A Survey of Contemporary Open-Source Honeypots, Frameworks, and Tools

Niclas Ilg[a,c,*], Paul Duplys[b], Dominik Sisejkovic[a], Michael Menth[c]

[a] *Robert Bosch GmbH, Corporate Research, Renningen, Germany*
[b] *Robert Bosch GmbH, Sector Mobility, Ludwigsburg, Germany*
[c] *University of Tuebingen, Chair of Communication Networks, Tuebingen, Germany*

Abstract

Automated attacks allow adversaries to exploit vulnerabilities in enterprise IT systems at short notice. To identify such attacks as well as new cyber security threats, defenders use honeypot systems; these monitored decoy resources mimic legitimate devices to entice adversaries. The domain of enterprise IT honeypots has been an active area of development and research, especially in the open-source community. In this work, we survey open-source honeypots, honeypot frameworks, and tools that help to develop or discover honeypot deployments. In contrast to existing surveys, our work provides a detailed discussion of the honeypots' system architecture, software architecture, and cloud-native deployment options. In addition, we cover the most recent academic research in honeypot detection and evasion techniques, and discuss how these advances impact current open-source honeypots. This work helps the reader to make an educated choice when selecting a honeypot for deployment or further development.

Keywords: honeypot, honeypot framework, cyber security, threat intelligence

1. Introduction

Ransomware, malware, and zero-day exploits are the main cyber threats identified by the European Union Agency for Cybersecurity in its latest cyber security report (European Union Agency for Cybersecurity, 2022). Moreover, the time to act on zero-day exploits and vulnerabilities is getting ever shorter: automated scanners are reported to find vulnerable devices within 15 minutes of the vulnerability disclosure (Palo Alto Networks, 2022) in the Common Vulnerabilities and Exposures (CVE) database (NIST, 2023). As a result, it is critical to understand emerging threats and actively monitor the attack landscape. Honeypots are an established tool for this purpose in enterprise IT.

Honeypots are decoy resources; they appear to be genuine systems but are actually dummies used to gather threat intelligence. To an attacker, a honeypot behaves like a real service or device, in reality, however, it is monitored by the defenders. Whether used as an early warning system for zero-day exploits or to collect malware samples from botnets (Bajpai et al., 2020), honeypots help strengthen defenses against cyber attacks. For example, they are deployed as an additional layer of intrusion detection, to support efforts against Distributed Denial-of-Service (DDoS) attacks, to research the approaches of adversaries, or to understand the workings of new malware. For this reason, the Internet of Things (IoT) and the Industrial Internet of Things (IIoT) also adopted the use of honeypots (Vetterl and Clayton, 2019; Franco et al., 2021).

In particular, open-source software contributes to the evolution of honeypots and their adoption in new domains. Community-driven development with many contributors results in honeypots with different architectures, capabilities, and features. While compiling this work, we encountered a wide variety of open-source honeypots; some offer an extensive list of network protocols, others specialize in a single service, some implement simple, scripted behavior, and others emulate entire systems. We not only highlight the honeypots' key features, but also provide an in-depth discussion of the underlying architecture, software, and deployment of each solution. Since we want to help the reader choose a

*Corresponding author.
E-mail address: Niclas.Ilg@bosch.com

honeypot for development or research, we focus on open-source solutions; these allow for free feature development and a detailed review of the honeypot's characteristics. Because of advances in honeypot detection and evasion techniques, we also cover tools that help with honeypot detection as well as the latest research in these areas. More precisely, our contribution is twofold:

- We provide an overview of existing honeypot solutions, frameworks, and tools that are either currently maintained or have unique capabilities. We focus on open-source honeypots and related results from academic research.
- We highlight challenges in honeypot development as well as deployment, and discuss how they are (partially) solved by open-source solutions and academic work.

Table 1: Comparison of honeypot surveys based on the reviewed honeypot characteristics. Tick marks in brackets indicate that characteristics were not considered for all honeypots, or only partially.

Survey	Focus	Honeypot services	Honeypot architecture	Deployment options	Logging	Detection	Proprietary solutions
Nawrocki et al. (2016)	Attack analysis	✓	(✓)	-	-	-	✓
Fan et al. (2017)	Taxonomy analysis	(✓)	-	✓	-	(✓)	✓
Dalamagkas et al. (2019)	Honeynets in smart grids	✓	✓	-	✓	-	-
Franco et al. (2021)	IoT, IIoT, and CPS	✓	-	(✓)	✓	-	-
This work (2023)	**Open-source enterprise IT**	✓	✓	✓	✓	✓	-

1.1. Comparison with Prior Honeypot Surveys

Honeypots are well known in enterprise IT. The first honeypot was implemented more than 20 years ago (Spitzner, 2003). Since then, honeypots have evolved from simple Linux installations (Spitzner, 2003) to lightweight cloud resources that mimic all kinds of different services and devices (Jose Nazario, 2022; Deutsche Telekom Security GmbH, 2023; Thinkst Applied Research, 2023).

Several surveys on honeypots have been published in recent years. Table 1 compares past honeypot surveys with this work. In addition to the focused domain, we also compare the honeypots' characteristics that were examined in the respective papers. Surveys especially separate in the depth of the honeypot review. Overviews with a more shallow discussion on honeypot architecture, however, offer a larger amount of honeypots and academic solutions. Although academic solutions are not proprietary, in some cases the source code is closed-source.

The two most recently published surveys focus on industrial production networks (Dalamagkas et al., 2019) and on IoT, IIoT, and Cyber-Physical Systems (CPS) (Franco et al., 2021). This is an area of growing interest as botnet activity and other automated attacks on these devices have increased in recent years. While some network protocols are common for enterprise IT and (I)IoT, devices in IoT, IIoT, and CPS have limited computing resources and run different services specific to these domains—two focal points that a honeypot must emulate realistically.

The authors in (Nawrocki et al., 2016) and (Fan et al., 2017) present solutions with a focus on enterprise IT. However, many of the surveyed honeypots are long outdated or have received major functionality updates since then. Moreover, these surveys include proprietary solutions without the possibility of a detailed discussion on their architecture and implementation.

In this survey, we take an in-depth look at enterprise IT honeypots. This choice is supported by multiple motivators. First, enterprise IT honeypots have witnessed strong momentum in research and development in recent years, especially in the open-source community. Second, the wide adoption of cloud-native technologies like Docker and Kubernetes now offers new honeypot deployment options. Finally, research on honeypot detection produced new ways to discover honeypots, which, in turn, forced honeypot developers and researchers to come up with novel ways of deception.

In contrast to existing surveys, our work covers new aspects regarding a honeypot's deceptiveness and, in addition, includes a detailed discussion on system architecture, software architecture, logging strategy, and deployment options of the surveyed honeypots.

1.2. Structure

The remainder of this work is structured as follows. Section 2 provides preliminaries on honeypots and honeypot detection. In Section 3, we survey open-source honeypots and honeypot frameworks

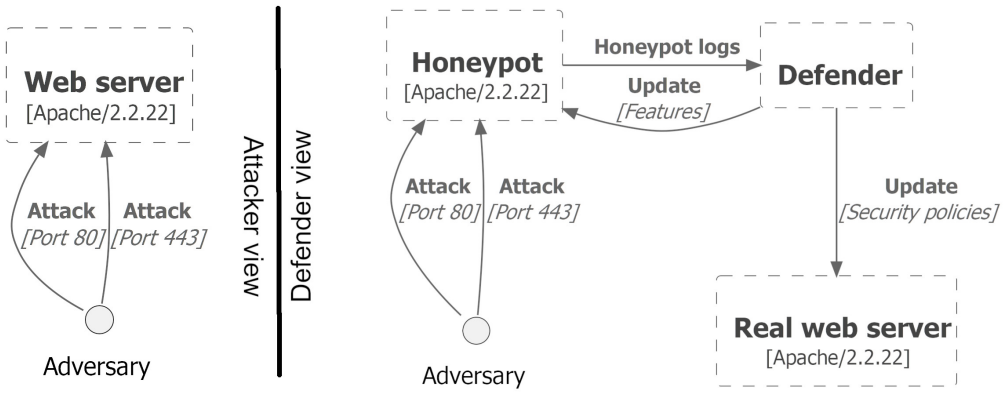


Figure 1: Attacker and defender view of a honeypot setup.

that are either currently maintained or provide unique capabilities. Furthermore, we discuss tools that assist in testing honeypots during development or deployment. In Section 4, we consider academic research on the detection and evasion of honeypots. The impact of honeypot detection is then discussed with regard to the surveyed honeypots in Section 5. Finally, Section 6 concludes the survey.

2. Preliminaries on Honeypots

Honeypots are resources that attempt to mimic a real computer system with the intention of being compromised by an attacker. These decoy systems are constantly monitored by the defenders and incoming attacks are recorded and analyzed. Typical applications of honeypots can range from individual services—many honeypots on the Internet offer only an Secure Shell (SSH) service—to entire web servers. Figure 1 shows a honeypot system from the attacker's and the defender's perspective. The adversary expects to attack a genuine system with, for example, DDoS attacks, SQL injections, or brute-force attacks. Meanwhile, defenders use the insights from captured attacks to strengthen their real deployments. Thus, honeypots are tools to gather intelligence, identify cybersecurity threats, and understand how adversaries would compromise the mimicked system (Stolfo et al., 2011). This allows defenders to make data-driven decisions on what security measures to invest in.

Depending on the expected attacks, different types of honeypots are deployed. In the following, we discuss different types and strategies of honeypots and their findings. Since honeypots rely on deception to convince an adversary to interact with the system (Sanders, 2020), another focus of this section is on honeypot detection.

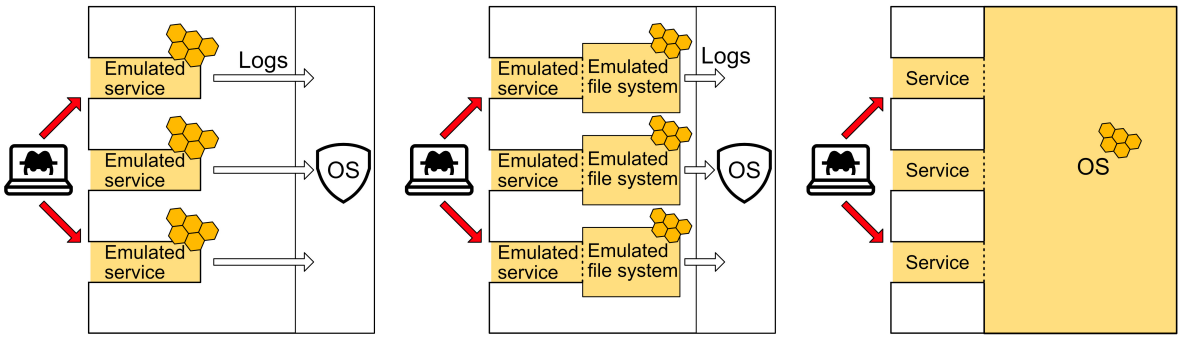
2.1. Honeypot Types

Honeypots are typically characterized by the interaction level a honeypot offers to the adversary. This interaction level varies from low to medium to high (Vetterl and Clayton, 2018).

Low-interaction Honeypots (LHs) provide the least amount of interaction for an adversary connecting to the system. They offer no Operating System (OS) to the attacker but rather a small number of attack paths, e.g., a log-in shell for a given service application. This kind of honeypot is predominantly used to catch credential brute-force attacks or monitor connection attempts. Figure 2a illustrates such an LH. The services are only implemented superficially and allow no further access to the underlying OS. All activity received by the emulated services is logged.

Medium-interaction Honeypots (MHs) still do not provide a real OS but can simulate a system shell to run commands on. Hence, these honeypots try to present a more attractive target and catch a greater scope of attacks. Depending on the depth of the simulated system, MHs cannot only run selected system commands but collect malware samples uploaded by the adversary. As seen in Figure 2b, the honeypot application is still separated from the operating system, making the development of a profound shell emulation a significant effort. LHs and MHs are useful tools for defenders to collect data on automated, large-scale attacks (Vetterl and Clayton, 2018). Further, an extensively emulated system shell can even trap human adversaries for short periods of time.

High-interaction Honeypots (HHs) have the highest odds of trapping a human adversary. On this interaction level, the honeypot has a real OS looking vulnerable to the outside world. The high-interaction possibilities allow for insight into attacker movement and activities on the system. However, these insights introduce risks; a vulnerable machine can also be used for subsequent attacks. Thus, an HH also includes the highest level of maintenance effort. As shown in Figure 2c, the OS is at the disposal of the attacker. There is no longer any separation between the honeypot and the OS.



(a) Emulated network services that are used as LH.

(b) MH containing an emulated system to track attacker movement.

(c) HHs deploy a real operating system for a realistic attack surface.

Figure 2: Different types of honeypots. Dotted lines illustrate a boundary intended to be breached by adversaries.

2.2. Deployment Scenarios

The purpose of deployment can be roughly split into two categories: research and production honeypots (Spitzner and Roesch, 2001). Research honeypots are mainly used to monitor the development of existing attacks and to get an early grasp on novel attack vectors. Figure 3a shows a honeypot deployed inside a Demilitarized Zone (DMZ). This gives the attackers access to the honeypot but prevents penetration into the internal network. Those honeypots can supply defenders with valuable information about the current attack landscape. In contrast, production honeypots (Figure 3b) are run to mimic a valuable component in a production setting. They can add an additional layer of intrusion detection as those honeypots are not part of the production process. In this way, every connection attempt can be assumed to be a hostile party with access to the internal production network.

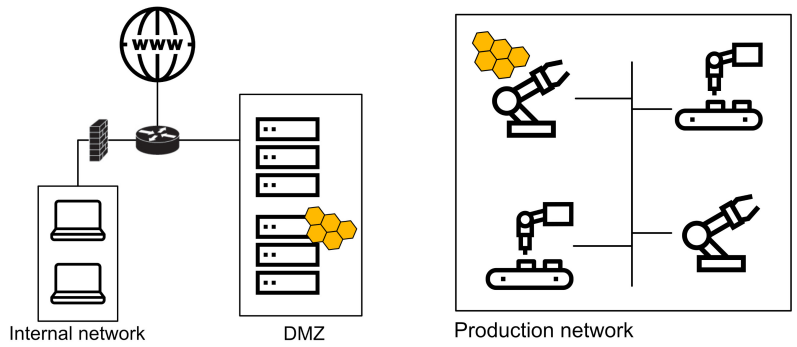
Multiple, interconnected honeypots are called a honeynet (Dalamagkas et al., 2019). Honeynets are helpful to improve the cover of production honeypots. While a single production machine without in- or out-going communication is not plausible, multiple honeypots can interchange protocol messages.

In addition, we can classify honeypots based on whether they run on a real server or use a virtual deployment option (Dalamagkas et al., 2019). On one hand, the deployment of honeypots on a server or production hardware results in the best possible imitation of the represented system. This creates realistic response times and system environments. On the other hand, virtual deployments on Virtual Machines (VMs) or inside containers are more flexible. This results in hardware independence and a light resource fingerprint, and allows for cheaper deployments, e.g., in a cloud environment. Build-in monitoring of virtual deployment techniques is another advantage, as tools like Virtual Machine Monitors (VMMs) can trace the attacker's actions (Asrigo et al., 2006).

2.3. Honeypot Findings

Deploying honeypots on the Internet can yield a lot of information about the threat landscape for selected protocols. This knowledge can be used to adjust existing security solutions. Honeypots are effective in collecting information about attacks because they have no value for legitimate clients. For every connection, it can be assumed that the connecting party has malicious intentions. As a result, one does not need to differentiate between legitimate and hostile connections. In the following, we take a look at honeypot findings sorted by interaction level.

Low-interaction honeypots detect scanning attempts as well as brute-force or dictionary attacks (Brown et al., 2012). New variants of vulnerabilities like Shellshock or Log4J can also be revealed by specialized low-interaction environments. While those low-interaction systems are useful to detect



(a) A research honeypot accessible from the Internet and used to monitor the threat landscape.

(b) Production honeypot placed within an internal production network to detect potential intruders.

Figure 3: Deployment scenarios for honeypots.

if there is interest in a target system, higher interaction levels are more suitable to reveal an attacker's intention.

Medium-interaction honeypots receive malware samples from different kinds of botnets (Zuzčák and Zenka, 2020) and command line inputs by automated and human adversaries. While some automated attacks try to inspect the target system—the CPU's core count or the kernel architecture provide valuable information—most botnets disregard the nature of their target and try to infect it. Malware that is used to harm vulnerable systems can range from simple viruses to rootkits and remote access trojans (Sethia and Jeyasekar, 2019).

High-interaction environments observe similar results. Malware infection and the installation of crypto miners are common findings. Moreover, attackers can try to use captured systems for further attacks; vulnerable devices are commonly turned into relays for spam E-Mail (Alata et al., 2006) or used for subsequent denial of service attacks.

2.4. Honeypot Detection

Honeypots are only useful if the attacking party is not aware of the real nature of their target system. If a honeypot is identified, adversaries simply avoid it. Apart from an unrealistic attack surface, there are multiple ways to identify a system as a possible honeypot from the outside: default configurations (Srinivasa et al., 2021a), timing attacks (Vetterl and Clayton, 2019), and service application fingerprinting (Vetterl and Clayton, 2018; Srinivasa et al., 2021a).

Default configurations. All honeypots examined in this paper come with a default configuration for instantaneous deployment. However, default configurations can cause detection. The default content of HTTP responses and static values in SSL/TLS certificates have led to the detection of honeypot instances. This unique content in default configurations or templates should be avoided by developers. As shown in (Srinivasa et al., 2021a), of 21,656 detected honeypot instances, only 351 honeypots did not run on the default configuration.

Timing attacks. Depending on the represented device and its original hardware, there may be a difference in response time between the real device and the honeypot. This is particularly the case when virtual deployment options are used for the honeypot. Protocol processes such as a TLS handshake or the time until the welcome message appears can be used to compare devices (Vetterl and Clayton, 2019). These attacks are not only dependent on the efficiency of the implementation. Device and memory virtualization can also contribute to timing discrepancies (Garfinkel et al., 2007). Although timing margins are getting smaller as hardware performance increases, a honeypot still depends on similar response times to the device it is representing. In cloud deployments, there are additional factors that can affect the timing of network packets, such as network nodes between the attacker and the target.

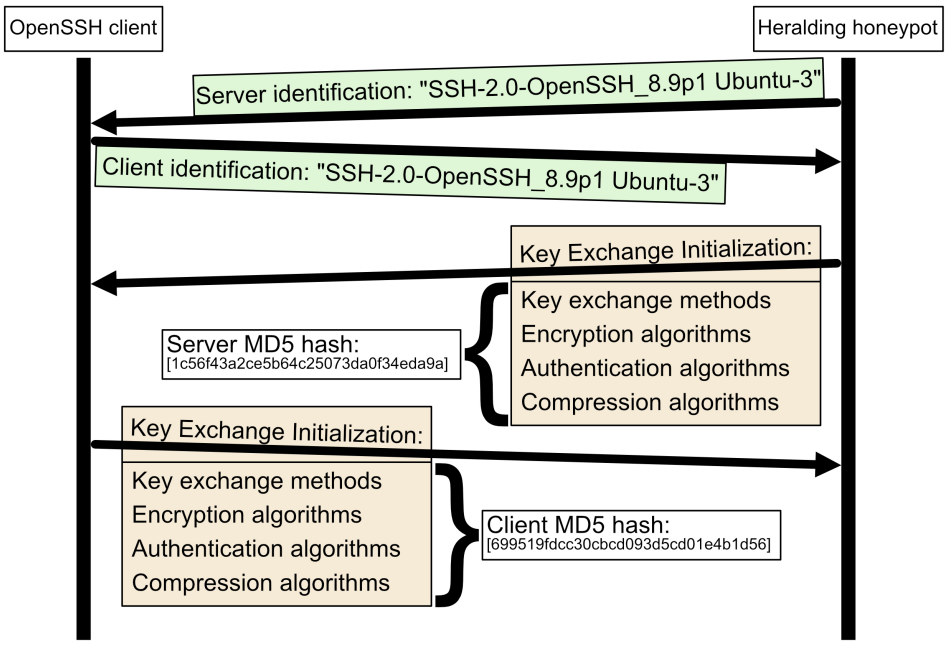


Figure 4: Comparison of an OpenSSH server fingerprint and a honeypot fingerprint. The MD5 hash is calculated from key exchange methods, encryption algorithms, authentication algorithms, and compression algorithms, which differ in the SSH implementations.

Application fingerprinting. Most current honeypots are not based on the same programming library as the protocol service they want to mimic. Thus, the developers need to adjust the honeypots' protocol responses manually to match the real implementation. *Fingerprinting* is done by searching for protocol strings that produce different response messages or response content depending on the protocol implementation in use. Differentiable error messages, padding, or used algorithms can reveal distinct protocol libraries. An example of successful fingerprinting can be seen in Figure 4. While both SSH servers (an OpenSSH server and a honeypot) return the same identification string during the handshake, they reveal differences in implementation through the key exchange methods, encryption algorithms, authentication algorithms, and compression algorithms which are exchanged at the key exchange initialization (SSH KEX_INIT) stage. Hashes differ even between different versions of the same implementation (e.g., OpenSSH 7.5 and OpenSSH 8.5) (Reardon et al., 2022). The tool used to create those hashes is HASSH (Reardon et al., 2022) and is described in Section 3.4.

In (Vetterl and Clayton, 2018) the authors present a breakthrough in fingerprinting, not patchable for the current generation of honeypots. They showcased a fingerprinting technique regarding state-of-the-art SSH, Telnet, and HTTP honeypots, identifying a large number of deployments, especially older versions. For example, they created 11,280,384 different protocol messages to observe the responses of various implementations of SSH. From these crafted messages, the authors searched for those that show the most divergence between implementations; these can later be used to test unknown clients.

Passive fingerprinting (observation of protocol header fields) was added by (Srinivasa et al., 2021a) to the probe-based approach of (Vetterl and Clayton, 2018) to further improve the findings; and we discuss additional academic publications on honeypot detection in Section 4.2. Not only academic work is focusing on fingerprinting: the popular scanner Shodan (Shodan, 2023d) offers a honeypot detection tool (Shodan, 2023a) that is believed to use a similar technique. Once exposed, fingerprinting methods can also be found as Metasploit (Rapid7, 2023) scripts for simple, automated testing of remote hosts. Shodan and Metasploit are also explained in more detail in Section 3.4. This shows that fingerprinting is not only a question of concept in the development phase but also requires constant updates to the system by developers and users.

3. Overview of Honeypots, Honeypot Frameworks, and Tools

In this section, we give an overview of open-source honeypots and honeypot frameworks that are actively maintained or have unique capabilities. The overview is clustered based on the interaction

level of the individual honeypots from low to medium to high. Moreover, we present a selection of tools that are helpful in testing honeypots.

3.1. Methodology

We have compiled and practically deployed the honeypots from mid-2022 to early 2023. To find honeypots, we searched for popular protocols (e.g., SSH, HTTP, and FTP) and completed them with honeypots that cover protocols not addressed by popular solutions. Another emphasis was on unique functionalities that are not offered by well-known honeypots. Valuable resources to start are lists like (Jose Nazario, 2022), GitHub repositories of honeypot foundations, and academic work investigating honeypots, extending them, or analyzing their data.

The insights into the system architecture stem from practical experience with the honeypots, as well as existing documentation. For practical experience, we have not only installed the honeypots but have tested their functionality as comprehensively as possible; we have investigated the system through network and vulnerability scans and investigated the resulting logging entries. For insights into the software architecture, we also examined the source code for dependencies and structure. The logging functionality and the deployment options were also drawn from practical experience and complemented by the honeypot's documentation. To draw conclusions about honeypot detectability, we looked at several factors: software libraries, configurability, network scans, and whether honeypots have already been fingerprinted in academic work.

Furthermore, the tools for honeypot testing were compiled in the same time frame. We included tools that were either used in academic work to analyze honeypots or that we found helpful for our experimental work with the honeypots. While libraries or platforms used to build the honeypot are included in the honeypot survey, we highlight tools for testing and comparing the honeypot's credibility.

3.2. Honeypots

To collect practical experience and to better understand the honeypots' features, we experimentally deployed all honeypots described in this section. Our focus was on the honeypots' deployment options, offered protocols, their unique deception capabilities, external libraries these honeypots depend on, and the programming language they are written in. In addition, we compare the honeypots' architectures. A summary of our findings is given in Table 2. This overview of honeypots is clustered based on the interaction level. Starting with LHs, the higher levels are briefly introduced at the first honeypot of the level.

3.2.1. Heralding

Heralding (Vestergaard, 2023) is a simple LH distributed under the `GNU General Public License v3.0`. Heralding is designed to mimic services typically found in enterprise IT systems, including SSH, Telnet, File Transfer Protocol (FTP), SMTP, VNC, PostgreSQL, SOCKS5 as well as HTTP, POP3, and IMAP as well as their secure variants HTTPS, POP3S, and IMAPS.

System architecture. Heralding is configured using a YAML file `heralding.yml`. It includes settings for individual services as well as generic settings such as the IP address to listen to and the logging configuration. Figure 5 shows Heralding's software architecture. Services mimicked by the honeypot only allow for log-in interaction and close the connection afterward. Listing 1 shows the result of an NMAP port scan of a deployed Heralding instance.

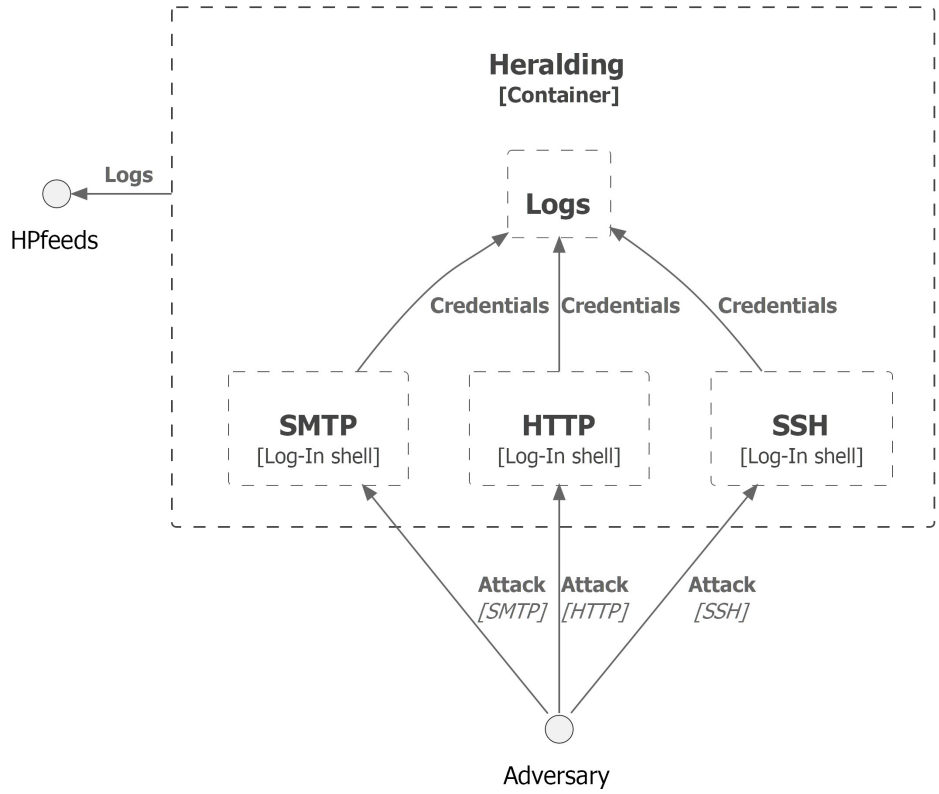


Figure 5: Architecture of a Heralding honeypot.

```
Nmap scan report for Herlading-IP
PORT      STATE  SERVICE
21/tcp    open   ftp
23/tcp    open   telnet
110/tcp   open   pop3
2222/tcp  open   ssh
8080/tcp  open   http
```

Listing 1: Port scan of a Heralding honeypot.

Software architecture. Heralding is written in Python. Running Heralding's latest release 1.0.7 requires Python version 3.7.0 or higher. Each service that the honeypot can mimic is implemented in its own Python module, and the modules can be activated in Heralding's configuration file. As a result, it is easy to extend Heralding with additional services or adjust the existing services if needed.

Heralding depends on Python libraries AsyncSSH, aiosmtpd, and pyOpenSSL, among others. To mimic HTTPS, Heralding generates digital certificates using pyOpenSSL, a Python wrapper for the popular cryptography library OpenSSL. Alternatively, the administrator can supply the certificates during the honeypot's deployment.

Logging. Heralding logs the activities performed by an attacker in a CSV or a JSON file. The data being logged includes the attacker's login credentials, the duration of the session, the protocol abused, and the remote IP and port. Alternatively, HPfeeds (Carr et al., 2021), a popular honeypot event forwarder, can be used to send the log data to a remote location. HPfeeds is a publish-subscribe protocol that has built-in TLS support. Heralding can be therefore configured to transmit its log data to a remote location over a secure channel.

Fingerprinting. Since Heralding mimics a rather large number of services, honeypot fingerprinting is a concern and can be difficult to address given current configuration options. As an example, while the SSH version that Heralding should mimic can be set in `heralding.yml`, the actual protocol responses

implemented in Heralding's SSH module are static. Consequently, an attacker can easily fingerprint Heralding's SSH service as shown in Figure 4.

Deployment options. Heralding can be easily deployed in a container-based environment since its GitHub repository (Vestergaard, 2023) includes a Dockerfile for creating the corresponding Docker image. It is based on the `python:3.7-slim-stretch` base image, with the resulting Docker image having merely 145 MB.

Popularity. With 344 stars and 77 Forks on GitHub, Heralding is one of the most popular LHs with nearly 1,400 commits from 19 contributors. However, the code frequency graph in Heralding's GitHub repository indicates that the majority of its functionality was implemented in the period 2013-2014, with smaller additions in 2019 and 2020.

3.2.2. Glutton

The authors of Glutton (MushMush Foundation, 2023) describe it as a generic LH. Glutton supports a wide range of protocols including Android Debug Bridge (ADB), FTP, HTTP, Extensible Messaging and Presence Protocol (XMPP/Jabber), SMTP, MQTT, SSH, Telnet, and base TCP. Glutton can even mimic BitTorrent and Ethereum Remote Procedure calls.

System architecture. The honeypot listens to ports defined in a rule set. The same rule set also includes mappings of incoming requests to specific connection handlers. Figure 6 shows an example of such mapping for incoming packets on ports 22, 80, and 1883.

In addition, Glutton can proxy the network traffic to another honeypot if a specific, situation-aware protocol response or an additional interaction with the attacker beyond credential catching is desired. The proxy feature is implemented for SSH, Telnet, and TCP protocols.

Software architecture. Glutton is written in the Go programming language, with each protocol handler implemented in a separate file. It is therefore easy to extend Glutton with further protocol handlers. The mapping between individual ports and protocol handlers is stored in the `rules.yml` configuration file and realized using `libnetfilter-queue` (netfilter.org, 2021b) and `iptables` (netfilter.org, 2021a).

Logging. If a log file path is provided, Glutton stores the log data locally. Alternatively, it can be configured to use `hpfeeds` remote logging protocol. The log data itself is in JSON format and contains information about the specific honeypot ID and the protocol handler as well as IP, port, and credentials of the attacker. This allows to maintain a central database where the information is collected from multiple Glutton instances.

One notable feature is Glutton's ability to highlight known scanner IPs like those of Censys (Censys, 2023) and ShadowServer (The Shadowserver Foundation, 2023) if they happen to appear in the log file (Veronica Valeros, 2022). It allows the honeypot operator to distinguish between scanner traffic and actual attempts to break into the systems mimicked by the honeypot.

Fingerprinting. Glutton provides no dedicated technical means to avoid fingerprinting. However, the proxy feature described above can be leveraged to implicitly mitigate the risk of fingerprinting by proxying the traffic to a more sophisticated honeypot that actually implements the relevant protocol. As an example, Glutton authors point to the Cowrie honeypot we discuss in 3.2.14 as a possible endpoint.

Deployment options. Glutton Git repository includes a Dockerfile based on a Go language Alpine Docker image. As a result, Glutton can be easily deployed in cloud environments. In addition, the honeypot's repository provides guidance for deploying Glutton to classical hosts and VM targets, including Debian- and Arch-based systems.

Popularity. Glutton is an actively maintained honeypot with nearly 200 stars and 54 forks on GitHub. The commit history shows that the main functionality is already implemented but minor changes and pull requests are still being incorporated.

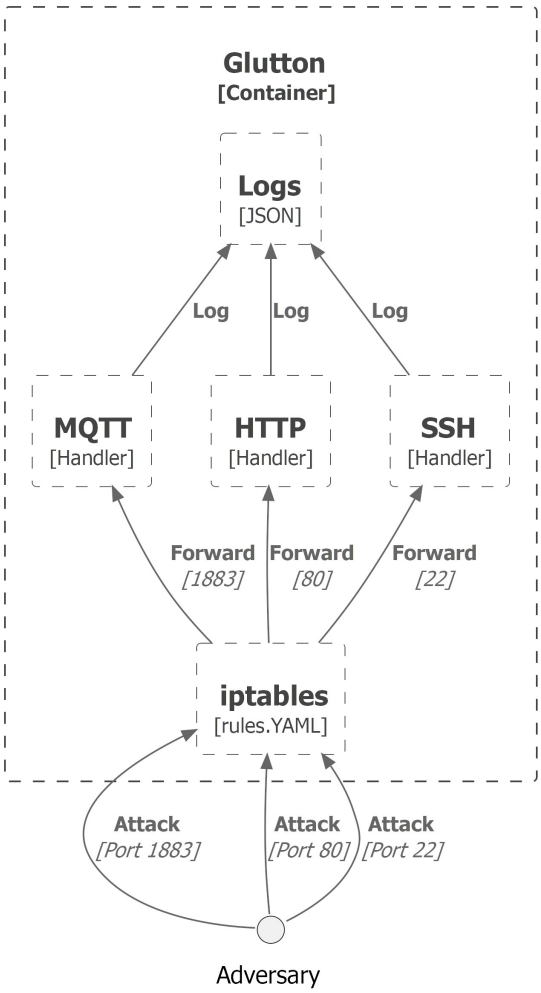


Figure 6: Glutton honeypot architecture.

Literature. The Glutton GitHub repository contains short examples of how to set up Glutton as an HTTP, SSH, TCP, UDP, and Telnet honeypot. Besides that, the documentation is very rudimentary. There is a compact deployment tutorial and initial results by the cybersecurity group of the University in Prague (Veronica Valeros, 2022).

3.2.3. Endlessh

Endlessh (Wellons, 2021) is an LH that acts as an SSH *tarpit*. An SSH tarpit traps the scanning client in an endless SSH handshake using the feature of SSH protocol that allows the SSH endpoint to send additional data, e.g., custom settings, before completing the SSH handshake. Based on this feature, Endlessh sends a random SSH banner to the scanning client and thereby detains automated attacks on SSH.

System and software architecture. Endlessh is a stand-alone, single-threaded C program. This translates into a simple architecture illustrated in Figure 7. Because the tarpit banner is sent previously to any cryptographic exchange, Endlessh does not depend on any cryptographic libraries. The configuration allows standalone IPv4 or IPv6 usage as well as options for banner length and supported number of clients.

Figure 7: Architecture of Endlessh.

Logging. If logging is activated, Endlessh log messages are written to the standard output. The option to write logs to the system log allows integration with any system without overhead.

Endlessh's configuration allows setting 3 different logging levels ranging from no logging (level 0) to comprehensive logging including debugging information (level 2).

Fingerprinting. Because Endlessh tarpits an attacker during the SSH handshake, fingerprinting the honeypot is not possible. The banner exchange is the first step of the SSH protocol initiation. As a result, Endlessh's working principle prevents automated fingerprinting attempts by design.

Deployment options. Endlessh GitHub repository provides a very small Docker image based on Alpine Linux. The Docker image has a size of merely 5.67 MB. The documentation also includes instructions for compiling Endlessh for Red Hat Enterprise Linux 6, Solaris, and OpenBSD operating systems as well as instructions for integration into Solaris Service Management Facility.

Popularity. On GitHub, Endlessh has over 5,800 stars and 242 forks. In total, the project has 14 contributors. Version 1.0 was released on April 29, 2019. To date, there were 100 commits made to the project. However, the last commit dates back to April 30, 2021.

Literature. DigitalOcean has published a detailed tutorial on an Endlessh cloud deployment (Tier, 2022). In the tutorial, the honeypot is deployed on a Ubuntu 22.04 server and also incorporates the necessary configuration for a firewall like UFW (Wallen, 2015).

3.2.4. Blacknet 2

Blacknet 2 (Mòrian, 2019) is an SSH honeypot system that allows the defenders to spawn an arbitrary number of SSH honeypots. These honeypots then transmit their log data to a central server. The individual SSH honeypots can be exposed to the Internet on different IPv4 addresses. Moreover, Blacknet 2 includes a web interface for live tracking of events—more precisely, SSH login attempts—reported by the honeypots. The interface can also be used to export basic statistics on honeypot events.

System architecture. Blacknet's system architecture is shown in Figure 8. Blacknet 2 is built using the server-client model where a central server, referred to as *master* server in Blacknet's documentation, receives data from the individual SSH honeypot sensors and persistently stores that data for subsequent inspection. The default storage option used by Blacknet 2 is a relational database like MySQL or MariaDB.

Blacknet 2 uses Transport Layer Security (TLS) to enable secure communication between the SSH honeypot sensors and the master server. Blacknet's authors recommend using EasyRSA, a command line tool for creating Certification Authorities (OpenVPN Inc, 2023), to generate digital certificates to be deployed on the honeypot sensors and the master server.

Software architecture. Blacknet's honeypot sensors are low-interaction SSH honeypots. The mocked SSH service is implemented in Python, the Python versions supported by Blacknet 2 are 2.7 and 3.4. The SSH honeypot sensors use the `Paramiko` (Forcier and Gaynor, 2022) SSH library to establish the initial SSH connection and to present the password prompt to the attacker.

The SSH logs are transmitted using the MessagePack binary serialization format (Furuhashi, 2021). MessagePack allows a very compact representation of the log data so that Blacknet 2 can be deployed in scenarios with limited network bandwidth. The persistent logging functionality utilizes the PyMySQL (Matsubara, 2023) library.

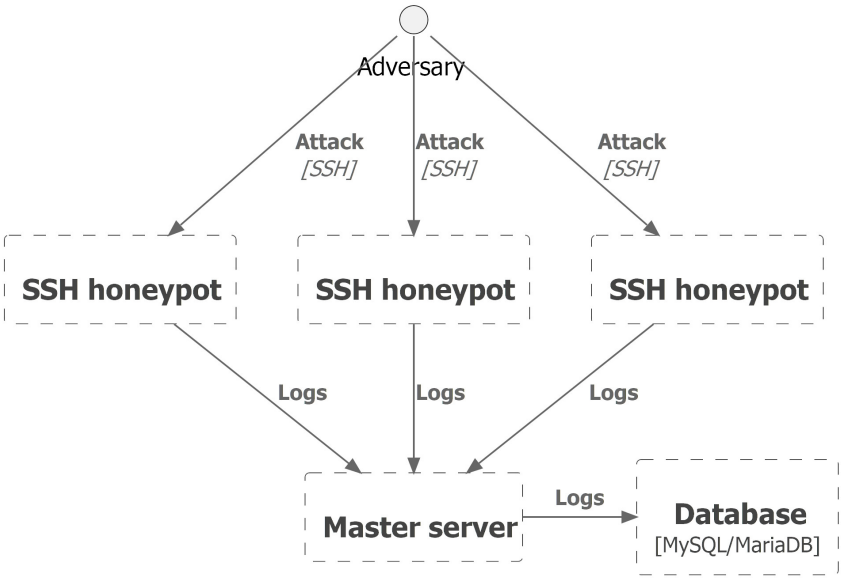


Figure 8: Architecture of Blacknet 2.

Logging. Logs can be saved locally or on a dedicated network-connected node. In the latter case, the master server establishes a connection to a MySQL or MariaDB database through TCP/IP or a Unix socket.

Fingerprinting. Fingerprinting of the `Paramiko` library is a concern, but an integration of other honeypots as sensors should be possible with moderate effort. The use of different honeypots as sensors is another interesting alternative.

Deployment options. Blacknet's GitHub repository provides no Docker images or instructions on how to build one. However, since the SSH honeypot sensors and the master server only depend on a handful of Python libraries, Blacknet can be easily dockerized. Moreover, packaging the honeypot sensors in a Docker container would further simplify the integration of other honeypots as sensors.

Popularity. Blacknet 2 was initially written in 2010 and rewritten in 2017 based on an LH design. As of the time of this writing, Blacknet's GitHub repository has only 10 stars, but the client-server architecture and the resulting capability of distributed honeypots are noteworthy features.

3.2.5. Masscanned

Masscanned (IVRE, 2023b) is an LH specialized in capturing scanning attempts and bots. It is published under the `GPL-3.0 open source license` and implements protocols on different Open Systems Interconnection (OSI) model layers including HTTP, SSH, SMB, DNS, ARP, ICMP version 6, TCP, and UDP. This, in turn, allows using Masscanned to implement a wide variety of honeypots.

System architecture. Masscanned was originally written for integration into Instrument de Veille sur les Réseaux Extérieurs (IVRE), an open-source reconnaissance framework composed of tools for passive and active reconnaissance (IVRE, 2023a). However, Masscanned can also be used independently from IVRE. The architecture of the honeypot is shown in Figure 9.

Masscanned implements the network stack in the user space. As a result, responses to the incoming network packets require no access to the operating system's kernel. A major advantage of this architecture is Masscanned's ability to customize IP and MAC addresses and generate responses on the different network layers.

For a stand-alone deployment, a virtual network interface is created, and the honeypot listens on all ports. Deployment and configuration can be done via the command line.

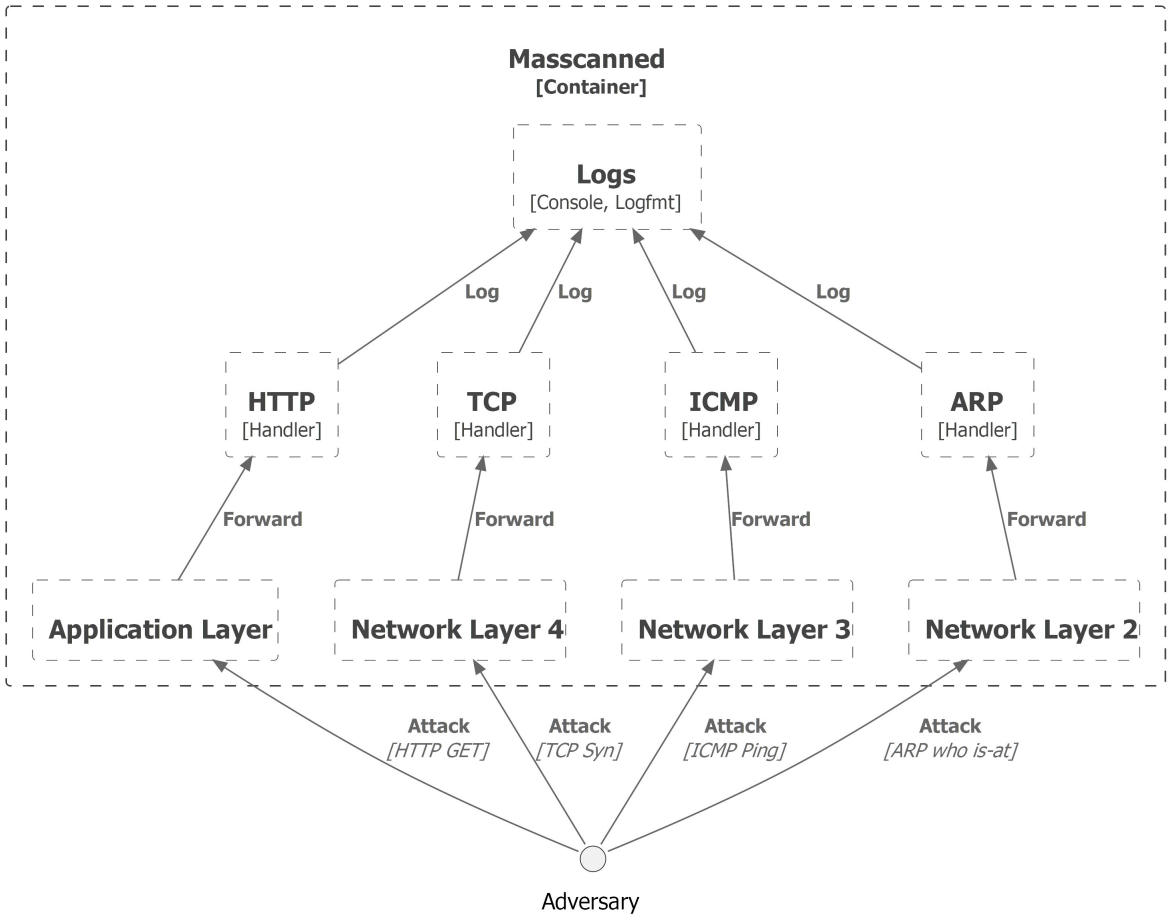


Figure 9: System architecture of masscanned. The honeypot does implement protocols on different network layers.

Software architecture. Masscanned is written in Rust programming language and packaged for Cargo, Rust's default package manager. Standard Rust (network) libraries are used to implement the honeypot's network protocols. This is possible since their implementation contains only basic functionality. As an example, the SSH implementation contains no cryptographic key exchange. Individual modules are sorted by layer—layers 2,3,4, and the application layer of the OSI model—and all modules are activated by default.

Logging. Logs are by default displayed on the console, but `logfmt` format is also supported. All incoming network packets are saved in Packet Capture (PCAP) files which can be used for subsequent analysis. In addition, the data from the network packets can be evaluated by integrating masscanned with IVRE.

Fingerprinting. Because masscanned is tailored to detect simple scanning attempts rather than meticulously disguise itself as a real IT system, the honeypot has no dedicated measures to prevent its detection. However, since the honeypot implements a custom *userland* network stack, masscanned operators can implement their own countermeasures against fingerprinting.

Deployment options. Masscanned authors suggest a deployment on a virtual private server and provide a build manual for a stand-alone deployment in the project's README on GitHub. While no Dockerfile is provided, packaging a Rust application in a Docker container is simple. Since masscanned listens on all network ports, a Docker container needs to use the host networking and have the capability to forge network packets.

Popularity. With 61 stars and 11 forks on GitHub, masscanned appears to be less popular in its standalone version. However, it is commonly used as part of the IVRE framework which, having over

2,800 stars and over 600 forks on GitHub, receives significant attention in the developer community. Moreover, masscanned's commit history shows that it continues to receive regular updates to its code base.

3.2.6. Glastopf

Glastopf (MushMush Foundation, 2021a) is a low-interaction web application honeypot. By mimicking a wide range of web application, database, and cross-site scripting vulnerabilities, Glastopf mimics a vulnerable web server with a large attack surface.

Glastopf is designed to be indexed by search engines such that, in combination with its large attack surface, the number of attack attempts that hit the honeypot is maximized.

System architecture. Glastopf is a pure web application honeypot designed to emulate a wide range of web application, database, and cross-site scripting vulnerabilities. The honeypot's system architecture is shown in Figure 10.

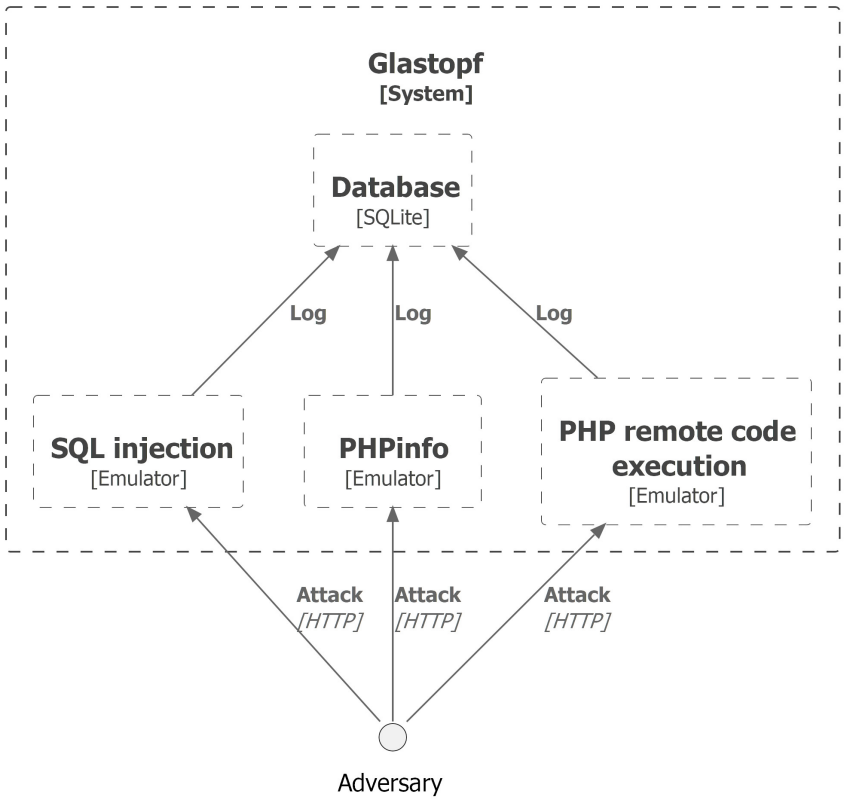


Figure 10: Glastopf system architecture.

Technically, Glastopf emulates vulnerability *types* like Remote File Inclusion (RFI) or HTML injection rather than actual, specific vulnerabilities. To achieve this, the honeypot first determines the HTTP request type (GET, POST, or HEAD). In the second step, the honeypot tries to determine the type of the attempted attack by searching for specific, pre-defined patterns in the HTTP request. As an example, using the pattern `=http://`, Glastopf would detect an RFI attack in the HTTP request:

```
GET http://example.com/vulnerable.php?color=http://evil.com/shell.php
```

Finally, once the attack type is identified, Glastopf calls an attack-specific handler which, in turn, generates the response simulating a successful attack.

In addition, Glastopf implements a simple parser for injected PHP files. The parser takes the injected PHP file, extracts statements that generate output, and generates a valid response corresponding to these statements. As an example, if the injected PHP file contains:

```
$un = @php_uname(); echo "uname -a: $un<br>"
```

Glastopf replaces `@php_uname()` with a valid (but fake) value and generates a response like:

```
uname -a: GNU/Linux","Linux my.leetserver.com 2.6.18-6-k7<br>
```

In addition, emulators for a login interface to catch brute-force attack attempts or attempted SQL injections can be complemented by staged `phpinfo` or `phpmyadmin` pages. Those contain hints of deficiencies such as a vulnerable PHP version. This can range up to the emulation of specific exploits like PHP remote code execution (National Vulnerability Database, 2018).

Software architecture. Glastopf is implemented in Python. The honeypot's code base consists of individual Python modules implementing HTTP functionality, event classification, vulnerability emulators, and data processing and logging.

The vulnerability type emulators can access pre-defined HTML pages to present a web page to the attacker. In addition, a PHP sandbox is implemented that executes a set of whitelisted PHP functions. Both the emulators and the sandbox can be easily extended with custom functionality if needed.

Glastopf mostly uses modules from the Python standard library such as `urllib` and `base HTTP server`. Moreover, the well-known `gevent` (Bilenko, 2019) is used as a networking library.

Logging. Glastopf logs contain the source IP of the attacker as well as the request URL and the attack pattern, e.g., the string of an SQL injection. The logs can either be written into an SQLite database or sent to a remote log collector using the HPfeeds protocol.

Fingerprinting. The emulation of different vulnerability types allows for a realistic picture of a vulnerable web server. While the honeypot can be fingerprinted in principle, this would require an attacker to craft dedicated HTTP requests to which the honeypot responds differently than an actual web server.

Deployment options. Since Glastopf is not actively maintained, a host deployment is not recommended. The honeypot's GitHub repository provides a Dockerfile based on Ubuntu 14.04 and Python 2.7.

Popularity. Glastopf received its last commit in October 2021. Since the last major changes to the code base happened in 2014, the authors recommend the use of the successor SNARE/TANNER. Nearly 500 stars and 174 forks on GitHub highlight how Glastopf brought novel approaches to the realm of web application honeypots.

Table 2: Overview of open-source honeypots including notable features. A * marks the successor in the following row.

Honeypot	Last Maintained	Protocols	Type	Language	Interaction Level	GitHub Stars
Heralding (Vestergaard, 2023)	2023	HTTP(s), SSH, FTP, POP3, Telnet, IMAP, SQL, PostgreSQL, SMTP, SOCKS5	Credentials	Python	Low	344
Glutton (MushMush Foundation, 2023)	2023	SSH, Telnet, ADB, FTP, HTTP, XMPP, SMTP, MQTT, TCP	Credentials/ Proxy	GO	Low	196
Endlessh (Wellons, 2021)	2021	SSH	Bot-trapping	C	Low	5,811
Blacknet 2 (Mòrian, 2019)	2019	SSH	Client-server	Python	Low	10
Masscanned (IVRE, 2023b)	2023	HTTP, SSH, STUN, SMB, DNS, ARP, ICMP(v6), TCP, UDP	Scans/Bots	RUST	Low	61
Glastopf* (MushMush Foundation, 2021a)	2021	HTTP(s)	Vulnerability type emulation	Python	Low	485
SNARE(MushMush Foundation, 2021b)/ TANNER(MushMush Foundation, 2022b)	2021/ 2022	HTTP(s)	Web application cloning	Python	Low	387/181
Dionaea (DinoTools, 2021)	2021	HTTP, FTP, MQTT, MySQL, UPnP	Malware extraction	C, Python	Low	628
OpenCanary (Thinkst Applied Research, 2023)	2023	HTTP, FTP, MSSQL, MySQL, NTP, RDP, SSH, SNMP, Telnet, VNC, TFTP, SAMBA, Git, TCP	Server imitation	Python	Low	1,684
Conpot (MushMush Foundation, 2022a)	2022	HTTP, S7comm, FTP, Modbus, EtherNet/IP	ICS imitation	Python	Low	1,076
Honeyntp (Fyodor, 2014)	2014	NTP	Connection logging	Python	Low	50
Log4Pot (Patzke, 2023)	2022	Log4Shell	Vulnerability pitfall	Python	Low	84
Kippo* (Upi Tamminen, 2016)	2016	SSH	Shell interaction	Python	Medium	1,457
Cowrie (Oosterhof, 2023)	2023	SSH, Telnet	Shell interaction	Python	Medium	4,184
Sshd-honeypot (amv42, 2018)	2018	SSH, Telnet	Cowrie proxy	C	Medium	22
Sshesame (Jakab, 2023)	2023	SSH	Docker environment	GO	Medium	1,259
Wetland (ohmyadd, 2018)	2018	SSH, SFTP, TCP	Proxy for a Docker environment	Python	High	120

3.2.7. SNARE/TANNER

The Super Next generation Advanced Reactive honEypot (SNARE) (MushMush Foundation, 2021b) and TANNER (MushMush Foundation, 2022b) are the successors of Glastopf. Thus, they build another web application honeypot system but the separate structure provides better camouflage and performance.

System architecture. The honeypot is divided into SNARE and TANNER. SNARE clones existing web pages and converts them into attack surfaces. The honeypot then looks like an NGINX web server presenting the cloned contents. As shown in Figure 11, incoming HTTP requests are forwarded to TANNER. TANNER analyzes and classifies the request before crafting a response. TANNER also contains the vulnerability emulators known from Glastopf.

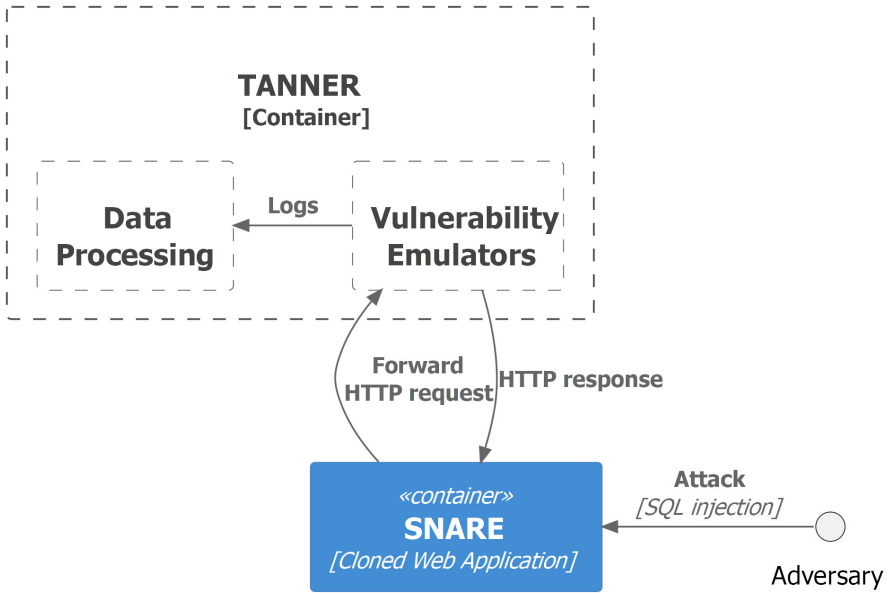


Figure 11: Architecture of a SNARE/TANNER setup.

Software architecture. SNARE and TANNER are developed in Python and mostly tested for Python 3.4+. SNARE uses `Beautiful Soup` (Richardson, 2020) to pull data from HTML files. For web server capabilities, `asyncio` (Python Standard Library, 2023) and `AIOHTTP` (aiohttp contributors, 2023) (both HTTP libraries) are utilized.

Tanner provides different vulnerability emulators, e.g., for Cross-Site Scripting (XSS). Patterns for XSS or SQL injections can be expanded and additional emulators can be implemented. Incoming SNARE sessions are analyzed similarly to Glastopf; the `geoip2` module (MaxMind, 2023) provides additional information about the attacker's IP.

Logging. SNARE forwards requests to TANNER where information about an attack is logged locally in a JSON file. The honeypot, further, implements HPfeeds and MongoDB clients via Python.

Fingerprinting. The web application cloning and the representation of an NGINX web server improve the cover of the honeypot compared to Glastopf. Fingerprinting the Python HTTP libraries is a concern, but requires actively probing the honeypot—which means additional overhead for an attacker. By passively watching the honeypot's HTTP traffic it looks like an NGINX web server.

Deployment options. The documentation (MushMush Foundation, 2016, 2018) contains instructions for local deployment. Further, both components have Dockerfiles based on Alpine images. The Docker images for SNARE (279 MB) and TANNER (330 MB) allow a compact Dockerization of the architecture considering the extensive capabilities.

Popularity. SNARE (387 stars and 122 forks) and TANNER (181 stars and 87 forks) are developed under `GPL-3.0 license`. Although the popularity of the Honeypot is inferior to its predecessor, it brings expanded capabilities that allow for even more specialized use.

3.2.8. Dionaea

Dionaea, the successor to Nepenthes (Baecher et al., 2006), emulates exploitable vulnerabilities that are exposed by services connected to the network. The honeypot's goal is to receive a malware copy, and it offers a wide range of protocols to get there: HTTP, FTP, TFTP, MySQL, MSSQL, UPnP, SMB, SIP, PPTP, MQTT, Mirror, Memcache, Blackhole, and epmap build Dionaea's protocol suite.

System architecture. Dionaea offers a wide range of features that make the system architecture quite complex. The honeypot consists of different modules that can be activated at wish. Figure 12 shows a sample architecture with the Python and the Emulation (EMU) module running. The Python module deploys the services offered by the honeypot. In addition, the EMU module receives all in and outgoing communication of deployed services to detect and emulate shell code from payloads. This is done to gain malware copies.

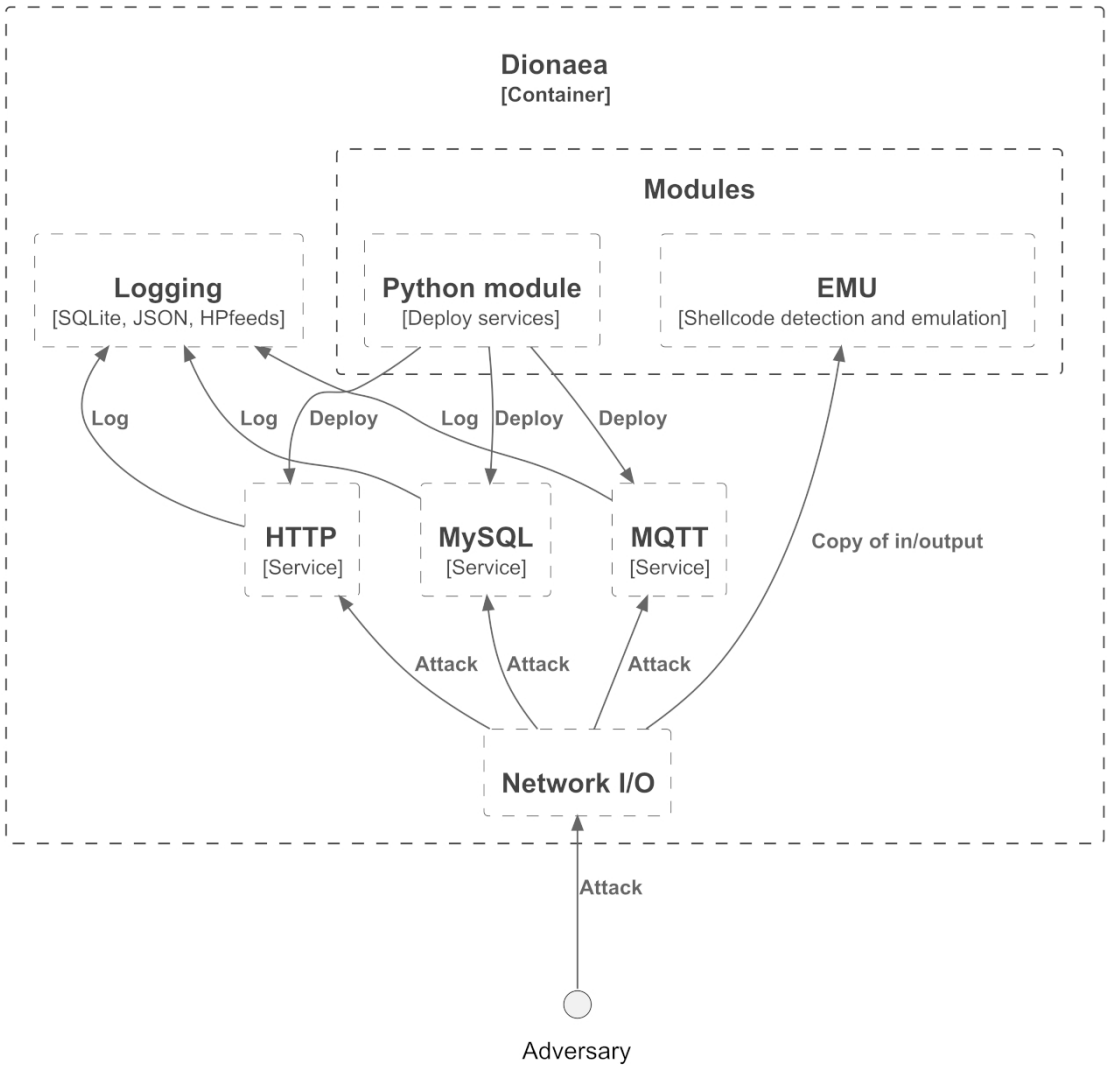


Figure 12: Illustration of the Dionaea architecture.

Software architecture. The core of Dionaea is implemented in C but it embeds Python via Python/C API (Python Software Foundation, 2023). The possibility to include services via Python is a unique feature of the honeypot. The connection handling below the application layer, as well as the modules

are written in C. OpenSSL libraries provide SSL/TLS support, and `libemu` (Angelo Dell'Aera, 2022) is used for the shell code detection and emulation.

The Python module allows for the deployment of services that are implemented in Python. Thus, this module contains all of the honeypot's Python scripts. Recommended Python versions are 3.8 and 3.9. In this part of the honeypot implementation, regular Python libraries like `sqlite3` and `Jinja2` are in use.

Logging. Dionaea supports logging as JSON files, in a database, or via HPfeeds. The honeypot has a filter module to set rules for log filtering and allows the activation of fail2ban via the configuration file. Fail2ban is an intrusion detection and prevention module, that scans log files for IPs causing malicious activity and sets firewall rules to prevent these from entering again.

Fingerprinting. Dionaea was successfully fingerprinted in (Srinivasa et al., 2021a) due to two reasons: static values in the generated SSL/TLS certificates and default configurations of services. Default configurations like protocol banner strings must be avoided upon deployment and can be changed in the configuration of each service. The X509 certificate for SSL/TLS is generated at startup and is self-signed. But parameters can be changed in the source code.

Deployment options. For host deployments, the authors recommend Ubuntu 18.04 or Debian 10. Ubuntu 20.04 and Debian 11 are not supported since those dropped libemu from the package repository (DinoTools, 2015).

For Docker deployment, an official Docker image (DinoTools, 2023) is provided, but same can be built from the GitHub repository. The image is also based on Ubuntu 18.04. With a size of 182 MB, the image is comparatively small. After the honeypot is built, build packages and source code are removed from the container which contributes to the small size.

Popularity. While a lot of the code base was written from 2010 to 2012, the honeypot was still maintained and expanded between 2016 and 2021. With 628 stars and 169 forks, Dionaea has received sizable attention in the honeypot community. The honeypot is released under `GPL-2.0 license`.

3.2.9. OpenCanary

OpenCanary (Thinkst Applied Research, 2023) is an LH that is maintained by Thinkst Canary (Thinkst Canary, 2023b) and published under the `BSD-3-Clause license`. The honeypot can mimic typical enterprise IT servers, e.g., a Linux Web server, a Windows server, or a MySQL server. Offered services are FTP, HTTP, MSSQL, MySQL, NTP, RDP, Redis, SNMP, SSH, Telnet, VNC, TFTP, Samba, GIT, and TCP. Further, the honeypot has a port scan detection module.

System architecture. OpenCanary uses the configuration file to assemble server configurations from offered services. The official documentation (Thinkst Canary, 2023a) provides sample configurations for Linux Web server, Windows server, MySQL server, and MSSQL server. The Linux Web server configuration, for example, deploys FTP, HTTP, and SSH services looking like an Apache Ubuntu server running OpenSSH. The architecture of this sample configuration is illustrated in Figure 13.

The Samba module requires a dedicated Samba installation. Then events are triggered as soon as somebody accesses the file share.

Software architecture. The honeypot is written in Python and supports versions 2.7 or 3.7+. The structure of the honeypot is modular and all services can be enabled through the configuration file. OpenCanary mostly uses `Twisted` (Twisted Matrix Labs, 2023) as a network library. The SNMP module additionally requires `Scapy`. A description of Scapy is provided in Section 3.4.

Logging. OpenCanary per default logs events in JSON format but can also write events into syslog. Further, the honeypot implements HPfeeds to send logs to a remote host.

Fingerprinting. Since there are known fingerprinting probes for Twisted (Vetterl and Clayton, 2018) active probing is a concern. But fingerprinting methods reported on the GitHub page were already fixed in the past. Further, a lot of passive fingerprinting depends on realistic system imitations. Those can be changed via the configuration file.

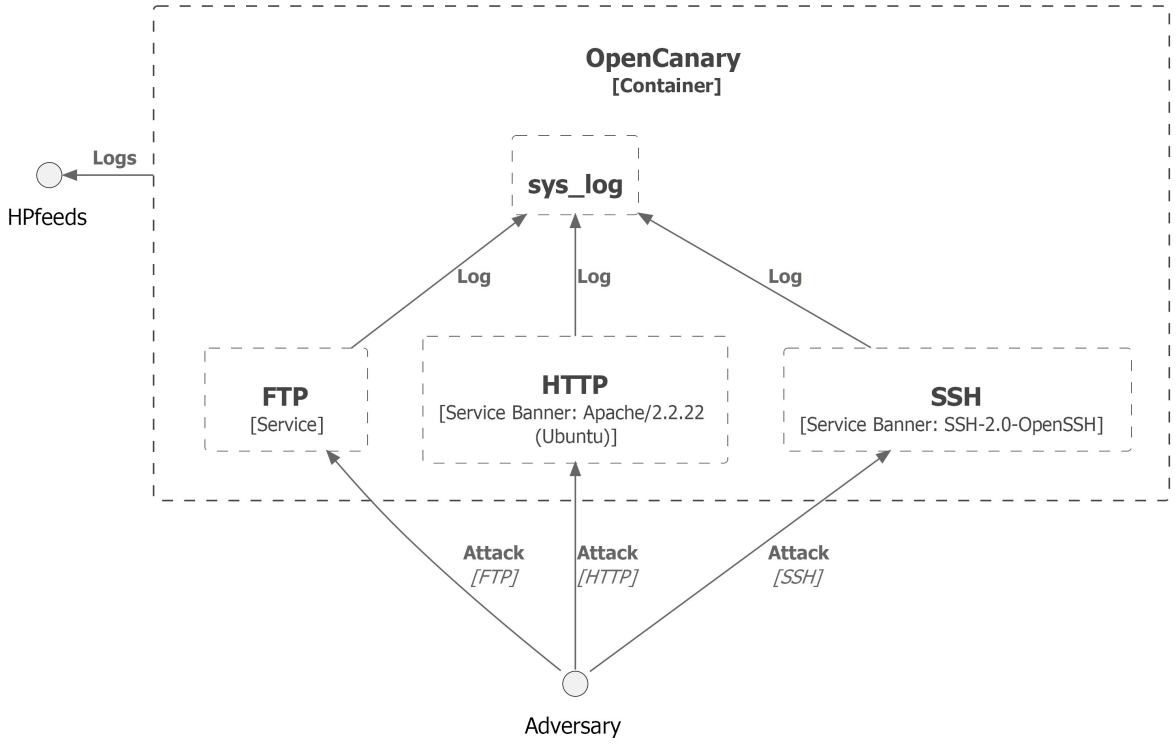


Figure 13: OpenCanary honeypot system architecture.

Deployment options. OpenCanary was initially supposed to run as a daemon on a host system. Therefore, it provides instructions for Ubuntu and OS X deployments. But the developers also made a Dockerfile available which is based on `python:3.7-buster`. The resulting Docker image is 964MB in size. This is quite sizable compared to other honeypots. By using a slimmer Python base image, for example, a lighter deployment is possible.

Popularity. OpenCanary is a maintained honeypot with over 1.6 thousand stars and over 300 forks on GitHub. The code history shows constant updates to the code base in the past three years. The development with the open-source community further allows for comprehensive testing; for example, a fingerprinting method for the SQL Server honeypot was reported on the GitHub page and subsequently fixed.

3.2.10. Conpot

Conpot (MushMush Foundation, 2022a) is an LH for Industrial Control Systems (ICS) that emulates multiple industrial control protocols and tasks, including HTTP and Supervisory Control and Data Acquisition (SCADA). The objective is a honeypot system that looks like an industrial network (The Honeynet Project, 2018). Supported protocols are HTTP, S7comm, FTP, Modbus, and EtherNet/IP. Further, there are templates for different industrial devices and interfaces that include IEC104, Kamstrup 382, Guardian AST, IPMI, and SNMP.

Although Conpot primarily aims to mimic ICS systems, it has a unique feature in inter-honeypot communication that can improve a honeypot's cover. This approach can also be useful in enterprise IT systems.

System architecture. Conpot wants to allow the user to freely design an industrial complex. Therefore, the honeypot includes different protocols and provides templates to mimic the industrial devices of various manufacturers. Those templates are XML files that contain properties of the represented device or Human-Machine Interface (HMI). As shown in 14, configured HMIs can connect to real clients to display realistic information to an adversary. Further, pre-configured traffic between devices and artificial delay to the service response times (The Honeynet Project, 2018) reinforces the illusion of a real industrial complex.

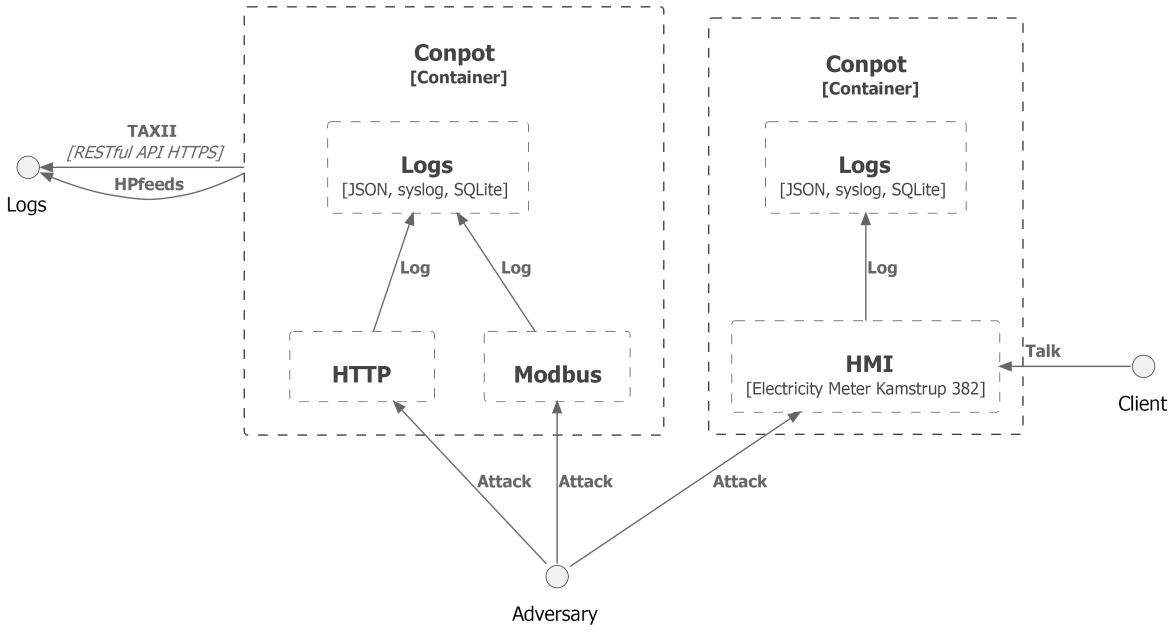


Figure 14: Conpot architecture. In addition to services, there is also the possibility of setting up a human-machine interface connected to a real client.

Software architecture. Conpot was written in Python and has a modular structure. The core contains basic honeypot functionality like logging, session handling, and protocol handlers. This is complemented by protocol and template modules. This structure allows an extension to specific parts of the honeypot.

The honeypot supports Python 3.6 or higher. For protocol implementations, the Python modules `socketserver` (Python, 2022) and `gevent` are frequently used. Gevent is a networking library.

Logging. Conpot has extensive logging capabilities. While the base output is JSON, the honeypot also provides support for syslog, HPfeeds, SQLite, as well as `STIX` (OASIS Open, 2022a), and `TAXII` (OASIS Open, 2022b). STIX and TAXII are open-source formats to serialize and exchange cyber threat intelligence. The exchanges happen via a RESTful API using HTTPS. Those protocols are published by OASIS Open (OASIS Open, 2022c).

Fingerprinting. Conpot is a very specialized environment with atypical protocols. While fingerprinting of HTTP and FTP is possible with the method described in (Vetterl and Clayton, 2018), other protocols need expert knowledge of industrial environments and protocols.

Deployment options. The latest release (0.6.0) provides a Docker image based on Python 3.8 as well as instructions for a host deployment of Conpot. The pre-built image comes with a size of 260 MB. Container deployments enable more possibilities for honeypots like Conpot. Especially, for large deployments representing an industrial network. Containerization saves hardware resources and deployment time compared with a server or VM installation.

Popularity. Conpot is published under the `GPL-2.0 license`. The honeypot provides a unique set of protocols that has broad appeal. Nearly 1.1 thousand stars and 400 forks on GitHub show a great interest in the honeypot. The core of the code base was developed between 2013 and 2016. But there were major updates to the code between 2018 and 2022, thus even after the latest official release (0.6.0) in 2018.

Literature. Conpot was extensively analyzed in (Gokhale et al., 2020). The authors have focused on finding interactions that end up in a dead end or in an infinite loop. The authors of (Dowling et al., 2019) and (Siniosoglou et al., 2020) used machine learning approaches to improve the honeypot's concealment.

3.2.11. Honeyntp

Honeyntp (Fyodor, 2014) is a Network Time Protocol (NTP) honeypot that is based on an open-source Python NTP server (limifly, 2015). NTP is a protocol to ensure time synchronization between IT systems. Although it does not provide access to a server, NTP and comparable protocols can still be targets of exploits that cause a denial of service or worse.

System and Software architecture. As illustrated in Figure 15 Honeyntp is an NTP server listening to incoming NTP packets. The honeypot collects received information in a Redis database.

The Python-implemented honeypot is a one-script application. The Redis library for Python is used for database communication and a configuration file provides the database's location.

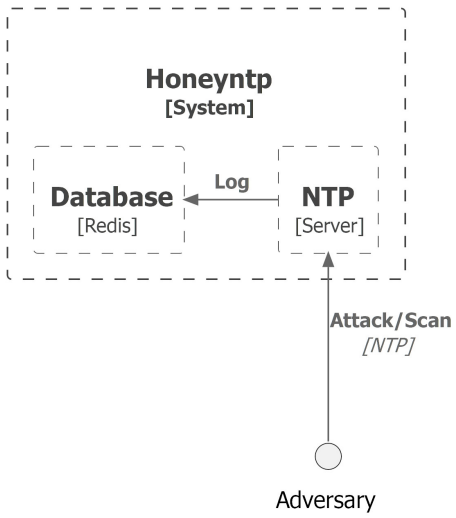


Figure 15: System architecture of Honeyntp. The implementation is based on a Python NTP server.

Logging. As mentioned earlier, Honeyntp logs into a redis database. But there are no packets logged, rather first-seen/last-seen information is stored per IP/port pair.

Fingerprinting. Since the underlying NTP server was last updated in 2015 there is a high probability, that the implementation is identifiable as a Python NTP server. This is a concern since there are approaches to OS fingerprinting utilizing the running NTP server implementation as a hint to the OS.

Deployment options. The base server to Honeyntp was only tested on Linux and Windows 7. There is no Dockerfile available in the GitHub repository. However, Honeyntp is a small Python application without large dependencies and, thus, can be integrated into a Docker environment.

Popularity. Honeyntp is a comparatively small project with 50 stars and 11 forks on GitHub. Despite that, time synchronization protocols are widespread and there is only a very limited number of honeypots implementing one.

3.2.12. Log4Pot

As soon as a new protocol vulnerability becomes public, many automated scanners start looking for potential targets. Log4Pot (Patzke, 2023) is a honeypot developed solely to catch attackers trying to exploit the Log4Shell vulnerability.

System and software architecture. The Log4Pot honeypot listens on different ports to receive payloads that try to exploit the Log4Shell vulnerability. As illustrated in Figure 16, incoming payloads are analyzed to detect malicious content.

Log4Pot was written in Python 3 and Poetry is used for dependency management. The server is implemented with HTTP and SSL modules from the Python standard library. In case traffic must be redirected to the honeypot, e.g., from port 80 and 433, `iptables` is needed separately. HTTP server

responses and HTML content can be individualized, as there is a custom directory to store those in JSON and HTML format.

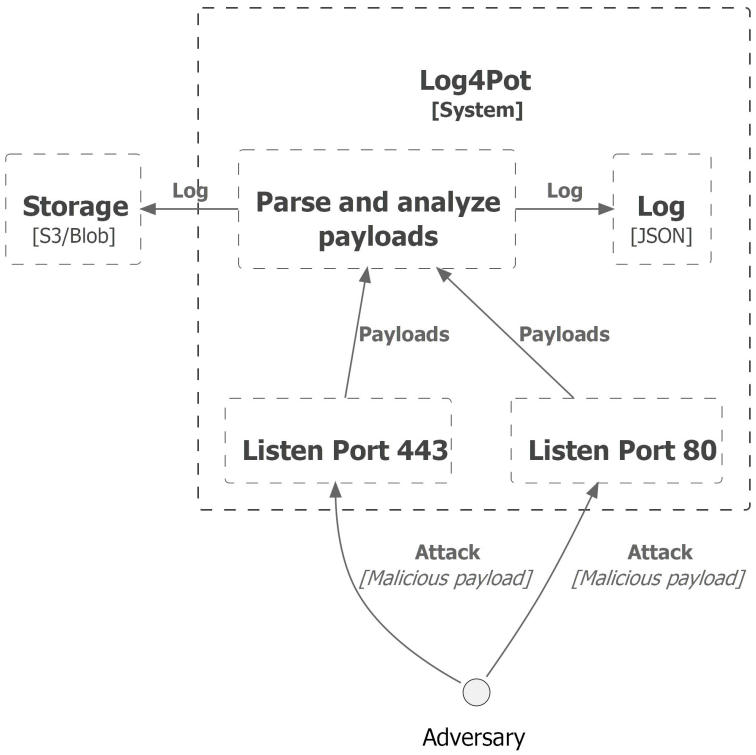


Figure 16: Architecture of Log4Pot. Logs can be stored in AWS S3 Buckets or Azure Blob.

Logging. Log4Pot's default logging option is JSON. The honeypot further implements functionality to support S3 buckets or Azure blob storage. This allows for seamless integration into a cloud environment.

Fingerprinting. The honeypot specializes in a vulnerability that is scanned for by a lot of automated scanners. Fingerprinting the HTTP implementation is not worth the effort, given that these scanners target a very large number of devices.

Deployment options. Log4Pot's GitHub repository currently does not contain a Dockerfile. However, considering that Log4Pot is a rather small Python project, containerization is possible. There are instructions for a host/VM deployment.

Popularity. Log4Pot was developed under `GPL-3.0 license`. With 84 stars and 21 forks, it is a rather small honeypot. But the monitoring of individual vulnerabilities is a concept that is often used, especially for known CVEs.

3.2.13. Kippo

MHs combine a low-risk environment of LHs with the greater knowledge gains of HHs. While those honeypots may provide a reasonable return for the moderate risk, the amount of functionality makes them difficult to develop. SSH is once again the most prominent service offered by the honeypots.

Kippo (Upi Tamminen, 2016) is an SSH honeypot with the purpose of logging brute force attacks and shell interactions. The honeypot was inspired by Kojoney but is no longer maintained. It is recommended to use the fork Cowrie.

System architecture. Kippo allows the attacker to gain access to the system via SSH login. After successful login, the attacker is presented with a shell and access to a file system. The general architecture is very similar to the successor Cowrie and comparable with the illustration in Figure 17. The file system resembles a Debian 5.0 installation and allows the defender to place custom files. The attacker can execute commands in the shell to modify the file system or to download files with `wget`.

Software architecture. The code base consists of the file system contents, the staged output of shell commands like `ifconig`, and the Python modules. Kippo requires Python 2.5+; Python modules are divided into core functionality, command implementations, and log functionality and are modular in structure. This allows the user to add new command functionality to the shell interaction.

Kippo depends on the `Twisted` library for networking, especially Twisted.Conch for SSH and Telnet capability. Further, `PyCrypto` is used as a cryptography toolkit and `MySQLdb` for database access.

Logging. Kippo has different logging modes to choose from: text logs and database logging in form of MySQL or XMPP. Session logs contain every command that is executed by the attacker and all downloaded files are stored for subsequent review.

Fingerprinting. Fingerprinting will be discussed for Cowrie since it is the most current version.

Deployment options. Kippo was tested on Debian, CentOS, FreeBSD, and Windows 7. Although there is no Dockerfile in Kippo's repository, Cowrie provides official Docker images.

Popularity. Although Kippo's last update ranges back to 2016, the honeypot was quite popular. Over 1.4 thousand stars on GitHub, as well as 274 forks, illustrate the popularity of Kippo's approach.

3.2.14. Cowrie

The Kippo successor Cowrie (Oosterhof, 2023) is a medium to high interaction honeypot expanding the capabilities of its predecessor. In addition to Telnet support, the developers added `curl`, `SFTP`, and `SCP` functionality to allow the attacker to perform file download and upload operations.

System architecture. The general architecture (see Figure 17) remained similar to Kippo. The attacker can connect to the system by SSH or Telnet and is presented with an emulated shell and file system structure. The attacker can download files with `wget` or `curl`, as well as upload files with `SCP` or `SFTP`.

In addition to Kippo features, Cowrie supports a high interaction proxy mode. The honeypot can function as a proxy to a honeypot system to monitor activity. Further, Cowrie can use QEMU-emulated servers as honeypot systems. In both modes, Cowrie can be configured to forward SMTP connections to an SMTP honeypot of choice.

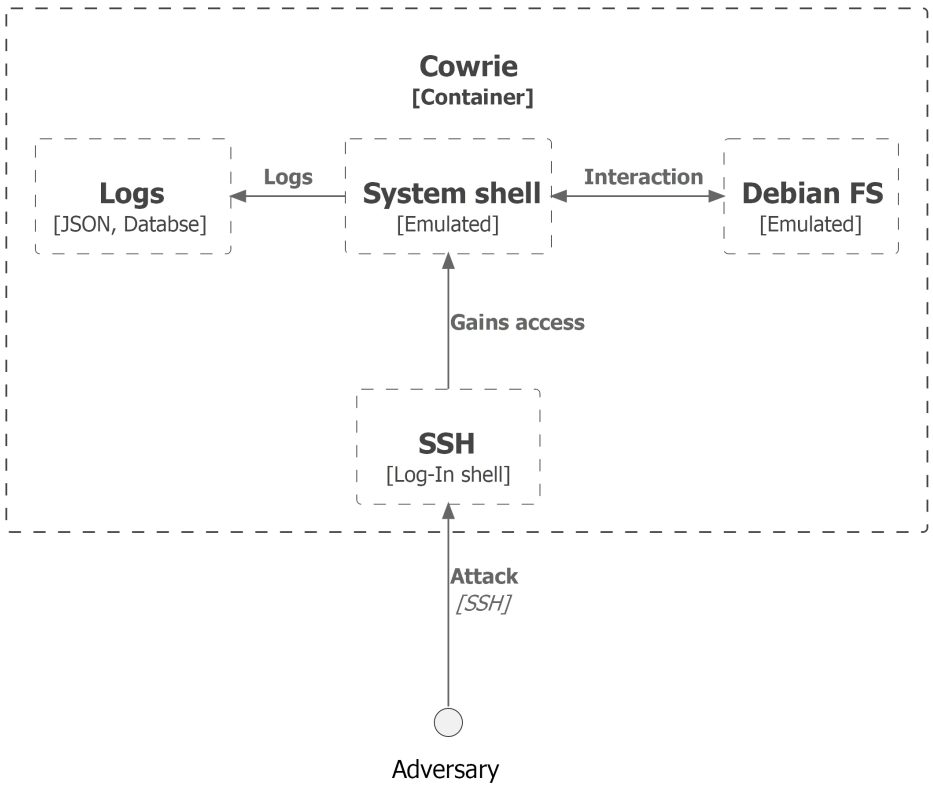


Figure 17: System architecture of the honeypots Cowrie (emulated shell mode) and Kippo.

Software architecture. The software architecture has not been changed significantly to Kippo's. However, the individual parts have increased in scope. For example, the list of command functionalities has tripled.

The Python version for Cowrie is 3.8+. Network functionality still depends on Twisted libraries, and Python libraries `bcrypt`, `cryptography`, and `pyOpenSSL` are used for hashes, cryptography, and SSL/TLS functionality.

A feature that was added after the release of (Vetterl and Clayton, 2018), is the possibility to adjust ciphers, MAC, and compression methods used by the SSH implementation. Next to the already existing list of SSH banners, this allows for deeper customization to prevent the honeypot's detection.

Logging. Logging functionality was improved to the predecessor. Text logs in the form of JSON, and a database connection are still available. Possibilities to send Cowrie's output to an ELK stack, Graylog, or Azure Sentinel were added. Files that are downloaded from or uploaded to Cowrie are still saved for later review.

Fingerprinting. While Cowrie deployments were fingerprinted in (Vetterl and Clayton, 2018) and (Srinivasa et al., 2021a), this involves a very high level of effort and time investment, especially for the up-to-date version. Since the developers adapted the responses of the underlying Twisted library (at least for SSH), fingerprinting the implementation requires a lot of overhead in SSH messages.

More than half the identified Cowrie instances detected by (Srinivasa et al., 2021a) were only labeled with a Shodan honeyscore of 0.0 or 0.3. Thus, they are difficult to identify with moderate effort. Honeyscore is described in Section 3.4.

Deployment options. Cowrie comes with a Dockerfile and an official image. The size of 382 MB as well as the availability of different architectures allow for versatile application. The image is based on a slim version of Debian Bullseye and offers options for `AMD`, `ARM`, and `MIPS`, among others.

Popularity. Cowrie is one of the most well-known honeypots for SSH. It is not only mentioned in every honeypot survey or comparison, but 4.1 thousand stars and 782 forks on GitHub show the open-source community's recognition.

3.2.15. Sshd-honeypot

Sshd-honeypot (amv42, 2018) is a modified OpenSSH implementation using Cowrie as back-end (amv42 and Oosterhof, 2018) for command interpretation and logging. The additional middleman was implemented to circumvent fingerprinting of the SSH library (Twisted.Conch) used by Cowrie (Vetterl and Clayton, 2018).

System architecture. Sshd-honeypot has a straightforward system architecture which is illustrated in Figure 18. A modified OpenSSH 7.3p1 daemon builds the honeypot. To ensure Cowrie functionalities, a modified Cowrie serves as the back-end. The SSH daemon listens on port 65222 by default. Thus, it requires an `iptables` rule to forward incoming SSH traffic from port 22. The port can be changed in the `sshd_config` file.

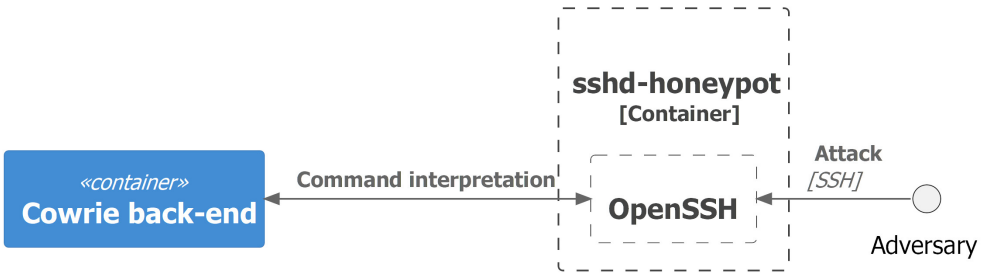


Figure 18: Architecture of the sshd-honeypot. The modified OpenSSH server utilizes a Cowrie back-end for command interpretation.

Software architecture. Since this honeypot depends on a real OpenSSH implementation, sshd-honeypot was written in C. It depends on libssh as an SSH library. `Release 7.3p1` is the version of OpenSSH that has been modified.

Logging. The back-end Cowrie is only modified to interpret commands for sshd-honeypot, logging capabilities persist.

Fingerprinting. The authors mention that this architecture was created to circumvent fingerprinting presented in (Vetterl and Clayton, 2018). This SSH honeypot cannot be distinguished from an OpenSSH 7.3p1 server from the outside. Sshd-honeypot nullifies fingerprinting but requires a custom-modified client for each represented OpenSSH version.

Deployment options. The honeypot runs as a daemon like a real OpenSSH server. Since there are container environments running OpenSSH servers, a Docker deployment of sshd-honeypot is possible.

Popularity. With 22 stars on GitHub sshd-honeypot is a rather small side project of Cowrie. Since the authors of (Vetterl and Clayton, 2018) presented a breakthrough in honeypot detection, it is worth mentioning possibilities to circumvent those attacks. Developed under the `BSD-3-Clause license`, it was last updated in 2018.

3.2.16. Sshesame

Sshesame (Jakab, 2023) is another medium-interaction SSH honeypot, that focuses on the attacker's behavior once they have access to the system. While the depth of the emulated system shell does not reach the likes of Cowrie, it has comprehensive configuration capabilities.

System architecture. The honeypot accepts incoming SSH connections and allows the adversary to log in to the system. As shown in Figure 19, the attacker is then presented with a system shell. The shell is emulated and executes a limited list of commands, e.g., `cat`, `su`, or `echo`.

A configuration can be passed to modify the offered SSH service; this includes the protocol banner as well as the welcome message. Sshesame was written with an emphasis on an adjustable attack surface. For this reason, the honeypot allows adjusting key exchange algorithms, ciphers, and macs used by SSH.

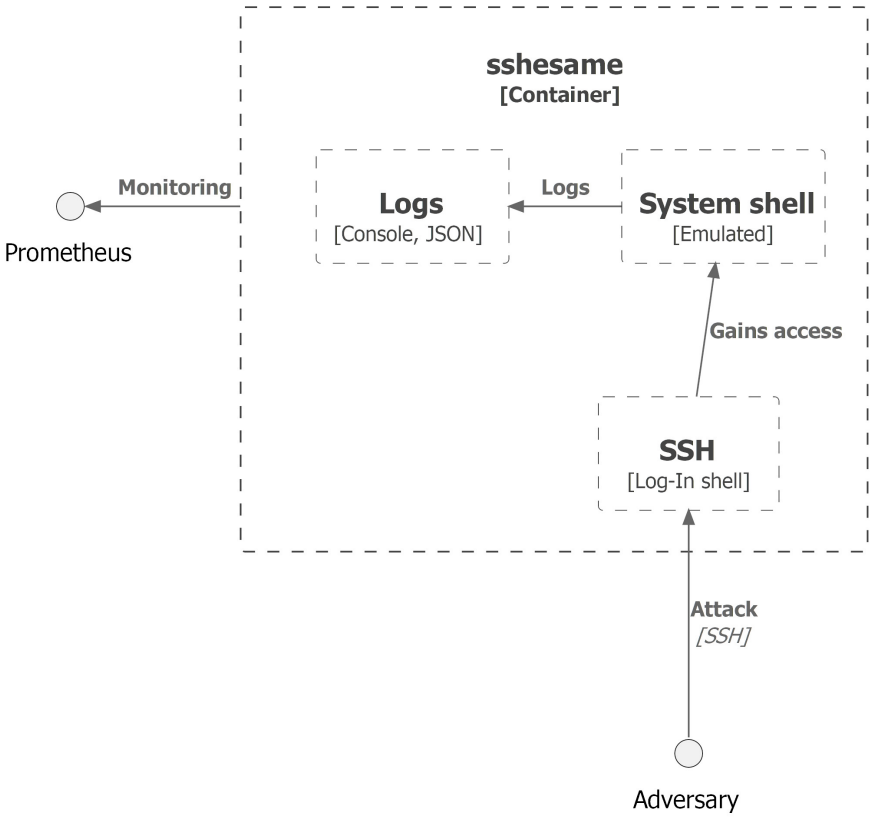


Figure 19: Architecture of sshesame.

Software architecture. Sshesame is written in GO and the latest release v0.0.27 depends on GO-lang 1.14.0. Every functionality is implemented in an individual script which allows for an expansion of functionalities. The implementation depends on standard GO crypto libraries (Golang.org, 2023) like `crypto/ssh` or `crypto/x509` for certificates, signatures, and SSH. Configurations are passed as `YAML` file.

Logging. On default sshesame logs connections and shell commands on the system console. The honeypot allows configuring JSON logging as well as a connection to Prometheus. Prometheus (Prometheus, 2023) is an open-source tool for system monitoring.

Fingerprinting. Sshesame is a unique case because the honeypot allows configuring every parameter used in the SSH handshake. This does not prevent extensive probing of the implementation, but it does prevent passive fingerprinting by recording only the SSH handshake. Fingerprinting is therefore a significant overhead for the adversary.

Deployment options. Sshesame provides introductions to run as a daemon or a Docker deployment. The Docker image is set up in a multi-stage build process: the code is first compiled in a GO-lang base image and then deployed in a slim Alpine or distroless image. This results in a comparatively very small image with `33.8MB` of size. Furthermore, sshesame provides builds for `linux-amd64`, `linux-arm64`, `linux-armv6`, and `linux-armv7` architectures in its GitHub repository.

Popularity. Sshesame has gained a lot of popularity considering it is a one-person project. The honeypot has received 1.2 thousand stars and has 74 forks on GitHub. It is released and maintained under the `Apache-2.0 license`. Most of the code base is from 2021, but sshesame still receives constant updates.

3.2.17. Wetland

High-interaction environments can be built through every communication protocol granting remote access to a system. Notable functionalities are different approaches to logging and the usage of Docker containers as high-interaction environments. These honeypots are associated with the highest maintenance effort because a complete system is given to an attacker. While caused damage is minimized by using Docker environments, the risk of systems being weaponized against other targets is still an issue.

Wetland (ohmyadd, 2018) is a high-interaction SSH honeypot. It uses Docker containers to minimize the maintenance effort and the risk of high-interaction deployments. Wetland uses HonSSH's (Nicholson, 2022) networking to be a transparent proxy. The honeypot combines the low-maintenance effort of MHs with the insights of high-interaction systems. While containers are a useful tool for this, they should not be left entirely unsupervised.

System architecture. As shown in the Figure 20, the honeypot utilizes container environments. Wetland only acts as a proxy and forwards incoming SSH connections to a Linux container. The container is not managed by Wetland, rather the address of the container is set in the configuration file. As the container environment is not part of the Wetland implementation, different distributions can be chosen as the base image for the honeypot.

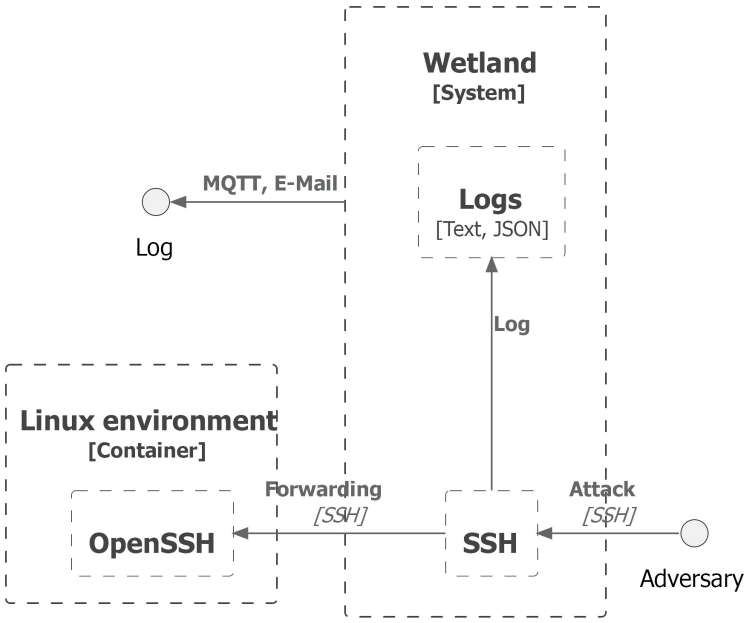


Figure 20: Architecture of the Wetland honeypot.

Software architecture. Wetland depends on Python 2.7, `Paramiko` for SSH, and `IPy` for IPv4 and IPv6 capabilities. The `main.py` script initializes the Wetland servers and services according to the configuration file `wetland.cfg.default`. Python servers for SFTP, SSH, and TCP are included.

Further, a Docker image running an SSH daemon is necessary to provide a high-interaction environment.

Logging. Wetland logs shell interaction, `SCP` and `SFTP` files, exec-commands, and reverse/direct-forward interaction of the adversary. Logs are either written to text/JSON files or can be sent via E-Mail, MQTT, or bearychat. Since logs are stored on the proxy, there is no risk of exposing logs on the honeypot machine to an adversary.

Fingerprinting. The dedicated high-interaction environment inside a container impedes fingerprinting. Since the Wetland proxy is transparent, an adversary only receives the container's SSH fingerprint—the SSH implementation can be chosen by the defender.

Deployment options. While the honeypot itself does not provide a Docker image, it relies on a Docker environment to function. As Wetland depends entirely on Python, it is possible to set it up as a container and deploy the entire honeypot system as containers.

Popularity. Wetland is a rather small project, with 120 stars and 25 forks on GitHub. Although the project has not received any commits since 2018, its early use of container technology is worth noting.

3.3. Honeypot Frameworks

Honeypot frameworks combine a collection of honeypots to offer a wider range of possibilities. A central configuration and deployment of honeypots for different protocols and services as well as a joint data collection are reasons for a framework deployment. This section provides an overview of state-of-the-art honeypot frameworks or ones with unique capabilities. Rather than focusing on the features of individual honeypots, we want to highlight data collection, presentation, as well as deployment possibilities. A comprehensive overview of all surveyed open-source frameworks can be found in Table 3.

Honeyd (Provos, 2004) not only was one of the first honeypot frameworks but implements functionalities rarely seen in other honeypot implementations. The framework is able to deploy multiple fake hosts inside the local network and simulate their network stack. TCP, UDP, and ICMP packets as well as scans of closed ports are answered correctly. The user can add arbitrary latency between hosts into the configuration to deceive traceroute tools. In addition, the framework supports the use of different

Table 3: Overview of open-source honeypot frameworks.

Framework	Last Maintained	Type
Honeyd (Provos, 2004)	2007	Honeynet
T-Pot (Deutsche Telekom Security GmbH, 2023)	2023	Collection of 23 honeypots
Chameleon (QeeqBox, 2022)	2022	Collection of 19 honeypots
OWASP Honeypot Framework (The OWASP Foundation, 2023)	2023	Deployment of modules
HoneyDB (Deception Logic, Inc, 2022a)	2022	Distributed honeypot network

TCP/IP fingerprints—these are used when remotely identifying OSs—for each host to simulate a real system.

T-Pot (Deutsche Telekom Security GmbH, 2023) is a honeypot framework developed by Deutsche Telekom Security GmbH looking to be an all-in-one solution for honeypot deployment. The framework contains more than 20 honeypots that can be deployed via a Docker container; including well-known implementations for different services, for example, SSH (Cowrie), HTTP (SNARE/TANNER), and SMTP. To complement a large number of deployable honeypots, the framework also contains a web interface to analyze the traffic seen by the deployed instances. Tools like p0fv3 (Zalewski, 2014), fatt (Adel Karimi, 2022) and geoip-attack-map (eddie4 and May, 2020) are used to gather data and visualize it afterward. T-Pot is designed for deployment on local machines and virtual machines in a cloud setup.

Chameleon (QeeqBox, 2022) contains customizable honeypots for 19 different protocols and services, including SSH, HTTP, E-Mail, and database protocols to analyze network traffic, scanning attempts, and credentials. The results are presented on a Grafana (Grafana Labs, 2023) web interface. The individual honeypots are written in Python and deployed into Chameleon via a Docker container. For most protocol implementations Twisted is used as a protocol library; protocol identification banners are customizable. Incoming traffic payloads are parsed and compared with typical patterns. The simple automation process supports cloud (AWS EC2) deployment.

OWASP Python Honeypot (The OWASP Foundation, 2023) is a honeypot framework supporting Python 3.x. The framework currently implements modules for SSH, FTP, HTTP, ICS, and SMTP for demonstration purposes. Further, the Docker-based modules come with a weak password version—these grant easy access for any adversary to monitor attacker behavior—as well as a strong password version to monitor brute-force attempts. The SSH module is implemented through the `Paramiko` library and reveals that when fingerprinted. Configuration possibilities include port configuration as well as a reset feature to automatically reset containers after a given time period in case of corruption. An API server and Elasticsearch (Elastic NV, 2023) are used to log data collected by honeypots and a Grafana Web UI for representation. Elasticsearch as well as the API server do not need to run on the same system as a honeypot instance and can be dockerized.

HoneyDB-Agent (Deception Logic, Inc, 2022a) is a honeypot framework for several different services, including SSH, Telnet, FTP, and HTTP. Even though the agent can log attack activity locally and function as a standalone honeypot, the ultimate goal is to contribute honeypot findings to the HoneyDB website (Deception Logic, Inc, 2022a). Thus, creating a network of honeypots that covers large parts of the globe. Incoming data is accumulated and a general overview is provided about the top attacked hosts and services. All data can be accessed via HoneyDB REST API and there is no need for local log management (Deception Logic, Inc, 2022b). The interaction level provided by the individual service honeypots depends on the functionalities of the emulation plugin.

3.4. Tools for Honeypot Testing

In this section, we discuss tools that provide useful functionality for honeypot development. In particular, these tools help the developer verify that the honeypot is correctly mimicking the target system. Another subset of these tools supports the implementation of packet inspection and deception techniques.

3.4.1. Network Scans

Network scans are an important first step when investigating a target system. Identifying running services can reveal critical information about the target system.

Nmap (Gordon Lyon, 2022), short for "Network Mapper", is an open-source tool for network reconnaissance. While the prevalent use is in port scanning, Nmap has a wide range of other functionalities to offer. Scans can be used to detect the OS running on a remote host by comparing the TCP/IP fingerprint to the Nmap OS database. Further, Nmap is able to do version detection of running service applications by inspecting IP packets sent by the remote host. This tool can be used for broad network scans as well as closer reconnaissance of single targets and is an important tool for attackers and defenders to get a first impression of the device in front of them.

Shodan (Shodan, 2023d) is a search engine for devices connected to the Internet. It allows the user to search for arbitrary terms and IP addresses to get general information like the autonomous system number, Internet service provider, and organization. Further, open ports are displayed as well as basic headers returned by the service. More functionalities include monitoring of owned devices and visualization of search results on a geo map interface (Shodan, 2023c). Shodan images (Shodan, 2023b) allows you to browse all screenshots collected by Shodan. Known alternatives for Shodan include Cencys (Censys, 2023) and ZoomEye (Knownsec, 2023).

3.4.2. Fingerprinting Operating Systems

While tools capable of fingerprinting OSs are also network scanners, it is useful to know not only how these OS scanning features work, but also how to deceive them.

Xprobe2 (binarytrails, 2021) is a tool designed for TCP/IP fingerprinting remote operating systems. While other OS scanners rely on static database comparisons to identify the remote OS, Xprobe2 values the results of each probe sent with a statistical approach to find the best match possible. This procedure is necessary because TCP/IP packets can be manipulated by network devices between the own device and the remote host (Arkin and Yarochkin, 2002). In this case, exact matches are difficult to obtain, therefore, a valuation of the individual results can be more accurate.

p0fv3 (Zalewski, 2014) is a passive fingerprinting tool to identify devices that communicate via TCP/IP without any active probing. In contrast to, for example, Nmap or Xprobe2, p0fv3 does not send any packets to the remote host. While this method allows one to do reconnaissance without attracting attention, it relies on being able to receive network traffic from the target device.

OSfooler-ng (Sanchez, 2019) is an open-source project that was developed to prevent the fingerprinting tools described above, from successfully identifying the OS running on the local machine; this is done without the need to modify the system kernel. For this to work, OSfooler-ng is using the databases provided by p0fv3 and Nmap to adopt the chosen TCP/IP fingerprint and fool remote network scanners.

3.4.3. Service Fingerprinting and Vulnerabilities

Once again, specialized network scanners can be used to get a better understanding of the target. What makes those tools especially useful is the capability to reveal honeypots running on a target system if those are susceptible to service fingerprinting or use default configurations known to scanners.

Hassh (Reardon et al., 2022) can be used to fingerprint SSH client and server implementations. After the initial identification message, server and client exchange their supported suites for key exchange, encryption, and authentication. Those lists are combined and hashed into an MD5 hash which can be stored and used for comparison. This tool further includes a hasshGen script to automatically generate Docker containers with a specific SSH implementation version to extract the given fingerprint. Hassh's repository includes a list of OpenSSH, Paramiko, and Dropbear SSH clients, for which the Hassh fingerprint database was already generated.

Shodan honeyscore (Shodan, 2023a) is an API offered by Shodan to evaluate whether a given host might be a honeypot. The score, ranging from 0 to 1.0, determines the probability of the host being a honeypot. As the source code is not openly available, we have no certainty about the way it operates; the authors of (Srinivasa et al., 2021a) believe that there are overlaps with their technique.

Open Vulnerability Assessment Scanner (OpenVAS) (Greenbone, 2023) is a comprehensive vulnerability scanner that combines the capabilities of many of the above tools. It creates scan reports for the entire system including lower levels of the OSI model. Scan operations can be customized and scheduled, thereby allowing automated testing of the network's systems and honeypots.

Honeyscanner (Koufakos et al., 2023) is an open-source tool that is developed to identify vulnerabilities in honeypots. As we discuss in Sections 2.4 and 4.2, attacks on honeypots are an increasing part of academic research. Honeyscanner is a tool that supports honeypot developers and maintainers

in keeping their honeypots secure. This ranges from passive attacks to active probing, denial-of-service attacks, or software library exploitation. To our knowledge, Honeyscanner is one of the first tools that specifically targets honeypots.

3.4.4. Network Packet Inspection

Network packet inspection is focusing on gathering and analyzing incoming network traffic to find interesting and new attack patterns and packets. Further, the handling of network-level packets can be useful to deceive reconnaissance tools.

Wireshark (Gerald Combs, 2022) is an open-source tool and the predominant choice for network packet inspection that can be used to analyze network traffic from a variety of different protocols. It supports live captures of traffic as well as offline analysis and decryption support for, among others, IPsec, TLS, and WPA2. The capture export possibilities allow automated analysis of the entire network traffic.

Snort (Cisco, 2022), an open-source Intrusion Prevention System (IPS), has many options for configuration and can also be used for packet sniffing and logging. Furthermore, there are implemented preprocessors for a range of popular protocols like SSH or HTTP which inspect incoming packets for known protocol exploits.

Scapy (SecDev, 2022) allows the user to inspect, manipulate, and craft network packets for a range of protocols. The Python library supports not only the manipulation of incoming and outgoing packets but also, for example, network scans, tracerouting, and attacks. It enables the specialization of capabilities of common network tools like arpspoof, tcpdump, and p0fv3 (SecDev, 2022).

4. Honeypot Research

This section provides an overview of academic research regarding honeypots, honeypot detection strategies, and honeypot evasion techniques. We found that, especially in the realm of honeypot evasion, academic research provides innovative methods that are yet to be seen in open-source solutions. To highlight more new approaches to honeypots, we present academic work independent of the target domain. An overview of honeypot evasion methods from presented publications is shown in Table 4.

4.1. Honeypots from Academic Research

Honware (Vetterl and Clayton, 2019) is a Customer Premise Equipment (CPE) and IoT honeypot framework allowing the user to upload their own firmware image and deploy it as a honeypot. This solution, based on a custom kernel is chosen by the authors to circumvent fingerprinting attacks and to improve existing emulation strategies regarding scalability.

The authors in (López-Morales et al., 2020) present a honeypot for Programmable Logic Controllers (PLCs). HoneyPLC emphasizes the credibility of the system and the automation of honeypot generation. The included PLC profiler tool scans a specified target to automatically create a PLC profile. These PLC profiles, afterward, are deployed on the honeypot. To prevent an attacker's attempt at fingerprinting, Honeyd's approach of modifying TCP/IP fingerprints is used. HoneyPLC is intended to track the rapid development of malware in the ICS domain. The honeypot is available on GitHub (López and Doupe, 2023).

In (Srinivasa et al., 2021b) the authors present RIoTPot; a honeypot for IoT and Operational Technology. The Honeypot—which is available on GitHub (Network Security Group, Wireless Communication Systems at Aalborg University, 2023)—is modular in design to allow versatility and expandability. RIoTPot can be deployed in low- and high-interaction mode, however, these modes can also be combined for different services resulting in a hybrid-interaction level. Low-interaction modules are integrated as GO-lang packages and, thus, allow the addition of custom packages. The high-interaction mode depends on container images to provide a seemingly realistic environment to the adversary. RIoTPot's currently implemented protocols are SSH, Telnet, HTTP, Modbus, MQTT, and Constrained Application Protocol (CoAP) but further protocols from the ICS domain are planned.

In (Touch and Colin, 2022) the authors compare the findings of conventional honeypots like Cowrie to those of their own honeypot Asguard (Touch and Colin, 2021). Asguard is a Reinforcement Learning (RL) agent that is deployed inside an SSH proxy to monitor attacker behavior on an HH. The RL agent is trained to prevent the high-interaction environment from being compromised by intercepting and substituting malicious command line inputs. Besides analyzing gathered honeypot data, the authors

also highlight the advantage of HHs: as long as an attacker is not able to compromise the system, high-interaction environments are the best source of attacker data. While Asguard is an approach capable of deceiving adversaries for a longer period of time compared to conventional solutions, the risk of fingerprinting attacks is still an ongoing concern discussed in the paper.

4.2. Detecting Honeypots

In addition to the methods that we discuss in Section 2.4, there are other publications on honeypot detection. In (Huang et al., 2019) the authors present an approach to automatically identify honeypot deployment. They use a machine learning model to predict a system's nature based on various input features: application layer header fields (e.g., HTTP options), network layer header fields, and other system characteristics (e.g., system fingerprint). Shodan and other scanners for Internet-connected devices are used to gather these input features. Although the authors limit their approach to HTTP, FTP, and SMTP, the successful results make this approach interesting for other protocols. In contrast to (Vetterl and Clayton, 2018) and (Srinivasa et al., 2021a), this work does not disclose which honeypots were identified.

A fuzzing-based technique to identify honeypots is presented in (Sun et al., 2020). As in traditional fuzzing, the authors aim to provoke error handling or unintended use cases to distinguish between honeypots and genuine systems. They probe systems on the Internet and evaluate the outputs supported by machine learning. Since Internet probing cannot be executed as aggressively as regular fuzzing (e.g., causing system crashes) network packets and mutation strategies are constrained.

Fingerprinting adaptive honeypots—honeypots that try to automatically identify threats—is discussed in (Obaidat et al., 2021). The authors present a concept to confuse adaptive honeypot solutions. By repeatedly attacking honeypots with non-malicious network packets or command line inputs, those adaptive systems learn to identify them as a threat. Those same inputs can subsequently be used to force a reaction of honeypot instances that deploy the identical threat model.

4.3. Evasion Strategies

The authors in (Albanese et al., 2014) discuss an algorithmic approach to obscure an attacker's view of the network. The algorithm assigns each device a changing state primitive. A primitive consists of the definition of an OS fingerprint and methods for protocol scrubbing—the change of service defined fields in network protocol headers (Watson et al., 2004)—that a client adopts temporarily. Presenting a manipulated view of the systems within the network prevents an attacker from reliably identifying OSs and services running on each device. This information is used to target specific hosts with known vulnerability exploits. Although the method is developed to further secure real network hosts, the idea of manipulating service or OS fingerprints to provide a different image of a device is a useful approach in the context of honeypot evasion.

The authors in (Naik et al., 2018) address fingerprinting of LHs by doing in-depth research about TCP, UDP, and ICMP probing. Further, they develop an approach to detect fingerprinting of honeypots. The system evaluates incoming TCP options and flags as well as UDP and ICMP packets to detect an ongoing fingerprinting attempt. It is tested against Nmap OS and service fingerprinting attacks and is able to classify the majority of attacks with a medium to high attack probability rating. Understanding the procedure of active fingerprinting attacks is important to prevent honeypots from being easily identified. Further, the identification of such attacks is helpful when analyzing scanning attempts in honeypot-acquired data.

Honeypot detection and evasion is an ongoing race between attackers and defenders. For example, malware infections are accompanied by initial checks to identify honeypots. In (Dowling et al., 2019), the authors present a honeypot that adapts to honeypot detection methods using RL. The approach specifically targets automated attacks and attempts to find optimal responses to the attacker's command line inputs. Evaluation results show that bots can use a large command sequence to test systems for authenticity, and that the presented honeypot is able to learn and withstand these sequences in case of repeated attacks.

The ability to send decoy traffic between honeypots is looked at in (Siniosoglou et al., 2020). But rather than making use of pre-configured network traffic, a deep neural network is generating realistically looking network traffic in a generative adversarial network architecture. This aims to improve Conpots' effectiveness in imitating a production device inside an ICS environment by introducing authentic network traffic to the honeypot.

Another approach to decoy traffic is presented in (Miah et al., 2022). In addition to adversarial learning, the authors used game theoretic models to optimize the decoy traffic. This supports decisions on how much traffic of which kind should be deployed. Furthermore, the authors create their models with software-defined networking in mind: decoy flows are designed to mislead attackers who passively monitor the network.

In (Asrigo et al., 2006) the authors implement three different monitoring designs on virtualization platforms to monitor honeypots. The sensors monitor kernel executes and can interrupt kernel instructions. This not only allows for event logging but, in theory, can also be used to limit an attacker's actions. In the evaluation, the authors show that VMMs are capable of efficiently logging an adversary's activity on the honeypot. Nevertheless, research has also shown that fully transparent monitoring is almost impossible (Garfinkel et al., 2007). Due to the increasing performance of hardware and virtualization techniques, however, timing discrepancies are difficult to detect, especially in a network where latency occurs.

In (Du et al., 2021) the authors present a logging approach based on the record and replay functionalities of VMs. The VMM captures activities on the honeypot and defenders can retrace all the attacker's steps in replay afterwards. To not record every trivial brute-force attack, the framework has an on-demand record functionality that starts the recording only if adversaries take notable actions. These are defined by static rules, e.g., certain command executions. Replays can also be automatically scanned for interesting events to provide a useful entry point.

The authors in (Mohammadzad and Karimpour, 2023) present a local monitoring technique that does not require virtualization. They utilize Direct Kernel Object Manipulation (DKOM), a method used by modern rootkits to hide their existence on infected systems; DKOM manipulates kernel data structures to hide data and processes. Usually, adversaries use rootkits to install backdoors, key loggers, and other monitors onto a victim's machine. In this case, the same technique is used to monitor the adversary's behavior on a honeypot. Within the evaluation, the authors show that their honeypot implementation leaves no traces in memory and requires only low kernel modifications.

Table 4: Summary of honeypot evasion techniques from academic research.

Evasion mechanism	**Description**	**References**
Manipulate network-facing attack surface	Honeypots adapt packets and header fields at the protocol level to prevent detection.	Albanese et al.; Naik et al.; Siniosoglou et al. López-Morales et al.; Miah et al.
Learn detection mechanisms	Honeypot learn and prevent automated detection mechanisms.	Dowling et al.
Hide monitoring	Honeypots hide that the system is monitoring the adversary's approach.	Asrigo et al.; Du et al. Mohammadzad and Karimpour
Defuse environment	Honeypots provide seemingly real environments that are protected from abuse.	Vetterl and Clayton; Srinivasa et al. Touch and Colin

5. Discussion of Honeypot Detection

During our work with the various honeypots and our research regarding the topic, we encountered different types of honeypots and frameworks. Whether the honeypot is a generic solution trying to detain bots or a specialized environment that targets human attackers, it can only operate as intended whilst not revealed as such. Recent research in fingerprinting has not only posed new challenges to honeypots but also brought new approaches to deception using container deployments.

In (Vetterl and Clayton, 2018), fingerprinting was entitled as not fixable for current honeypot architectures. While we agree that the capability to reveal honeypots is severe, we believe that differentiation by use case is necessary. The ease of fingerprinting must match the desired attacker profile. Crafting various protocol strings to detect different honeypots is an additional expense as well as a lot of overhead for botnets or automated scans which are targeting millions of devices. Thus, LHs are probably at a lower risk of being detected, simply because the attack is not worth the additional cost. This being said, fingerprints taken through the protocol handshake must be considered even for low-interaction devices. This kind of detection is trivial and done without a lot of overhead in computing or time investment.

For specialized systems and higher interaction levels, the effort and cost of fingerprinting must increase. Since high-interaction environments target human attackers, the time invested into individual attacks rises. Thus, it is expected that the adversary will take the time to investigate the system. The

same holds true for specialized environments: adversaries that are targeting uncommon systems with extraordinary services or operating systems are probably familiar with the target environment and will detect superficial replicas.

Kippo, Cowrie, Glastopf, Dionaea, and Conpot were all fingerprinted in (Vetterl and Clayton, 2018) and (Srinivasa et al., 2021a). While in (Srinivasa et al., 2021a) default configurations were a major reason for the detection of many deployments, the authors of (Vetterl and Clayton, 2018) used active probing. Both discovered a particularly large number of deployments of outdated versions. As a result of the approach in (Vetterl and Clayton, 2018) the developers of Cowrie customized the protocol responses of the utilized SSH library and developed the proxy sshd-honeypot. Although this reaction is optimal and desirable, it is very time-consuming, especially as the number of supported protocols and protocol versions is increasing. For highly customizable environments with a lot of different protocols, the value of findings does not justify the effort. This has also been observed in our work with the surveyed honeypots. Versatility in the offered protocols and generic use cases are mostly coupled with easier fingerprinting. Honeypots which are specialized on one protocol or have a very specific use case, are way more difficult to detect. An example of such a specialized honeypot is SNARE/TANNER which is used for web-application cloning. While this allows only HTTP usage, the quality of the impression is more convincing. Another approach is the use of containers as a proxy, replacing the fingerprint of the honeypot service with a real service implementation. This technology is rather used by MHs or HHs like (amv42, 2018) or (ohmyadd, 2018) to justify the additional expense.

Despite the benefits of container environments, they should not be used carelessly. Misconfigurations of the Docker environment can lead to lesser security or even vulnerabilities (The OWASP Foundation, 2022). For example, exposed docker daemons have already caused the takeover of containers (Palo Alto Networks, 2021) and honeypots (Sebastian Walla, 2022).

In Sections 3.2.10 and 4, we mention attempts to use machine learning to improve the deception of honeypots. While those approaches seem promising to improve a honeypot's cover, we have not yet seen such approaches in current open-source software.

6. Conclusion

Honeypots are valuable decoy resources that can mimic various types of systems. In this survey, we covered open-source honeypots with different focuses and levels of detail. Whether one seeks a web application honeypot, an SSH honeypot, or a honeypot mimicking a mail server, there are solutions for most use cases. Especially bigger open-source projects offer high quality due to the great number of contributors.

During the interaction with deployed honeypots, we observed that customization for different use cases comes with a trade-off in deceptiveness. This is confirmed by recent research in honeypot discovery: honeypots that specialize in mimicking a single system or service are more difficult to detect than their customizable counterparts. But depending on the type of attacker that is targeted by the honeypot, discovery methods do not nullify the honeypot's value. Especially automated attacks targeting a large number of devices cannot invest the additional overhead that is needed to detect the presence of such a decoy system.

Another observation is the widespread utilization of container technology. Almost all honeypots offer docker images or even rely on containers as a honeypot or proxy. Although we see great opportunities in container deployments, especially for automation purposes, they certainly introduce new challenges regarding the security of the honeypot.

During the course of this paper, we worked with open-source solutions for many use cases. These are ideally not deployed out of the box without preparation. While most are deployment-ready, default configurations should be changed, and there might be a need to further customize protocol responses if specialized environments are wanted. However, this is the advantage of open-source solutions, as they allow both: near-instant deployment and protocol-level customization for experienced users.

Acknowledgments

We thank all our colleagues who joined us to discuss honeypots and their applications.

References

National Vulnerability Database, 2018. CVE-2012-1823 Detail. https://nvd.nist.gov/vuln/detail/cve-2012-1823. Last Accessed: 2023-01-25.

Adel Karimi, 2022. FATT /fingerprintAllTheThings. https://github.com/0x4D31/fatt. Last Commit: c29e553 on 23 Mar 2022.

aiohttp contributors, 2023. Welcome to AIOHTTP. https://docs.aiohttp.org/en/stable/. Last Accessed: 2023-01-25.

Alata, E., Nicomette, V., Kaaniche, M., Dacier, M., Herrb, M., 2006. Lessons learned from the deployment of a high-interaction honeypot, in: Sixth European Dependable Computing Conference, pp. 39–46. doi:10.1109/EDCC.2006.17.

Albanese, M., Battista, E., Jajodia, S., Casola, V., 2014. Manipulating the Attacker's View of a System's Attack Surface, in: 2014 IEEE Conference on Communications and Network Security, pp. 472–480. doi:10.1109/CNS.2014.6997517.

amv42, 2018. sshd-honeypot. https://github.com/amv42/sshd-honeypot. Last Commit: f588301 on 20 Dec 2018.

amv42, Oosterhof, M., 2018. cowrie-sshd. https://github.com/amv42/cowrie-sshd. Last Commit: c53f4c9 on 20 Dec 2018.

Angelo Dell'Aera, 2022. libemu - x86 emulation and shellcode detection. https://github.com/buffer/libemu. Last Commit: eb727eb on 14 Jun 2022.

Arkin, O., Yarochkin, F., 2002. A Fuzzy Approach to Remote Active Operating System Fingerprinting. https://github.com/binarytrails/xprobe2/blob/master/docs/xprobe2-defcon10.pdf.

Asrigo, K., Litty, L., Lie, D., 2006. Using VMM-Based Sensors to Monitor Honeypots, in: Proceedings of the 2nd international conference on Virtual execution environments, pp. 13–23.

Baecher, P., Koetter, M., Holz, T., Dornseif, M., Freiling, F., 2006. The Nepenthes Platform: An Efficient Approach to Collect Malware, in: International Workshop on Recent Advances in Intrusion Detection, Springer. pp. 165–184.

Bajpai, P., Enbody, R., Cheng, B.H., 2020. Ransomware Targeting Automobiles, in: Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security, Association for Computing Machinery. pp. 23–29. URL: https://doi.org/10.1145/3375706.3380558, doi:10.1145/3375706.3380558.

Bilenko, D., 2019. gevent - a Python networking library. http://www.gevent.org/index.html. Last Accessed: 2023-01-23.

binarytrails, 2021. Xprobe2. https://github.com/binarytrails/xprobe2. Last Commit: f14af2e on 22 Mar 2021.

Brown, S., Lam, R., Prasad, S., Ramasubramanian, S., Slauson, J., 2012. Honeypots in the Cloud.

Carr, J., Schloesser, M., Lombardo, D., Huanjie, Z., Durechova, K., Werner, T., 2021. hpfeeds. https://hpfeeds.org/. Last Accessed: 2023-01-12.

Censys, 2023. Censys Internet Scanning Intro. https://support.censys.io/hc/en-us/articles/360059603231-Censys-Internet-Scanning-Intro. Last Accessed: 2023-02-02.

Cisco, 2022. Snort. https://www.snort.org/. Last Accessed: 2022-07-06.

Dalamagkas, C., Sarigiannidis, P., Ioannidis, D., Iturbe, E., Nikolis, O., Ramos, F., Rios, E., Sarigiannidis, A., Tzovaras, D., 2019. A Survey On Honeypots, Honeynets And Their Applications On Smart Grid, in: IEEE Conference on Network Softwarization (NetSoft), IEEE. pp. 93–100.

Deception Logic, Inc, 2022a. HoneyDB. https://honeydb.io/. Last Accessed: 2022-07-13.

Deception Logic, Inc, 2022b. HoneyDB Agent Docs. https://honeydb-agent-docs.readthedocs.io/en/latest/. Last Accessed: 2022-07-13.

Deutsche Telekom Security GmbH, 2023. T-Pot - The All In One Multi Honeypot Plattform. https://github.com/telekom-security/tpotce. Last Commit: 9941818 on 12 May 2023.

DinoTools, 2015. dionaea Docs - Introduction. https://dionaea.readthedocs.io/en/latest/introduction.html. Last Accessed: 2022-06-16.

DinoTools, 2021. dionaea honeypot. https://github.com/DinoTools/dionaea. Last Commit: 4e459f1 on 8 Feb 2021.

DinoTools, 2023. Official image for dionaea a low interaction honeypot. https://hub.docker.com/r/dinotools/dionaea. Last Updated: on 01 Feb 2023.

Dowling, S., Schukat, M., Barrett, E., 2019. Using Reinforcement Learning to Conceal Honeypot Functionality, in: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part III 18, Springer. pp. 341–355.

Du, C., Zhao, S., Wang, W., 2021. Rrpot: A record and replay based honeypot system. Journal of Physics: Conference Series 1757, 012183. doi:10.1088/1742-6596/1757/1/012183.

eddie4, May, M., 2020. Cyber security geoip attack map. https://github.com/eddie4/geoip-attack-map. Last Commit: 7d34b27 on 26 Jun 2020.

Elastic NV, 2023. Elasticsearch - The heart of the free and open Elastic Stack. https://www.elastic.co/elasticsearch/. Last Accessed: 2023-08-08.

European Union Agency for Cybersecurity, 2022. ENISA threat landscape 2022: July 2021 to July 2022. European Network and Information Security Agency. doi:doi/10.2824/764318.

Fan, W., Du, Z., Fernández, D., Villagra, V.A., 2017. Enabling an Anatomic View to Investigate Honeypot Systems: A Survey. IEEE Systems Journal 12, 3906–3919.

Forcier, J., Gaynor, A., 2022. Paramiko - The leading native Python SSHv2 protocol library. https://www.paramiko.org/. Last Accessed: 2022-07-06.

Franco, J., Aris, A., Canberk, B., Uluagac, A.S., 2021. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. IEEE Communications Surveys & Tutorials 23, 2351–2383.

Furuhashi, S., 2021. MessagePack: Efficient Binary Serialization Format. https://msgpack.org. Last Accessed: 2023-03-29.

Fyodor, Y., 2014. honeyntp - NTP logger/honeypot. https://github.com/fygrave/honeyntp. Last Commit: 0c9d938 on 27 Mar 2014.

Garfinkel, T., Adams, K., Warfield, A., Franklin, J., 2007. Compatibility Is Not Transparency: VMM Detection Myths and Realities., in: HotOS.

Gerald Combs, 2022. Wireshark - Go Deep. https://www.wireshark.org/. Last Accessed: 2022-07-06.

Gokhale, S., Dalvi, A., Siddavatam, I., 2020. Industrial Control Systems Honeypot: A Formal Analysis of Conpot. International Journal of Computer Network & Information Security 12.

Golang.org, 2023. Go Cryptography. https://pkg.go.dev/golang.org/x/crypto/. Last Accessed: 2023-01-31.

Gordon Lyon, 2022. Nmap: Discover your network. https://nmap.org/. Last Accessed: 2022-07-06.

Grafana Labs, 2023. Grafana. https://github.com/grafana/grafana. Last Commit: 98f3b5f on 16 Aug 2023.

Greenbone, 2023. openvas-scanner. https://github.com/greenbone/openvas-scanner. Last Commit: cab4d7c on 07 Aug 2023.

Huang, C., Han, J., Zhang, X., Liu, J., 2019. Automatic Identification of Honeypot Server Using Machine Learning Techniques. Security and Communication Networks 2019, 1–8.

IVRE, 2023a. IVRE - Network recon framework. https://github.com/ivre/ivre. Last Commit: 5f67435 on 16 Jan 2023.

IVRE, 2023b. masscanned. https://github.com/ivre/masscanned. Last Commit: 511c816 on 30 Mar 2023.

Jakab, K., 2023. sshesame. https://github.com/jaksi/sshesame. Last Commit: 2036179 on 18 Jan 2023.

Jose Nazario, 2022. an awesome list of honeypot resources. https://github.com/paralax/awesome-honeypots. Last Commit: a87cce9 on 29 Nov 2022.

Knownsec, 2023. ZoomEye. https://www.zoomeye.org/. Last Accessed: 2023-04-04.

Koufakos, A.C., Vasilomanolakis, E., Srinivasa, S., Yaben, R., 2023. Honeyscanner: A vulnerability analyzer for honeypots. https://github.com/honeynet/honeyscanner. Last Commit: 7adf647 on 09 Aug 2023.

limifly, 2015. A Python based ntp server. https://github.com/limifly/ntpserver. Last Commit: 69ec28b on 06 Apr 2015.

López-Morales, E., Rubio-Medrano, C., Doupé, A., Shoshitaishvili, Y., Wang, R., Bao, T., Ahn, G.J., 2020. HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Association for Computing Machinery, New York, NY, USA. p. 279–291. doi:10.1145/3372297.3423356.

López, E., Doupe, A., 2023. honeyplc - High-interaction Honeypot for PLCs and Industrial Control Systems. https://github.com/sefcom/honeyplc. Last Commit: 6188234 on 16 May 2023.

Matsubara, Y., 2023. PyMySQL - Pure Python MySQL Client. https://github.com/PyMySQL/PyMySQL. Last Commit: e91d097 on 09 Jan 2023.

MaxMind, 2023. Python code for GeoIP2 webservice client and database reader. https://github.com/maxmind/GeoIP2-python. Last Commit: cf2d16f on 17 Jan 2023.

Miah, M.S., Zhu, M., Granados, A., Sharmin, N., Anjum, I., Ortiz, A., Kiekintveld, C., Enck, W., Singh, M.P., 2022. Optimizing Honey Traffic Using Game Theory and Adversarial Learning, in: Cyber Deception: Techniques, Strategies, and Human Aspects. Springer, pp. 97–124.

Mohammadzad, M., Karimpour, J., 2023. Using rootkits hiding techniques to conceal honeypot functionality. Journal of Network and Computer Applications 214, 103606. URL: https://www.sciencedirect.com/science/article/pii/S1084804523000255, doi:https://doi.org/10.1016/j.jnca.2023.103606.

Mòrian, 2019. Blacknet 2. https://github.com/morian/blacknet. Last Commit: d0a5730 on 20 Dec 2019.

MushMush Foundation, 2016. Welcome to TANNER's documentation! https://tanner.readthedocs.io/en/latest/. Last Accessed: 2023-02-01.

MushMush Foundation, 2018. Welcome to SNARE's documentation! https://snare.readthedocs.io/en/latest/. Last Accessed: 2023-02-01.

MushMush Foundation, 2021a. Glastopf - Web Application Honeypot. https://github.com/mushorg/glastopf. Last Commit: d17fcb6 on 16 Oct 2021.

MushMush Foundation, 2021b. SNARE - Super Next generation Advanced Reactive honEypot. https://github.com/mushorg/snare. Last Commit: 0919a80 on 13 Jun 2021.

MushMush Foundation, 2022a. Conpot - ICS/SCADA honeypot. https://github.com/mushorg/conpot. Last Commit: f0e6925 on 29 Jun 2022.

MushMush Foundation, 2022b. TANNER - He who flays the hide. https://github.com/mushorg/tanner. Last Commit: 2fdce2e on 16 Jan 2022.

MushMush Foundation, 2023. Glutton - Generic Low Interaction Honeypot. https://github.com/mushorg/glutton. Last Commit: c896cd5 on 02 Apr 2023.

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, in: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–8. doi:10.1109/FUZZ-IEEE.2018.8491456.

Nawrocki, M., Wählisch, M., Schmidt, T.C., Keil, C., Schönfelder, J., 2016. A Survey on Honeypot Software and Data Analysis. ArXiv e-prints: 1608.06249 .

netfilter.org, 2021a. The netfilter.org "iptables" project. https://www.netfilter.org/projects/iptables/index.html. Last Accessed: 2023-02-02.

netfilter.org, 2021b. The netfilter.org "libnetfilter_queue" project. https://www.netfilter.org/projects/libnetfilter_queue/index.html. Last Accessed: 2023-02-02.

Network Security Group, Wireless Communication Systems at Aalborg University, 2023. RIoTPot - IoT and Operational Technology Honeypot. https://github.com/aau-network-security/riotpot. Last Commit: 7475a3a on 07 Jun 2023.

Nicholson, T., 2022. HonSSH. https://github.com/tnich/honssh. Last Commit: 821ce87 on 02 Jan 2022.

NIST, 2023. National Vulnerability Database. https://nvd.nist.gov/. Last Accessed: 2023-08-07.

OASIS Open, 2022a. Introduction to STIX. https://oasis-open.github.io/cti-documentation/stix/intro. Last Accessed: 2023-01-23.

OASIS Open, 2022b. Introduction to TAXII. https://oasis-open.github.io/cti-documentation/taxii/intro.html. Last Accessed: 2023-01-23.

OASIS Open, 2022c. OASIS Open - Setting the standard for open collaboration. https://www.oasis-open.org/. Last Accessed: 2023-01-23.

Obaidat, M., Brown, J., Alnusair, A., 2021. Blind Attack Flaws in Adaptive Honeypot Strategies, in: IEEE World AI IoT Congress (AIIoT), pp. 0491–0496. doi:10.1109/AIIoT52608.2021.9454206.

ohmyadd, 2018. wetland - A high interaction SSH honeypot. https://github.com/ohmyadd/wetland. Last Commit: 76d296e on 26 Dec 2018.

Oosterhof, M., 2023. Cowrie SSH/Telnet Honeypot. https://github.com/cowrie/cowrie. Last Commit: 65fc49e on 09 Jan 2023.

OpenVPN Inc, 2023. easy-rsa - Simple shell based CA utility. https://github.com/OpenVPN/easy-rsa. Last Commit: 2cadb05 on 28 Mar 2023.

Palo Alto Networks, 2021. Docker Honeypot Reveals Cryptojacking as Most Common Cloud Threat. https://unit42.paloaltonetworks.com/docker-honeypot/. Last Accessed: 2022-12-14.

Palo Alto Networks, 2022. Attackers Move Quickly to Exploit High-Profile Zero Days: Insights From the 2022 Unit 42 Incident Response Report. https://unit42.paloaltonetworks.com/incident-response-report/. Last Accessed: 2022-11-22.

Patzke, T., 2023. Log4Pot - A honeypot for the Log4Shell vulnerability (CVE-2021-44228). https://github.com/thomaspatzke/Log4Pot. Last Commit: e224c0f on 26 Apr 2022.

Prometheus, 2023. The Prometheus monitoring system and time series database. https://github.com/prometheus/prometheus. Last Commit: e023d89 on 31 Jan 2023.

Provos, N., 2004. A Virtual Honeypot Framework, in: Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13, USENIX Association, USA. p. 1.

Python, 2022. The Python programming language - socketserver. https://github.com/python/cpython/blob/3.11/Lib/socketserver.py. Last Commit: ecfff63 on 11 Mar 2022.

Python Software Foundation, 2023. Python/C API Reference Manual. https://docs.python.org/3/c-api/index.html. Last Accessed: 2023-02-01.

Python Standard Library, 2023. asyncio — Asynchronous I/O. https://docs.python.org/3/library/asyncio.html. Last Accessed: 2023-01-25.

QeeqBox, 2022. Chameleon. https://github.com/qeeqbox/chameleon. Last Commit: 0aad988 on 18 Apr 2022.

Rapid7, 2023. Metasploit Framework. https://github.com/rapid7/metasploit-framework. Last Commit: 253290d on 15 Aug 2023.

Reardon, B., Karimi, A., Salesforce, 2022. "HASSH" - a Profiling Method for SSH Clients and Servers. https://github.com/salesforce/hassh. Last Commit: 9a6c29a on 29 Apr 2022.

Richardson, L., 2020. Beautiful Soup Documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/. Last Accessed: 2023-01-25.

Sanchez, J., 2019. OSfooler-ng. https://github.com/segofensiva/OSfooler-ng. Last Commit: c0b20d6 on 26 May 2019.

Sanders, C., 2020. Intrusion Detection Honeypots: Detection Through Deception. Applied Network Defense. URL: https://books.google.de/books?id=suubzQEACAAJ.

Sebastian Walla, 2022. Compromised Docker Honeypots Used for Pro-Ukrainian DoS Attack. https://www.crowdstrike.com/blog/compromised-docker-honeypots-used-for-pro-ukrainian-dos-attack/. Last Accessed: 2022-12-14.

SecDev, 2022. Scapy - Packet crafting for Python2 and Python3. https://scapy.net/. Last Accessed: 2022-07-08.

Sethia, V., Jeyasekar, A., 2019. Malware Capturing and Analysis using Dionaea Honeypot, in: International Carnahan Conference on Security Technology (ICCST), pp. 1–4. doi:10.1109/CCST.2019.8888409.

Shodan, 2023a. Honeypot Or Not? https://honeyscore.shodan.io/. Last Accessed: 2023-08-03.

Shodan, 2023b. Images. https://images.shodan.io/. Last Accessed: 2023-08-03.

Shodan, 2023c. Maps. https://maps.shodan.io/. Last Accessed: 2023-08-13.

Shodan, 2023d. Search Engine. https://www.shodan.io/. Last Accessed: 2023-08-03.

Siniosoglou, I., Efstathopoulos, G., Pliatsios, D., Moscholios, I.D., Sarigiannidis, A., Sakellari, G., Loukas, G., Sarigiannidis, P., 2020. NeuralPot: An Industrial Honeypot Implementation Based On Deep Neural Networks, in: ISCC, pp. 1–7. doi:10.1109/ISCC50000.2020.9219712.

Spitzner, L., 2003. Honeypots: Tracking Hackers. volume 1. Addison-Wesley Reading.

Spitzner, L., Roesch, M., 2001. The Value of Honeypots, Part One: Definitions and Values of Honeypots. Security Focus .

Srinivasa, S., Pedersen, J.M., Vasilomanolakis, E., 2021a. Gotta Catch 'em All: a Multistage Framework for Honeypot Fingerprinting. ArXiv e-prints abs/2109.10652. URL: https://arxiv.org/abs/2109.10652, arXiv:2109.10652.

Srinivasa, S., Pedersen, J.M., Vasilomanolakis, E., 2021b. RIoTPot: A Modular Hybrid-Interaction IoT/OT Honeypot, in: Computer Security–ESORICS 2021, Springer. pp. 745–751.

Stolfo, S.J., Bowen, B.M., Ben Salem, M., 2011. Insider Threat Defense. Springer US, Boston, MA. pp. 609–611. URL: https://doi.org/10.1007/978-1-4419-5906-5_904, doi:10.1007/978-1-4419-5906-5_904.

Sun, Y., Tian, Z., Li, M., Su, S., Du, X., Guizani, M., 2020. Honeypot Identification in Softwarized Industrial Cyber–Physical Systems. IEEE Transactions on Industrial Informatics 17, 5542–5551.

The Honeynet Project, 2018. Conpot - Low interaction server side ICS honeypot. https://conpot.readthedocs.io/en/latest/. Last Accessed: 2022-07-05.

The OWASP Foundation, 2022. OWASP Docker Security Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html.

The OWASP Foundation, 2023. OWASP Honeypot, Automated Deception Framework. https://github.com/OWASP/Python-Honeypot. Last Commit: 4235acf on 09 May 2023.

The Shadowserver Foundation, 2023. ShadowServer - Network Reporting. https://www.shadowserver.org/what-we-do/network-reporting/. Last Accessed: 2023-02-02.

Thinkst Applied Research, 2023. Modular and decentralised honeypot. https://github.com/thinkst/opencanary. Last Commit: e074f15 on 24 Mar 2023.

Thinkst Canary, 2023a. OpenCanary - Welcome to the OpenCanary guide. https://opencanary.readthedocs.io/en/latest/. Last Accessed: 2023-02-01.

Thinkst Canary, 2023b. Thinkst Canary - Know. When it Matters! https://canary.tools/. Last Accessed: 2023-01-01.

Tier, R., 2022. How To Set Up an Endlessh Tarpit on Ubuntu 22.04. https://www.digitalocean.com/community/tutorials/how-to-set-up-an-endlessh-tarpit-on-ubuntu-22-04. Last Accessed: 2023-01-20.

Touch, S., Colin, J.N., 2021. Asguard: Adaptive Self-guarded Honeypot, in: 17th International Conference on Web Information Systems and Technologies-Volume 1: DMMLACS,, SciTePress. pp. 565–574.

Touch, S., Colin, J.N., 2022. A Comparison of an Adaptive Self-Guarded Honeypot with Conventional Honeypots. Applied Sciences 12. URL: https://www.mdpi.com/2076-3417/12/10/5224, doi:10.3390/app12105224.

Twisted Matrix Labs, 2023. Twisted. https://twisted.org/. Last Accessed: 2023-02-01.

Upi Tamminen, 2016. Kippo - SSH Honeypot. https://github.com/desaster/kippo. Last Commit: 0d03635 on 30 Sep 2016.

Veronica Valeros, 2022. Installing Glutton Honeypot in the Cloud. https://www.stratosphereips.org/blog/2022/5/3/installing-glutton-honeypot-in-the-cloud. Last Accessed: 2023-01-16.

Vestergaard, J., 2023. Heralding - Credentials catching honeypot. https://github.com/johnnykv/heralding. Last Commit: 6437605 on 02 Aug 2023.

Vetterl, A., Clayton, R., 2018. Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale, in: WOOT @ USENIX Security Symposium.

Vetterl, A., Clayton, R., 2019. Honware: A Virtual Honeypot Framework for Capturing CPE and IoT Zero Days, in: APWG Symposium on Electronic Crime Research (eCrime), pp. 1–13. doi:10.1109/eCrime47957.2019.9037501.

Wallen, J., 2015. An Introduction to Uncomplicated Firewall (UFW). https://www.linux.com/training-tutorials/introduction-uncomplicated-firewall-ufw/. Last Accessed: 2023-01-20.

Watson, D., Smart, M., Malan, G.R., Jahanian, F., 2004. Protocol Scrubbing: Network Security Through Transparent Flow Modification. IEEE/ACM transactions on Networking 12, 261–273.

Wellons, C., 2021. Endlessh: an SSH tarpit. `https://github.com/skeeto/endlessh`. Last Commit: dfe44eb on 30 Apr 2021.

Zalewski, M., 2014. p0f v3. `https://lcamtuf.coredump.cx/p0f3/`. Last Accessed: 2022-07-06.

Zuzčák, M., Zenka, M., 2020. Expert system assessing threat level of attacks on a hybrid SSH honeynet. Computers & Security 92, 101784.