ATT&CK T1501: Understanding Systemd Service Persistence

By Dominic Heidt

Archived: 2026-04-05 15:18:23 UTC

The Red Canary intel team has spent a lot of time researching the endpoint threat landscape and related detection possibilities for Linux systems. While examining one particularly interesting persistence strategy, we were somewhat surprised to learn that the technique didn't already exist in the ATT&CK™ Framework. While malicious use of systemd services isn't necessarily new to Linux systems, there is little public research or documentation about how adversaries leverage it for persistence.

What are systemd services?

Put succinctly (and derived largely from the systemd manual page), systemd is a system and service manager for Linux distributions. From the Windows perspective, this process fulfills the duties of wininit.exe and services.exe combined. At the risk of simplifying the functionality of systemd, it initializes a Linux system and starts relevant services that are defined in service unit files. These unit files may define the execution of binaries like httpd or the execution of scripts with command lines similar to `bash -c scriptedService.sh`.

Older Linux distributions have used different initialization systems in the past, usually SysVInit. This system leveraged shell scripts that would execute the relevant components needed to provide a service. For several years now, modern Linux systems have adopted systemd to define services, deprecating the init scripts of yesteryear. Many people didn't greet this decision with open arms, but it seems to be here to stay.

What makes up a systemd service?

A service unit file defines a structure similar to this:

```
[Unit]
Description=Atomic Red Team Service

[Service]
Type=simple
ExecStart=/bin/touch /tmp/art-systemd-execstart-marker

[Install]
WantedBy=default.target
```

Let's break this down in pieces.

Unit

The description provides an understandable name for the defined service.

Service

To specify service execution information, you can use several directives:

- ExecStart: commands executed when the specified service starts
- ExecStartPre: commands executed before ExecStart
- ExecStartPost: commands executed after ExecStart
- ExecStop: commands executed when the specified service stops
- ExecStopPre: commands executed before ExecStop
- ExecStopPost: commands executed after ExecStop
- ExecReload: commands to trigger configuration reload of a service

Install

Finally, the unit file specifies a WantedBy directive. If you're a veteran of SysVinit systems, you'll remember runlevels and how each init script existed under a folder numbered to coincide with the relevant runlevel. This let systems determine which services would execute based on how the host booted or changed modes. In the world of systemd, `targets` replace runlevels. Each target roughly corresponds to a runlevel from SysVinit, and, in this case, our target is the default target. It specifies that this service should execute for whatever target is currently set, ensuring it will execute at the next boot.

Of course it's more complicated than that…

There are many places where administrators and adversaries may stash systemd service files. For systemd services executing at the system-level, you can place service files in these folders:

```
/lib/systemd/system
/usr/lib/systemd/system
/etc/systemd/system
/run/systemd/system
```

For systemd services executing in a single user's context, you can place service files in these folders:

```
/etc/systemd/user
~/.config/systemd/user
~/.local/share/systemd/user
/run/systemd/user
/usr/lib/systemd/user
```

In many cases, software installations will place symbolic links (like Windows shortcuts) in these folders, allowing them to place their own service file anywhere on the filesystem while allowing systemd to discover it.

What does normal look like?

When executing, commands specified by a systemd service should always have a parent process of `systemd`. In most cases, the parent `systemd` process should have a process ID of 1, but there are edge-cases when it may not. By default, system-level services will execute as the `root` user. This would be equivalent to the Windows Administrator account. If desired, you may specify an additional configuration directive, `User=`, to allow system-level services to impersonate another user. User-level services that are not in the security context of `root` cannot impersonate another user, however.

Finally, when managing systemd services, you must use the `systemctl` utility to perform some tasks. After something writes a unit file to disk, `systemctl enable <name>` allows systemd to load it as a service at boot, and `systemctl start <name>` executes the commands specified by the service file on demand.

Adversary use of systemd services

Systemd services are an awesome artifact for adversaries to leverage for persistence. With the proper privileges, adversaries can create a malicious service (just like in Windows) that will allow them to persist between reboots, potentially as root.

The most notable reported use of systemd services for persistence involved the compromise of orphaned software packages no longer maintained by the "acroread" software package for Arch Linux in 2018. Unsuspecting users downloaded acroread, allowing the malicious script within to create persistence via a systemd service. Once discovered by users, maintainers of the Arch User Repository reverted those packages to benign versions. Here, the adversary was stopped before they could do much damage, in large part because one payload didn't execute as designed.

At least one major threat actor, Rocke, has used this persistence technique in opportunistic attacks on Linux servers. When they evolved their payloads away from simple scripts to leverage Golang binaries, Rocke also included a Systemd service to ensure their payload would execute on boot.

The Metasploit Framework has already incorporated this persistence technique into its functionality. By using the service persistence exploit module, adversaries may create service files on disk to execute as either a system-level service or a user-level service. This functionality has been part of the tool since around December 2018. To leverage this, adversaries can specify their own payload for execution, and Metasploit can handle the details of writing the payload to disk, writing the service file to disk and scheduling it for execution at the next boot.

If you want to test this out in your environment without setting up Metasploit, you can do so with Atomic Red Team. We have an Atomic Test that guides you through creating a simple systemd service and scheduling it for execution at boot.

Hunting for malicious systemd services

As mentioned before, systemd service files can live in multiple places on disk. When hunting with EDR technology, you can check out suspicious file modifications of those folders. By default, the processes that typically write these files are Python because of package management via Yum on CentOS/RHEL, dpkg because of package management via apt on Debian derivatives, and systemctl itself to enable and disable services. Bash

shell scripts and processes other than those mentioned touch these files less often, so they'll be more suspect while hunting.

If you don't have EDR tools and you want to hunt for malicious persistence via the Linux command line, you can do so with these commands:

```
find / -path "*/systemd/system/*.service" -exec grep -H -E "ExecStart|ExecStop|ExecReload" {} \;
2>/dev/null
```

```
find / -path "*/systemd/user/*.service" -exec grep -H -E "ExecStart|ExecStop|ExecReload" {} \;
2>/dev/null
```

These commands will enumerate all the commands specified by system and user systemd services on a host. If you're daring, you might combine these commands with `ssh` commands to execute them on remote hosts, saving the output to files for hunting and processing later. Pay particular attention to service files corresponding to services you don't recognize and execution commands that contain `bash`, `python`, or `sh`.

When hunting on EDR platforms, be aware of process ancestry involving systemd. On modern Linux systems, the systemd process should start all services, operating as the root of all process trees. Most of the time, systemd will spawn daemon processes, such as httpd or mysqld. When it spawns shell processes such as bash, this definitively shows that bash was specified as a service execution. Legitimate user instances of bash shells that occur after logon do not spawn as direct children of systemd, and they exist further down the ancestry chain. While a systemd -> bash ancestry is not certainly evil, it's a good place to start when hunting in your own environment.

Conclusion

By understanding a bit of Linux system internals, we can hunt down adversaries and remove their footholds. Malicious persistence takes many forms and, like with Windows, knowing the capabilities of the operating system makes all the difference.
