediction rate of an attack was almost 80% with D-FRI enabled. The use of multiple fingerprinting tools, and types of finger printing attacks, makes this research very rigorous in terms of testing, alongside the repetitive testing of each tool that showed consistent results. The proposed method could continue to develop into quite an effective anti-detection technique if it were fleshed out to work with more protocol or more niche scanning tools. Compared to his earlier research, this new technique proposed by Naik appeared by to 10% more effective at detection, even over a wider spread of tools, as can be seen in the below figure.

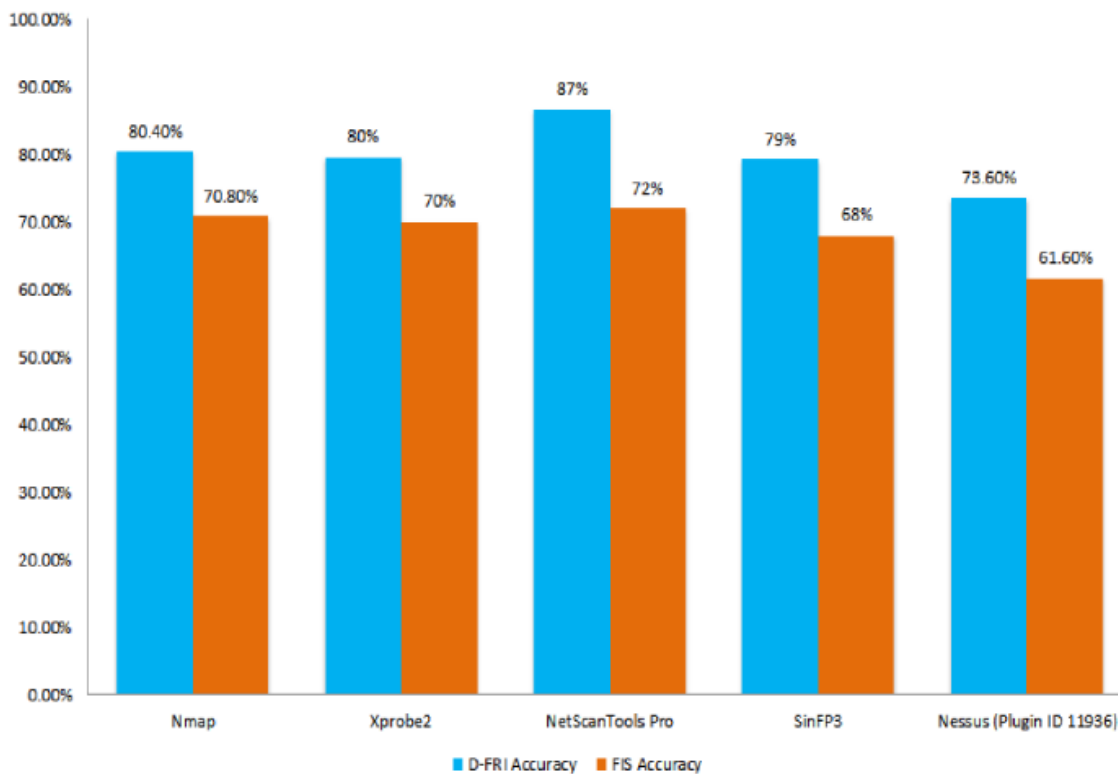


Figure 16: The accuracy of the D-FRI tool against different scanning tools.

2.6 Conclusion

This section has aimed to critically analyse the existing research around honeypots, RDP, honeypot detection methods, and honeypot detection countermeasures. This section has highlighted the strengths and weaknesses of the research alongside the most applicable sections for this current research. This remainder of this section will highlight the parts of each area of research which will be used as part of the rest of this report.

Examining RDP showed just how popular it is across the globe, which makes it such a popular attack vector. Critical vulnerabilities RDP has faced were also mentioned, namely BlueKeep,

which highlights how important RDP honeypots can be in terms of detection and research, thus why being able to effectively fingerprint them is an important research area.

Comparing research around detection methods highlighted an array of different techniques all with their own strengths and weaknesses. Many techniques overlapped with one another but still had their own merit.

The research showed how effective network fingerprinting could be against some honeypots, (Dahbul, et al., 2017), but was not always a sure-fire way of detection. This type of network scanning will be used later in this paper as a starting point, using multiple tools rather than only NMAP. Types of effective scans were also highlighted, (Mohammadzadeh, et al., 2013), but again, only for Nmap, so time will need to be spent transferring these types of scans into other tools. The technique proposed by Vetterl, while effective, would not be usable in a purple team engagement and will therefore not be implemented in this research, (Vetterl & Clayton, 2018). Application fingerprinting proved to be an easy practice to perform but lacks in-depth research and can be rarely used on its own, (Uitto, et al., 2017). However, it will still be considered later in this report. Finally, system fingerprinting, while effective, requires a large amount of overhead and accessing the system in the first place which makes it significantly less stealthy.

Behaviourally analysis proved to be an interesting research area, with key characteristic differences being highlighted including logging and bandwidth restrictions, (Mukkamala, et al., 2007), as well automatic redeployment and dynamic intelligence, (Tsikerdekis, et al., 2018), and finally operational differences, (Uitto, et al., 2017). The nature of these techniques makes them much more complex to automate but will be attempted to be performed manually against the RDP honeypots.

Anti-virtualisation techniques, while quite complex, proved to also be an interesting area of research. Due to the low-level nature of them and the complexity they pose they will not be further examined in this report. This is also due to the fact no research was found on detecting KVM/QEMU environments which will be used to run two out of three of the RDP honeypots. Latency was also a real researched around for detecting honeypots. The techniques highlighted by both Holz (Holz & Raynal, 2005) and Fu (Fu, et al., 2006) relied on first obtaining a benchmark of latency within the environment. This will be possible in this further research but will likely rank low in terms of effectiveness due to the high amount of overhead required.

The final detection technique, default configurations, was one of the most effective in terms of detecting systems. It was used effectively to detect over 19,000 honeypots across the internet, (Morishita, et al., 2019). It cannot be used in this further research, however, the default configuration of the systems will be examined and noted and could be used in further research. Research around counter measures primary focused on the use of fuzzing to trick adversaries. Both pieces posed effective techniques for detecting incoming fingerprinting attacks, with the later research being slightly more effective, (Naik, et al., 2018) (Naik, et al., 2021).

3. Methodology

3.1 Introduction

This section of the report will outline the research, design, and implementation of the practical element of this research. This consists of both the experiments that will be carried out and the network that will be used to perform these tests. The design of the experiments has been based off the research outline in section two. This section will also outline how the experiments are run and evaluate the methodology before concluding this section. The results from the experiments will not be analysed until section 4.

A risk assessment and ethics report were carried out before any testing was undertaken. These can be found in Appendix A - Risk Analysis and Appendix B - Ethics Report respectively.

3.2 Research

The first chapter of the project will contain a literature review that will critically analyse the small pool of existing research into honeypot detection methods. Since this current area of research is lacking, each honeypot will have its behaviour examined to find flaws or inconsistencies within its design. These would then be used to develop more detection techniques or could be used as markers to denote likelihood.

The second chapter of this project was dedicated to a review of the existing literature around the topics of this paper, including detection methods. From this literature, a handful of detection methods have been chosen as part of the testing this paper will conduct. These techniques are:

1. Fingerprinting
 - a. NMAP
 - b. Xprobe2
2. Latency
 - a. ICMP Packers
 - b. TCP Packets
3. Behaviour
 - a. Object permanence
 - b. Further Attacks

Aside from these techniques, each honeypot was also be examined for logical and behavioural anomalies that could be used to denote the systems purpose.

3.3 Design

3.3.1 Honeypots

For the purposes of testing fingerprinting techniques, three viable RDP honeypots were chosen; ad-honeypot-autodeploy (AHA), Octopus, and rdppot, (tothi, 2022) (qeeqbox, 2021) (Kryptos Login, 2019). Each honeypot is free and opensource software and have all been maintained within the last three years. These honeypots were chosen for several factors. Firstly, each works slightly differently and uses a different operating system as its basis. AHA is the most complex and deploys not only a Windows 10 PC, but also a domain controller and an Ubuntu machine

to monitor intrusions in three different QEMU/KVM virtual machines. Octopus simulates an XRDP Linux service and is the most basic and straightforward of the three. It does not make use of virtual machines and therefore simulates the service the closest to bare metal. Finally, rdppot deploys a given number of QEMU/KVM virtual machines in a pool running Windows XP. It swaps out and deploys the virtual machines as requested by incoming connections. These three systems are summarised in Table 2.

Table 2: The three honeypot services used for testing.

Honeypot	Operating System	Notable Features
ad-honeypot-autodeploy	Windows 10	Deploys an automatically populated domain controller
octopus	Ubuntu 20.04	Does not use a virtual machine to deploy the honeypot
rdppot	Windows XP Pro SP3	Runs a pool of virtual machines and swaps them on the fly to support multiple connections

3.3.2 Legitimate Services

To use as a benchmark, three legitimate RDP services will also be deployed, one to match each honeypots underlying architecture. The first was a Windows 10 system, running the same version of Windows 10 as AHA, (Kandakatla, 2022). The only modification made to the system was to enable RDP connections. An Ubuntu 20.04 system was the second legitimate system, running XRDP as the RDP service, (Ubuntu, 2022) (Sorg, 2022). Finally, a Windows XP Pro SP3 system will also be used running the same copy of the OS as rdppot. Each system can be used as a direct comparison to its corresponding honeypot to allow more accurate finding to be drawn about each individual honeypot, as shown in the below table.

Table 3: The legitimate and honeypot RDP server that will be used for research.

Legitimate Service	Honeypot
Windows 10 RDP	ad-honeypot-autodeploy
XRDP	octopus
Windows XP RDP	rdppot

3.3.3 Virtual Machines

Deploying each honeypot and legitimate service on its own dedicated hardware in a real network would yield the most accurate and interesting results, however, those type of resources are out of the scope of this project. Therefore, virtual machines will be used to host each system, as well as two other systems that will be used as part of testing. All the virtual machines will be run from a Windows 10 host using VMWare Workstation 16 Pro version 16.1.2, (VMWare, 2022). Each system will have its own virtual machine connected in a private virtual network with enough resources to run the system comfortably. Exact specifications have been outlined in section 3.4.

The other two virtual machines will be used to simulate an attacker and host a virtual machine used within the further attacker testing. The simulated attacker system will run the latest version of Kali Linux as of writing, version 2022.1, installed on a virtual machine with sufficient virtual hardware assigned, (Kali, 2022). Kali Linux was chosen since it is a popular operating system

used to test the security of systems. The ‘victim’ virtual machine will run Windows 10 and host an SSH server and an FTP server to be used as part of the behavioural testing.

3.3.4 Network

All eight virtual machines will be connected in a virtual network in host-only mode within the 192.168.10.0/24 subnet, using DHCP to assign IP addresses to each machine. A diagram depicting the network has been shown in Figure 17, but it should be noted that all machines could communicate with each other in the network. Each machine in the figure has been labelled with its purpose, followed by its operating system and local IP address.

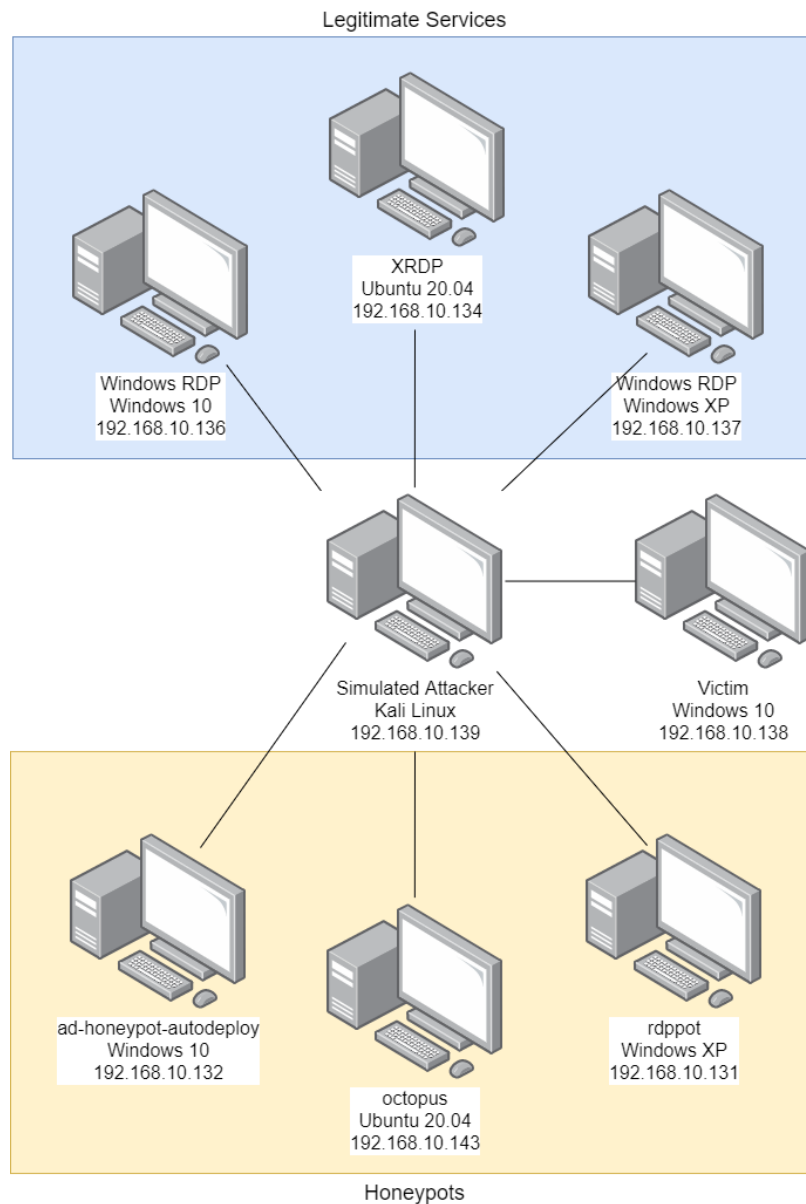


Figure 17: Virtual Network design that will be used during research

3.4 Implementation

This section will outline the steps taken to setup each system prior to testing. It will not cover the installation of the operating systems as this is trivial, it will only cover the steps taken to deploy the RDP service or honeypot. All systems were run virtually using VMWare Workstation 16 Pro 16.1.2 running on Windows 10 Home version 19044. All systems were set up as instructed by software authors when possible, however, many extra steps had to be taken to set up some of the honeypots. Each honeypot and legitimate service has been setup as close to default as possible.

3.4.1 Windows 10 RDP

The following steps were taken to enable RDP.

1. Select 'Start' (Windows icon), and then 'Settings'
2. Select 'System', then 'Remote Desktop' and enable it.

RDP can be seen working in Figure 18 using Remmina from the attacker's machine.

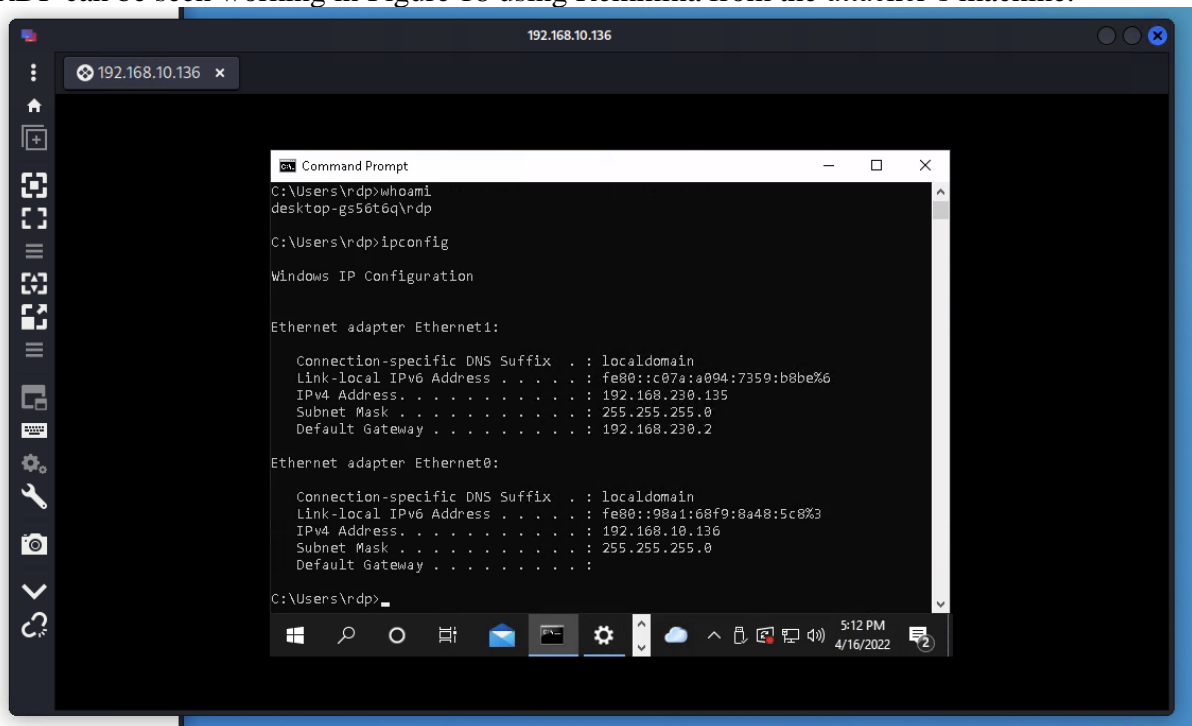


Figure 18: Windows 10 RDP connected to via the Attack machine.

3.4.2 XRDP

The following steps were taken to deploy XRDP on the Ubuntu system:

1. Open a terminal window and run the following commands:

```
sudo apt update && sudo apt install xrdp
```

The service is now installed and should begin running automatically. XRDP can be seen working in Figure 19.

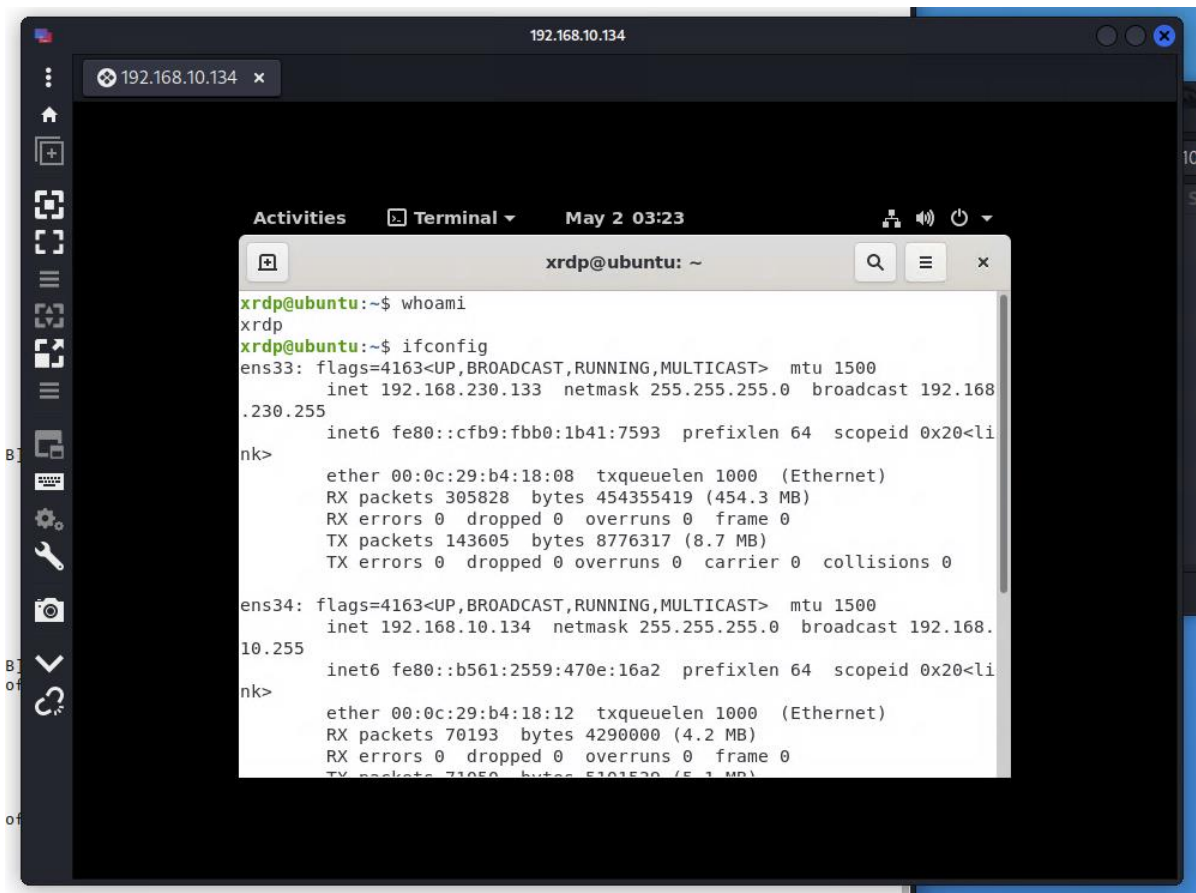


Figure 19: XRDp connected to via the Attack machine.

3.4.3 Windows XP RDP

The following steps were taken on the Windows XP system to enable RDP:

1. Right click on 'My Computer' and click 'Properties'
2. Select the 'Remote' tab and then 'Allow users to connect remotely to this computer.'
3. Press 'OK' and then 'OK'.

RDP can be seen working in the below figure.

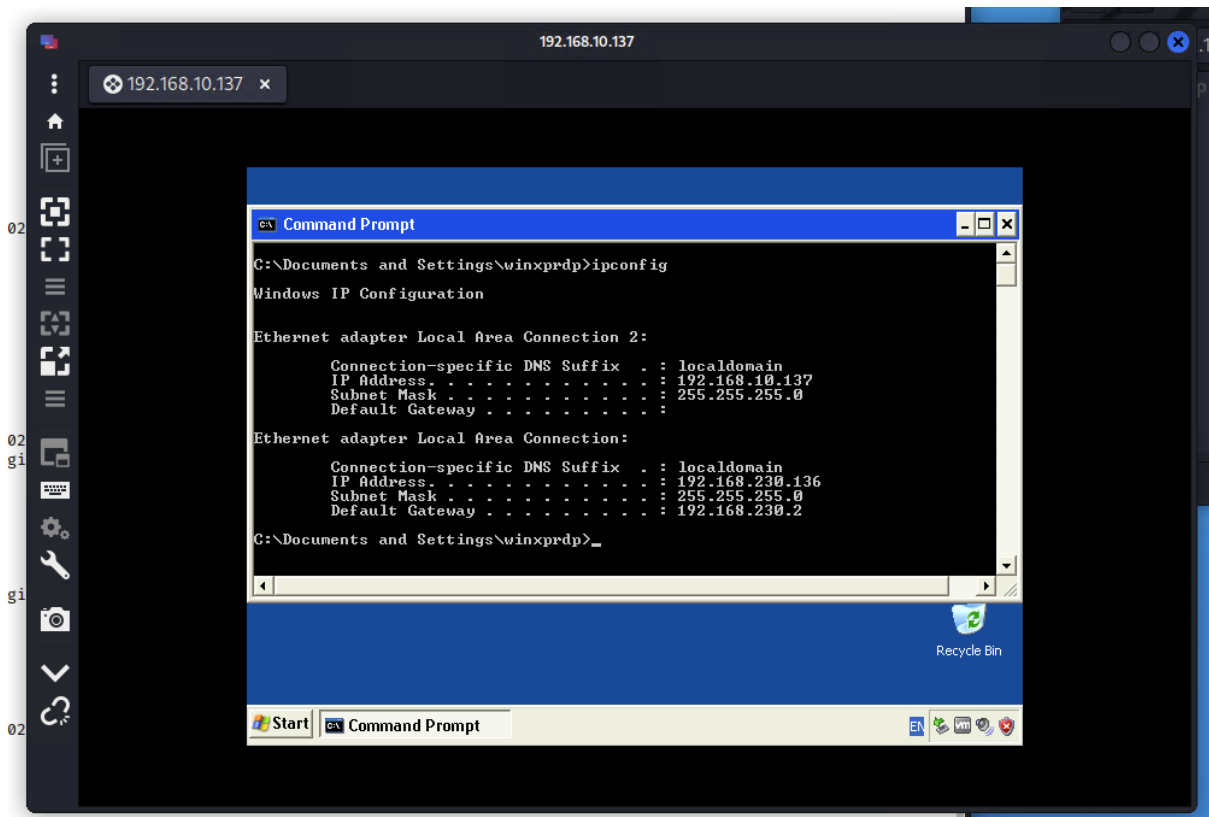


Figure 20: Windows XP RDP connected to via the Attack machine.

3.4.4 ad-honeypot-autodeploy

This software required 12GB of memory, 8CPU cores and 100GB of storage which were provided via the virtual machine. The software was installed on Ubuntu 20.04.3 Desktop and virtualisation was enabled for the virtual machine prior to booting. To install the software, the following steps were taken. An important prerequisite was to download a copy of a Windows 10 ISO file as the software could not do that automatically like the other ISO files it uses.

Packer (HashiCorp, 2022)

1. Other required software was first installed using the following command:

```
sudo apt install git vim p7zip-full curl qemu-kvm libvirt-daemon-system libvirt-clients  
bridge-utils xsltproc pwgen whois python3.8-venv
```

2. Then the software was installed from the GitHub repository using the following command:

```
git clone https://github.com/tothi/ad-honeypot-autodeploy.git
```

3. Change into the newly installed directory and run the 'init-passwords.sh' script. All passwords were set to '!Password123' to meet password requirements.

```
cd ad-honeypot-autodeploy && ./init-passwords.sh
```

4. Move into the 'packer' directory and run the 'get-virtio.sh' script.

```
cd packer && ./get-virtio.sh
```

5. Copy of the previously downloaded Windows 10 ISO file into the ISO directory and rename it to 'Win10_21H2_EnglishInternational_x64.iso'

6. Download GeoList2-City.mmb and place it in the resource's directory, (MaxMind, 2022).
7. Run the below commands to install packer.
 - `curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -`
 - `sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"`
 - `sudo apt-get update && sudo apt-get install packer`
8. Run the 'packer-build-all.sh' script.
`./packer-build-all.sh`

Terraform (HashiCorp, 2022)

1. Install terraform using this command:
 - a. `sudo apt install terraform`
2. Edit the qemu.conf file to fix an issue faced during this research by following these steps.
 - a. `sudo vim /etc/libvirt/qemu.conf`
 - b. Uncomment 'security_driver = 'none''
3. Change into the terraform directory
 - a. `cd ../terraform`
4. Initiate and apply the terraform build. This may take sometime.
 - a. `terraform init`
 - b. `terraform apply`

Ansible (RedHat, 2022)

1. Change into the Ansible directory
`cd ../ansible`
2. Run the following commands:
`python3 -m venv venv`
`./venv/bin/activate`
`pip3 install -r requirements.txt`
`pip wheel install ansible pywinrm faker`
3. Finally deploy the ansible configuration. This step may take a while.
`ansible-playbook -i hosts setup-domain.yml -v`

The honeypot is now ready to be used. To make it accessible to the attacker's machine, the following commands were run:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 3389 -j DNAT --to-destination 192.168.3.112:3389
sudo iptables -A FORWARD -p tcp --dport 3389 -j ACCEPT
sudo iptables -t nat -A POSTROUTING -j MASQUERADE
```

The systems were now ready to be tested. An RDP connection to the system can be seen in Figure 21.

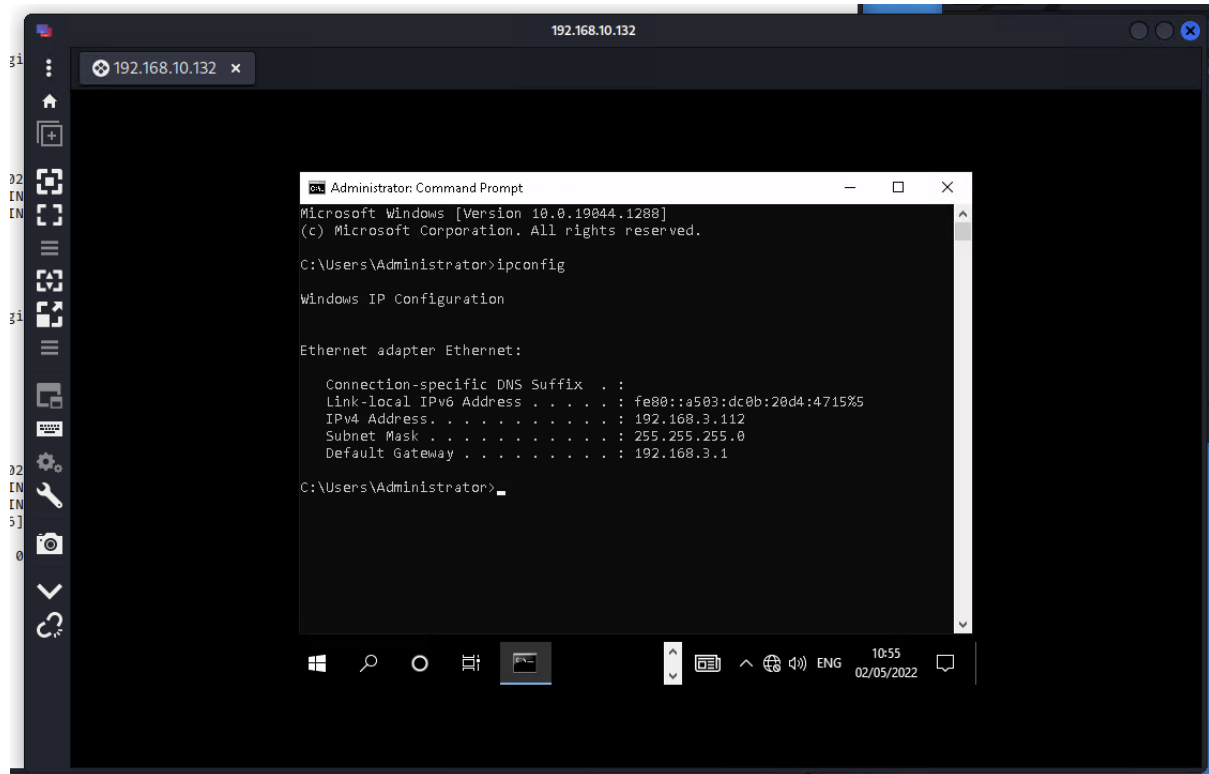


Figure 21: AHA connected to via the Attack machine.

3.4.5 rdppot

This software required 4GB of memory, 2CPU cores and was given 40GB of hard disk space. It was installed on Ubuntu 20.04.3 within a virtual machine with virtualisation support enabled. This honeypot required a Windows XP ISO file to be obtained by the user to be used as part of the honeypot. A copy of Windows XP Pro SP3 was used for this setup. The following steps were taken to setup this device:

1. Install prerequisite software

```
sudo apt install net-tools git python3 python3-pip qemu qemu-kvm libvirt-daemon-system libvirt-dev vim virt-manager bridge-utils
```

2. Clone the GitHub repository

```
git clone https://github.com/kryptoslogic/rdppot
```

3. Change into the new directory and install required python libraries as sudo.

```
cd rdppot && sudo pip install -r requirements.txt, recordclass, pip3 install git+git://github.com/tkuebler/pytyping
```

4. Downgrade 'aiorwlock' python library to version 1.0.0 to fix incompatibility issues.

```
sudo pip3 install --upgrade aiorwlock==1.0.0
```

5. Modify the 'main.py' file in lines 23-25 to reflect the correct path, in this case '/home/honeypot/rdppot'.

```
23 finished_disk_location = "/home/honeypot/rdppot/disks"
24 analysis_location = "/home/honeypot/rdppot/output"
25 pcap_location = "/home/honeypot/rdppot/pcaps"
26
27 if not os.path.isdir(finished_disk_location):
28     raise Exception("finished_disk_location is missing")
29
30 if not os.path.isdir(analysis_location):
31     raise Exception("analysis_location is missing")
32
33 if not os.path.isdir(pcap_location):
34     raise Exception("pcap_location is missing")
35
36 # No underscore is allowed because we add one and split on it to get the ID
37 vm_template_name = "rdppot"
38 pool_size = 2
39
```

Figure 22: Modifications made to the main.py file on lines 23-25 and line 38.

6. Modify to pool size of line 38 to 2.
7. Run virt-manager and create a new virtual machine with the Windows XP ISO, 512mb RAM, 2 CPU cores and 3GB of hard disk space.

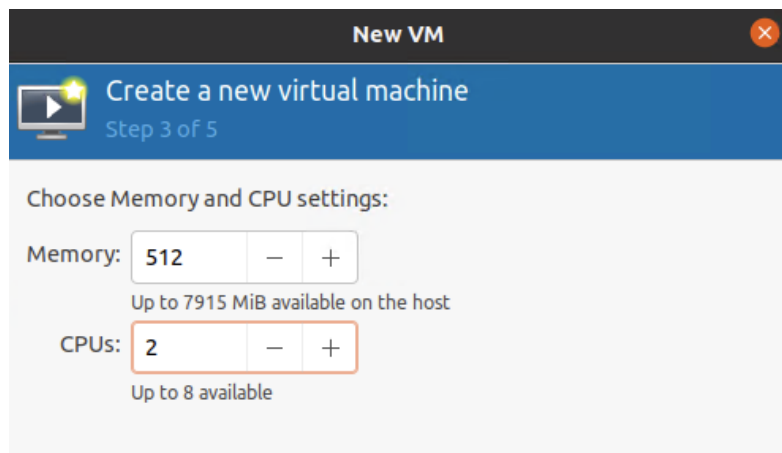


Figure 23: Setting the memory and CPU specifications on the VM.

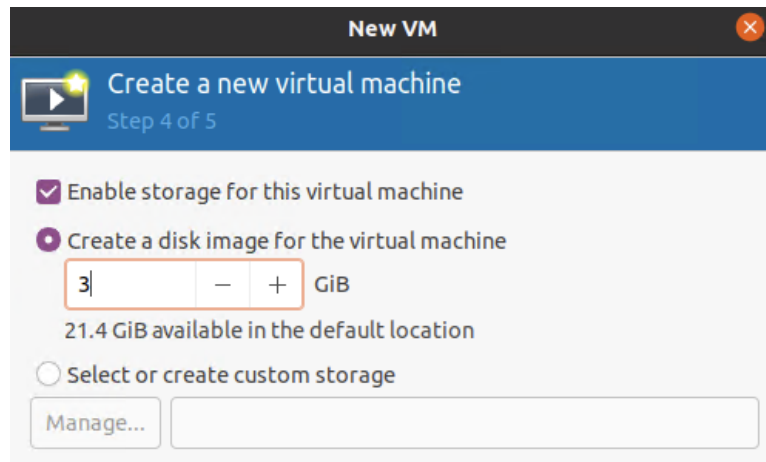


Figure 24: Setting the hard disk size on the VM.


8. Name the virtual machine 'winxp_template'.
9. Change the network adapter to virtio

Virtual Network Interface

Network source: Virtual network 'default' : NAT ▼

Device model: virtio ▼

MAC address: 52:54:00:83:16:a7

IP address: Unknown 

Link state: active

Figure 25: Setting the network adapter on the VM.

10. Boot up the VM and install virtio drivers for the network adapter, as outlined in this video from 16:30, (Installing Windows XP into QEMU/KVM , 2020).
11. Create a password for the admin account.
12. Enable RDP following the same steps outlined in section 3.4.3.
13. Enable auto login to allow the VMs to start up properly on their own, as outlined by John Savill, (Savill, 2002).
14. Deploy the honeypot with the following command:

```
sudo python3 main.py
```

An RDP connection to the service can be seen in the below figure.

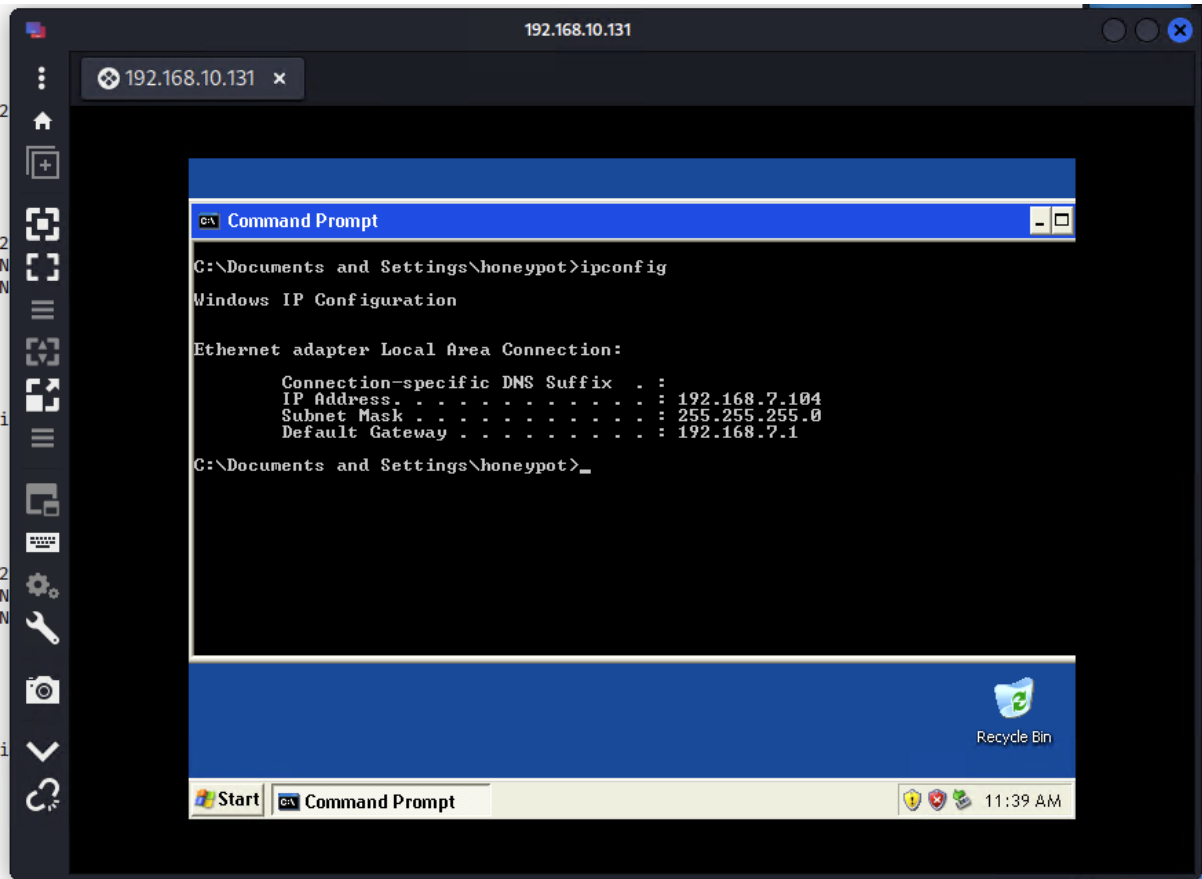


Figure 26: rdpot connected to via the Attack machine.

3.4.6 Octopus

Out of the three honeypots, octopus was the most straightforward to deploy. It was deployed on Ubuntu version 20.04.03 and used XRDP as its RDP service. The virtual machine was provided 4GB of RAM, 2 CPU cores and 20GB hard drive space. The following step was taken to install the software:

1. Run the below two commands to download and install the honeypot:

```
git clone https://github.com/qeeqbox/octopus.git && cd octopus && chmod +x setup.sh  
sudo ./setup.sh "rdp"
```

An RDP connection to Octopus can be seen in Figure 27.

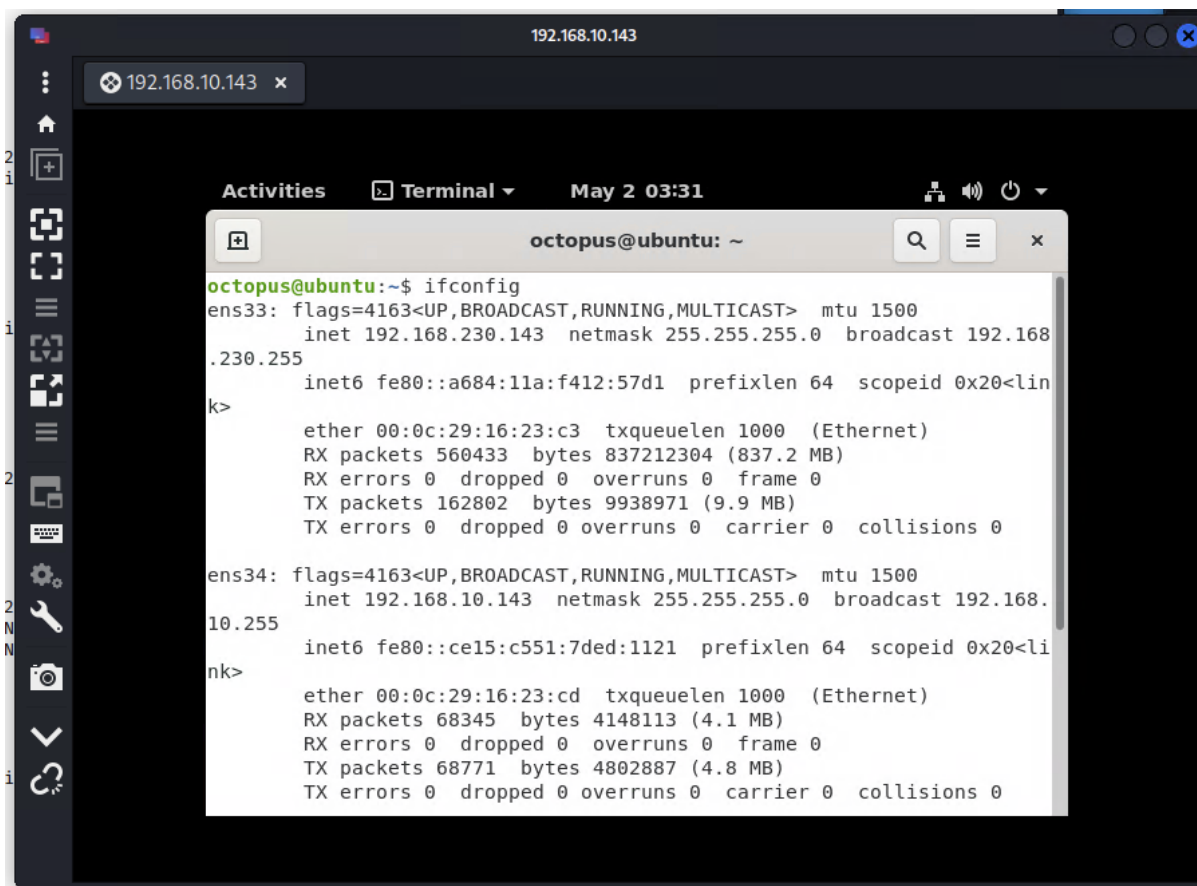


Figure 27: Octopus connected to via the Attack machine.

3.5 Experimentation

3.5.1 Fingerprinting

Fingerprinting will be conducted against the devices primarily on a network and application level using two different tools: Nmap, and Xprobe2, (Lyon, 2022) (binarytrails, 2021). Nmap excels in port-based scanning and identification of systems primarily using TCP packets, while Xprobe2 is designed to fingerprint operating systems using ICMP packets. These experiments are based off the research analysed in section 2.4.1, primarily research discussed in section 2.4.1.1 carried out by Dahbul et al. which used networking fingerprinting as a means of identifying anomalies in honeypots, (Dahbul, et al., 2017). The Nmap scans used are based off similar research by Mohammadzadeh et al. which used multiple types of Nmap scans, some of which have been used as the basis for the NMAP scans used in these experiments, (Mohammadzadeh, et al., 2013). The use of Xprobe2 is based off research conducted by Srinivasa et al. which outlined a honeypot fingerprinting framework, as discussed in section 2.4.1.4, (Srinivasa, et al., 2021). This paper highlighted Xprobe2 as a potential fingerprinting tool that would be effective at identifying honeypots. Furthermore, the tool is also mentioned in two of the countermeasure research papers discussed in section 2.5 by Naik et al, (Naik, et al., 2018) (Naik, et al., 2021).

Firstly, as seen in previous research, Nmap will be used to probe each device using two different types of scans, a quick scan, and a more thorough scan. The first scan attempts to fingerprint operating system, running services, and trace a route to the device. The second scan

does the same but also scans all the ports with a higher intensity. Each scan will be run 3 times against each machine, and if any differences are found between each scan, these will be noted. The flags for the two scans can be seen in Table 4.

Table 4: The two different types of NMAP scans and their respective flags.

Type	Flags
Quick	-A -Pn
Intense	-sV -O -p 1-65535 -Pn --intensity 9

Two types of Xprobe2 scans will be performed, a standard scan and a port based one targeting the RDP port 3389 using the ‘p’ flag, as can be seen in an example command below.

```
xprobe2 192.168.1.4 -p tcp:3389:open
```

Each scan will have its results recorded as well as its time to complete, alongside a network capture taken from the attacker’s machine using Wireshark. This capture file will be stripped of all traffic that was not to or from the attacker’s machine, using a script found in Appendix E - . This data can be used to retrieve accurate timing data and evaluate a level of network traffic created by each scan to determine how they compare in terms of noise.

3.5.2 Latency

Latency testing will consist of pinging each virtual machine and measuring the time it takes to reply as accurately as possible. This will be done using ICMP packets via the ping command, and TCP packets using paping, (Google, 2010). The advantage of using two tools is to demonstrate any differences in results between the two types of ‘pings’ that may be notable for determining the type of system. This experiment is based off of work discussed in section 2.4.4, first highlighted by Holz and Raynal, but further developed by Fu et al. who used both ICMP and TCP packets to check for latency, (Holz & Raynal, 2005) (Fu, et al., 2006).

Each system will be tested 100 times from the attacker machine and the round-trip time will be recorded. These results can then all be compared to find any anomalies or outliers. The actual scans were carried out using two scripts to automate the process of running and parsing the results. There two scripts can be found in Appendix C.

3.5.3 Behaviour

Two tests will be done against the machines to determine behaviour as a factor. These two tests will focus on how the devices behave in certain circumstances to see if there are any clear differences in behaviour between legitimate and honeypot devices. This test is based on the fact that honeypots should not be able to connect out of their honey network and will determine whether or not the author has put in any measures to cripple the systems capabilities in this way. This experiment is primarily based off the research examined in section 2.4.2. The first experiment is inspired by a honeypot behaviour highlighted by Tsikerdekis et al. in section 2.4.2.1, that honeypots often automatically redeploy themselves after having been accessed, (Tsikerdekis, et al., 2018). The further attack test is based off research by Uitto et al. which highlights the inability to conduct further attacks as a key behavioural characteristic of honeypots, (Uitto, et al., 2017). Therefore, if a device blocks further network communication it could be assumed it was a honeypot.

The first will be an object permeance test where a new text file will be created on the systems home drive and left for 10 minutes. Any changes made to the file upon return will be noted.

The second test will involve furthering an attack from the device. For the Windows XP based rdppot, an attempt will be made to use FTP to access the 'Victim' virtual machine. The Windows 10 and Ubuntu based honeypots will use SSH to attempt to access the device.

3.6 Evaluation

The results from the experiments are to be evaluated on their effectiveness at correctly identifying honeypots from legitimate services in a purple teaming engagement. This effectiveness is judged on three factors: accuracy, speed, and discreetness. The most effective technique will be the one that can most accurately fingerprint a system in the fastest and most inconspicuous manner. The least effective method will be the one that is the least accurate, takes the most time, and creates the most notice. These evaluation parameters are based off similar fingerprint technique evaluation research by Mohammadzadeh et al. in their dynamic honeypot research, (Mohammadzadeh, et al., 2013).

3.7 Conclusion

This section of the report has discussed the methodology that will be used to carry out experiments fingerprinting RDP honeypots. The main experiments inspired by the research were first discussed, laying out the plan of testing. This was followed by the design of various parts of the experiment including the honeypots and legitimate services to be used, as well as the specifications around the virtualised environment and the virtual network that would be used to communicate between devices. The implementation of this design was then laid out, including full steps to recreate and configure each virtual machine. This was followed by an in-depth look at each experiment that was to be carried out, including the steps that were to be taken and references to any relevant literature that the experiment were based off. Finally, the evaluation method was discussed that is to be used to evaluate the results of the tests in regards to the overall aim of the report.

4. Results

4.1 Introduction

This section will analyse the results obtained from the experiments discussed in section 3.5 of this report. Any network data being examined has been cleaned to remove any network traffic that was out of the testers control, as mentioned in section 3.5.1. All experimentation was done on the same bare-metal hardware within a two-hour window, with each virtual machine being deployed as needed. The results will be analysed through the lens of effectiveness, judging them based off speed, accuracy, and stealth.

4.2 Experiment 1 - Fingerprinting

The first experiment aimed to evaluate the effectiveness of fingerprinting tools, Nmap and Xprobe2, against the honeypots. Each tool will be evaluated separately, looking at both types of scans that were conducted by each and comparing the results from the honeypots to their legitimate counterparts.

4.2.1 NMAP

The first scan performed with Nmap was a quick, all-rounder, scan that would be used to obtain quick results. Comparing the services and honeypots, no real difference was present in the scan results, as has been shown in Figure 28 and Figure 29. The results for the other four services can be found in Appendix F - NMAP Fingerprinting Results

```
(kali@kali)-[~]
└─$ nmap -A -Pn 192.168.10.131
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:25 EDT
Nmap scan report for 192.168.10.131
Host is up (0.0016s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.43 seconds
```

Figure 28: Nmap quick scan results for rdppot.

```
(kali@kali)-[~/scripts]
└─$ nmap -A -Pn 192.168.10.137
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 12:23 EDT
Nmap scan report for 192.168.10.137
Host is up (0.00035s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 26.01 seconds
```

Figure 29: Nmap quick scan results for Windows XP.

These figures show no real difference between legitimate system and honeypot, showing that a quick Nmap scan does not add to any evidence of a honeypot.

However, the intense Nmap scan showed more details as to the underlying architecture of the systems. For example, the difference between rdppot and Windows XP were very notable, as shown in Figure 30 and Figure 31.

```

(kali㉿kali)-[~]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.131
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:26 EDT
Nmap scan report for 192.168.10.131
Host is up (0.00032s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
3389/tcp  open  tcpwrapped
MAC Address: 00:0C:29:EA:08:DA (VMware)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.47 seconds

```

Figure 30: Nmap intense scan results for rdppot.

```

(kali㉿kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.137
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 12:24 EDT
Nmap scan report for 192.168.10.137
Host is up (0.00021s latency).
Not shown: 65534 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
3389/tcp  open  ms-wbt-server Microsoft Terminal Services
MAC Address: 00:0C:29:E8:14:73 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running: Microsoft Windows XP
OS CPE: cpe:/o:microsoft:windows_xp::sp3
OS details: Microsoft Windows XP SP3
Network Distance: 1 hop
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 125.75 seconds

```

Figure 31: Nmap intense scan results for Windows XP.

Firstly, the scan outright defines the honeypot as a Linux system. Comparing this to the initial quick scan is a giveaway of something being afoot. Furthermore, the service of 3389 is now identified as ‘tcpwrapped’ rather than the expected ‘ms-wbt-server’. Finally, the scan completed much quicker, taking only 16.47 seconds compared to the 125.75 seconds that the legitimate service took.

This was a similar story for AHA and Windows 10, expect slightly more subtle, as shown in the two below figures.

```

(kali@kali)-[~]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.132
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:02 EDT
Nmap scan report for 192.168.10.132
Host is up (0.00035s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
MAC Address: 00:0C:29:65:4D:26 (VMware)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.92%E=4%D=4/16%OT=3389%CT=1%CU=37670%PV=Y%DS=1%DC=D%G=Y%M=000C29
OS:%TM=625ADA93P=x86_64-pc-linux-gnu)SEQ(SP=103%GCD=2%ISR=109%TI=I%CI=Z%II
OS:=I%TS=U)SEQ(SP=103%GCD=1%ISR=109%TI=I%CI=Z%II=I%SS=S%TS=U)OPS(O1=M5B4NW0
OS:NNS%O2=M5B4NW0NNS%O3=M5B4NW0%O4=M5B4NW0NNS%O5=M5B4NW0NNS%O6=M5B4NNS)WIN(
OS:W1=FA00%W2=FA00%W3=FA00%W4=FA00%W5=FA00%W6=FA00)ECN(R=Y%DF=Y%T=7F%W=FA00
OS:%O=M5B4NW0NNS%CC=N%Q=)T1(R=Y%DF=Y%T=7F%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=Y
OS:%T=40%W=0%S=Z%A=S+F=AR%O=%RD=0%Q=)T3(R=Y%DF=Y%T=40%W=0%S=Z%A=O%F=AR%O=%R
OS:D=0%Q=)T4(R=N)T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%
OS:T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD
OS:=0%Q=)U1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE
OS:(R=Y%DFI=N%T=40%CD=S)

Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 32.59 seconds

```

Figure 32: Nmap intense scan for ad-honeypot-autodeploy (AHA).

```

(kali@kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.136
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:59 EDT
Nmap scan report for 192.168.10.136
Host is up (0.00029s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
7680/tcp  open  pando-pub?
MAC Address: 00:0C:29:9A:8E:E8 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): Microsoft Windows XP|2008|7 (88%)
OS CPE: cpe:/o:microsoft:windows_xp::sp2 cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_server_2008:r
2 cpe:/o:microsoft:windows_7
Aggressive OS guesses: Microsoft Windows XP SP2 (88%), Microsoft Windows Server 2008 SP1 or Windows Server 2008 R2 (86%),
Microsoft Windows 7 (85%), Microsoft Windows Fundamentals for Legacy PCs (XP Embedded derivative) (85%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 223.51 seconds

```

Figure 33: Nmap intense scan for Windows 10.

The scan against the honeypot returns an unknown TCP/IP fingerprint, however, clear indicators of a Linux system can be seen within it by noticing the string ‘pc-linux-gnu’. The Windows 10 system also does not return an unknown fingerprinting, instead guessing it to be a Windows system. The Windows 10 system also has port 7680 open, something missing from the honeypot. Finally, as seen with Windows XP and rdppot, there is a large difference in time to complete the scan. The scan against AHA only took 32.59 seconds compared to the 223.51 seconds it took to complete the scan against the Windows 10 system.

Finally, XRDP and Octopus showed absolutely no difference in their intense scan results. This is likely due to how basic Octopus is as a honeypot and is extremely similar to a standard XRDP service in its design. The only slight difference was in the time it took to complete, as shown in Figure 34 and Figure 35, but it is such a short amount of time it could not be reliably used.


```

(kali@kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.143
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:44 EDT
Nmap scan report for 192.168.10.143
Host is up (0.00024s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
3389/tcp  open  ms-wbt-server xrdp
MAC Address: 00:0C:29:16:23:CD (VMware)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 27.36 seconds

```

Figure 34: NMAP intense scan for Octopus.

```

(kali@kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.134
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 12:13 EDT
Nmap scan report for 192.168.10.134
Host is up (0.00041s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
3389/tcp  open  ms-wbt-server xrdp
MAC Address: 00:0C:29:B4:18:12 (VMware)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 27.09 seconds

```

Figure 35: NMAP intense scan against XRDP.

These results show that for two out of three of the honeypots, comparing an intense and quick NMAP scan is an accurate and quick method of identifying a honeypot. This was not the case for the Linux based RDP service and honeypot which showed no difference in either scan. A table has been created to note if there were differences in the scans between the honeypots, as can be seen below.

Table 5: A table noting if differences were seen in the scans against the pairings of honeypot and legitimate system.

Services	NMAP Quick	NMAP Intense	Xrpobe2	Xprobe2 Port
rdppot and Windows XP	N	Y	Y	Y
aha and Windows 10	N	Y	Y	Y
octopus and XRDP	N	N	Y	Y

In terms of noise, the intense scans generated significantly more network traffic than the quick scans, as was expected. The number of packets generated by each scan has been summarised in Table 6.

Table 6: The number of network packets generated by each NMAP scan against each service.

Scan Type	rdppot	Windows XP	AHA	Windows 10	Octopus	XRDP
Quick	2089	2076	2075	2112	2106	2104
Intense	131128	131377	131285	132171	131133	131133

The intense scan was much louder for each service, generating around 62 times as many packets over the course of the scan. This demonstrates how much more noise the intense Nmap scan generated compared to the quick scan, making the quick scan a stealthier, but significantly less accurate option.

The second scan also took considerably longer to complete on the two legitimate Windows services. The time to complete in seconds has been plotted in a table and a figure below.

Table 7: Time to complete both NMAP scans against each service, in seconds.

Scan Type	rdppot	Windows XP	AHA	Windows 10	Octopus	XRDP
Quick	24.43	26.01	20	25.93	24.4	24.42
Intense	16.47	125.75	32.59	223.51	27.36	27.09

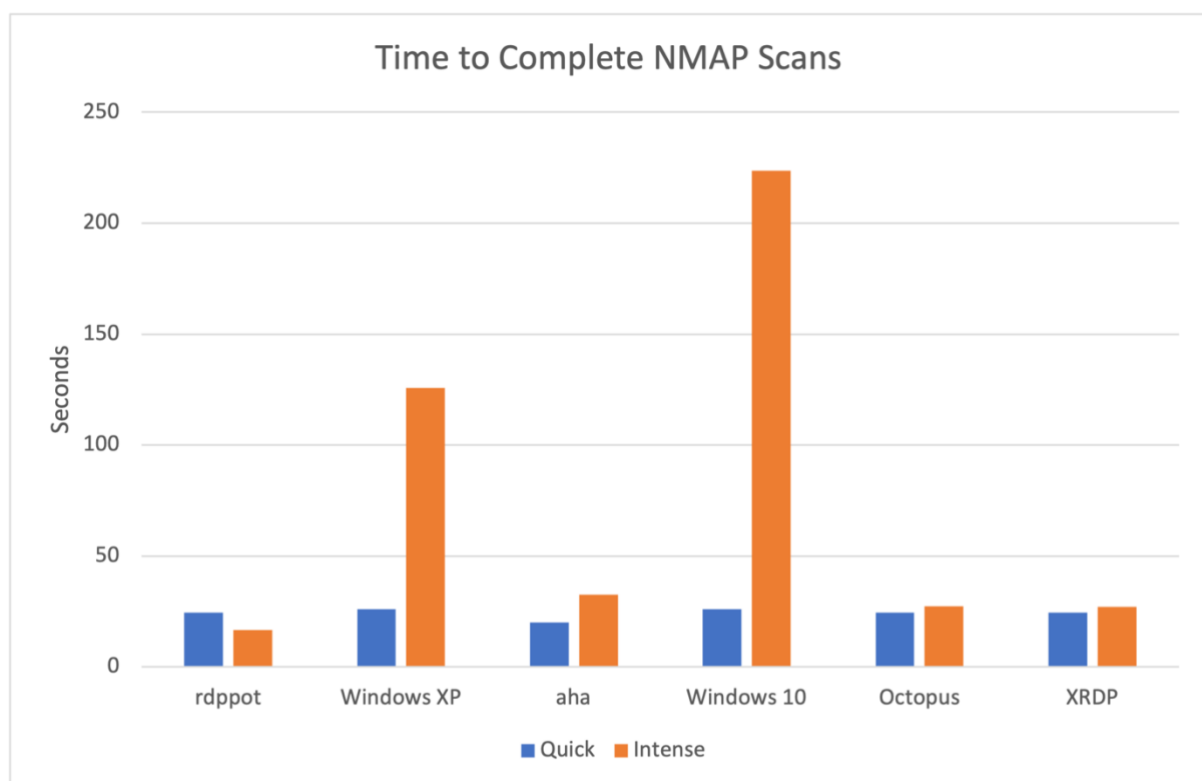


Figure 36: A bar chart of the time to complete each Nmap scan against each service.

4.2.2 Xprobe2

The first scan performed with Xprobe2 was the default scan. This made use of all the scripts used by Xprobe2 to identify operating systems with no extra configuration options used. Considering the age of Xprobe2 as a tool and its lack of popularity in recent years, the results obtained using it were extremely accurate. The tool identified the family of the operating system, Windows, or Linux, perfectly, with the only slight mistake being classifying Windows 10 as Windows Server 2000 with a probability of 91%. These results have been summarised in the below table.

Table 8: Results of the normal Xprobe2 scan along with the real OS and probability.

Service	Xprobe2 OS	Real OS	Probability (%)
rdppot	Linux	Linux	100
Windows XP	Windows XP	Windows XP	100
AHA	Linux	Linux	100
Windows 10	Windows Server 2000	Windows 10	91
Octopus	Linux	Linux	100
XRDP	Linux	Linux	100

The second Xprobe2 scan took advantage of specific ports, was not any more accurate than the first. It identified the same operating systems, with the only notable difference being that it identified Windows 10 as Windows NT 4 Server, with only a 77% probability.

The most notable difference between the two scans was the number of packets used. While both low amounts, the second scan used almost twice as many packets on average, as shown in Figure 37.

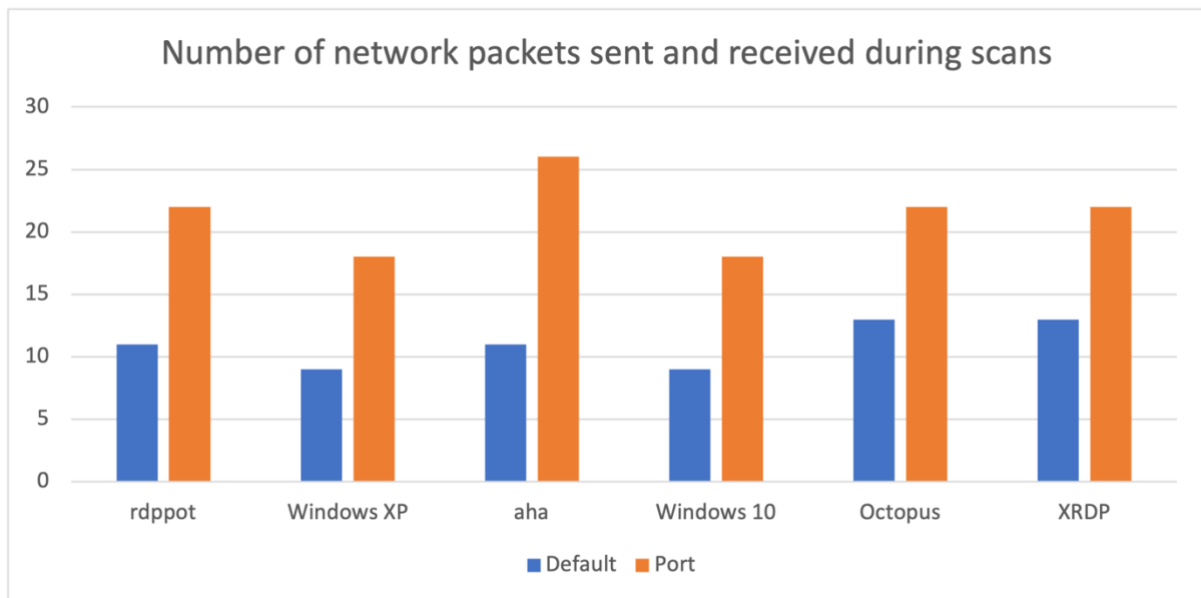


Figure 37: A bar chart of the number of network packets transferred during each scan against each service.

There was a clear increase in the number of packets used by Xprobe2 with no notable difference in identification.

Regarding speed, Xprobe2 scans completed quickly, with no scan taking longer than even a minute. The Linux based services, Octopus and XRDP, completed the quickest in both categories, with rdppot and Windows 10 taking the longest with their port-based scans, and Windows XP and AHA being the quickest in the default category, as shown in the below table.

Table 9: Time to complete in seconds for each Xprobe2 scan against each service.

Scan Type	rdppot	Windows XP	AHA	Windows 10	Octopus	XRDP
Default	22.54	2.93	2.4	25.61	4.25	4.86
Port	46.4	7.32	11.2	44.97	5.55	5.52

This data has also been graphed for easy of comparison.

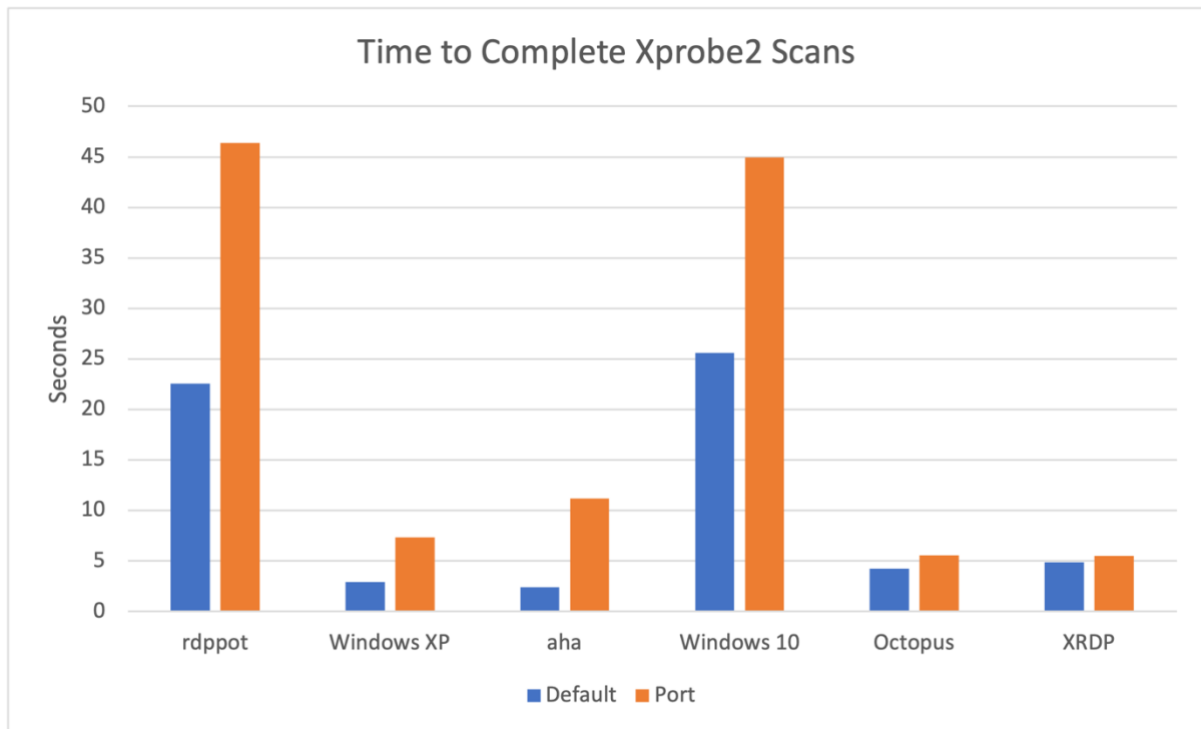


Figure 38: A bar chart showing the time to complete each Xprobe2 scan against each service.

The full raw results for Xprobe2 scans can be found in Appendix G - Xprobe2 Fingerprinting Results.

4.3 Experiment 2 - Latency

The results obtained from both latency tests were processed using standard deviation to account of any outliers in network traffic spikes or anomalies. These statistics have then been processed into box plots for easy comprehension. Network captures were also taken during the period the requests were being sent. These will be analysed to display how much traffic was generated to compare speed and noise.

4.3.1 ICMP

The first latency test relied on ICMP packets sent using the standard Linux 'ping' binary. The 100 different requests sent to each system have had the round-trip times plotted in a box plot, as seen in Figure 39. The full raw data for ICMP RTT can be found in Appendix H - ICMP Latency.

Round Trip Time for ICMP Ping

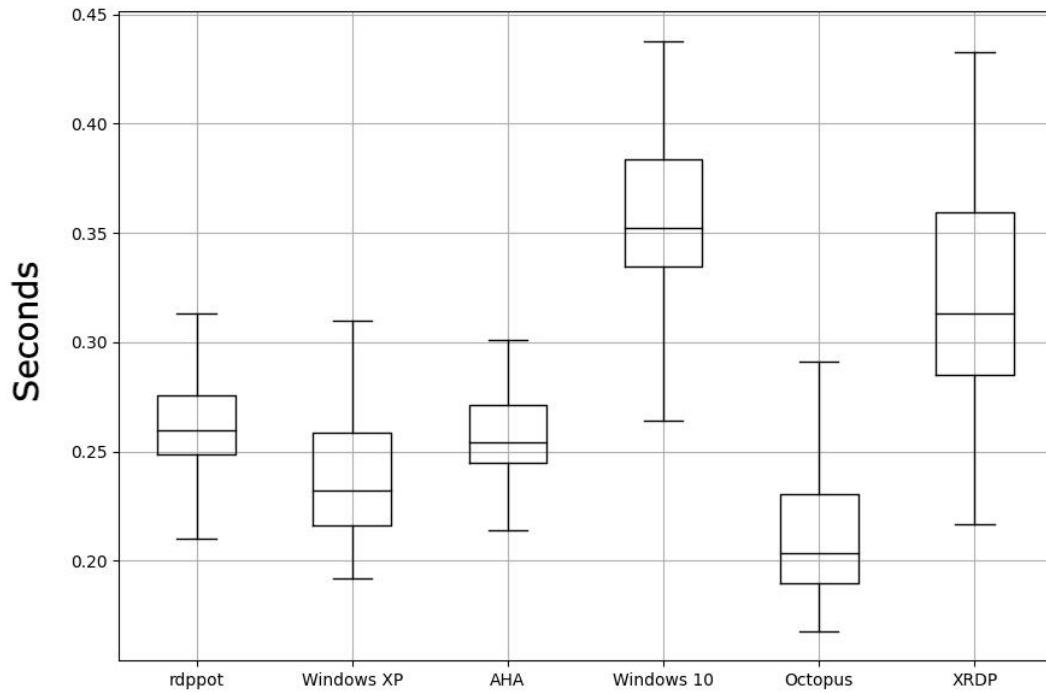


Figure 39: A box plot of the round-trip time for ICMP pings against each service

All the ICMP requests experiments generated exactly 200 network packets. Each test took slightly over 100 seconds to complete, as illustrated in the below figure.

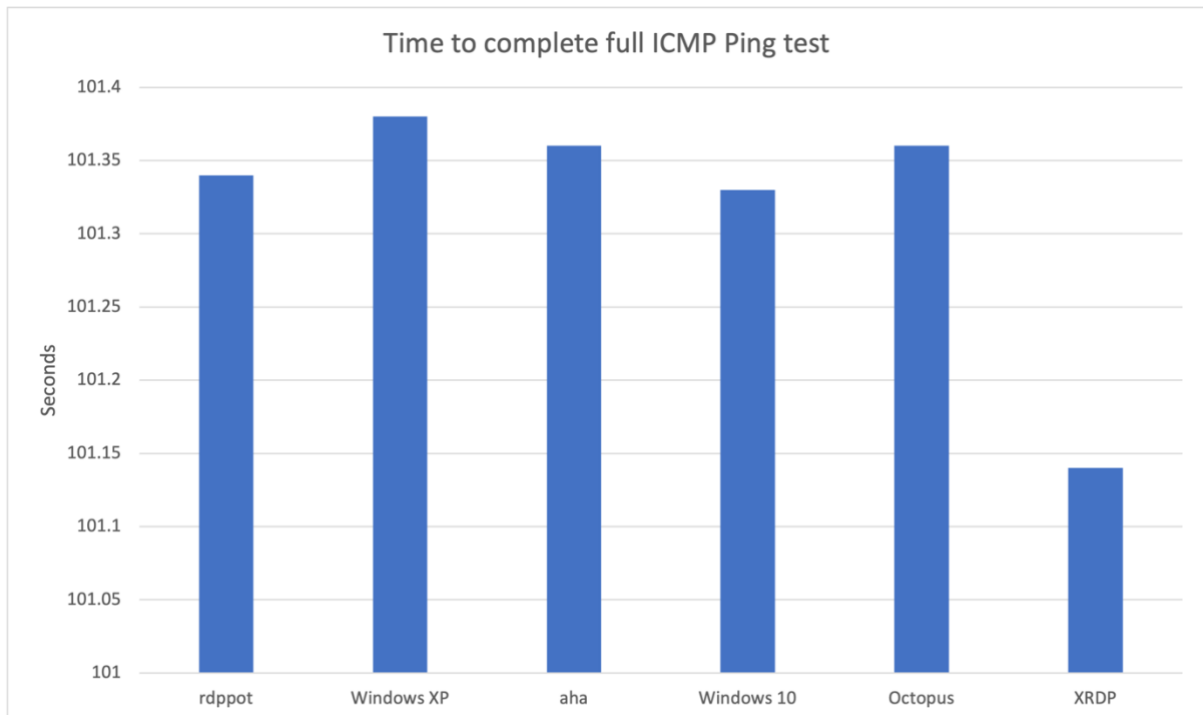


Figure 40: Time to complete ICMP ping test against each service

4.3.2 TCP

The second latency test relied on sending TCP packets using ‘paping’. Each device had 100 different requests sent. The round-trip time for each device has been plotted in the box plot below. The full raw results for TCP RTT can be found in Appendix I - TCP Latency

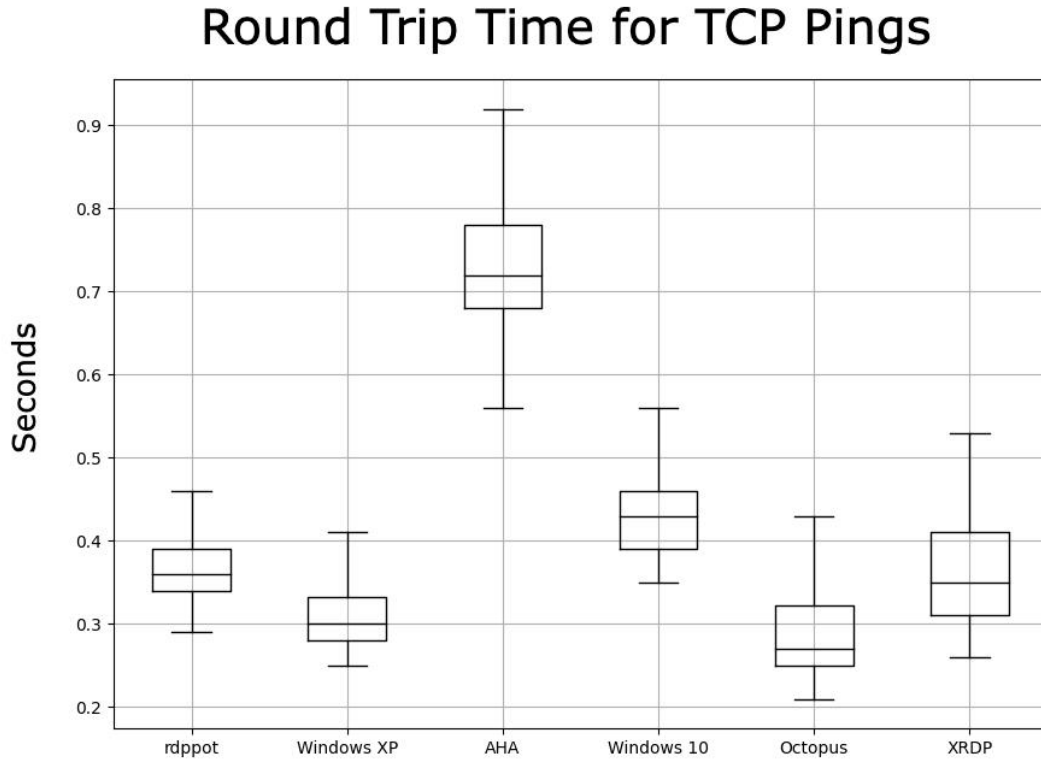


Figure 41: A boxplot of the round-trip time for TCP pings to each service.

Compared to the ICMP based test, there was a difference in the number of TCP packets sent to each device, as illustrated in Figure 42.

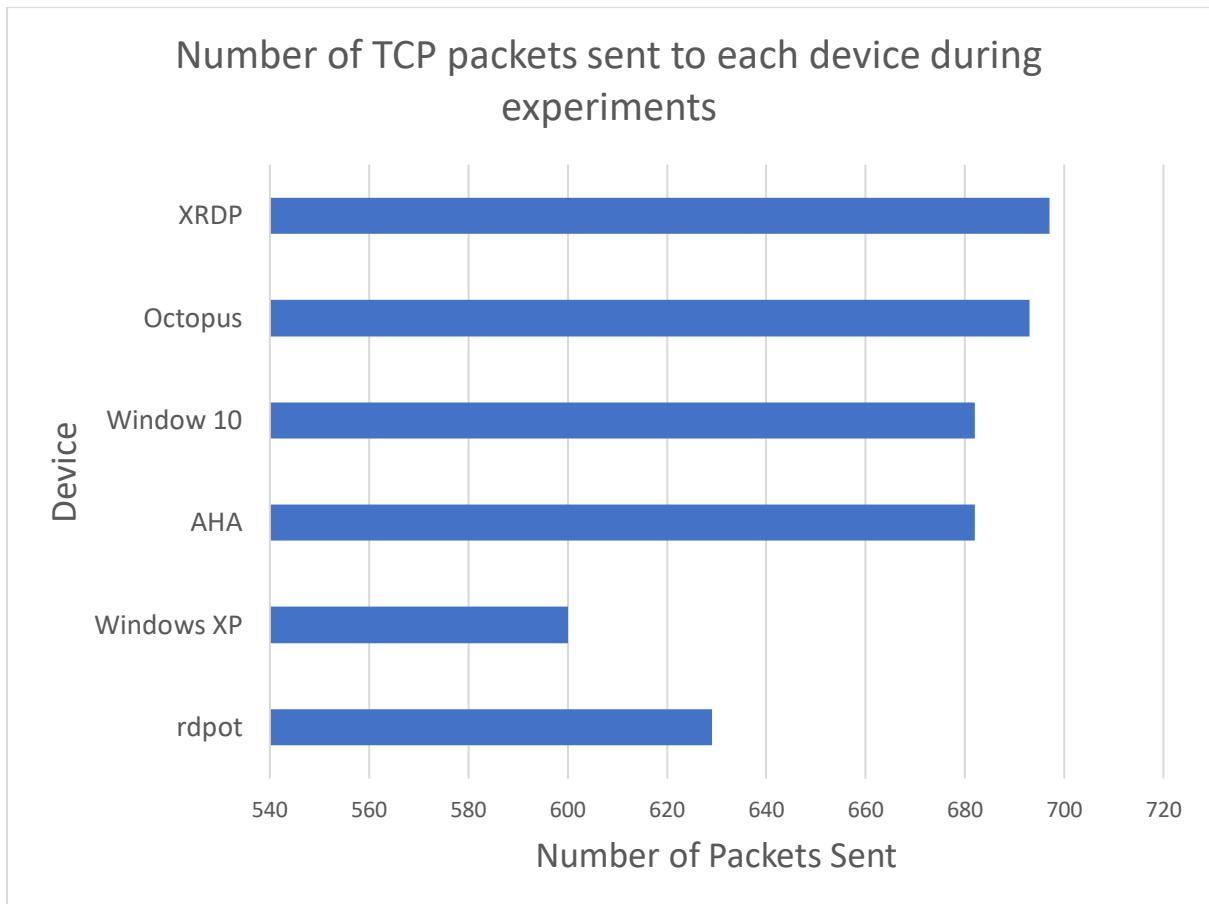


Figure 42: Number of TCP packets sent to each device during experiments

There was a notable difference between Windows XP, rdppot, and the other four systems. This is likely due to an underlying difference between Windows XP and newer operating systems like Windows 10 and Ubuntu.

Each test took slightly over 111 seconds to complete. The full timings have been shown in Figure 43.

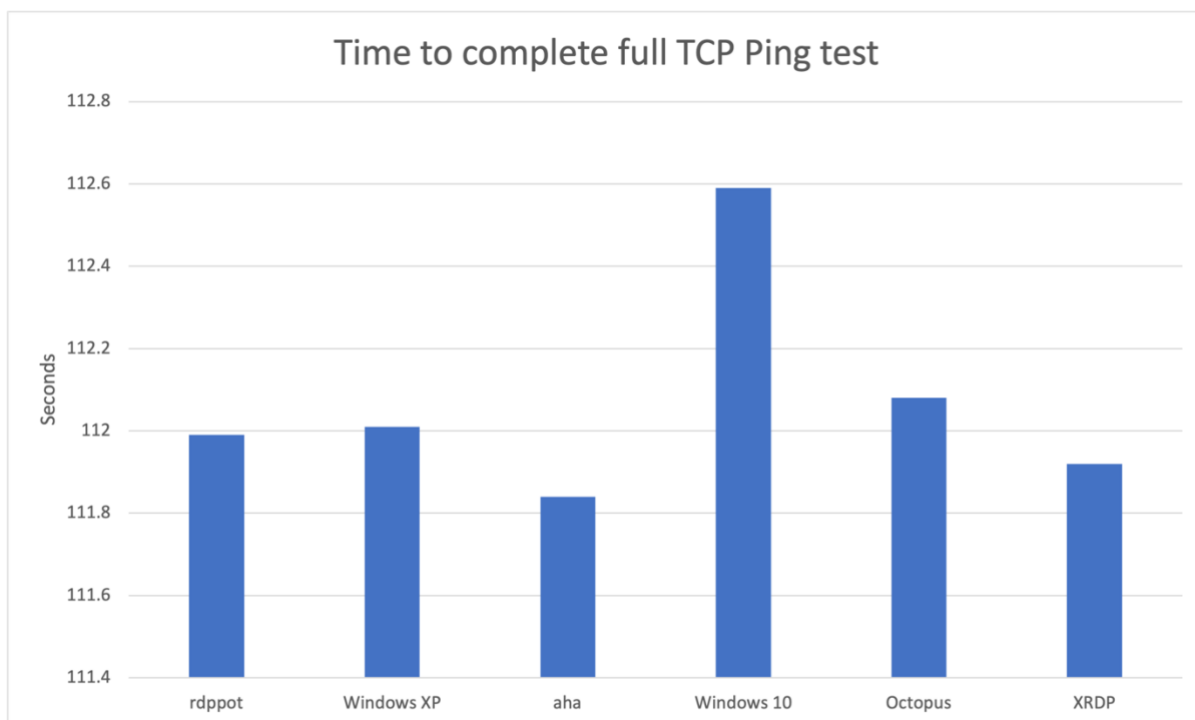


Figure 43: Time to complete TCP ping test against each service.

4.4 Experiment 3 - Behaviour

The final experiment involved analysing the behaviour of the three different honeypots in two manners: the first for the persistence of a file after a 10-minute period, and the second the ability for perform a ‘further attack’ that was tested by attempting to connect to an SSH or FTP server on a ‘victim’ machine. The results have been recorded as either passing or failing the experiment. A pass counts as exhibiting behaviour expected from a normal machine, while a fail shows evidence the device is a honeypot.

4.4.1 File Persistence

This test involved creating a file on the honeypot, disconnecting, and reconnecting after 10 minutes to examine if any changes had been made to the file. Of the three honeypots, rdppot had the only notable result. The file could not be found after reconnecting to the machine, showing the honeypot had been reset at some point between disconnecting and reconnecting. As shown in Figure 44, the file was created successfully under the user’s profile.

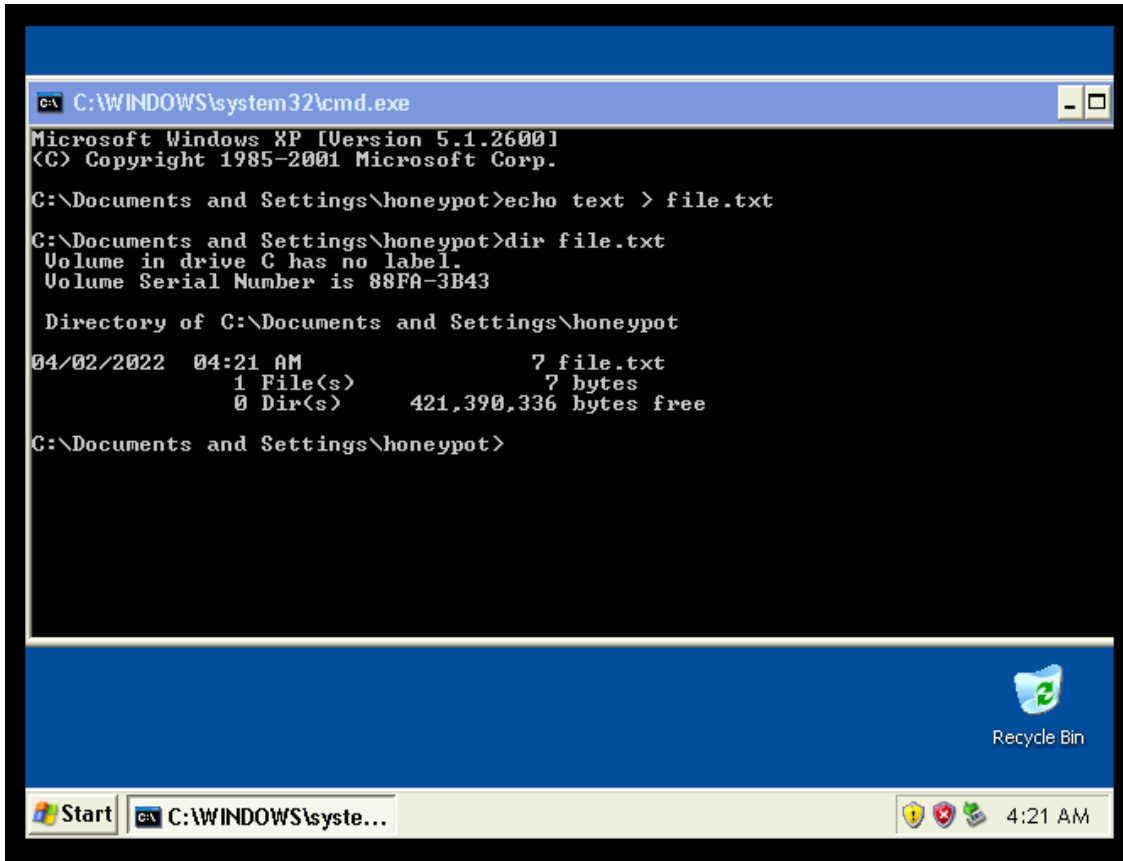


Figure 44: File "file.txt" created on the device and checked using the 'dir' command.

When the tester reconnected to the device after 10 minutes, the file was no longer present, as shown in Figure 45.

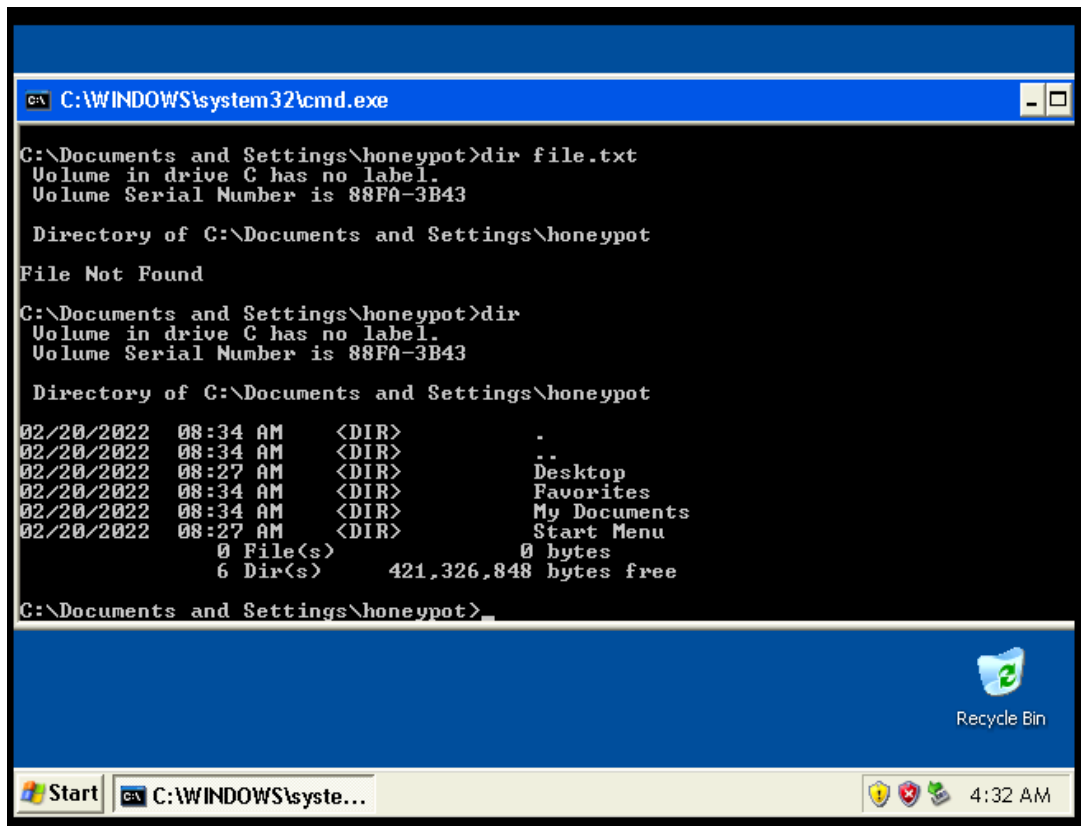


Figure 45: The same directory after reconnecting, showing the honeypot has been tampered with.

The other two honeypots did not show the same behaviour, with the file still being present on each device after the 10 minute testing period, as demonstrated in Figure 46 and Figure 47 on the AHA honeypot.

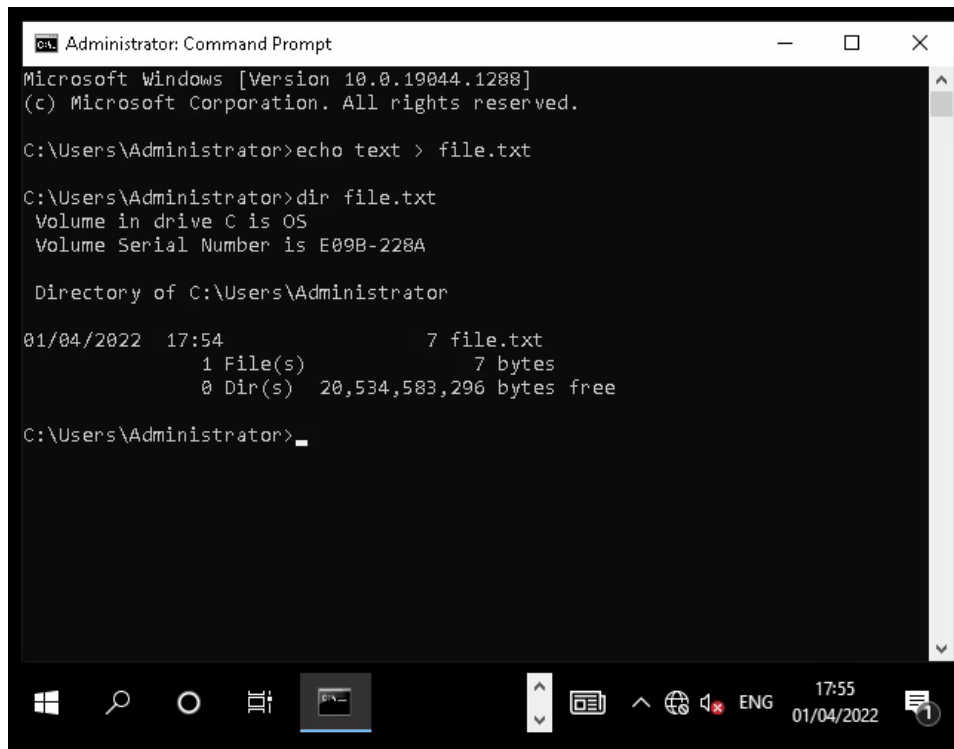


Figure 46: The file being created inside the AHA honeypot.

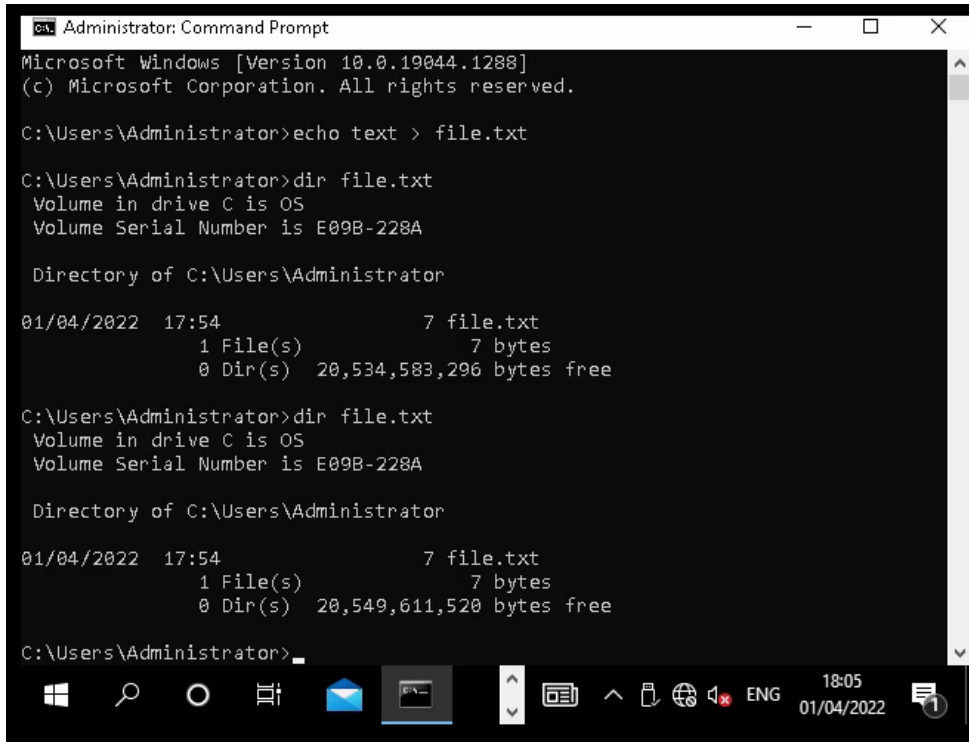


Figure 47: The file persisting on AHA after 10 minutes.

The same behaviour was seen on the Octopus honeypot as was seen on AHA. These results, along with the further attacks test, have been summarised in Table 10. The full results for this experiment can be found in Appendix J - File Persistence Results.

4.4.2 Further Attacks

The second behaviour test attempted to further an attack from the honeypot. Due to the nature of honeypots compared to legitimate systems, they should not be able to engage in any form of unethical behaviour. All three honeypots were able to do connect to either an SSH or FTP server, hence proving further attack would be possible. This has been summarised in Table 10. The full results for this experiment can be found in Appendix K - Further Attack Results.

Table 10: Results of the two different behaviour tests (F = fail, P= pass)

Test	rdppot	AHA	Octopus
File Persistence	F	P	P
Further Attack	P	P	P

5. Discussion

This project was undertaken with the aim of identify and evaluate the most effective detection methods for purple team engagements against contemporary RDP honeypots. This section will answer the overall aim of this research by addressing each sub aim using the methods set out above.

5.1 Existing Detection Methods

The first research question aimed to answer what the existing honeypot detection methods were and how they compared against each other. This was key to identifying techniques that may be used against RDP honeypots for the purposes of the overall aim. The question was answered in an extensive literature review that examined many different academic sources from a variety of related topics, not limited to detection methods alone, but also the most recent honeypot developments, the remote desktop protocol itself, and the recent honeypot detection countermeasures.

5.1.1 Fingerprinting

The literature surrounding detection methods specifically was critically analysed in section 2.4. It began with an overview of the current research which highlighted the fact there was little research into anti-honeypot techniques, and no research specifically aimed at anti-honeypot techniques for RDP honeypots. However, from the research that did exist, a handful of different techniques could be found. The first technique, and one of the most popular, was various types of fingerprinting. Three different types of fingerprinting were proposed by Uitto et al. in a 2017 survey of anti-honeypot techniques, (Uitto, et al., 2017). These were network, application, and system. Network fingerprinting took the form of using network scanning tools, like Nmap, or network latency testing, like the research conducted by Fu et al. that used round trip time as a distinguishing factor between honeypots and real systems, (Fu, et al., 2006). Also highlighted in the literature review was the use of Nmap to fingerprint honeypots. Dahbul et al. used Nmap to scan various honeypots and found differences in the results between them and the systems they were supposedly simulating, (Dahbul, et al., 2017). This technique proved effective after highlighting suspicious open ports, or just outright flagging the system as a honeypot. Similarly, Mohammadzadeh et al. used Nmap to test their dynamic honeypot systems, using a variety of scans for Nmap, some of which we used in the methodology of this research, (Mohammadzadeh, et al., 2013). These research papers offered tool-based network fingerprinting as an effective honeypot detection method, however, not a perfect technique of their own. Mohammadzadeh et al. did manage to evade Nmap scanning with their dynamic honeypots.

Application fingerprinting was another type of fingerprinting technique discussed by Uitto. This involved examine the applications and services a device offered and comparing them to what was expected e.g., a user desktop with SSH, FTP, TELNET, HTTP, and SQL open would be deemed as suspicious, while a server with SSH, HTTP, and HTTPS would be seen as ordinary. While an important factor to keep in mind for those participating in offensive engagements, it is hard to draw concrete conclusions from these factors alone, however, combined with other techniques, this could be used as evidence, or an early-stage red flag of suspicion.

The final technique proposed by Uitto was system fingerprinting. This was by far the most complex but could likely be the most accurate if performed correctly and could be very applicable to RDP honeypots since it only really works on high interaction honeypots. This

technique involved searching for logging or monitoring tools that had been hooked into the system to allow it to be monitored or maintained. This technique has been demonstrated in research by Holz and Raynal in 2005 in which they managed to detect the monitoring tool Sebek on a honeynet, on Linux, Windows, and OpenBSD systems, (Holz & Raynal, 2005). Within this category of system fingerprinting could also be categorised User-Mode Linux and virtualisation detection. UML was an old technique for virtualising systems and was used for honeypots to separate them from the bare metal system. Holz and Raynal proposed some methods of detecting these types of systems, but UML was not mentioned in any other research analysed in the literature review, therefore, this technique has been largely ignored. Virtualisation detection was another detection technique mentioned by Holz and Raynal and discussed in literature by Issa et al. in which they lay out methods for detecting a virtualised system, (Issa, 2012). Honeypots are almost always run virtually, therefore, being able to label a system as a virtual machine that would not be expected to be virtualised, like a user desktop, could be a giveaway of its nature. None of the techniques were applicable for QEMU/KVM systems, which were used for the RDP honeypots later examined in this research, however, the techniques for VMWare did prove effective for identifying Windows XP and Windows 7 virtual machines. Therefore, system fingerprinting can be used effectively to identify honeypots, particularly being able to detect logging systems connected to the device as this could be an immediate red flag on a honeypot.

Overall, a handful of the fingerprinting methods discussed stood out from the others, namely network fingerprinting and the ability to detect logging systems on the device. Network fingerprinting appeared to be the most straightforward technique and could be applicable to the greatest number of honeypots. Detecting logging systems could also be effective, however, only works on high interaction honeypots and requires a much greater technical ability from the attacker, and a lot of thorough prior research.

5.1.2 Behaviour

Several different papers examined how the behaviour of a system could be analysed to determine whether it is a honeypot. Mukkamala outlined a few key behavioural characteristics that honeypots exhibit compared to normal systems, (Mukkamala, et al., 2007). This included:

1. A lack of network activity on the honeypot
2. All interactions being logged
3. Restricted bandwidth to prevent further attacks from said honeypot

Mukkamala suggested the first attribute was nearly impossible to use as a reliable metric due to the requirement of long-term monitoring and a high chance of false positives. The other two attributes are exclusive to honeypots and, if observed, could be an instant giveaway of a honeypot. Other behaviours were suggested by Tsikerdekis et al. in 2018, (Tsikerdekis, et al., 2018). The first of these techniques were automatic deployments. Many honeypots reset themselves to a default state after being attacked. Therefore, if an attacker connects to a machine, creates a file, disconnects, and reconnects after an amount of time, and the file is gone, this could be a sign the system had been reset, and would be an immediate giveaway of honeypot behaviour. Tsikerdekis also discussed dynamic intelligence. Some advanced honeypots will use AI to adapt to the user and further lure or convince them by changing file names. This type of behaviour could be implemented into an RDP honeypot and could be another obvious honeypot behaviour if a reliable technique was found to exploit it. The final

behavioural technique examined in the literature review was proposed by Uitto. This technique, which Uitto named Operation Analysis, involved attempting to further an attack from a honeypot to an external system. As mentioned by Mukkamala, honeypots should not be able to perform further attacks, therefore, if a system was unable to connect to other external systems for no obvious reason, it could be concluded it had been limited in its network ability due to its nature.

Overall, a handful of different behavioural techniques were suggested, some more effective than others. Out of all the techniques, automatic redeployment is likely the most reliable as of writing and is one of the most straight forward. If a way of detecting dynamic intelligence reliable could be ascertained then that may become more reliable, but this is yet to be seen.

5.1.3 Latency

While it could be categorised under network fingerprinting, there is a significant amount of research done into latency testing against honeypots and, therefore, warrants its own category of technique. Latency testing takes advantage of the virtualised nature of honeypots, concluding that a virtualised system, often with additional logging factors, would take longer to respond to a network request than an ordinary system. This technique was initially highlighted by Holz and Raynal in 2005 but is still relevant today, (Holz & Raynal, 2005). Tsikerdekis et al. also mention latency testing as a viable technique in 2018, (Tsikerdekis, et al., 2018). Actual testing of this method was undertaken in 2006 by Fu et al. against Honeyd systems using ICMP, TCP, and UDP based pings, (Fu, et al., 2006). They used the round-trip time to calculate a two-way link latency which could be used with their model to determine if a system was legitimate or a Honeyd device. This technique proved effective in their testing, however, required a lot of data gathering as a baseline to begin with and was therefore not quick to perform. However, the fundamentals of the technique are still visible and effective in certain scenarios.

5.1.4 Default Configurations

The final notable technique discussed in the literature review was default configurations. A very successful way of identifying honeypots was demonstrated by Morishita et al. in 2019 by creating fingerprints of default honeypots configurations and scanning the web for matches, (Morishita, et al., 2019). This research was done against 14 different SSH, TELNET, and HTTP honeypots and successfully identified over 19,000 honeypots. This technique proved to be extremely effective at identifying poorly deployed honeypots and could further be automated into a useful tool. It falls short due to not being able to identify honeypots that have had changes made to them, or systems that have not been fingerprinted yet.

5.1.5 Conclusion

Many different detection techniques were discussed in the literature review, under the general categories of fingerprinting, behaviour, and latency. Many of these techniques proved to be viable from the literature examined. The most stand out techniques were network fingerprinting, due to its simplicity, speed, and effectiveness, as well as some of the behavioural aspects like automatic redeployment since it appears to be a reliable and repeatable test, although falls short due to its time-consuming nature and suspicious activity carried out by the attacker. Latency testing also proved effective, although did require a lot of prior work in order to set an expected baseline that could be used for comparisons. Default configurations were very successful for poorly deployed honeypots but cannot be a reliable method in an offensive engagement since it only takes slight configuration changes to identify.

5.2 Most effective RDP honeypot detection method

The second research question asked which of the detection methods discussed would be most effective in a purple team engagement. This question was answered by designing a methodology to test several techniques against real RDP honeypots and legitimate services in a simulated offensive engagement environment, as mentioned in the second objective. Three categories of tests were decided upon, fingerprinting, latency, and behaviour. Within these, two tests were designed under each: Nmap and Xprobe2 scanning for fingerprinting, ICMP and TCP packet-based pings for latency, and object permanence and further attacks tests for behavioural analysis.

The Nmap test consisted of two types of scans, a quick scan, and an intense scan. This was done to determine if a more intense scan would be more effective at identifying honeypots. Similarly, two Xprobe2 scans were used, a default scan and a port based one. Using multiple tools would provide more interesting results since one tool may perform more effectively than the other, like the work carried out by Mohammadzadeh, (Mohammadzadeh, et al., 2013).

Two different latency tests were performed to identify any differences in effectiveness between ICMP and TCP packets, like the tests carried out by Fu et al., (Fu, et al., 2006). Pings were sent 100 times to each device and standard deviation used to standardise the results to account for any latency spikes.

Finally, the behavioural tests were based off the research by Tsikerdekis around automatic redeployment, (Tsikerdekis, et al., 2018). This was the basis for the file persistence tests that consisted of creating a file on the system, disconnecting, and reconnecting after 10 minutes to see if the file had been edited. The further attack test was done by attempting to connect to a SSH or FTP server as means of simulating a continued attack, as inspired by Mukkamala's statement that honeypots should not be able to further an attack, (Mukkamala, et al., 2007).

Each set of tests were carried out against three RDP honeypots and three legitimate RDP service, except the behaviour tests which were only tested against the honeypot systems.

These experiments were then executed, and the results were recorded, to meet objective three. The raw results were processed into graphs and tables to be presented in section 4. The results were to be evaluated based off three factors that would make them effective for a purple team engagement: accuracy, speed, and discretion. In the following three sections, the results from the three categories of experiments will be discussed and compared with the literature discussed in section 2 of this report. The results will be evaluated using the proposed metrics above and a conclusion drawn about the effectiveness of each, thus meeting the first half of object four.

5.2.1 Fingerprinting

The results show that fingerprinting is a viable method for identifying certain RDP honeypots, specifically AHA and rdppot which were emulating a Windows system. Since they were being hosted on a Linux system, this could be identified by both Nmap and Xprobe2, thus hinting heavily that they were honeypots. This could be explained by a legitimate Linux system being used to host legitimate Windows virtual machines; however, this would be uncommon behaviour to see widely across an enterprise, especially for basic user machines. Comparing the two tools, Nmap and Xprobe2, the later managed to identify each system correctly using a fraction of the network packets than Nmap. It also ran the scans significantly faster than Nmap. While Nmap did produce more results, showing open ports and an OS fingerprint, as well as attempting to enumerate services, these details were not particularly relevant to determining the nature on the system within the given parameters of these experiments.

Of the two different Xprobe2 scans, there was not a significant difference between the findings of the two. Therefore, the first scan was more effective due to the smaller amount of traffic generated. This scan was fast, quiet, and accurate compared to the other three and would be best suited to be used by a purple team member attempting to evaluate the nature of a system. Compared to a similar fingerprint test by Dahbul, which only used Nmap to probe devices, this experiment tested two different network fingerprinting tools and found varying levels of success with both, (Dahbul, et al., 2017). This test also used multiple types of scans, like the research carried out by Mohammadzadeh, however, not as many, choosing only to do a quick scan and an intense scan, (Mohammadzadeh, et al., 2013). The results found in this experiment were in line with Mohammadzadeh results of Nmap scans. His research concluded Nmap was effective at identifying operating systems but could be loud and intrusive. He also noted some of Nmap's short comings like its failure to obtain accurate results of systems behind firewalls that were not tested in this experiment.

5.2.2 Latency

The findings from the latency experiments were unexpected compared to prior research, which found that honeypots have a longer RTT than the service they imitate, (Fu, et al., 2006). The ICMP and TCP scans showed differing timings between the two for the RTT towards the honeypots. The ICMP results showed no real pattern, with some services taking longer to respond than their honeypot counterpart and vice versa. The TCP tests did show the honeypots taking longer on average than their honeypot counterpart, except for Octopus and XRDP which showed XRDP as taking longer on average. Each ICMP test took around 100 seconds, and the average TCP scan took around 111 seconds, making neither of them faster than the majority of the Nmap or Xprobe2 scans. Each latency test sent less packets than any Nmap scan but more than either of the Xprobe2 scans. Therefore, due to the low accuracy, and overhead of first requiring a real system to create a benchmark, combined with the length of time and noise, latency testing is the weakest of the three.

Examining the ICMP results, these were unexpected compared to the literature reviewed. One of the most prominent pieces of research carried out in this area by Fu et al. found a consistent difference between their honeypots and legitimate services in terms of ICMP and TCP latency, however, the testing carried out in this report found inconsistent behaviour for ICMP testing, with more consistent behaviour for the TCP pings, (Fu, et al., 2006). Surprisingly, many of the honeypots responded quicker to ICMP pings than their legitimate counterpart, like AHA and Octopus. The TCP tests were more in line with Fu's research, with both AHA and rdppot taking longer to respond, however, octopus still responded faster than its counterpart XRDP.

5.2.3 Behaviour

The behaviour tests highlighted an area of weakness for only one honeypot, rdppot. The failure to behave as a normal service could be an immediate giveaway of the nature of the device, more so than things like fingerprinting where it could be viable to have a Windows virtual machine running on a Linux server without it being a honeypot. However, this was the only honeypot to fail the file persistence test. Both AHA and Octopus did not tamper with the file after the ten-minute period. It is unclear how these two-services handle cleaning themselves post-attack. All three honeypots allowed further attacks; however, it could be argued that segmentation of the honeypot is the role of the administrator deploying the devices and not a task the honeypots themselves should be responsible for. For this reason, it is unclear its further attack simulation is a viable method for detecting honeypots in real world engagements, however, due to the fact an organisation could be held responsible for breaking the law if their

honeypot was used to carry out further attacks, it can be assumed that most administrators deploying honeypots would properly segment them from the rest of the environment, especially the internet. Therefore, while this technique appears useless in these experiments, on paper it should still be noted as having great potential in a real enterprise environment.

These behavioural tests were based off literature examined within the literature review. The file persistence test was based off Tsikerdekis analysis of automatic redeployment, (Tsikerdekis, et al., 2018). Their statement that honeypots relied on automatic redeployment to easily maintain themselves did prove true for one of the honeypots, rdppot, which appeared to reset itself back to a default state in the time between disconnecting and reconnecting. This behaviour was not exhibited by any of the other honeypots examined. Furthermore, the further attack test, based on research by both Uitto and Mukkamala did not appear to be inherently true of any of the honeypots, (Uitto, et al., 2017) (Mukkamala, et al., 2007). This shows there was no inbuilt mechanism to limit the honeypots behaviour, however, it could be argued that the task of limiting the honeypots network capability lay with the party deploying the honeypot, not the honeypot itself. Therefore, this study was unable to determine if this behavioural metric would be viable in a purple team engagement.

5.2.4 Conclusion

By analysing the results of the experiments undertaken in this report and comparing them with the literature examined earlier, a conclusion can be drawn around which detection technique is most effective in a purple team engagement. From the data obtained, Xprobe2 default scans were the most accurate, fastest, quietest, and thus the most effective at identifying honeypots by revealing their underlying operating system. Nmap scans were also effective, and by comparing the results of the quick and intense scans, discrepancies could be seen regarding operating systems. However, Nmap was much louder and slower than Xprobe2 for only slightly more verbose results. Both tools could be used in combination in a purple team exercise to build a confident case of the nature of a device. These results were in line with what was expected based on the literature discussed above. Latency testing appears infeasible outside of a simulated lab environment. The inconsistency in round trip time in a live network would ruin any chance of accurately evaluating a machine in a short time frame. This was in stark contrast to Fu et al. who found the technique to be reliable and effective, (Fu, et al., 2006). To truly evaluate the potential of behavioural analysis, a more realistic testing environment is required, however, exploiting automatic redeployment was an accurate way to identify a honeypot, even if it was slow. There was a lack of literature around testing these attributes in simulated or real world scenarios, therefore, it is challenging to compare these results to other research, however, some of the behaviour exhibited by rdppot lined up with Tsikerdekis evaluation of automatic redeployment, while the rest of the honeypots did not, (Tsikerdekis, et al., 2018).

5.3 Technique, Honeypot, and Network Improvements

The final research question asks what improvements could be made to the detection method discussed, the RDP honeypots themselves, and the networks they are deployed into to improve their effectiveness. This question will be answered by meeting the second half of objective four.

5.3.1 Detection Method

Improvements could be made to the network fingerprinting in a number of ways. Firstly, only two network fingerprinting tools were used, Nmap and Xprobe2. Tools like NetScan Tools Pro or SinFP3 could also be tested, as were used by Mohammadzadeh et al. in their research,

(Mohammadzadeh, et al., 2013). Specifically, to the tools used, further research could be done using more types of Nmap scans to fine tune what the most effective scan would be to identify an RDP honeypot. Only two types of Nmap scans were used - quick and intense - but there are a variety of Nmap scans and scripts that could be experimented with to find the most effective results. Mohammadzadeh et al. make note of seven different scans used in their research, but even more scans could be designed and tested, (Mohammadzadeh, et al., 2013).

It is unlikely anything can be done to improve latency testing to the point where it would be suitable in a purple team engagement. If a method of obtaining a consistent round trip time that is not marred by unpredictable factors, then perhaps this method may be viable.

Regarding behavioural tests, analysis of automatic redeployment could be automated into a tool to carry it out more conveniently. This tool could be configurable for the actions it takes once connecting and how long it takes to reconnect. In general, the behavioural techniques would likely implement from machine learning. This could be used to detect unusual behaviour and thus detect a honeypot.

5.3.2 RDP Honeypot

This report has inadvertently highlighted the lack of quality solutions within the RDP honeypot space. All three honeypots were easily identifiable via fingerprinting, with seemingly a poor attempt made to hide this fact.

There is no standout service within open-source RDP honeypots. AHA is the strongest contender but could greatly benefit from more effective countermeasures.

Existing techniques like fuzzing could be implemented into these various RDP honeypots to prevent fingerprinting attacks and better mimic their legitimate counterparts, as shown in research by Naik et al. who found fuzzing to be an effective technique to disguise a honeypot, (Naik, et al., 2018). This would require significant further study and development on one of the three open-source projects.

The most effective countermeasure to the techniques discussed in this report would be either using, or spoofing, Windows as the underlying operating system the honeypot is deployed on. Xprobe2 was able to identify the honeypots as running on Linux, a suspicious factor that could lead to them being identified as honeypots. An RDP honeypot could be developed to run on Windows, making use of Microsoft's Hyper-V virtual machines instead of KVMs, (Microsoft, 2022).

Like rdppot, pooling could be used by other honeypots to improve their capabilities. Rdppot sets up a pool of virtual machines that are deployed as attackers connect, allowing multiple attackers to be logged at once, (Kryptos Login, 2019). This could be taken a step further by reassigning an attacker the same virtual machine if they connect within a given time window. This would defeat the automatic redeployment behaviour while still allowing other attackers to be logged on other virtual machines.

5.3.3 Networking

There are a handful of steps that could be taken by network administrators when setting up an RDP honeypot in their network. Research published in 2006 by Xuxian Jiang et al. found honeynets to be an effective next step to the singular honeypot, (Jiang, et al., 2006). Their research included the design and implementation of VM based honey farm architecture which, when tested in the real world, proved to be an effective way of capturing and detecting attacks. The advantage of using a honeynet rather than a singular honeypot is that it can simulate a more realistic network to convince an attacker that the system they have accessed is legitimate. It

can also allow for more information to be gathered about an attacker, regarding motive or goal, if there is an opportunity for lateral or horizontal movement within the network.

It is important that honeypots are deployed in parts of the network that isolate them from any production systems. This way they cannot be used to perform traversal into other parts of the network once accessed. Moreover, virtualising honeypots, or deploying them within containers, will also increase the security of the network. While most of the honeypots mentioned in this paper deploy virtual machines of their own in the form of KVM/QEMU VMs, the added layer of security builds a defence-in-depth model that prevents the network getting attacked even if an escape is found for KVM/QEMU virtual machines.

5.4 Future Work

This final section of discussion will present future avenues of research that could build on what has been presented in this report, including other fingerprinting techniques, counter measures, and other angles of research.

This study could be repeated using different tools, specifically for the network fingerprinting tests. Only Nmap and Xprobe2 were used for testing which only allowed the performance of those two tools to be evaluated. Other network fingerprinting tools, like Ettercap and p0f could also be tested in a similar manner to determine if they could be more effective ways of labelling honeypots, (Ettercap, 2022) (Zalewski, 2014). The primary difference these two tools present over Nmap and Xprobe2 is that they engage in passive fingerprinting, compared to Nmap and Xprobe2's active fingerprinting. Research conducted into passive fingerprinting found it to be an effective technique for identifying access points in an offensive scenario while also being less intrusive, therefore, this could also be a viable technique for identifying honeypots, (Gao, et al., 2010).

A further study could analyse the fingerprinting tools used in more details. Nmap has a plethora of scanning options, however, this research only used two different types of scans. Many other types of scans were noted in the literature review, specifically presented by Mohammadzadeh et al. in their research, (Mohammadzadeh, et al., 2013). The other scans presented by them could also be tested to determine if they are significantly more or less effective than the two scans used in this research. Furthermore, the authors also highlighted one of Nmap's main weaknesses; an inability to work past firewalls. For this reason, future work could look to repeat the methodology in this report with a more realistic, enterprise network environment that uses firewalls or an IDS/IPS solution. This has also been highlighted in research published in 2016 which found Nmap to give mixed results when identifying systems behind firewalls, (Im, et al., 2016).

Further work needs to be done to establish whether the countermeasures posed in this research would be truly effective against the techniques outlined. Implementing a fuzzing system, as proved to be effective by Naik et al., could be an interesting area of research to identify whether this technique could also effectively be applied to honeypots, (Naik, et al., 2018). Moreover, building a new RDP honeypot from the ground up, with these detection techniques in mind, could also produce interesting findings as to whether it is possible to develop an RDP honeypot that is immune to these detection techniques. This could include better OS spoofing if the honeypots are Linux based or delaying Nmap scan times to be more in line with their legitimate counterparts. Further research could also explore building a honeypot natively on Windows to tackle the operating system identification issue.

Further research is also required to determine the effectiveness of techniques that were not included in this research but were highlighted in the literature review, mainly logging and virtual machine detection. Both of these areas of research were highlighted in research by Holz

and Raynal but were unable to be performed in this research due to lack of technical expertise, (Holz & Raynal, 2005). Furthermore, there was no research found on detect KVM/QEMU virtual machines, which were the primary virtual machine used to deploy systems by the honeypots that did so. Finding techniques that could effectively detect anomalous logging systems could be an accurate way of identifying honeypots.

6. Conclusions

The main goal of this study was to identify and evaluate the most effective detection methods for purple team engagements against contemporary RDP honeypots. For a technique to be effective, it would have to be fast, accurate, and quiet with regards to network traffic. This report began by outlining this aim along with a discussion around the high number of attacks against RDP services, especially over the last few years. The scope and structure of the research was also outlined here.

This section was followed by a thorough literature review which looked at several topics under the umbrella of RDP honeypots, including an overview of honeypots and their most recent developments, the RDP service itself, existing honeypot detection methodologies used against other services like SSH and TELNET, and finally, potential countermeasures to these techniques. The detection methods discussed included multiple types of fingerprinting attacks, behavioural analysis, virtualisation detection, latency disparities, and default configurations.

Information gathered from the literature review was used as a basis to create the third section of this study, the methodology. This section outlined the design and implementation of a network of three honeypots and three comparable legitimate services, including the honeypots and services to be used, the design of the virtual network and virtual machines, and instructions as how to deploy the services before testing. This section also laid out the experiments that would be run against each service, backing each one up with corresponding literature discussed in the prior section. This section also discussed how the results of the experiments would be evaluated.

After laying out the methodology, the experiments were executed, and the results were recorded for each experiment against each service. Results were recorded using screenshots of scans, Wireshark traffic analysis and bespoke scripts to automate the running and processing of results for latency scans. Full, raw results were recorded within the appendices with the most prevalent results displayed within this section, along with tables used to summarise things like timings and packet counts.

The results were followed by the discussion section, where the results were analysed, and the three techniques were evaluated in depth against the three characteristics of an effective detection method, speed, accuracy, and noise. This section also discussed countermeasures for the outlined techniques as well as potential future work based off the finding of this research.

This study has identified three techniques for identifying honeypots, all to various levels of effectiveness. Of the three techniques discussed, fingerprinting using the basic Xprobe2 scan would be the most effective for someone engaging in a purple team exercise attempting to evaluate the nature of a device. It was very accurate, identifying the underlying operating system nearly perfectly. It was fast, completing the scan in 10 seconds on average and it was also the quietest, generating only 11 packets on average. The other fingerprinting techniques were also accurate and quick compared to other types of network scans, however, were either slower or louder than the basic Xprobe2 scan. The latency tests, while could be used to distinguish a real device from a honeypot, relied on the attacker first knowing for sure that a device was legitimate to use as a benchmark. Furthermore, in a real-world exercise, there would be a lot of uncontrollable and unknowable variables like distance between machines, currently flowing network traffic, bandwidth bottlenecks and network configurations, that make this technique unviable in a real-world setting. Finally, the behaviour tests were effective to some extent against only rdppot, where the file was removed after disconnecting. From the research, this appears to be one of the weakest experiments. However, it is likely that if these honeypots were deployed in the real world, more care would be taken to segregate them from the rest of

the network. Therefore, the further attack test, while completely unsuccessful in this research, may be a far more effective technique in the real world. For an organisation to deploy a honeypot and not take proper steps to mitigate the damage it could do would be irresponsible. However, from the research carried out in this report, in the given test environment, fingerprinting with Xprobe2 was the most effective method for a purple team to use to identify RDP honeypots.

In general, therefore, it seems that RDP honeypot detection is still an under researched topic. All three honeypots had glaring weaknesses that need to be addressed if these honeypots hope to be at all effective solutions. These findings will be of interest to any persons who develop RDP honeypots, or those interested in making use of them. Primarily, these finding will be of interest of those engaging in purple teaming exercises who are attempting to distinguish real and fake systems.

One of the major limitations of this study was the behavioural tests. Due to the simulated nature of the design, it lacked the real-world implantation of a honeypot in an existing enterprise network. This heavily influenced the results of those experiments. A better testing ground would be having the honeypot deployed in a real environment to truly examine its behaviour. Notwithstanding these limitations, the study suggests that the behaviour of these RDP honeypots is primarily dependents on the way they are implemented. All three lacked any of their own boundaries when it came to further attacks.

This area of research could be continued in two directions. The first would be to research and develop effective countermeasures that could be implemented on top of, or part of, existing RDP honeypots. Alternatively, since there is no standout RDP honeypot solution, effort could be put into developing an effective, dynamic RDP honeypot from the ground up, with the techniques in this paper kept in mind during design and development.

There is, therefore, a definite need for further research into RDP honeypot detection, as well as further development into opensource RDP honeypots that are simply to use while being highly customisable, versatile, and inconspicuous.

7. Bibliography

- A., G., 2021. *octopus*. [Online]
Available at: <https://github.com/qqeekbox/octopus>
[Accessed 9 April 2022].
- Adrian, D., Durumeric, Z., Singh, G. & Halderman, J. A., 2014. Zippier ZMap: Internet-Wide Scanning at 10 Gbps. *USENIX Workshop on Offensive Technologies*, Volume 6, pp. 1-8.
- binarytrails, 2021. *xprobe2*. [Online]
Available at: <https://github.com/binarytrails/xprobe2>
[Accessed 2 May 2022].
- Boddy, M., Jones, B. & Stockley, M., 2019. *RDP Exposed - The Threat That's Already at Your Door*, s.l.: Sophos.
- CISA, 2019. *Microsoft Operating Systems BlueKeep Vulnerability*. [Online]
Available at: <https://www.cisa.gov/uscert/ncas/alerts/AA19-168A>
[Accessed 16 March 2022].
- CISA, 2020. *COVID-19 Exploited by Malicious Cyber Actors*. [Online]
Available at: <https://www.cisa.gov/uscert/ncas/alerts/aa20-099a>
[Accessed 26 February 2022].
- Dahbul, R., Lim, C. & Purnama, J., 2017. Enhancing Honeypot Deception Capability Through Network Service Fingerprinting. *Journal of Physics: Conference Series*, Volume 801, pp. 1-6.
- Danchenko, N. M., Prokofiev, A. O. & Silnov, D. S., 2017. Detecting suspicious activity on remote desktop protocols using Honeypot system. *Proceedings of the 2017 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference*, pp. 127-128.
- Ettercap, 2022. *Ettercap*. [Online]
Available at: <https://www.ettercap-project.org/index.html>
[Accessed 12 May 2022].
- Fu, X. et al., 2006. On Recognizing Virtual Honeypots and Countermeasures. *IEEE Security & Privacy*, pp. 1-8.
- Gao, K., Corbett, C. & Beyah, R., 2010. A Passive Approach to Wireless Device Fingerprinting. *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pp. 383-392.
- Gartner, 2020. *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 6.3% in 2020*. [Online]
Available at: <https://www.gartner.com/en/newsroom/press-releases/2020-07-23-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-6point3-percent-in-2020>
[Accessed 2 March 2022].
- Google, 2010. *paping*. [Online]
Available at: <https://code.google.com/archive/p/paping/>
[Accessed 2 May 2022].
- Greenberg, A., 2019. *BlueKeep Attacks Arrive, Bearing Cryptomining Malware*. [Online]
Available at: <https://www.bankinfosecurity.com/new-bluekeep-attacks-drop-cryptominer-malware-a-13341>
[Accessed 10 10 2021].
- GreyNoise, 2022. *GreyNoise*. [Online]
Available at: <https://www.greynoise.io/viz/query/?gnql=tags%3A%22RDP%22>
[Accessed 28 March 2022].

HashiCorp, 2022. *Packer*. [Online]
 Available at: <https://www.packer.io/>
 [Accessed 2 May 2022].

HashiCorp, 2022. *Terraform*. [Online]
 Available at: <https://www.terraform.io/>
 [Accessed 2 May 2022].

Holz, T. & Raynal, F., 2005. *Defeating Honeybots: System Issues, Part 2*. [Online]
 Available at: <https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=5d2135d2-1cef-498a-ae10-0c418ce18030&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments>
 [Accessed 28 February 2022].

Holz, T. & Raynal, F., 2005. Detecting Honeybots and other suspicious environments. *IEEE*, pp. 29-36.

Im, S.-y., Shin, S.-H., Ryn, K. Y. & Roh, B.-h., 2016. Performance Evaluation of Network Scanning Tools with Operation of Firewall. *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 876-881.

Installing Windows XP into QEMU/KVM. 2020. [Film] Directed by Kevin Veroneau. United States of America: s.n.

Issa, A., 2012. Anti-virtual machines and emulations. *Journal in Computer Virology*, 8(4), pp. 141-149.

Jiang, X., Xu, D. & Wang, Y.-M., 2006. Collapsar: A VM-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention. *Journal of Parallel and Distributed Computing*, 66(9), pp. 1165-1180.

Kali, 2022. *Kali Linux 2022.1 Release*. [Online]
 Available at: <https://www.kali.org/blog/kali-linux-2022-1-release/>
 [Accessed 2 May 2022].

Kandakatla, D. K., 2022. *Windows 10, version 21H2*. [Online]
 Available at: <https://docs.microsoft.com/en-us/windows/release-health/status-windows-10-21h2>
 [Accessed 2 May 2022].

Krawetz, N., 2004. Anti-Honeybot Technology NEAL. *The Honeybot Files*, pp. 76-79.

Kryptos Login, 2019. *rdppot*. [Online]
 Available at: <https://github.com/kryptoslogic/rdppot>
 [Accessed 28 March 2022].

Liang, H., Chen, L. & Xu, S., 2021. *Understanding the Remote Desktop Protocol (RDP)*. [Online]
 Available at: <https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol>
 [Accessed 19 October 2021].

Liston, T., 2001. *LaBrea: "Sticky" Honeybot and IDS*. [Online]
 Available at: <https://labrea.sourceforge.io/labrea-info.html>
 [Accessed 17 March 2022].

Livshitz, I., 2019. *What's the Difference Between a High Interaction Honeybot and a Low Interaction Honeybot?*. [Online]
 Available at: <https://www.guardicore.com/blog/high-interaction-honeybot-versus-low-interaction-honeybot-comparison/>
 [Accessed 4 October 2021].

Lockheed Martin, 2022. *Cyber Kill Chain*. [Online]
 Available at: <https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/photo/cyber/THE-CYBER-KILL-CHAIN-body.png.pc-adaptive.1920.medium.png>
 [Accessed 17 March 2022].

Lockheed Martin, 2022. *The Cyber Kill Chain*. [Online]
 Available at: <https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html>
 [Accessed 16 March 2022].

Lyon, G., 2022. *Nmap*. [Online]
 Available at: <https://nmap.org/>
 [Accessed 2 May 2022].

MaxMind, 2022. *GeoLite2 Free Geolocation Data*. [Online]
 Available at: <https://dev.maxmind.com/geoip/geolite2-free-geolocation-data?lang=en>
 [Accessed 14 May 2022].

Microsoft, 2022. *Introduction to Hyper-V on Windows 10*. [Online]
 Available at: <https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/>
 [Accessed 29 April 2022].

Miessler, D., 2022. *The Definition of a Purple Team*. [Online]
 Available at: <https://danielmiessler.com/study/purple-team/#:~:text=A%20Purple%20Team%20is%20a,organization's%20Red%20and%20Blue%20teams.&text=The%20Purple%20Team%20should%20not,existing%20Red%20and%20Blue%20capabilities.>
 [Accessed 7 February 2022].

Mohammadzadeh, H., Mansoori, M. & Welch, I., 2013. Evaluation of Fingerprinting Techniques and a Windows-based Dynamic Honeypot. *Conferences in Research and Practice in Information Technology Series*, Volume 138, pp. 59-66.

Morishita, S. et al., 2019. Detect me if you... Oh wait. An internet-wide view of self-revealing honeypots. *FIP/IEEE Symposium on Integrated Network and Service Management*, pp. 134-143.

Morrison, J. & van den Brekel, J., 2020. *PURPLE TEAM: DRIVE DEFENSE WITH OFFENSE*. [Online]
 Available at: <https://www.compact.nl/en/articles/purple-team-drive-defense-with-offense/>
 [Accessed 26 February 2022].

Mukkamala, S. et al., 2007. Detection of Virtual Environments and Low Interaction Honeypots. *EEE SMC Information Assurance and Security Workshop*, Volume 2007, pp. 92-98.

Naik, N., Jenkins, P., Cooke, R. & Yang, L., 2018. Honeypots That Bite Back: A Fuzzy Technique for Identifying and Inhibiting Fingerprinting Attacks on Low Interaction Honeypots Nitin. *IEEE International Conference on Fuzzy Systems*, Volume 2018-July, pp. 1-8.

Naik, N., Shang, C., Jenkins, P. & Shen, Q., 2021. D-FRI-Honeypot: A Secure Sting Operation for Hacking the Hackers Using Dynamic Fuzzy Rule Interpolation. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(6), pp. 893-907.

Nawrocki, M. et al., 2016. A Survey on Honeypot Software and Data Analysis. pp. 1-38.

NVD, 2021. *CVE-2019-0708 Detail*. [Online]
 Available at: <https://nvd.nist.gov/vuln/detail/cve-2019-0708>
 [Accessed 26 January 2022].

qeeqbox, 2021. *octopus*. [Online]
Available at: <https://github.com/qeeqbox/octopus>
[Accessed 2 May 2022].

Rapid7, 2022. *What is a honeypot?*. [Online]
Available at: <https://www.rapid7.com/fundamentals/honeypots/>
[Accessed 26 February 2022].

RedHat, 2022. *Ansible*. [Online]
Available at: <https://www.ansible.com/>
[Accessed 2 May 2022].

Savill, J., 2002. *How can I enable autologon for Windows XP?*. [Online]
Available at: <https://www.itprotoday.com/cloud-computing/how-can-i-enable-autologon-windows-xp>
[Accessed 14 May 2022].

Shing, L., 2016. *AN IMPROVED TARPIT FOR NETWORK DECEPTION*, California: Naval Postgraduate School.

Shodan, 2022. *product:"Remote Desktop Protocol"*. [Online]
Available at:
<https://www.shodan.io/search?query=product%3A%22Remote+Desktop+Protocol%22>
[Accessed 16 March 2022].

Sorg, J., 2022. *XRDP*. [Online]
Available at: <http://xrdp.org/>
[Accessed 2 May 2022].

Spitzner, L., 2002. *Medium-Interaction Honeypots*. [Online]
Available at: https://www.oreilly.com/library/view/honeypots-tracking-hackers/0321108957/0321108957_ch05lev1sec3.html
[Accessed 11 October 2021].

Srinivasa, S., Pedersen, J. M. & Vasilomanolakis, E., 2021. Gotta catch 'em all: a Multistage Framework for honeypot fingerprinting. *ArTix*, Volume 7, pp. 1-26.

Sullivan-Hasson, E., 2020. *Remote Desktop Software Statistics and Trends*. [Online]
Available at: <https://www.trustradius.com/vendor-blog/remote-desktop-buyer-statistics-and-trends>
[Accessed 30 September 2021].

tothi, 2020. [Online]
Available at: <https://github.com/tothi/ad-honeypot-autodeploy>
[Accessed 28 March 2022].

tothi, 2022. *ad-honeypot-autodeploy*. [Online]
Available at: <https://github.com/tothi/ad-honeypot-autodeploy>
[Accessed 2 May 2022].

Tsikerdekis, M., Zeadally, S., Schlesener, A. & Sklavos, N., 2018. Approaches for Preventing Honeypot Detection and Compromise. *Global Information Infrastructure and Networking Symposium*, pp. 1-6.

Ubuntu, 2022. *Ubuntu 20.04.4 LTS (Focal Fossa)*. [Online]
Available at: <https://releases.ubuntu.com/20.04/>
[Accessed 2 May 2022].

Uitto, J., Rauti, S., Lauren, S. & Leppanen, V., 2017. A Survey on Anti-honeypot and Anti-introspection Methods. *Joni. A Survey on Anti-honeypot and Anti-introspection Methods Joni*, Volume 570, pp. 125-134.

User Mode Linux Core Team, 2020. *User Mode Linux*. [Online]
Available at: https://www.kernel.org/doc/html/v5.9/virt/uml/user_mode_linux.html
[Accessed 1 February 2022].

Vetterl, A. & Clayton, R., 2018. Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale.

VMWare, 2022. *VMware Workstation Pro*. [Online]
Available at: <https://www.vmware.com/uk/products/workstation-pro.html>
[Accessed 2 May 2022].

Zalewski, M., 2014. *p0f v3*. [Online]
Available at: <https://lcamtuf.coredump.cx/p0f3/>
[Accessed 12 May 2022].

Zhang, L. & Thing, V., 2021. Three decades of deception techniques in active cyber defense - Retrospect and outlook. *Elsevier*, Volume 106, pp. 1-22.

8. Appendix

8.1 Appendix A - Risk Analysis

Risk	Likelihood (L)	Consequence (C)	Impact (L*C)	Remediation	Type	Class
Unable to run demanding VMs on existing hardware.	2	3	6	Use university resources to run more demanding virtual machines.	Technical	Acute
Losing Data	2	5	10	All data will be backed up to the cloud (Google Drive) and a local hard drive.	Technical	Acute
Outdated Software	2	3	6	Replacements will be searched out, or alternative virtual machines set up to support older technologies.	Technical	Chronic
Personal sickness	1	4	4	Project work will be spread as evenly as possible over the given time period meaning any short-term illness should not lead to a drastic increase in work.	Non-technical	Chronic
Time Management Failures	2	5	10	The project has been planned out in a Gantt chart on a week-by-week basis. As long as each week's tasks are completed the project will stay on track.	Non-Technical	Chronic
Estimation Errors	2	5	10	Extra time has been left at the end of the project plan before the deadline to compensate for any estimation errors.	Non-Technical	Chronic
Another COVID-19 Lockdown	1	3	3	The project could be fully completed outside of the university with only a handful of issues.	Non-Technical	Chronic

8.2 Appendix B - Ethics Report



A: Applicant details	
Your Name	Christopher Di-Nozzi
Student/Staff Number	1800317
Abertay email address	1800317@abertay.ac.uk
Name Of Programme (if applicable)	Evaluating and Identifying Effective Detection Methods for Purple Team Engagements Against RDP Honeypots
Module code	CMP400
School	School of Design and Informatics (SDI) <input type="text"/>
	<small>Required</small>
Is this a revised re-submission?	Yes <input type="text"/>
	<small>Required, feedback is available in the 'View Notes' section</small>
Supervisor email address	j.o'hare@abertay.ac.uk

A2: Resubmission details	
Revision details	Modified section F to emphasize the isolation and ownership of the virtual network.

B: Project details	
Project title	Evaluating and Identifying Effective Detection Methods for Purple Team Engagements Against RDP Honeypots



8.3 Appendix C - TCP and ICMP latency scripts

8.3.1 TCP

```
#!/bin/bash
```

```
#requires the paping binary in the same directory this script is run
```

```
output_file=$1_TCP_$2.csv
```

```

echo "Sending $2 TCP packets to $1"
./paping $1 -p 3389 -c $2 --nocolor > output.txt
echo "grep-ing timings..."
grep -Po "time=\K.{0,4}" output.txt > output_c.txt
artt=$(grep -Po "Average = \K.*" output.txt)
echo "average rtt = $artt"
echo "deleting old output file..."
rm output.txt
c=1
while IFS="" read -r l || [ -n "$l" ]
do
    printf '%s\n' "$c, $l" >> $output_file
    c=$((c+1))
done < output_c.txt
rm output_c.txt
echo "average, $artt" >> $output_file
echo "Done"

```

8.3.2 ICMP

```

#!/bin/bash
output_file=$1_ICMP_$2.csv
echo "Sending $2 ICMP packets to $1"
ping -c $2 $1 > output.txt
echo "grep-ing timings..."
grep -Po "time=\K.*" output.txt > output_c.txt
artt=$(cat output.txt | tail -1 | awk '{print $4}' | cut -d '/' -f 2)
echo "average rtt = $artt"
echo "deleting old output file..."
rm output.txt
c=1
while IFS="" read -r l || [ -n "$l" ]
do
    printf '%s\n' "$c, $l" >> $output_file
    c=$((c+1))
done < output_c.txt
rm output_c.txt
echo "average, $artt" >> $output_file
echo "Done"

```

8.4 Appendix E - PCAPNG Cleaning Script

```

#!/bin/bash

FILE_COUNT=$(ls *.pcapng | wc -l)
FILTER="ip.addr==192.168.10.139&&ip.addr!=192.168.10.1"
echo "[*]Cleaning $FILE_COUNT pcapng files for the filter $FILTER"
for f in *.pcapng
do
    echo "[+]Cleaning $f ..."

```

```

    /Applications/Wireshark.app/Contents/MacOS/tshark -r $f -Y $FILTER -w
clean_{$f}
    echo "[+]Saved clean file as clean_{$f}"
    echo "[-]Deleting $f ..."
    rm $f
done

echo "[*]Finished cleaning $FILE_COUNT files"

```

8.5 Appendix F - NMAP Fingerprinting Results

8.5.1 Rdpptot vs Windows XP - Quick Scan

```

(kali@kali)-[~]
└─$ nmap -A -Pn 192.168.10.131
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:25 EDT
Nmap scan report for 192.168.10.131
Host is up (0.0016s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.43 seconds

```

Figure 48: Result of quick scan against rdpptot.

```

(kali@kali)-[~/scripts]
└─$ nmap -A -Pn 192.168.10.137
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 12:23 EDT
Nmap scan report for 192.168.10.137
Host is up (0.00035s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 26.01 seconds

```

Figure 49: Result of quick scan against Windows XP.

8.5.2 Octopus vs XRDP - Quick Scan

```

(kali@kali)-[~/scripts]
└─$ nmap -A -Pn 192.168.10.134
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 12:12 EDT
Nmap scan report for 192.168.10.134
Host is up (0.00051s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   xrdp
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.42 seconds

```

Figure 50: NMAP quick scan against Octopus.

```

(kali@kali)-[~/scripts]
└─$ nmap -A -Pn 192.168.10.143
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:42 EDT
Nmap scan report for 192.168.10.143
Host is up (0.00068s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE      VERSION
3389/tcp  open  ms-wbt-server xrdp

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.40 seconds

```

Figure 51: NMAP quick scan against XRDP.

8.5.3 AHA vs Windows 10 - Quick Scan

```

(kali@kali)-[~/scripts]
└─$ nmap -A -Pn 192.168.10.132
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:00 EDT
Nmap scan report for 192.168.10.132
Host is up (0.00030s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE      VERSION
3389/tcp  open  ms-wbt-server Microsoft Terminal Services
| ssl-cert: Subject: commonName=desktop12.ecorp.local
| Not valid before: 2022-04-15T14:22:24
|_Not valid after: 2022-10-15T14:22:24
|_ssl-date: 2022-04-16T13:51:07+00:00; -1h09m41s from scanner time.
| rdp-ntlm-info:
|   Target_Name: ECORP
|   NetBIOS_Domain_Name: ECORP
|   NetBIOS_Computer_Name: DESKTOP12
|   DNS_Domain_Name: ecorp.local
|   DNS_Computer_Name: desktop12.ecorp.local
|   DNS_Tree_Name: ecorp.local
|   Product_Version: 10.0.19041
|_ System_Time: 2022-04-16T13:51:07+00:00
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_clock-skew: mean: -1h09m41s, deviation: 0s, median: -1h09m41s

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 20.00 seconds

```

Figure 52: NMAP quick scan results against AHA.


```

(kali㉿kali)-[~/scripts]
└─$ nmap -A -Pn 192.168.10.136
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:58 EDT
Nmap scan report for 192.168.10.136
Host is up (0.00042s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
| rdp-ntlm-info:
|   Target_Name: DESKTOP-GS56T6Q
|   NetBIOS_Domain_Name: DESKTOP-GS56T6Q
|   NetBIOS_Computer_Name: DESKTOP-GS56T6Q
|   DNS_Domain_Name: DESKTOP-GS56T6Q
|   DNS_Computer_Name: DESKTOP-GS56T6Q
|   Product_Version: 10.0.19041
|_  System_Time: 2022-04-16T15:58:52+00:00
|_ssl-date: 2022-04-16T15:58:52+00:00; -1s from scanner time.
| ssl-cert: Subject: commonName=DESKTOP-GS56T6Q
| Not valid before: 2022-03-30T07:52:54
|_Not valid after: 2022-09-29T07:52:54
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 25.93 seconds

```

Figure 53: NMAP quick scan results against Windows 10.

8.5.4 Rdpport vs Windows XP - Intense Scan

```

(kali㉿kali)-[~/]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.131
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:26 EDT
Nmap scan report for 192.168.10.131
Host is up (0.00032s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE          VERSION
3389/tcp  open  tcpwrapped
MAC Address: 00:0C:29:EA:08:DA (VMware)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.47 seconds

```

Figure 54: Results of intense Nmap scan against rdpport.

```

(kali㉿kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.137
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 12:24 EDT
Nmap scan report for 192.168.10.137
Host is up (0.00021s latency).
Not shown: 65534 filtered tcp ports (no-response)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
MAC Address: 00:0C:29:E8:14:73 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running: Microsoft Windows XP
OS CPE: cpe:/o:microsoft:windows_xp::sp3
OS details: Microsoft Windows XP SP3
Network Distance: 1 hop
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 125.75 seconds

```

Figure 55: Results of intense Nmap scan against Windows XP.

8.5.5 Octopus vs XRDP - Intense Scan

```
(kali@kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.143
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:44 EDT
Nmap scan report for 192.168.10.143
Host is up (0.00024s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
3389/tcp  open  ms-wbt-server xrdp
MAC Address: 00:0C:29:16:23:CD (VMware)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 27.36 seconds
```

Figure 56: Results of the intense Nmap scan against Octopus.

```
(kali@kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.134
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 12:13 EDT
Nmap scan report for 192.168.10.134
Host is up (0.00041s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
3389/tcp  open  ms-wbt-server xrdp
MAC Address: 00:0C:29:B4:18:12 (VMware)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 27.09 seconds
```

Figure 57: Results of the intense Nmap scan against XRDP.

8.5.6 AHA vs Windows 10 - Intense Scan

```

(kali㉿kali)-[~]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.132
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:02 EDT
Nmap scan report for 192.168.10.132
Host is up (0.00035s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
MAC Address: 00:0C:29:65:4D:26 (VMware)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.92%E=4%D=4/16%OT=3389%CT=1%CU=37670%PV=Y%DS=1%DC=D%G=Y%M=000C29
OS:%TM=625ADA93P=x86_64-pc-linux-gnu)SEQ(SP=103%GCD=2%ISR=109%TI=I%CI=Z%II
OS:=I%TS=U)SEQ(SP=103%GCD=1%ISR=109%TI=I%CI=Z%II=I%SS=S%TS=U)OPS(O1=M5B4NW0
OS:NNS%O2=M5B4NW0NNS%O3=M5B4NW0%O4=M5B4NW0NNS%O5=M5B4NW0NNS%O6=M5B4NNS)WIN(
OS:W1=FA00%W2=FA00%W3=FA00%W4=FA00%W5=FA00%W6=FA00)ECN(R=Y%DF=Y%T=7F%W=FA00
OS:%O=M5B4NW0NNS%CC=N%Q=)T1(R=Y%DF=Y%T=7F%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=Y
OS:%T=40%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)T3(R=Y%DF=Y%T=40%W=0%S=Z%A=O%F=AR%O=%R
OS:D=0%Q=)T4(R=N)T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%
OS:T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD
OS:=0%Q=)U1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE
OS:(R=Y%DFI=N%T=40%CD=S)

Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 32.59 seconds

```

Figure 58: Results of the intense Nmap scan against AHA.

```

(kali㉿kali)-[~/scripts]
└─$ sudo nmap -Pn -sV -O -p 1-65535 --version-all 192.168.10.136
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-16 11:59 EDT
Nmap scan report for 192.168.10.136
Host is up (0.00029s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT      STATE SERVICE          VERSION
3389/tcp  open  ms-wbt-server   Microsoft Terminal Services
7680/tcp  open  pando-pub?
MAC Address: 00:0C:29:9A:8E:E8 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): Microsoft Windows XP|2008|7 (88%)
OS CPE: cpe:/o:microsoft:windows_xp::sp2 cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_server_2008:r
2 cpe:/o:microsoft:windows_7
Aggressive OS guesses: Microsoft Windows XP SP2 (88%), Microsoft Windows Server 2008 SP1 or Windows Server 2008 R2 (86%),
Microsoft Windows 7 (85%), Microsoft Windows Fundamentals for Legacy PCs (XP Embedded derivative) (85%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 223.51 seconds

```

Figure 59: Results of the intense Nmap scan against Windows 10.

8.6 Appendix G - Xprobe2 Fingerprinting Results

8.6.1 Rdpptot vs Windows XP - Normal

```
(kali@kali)-[~]
└─$ sudo xprobe2 192.168.10.131

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 192.168.10.131
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.10.131. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.131. Module test failed
[-] No distance calculation. 192.168.10.131 appears to be dead or no ports known
[+] Host: 192.168.10.131 is up (Guess probability: 50%)
[+] Target: 192.168.10.131 is alive. Round-Trip Time: 0.49320 sec
[+] Selected safe Round-Trip Time value is: 0.98640 sec
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.25" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.0.36" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.6.9" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.0.30" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.6.11" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.20" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.23" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.22" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.21" (Guess probability: 100%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.24" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 60: Result of Xprobe2 normal scan against rdpptot.

```
(kali@kali)-[~/scripts]
└─$ sudo xprobe2 192.168.10.137
```

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

```
[+] Target is 192.168.10.137
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.10.137. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.137. Module test failed
[-] No distance calculation. 192.168.10.137 appears to be dead or no ports known
[+] Host: 192.168.10.137 is up (Guess probability: 50%)
[+] Target: 192.168.10.137 is alive. Round-Trip Time: 0.47184 sec
[+] Selected safe Round-Trip Time value is: 0.94367 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.137 Running OS: "Microsoft Windows XP" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Workstation SP1" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Workstation SP3" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Workstation SP4" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server Service Pack 1" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server Service Pack 2" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server Service Pack 3" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server Service Pack 4" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 61: Result of Xprobe2 normal scan against Windows 10.

8.6.2 Octopus vs XRDP- Normal

```
(kali@kali)-[~/scripts]
└─$ sudo xprobe2 192.168.10.143
```

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

```
[+] Target is 192.168.10.143
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:ttl_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.10.143. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.143. Module test failed
[-] No distance calculation. 192.168.10.143 appears to be dead or no ports known
[+] Host: 192.168.10.143 is up (Guess probability: 50%)
[+] Target: 192.168.10.143 is alive. Round-Trip Time: 0.50640 sec
[+] Selected safe Round-Trip Time value is: 1.01281 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.30" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.22" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.23" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.21" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.20" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.19" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.24" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.25" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.26" (Guess probability: 100%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.27" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 62: Result of Xprobe2 normal scan against Octopus.

```
(kali@kali)-[~/scripts]
└─$ sudo xprobe2 192.168.10.134
```

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

```
[+] Target is 192.168.10.134
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.10.134. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.134. Module test failed
[-] No distance calculation. 192.168.10.134 appears to be dead or no ports known
[+] Host: 192.168.10.134 is up (Guess probability: 50%)
[+] Target: 192.168.10.134 is alive. Round-Trip Time: 0.51105 sec
[+] Selected safe Round-Trip Time value is: 1.02210 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.30" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.22" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.23" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.21" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.20" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.19" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.24" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.25" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.26" (Guess probability: 100%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.27" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 63: Result of Xprobe2 normal scan against XRDP.

8.6.3 AHA vs Windows 10- Normal

```
(kali㉿kali)-[~]  
└─$ sudo xprobe2 192.168.10.132
```

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

```
[+] Target is 192.168.10.132  
[+] Loading modules.  
[+] Following modules are loaded:  
[x] [1] ping:icmp_ping - ICMP echo discovery module  
[x] [2] ping:tcp_ping - TCP-based ping discovery module  
[x] [3] ping:udp_ping - UDP-based ping discovery module  
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation  
[x] [5] infogather:portscan - TCP and UDP PortScanner  
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module  
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module  
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module  
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module  
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module  
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module  
[x] [12] fingerprint:smb - SMB fingerprinting module  
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module  
[+] 13 modules registered  
[+] Initializing scan engine  
[+] Running scan engine  
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.10.132. Module test failed  
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.132. Module test failed  
[-] No distance calculation. 192.168.10.132 appears to be dead or no ports known  
[+] Host: 192.168.10.132 is up (Guess probability: 50%)  
[+] Target: 192.168.10.132 is alive. Round-Trip Time: 0.48319 sec  
[+] Selected safe Round-Trip Time value is: 0.96638 sec  
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)  
[-] fingerprint:smb need either TCP port 139 or 445 to run  
[-] fingerprint:snmp: need UDP port 161 open  
[+] Primary guess:  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.25" (Guess probability: 100%)  
[+] Other guesses:  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.0.36" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.6.9" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.0.30" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.6.11" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.20" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.23" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.22" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.21" (Guess probability: 100%)  
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.24" (Guess probability: 100%)  
[+] Cleaning up scan engine  
[+] Modules deinitialized  
[+] Execution completed.
```

Figure 64: Result of Xprobe2 normal scan against AHA.


```
(kali@kali)-[~/scripts]
└─$ sudo xprobe2 192.168.10.136
```

```
Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu
```

```
[+] Target is 192.168.10.136
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 192.168.10.136. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.136. Module test failed
[-] No distance calculation. 192.168.10.136 appears to be dead or no ports known
[+] Host: 192.168.10.136 is up (Guess probability: 50%)
[+] Target: 192.168.10.136 is alive. Round-Trip Time: 0.50226 sec
[+] Selected safe Round-Trip Time value is: 1.00451 sec
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2000 Server Service Pack 1" (Guess probability: 91%)
[+] Other guesses:
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2003 Server Enterprise Edition" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2000 Server" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2000 Server Service Pack 4" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows XP SP1" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2003 Server Standard Edition" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2000 Workstation" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2003 Server Standard Edition" (Guess probability: 91%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess probability: 91%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 65: Result of Xprobe2 normal scan against Windows 10.

8.6.4 Rdpptot vs Windows XP - Port

```
(kali㉿kali)-[~]
└─$ sudo xprobe2 -p tcp:3389:open 192.168.10.131

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 192.168.10.131
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPV2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.131. Module test failed
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
[+] Host: 192.168.10.131 is up (Guess probability: 66%)
[+] Target: 192.168.10.131 is alive. Round-Trip Time: 0.51557 sec
[+] Selected safe Round-Trip Time value is: 1.03113 sec
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.19" (Guess probability: 94%)
[+] Other guesses:
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.20" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.21" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.22" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.23" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.24" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.25" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.26" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.27" (Guess probability: 94%)
[+] Host 192.168.10.131 Running OS: "Linux Kernel 2.4.28" (Guess probability: 94%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 66: Result of Xprobe2 port scan against rdpptot.

```

(kali@kali)-[~/scripts]
└─$ sudo xprobe2 -p tcp:3389:open 192.168.10.137

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 192.168.10.137
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.137. Module test failed
[+] Host: 192.168.10.137 is up (Guess probability: 66%)
[+] Target: 192.168.10.137 is alive. Round-Trip Time: 0.51971 sec
[+] Selected safe Round-Trip Time value is: 1.03942 sec
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.137 Running OS: "Microsoft Windows XP" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.168.10.137 Running OS: "Microsoft Windows XP SP1" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2003 Server Standard Edition" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2003 Server Enterprise Edition" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server Service Pack 3" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server Service Pack 4" (Guess probability: 100%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows XP SP2" (Guess probability: 95%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Workstation SP1" (Guess probability: 95%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Workstation SP3" (Guess probability: 95%)
[+] Host 192.168.10.137 Running OS: "Microsoft Windows 2000 Server" (Guess probability: 95%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.

```

Figure 67: Result of Xprobe2 port scan against Windows XP.

8.6.5 Octopus vs XRDP - Port

```
(kali@kali)-[~/scripts]
└─$ sudo xprobe2 -p tcp:3389:open 192.168.10.143
```

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

```
[+] Target is 192.168.10.143
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.143. Module test failed
[+] Host: 192.168.10.143 is up (Guess probability: 66%)
[+] Target: 192.168.10.143 is alive. Round-Trip Time: 0.49883 sec
[+] Selected safe Round-Trip Time value is: 0.99766 sec
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.6.8" (Guess probability: 93%)
[+] Other guesses:
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.6.1" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.21" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.23" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.29" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.25" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.27" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.27" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.25" (Guess probability: 90%)
[+] Host 192.168.10.143 Running OS: "Linux Kernel 2.4.29" (Guess probability: 90%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 68: Result of Xprobe2 port scan against Octopus.

```
(kali@kali)-[~/scripts]
└─$ sudo xprobe2 -p tcp:3389:open 192.168.10.134
```

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

```
[+] Target is 192.168.10.134
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.134. Module test failed
[+] Host: 192.168.10.134 is up (Guess probability: 66%)
[+] Target: 192.168.10.134 is alive. Round-Trip Time: 0.49938 sec
[+] Selected safe Round-Trip Time value is: 0.99875 sec
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.6.8" (Guess probability: 93%)
[+] Other guesses:
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.6.1" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.21" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.23" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.29" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.25" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.27" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.27" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.25" (Guess probability: 90%)
[+] Host 192.168.10.134 Running OS: "Linux Kernel 2.4.29" (Guess probability: 90%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 69: Result of Xprobe2 port scan against XRDP.

8.6.6 AHA vs Windows 10 - Port

```
(kali@kali)-[~]
└─$ sudo xprobe2 -p tcp:3389:open 192.168.10.132
```

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

```
[+] Target is 192.168.10.132
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.132. Module test failed
[+] TTL distance to target: 1 hops
[+] Host: 192.168.10.132 is up (Guess probability: 100%)
[+] Target: 192.168.10.132 is alive. Round-Trip Time: 0.51513 sec
[+] Selected safe Round-Trip Time value is: 1.03026 sec
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.25" (Guess probability: 84%)
[+] Other guesses:
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.28" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.23" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.30" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.21" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.26" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.19" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.24" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.29" (Guess probability: 84%)
[+] Host 192.168.10.132 Running OS: "Linux Kernel 2.4.22" (Guess probability: 84%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Figure 70: Result of Xprobe2 port scan against AHA.

```

(kali@kali)-[~/scripts]
└─$ sudo xprobe2 -p tcp:3389:open 192.168.10.136

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 192.168.10.136
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping - ICMP echo discovery module
[x] [2] ping:tcp_ping - TCP-based ping discovery module
[x] [3] ping:udp_ping - UDP-based ping discovery module
[x] [4] infogather:tll_calc - TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan - TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo - ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp - ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask - ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach - ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake - TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst - TCP RST fingerprinting module
[x] [12] fingerprint:smb - SMB fingerprinting module
[x] [13] fingerprint:snmp - SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:udp_ping module: no closed/open UDP ports known on 192.168.10.136. Module test failed
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
[+] Host: 192.168.10.136 is up (Guess probability: 66%)
[+] Target: 192.168.10.136 is alive. Round-Trip Time: 0.49130 sec
[+] Selected safe Round-Trip Time value is: 0.98260 sec
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Server Service Pack 5" (Guess probability: 77%)
[+] Other guesses:
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Server Service Pack 4" (Guess probability: 77%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Workstation Service Pack 6a" (Guess probability: 77%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Workstation Service Pack 5" (Guess probability: 77%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Workstation Service Pack 4" (Guess probability: 77%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Server Service Pack 6a" (Guess probability: 77%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Server" (Guess probability: 72%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows 2000 Server Service Pack 3" (Guess probability: 72%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows XP SP1" (Guess probability: 72%)
[+] Host 192.168.10.136 Running OS: "Microsoft Windows NT 4 Server Service Pack 2" (Guess probability: 72%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.

```

Figure 71: Result of Xprobe2 port scan against Windows 10.

8.7 Appendix H - ICMP Latency

Table 11: The results of ICMP latency testing against each service in milliseconds.

rdppot	winxp	AHA	win10	octopus	xrdp
2.22	0.663	0.263	0.264	0.177	0.734
0.248	0.211	0.214	0.334	0.197	0.309
0.243	0.239	0.267	0.32	0.187	0.291
0.226	0.27	0.315	0.337	0.168	0.3
0.243	0.47	0.25	0.384	0.173	0.25
0.265	0.249	0.231	0.326	0.203	0.309
0.268	0.234	0.245	0.398	0.2	0.266
0.272	0.227	0.249	0.425	0.202	0.256
0.708	0.278	0.246	0.375	0.217	0.308
0.285	0.203	0.251	0.344	0.197	0.285
0.238	0.203	0.243	0.325	0.523	0.249
0.273	0.232	0.238	0.52	0.243	0.217
0.27	0.292	0.631	0.409	0.198	0.271
0.261	0.205	0.289	0.378	0.169	0.318

0.256	0.266	0.258	0.315	0.199	2.29
0.253	0.238	0.248	0.344	0.197	0.516
0.294	0.31	0.275	0.341	0.18	0.28
0.278	0.241	0.232	0.345	0.201	0.334
0.251	0.272	0.297	0.636	0.19	0.267
0.259	0.215	0.273	0.351	0.218	0.253
0.286	0.254	0.269	0.404	0.192	0.28
0.299	0.241	0.288	0.387	0.2	0.342
0.275	0.212	0.301	0.337	0.27	0.31
0.247	0.203	0.265	0.351	0.374	0.312
0.269	0.227	0.279	0.309	0.355	0.328
0.253	0.262	0.277	0.371	0.204	0.336
0.255	0.243	0.246	0.459	0.212	0.666
0.266	0.213	0.231	0.328	0.237	0.307
0.262	0.222	0.235	0.339	0.221	0.283
0.28	0.223	0.244	0.384	0.208	0.328
0.282	0.23	0.251	0.388	0.224	0.507
0.268	0.21	0.339	0.363	0.188	0.324
0.258	0.244	0.242	0.33	0.21	0.409
0.289	0.255	0.233	0.401	0.184	0.385
0.593	0.248	0.267	0.36	0.19	0.547
0.236	0.192	0.278	0.348	0.217	0.245
0.256	0.255	0.286	0.622	0.219	0.334
0.313	0.264	0.287	0.389	0.203	0.242
0.321	0.267	0.27	0.34	0.211	2.5
0.302	0.256	0.229	0.28	0.277	2.25
0.217	0.213	0.245	0.335	0.313	0.338
0.21	0.302	0.255	0.335	0.229	0.473
0.312	0.225	0.259	0.38	0.218	0.291
0.267	0.208	0.283	0.342	0.168	0.314
0.27	0.276	0.25	0.416	0.2	0.26
0.249	0.229	0.248	0.357	0.197	0.691
0.246	0.405	0.25	0.335	0.183	0.316
0.27	0.327	0.252	0.749	0.185	0.536
0.256	0.448	0.263	0.353	0.174	2.26
0.23	0.251	0.258	0.316	0.175	0.339
0.26	0.217	0.243	1.14	0.192	0.248
0.276	0.26	0.26	0.322	0.244	0.284
0.253	0.246	0.245	0.298	0.182	0.266
0.228	0.597	0.241	0.304	0.208	0.294
0.238	0.213	0.234	0.339	0.37	0.295

0.271	0.215	0.281	0.359	0.183	0.358
0.252	0.212	0.246	0.329	0.194	0.287
0.28	0.218	0.244	0.33	0.251	0.249
0.289	0.217	0.245	0.313	0.693	0.279
0.263	0.246	0.233	0.355	0.194	0.246
0.251	0.197	0.265	0.373	0.222	0.365
0.254	0.228	0.257	0.438	0.184	0.261
0.282	0.223	0.248	0.351	0.245	0.536
0.257	0.235	0.248	0.327	0.195	2.28
0.249	0.216	0.266	0.367	0.22	0.392
0.297	0.224	0.258	0.406	0.183	0.337
0.245	0.231	0.266	0.361	0.185	0.356
0.842	0.216	0.278	0.356	0.181	0.319
0.242	0.216	0.25	0.352	0.203	0.301
0.246	0.233	0.329	0.362	0.182	0.327
0.259	0.226	0.266	0.339	0.214	0.3
0.263	0.22	0.333	0.365	0.217	0.372
0.272	0.224	0.289	0.34	0.435	0.286
0.301	0.222	0.264	0.383	0.603	0.433
0.281	0.209	0.282	0.349	0.316	0.295
0.265	0.252	0.567	0.357	0.214	0.284
0.273	0.209	0.259	0.368	0.3	0.421
0.229	0.192	0.257	0.344	0.207	0.286
0.253	0.401	0.253	0.368	0.189	1.61
0.241	0.272	0.251	0.325	0.221	2.26
0.263	0.258	0.237	0.339	0.206	0.286
0.249	0.274	0.271	0.334	0.19	0.303
0.244	0.225	0.281	0.464	0.236	0.305
0.238	0.84	0.236	0.712	0.271	0.286
0.252	0.212	0.229	0.409	0.2	0.358
0.273	0.22	0.257	0.361	0.199	0.314
0.237	0.232	0.244	0.458	0.28	0.344
0.25	0.232	0.251	0.379	0.253	0.34
0.259	0.257	0.245	0.462	0.198	0.352
0.277	0.224	0.244	0.313	0.178	0.309
0.257	0.204	0.273	0.303	0.186	0.285
0.276	0.21	0.289	0.328	0.343	0.324
0.257	0.346	0.254	0.317	0.207	2.28
0.23	0.226	0.251	0.381	0.193	0.685
0.242	0.269	0.25	0.318	0.21	0.255
0.251	0.24	0.265	0.375	0.27	0.338

0.238	0.221	0.255	0.392	0.255	0.35
0.542	0.285	0.235	0.337	0.291	0.307
0.26	0.24	0.243	0.398	0.168	0.311
0.272	0.206	0.246	0.338	0.222	0.238

8.8 Appendix I - TCP Latency

Table 12: The results of TCP latency testing against each service in milliseconds.

rdppot	winxp	AHA	win10	octopus	xrdp
2.92	0.3	0.7	0.6	0.23	0.44
0.32	0.32	0.81	0.43	0.49	0.37
0.33	0.28	0.63	0.4	0.36	0.4
0.34	0.28	1.03	0.45	0.32	0.45
0.43	0.27	0.73	0.64	0.33	2.26
0.33	0.28	0.85	0.42	0.3	0.31
0.39	0.29	0.71	0.41	0.32	0.38
0.34	0.26	1.75	0.39	0.29	0.36
0.36	0.3	0.88	0.42	0.28	0.33
0.31	0.31	0.78	0.44	0.3	0.4
0.33	0.34	0.78	0.39	0.26	0.33
0.31	0.33	0.71	0.44	0.26	0.33
0.32	0.28	0.64	0.46	0.28	2.27
0.34	0.28	0.84	0.38	0.43	0.33
0.35	0.28	0.71	0.51	0.28	0.31
0.34	0.31	0.62	0.46	0.24	0.4
0.34	0.28	0.68	0.42	0.29	0.35
0.38	0.25	0.72	0.52	0.27	0.42
0.34	0.25	0.92	0.44	0.25	0.42
0.3	0.26	0.8	0.46	0.37	0.52
0.56	0.26	0.75	0.47	0.31	2.29
0.44	0.28	0.84	0.44	0.24	0.3
0.35	0.28	0.74	0.49	0.25	0.42
0.33	0.35	0.72	0.45	0.29	0.31
0.41	0.28	0.66	0.37	0.29	0.34
0.41	0.3	0.62	0.39	0.26	0.33
0.33	0.31	0.71	0.44	0.27	0.53
0.46	0.41	0.78	0.44	0.25	0.41
0.37	0.38	0.74	0.45	0.28	0.33
0.39	0.26	0.57	0.39	0.27	0.29
0.39	0.38	0.77	0.36	0.26	0.31
0.35	0.42	0.7	0.46	0.31	0.36

0.39	0.32	0.99	0.66	0.29	0.28
0.47	0.48	0.75	0.94	0.36	0.36
0.37	0.25	0.71	0.93	0.29	0.29
0.36	0.3	0.77	0.37	0.31	0.29
0.4	0.34	0.63	0.46	0.51	3.02
0.46	0.28	0.73	0.38	0.25	0.29
0.35	0.28	0.75	0.42	0.26	0.32
0.37	0.3	0.67	0.4	0.27	0.34
0.38	0.35	0.65	0.36	0.63	0.33
0.36	0.25	0.67	0.36	0.23	0.27
0.45	0.3	0.71	0.36	0.28	0.26
0.34	0.31	0.77	0.35	0.21	0.31
0.42	0.26	0.68	0.39	0.23	0.3
0.37	0.34	0.85	0.4	0.22	0.44
0.36	0.3	0.73	0.5	0.24	0.37
0.33	0.26	0.74	0.38	0.27	0.31
0.36	0.32	0.76	0.38	0.26	0.31
0.38	0.3	0.8	0.37	0.26	0.38
0.43	0.32	0.69	0.38	0.25	0.28
0.45	0.25	0.72	0.39	0.35	0.31
0.36	0.31	0.72	0.38	0.21	0.64
0.34	0.42	0.71	0.39	0.23	0.5
0.38	0.28	0.7	0.44	0.26	0.3
0.34	0.36	1.4	0.39	0.26	0.33
0.36	0.28	0.75	0.43	0.23	0.3
0.33	0.32	0.81	0.43	0.24	0.38
0.35	0.32	0.65	0.4	0.24	0.3
0.34	0.3	0.68	0.45	0.39	0.3
0.33	0.29	0.65	0.46	0.27	0.3
0.41	0.29	0.74	0.5	0.22	0.29
0.34	0.32	0.7	0.47	0.24	0.3
0.31	0.3	0.59	0.5	0.34	0.37
0.36	0.29	0.8	0.45	0.28	0.35
0.34	0.33	0.81	0.47	0.34	0.36
0.37	0.32	0.78	0.42	0.4	0.37
0.35	0.28	0.68	0.4	0.25	0.53
0.6	0.33	0.8	0.4	0.23	0.35
0.33	0.28	0.7	0.39	0.24	0.36
0.42	0.29	0.78	0.41	0.23	0.32
0.41	0.37	0.75	0.36	0.23	0.33
0.79	0.41	0.78	0.45	0.26	0.41

0.34	0.27	0.68	0.46	0.23	0.4
0.37	0.4	0.71	0.4	0.25	0.32
0.37	0.3	0.65	0.42	0.23	0.37
0.33	0.26	0.89	0.4	0.23	0.35
0.33	0.32	0.66	0.47	0.26	0.33
0.36	0.36	0.63	0.37	0.29	0.3
0.37	0.3	0.72	0.44	0.37	0.86
0.33	0.37	0.68	0.5	0.3	0.38
0.34	0.25	0.73	0.4	0.39	0.34
0.36	0.3	0.67	0.45	0.58	0.33
0.39	0.29	0.76	0.44	0.36	2.36
0.39	0.29	2.3	0.52	0.38	0.39
0.39	0.28	0.59	0.38	0.36	0.43
0.35	0.39	0.64	0.43	0.32	0.35
0.36	0.35	0.71	0.49	0.26	0.4
0.35	0.55	0.72	0.69	0.27	0.36
0.37	0.32	0.76	0.4	0.38	0.73
0.34	0.31	0.67	0.45	0.23	0.33
0.32	0.44	0.76	0.48	0.33	0.41
0.29	0.31	0.78	0.48	0.33	0.41
0.38	0.3	0.71	0.41	0.46	0.39
0.34	0.35	0.56	0.4	0.34	0.38
0.4	0.31	0.7	0.41	0.35	0.32
0.33	0.34	0.81	0.41	0.25	0.34
0.39	0.51	0.76	0.4	0.27	0.42
0.39	0.36	0.69	0.43	0.26	0.41
0.35	0.28	0.65	0.56	0.25	0.47

8.9 Appendix J - File Persistence Results

8.9.1 Rdpptot

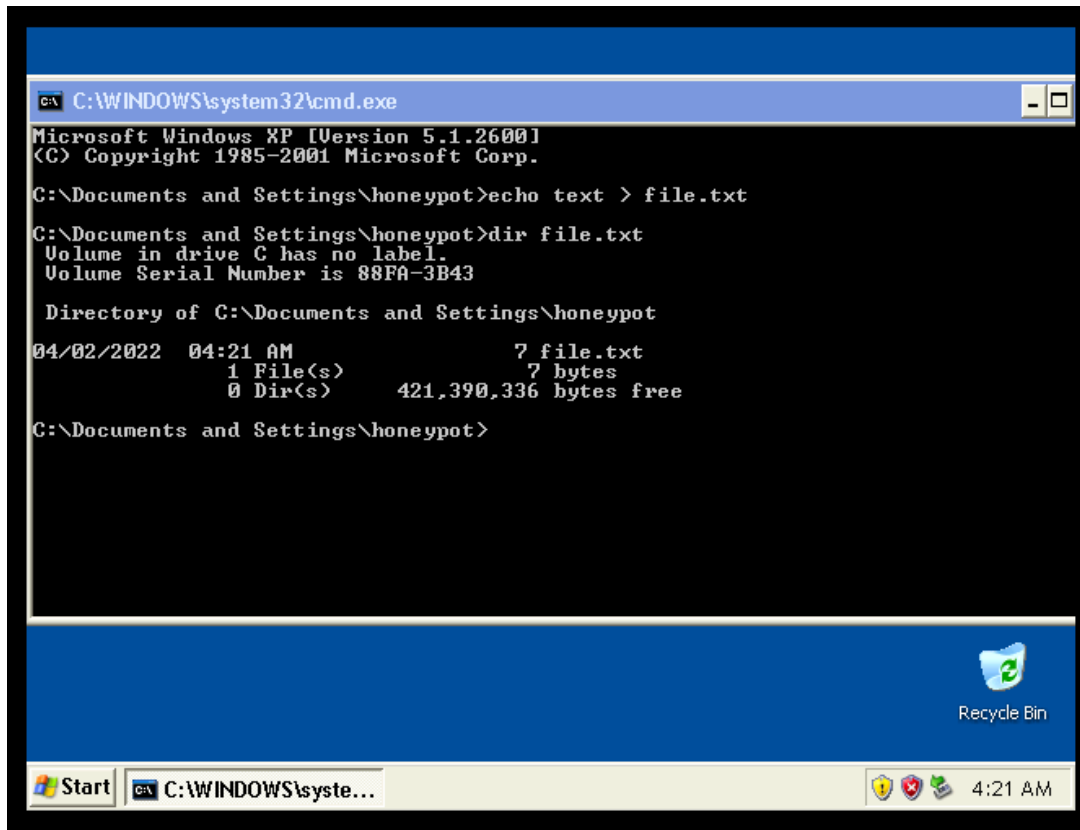


Figure 72: Creating the file on rdpptot to test for file persistence.

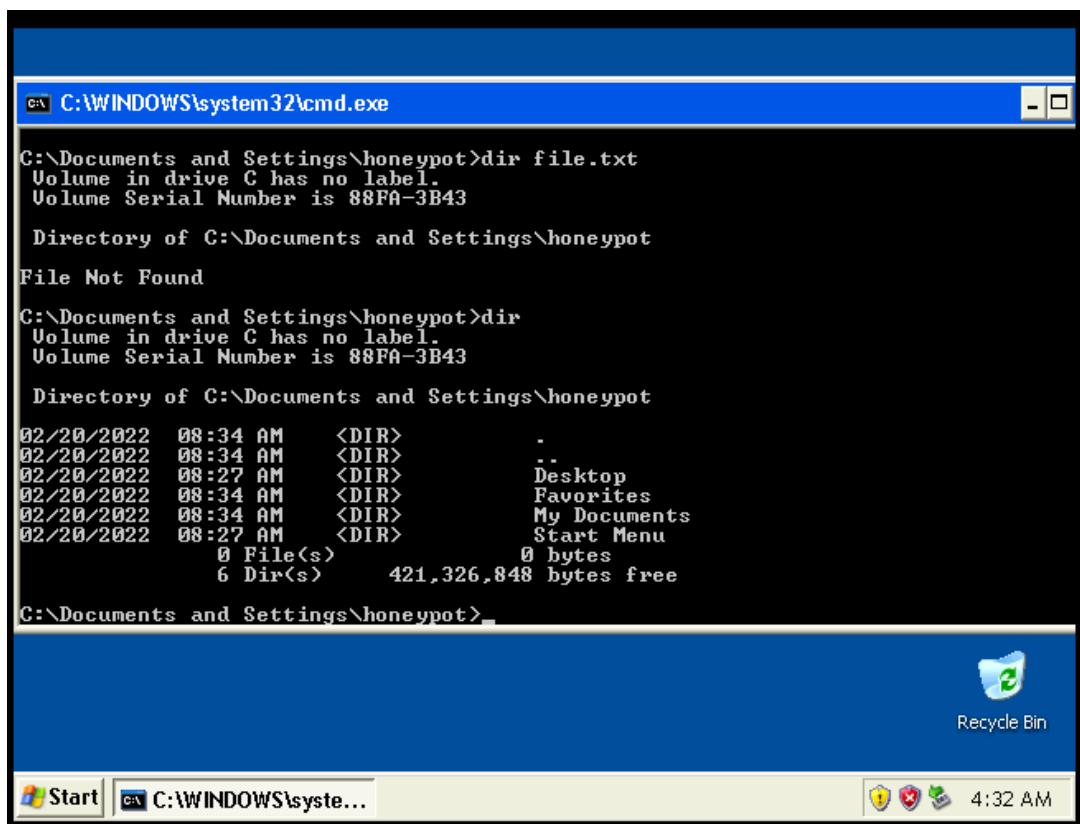
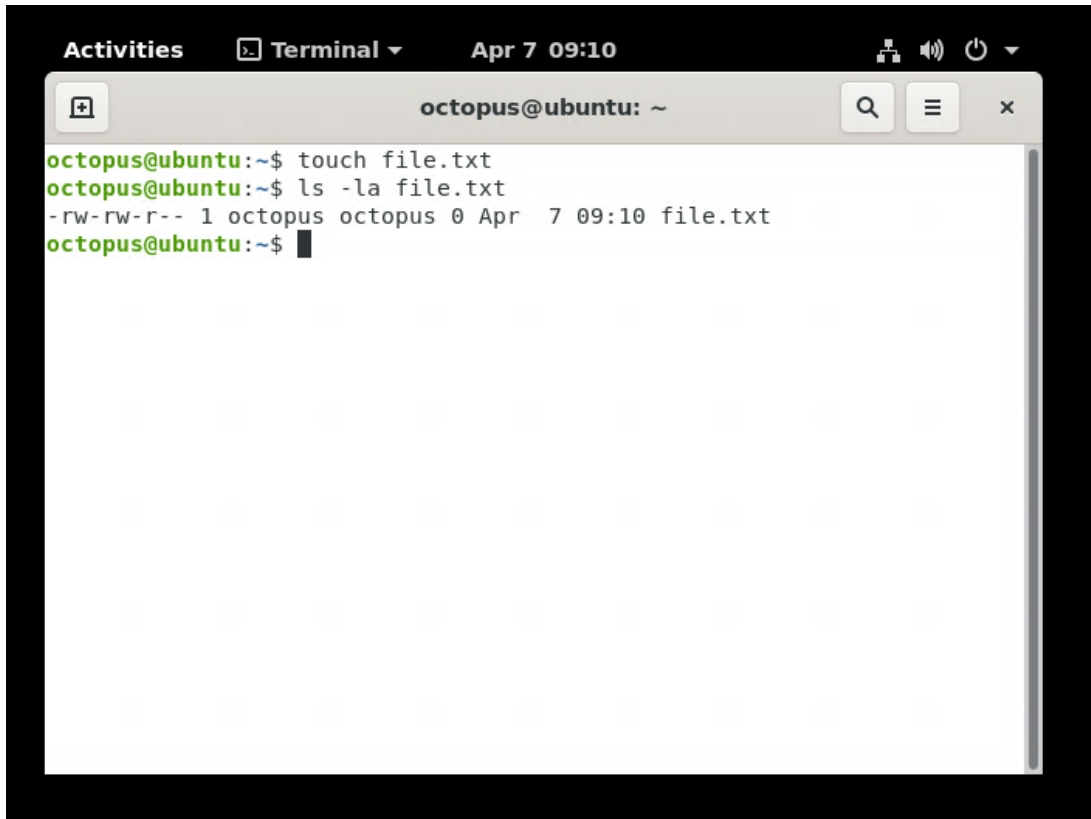


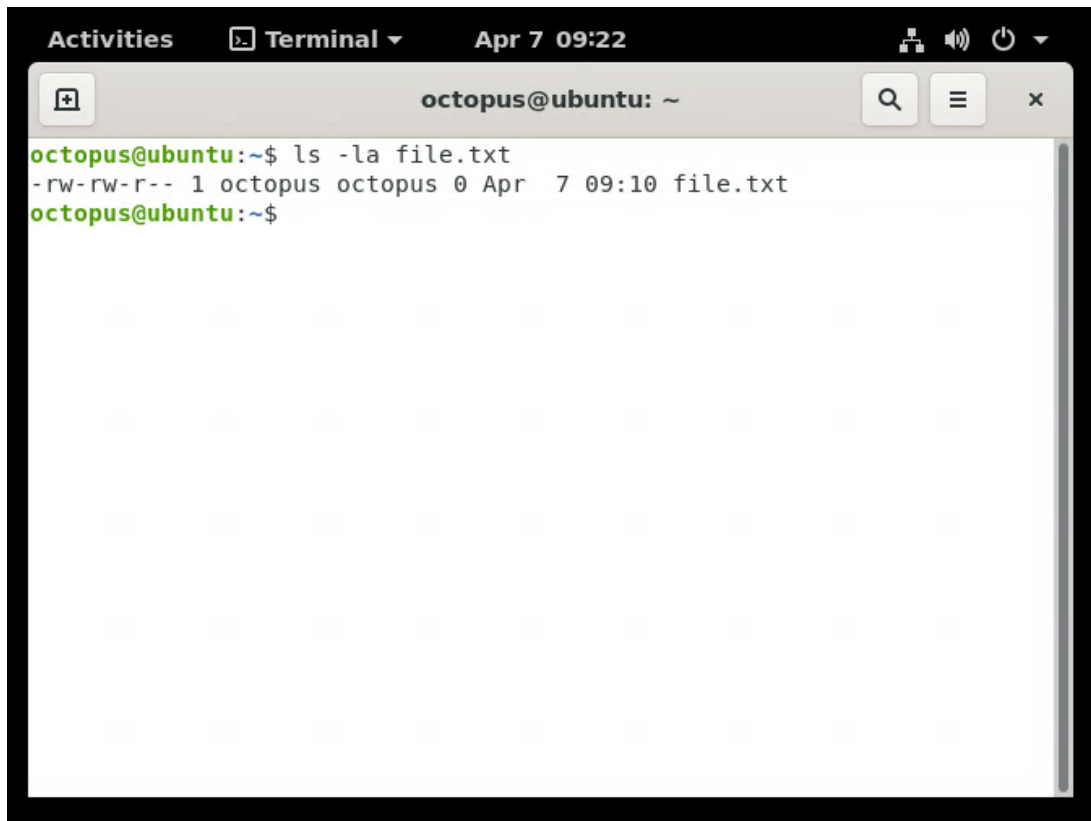
Figure 73: The file no longer being found after reconnecting.

8.9.2 Octopus

A terminal window titled 'octopus@ubuntu: ~' with a search icon, menu icon, and close icon. The terminal shows the following commands and output:

```
octopus@ubuntu:~$ touch file.txt
octopus@ubuntu:~$ ls -la file.txt
-rw-rw-r-- 1 octopus octopus 0 Apr  7 09:10 file.txt
octopus@ubuntu:~$
```

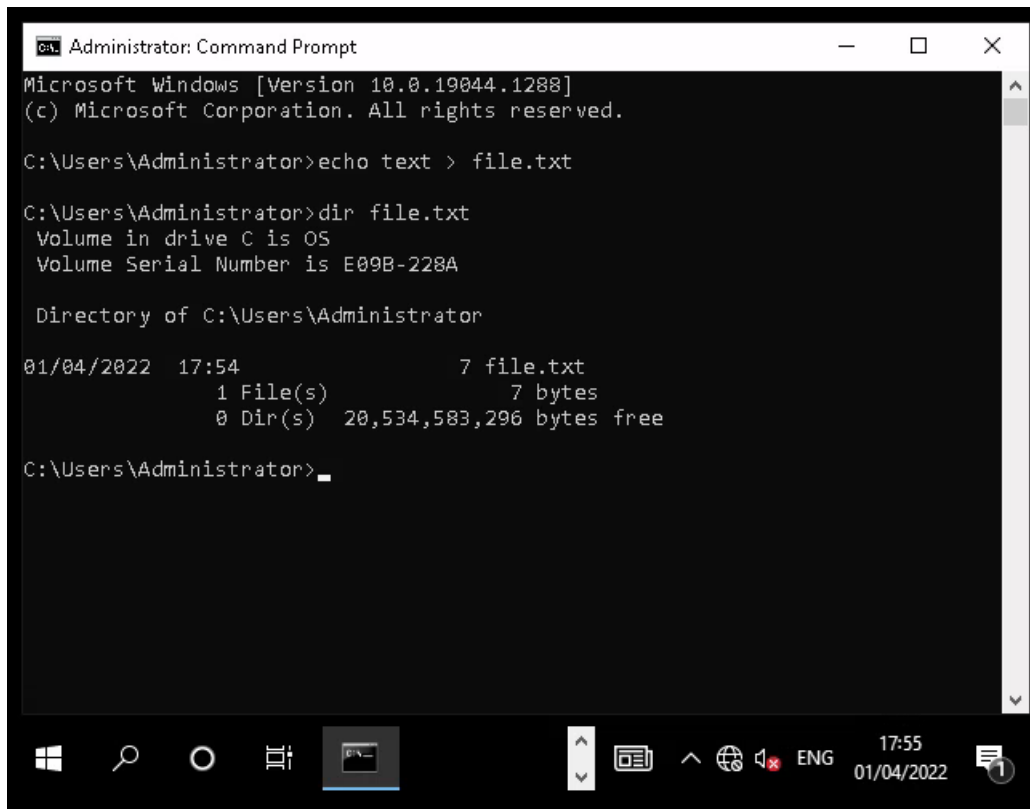
Figure 74: Creating the file on Octopus to test for file persistence.

A terminal window titled 'octopus@ubuntu: ~' with a search icon, menu icon, and close icon. The terminal shows the following command and output:

```
octopus@ubuntu:~$ ls -la file.txt
-rw-rw-r-- 1 octopus octopus 0 Apr  7 09:10 file.txt
octopus@ubuntu:~$
```

Figure 75: The file is still present after reconnecting.

8.9.3 AHA



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19044.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator>echo text > file.txt

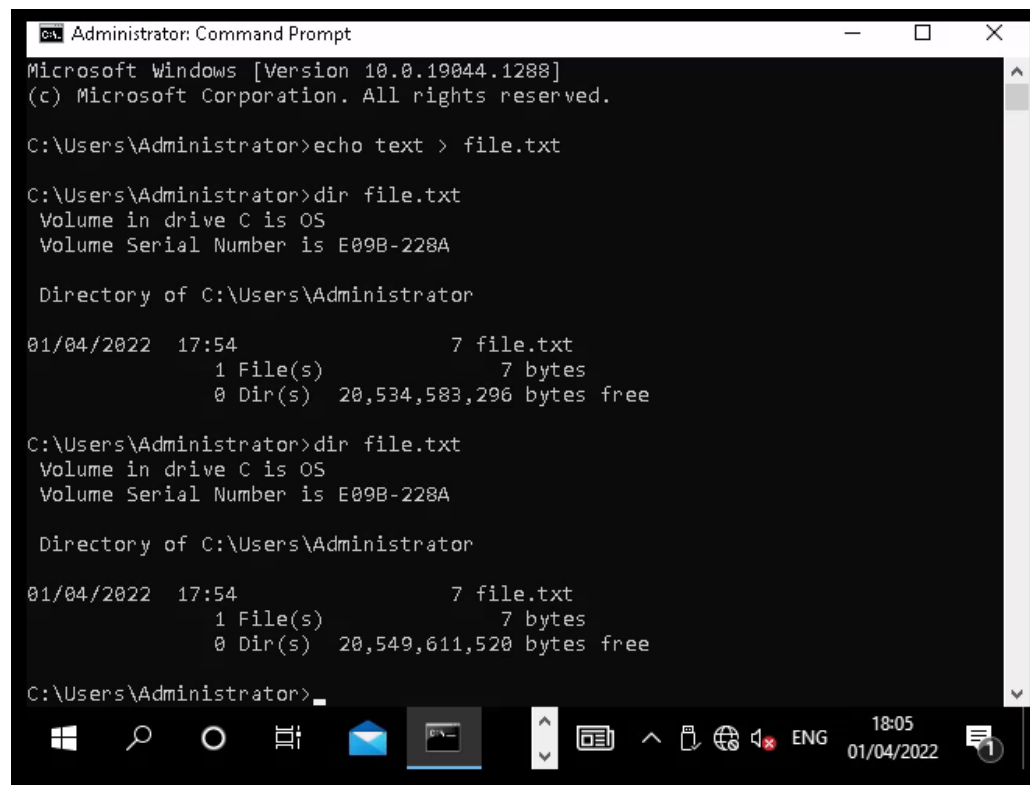
C:\Users\Administrator>dir file.txt
Volume in drive C is OS
Volume Serial Number is E09B-228A

Directory of C:\Users\Administrator

01/04/2022  17:54                7 file.txt
             1 File(s)                7 bytes
             0 Dir(s) 20,534,583,296 bytes free

C:\Users\Administrator>
```

Figure 76: Creating the file on AHA to test for file persistence.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19044.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator>echo text > file.txt

C:\Users\Administrator>dir file.txt
Volume in drive C is OS
Volume Serial Number is E09B-228A

Directory of C:\Users\Administrator

01/04/2022  17:54                7 file.txt
             1 File(s)                7 bytes
             0 Dir(s) 20,534,583,296 bytes free

C:\Users\Administrator>dir file.txt
Volume in drive C is OS
Volume Serial Number is E09B-228A

Directory of C:\Users\Administrator

01/04/2022  17:54                7 file.txt
             1 File(s)                7 bytes
             0 Dir(s) 20,549,611,520 bytes free

C:\Users\Administrator>
```

Figure 77: The file is still present after reconnecting.

8.10 Appendix K - Further Attack Results

8.10.1 Rdpot

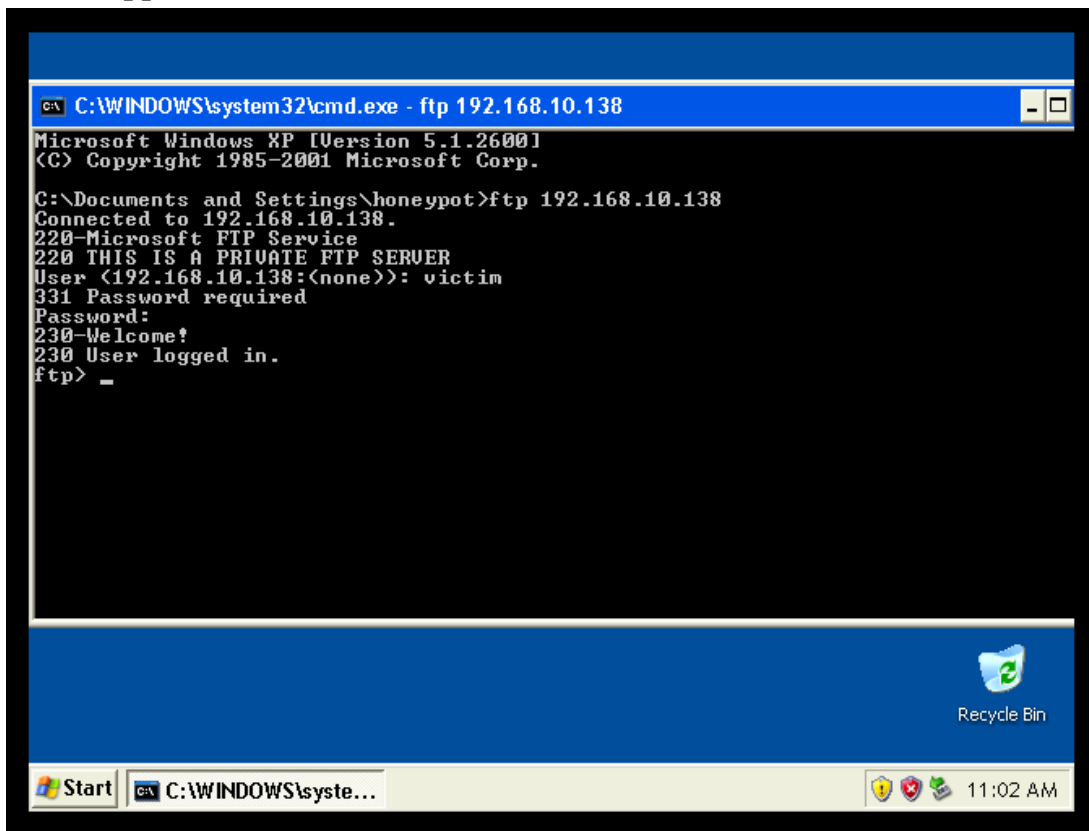
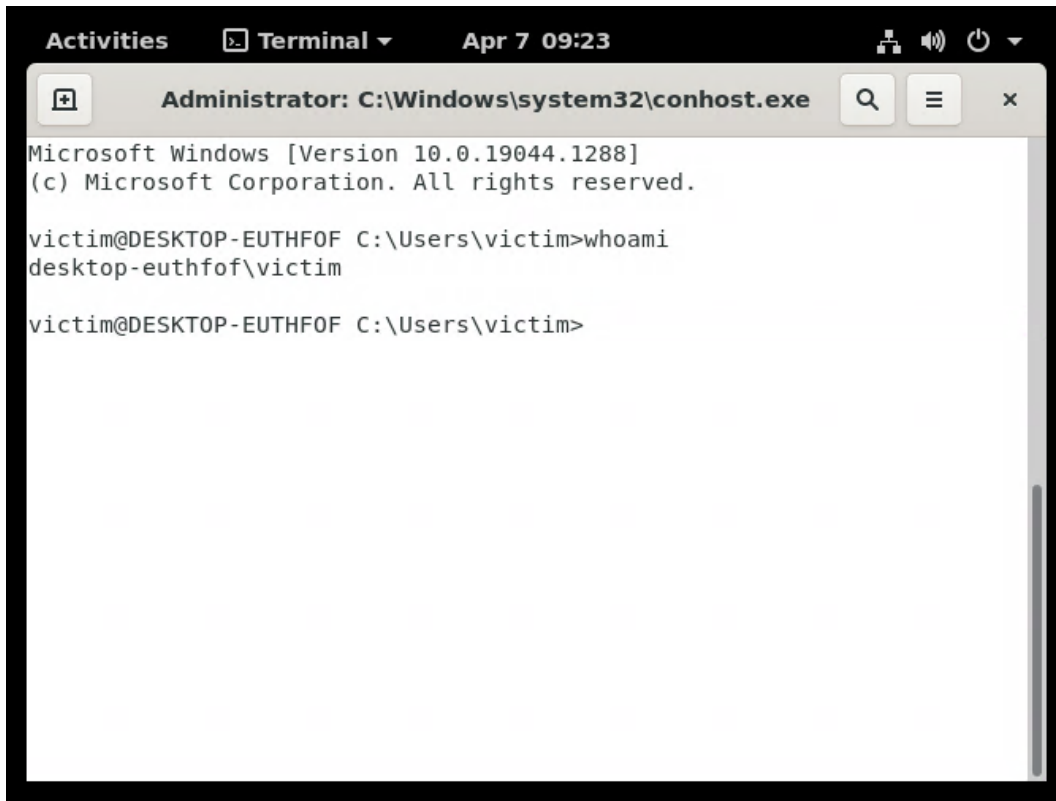


Figure 78: Result of attempting a further attack from the rdpot honeypot, successfully accessing another machine via FTP.

8.10.2 Octopus



The image shows a Windows terminal window titled "Administrator: C:\Windows\system32\conhost.exe". The terminal output is as follows:

```
Microsoft Windows [Version 10.0.19044.1288]
(c) Microsoft Corporation. All rights reserved.

victim@DESKTOP-EUTHFOF C:\Users\victim>whoami
desktop-euthfof\victim

victim@DESKTOP-EUTHFOF C:\Users\victim>
```

Figure 79: Result of attempting a further attack from the Octopus honeypot, successfully accessing another machine via SSH.

8.10.3 AHA

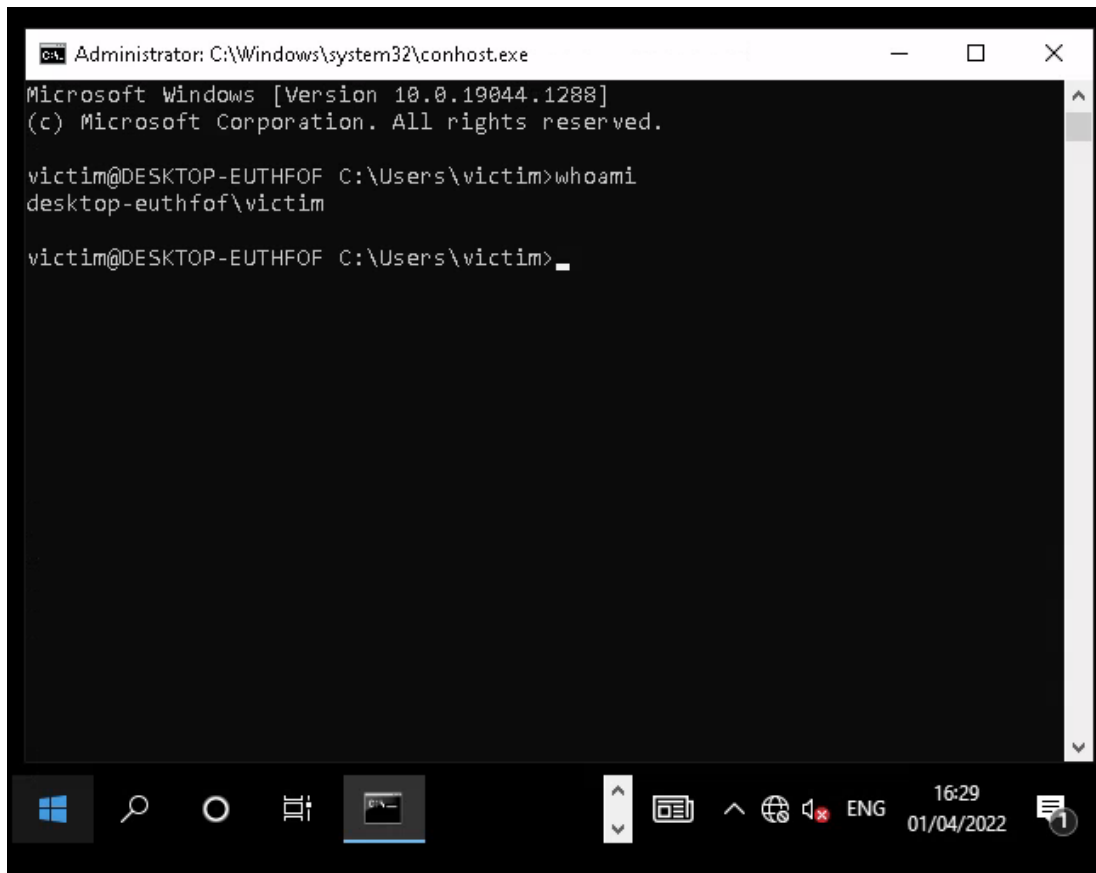


Figure 80: Result of attempting a further attack from the AHA honeypot, successfully accessing another machine via SSH.