Looking for Honey Once Again: Detecting RDP and SMB Honeypots on the Internet

Fabian Franzen
Technical University of Munich
franzen@sec.in.tum.de

Lion Steger
Technical University of Munich
lion.steger@tum.de

Johannes Zirngibl, Patrick Sattler
Technical University of Munich
{zirngibl,sattler}@net.in.tum.de

***Abstract*—Honeypots are a widely used technique to observe the spread of malware and the emergence of new exploits. Attackers try to avoid connecting to honeypots as they reveal the attacker's methods, tools, and exploits.**

While different honeypot implementations have been fingerprinted in the past, we see a lack of studies covering Windows-related protocols such as Remote Desktop Protocol (RDP) and Server Message Block (SMB) honeypots. However, these protocols have seen at least two major security vulnerabilities in the past 5 years and are commonly exploited.

We adapted existing fingerprinting algorithms to allow an accurate identification of RDP and SMB honeypots checking how implementations behave in error conditions. We present a new improvement, namely the inclusion of system TLS stack features previously not used for honeypot detection. We are the first to perform an internet-wide scan searching for RDP and SMB honeypots. We are able to effectively uncover the presence of two common open-source honeypots for RDP and SMB each.

We identified 84 instances of *Heralding* (RDP), 1123 instances of *RDPY* (RDP), 60 instances of *Impacket* (SMB), and 1461 instances of *Dionaea* (SMB) during our scans. Furthermore, we found several hosts, which do not use Microsoft's *SChannel* TLS stack, but advertise themselves as Windows machines. This indicates the presence of a Man-in-the-Middle (MitM) box and could be a sign of a honeypot. Eventually, we analyzed how attackers interact with detectable honeypots. We deployed instances of RDP honeypots ourselves and found that credential guessing attackers seem to avoid them.

This proves that RDP and SMB honeypots are fingerprintable and that even MitM-box-based high-interaction honeypots leave detectable traces.

***Index Terms*—honeypots, Internet scanning, RDP, SMB**

1. Introduction

Honeypots have become a settled and well-researched technique to watch the emergence of exploits on different services on the Internet. They allow researchers to detect the rise of new exploits and to keep track of how widely specific vulnerabilities are being exploited.

Attackers have an interest in detecting and avoiding honeypots, as they do not want to draw attention to their specific exploitation techniques and tools [1]. Tools could be as simple as the list of passwords most often tried by an attacker, which a system administrator, in turn, may then place on a weak-password list.

Despite much research having been conducted on honeypots, security researchers still suggest new usage scenarios and designs. For example, in 2020, an improved ICS honeypot to catch malware directed at industrial control systems (ICS) [2] and a specialized honeypot for hardening neural networks against attacks [3] have been proposed.

Organizations analyzing the Internet have also shown interest. Portals like *Censys.io* and *Shodan.io* allow reasoning about the number of services on the Internet. Their data often includes information about security vulnerabilities and used software. Detecting honeypots is a logical next step. *Shodan.io* started the *Honeyscore* project to detect honeypots specifically for ICS [4] and announced they will directly annotate their search results with honeypot tags [5]. This indicates the interest of the industry in further developing honeypot detection techniques. Unfortunately, they do not share their detection methods with the public.

The Windows world has received less attention by researchers, but faced at least two wormable security vulnerabilities in the recent past. First, CVE-2017-0144 in Microsoft's Server Message Block (SMB) protocol, which is exploited by the *EternalBlue* exploit. Second, BlueKeep (CVE-2019-0708) is a vulnerability in Microsoft's Remote Desktop Protocol (RDP). Multiple security researchers reported the use of honeypots to monitor the attacks [6]. SMB and RDP protocol traffic is in the top 7 on SurfNet's Internet telescope data [7]. However, existing research primarily targets SSH, HTTP, ICS, and Telnet honeypots, but no Windows protocols such as RDP and SMB as we will discuss further in Section 2.

In this paper, we present the results of an internet-wide scan for RDP and SMB honeypots. We created a small set of network packets that allow for a fast internet-wide scan following the methods of Vetterl et al. [8] with extensions we describe further in Section 4. The set allows for an accurate classification of the hosts, as we demonstrate by performing an internet-wide scan to estimate the number of honeypots of selected open-source implementations. Moreover, we illustrate how unique features of the Microsoft SChannel and OpenSSL TLS implementation can be used to detect *abnormal* RDP stacks.

Our contributions can be summarized as follows:

(i) We present how existing honeypot fingerprinting approaches can be adapted to detect SMB and RDP honeypots. We implemented our ideas in our own *honeypot*

detector. Furthermore, we conducted a 34-day experiment to collect evidence that attackers avoid RDP honeypots.

(ii) We show that a fingerprintable TLS stack allows improved distinction between Windows and non-Windows hosts.

(iii) We performed internet-wide scans for RDP and SMB hosts and present numbers on the spread of selected open-source honeypots for these protocols.

(iv) We verified our results by connecting to a random subset of each of our classification groups and found that we have only classified 5 out of 1097 hosts incorrectly (i.e., less than 1%). In the honeypot classification groups, we have not identified a single misclassified host.

(v) We publish[1] our scanning and detection tools in order to support reproducibility such that others can use and improve upon our work. Additionally, we provide our dataset on request to interested researchers. In contrast to existing data from Censys and Shodan, our dateset includes the raw exchanged data in unparsed form and covers multiple protocol versions.

2. Related Work

Many honeypot detection approaches [7]–[10] are based on detecting shortcomings in the honeypot implementation of the respective protocol. However, the detection can also be based on the *combination of services offered* [11] or how the honeypot interacts with other services on the Internet [12]. Several open-source honeypots offer a wide set of services they can emulate. A host offering myriads of different services is unusual, especially if running services requires different operating systems. An example for interaction based detection is *HoneypotHunter* [12]. It checks for SMTP honeypots by connecting to potential open SMTP relays and sending an e-mail, addressed to a server under the detector's control. If the attacker does not receive an inbound connection on his server, especially if the host announced correct delivery, a honeypot has been found.

Vetterl et al. [8] detected SSH, Telnet, and HTTP honeypots by fuzzing honeypots and their real-world counterparts. Out of these data, they derived a *most distinctive probe* being used to perform an internet-wide scan. We based our research on this idea and will discuss it further in Sec. 4. In 2019, Morishita et al. [7] investigated the prevalence of 14 open-source honeypots in Censys Internet scanning data. They derived signatures based on protocol banners, HTTP responses, and error responses. However, they only analyze FTP, SSH, Telnet, SMTP, HTTP, and IMAP honeypots. They found about 17k honeypots for different protocols.

Honeypots and honeynets, a whole network of honeypots, have also been detected by abusing timing and network characteristics. In 2006, network packets traveling through the honeynet implementation *honeyd* were known to have a round-trip time of a multiple of ten due to characteristics of the OS and the simulation [13]. Another network latency-based detection for honeynets, which takes other fields of TCP/IP packets into consideration, is discussed in [14]. Holz et al. [1] propose the detection of so-called high-interaction honeypots, which are instrumented or sandboxed versions of the real service through artifacts of the virtualization environment or emulator.

The detection of honeypots has also found its way into the products of internet-scanning companies such as Shodan. Shodan implemented *Honeyscore* [4], which focuses on ICS honeypots and is reported to become a standard feature in their search engine [5]. Honeyscore uses a mixture of the discussed detection features. First, they consider if a service has too many open network ports. Second, they consider if a service is running in an unusual IPv4 address space (i.e., a PLC in the Amazon EC2 cloud). Third, they search if a service response matches known honeypot configurations. Lastly, they employ machine learning in an undisclosed manner for detection [2].

The interaction between honeypots and attackers has also received research interest. In 2018, Metongnon et al. [15] have analyzed data of the *SurfNet* network telescope and compared it with the incoming traffic on their telnet honeypots. In 2019, Ghiette et al. [16] conducted a large-scale study to fingerprint attackers of SSH servers by using features such the SSH banner of the client, the MD5 hash of the offered crypto algorithms and the passwords used at login attempts.

3. Background

RDP as well as SMB have a rich feature set and a long history. In the following, we introduce the basic functionality of both protocols before we outline our ideas for fingerprinting and honeypot detection in Section 4.

3.1. RDP

The Remote Desktop Protocol (RDP) allows for remote access and maintenance of the Windows operating system. Microsoft released RDP 5.1 and 5.2 together with Windows XP and Windows Server 2003 and continuously developed it ever since; now RDP 10 for Windows 10/11 and Windows Server 2016 onward is available [17]. Besides its primary screen sharing functionality, it offers additional features like clipboard sharing, redirection of peripherals such as smart card readers and drives to the RDP server. The huge amount of features inhibits a free reimplementation, even though the protocol is described as part of the Microsoft Open Specifications program [18].

Nonetheless, RDP is also used by third parties to offer remote control to non Windows operating systems. For example, a popular open-source implementation is *XRDP* for Unix-based operating systems; Oracle also offers a closed-source extension for *VirtualBox* to enable access to virtual machines using RDP.

Figure 1 shows the initial steps of the RDP protocol which are embedded in the *TPKT* format. RDP's security evolved over time and covers multiple security modes. The client and server agree upon which mode should be used during the first steps of the protocol (Step 1 in Figure 1). In the context of our work, the following modes are relevant:
PROTOCOL_RDP: This mode must be supported by the client and indicates the use of standard RDP security; a mode where encryption and integrity protection *can* be done in the RDP protocol itself. Step 2 and 3 in

1. https://github.com/tum-itsec/looking-for-honey-once-again

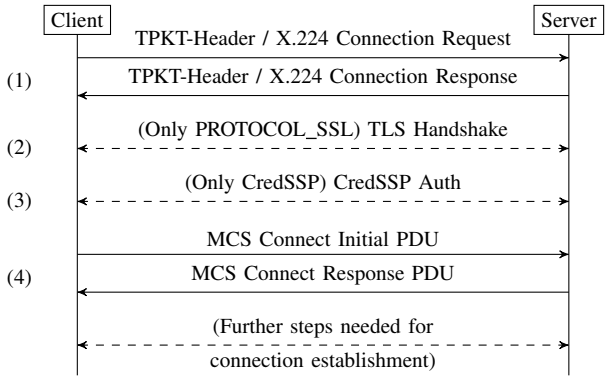

Figure 1. The first steps of the RDP protocol.

Figure 1 are skipped in this case. Most of the selectable cryptographic primitives in this mode (such as RC4 and MD5) are outdated nowadays.

PROTOCOL_SSL: After the first message exchange, the negotiation phase, both parties establish a secure channel via the TLS protocol. Authentication of the server is realized through an X.509 certificate. Authentication of the client is done by the user entering their credentials into the login UI after all other negotiations are finished.

PROTOCOL_HYBRID: The authentication of the user is done immediately after the TLS handshake has finished. All further negotiations are done *afterwards*. Microsoft refers to this mode also as *Network Level Authentication (NLA)*. Authentication of the user could be done via a classic (username, password) tuple, by passing a Kerberos ticket, or by using a smart card.

Note that only in the *PROTOCOL_HYBRID* mode the user has to authenticate itself early in the third step. In all other modes listed above, the server will willingly provide a lot of information and commit resources on establishing the connection before the user is authenticated. Therefore, recent Windows installations suggest the use of Network Level Authentication, which is equivalent to enforcing *PROTOCOL_HYBRID* during connection establishment (from Windows Server 2008 R2 onward).

3.2. SMB

The Server Message Block protocol (SMB) is primarily used by Windows operating systems to share locally stored files with other machines in the same network to allow browsing, editing, and deletion. Besides sharing files, it also offers inter-process communication, e.g., in the form of named pipes. On top of that, Microsoft Remote Procedure Calls (MSRPC) can be used.

Microsoft released SMB 1.0 together with Windows 2000 and SMB 3.1.1 with Windows 10. Windows supports end-to-end encryption and integrity protection since SMB 3.0. Beforehand, SMB offered none of these services, rendering it an easy target to attackers and making the protocol unsuitable for communication over an untrusted network such as the Internet. Therefore, many network operators filter the default SMB port 445. Nevertheless, around 1M hosts are offering their SMB service on the Internet according to Censys data.

Besides the implementation in Microsoft Windows, *Samba* has become the most prominent open-source implementation of SMB. We chose two versions, 3.5.6, as it is the last version of Samba without support for SMBv2 and 4.10.0, being the latest version available to Ubuntu 19.10 at the time of our experiments. Furthermore, commercial competitors such as Visuality Systems have also implemented SMB.

SMB communication starts by negotiating a dialect (e.g., “NT LANMAN 1.0” or “SMB 2.002”) with the server. If backward compatibility is required, the negotiation is started by sending an SMBv1 negotiation packet. A list of supported dialects is sent alongside this negotiation packet, similar to TLS. If backward compatibility is not desired, a connection can also start with an SMBv2 packet. The server selects one dialect from the offered list and responds with its choice.

After dialect negotiation, the sequence continues with the Session setup phase. During this phase, the user is authenticated via the Simple and Protected GSSAPI Negotiation (SPNEGO) that uses the General Security Service API (GSSAPI) and which in turn transports authentication mechanisms such as Kerberos or the NT Lan Manager Protocol (NTLM). The server can choose to authenticate the user without credentials and grant access to the server anonymously.

3.3. Honeypot Implementations

Honeypot implementations are grouped into *low*, *medium* and *high-interaction* honeypots. Low- and medium-interaction honeypots focus on easy deployment and maintenance while implementing only basic functionality. High-interaction honeypots try to mimic a service indistinguishable from the original and are often based on virtual machines to minimize operational risks.

We focused on finding open-source implementations for RDP and SMB mentioned above. We identified two honeypot implementations respectively, which we will describe in the following.

***Heralding* (RDP)** *Heralding* [19] focuses on catching login credentials of everybody logging into the system. Therefore, it is not in focus of the developers to provide a complete protocol implementation. RDP connections are terminated early if the credentials are not submitted during connection establishment.

***RDPY* (RDP)** Another honeypot implementation for the RDP protocol is *RDPY* [20], written in Python 2 on top of the Twisted framework. The first commit dates back to the year 2013. Since then, it has been heavily developed until 2015, but no RDP protocol related fix happened afterwards. *RDPY* supports `PROTOCOL_SSL` and `PROTOCOL_RDP` security. However, if the administrator does not create an X.509 certificate, `PROTOCOL_SSL` will be disabled. In contrast to *Heralding*, an attacker can complete the full RDP connection sequence and will be able to watch an emulated screen. During operation as honeypot, content and events on the emulated screen are replayed from a session recorded beforehand. In order to create a recording, RPDY is used in a preparation phase as a proxy to a regular server. The implementation proxys the requests to the regular server and records all screen update and keypress events. Afterwards, the recording is replayed to every attacker. This allows the honeypot to

TABLE 1. FEATURES OF ANALYZED HONEYPOT IMPLEMENTATIONS

Honeypot	Age	RDP			SMB		
		RDPSec	SSL	Hybrid	v1	v2	v3
Impacket	2003				✔	✔	✔
Dionaea	2009				✔		
RDPY	2013	✔	✔				
Heralding	2012		✔				

mimic a real system as long as the attacker does not try to interact with the system.

***Dionaea* (SMB)** *Dionaea* [21] is a honeypot for a wide range of protocols. Since its initial git commit back in 2009, it advertises support for over 14 protocols, including FTP, HTTP, Memcached, MSSQL, SMB, SIP and UPNP. The *Dionaea* core is implemented in the C programming language with support for Python modules implementing most of its protocol logic. *Dionaea* does not offer RDP support. SMB support is implemented in Python and restricted to SMBv1.

***Impacket* (SMB)** *Impacket* is a collection of Python classes for working with networking protocols especially from the Windows domain. This includes high level implementations of SMBv1, SMBv2, and SMBv3 [22]. While *Impacket* is not a honeypot on its own, we found honeypot implementations using this library during our Internet scans. The library is provided by SecureAuth, a security company, and seems to be solely developed for the needs of penetration testers. SecureAuth provides numerous examples how to use the library in this field, e.g., an exploit for the SMB Relay Attack (CVE-2015-0005).

Besides the honeypots mentioned, further implementations targeting SMB are for example *DTK* or *Smoke Detector* (see also [23] for a more complete survey of popular honeypot implementations). However, these implementations have stalled in their development since 2005 and we will therefore not consider them here.

Table 1 summarizes the different characteristics of selected honeypots. These honeypots can also be combined to cover multiple protocols or scenarios. An example are the T-Pot Docker and virtual machine images created by T-Systems, which also hosts T-Pot actively for their Sicherheitstacho project[2]. It contains all presented tools but *Impacket*.

3.4. TLS Fingerprinting

As our approach will leverage the fact that many honeypots implement their services without taking the TLS stack into consideration, we provide a short overview of TLS fingerprinting techniques. We found that modern RDP implementations mainly use TLS 1.2. Therefore, we will not describe the specifics of the more recent TLS 1.3.

During the establishment of a TLS connection, the cipher suite for encryption and integrity protection as well as the algorithms for key exchange need to be negotiated. In addition, the client can present a set of supported TLS extensions. The server decides which extension subset is accepted and signals this together with the selected cipher in the *Server Hello Message*.

2. https://www.sicherheitstacho.eu/start/main

This property was used by research [24], [25] and different groups to fingerprint TLS clients. Notable tools are JA3 [26], Cisco Mercury [27] and Cisco Joy [28]. They are based on similar approaches, creating a hash of the mentioned parameters of the TLS Client Hello to identify implementations. JA3 provides an additional extension JA3s to fingerprint servers besides clients. The server behavior is not only influenced by the used implementation but also by each Client Hello because it can only select offered properties for a successful handshake. Therefore, JA3s relies on the combination of client and server fingerprints. In order to proactively get server fingerprints, JARM [29] was developed. It fingerprints a server based on its behavior for 10 manually crafted Client Hellos.

The approach presented in this paper is based on our observation of two further characteristics. First, if the server agrees on a PFS-enabled cipher, an algorithm for key exchange needs to be negotiated. We observed frequent use of ECC here. If chosen, the curve needs to be agreed upon before the key exchange can happen.

Second, TLS allows multiple handshake messages to be packed together inside a single TLS record packet. The *SChannel Service Provider* (SSP) implementation of Windows packages does this. In contrast, OpenSSL creates a TLS record for each handshake message.

4. Our Honeypot Detector

SMB and RDP are both complex protocols that are very hard to implement identical and feature-complete with respect to a reference implementation when developing a low- to medium-interaction honeypot.

Following the methods of Vetterl et al. [8], we fuzzed all RDP and SMB implementations we are aware of to find the most distinctive probe. In an optimal case, a single packet can be used to classify all hosts. In contrast to Vetterl et al., we use *a set of distinctive packets* to increase the resilience of our scanner to unknown implementations. Our request set is still small enough to enable scan speeds high enough to perform an internet-wide scan.

Where applicable, we also fingerprint the TLS stack used by the respective implementation. Furthermore, our fingerprinting tool does not consider the specifics of the operating system TCP stack. Some fingerprinted implementations also support other operating systems such as Linux.

We created two separate tools to create and compare fingerprints of *known* RDP and SMB implementations respectively. We created the fingerprints on our lab instances of all mentioned honeypots and benign Windows installations. Afterwards, we tested the fingerprints on real-world instances of Windows Server 2012, 2016, 2019 in the Amazon EC2 cloud and Microsoft Azure. We found that the Windows end-user versions share RDP and SMB server code with the server versions (see Table 2).

4.1. Packet Similarity Scoring

In order to check for the similarity of the collected packet exchanges, we developed a parser for the SMB and RDP protocol which tokenizes a single response

TABLE 2. INDISTINGUISHABLE VERSIONS OF WINDOWS

End-user	Server
XP	2003
7	2008R2
8	2012R2
10	2016, 2019

into multiple labeled fields. The fields of two responses are compared by checking if their values are identical, in a certain range or have certain bits enabled. Fields containing random values, length indicators (where different lengths are allowed), timestamps or fields containing configurable values are ignored during comparison. The similarity score is eventually set to be $\frac{1}{1+n}$ where n is the number of differing fields. Thereby, the similarity decreases the more fields inside the responses are found to differ.

One notable exception is the parsing of TLS traffic occurring inside RDP traffic. For the sake of simplicity, we only partially parse the TLS traffic and convert it into a single synthetic tokenized packet. This synthetic packet contains all relevant characteristics of the TLS handshake mentioned in Section 3.4.

4.2. Differential Fuzzing for Fingerprinting

Equipped with a metric to calculate the similarity of two packets, we can build a small fuzzer upon these. The fuzzer constructs a packet, sends it to each implementation, and utilizes the similarity scoring to decide if a notable difference between the answers exists. The responses are then compared to each other according to the similarity metric mentioned above. The request packet generating the set of responses least similar to each other is selected as the most distinctive probe.

For RDP, we utilize a bit mutation strategy of the first packet. For SMB, we construct the set of all reasonable combinations of values inside the request packet fields to find the most distinctive probe. While this fuzzing approach is simple, it is sufficient to identify fields in the protocol causing distinguishable answers for each implementation.

4.3. Our RDP scanner

Based on out fuzzing, the protocol field with the most notable impact on the server responses is the proposed security mode of the client (see Sec. 3.1). Therefore, we establish four connections with the target host. Three of them are regular connection attempts, advertising all major security modes (`PROTOCOL_RDP`, `PROCOTOL_SSL`, and `PROTOCOL_HYBRID`). This is done because some implementations choose to downgrade from or directly refuse connection attempts advertising support for certain security modes. The fourth connection contains an invalid set length field, triggering different error handling behavior between *Heralding* and *RDPY* and all Windows versions. Regular Windows versions close the connection with a Connection Reset. *RDPY* closes the connection by sending a regular TCP FIN packet to the client and *Heralding* sends the client an RDP Negotation Failure Message.

Still, in many cases just collecting the first packet exchange for each connection does not provide enough information to differentiate between implementations. Therefore, depending on the negotiated connection type we try to either establish a TLS connection and send the next packet over TLS or send it unencrypted. Note that the TLS fingerprint of *Heralding* and *RDPY* is not affected if the implementations are operated on a non-Linux operating system. Both packages rely on OpenSSL and do not switch to the native operating system TLS stack.

If the server accepts our `PROTOCOL_HYBRID` attempt, the protocol proceeds with Network Level Authentication (NLA) after the TLS handshake. We send a regular SPNEGO packet, starting the exchange of security mechanisms of both parties, but abort the connection directly afterwards in order to avoid supplying credentials for ethical reasons (see Sec. 5.1).

In case of a `PROTOCOL_RDP` or `PROTOCOL_SSL` connection, we proceed with the second step and send an encrypted or unencrypted version (dependent on the security mode setting) of the client capabilities which we recorded as well from the standard Windows RDP client.

We found that all Windows Servers which can be rented in the Amazon EC2 and Azure cloud have NLA enabled by default.

Because of the limited information we are able to obtain in an NLA enforced environment, it is impossible to distinguish Windows 10, Windows 8 and the different versions of the Windows Server 2012, 2016, 2019 solely on the characteristics of exchanged RDP packets.

The flag field of the RDP Negotiation Response PDU has two flags which are only available in recent RDP versions. If NLA login is not enforced, we can find more distinguishing features in the second round trip of the protocol. We show an example of which fields can vary in the responses of the second round trip in Table 7. It shows the relevant parts of the reference responses to the `PROTOCOL_RDP` packet. Packet headers that do not contribute to the fingerprint are intentionally left blank. Some implementations, e.g., Windows 10, do not respond to the second packet because they discontinue the connection after receiving it, which also counts as distinguishing behavior.

4.4. Our SMB scanner

We found the following fingerprintable implementation pitfalls for SMB: If no common dialect can be negotiated between client and server, different implementations have different ways of discontinuing the connection, some sending error codes while some close the connection without sending additional data. Furthermore, our fuzzer found different behavior by setting reserved fields to non-zero values or by setting other fields to values that are not reasonable in the current context. For fields inside the request which have corresponding fields in the response, it is up to the implementation whether such atypical values are ignored by mirroring them or by resetting them to zero.

Lastly, another valuable factor contributing to the fingerprint of an implementation is the combination of the *Native OS* and *Native LAN Manager* fields inside the second packet in an SMBv1 protocol sequence. It can

offer hints about the platform and operating system or environment which the server runs on and while honeypots can trivially replace this information, it does contribute to the overall fingerprint.

Therefore, our final SMB scanner establishes four connections to a scanned host, each having a different function:

1) An SMBv1 packet checking if the targeted host supports SMBv1 (as we consider this to be a requirement for a honeypot, see Sec. 5.5)
2) A packet that checks if the targeted host supports anonymous login
3) A packet checking for the support of the most recent version SMBv3
4) A specifically crafted packet that we found to trigger the most distinctive responses from implementations that we consider in our work

5. Internet-wide Scan

In order to test our ideas and developed tools, we conducted an internet-wide scan. We will describe its execution and its results in this section.

5.1. Ethical Considerations

We follow the principles of informed consent [30] and best practices [31]: we avoid the collection of personal or sensitive data and we try to avoid causing any harm to online servers during our active scans. None of the scan probes we send have affected our test machines negatively.

As described in Section 4, both the RDP and SMB protocol usually provide enough data for fingerprinting before we have to actually login into the machine. Since we do not provide any credentials to our targets, we do not consider it as a hacking attempt.

We took precautions to minimize the impact of our scans, following established practices as, for instance, described in [32]. In particular, we maintain a block list to avoid scanning systems that have in the past indicated to us that they want to be excluded from scans. Our abuse email address is published in the WHOIS and all abuse emails are forwarded to us by our IT department. We assess the impact of our scans in terms of potential harm to other systems and human beings, as proposed by the Menlo report [30]. We use a relatively low scanning rate to minimize any impact and respond immediately to complaints. All our scanning hosts run a web server which provides information on the scan including an email contact. We received 9 new requests regarding our scan activities, added IP addresses of eight requests to our block list and resolved one request, allowing us to continue scans as research project.

5.2. Setup

For our Internet-wide scan to find honeypots, we combine the implemented detectors with ZMap [32]. ZMap provides us the possibility to scan the complete IPv4 address space searching for hosts with open RDP (3389) or SMB (445) ports. If a host with an open port is detected, it is directly handed to the respective honeypot detector. This combination of ZMap as a stateless, fast Internet-wide scanning tool and our implemented stateful detector allows fast but informative scans.

Due to the properties of ZMap to focus on a single protocol and port to drastically decrease scan duration, we scanned SMB and RDP sequentially. Each scan was conducted with a rate of 20 000 packets per second to reduce the impact on target systems. For each scan, we use an up-to-date BGP dump from our local upstream provider with the complete set of reachable prefixes as a ZMap allow list to reduce the scan duration. This excludes all address ranges from the scan that are not announced by any AS and thus not reachable. Furthermore, we use a self-administered block list to prevent scanning targets which requested to be excluded from our scans. This block list was created over time based on abuse mails received by our research group to follow the ethical considerations described in Section 5.1. The list is solely built from requests to be excluded from active scans not including any external sources. The block list contains different addresses and prefixes covering 5.7M addresses in total. With the given setup and rate, each scan probes 2.8 billion IP addresses (67% of the complete IPv4 address space) and has a duration of around 37 hours.

5.3. Classification Results

During our internet-wide scan started on March 26th 2021 we discovered in total 7.5M hosts with an open RDP port. We could successfully assign 2.1M hosts to known RDP implementations. Regarding SMB, we found 2.7M hosts with an open port 445 during the scan started on the March 30th 2021. 1.1M hosts were categorized after a successful connection could be completed. For both protocols, we only label a connection with a specific implementation if the fingerprint matches *exactly*. Results can be seen in table 3 and we will describe the results for both protocols separately.

56.7% of the RDP scans and 57.2% of the SMB scans resulted in an error. For both protocols more than 97% of the errors were caused by a closed connection. Either a host is not reachable at all or it closes the connection after seeing our initial RDP connection attempt. This means most likely that those hosts do not provide the respective service behind the scanned port or that the port was erroneously reported by ZMap. In the remaining cases (less than 3%), we are not able to parse the answer of the host. We assume this is due to missing functionality in our parser or the host offers a different service. 14.2% of RDP hosts and 1.1% of the SMB hosts are not categorizable. Hosts are considered as not categorizable, if they do not match one of our recorded fingerprints.

RDP We were able to find 1207 matches with the *Heralding* and *RDPY* honeypots with 100% similarity. Predominantly, we found instances of *RDPY* in a configuration not offering `PROTOCOL_SSL` and always falling back to standard RDP security. Furthermore, we observed that the RDP port is frequently used for non RDP services. If a response packet of a host does not appear to be a valid TPKT, we classified it as non-RDP. Investigating this response class, we found several hosts answering with `HTTP/1.1 400 Bad request` indicating an HTTP

TABLE 3. SCAN RESULTS OF OUR INTERNET WIDE SCAN.

Category	RDP	SMB
Total ZMap results[1]	7 577 919	2 704 250
Categorizable[2]	2 125 428	1 125 838
Regular Implementations	1 940 159	1 124 317
Windows 8 & 10[3]	1 401 465	96 361
SChannel	1 401 452	
non-SChannel	13	
Windows 10 (no NLA)	96 003	
SChannel	96 003	
Windows 8 (no NLA)	35 126	
SChannel	35 126	
Windows 7	357 179	685 701
No Data	296 065	
SChannel	61 113	
non-SChannel	1	
Windows XP	36 823	3834
No data	36 823	
XRDP	13 355	
non-SChannel	13 355	
VirtualBox	208	
No data	208	
Samba 3.5.6		153 071
Samba 4.10.0		181 931
YNQ		2741
Misc. implementations[4]		678
Honeypots	1207	1521
Heralding	84	
non-SChannel	84	
RDPY	50	
OpenSSL	50	
RDPY (no TLS)	1073	
No data	1073	
Dionaea		1461
impacket		60
Non RDP/SMB protocols	245 300	
HTTP	185 948	
SSH	59 352	
Uncategorizable[2]	1 080 773	31 152
Errors	4 310 480	1 547 260
Unparseable	11 649	36 395
Connection closed[9]	3 127 932	1 419 216
No connectivity[8]	1 170 899	91 649

[1] ZMap only reports hosts with an open port. An open port is no proof the respective service is also provided.
[2] Hosts are only labeled with a classification if the fingerprint shows an *exact* match.
[3] and Windows Server 2012R2, 2016 and 2019
[4] Combined set of different rare implementations.
[5] Unknown Non-RDP protocol
[7] Hosts which did not respond to all of our packets preventing an exact classification
[8] Hosts without any response to our scanner despite being reported by ZMap
[9] Hosts which we established a connection with but closed the connection before sending any data

TABLE 4. TOP 10 AUTONOMOUS SYSTEMS HOSTING HONEYPOTS

CO	ASN	Organization	SMB	RDP	**Total**
US	16509	AMAZON	232	167	399
US	20473	CHOOPA	126	95	221
US	14061	DIGITALOCEAN	102	90	192
DE	197540	netcup	66	72	138
TW	1659	TANet	131	1	132
US	8075	MICROSOFT	48	25	73
US	63949	Linode	33	37	70
US	14618	AMAZON	41	28	69
US	15169	GOOGLE	35	32	67
US	22773	Cox Communications	50	3	53

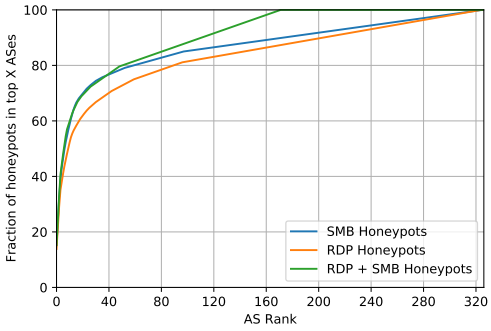

Figure 2. AS distribution of honeypot addresses

server. Moreover, we found many SSH banners sent in response to our probe indicating an SSH server.

We found the RDP stacks of Windows 8 and Windows 10, which are also used in Windows Server 2012 and 2019 respectively, predominantly used. In about 1.4M cases the hosts enforce the use of Network Layer Authentication (see Section 3.1) which gives our scanner only limited opportunity for fingerprinting. In almost all cases, the Windows hosts have a fingerprint which indicates the use of *S-Channel* the Microsoft Windows TLS implementation. However, a few hosts show perfect match with Windows, but have a fingerprint indicating the use of a non-Microsoft TLS library.

We also found Windows 7 and XP hosts unwilling to negotiate a TLS based connection with our scanner, but downgrading us to Standard RDP security for unclear reasons. This is marked in Table 3 as *No Data*.

SMB 1521 hosts have responded to our probe packets in exactly the same way as our lab instances of *Dionaea* and *Impacket*. With 60%, a majority of hosts that can be classified use Windows 7 followed by Samba 4.10.0 (16.3%) and Samba 3.5.6 (14.3%). Further analyzing the set of values we found inside the *NT Lan Manager* field, we discovered numerous rarely appearing and some almost undocumented implementations of the SMB protocol such as smbx (Apple), Alfresco CIFS and SXLM for which we had no fingerprint from lab tests. Since we did not expect any honeypot to mimic them, we categorized them as *Misc. implementations* without further comparison of their fingerprint.

RDP + SMB The two IP address sets of identified RDP and SMB honeypots are not distinct. With 606 mutual addresses, around 40% combine honeypots for the RDP as well as for the SMB protocol on the same host. This high overlap supports our finding that the presented methodology is able to detect honeypots. Selective searching of the hosts on portals like Censys.io or Shodan.io reveals that these hosts usually offer services associated with the honeypots *Dionaea* and *cowrie*, a honeypot for SSH and Telnet. We found eight hosts that mix an RDP honeypot with an *Impacket* installation, supporting our argument that *Impacket* is indeed used for building up honeypots.

Furthermore, hosts of 36 RDP honeypots have an open SMB port which can not be classified as a honeypot. Out of these, 35 SMB connection attempts ended in a premature session exit, indicating an unresponsive service. The one remaining host is classified as *uncategorized*, however the handshake sequence has some similarity with

Windows XP, which could indicate an unknown honeypot implementation. Vice versa, 87 SMB honeypots additionally have an open RDP port reported by ZMap. In 77 cases our scanner was not able to successfully establish a RDP connection, in eight cases a non-RDP service answered our probe and two hosts have not been classified.

5.4. AS Coverage

To highlight the widespread usage of detectable honeypots by a multitude of different organizations and providers, we analyze the distribution of honeypots across autonomous systems (AS). Figure 2 shows the distribution of all identified honeypots across ASes. 50% of all detected SMB honeypots are located on only 12 ASes but the remaining 50% are spread across 314 ASes. RDP honeypots are located in more ASes with a total coverage of 325 but with 50% located in 10 ASes, a majority can again be found in a small subset. Table 4 lists the top 10 ASes based on the total amount (SMB and RDP) of honeypots. It can be seen that multiple listed organizations are mainly cloud providers, (e.g., Amazon, DigitalOcean, and Netcup) and research networks (AS1659 is the Taiwan Academic Network). While some honeypots might be hosted by the cloud hosting organizations themselves, a majority is most likely set up by the provider's customers. Some of these hosting providers are also known to be scanning and research friendly. That might be an explanation for smaller hosters appearing in this top list. Based on these results, we infer that these types of honeypots are a widely used tool across various ASes.

5.5. Result Validation

In the following, we will describe our attempts to validate our scanning results by using additional indicators, because of the lack of ground-truth data for the Internet. We already observed in Section 5.3 that hosts offering both analyzed services are highly likely to have both services classified as honeypot. This is a clear indicator that these hosts are honeypots.

Additionally, we automatically connect to hosts of each category using established open-source tools and collect their diagnostic data where possible. If an automatic validation is not possible for a category, we fall back to manual methods. Table 5 summarizes our validation results. Unresponsive machines were most likely taken offline during the time period between scan and verification. We miss-classified less than 1% of responsive hosts. Additionally, *none* of the hosts in the honeypot group was miss-classified.

5.5.1. RDP Validation. First, we check if hosts being classified as normal Windows machines have been labeled with the correct Windows version. To achieve that, we utilize *rdesktop*[3] to obtain a screenshot of 100 target hosts in each category in an automated way. This is only possible for hosts not enforcing NLA. The captured screenshots of the login screens can then be quickly checked by a human analyst if they match the respective version of Windows.

3. https://github.com/rdesktop/rdesktop

TABLE 5. VERIFICATION RESULTS

	Correct	Incorrect	Unresponsive / Error
RDP	**448**	**5**	**127**
Heralding [1]	29	0	1
RDPY [1]	29	0	1
RDPY (no TLS)[1]	18	0	12
Windows 10 (no NLA)	89	1	10
Windows 8 (no NLA)	77	0	23
Windows 7 (no NLA)	58	0	32
Windows Server 2003	55	4	41
XRDP	93	0	7
SMB	**649**	**0**	**863**
Dionaea	628	0 (9)[2]	824
Impacket [1]	21	0	39
Total	**1097**	**5**	**951**

[1] Only manually verified
[2] Manual additional test deemed all hosts as honeypots

The results are shown in Table 5, only one Windows 10 host is misclassified.

Second, we check if the honeypots have been labeled correctly by our scanner tooling. Unfortunately, the screen scraping approach is not viable to detect the selected honeypots. As *RDPY* only replays pre-recorded sessions, a human analyst will quickly notice that keypresses are not displayed and that he is viewing how somebody else is using the machine. However, to a screen scraping tool, the host will appear as a benign machine. *Heralding* is unable to go through the whole connection sequence and will terminate the connection early and never continue to a point where a user can see the Windows login screen. Therefore, we connect to 30 instances of each honeypot category manually and analyze the exchanged packets and check if the machine reacts to mouse movements.

During our manual verification process, we observed that many of the *Heralding* honeypots use the same subject and issuer names in the TLS certificates and that they show the abnormal protocol behavior described above. This strengthens our assumption that these hosts are indeed *Heralding* honeypots. Multiple *RDPY* instances share desktop session replays they present to the connecting user. Foremost, we observed a login screen of Windows 8/10 fading in, but a few instances presented us with a full Windows 7 desktop session where somebody moves the mouse on the desktop while our test user did not interact with the system.

We also performed manual validation on the small set of non-SChannel hosts being classified as regular Windows instances. As our classifier showed good precision for regular hosts, we believe that a real windows host is involved in the connection. We conclude that the connection is interrupted by a MitM-box. The non-SChannel Windows machines appear to be fully functional. Dialogs, for example, the accessibility menus, react normally.

5.5.2. SMB Validation. We connect to each host using the tool *smbclient*[4], which is capable of showing the offered file shares of an SMB server. We observed that the shares offered to a client by *Dionaea* are configurable in the most recent versions, but we assume many users

4. https://www.samba.org/samba/docs/current/man-html/smbclient.1.html

might not change the default values. We connected to all *Dionaea* honeypots and found that 628 hosts have not switched the offered default shares and displayed comments. Nine hosts presented a different list of file shares. We manually connected to these instances and found that the share names are only slightly altered and that the connection dies with unusual error codes when we browse around. For the validation of *Impacket* instances, we observed that the amount of free disk space that is displayed to the user while browsing through offered files is a hard coded value in the source code. Therefore, we employ this as a detection feature.

6. Attacker Behavior Analysis

To study how attackers react to detectable honeypots and how they adopt their behavior, we deployed multiple instances of RDP honeypots at various cloud providers. We captured their traffic and analyzed the performed attacks.

6.1. Setup

Our honeypots were deployed in the Amazon EC2 cloud from June 17 to July 19, 2021 (34 days). To reduce the influence of the previous owner of the IP address we received from Amazon, we assigned a new IPv4 address obtained from the Amazon EC2 pool to each of our honeypots every 7 days. All traffic was captured and recorded in PCAP files using the EC2 infrastructure during our experiment. Because RDP is already the default remote acccess protocol for Windows machines in the EC2 cloud and our scans identified more open RDP servers than SMB servers we decided to only set up RDP honeypots. For the comparison, we use an unmodified Windows Server 2019, an instance of *RDPY* (the most common RDP honeypot detected during our scans) and two instances of *PyRDP* [33], a pure MitM honeypot for the Windows RDP protocol (one in the default configuration, and one in "no downgrade" (ND) mode to decrease the detection surface).

To analyze the potential impact of the high concentration of honeypots in the autonomous systems of a few cloud providers, we rented a single Linux host in AS 197540 (netcup), AS 14061 (DIGITALOCEAN) and AS 6724 (STRATO) each, and set up a host in a research network, namely AS 209335 (TUM). On the rented hosts, we use `iptables NAT` to transparently forward the RDP traffic to our EC2 installation. In order to evenly spread the traffic from these proxies over the EC2 honeypots, each incoming connection is sorted into a hash bucket (`iptables HMARK`) based on the source IP of the client. Therefore, the IP appears to be a different honeypot depending on the client IP. However, as long as a client keeps its IP address, it will always connect to the same honeypot instance.

6.2. Results

We found that our honeypots seem primarily subject to credential guessing attacks, where an attacker tries out different well known usernames and weak passwords.

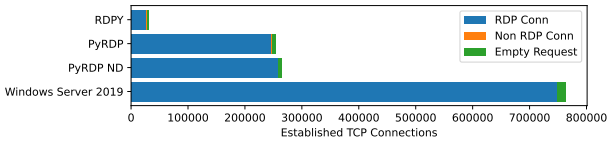

Figure 3. Traffic distribution between our honeypot implementations

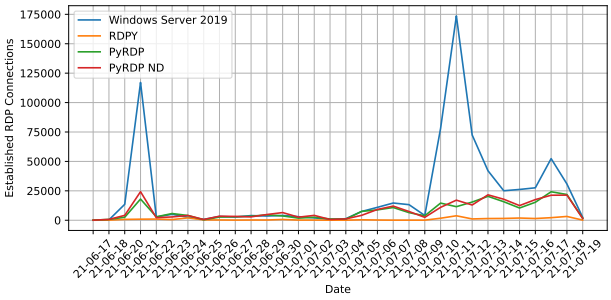

Figure 4. Traffic distribution between our honeypot implementations

Fig. 3 visualizes the distribution of incoming connections between our machines. We categorized the incoming traffic for each honeypot into three categories: (1) *RDP traffic*, which is indicated by a TPKT header starting with a `0x3` byte, (2) an *empty request*, which is a completed TCP three-way handshake resulting in a connection that is immediately closed afterwards, and (3) *non RDP traffic*, where a client sends a non-empty request not matching the criteria of (1). However, all honeypots received less than 1% of (2) *empty requests* and less than 0.2% of (3) *non-RDP traffic* on the RDP port. We removed these connections from the dataset prior to analysis. We observed connections from *Censys.io* and *Shodan.io* of type (2) and we expect the remaining to be opened by lesser known Internet scanning services.

Our non-honeypot Windows Server 2019 seems to be preferred by attackers over our remaining honeypots. Fig. 4 illustrates the number of RDP connections to our honeypots over time. The traffic originates primarily from three attacks on our honeypot infrastructure on June 20, July 10, and July 16, 2021.

The first attack on 20th of June is mainly caused by a massive amount of connections originating from a /24 IP subnet from a single autonomous system (ASN 49505, Selctel, located in Russia). The honeypots on the Amazon EC2 instance were attacked directly via their respective IPs and not via one of our forwarding proxies.

About 40% of the connections established on July 10 originate from two IPs that are likewise located in the same AS and that have not communicated with any other honeypot. Furthermore, about 6% of the connections were proxied from AS 197540 (netcup) on this day and spread almost evenly across all honeypots with the Windows Server 2019 in a slight lead.

The traffic peak on July 16 consists of about 23% out of connection attempts from ASN 209335. Despite proxied traffic being balanced across targets, most connections (63 %) were made to the Windows Server 2019, followed by *PyRDP* (13%) and *RDPY* (1%). Unfortunately, our proxy setup does not allow us to further trace the connections back to the original IPs of the attacker.

The data suggests that honeypots are indeed avoided by attackers even if they are just doing credential stuffing attacks. We observed that multiple hosts in an IP address range seem to collaborate, i.e., a host *A* sends an initial probe, terminates the connection and processes our answer internally. Afterward, host *B* gets notified by *A* if the host is of interest and performs further scanning or an attack. This pattern is provably present for probes we received from *Censys.io* as the scanning machines have meaningful RDNS records and the the targeted attacks on two honeypots on July 10 suggests that attacking host act likewise. For a more in depth analysis, this experiment needs to be repeated at larger scale.

7. Discussion

Our analysis relies on active internet-wide scans and is therefore affected by inaccuracies due to the heterogeneity of the Internet and the impact of a single vantage point. Data from internet-wide scans always shows a certain degree of noise due to the fact that the Internet is built from individually managed components. This impacts the data quality and classification rate as shown in Table 3. While some of the hosts reported by ZMap can not be classified due to connection timeouts or unspecified behavior, others even provide different services on the scanned port. Furthermore, internet-wide scans from a single vantage point might not be able to reach the complete Internet as shown recently by Wan et al. [34], which can affect our results.

Nevertheless, we avoid inaccuracies with strict constraints in our classification methodology. Our fingerprints ignore all configurable options in the respective implementations we are aware of. A mismatch between fingerprints therefore indicates a difference in implementations, rather than in configurations. On the one hand, this allows us to provide a meaningful and accurate lower bound of existing honeypots spread across a multitude of ASes and organizations. On the other hand, we consider many hosts to be *uncategorizable*.

A few of the classifications might be caused by the heterogeneity of the Internet. Other factors include differences caused by different patch levels of the SMB and RDP implementation. In samples we took from the *Uncategorizable* group, we find a large amount of Windows hosts with varying operating system versions for which we rejected the correct classification label because of our strong matching criteria. However, we believe this does not impair our ability to find known honeypot implementations as their fingerprint is quite unique and not subject to much change. The *RDPY* source code has not changed in its protocol handling for 6 years, and *Dionaea* saw its latest change in the module responsible for the handling of SMB back in 2017.

7.1. What about high interaction honeypots?

We believe that hosts that we classified as *Non-SChannel Windows RDP Hosts* are likely high interaction honeypots, where the TLS connection is interrupted by a non-SChannel MitM-box in order to observe the traffic. Unfortunately, 13 exact matches with Windows enforce NLA, so that we could not further investigate if the classification is correct without ethical issues caused by providing login credentials.

During analysis of our Internet scans, we became aware of the Wallix Redemption MitM-box for RDP[5] and the *PyRDP* [33] MitM honeypot. Both use OpenSSL for re-encryption of the traffic, with a clearly distinguishable fingerprint. The one non-NLA-enforced host could be such a *PyRDP* honeypot: It reacts to user input in a meaningful way (e.g dialog boxes and keyboard actions are processed), and allows "normal" interaction with the system (e.g. the *Disconnect* button actually disconnects the session) despite the abnormal SSL stack.

If we relax the 100% similarity constraint, we were able to find other hosts that behave like Windows machines, do not require login on connect, and are attached to the AS of DigitalOcean. To our knowledge DigitalOcean does not offer or officially support Windows machines. Furthermore, we found similar implementations in their network offering different Windows versions with the same TLS certificate. In contrast to the 13 hosts mentioned above, these hosts downgrade the protocol security level from `PROTOCOL_HYBRID` to `PROTOCOL_SSL`, which we have never observed in a unmodified Windows version during our tests, being another indicator for an abnormal RDP stack.

We believe that our data set has more interesting high-interaction honeypots in the *Uncategorized Hosts* section. However, our existing results already show that this detection method is viable.

7.2. Why should we care?

Our experiments confirm that also commonly used RDP and SMB honeypots are fingerprintable and that a fingerprint can easily be created. In Section 6 we conducted a study to check if fingerprinting techniques are applied by attackers. While our experiment should be repeated to increase confidence, our collected data suggests that this is the case even if we have not observed their exact fingerprinting technique. It is also worth noting that both *PyRDP* instances (with we would classify as a high interaction honeypot) received less attack traffic, indicating that more care needs to be taken with regards to details like the *abnormal* TLS stack.

We suggest the following improvements to honeypot implementers and operators:

Employ differential fuzzing yourself. Implementers can use the presented fingerprinting methods themselves in order to minimize the observable behavioral gap between their honeypot and the original protocol implementation.

Be careful about the TLS stack implementation. If you are operating a MitM box in order to monitor encrypted TLS traffic in a protocol of interest, select a TLS implementation that matches the target machine. For example, SChannel has a standardized API on Windows, which allows third-party code to use it. This seems to not be widely known.

8. Conclusion

We demonstrated a viable approach to detect three popular honeypots and one popular framework to create

5. https://github.com/wallix/redemption

SMB honeypots (*Impacket*) on the Internet. Our validation has shown that exact fingerprint matches allow identification of a host with high accuracy.

We reused existing concepts such as the use of erroneous requests and the method of a most distinctive probe to separate honeypots from regular implementations. However, our results indicate that TLS fingerprints are well suited to be combined with a honeypot detector. The results of our experiment on self-hosted honeypots suggest that attackers also employ fingerprinting methods to avoid both low- and high-interaction honeypots.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable feedback. We would like to thank Visuality Systems for providing us with a free license of their product YNQ and for the inspiring discussions about the SMB protocol. This work was partially funded by the German Federal Ministry of Education and Research under the project PRIMEnet, grant 16KIS1370.

References

[1] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. IEEE, 2005, pp. 29–36.

[2] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G.-J. Ahn, "Honeyplc: A next-generation honeypot for industrial control systems," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 279–291.

[3] S. Shan, E. Wenger, B. Wang, B. Li, H. Zheng, and B. Y. Zhao, "Gotta catch'em all: Using honeypots to catch adversarial attacks on neural networks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 67–83.

[4] Shodan.io, "Honeyscore," https://honeyscore.shodan.io/.

[5] ShodanHQ, "Honeyscore announcement," https://twitter.com/shodanhq/status/1311661444765806593, 10 2020, Twitter.

[6] R. Blog, "Nicer protocol deep dive: Internet exposure of remote desktop (rdp)," https://blog.rapid7.com/2020/10/23/nicer-protocol-deep-dive-internet-exposure-of-remote-desktop-rdp/, 2020.

[7] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. J. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you… oh wait. an internet-wide view of self-revealing honeypots," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 134–143.

[8] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low-and medium-interaction honeypots at internet scale," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.

[9] D. Sysman, G. Evron, and I. Sher, "Breaking honeypots for fun and profit," *Black hat USA*, 2015.

[10] ——, "Breaking honeypots for fun and profit," *32c3 - Chaos Communication Congress*, 2015.

[11] J. Uitto, S. Rauti, S. Laurén, and V. Leppänen, "A survey on anti-honeypot and anti-introspection methods," in *World Conference on Information Systems and Technologies*. Springer, 2017, pp. 125–134.

[12] N. Krawetz, "Anti-honeypot technology," *IEEE Security & Privacy*, vol. 2, no. 1, pp. 76–79, 2004.

[13] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham, "On recognizing virtual honeypots and countermeasures," in *2006 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*. IEEE, 2006, pp. 211–218.

[14] S. Mukkamala, K. Yendrapalli, R. Basnet, M. Shankarapani, and A. Sung, "Detection of virtual environments and low interaction honeypots," in *2007 IEEE SMC Information Assurance and Security Workshop*. IEEE, 2007, pp. 92–98.

[15] L. Metongnon and R. Sadre, "Beyond telnet: Prevalence of iot protocols in telescope and honeypot measurements," in *Proceedings of the 2018 workshop on traffic measurements for cybersecurity*, 2018, pp. 21–26.

[16] V. Ghiëtte, H. Griffioen, and C. Doerr, "Fingerprinting tooling used for {SSH} compromisation attempts," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 61–71.

[17] J. van Eesteren, "Remote Desktop Protocol (RDP) 10 AVC/H.264 improvements in Windows 10 and Windows Server 2016 Technical Preview," https://techcommunity.microsoft.com/t5/microsoft-security-and/remote-desktop-protocol-rdp-10-avc-h-264-improvements-in-windows/ba-p/249588, 2016.

[18] *Remote Desktop Protocol: Basic Connectivity and Graphics Remoting*, Microsoft, 8 2020, Revision 53.0.

[19] GitHub, "Heralding," https://github.com/johnnykv/heralding, 2020.

[20] ——, "Rdpy," https://github.com/citronneur/rdpy, 2020.

[21] ——, "Dionaea," https://github.com/DinoTools/dionaea, 2020.

[22] SecureAuth, "Impacket," https://www.secureauth.com/labs/open-source-tools/impacket/, 2020.

[23] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," *arXiv preprint arXiv:1608.06249*, 2016.

[24] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, "Coming of Age: A Longitudinal Study of TLS Deployment," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 415–428.

[25] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The Security Impact of HTTPS Interception," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/security-impact-https-interception/

[26] J. Althouse. TLS Fingerprinting with JA3 and JA3S. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967.

[27] Cisco. Mercury: network fingerprinting and packet metadata capture. https://github.com/cisco/mercury.

[28] ——. Joy. https://github.com/cisco/joy.

[29] J. Althouse. Easily Identify Malicious Servers on the Internet with JARM. https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a?gi=3d3703067810.

[30] D. Dittrich, E. Kenneally *et al.*, "The menlo report: Ethical principles guiding information and communication technology research," *US Department of Homeland Security*, 2012.

[31] C. Partridge and M. Allman, "Addressing Ethical Considerations in Network Measurement Papers," *Communications of the ACM*, vol. 59, no. 10, Oct. 2016.

[32] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *USENIXSEC*, Washington, D.C., USA, 2013.

[33] GoSecure, "Pyrdp," https://github.com/GoSecure/pyrdp, 2018.

[34] G. Wan, L. Izhikevich, D. Adrian, K. Yoshioka, R. Holz, C. Rossow, and Z. Durumeric, "On the Origin of Scanning: The Impact of Location on Internet-Wide Scans," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020.

Appendix

TABLE 6. TOP 50 AUTONOMOUS SYSTEMS HOSTING HONEYPOTS

CO	ASN	Organization	SMB	RDP	**Total**
US	16509	AMAZON	232	167	399
US	20473	CHOOPA	126	95	221
US	14061	DIGITALOCEAN	102	90	192
DE	197540	netcup	66	72	138
TW	1659	TANet	131	1	132
US	8075	MICROSOFT	48	25	73
US	63949	Linode	33	37	70
US	14618	AMAZON	41	28	69
US	15169	GOOGLE	35	32	67
US	22773	Cox Communications	50	3	53
HK	135377	UCLOUD	26	25	51
FR	16276	OVH	17	22	39
CA	32613	IWEB	2	34	36
CN	132203	TENCENT	9	26	35
CA	25820	IT7NET	30	0	30
JP	2500	WIDE-BB WIDE Project	30	0	30
CN	45102	Alibaba	20	9	29
IT	137	GARR	12	16	28
JP	2497	Internet Initiative Japan	25	1	26
GB	9009	M247	12	10	22
DE	3320	DTAG	9	13	22
LT	56630	MELBICOM	8	10	18
JP	9370	SAKURA Internet Inc.	9	7	16
HK	136907	HUAWEI CLOUDS	9	3	12
US	7922	COMCAST	0	11	11
JP	2514	NTT PC Communications	6	5	11
JP	4713	NTT PC Communications	6	4	10
CN	45062	NETEASE	0	10	10
FR	12876	Online SAS	1	8	9
CZ	5588	GTS Central Europe	6	3	9
IT	3269	Telecom Italia	2	7	9
IN	4755	TATA Communications	4	5	9
MY	38182	Extreme Broadband	9	0	9
US	46562	PERFORMIVE	2	6	8
DE	51167	CONTABO	3	5	8
CA	31798	DATACITY	0	8	8
US	63473	HOSTHATCH	4	4	8
CA	136258	OBrainStorm Network Inc	4	4	8
EE	206804	ESTNOC	2	6	8
AU	133159	Mammoth Media Pty Ltd	4	4	8
RS	8400	TELEKOM SRBIJA	7	0	7
AR	263812	IPXON Networks	3	4	7
CN	4134	CHINANET	0	7	7
IR	58224	PJS	7	0	7
ES	39020	COMVIVE	3	3	6
GR	6799	OTENET	3	3	6
NL	6830	Liberty Global	2	4	6
JP	9355	NICT	1	5	6
HK	137280	Kingsoft cloud corporation	3	3	6
US	36352	COLOCROSSING	3	3	6

TABLE 7. RESPONSE COMPARISON FOR **PROTOCOL_RDP**

Field name	**XRDP**	**Win10**	**Win8**	**Win7**	**WinXP**	***RDPY***	***Heralding***
T.125 Conn. Resp.							
...							
Domain Parameters							
Max Channel IDs	22	✖	34	34	✖	22	✖
...							
RDP Server Data							
Server Core Data							
...							
Length	12	✖	16	12	✖	16	✖
Early Capability Fl.	✖	✖	1	✖	✖	0	✖
...							