How Malicious Actors Abuse Native Linux Tools in Their Attacks

By Nitesh Surana, David Fiser, Alfredo Oliveira (words)

Published: 2022-09-08 · Archived: 2026-04-05 23:06:59 UTC

Cloud

Through our honeypots and telemetry, we were able to observe instances in which malicious actors abused native Linux tools to launch attacks on Linux environments. In this blog entry, we discuss how these utilities were used and provide recommendations on how to minimize their impact.

By: Nitesh Surana, David Fiser, Alfredo Oliveira Sep 08, 2022 Read time: 7 min (1973 words)

Introduction

Container adoption has become mainstream, with usage having risen across organizations globally. Based on a survey from CNCF, 93% of respondents are currently using or planning to use containers in their production. Container orchestration projects like Kubernetes and other tools available in the cloud and across the internet has led to a wave of transformations in how organizations operate, from monolithic architectures to the creation of distributed systems consisting of microservices.

However, these changes have also resulted in the expansion of the attack surface, particularly through security misconfigurations or vulnerabilities introduced in the deployments. Cloud securitynews article is further complicated by the fact that patch management can often be an enormous undertaking for organizations, which means that updates are not always implemented in a timely manner.

When it comes to public-facing web applications, we have been observing critical vulnerabilities arising from a wide range of sources, ranging from vulnerable open-source libraries (Log4Shell and Spring4Shell) to frameworks (Apache Struts and Drupal), and even applications such as Atlassian Confluence, Oracle WebLogic Server, and Apache HTTP Server. Once proof-of-concepts (POCs) for vulnerabilities are disclosed, attackers can exploit them to perform malicious tasks from mining cryptocurrency to, at times, deploying ransomwarenews-cybercrime-and-digital-threats.

From the perspective of a defender, the ideal outcome would be to prevent the attacker from even gaining an initial foothold. However, this isn't what always the case. If an attacker does manage to enter the system, it is the defender's job to make it more difficult for attackers to successfully pull off their routines by using defense-in-depth strategies.

Through our network of honeypots and verbose telemetry, we were able to observe some interesting characteristics for most of the successful exploit attempts, particularly, how attackers use native Linux tools in their routines.

Examining attacks using legitimate utilities and tools on Linux environments

An attack on a Linux-based system typically follows a standard exploitation chain. First, an attacker exploits a vulnerability (or a chain of vulnerabilities) to gain **initial access** into the environment (which we can now consider as compromised). From there, an attacker may take different paths to move further inside the compromised environment:

1. Enumerating the context of the current environment (Discovery)
2. Exfiltrating sensitive data from the environment (Exfiltration, Impact)
3. Performing a denial-of-service attack by removing the application (Impact)
4. Mining cryptocurrency by downloading miners (Impact)
5. Attempting other techniques (Privilege Escalation, Lateral Movement, Persistence, or Credential Access)

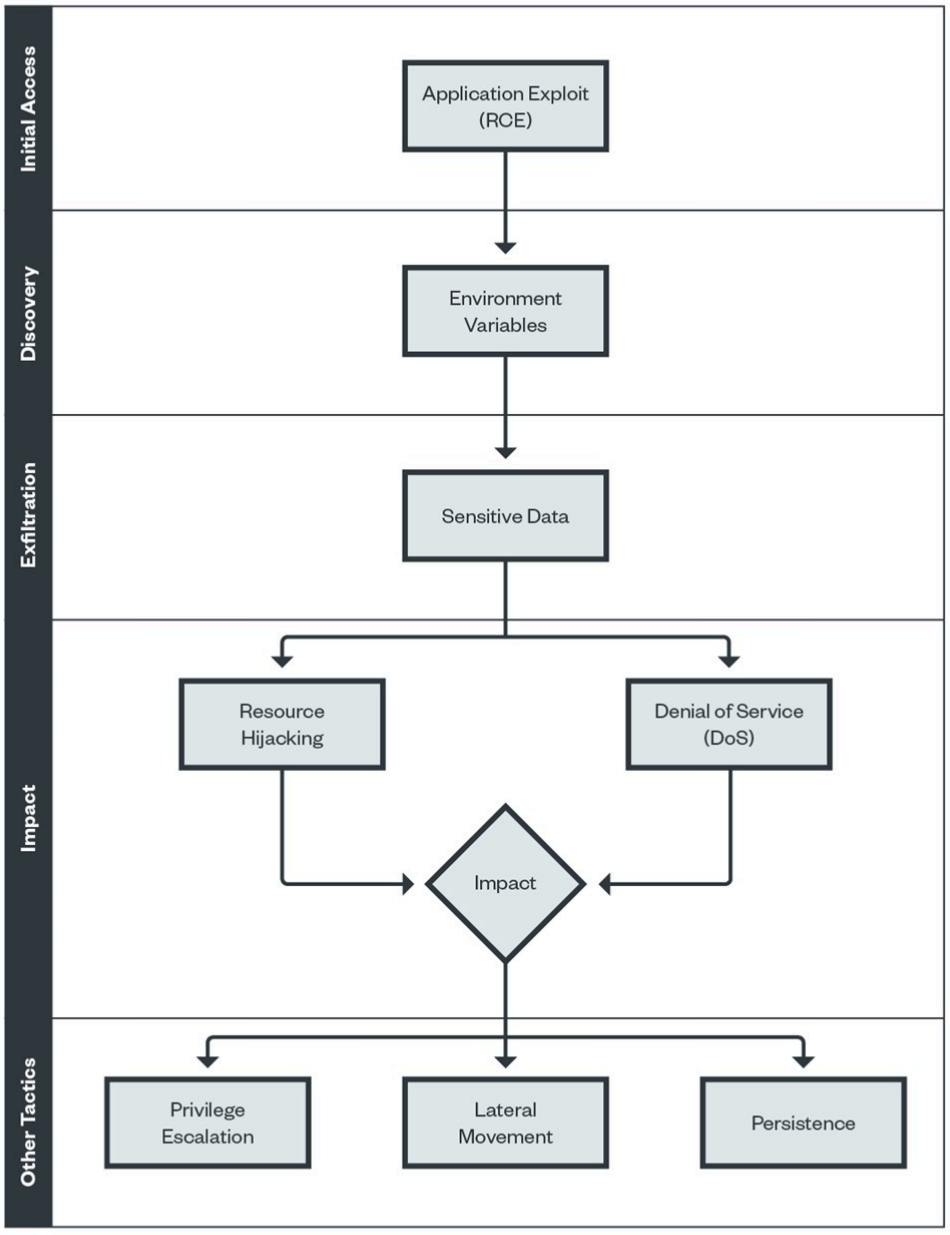

Figure 1. How an attacker can pivot further within a compromised environment

Based on real-world attacks and our honeypots, we observed that malicious actors use a variety of enabled tools that come bundled with Linux distributions, such as curl, wget, chmod, chattr, ssh, base64, chroot, crontab, ps, and pkill, that are abused by attackers for nefarious purposes.

We have seen malicious actors abusing these tools in the wild. The presence of these utilities, especially inside container environments, should be at least considered, since they provide additional avenues for attackers.

Let's examine some real-world attacks and instances of abuse that we observed via Trend Micro Cloud One™ and Vision One.

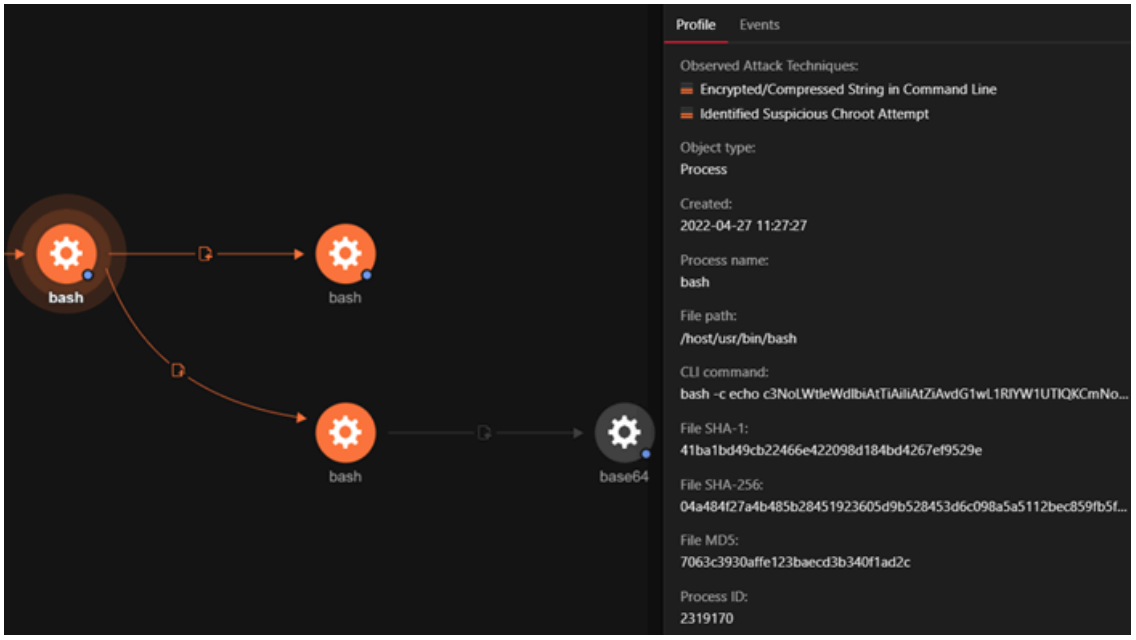

Figure 2. Using base64 to decode the payload for later execution

The base64 tool is a Linux utility that decodes strings encoded in base64 format. Attackers often obfuscate their payloads and commands using base64 encoding to evade detection (T1027), a technique we describe in detail in our previous article The Evolution of Malicious Shell Scripts.

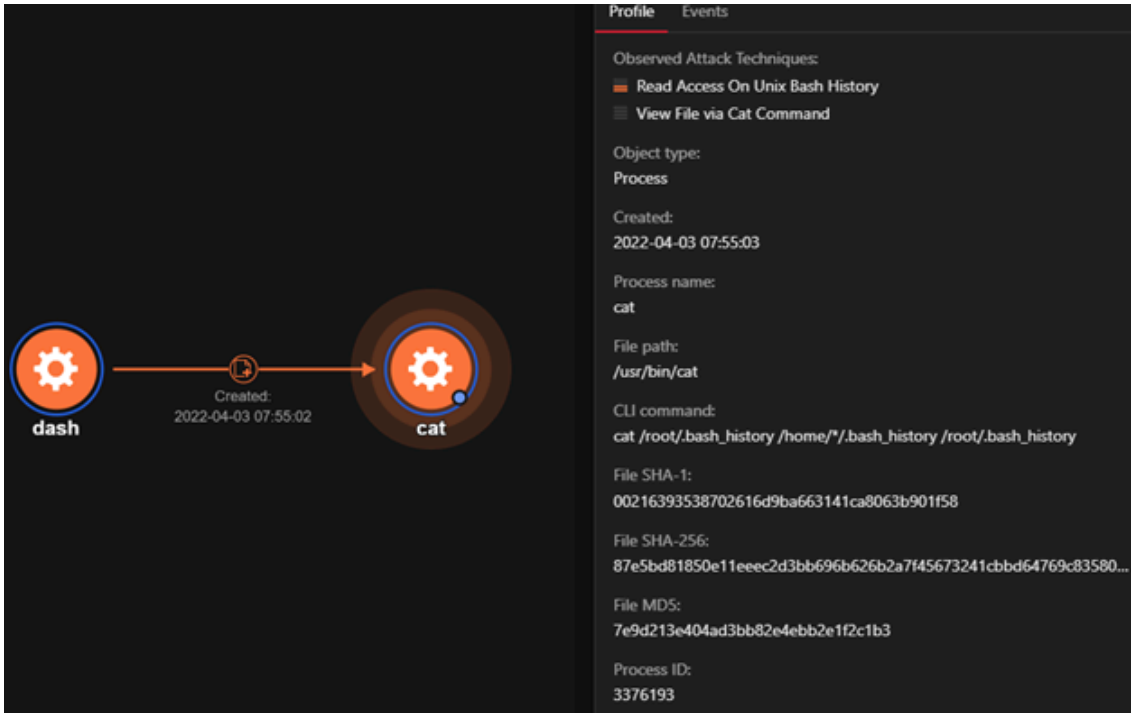

Figure 3. Using the "cat" process to view the .bash_history for all users

The .bash history file, which is stored in the user's home directory, logs the commands executed by users on their bash shell. Attackers have been known to extract information from these files to understand the context of the current environment, as we previously detailed in another article — [Misconfigured Docker Daemon API Ports Attacked for Kinsing Malware Campaignnews article](#).

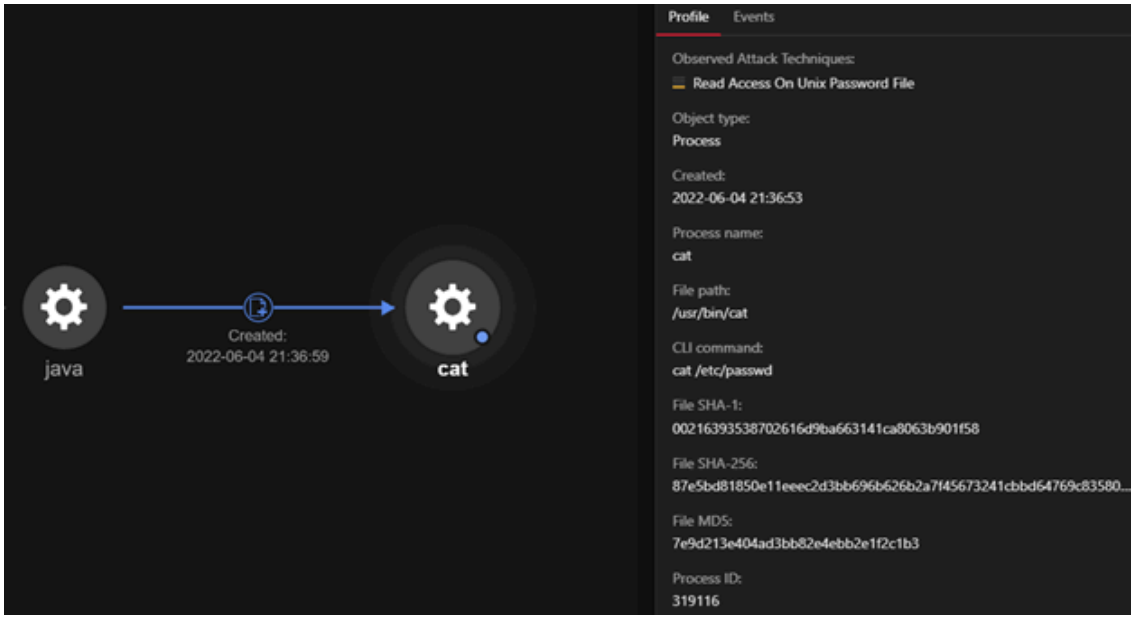

Figure 4. Using the "cat" process to view '/etc/passwd'

As a part of the enumeration step, the attacker accesses the */etc/passwd* file, which contains a list of the registered users within the environment and shows whether a given user has an associated shell with their login. This information helps the attacker understand the environment and pinpoint users of value. (T1003.008)

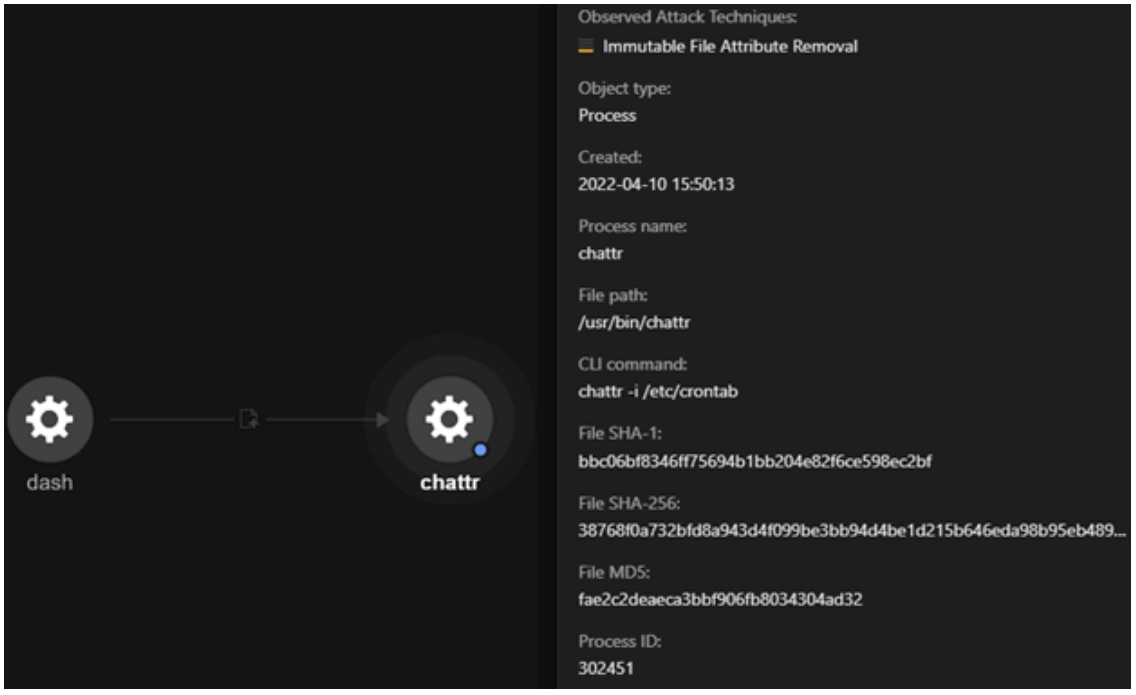

Figure 5. Using "chattr"` to modify the /etc/crontab file to be mutable

The chattr utility is used to alter file and folder attributes to control sudden operations like the deletion and modification of files. The example in Figure 4 shows that the attributes of the */etc/crontab* file has been altered, making the file unsecure. This utility has previously been observed to be abused by TeamTNT, as discussed in our white paper, Tracking the Activities of TeamTNT.

Figure 6. Using “chmod” to make a file executable

The chmod tool is used to change the file mode and granularize access per user or group. It’s required to execute newly downloaded executables, and, in this case, we see the *agettyd* file at the path */tmp* being set with the executable bit.

Figure 7. Using “crontab” to remove all existing cron jobs

A cron job is a utility used to schedule tasks (or jobs). Attackers have been known to abuse cron jobs and modify the ‘crontab’ to perform execution, persistence, and, at times, privilege escalation techniques (T1053.003). The example in Figure 7 shows the removal of existing cron jobs. This is a common occurrence where cryptocurrency miners compete against each other by removing traces of other miners to hijack the maximum amount of

resources possible. Our blog entry, War of Linux Cryptocurrency Miners: A Battle for Resources, discusses these activities in-depth.

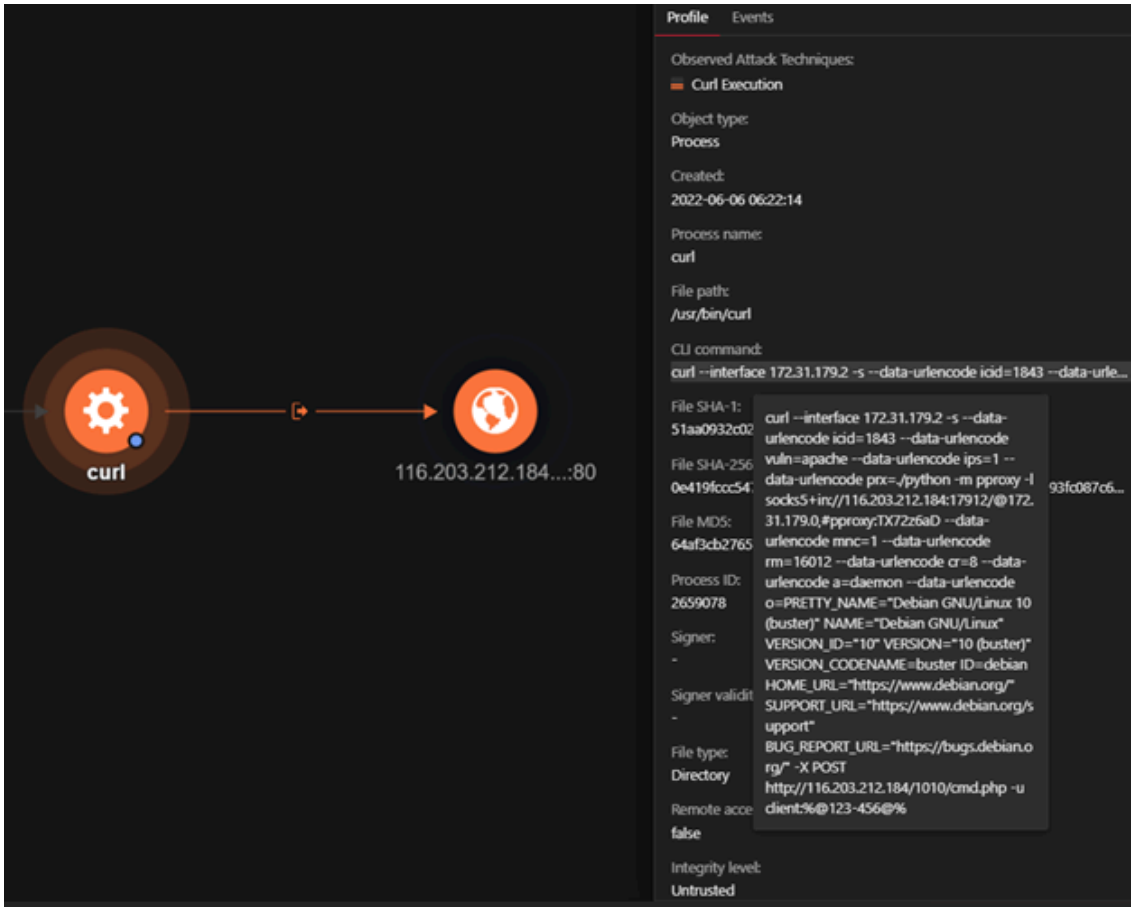

Figure 8. Using “curl” to exfiltrate system information to the attacker

The curl, or cURL, utility is used to transfer data across different protocols, such as HTTP, HTTPS, and File Transfer Protocol (FTP). The example in Figure 8 shows that system information such as the OS version and release version is sent as a POST request to the attacker’s infrastructure.

Figure 9. Using “curl” to download xmrig binaries from GitHub

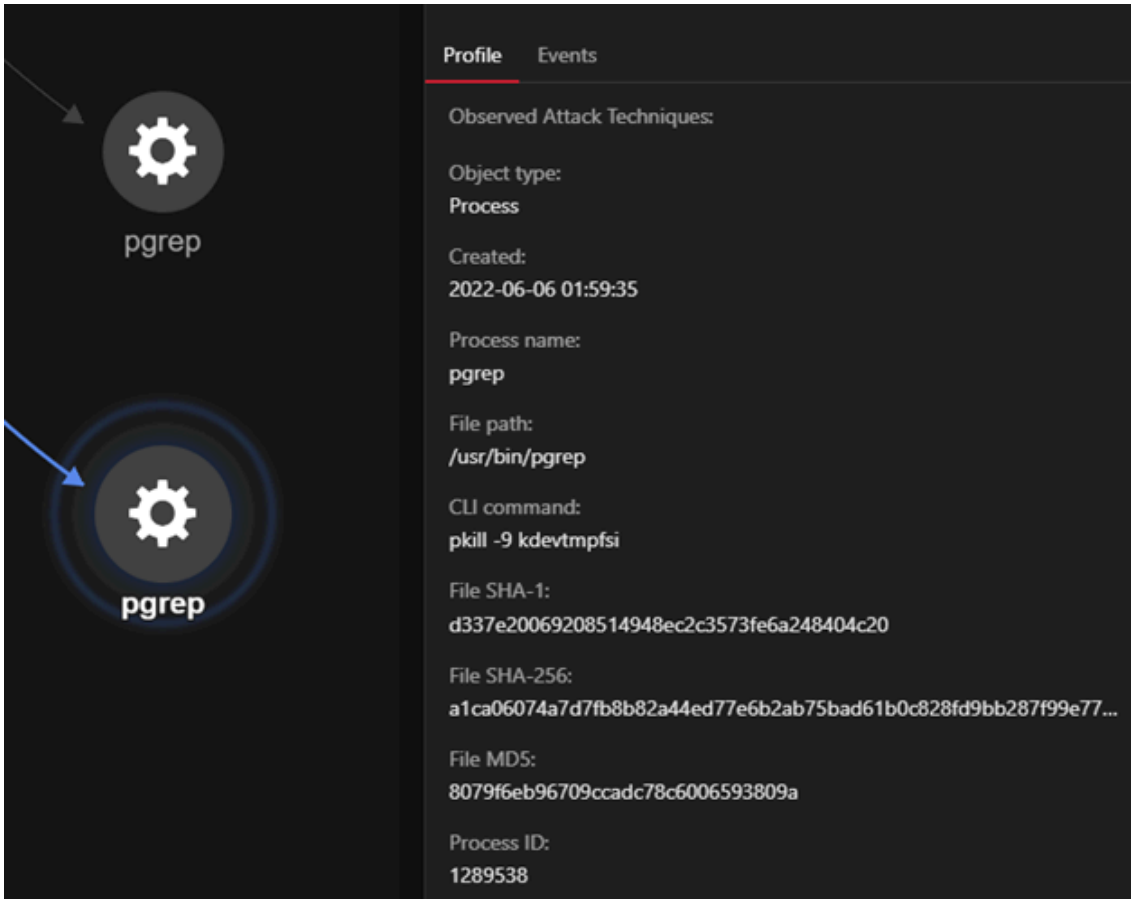

Figure 10. Using “pkill” to kill competing processes/coinminers

The kill suite utility is used to send signals to processes and, as illustrated in the example in Figure 10, it sends the SIGKILL signal to the process named “kdevtmpfsi”. We have been observing cryptocurrency miners named *kdevtmpfsi* as early as 2020. Our blog entry, Analysis of Kinsing Malware's Use of Rootkit, shows another example of a competing miner being terminated.

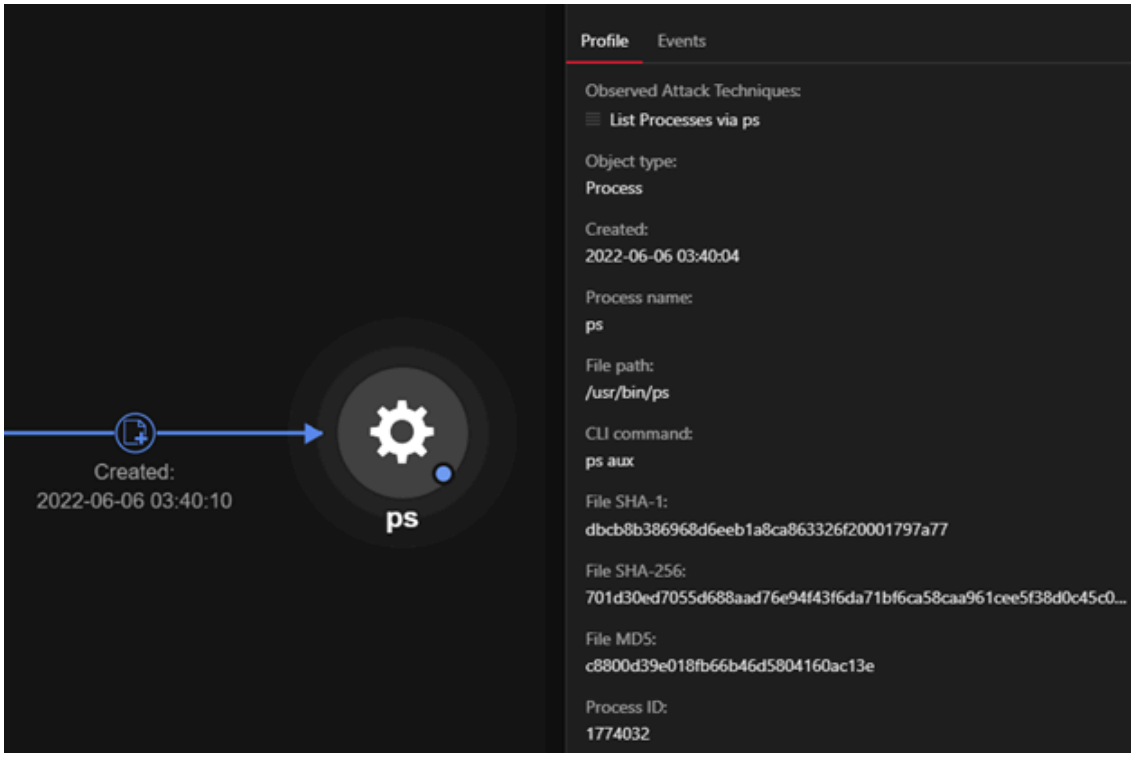

Figure 11. Using “ps” to view running processes

The ps utility is used to view the status of a process. Figure 11 shows the *ps aux* command fetching verbose information about the processes, such as currently running processes, process IDs, and process privileges, on the system. This information can aid attackers in performing discovery-related techniques (T1057 – Process Discovery) and gaining information about the environment they’re in.

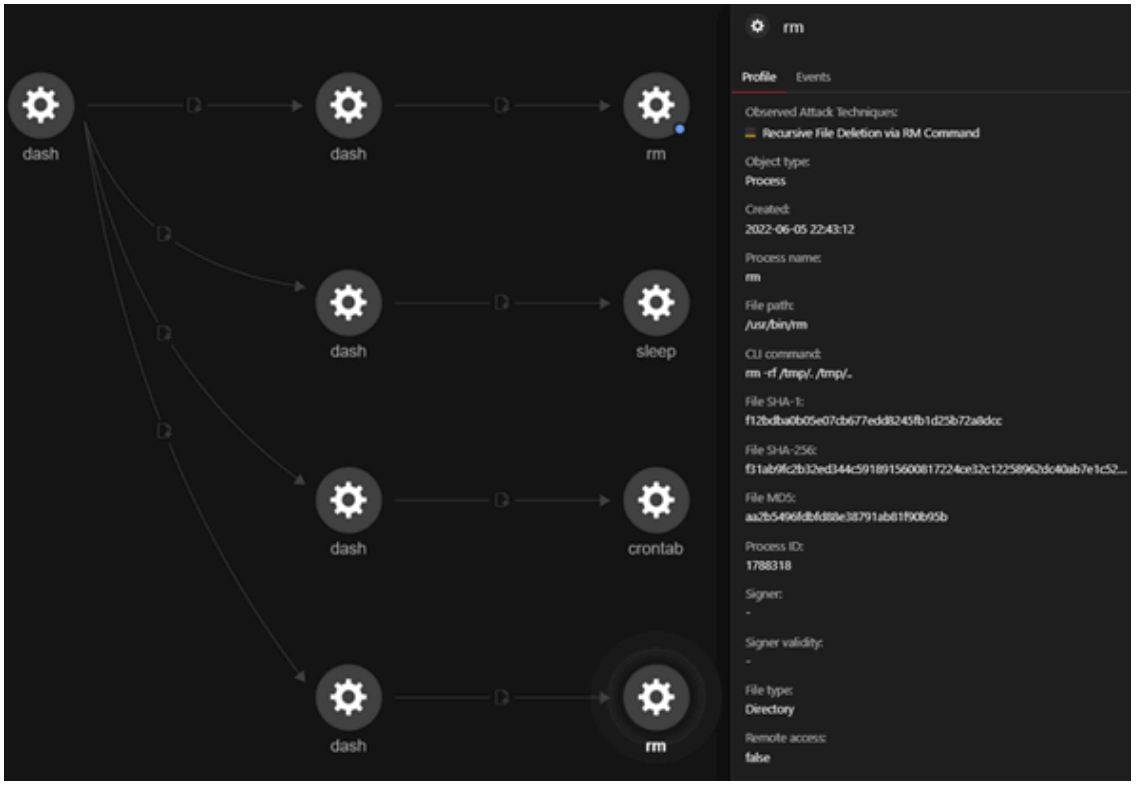

Figure 12. Using “rm” to remove hidden files from /tmp directory

In Figure 11, we see the rm tool being used to delete the hidden files and folders under the */tmp* directory. Attackers can create hidden directories to evade detection by adding “.” before the file or folder name (Hide Artifacts: Hidden Files and Directories - T1564.001).

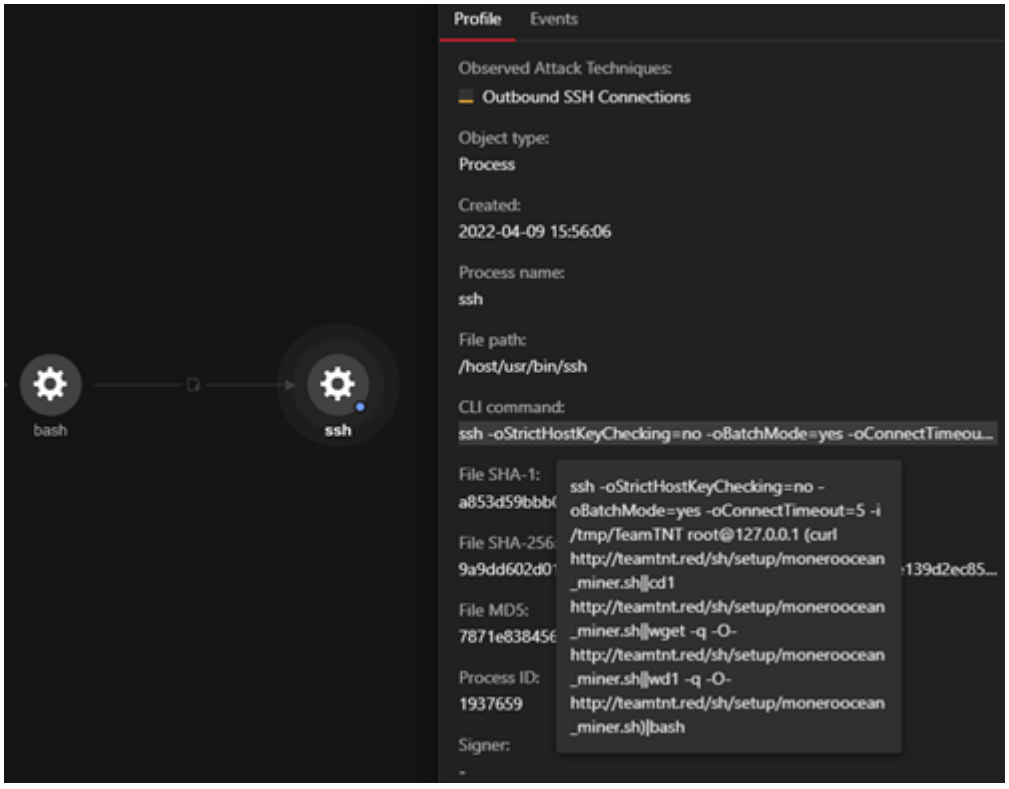

Figure 13. Using “ssh” to infect underlying host with XMR miners

The ssh utility is the remote client used for accessing systems over Secure Shell (SSH) in a worm-like fashion. In Figure 13, the attacker tries to download the Monero miner (using wget/curl) and infect the remote machine in which the SSH is being attempted (127.0.0.1). Once attackers mount the underlying host’s file system due to unsecure configuration (for example, privileged containers) of containers, they create new pairs of SSH keys, use it to establish an “ssh” session, and infect the underlying host with cryptocurrency miners.

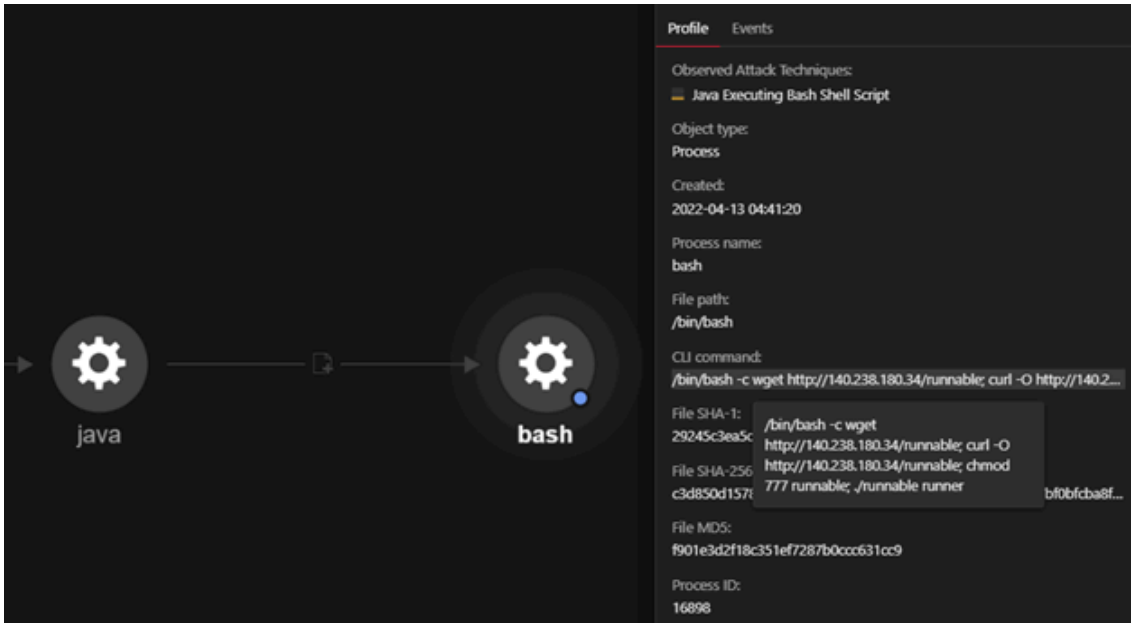

Figure 14. Using “wget”, “curl”, “chmod” to download and execute Mirai malware

In this example, we see the combined use of different Linux utilities wherein the binary is downloaded, permissions are modified, and then later executed. The executable named “runnable” is a Mirai sample delivered after the exploitation of the Log4shell vulnerability tracked under CVE-2021-44228.

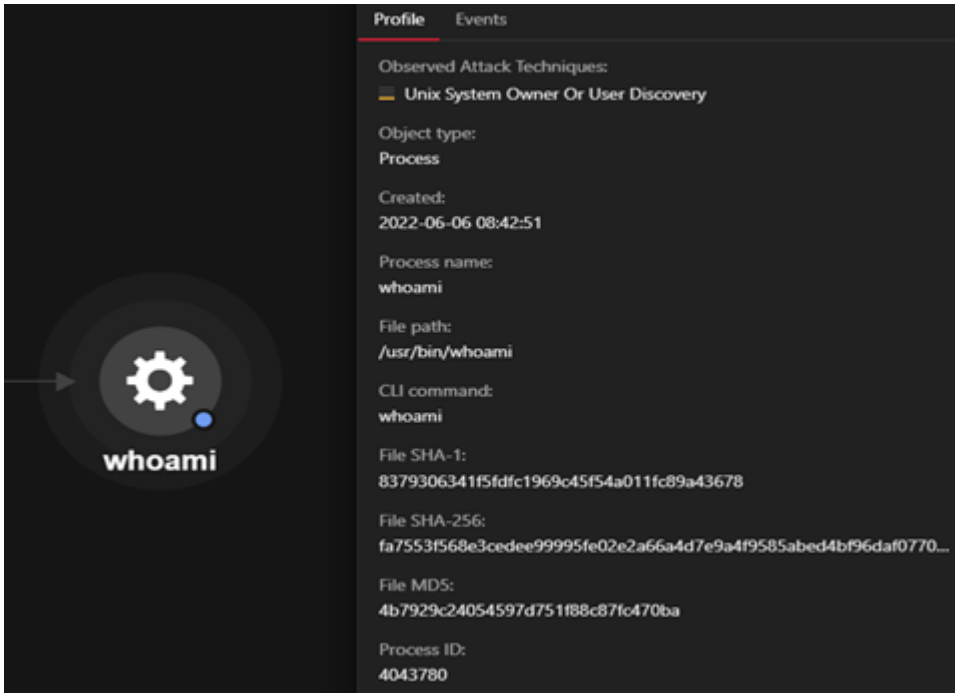

Figure 15. Using “whoami” to view current user context

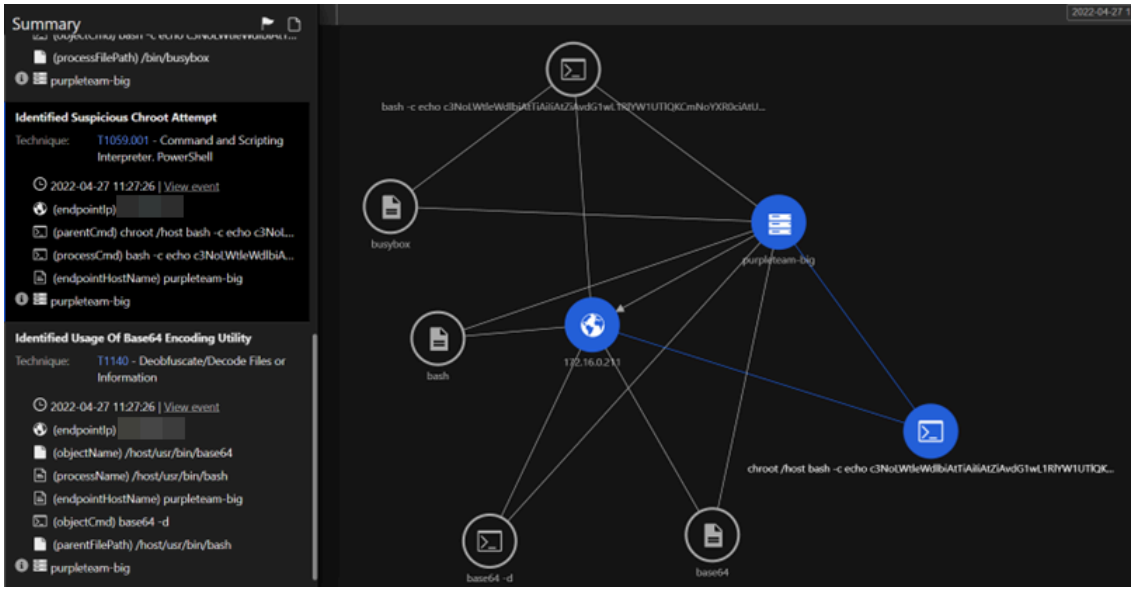

Figure 16. Workbench showing an attacker using “chroot” and “base64”

Using the Vision One workbench, we see the chroot and base64 utilities being used by an attacker. Note that chroot is used to change the root to the directory supplied (in this case, */host*), where the underlying host’s file system is mounted within the container. We explore the weakness posed by this function when granted to a container in our article, Why A Privileged Container in Docker is a Bad Idea.

Best practices for protecting Linux systems from utility abuse

By observing the techniques discussed in the previous section, we see that the attackers can use a set of tools that come bundled with a full-scale operating system. As defenders, it would be safer to have container images that contain only the tools we require, and remove the tools that aren’t needed.

This approach to security can help mitigate risk to a great extent, even against critical vulnerabilities such as Log4Shell. Reducing the number of tools needed for applications to run also reduces the attack surface introduced by the dependency vulnerabilities in open-source libraries and tools. Here enters the concept of distroless images, which are described as images that contain only the application and its runtime dependencies, doing away with the programs you would expect to find in a typical Linux distribution such as package managers and shells.

From a defender's perspective, the focus should be on disabling, or rather, disarming the attacker via defense-in-depth strategies. While making changes to the system to minimize or even prevent abuse would help, a multilayered approach that leverages multiple security measures would provide the strongest level of security, ideally by combining best practices with effective defense technologies.

For non-containerized environments, Cloud One Workload Security provides the Application Control module, which monitors software changes and allows or blocks them based on the set configuration. It creates a baseline of the existing applications and applies the rules to the new applications that are downloaded and installed. It works based on the SHA256 hash for a binary.

It provides options for users to do the following:

1. Block unrecognized software until explicitly allowed
2. Allow unrecognized software until explicitly blocked

We download a pre-compiled binary of the nmap network enumeration tool from GitHub using wget on an Ubuntu 20.04 long-term support (LTS) server. The server was then configured with the Cloud One Workload Security agent running with the Application Control module set to ‘Block’ mode for unrecognized software. As shown in the Figure 17, the execution was prevented by Application Control.

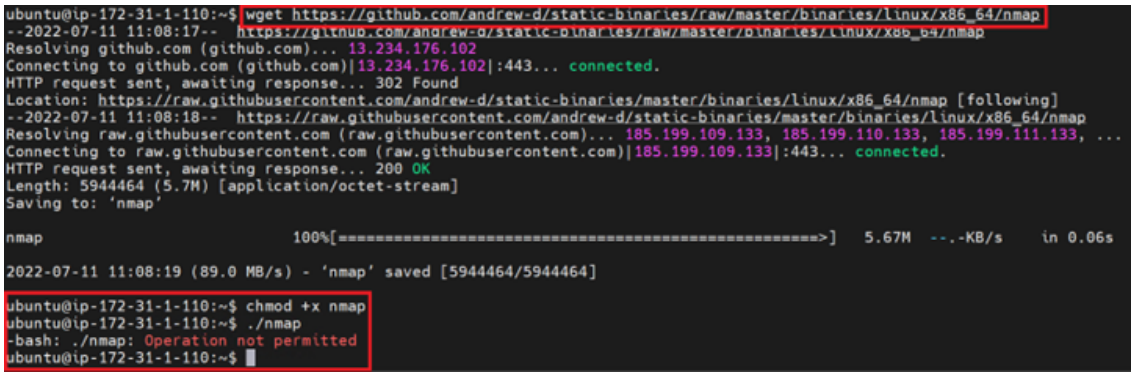

Figure 17. Prevention the execution of the “nmap” binary using the Application Control module from Cloud One Workload Security

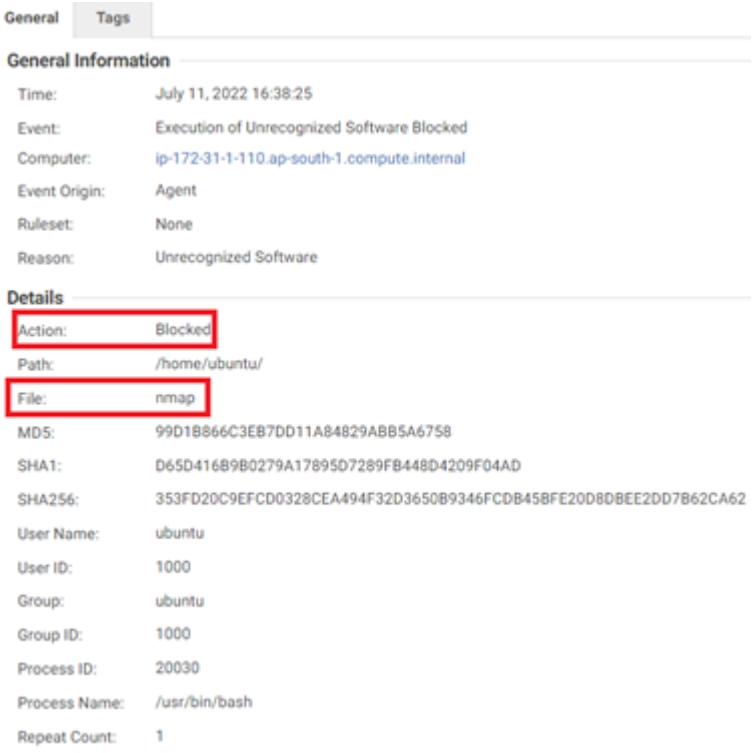

Figure 18. The corresponding event on Cloud One Workload Security, where we see that the “nmap” binary was blocked from being executed

Conclusion

With attackers making use of legitimate tools and utilities that are built into the operating system, defenders will need to prioritize how they can set up controls during the different phases of an attack. Minimizing the attack surface by using distroless images in containers and applying preventive controls like Cloud One Workload Security's Application Control go a long way in slowing down attackers targeting cloud environments. In cases where organizations cannot go with a distroless implementation, "slimmed-down" versions of the same images can also be used to reduce the attack surface and strengthen the security of cloud deployments.

Tags
