Linux Threat Hunting Primer — Part II

By VerintCyberSec

Published: 2020-01-05 · Archived: 2026-05-07 02:22:29 UTC

10 min read

Jan 5, 2020

By Shachar Roitman

In the previous post "Linux Threat Hunting Primer — Part 1" , we discussed how to start the threat hunting process and reviewed the statistical distribution of the Linux tactics and techniques. We also created lists of techniques to search for after performing ROI estimation. Moreover, we began to list the different stages required in the process of threat hunting. In this post, we will describe and demonstrate a full threat hunting process on one MITRE ATT&CK technique.

All of the queries I'm going to show will be in the TPSQL language (Verint Threat Protection System (TPS) Query Language). Don't worry, the language is simple to understand.

TPS agents collect data using various techniques including *auditd* and custom kernel modules. You can refer to *strace*, *ptrace* and *auditd* documentation or output on your system to get additional insight about the fields and queries we're going to use in the post.

Credential Dumping — Example of a full threat hunting process

Stage 0: Understand the attack and/or technique you'd like to find

There are two important questions to ask ourselves at this point :

1. What does the attacker want to accomplish when performing the attack?
2. How is he going to do it?

What is credential dumping?

> "Credential dumping is the process of obtaining account login and password information, normally in the form of a hash or a clear text password, from the operating system and software." [MITRE ATT&CK definition, T1003]

In Linux, abusing the */proc* directory is one of the most common courses of action for this kind of attack. A common technique is reading the memory of a process from */proc/[pid]/maps* and dumping/harvesting passwords using root privileges.

As a demonstration, we will check if we can find plain text passwords in memory dumps. For the following example, I am using Linux Memory Extractor (LiMe):

The "*shachar*" user is in the *sudo* group, so this account can be used for privilege escalation. I dumped the memory using LiMe and searched for plain text passwords (using *strings* and *grep* commands).

From the users point of view:

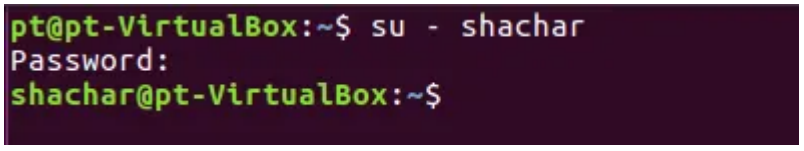
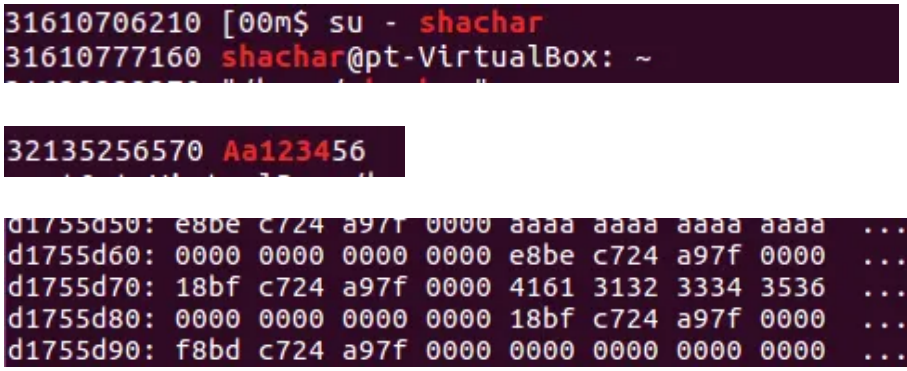


Image 1: The user wanted to perform an action, which required "root" privileges

From the attacker point of view (Memory dump):



Images 2–4: I looked for Shachar's user login action in the memory and for a password pattern

Now that we verified that some plain-text passwords can be found in memory it's clear that an attacker can also leverage this to steal passwords. Next, we'll look at the different ways this attack can be implemented.

Stage 1: Find ways to implement this technique

Now that we understand the technique, we'd like to find the variety of ways to implement it. For example, which syscalls, files or process are involved in the different stages of the attack?

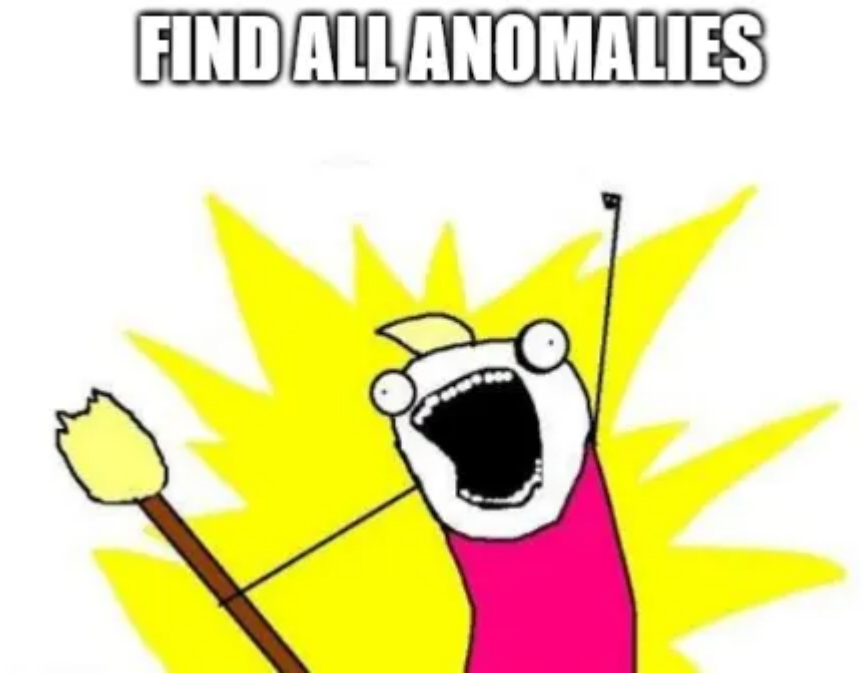
In this stage, we will see malicious tools that implement the attack and how they do it. In addition, we will look for the attack footprint on the OS.

Reading previous research papers on credential dumping (see: http://www.foo.be/cours/mssi-20072008/davidoff-clearmem-linux.pdf) confirms and reinforces the behavior that we witnessed in the previous stage. The research indicates that an attacker can see the user's account information in different processes memory, such as:

- By looking at Gnome Display Manager Process memory dump, we can see the Linux login password in ASCII as well as information from */etc/shadow* and */etc/passwd*. This includes the login shadow password, username, long name, UID, GID, home directory, and shell.
- The *thunderbird-bin* process memory contains the user's plain text email password, name, email address, mail server and related information in ASCII format.

- The cleartext SSH password was stored as ASCII text within a large block of nulls in the memory image.

Stage 2: Find anomalies



Using the information we gathered from the research on the technique. We'll now search for the suspicious behaviors in the network by focusing on anomalies (Look for techniques using multiple dimensions: parent/child relationships, command lines arguments, environment variables, accounts, permissions, memory etc.)

The following examples will help you define your searches:

Search for processes that use */proc/<pid>/maps*, */etc/passwd*, */etc/shadow* files or modifications of */etc/login.defs* file which provides the default configuration information for several user account parameters.

Pay attention that *useradd*, *usermod*, *userdel* and *groupadd* system commands as well as other user management utilities read the *login.defs* file. We can study the interaction between these commands and *login.defs* so that we can filter out expected behavior (i.e. false positives) in the next steps of our investigation

Look for users that performed file activity on memory dump files that were created by the OS.

Normally, memory crash files are located in */var/crash*, but can also be found in */var/spool* or */var/lib/systemd/coredump*

kdump is a kernel crash dumping utility. This utility can be enabled using a *systemctl* command. We can look for commands like:

- *$ systemctl enable kdump.service*
- *$ systemctl start kdump.service*

In addition, we can check for modifications to coredump sysctl config */etc/sysctl.d/50-coredump.conf*. Following are additional files that can be manipulated for malicious dumping:

- */etc/systemd/coredump.conf*
- */etc/systemd/coredump.conf.d/*.conf*
- */run/systemd/coredump.conf.d/*.conf*
- */usr/lib/systemd/coredump.conf.d/*.conf*
- */etc/systemd/systemd.conf*

We can also look for the use of memory dumping commands, like:

- *gcore*
- *cat /proc/<pid>/maps*
- *gdb -pid <pid>* Then in the GDB shell: *(gdb) dump memory /root/output offset*

And common credential dumping tools, like:

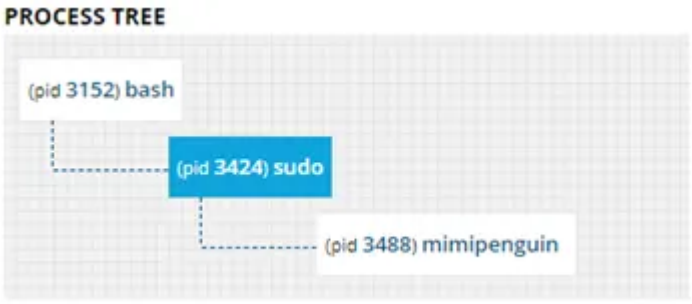
- mimipenguin
- 3snake

Let's get a more in-depth look at the these tools, starting with mimipenguin.

[mimipenguin](#) — A tool to dump login passwords of Linux users

To understand *mimipenguin* process activity we'll take a look at the process tree. More specifically, we'll search for the *gcore* utility in the process tree (See below - any *gcore* image running under *bash → sudo*) because we know *mimipinguin* uses it to dump memory.

By analyzing the process tree we can find suspicious parent-child relationships and understand the process activity:



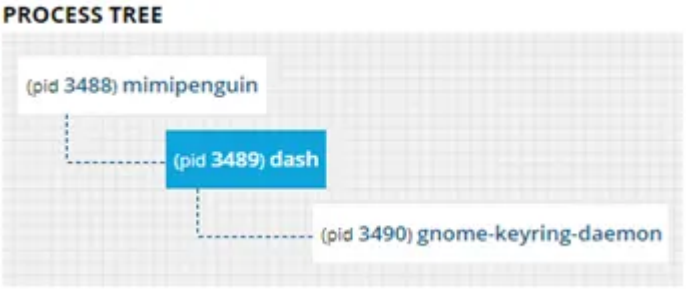


Image 5–6: Verint TPS builds a tree for each executed process. We can see in this image that "mimipenguin" runs under the "*sudo*" process who is a child of "bash" and creates a "dash" child

process that is the parent of “gnome-keyring-daemon” (service that stores passwords) process. The process tree helps us understand the full activity in the process parent — child dimension.

After we have an initial lead from the process tree, we can further investigate the raw data and analyse parameters such as the command line, path and users.

Press enter or click to view image in full size

Press enter or click to view image in full size

Image 7–8: Verint’s TPS presents full command line, image path, user and MD5 for each executed process. We can see in this image the full commands and that the process runs with “sudo” privileges under the user “root”. This is important for behavioral analysis and threat hunting.

(Another way to get this full information will be — combining the output from *ps*, *who*, *uname*, *ptrace*, *strace* and *file* commands)

Get VerintCyberSec’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

We will look for *gnome-keyring-daemon* process that does not run under its normal parent process. To understand what is “normal” we will look at a benign “gnome-keyring” process tree.

Image 9: we will look at the child and parent process of the benign “gnome-keyring-daemon” process from the process tree.

Image 10: looking for “Xsession” parent process to find “gnome-keyring-daemon” full process tree.

In conclusion, we will look for *gnome-keyring-daemon* that does not run under the following tree:

gdm-session-worker → Xsession → gnome-keyring-daemon → gnome-keyring-daemon

Now we’ll turn to look at the other tool mentioned above called “3snake”.

3snake — Dump *sshd* and *sudo* credential related strings

By understanding how “3snake” works, you can learn how to search for it in the data:

- 3snake reads memory from sshd and sudo system calls that handle password-based authentication
- It doesn’t write to the memory of the traced process
- 3snake spawns a new process for every sshd and sudo command that it runs
- Listens for the proc event using netlink sockets to get candidate processes to trace
- When it detects a running process that uses sshd or sudo, ptrace is attached and traces read and write system calls, extracting strings related to password based authentication

From the above analysis, we conclude that 3snake creates multiple threads and processes. We will look for an excessive process activity as in the process tree below:

You should look for this anomaly — multiple processes in 3 generations of the process “3snake” executing on a single endpoint.

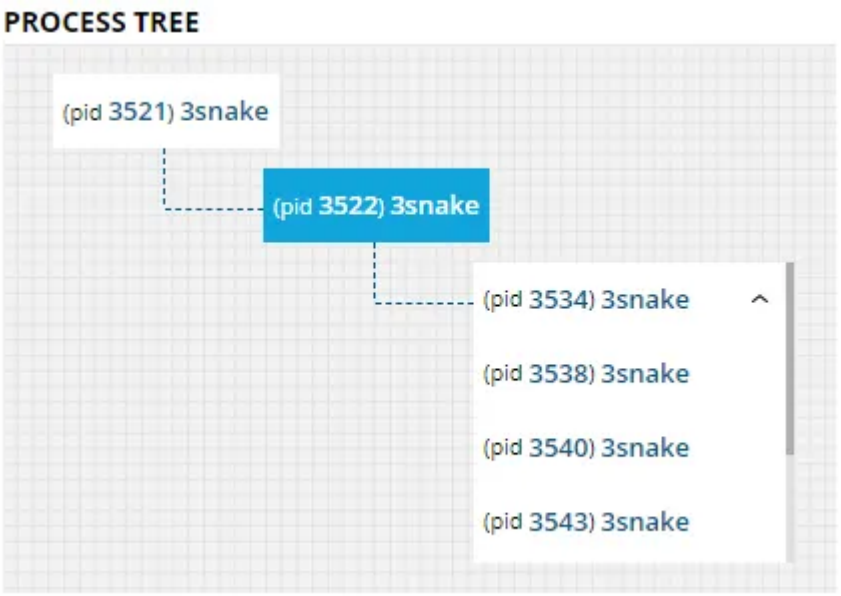


Image 11: Verint's TPS builds a tree for each executed process. In the above image, we notice that "3snake" spawns itself multiple times.

Stage 3: Filter out "normal" activities

In this stage, we'll learn the normal activity of the network. This will help you to reduce the number of results. Be careful with your assumptions, so that you don't filter out too much data or overfit to a specific system. Do not forget to refactor the query. When you finish, check if there is a technique missing or more "known good" data to filter out.

In our credential-dumping hunt, we can do the following:

1. Identify normal passwd activity, for example, look for all passwd references as a process or as a process command line:

   Press enter or click to view image in full size

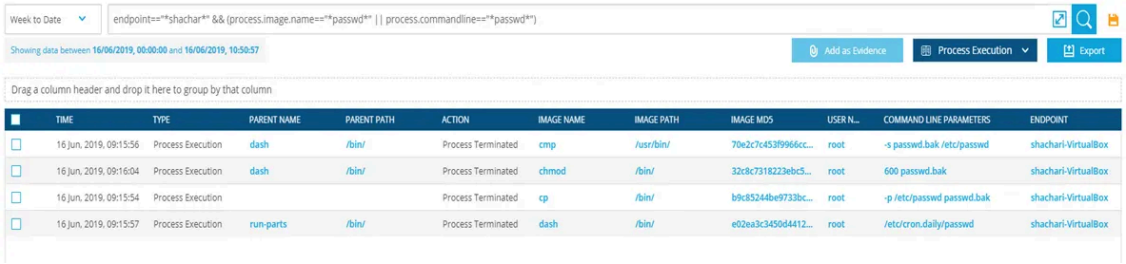


Image 12: TPS endpoint forensics agent collects process execution information from the Kernel. We can search for "passwd" file/system command reference on that data. Another way to get this full information will be — combining the output from "ps", "who", "uname","ptrace" , "strace" and "file" commands.

Press enter or click to view image in full size

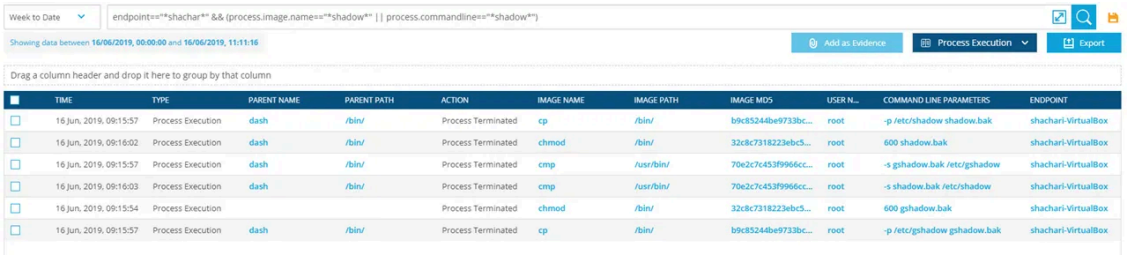


Image 13: We can search for “shadow” file\ system command reference on that data in order to baseline benign activity.

3. Check System processes normal activity for relevant commands (like *compgen*, *getent*, *passwd*, *useradd*, *groupadd*, *usermod*, *chsh*, *chfn*, *users*, *id*, *groups*, *last*, *logname*, *w*, *who*, *whoami*, *members*, *groupmod*, *finger*, *su*, *gpasswd*, *chgrp*)

4. Search whether */etc/shadow* and */etc/passwd* were copied (which are used to unshadow with ‘John the Ripper’, an open source tool used for password cracking) by the same user or at the same time.

5. Check system binary activity; Try to specify the suspicious activity by multiple parameters (command line, privileges, memory etc.) to avoid whitelisting a full binary that can be poisoned (i.e. replacing system binary with a malicious one).

For example: If you whitelist “ls” binary activity (by name and process tree only) to avoid result overload, you can miss malicious activity in case “ls” was replaced with a malicious file.

Stage 4: query, refine and repeat

Now that we understand how an attack looks like and how normal behavior looks like, we will combine both and use it for threat hunting.

Hunting for memory dumps files

We’ll look for all dump files that were not created by the *abrt* process (*abrt*, atomic bug report tool, is Linux system daemon that reads process memory and may seem suspicious).

Query example:

> (Relying on the dump file name to contain the string “dump” in it is bad, but bear with me for the sake of this explanation).

To hunt for processes that creates activity using user credentials, like:

- Processes that perform activities on files that contain passwords and don’t run unser “run-parts” , “getnet” or “abrt-watch-log” system commands , which are part of the system normal activity.
- Processes with commandline that includes important user control files like shadow, passwd, login.
- Dont forget to filter the results of this subqueries from normal os behavior (like the “600 shadow.bak” — specific command)
- Look for dump file creation in the process commandline

We'll look for dumping tools activity, for example:

To hunt for user credential related file, we'll search all */etc/passwd*, */etc/shadow* files activity that is not created by system user control command, for example:

Now, for each query output , analyze the data and try to better understand the system behavior. Refactor the query using your conclusions about the behavior. In addition, filter out the "known good" activity that is specific to your network (e.g. Remove SAP utilities activity)

Beware — filtering out full commands will lower your ability to detect injections or binary poisoning.

Summary & Conclusions

We described a threat hunting process which includes four stages:

- Understanding the attack techniques you'd like to find
- Conducting research on how attackers implement these technique
- Searching the suspicious data in the organization to find anomalies which require further analysis
- Filtering out normal activities which look anomalous
- Repeating the above process while refining the queries until no anomalies are left or an attack was identified

To exemplify the implementation of this process, we used information from MITRE ATT&CK Matrix as well as academic papers which surveyed past attacks against Linux based systems to prioritize a hunting hypothesis. I focused on the "credential dumping" technique since it is common, easy to understand and does not require a lot of research to threat hunt most of its implementations.

The information provided throughout this blog includes queries against malicious data and examples of known good behavior which we can carefully whitelist. Each environment has its own unique anomalies. You need to carefully analyze all the anomalies you find and remove those which do not describe real threats to your network. It is a tedious and iterative process, but at the end you'll be able to come to a conclusion about your hunt hypothesis.

I hope that this post helped you become even more excited about dealing with Linux threat hunting. I think that if you take each tactic systematically, you will find it interesting. Look at this experience as a new opportunity to see the beauty of Linux Internals.

—

Thanks to Oren Biderman and Michael Gendelman for reviewing this post and providing useful suggestions.
