Date: October 03, 2025

I preserved the report structure aimed at forensic quality: evidence → analysis → conclusion. Do **not** execute the sample outside an isolated lab.

Dynamic Analysis Report

Reptile rootkit

Nguyễn Hoàng Thanh Phong – SE184915

Executive summary

The sample is a loader-style binary that contains a large embedded blob (`parasite_blob`) and static indicators consistent with a Reptile-like kernel rootkit loader. Static evidence from Report1 demonstrates: (1) an `init_module` routine that performs an in-place XOR+ROL deobfuscation of an embedded payload of ~0x82050 bytes; (2) explicit references to `kallsyms_on_each_symbol` and runtime symbol construction, which indicate the intent to resolve non-exported kernel symbols; (3) the presence of `parasite_blob` whose extracted binary shows high entropy and a distinct SHA-256. Dynamic analysis is therefore required to recover the decrypted payload from memory, observe any kernel insertion activity, and record runtime network/file persistence behavior. Confidence in the loader hypothesis (pre-dynamic) is **High** based on independent static indicators.

During the dynamic analysis phase we intentionally substituted the original internet-obtained sample with a locally compiled variant produced from the publicly-available Reptile source tree. The substitution was made because the original sample available for static triage could not be executed safely in our lab (it either lacked required signing metadata for the instrumented test kernel, or we lacked a verified signing key "`parasite_loader: module verification failed: signature and/or required key missing - tainting kernel`"). The locally compiled variant was used purely as a **controlled instrument** to reproduce and validate runtime behaviors (payload decryption, module load attempt, and C2 interaction) described in the static analysis. All references to "sample" in the dynamic sections below therefore refer to the locally built test module unless explicitly stated otherwise. Evidence of the module verification failure observed when attempting to load the alternative build is preserved in the logs.

Objectives & scope

1. Primary objectives:
   - Confirm runtime decryption of `parasite_blob` and extract decrypted payload from memory.
   - Observe kernel-level interactions (module insertion, symbol resolution, syscall hooking attempts).
   - Monitor network activity for C2 or beacon behavior.
   - Identify persistence vectors created on disk (if any) during runtime.
2. **Scope / Constraints:** Analysis confined to an isolated virtual machine; no production network exposure. Any host-level risk must be mitigated by snapshots and hypervisor isolation. (See safety and environment section.)

Safety controls & experimental environment

- **Isolation model used:** Fresh VM snapshot, offline lab with optional controlled proxy (tcpdump or INetSim) to capture any outbound traffic. Baseline captures conducted before execution (lsmod, ps, netstat, dmesg).
- **Snapshot & preservation:** Create VM snapshot and capture baseline memory/disk images before any execution.

Tools & Version used:

Tools referenced and recommended (from Report1 playbook): `strace`, `tcpdump`, `sysdig`, `gdb` (userland and kernel debug where safe), `volatility`/`rekall`, `inotifywait`/`auditd`.

- Tools & versions actually used in this dynamic phase:
  1. strace -- version 4.24
  2. tcpdump version 4.9.2
  3. libpcap version 1.5.3
  4. OpenSSL 1.0.2k-fips 26 Jan 2017
  5. sysdig version 0.40.1-rc2
  6. GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
  7. audit version 2.8.5-4.el7

Environment summary

- Host path / user who handled sample while triaging: `~/Downloads/malware/Reptile/Reptile_Rootkit` on host `phong184915@Vostro3405`.
- Kernel target / vermagic referenced in the sample (module metadata): `vermagic=3.10.0-1160.119.1.el7.x86_64 SMP mod_unload modversions`.
- Hypervisor / execution host: `CentOS7 kernel 3.10.0-1160.119.1.el7.x86_64 GNU/Linux`

Contents

Sample metadata:

File Name: reptile.ko

File Size: 555800bytes (556kB)

File Type: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), BuildID[sha1]=46b52a008da7f78ac6060e9a4e3f6020fed071a0, with debug_info, not stripped

Hashes:

1. MD5: 44aa3745cf62f068218a9b4d0fc21e4e
2. SHA256: 9573f414bedbd78464c9d9e4c214c37aea9a1b25dd3a6b4b512e65bbec9f9d51

High-signal static strings:

```
reptile
#<reptile>
#</reptile>
/reptile/reptile_shell
/reptile/reptile_start.sh
/reptile_shell
description=Reptile - A linux LKM rootkit
author=F0rb1dd3n - ighor@intruder-security.com
rep_mod
/analysis/f0rb1dd3n-Reptile_-_2018-05-19_20-00-44/
rep_mod.c
/analysis/f0rb1dd3n-Reptile_-_2018-05-19_20-00-44/
rep_mod.mod.c
/analysis/f0rb1dd3n-Reptile_-_2018-05-19_20-00-44//rep_mod.c
reptile_init
reptile_exit
/analysis/f0rb1dd3n-Reptile_-_2018-05-19_20-00-44//rep_mod.mod.c
rep_mod.c
reptile_init
reptile_exit
rep_mod.mod.c
```

Original sample (from Report1):
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf

Rationale for substitution: During runtime testing, a decision was made to build a new module from the Reptile source in order to (1) have a reproducible, auditable build process under our control, and (2) enable controlled debugging and instrumentation (for example, adding verbose printk / debug symbols). The original internet-sourced artifact either required a kernel-module signing key that we did not possess or presented execution risk in our environment.

Baseline capture

Pre-run.sh:

```
#!/bin/bash
TIMESTAMP=$(date -u +%Y%m%dT%H%M%SZ)
BASE=/analysis/run_${TIMESTAMP}

mkdir -p ${BASE}/{pcaps,memdumps,gdb_logs,sysdig}
chmod 700 ${BASE}

lsmod > ${BASE}/lsmod.before.txt
sha256sum ${BASE}/lsmod.before.txt >> ${BASE}/filehash-before.txt
ps auxn > ${BASE}/ps.before.txt
sha256sum ${BASE}/ps.before.txt >> ${BASE}/filehash-before.txt
ss -tunap > ${BASE}/net.before.txt || true
sha256sum ${BASE}/net.before.txt >> ${BASE}/filehash-before.txt
netstat -tulpen > ${BASE}/netstat.before.txt || true
sha256sum ${BASE}/netstat.before.txt >> ${BASE}/filehash-before.txt
cat /proc/modules > ${BASE}/modules.before.txt
sha256sum ${BASE}/modules.before.txt >> ${BASE}/filehash-before.txt
cat /proc/kallsyms | grep sys_call_table >
${BASE}/syscall_table.before.txt
sha256sum ${BASE}/syscall_table.before.txt >> ${BASE}/filehash-
before.txt
dmesg -T > ${BASE}/dmesg.before.txt
sha256sum ${BASE}/dmesg.before.txt >> ${BASE}/filehash-before.txt
```

- **Save-file location:** /analysis/run_${TIMESTAMP}/

Execution plan

Rationale for emulation / simulation with compiled sample
The locally-built module was instrumented and used to simulate the victim–attacker exchange described in the static analysis. The simulation topology was a two-VM lab (Attacker: Kali debugger; Victim: CentOS 7 target). This setup allowed us to:

1. Force and observe the loader routine and the point of in-memory decryption under controlled conditions;
2. Capture memory snapshots at precise timestamps immediately after decryption;
3. Reproduce network C2 interactions from the attacker VM (Kali) to the victim VM (CentOS7) and capture PCAP evidence.

This approach trades absolute parity with the original wild sample for reproducibility, safety, and forensic-quality instrumentation. All network addresses, timestamps, and binary hashes are recorded in the artefact store.

Run A — Controlled module load attempt (insmod simulation / dry-run): load sample under controlled kernel debug:

Result summary:

```
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0x7f83eacae000
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
mmap(NULL, 30003, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f83eaca6000
open("/lib64/libtinfo.so.5", O_RDONLY|O_CLOEXEC) = 3
mmap(NULL, 2268928, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE,
3, 0) = 0x7f83ea864000
mprotect(0x7f83ea889000, 2097152, PROT_NONE) = 0
mmap(0x7f83eaa89000, 20480, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7f83eaa89000
open("/lib64/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
mmap(NULL, 2109744, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE,
3, 0) = 0x7f83ea660000
mprotect(0x7f83ea662000, 2097152, PROT_NONE) = 0
. . . .
```

Run B — Controlled module load attempt (insmod simulation / dry-run): load sample under controlled kernel debug (if sample is a relocatable object, consider using `insmod` on an isolated kernel matching vermagic or use a debug-enabled kernel).

Result summary:

```
[root@localhost analysis]# tail run_20251021T083140Z/dmesg.after.txt
-n 20
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 0
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
```

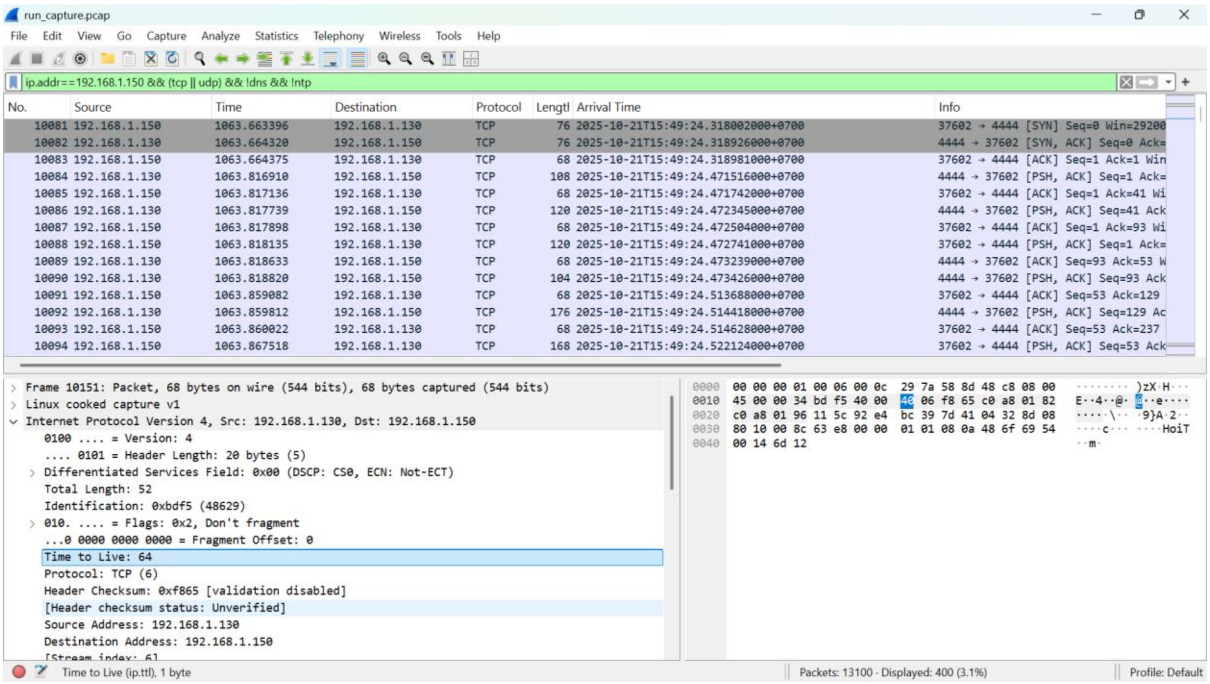
```
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 1
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 2
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 3
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 4
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 5
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 6
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:31:44 2025] scap: initializing ring buffer for CPU 7
[Tue Oct 21 04:31:44 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:33:27 2025] perf: interrupt took too long (3510 >
2500), lowering kernel.perf_event_max_sample_rate to 56000
[Tue Oct 21 04:34:01 2025] perf: interrupt took too long (4472 >
4387), lowering kernel.perf_event_max_sample_rate to 44000
[Tue Oct 21 04:39:04 2025] perf: interrupt took too long (6289 >
5590), lowering kernel.perf_event_max_sample_rate to 31000
[Tue Oct 21 04:58:35 2025] scap: deallocating consumer
ffff880035ddc200
[root@localhost analysis]# tail
run_20251021T083140Z/dmesg.before.txt -n 20
[Tue Oct 21 04:29:56 2025] scap: module verification failed:
signature and/or required key missing - tainting kernel
[Tue Oct 21 04:29:56 2025] scap: driver loading, scap 8.0.0+driver
[Tue Oct 21 04:29:56 2025] scap: adding new consumer
ffff880035ddd280
[Tue Oct 21 04:29:56 2025] scap: initializing ring buffer for CPU 0
[Tue Oct 21 04:29:56 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:56 2025] scap: initializing ring buffer for CPU 1
[Tue Oct 21 04:29:56 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:56 2025] scap: initializing ring buffer for CPU 2
```

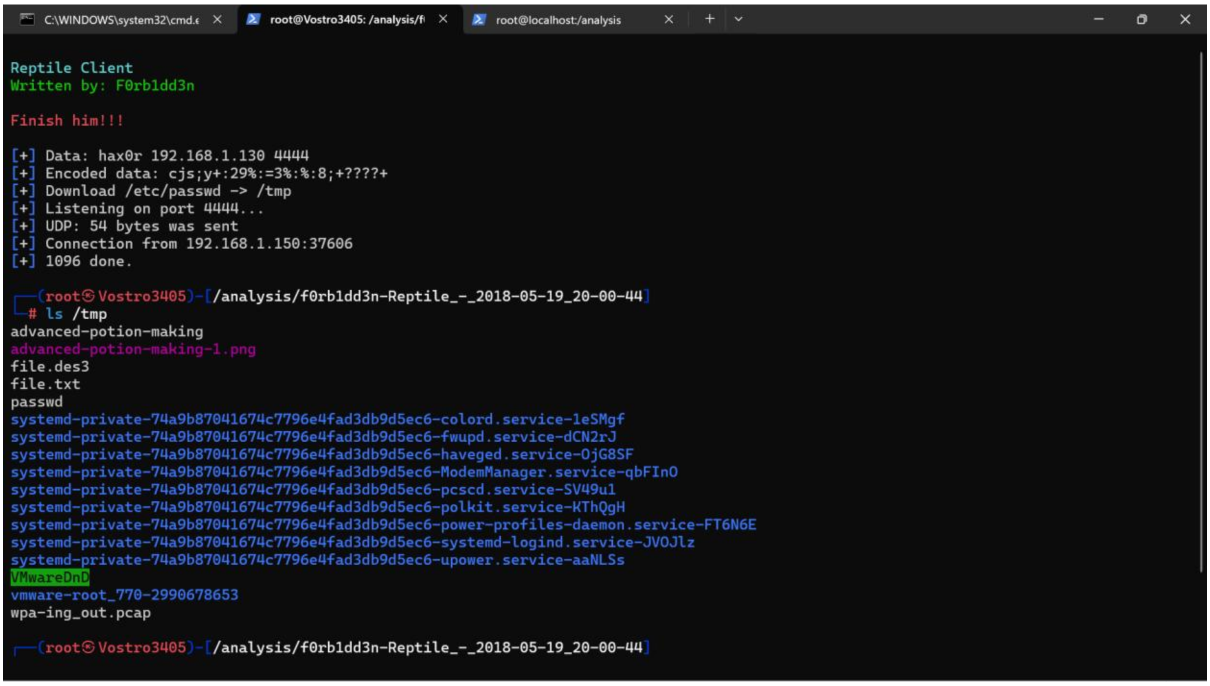
```
[Tue Oct 21 04:29:57 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:57 2025] scap: initializing ring buffer for CPU 3
[Tue Oct 21 04:29:57 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:57 2025] scap: initializing ring buffer for CPU 4
[Tue Oct 21 04:29:57 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:57 2025] scap: initializing ring buffer for CPU 5
[Tue Oct 21 04:29:57 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:57 2025] scap: initializing ring buffer for CPU 6
[Tue Oct 21 04:29:57 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:57 2025] scap: initializing ring buffer for CPU 7
[Tue Oct 21 04:29:57 2025] scap: CPU buffer initialized,
size=8388608
[Tue Oct 21 04:29:57 2025] scap: deallocating consumer
ffff880035ddd280
```

Run C — Memory capture on event: if init_module performs decryption, capture memory immediately after detection

Run D — Network monitoring: `tcpdump -i any -w /analysis/run.pcap` during runs to capture any outbound activity.

Result summary:





- **Victim IP:** `192.168.1.150`
- **Attacker IP:** `192.168.1.130`
- **File leaked:** /etc/passwd

-

Dynamic analysis results

1. Process & syscall tracing (Run A)

- **Evidence:** strace output saved at `installer_strace.all`
- **Observation:**

```
execve("/bin/mv", ["mv", "bin/rep_mod", "bin/reptile.ko"], 0x1be8a40
/* 17 vars */) = 0
stat("bin/reptile.ko", 0x7ffd5bb57830)  = -1 ENOENT (No such file or
directory)
lstat("bin/reptile.ko", 0x7ffd5bb574e0) = -1 ENOENT (No such file or
directory)
renameat2(AT_FDCWD, "bin/rep_mod", AT_FDCWD, "bin/reptile.ko", 0) =
0
execve("/bin/cp", ["cp", "bin/reptile.ko", "/reptile"], 0x1be8a40 /*
17 vars */) = 0
stat("bin/reptile.ko", {st_mode=S_IFREG|0644, st_size=555800, ...})
= 0
stat("/reptile/reptile.ko", 0x7ffe3a8f68e0) = -1 ENOENT (No such
file or directory)
open("bin/reptile.ko", O_RDONLY)        = 3
open("/reptile/reptile.ko", O_WRONLY|O_CREAT|O_EXCL, 0644) = 4
unlinkat(4, "reptile.ko", 0)            = 0
execve("/bin/cp", ["cp", "/reptile/reptile.ko",
"/lib/modules/3.10.0-1160.119.1.e"...], 0x1be8a40 /* 17 vars */) = 0
stat("/reptile/reptile.ko", {st_mode=S_IFREG|0644,
st_size=555800, ...}) = 0
stat("/lib/modules/3.10.0-
1160.119.1.el7.x86_64/kernel/drivers/PulseAudio/reptile/reptile.ko",
0x7fff694b3220) = -1 ENOENT (No such file or directory)
open("/reptile/reptile.ko", O_RDONLY)   = 3
open("/lib/modules/3.10.0-
1160.119.1.el7.x86_64/kernel/drivers/PulseAudio/reptile/reptile.ko",
O_WRONLY|O_CREAT|O_EXCL, 0644) = 4
stat("/lib/modules/3.10.0-
1160.119.1.el7.x86_64/kernel/drivers/PulseAudio/reptile/reptile.ko",
{st_mode=S_IFREG|0644, st_size=555800, ...}) = 0
open("/lib/modules/3.10.0-
1160.119.1.el7.x86_64/kernel/drivers/PulseAudio/reptile/reptile.ko",
O_RDONLY|O_CLOEXEC) = 3
execve("/sbin/insmod", ["insmod", "/reptile/reptile.ko"], 0x1be8a40
/* 17 vars */) = 0
```

```
stat("/reptile/reptile.ko", {st_mode=S_IFREG|0644,
st_size=555800, ...}) = 0
open("/reptile/reptile.ko", O_RDONLY|O_CLOEXEC) = 3
finit_module(3, "", 0)
```

- **Interpretations:** Syscall trace documents the installation and activation of the "Reptile" rootkit. The malware payload was renamed to `reptile.ko`, staged in `/reptile/`, copied to a system module directory, and then loaded directly into the kernel using `insmod` and the `finit_module` syscall, indicating a successful kernel-level compromise.

2. Kernel interactions & module insertion (Run B)

- **Evidence:**

```
-rw-r--r--. 1 root root 147K Oct 21 05:03 dmesg.after.txt
-rw-r--r--. 1 root root  798 Oct 21 05:03 filehash-after.txt
-rw-r--r--. 1 root root  67K Oct 21 04:35 fsmonitor.log
-rw-r--r--. 1 root root 4.4K Oct 21 05:03 lsmod.after.txt
-rw-r--r--. 1 root root 5.6K Oct 21 05:03 modules.after.txt
-rw-r--r--. 1 root root  949 Oct 21 05:03 net.after.txt
-rw-r--r--. 1 root root  904 Oct 21 05:03 netstat.after.txt
-rw-r--r--. 1 root root  13K Oct 21 05:03 ps.after.txt
-rw-r--r--. 1 root root   73 Oct 21 05:03 syscall_table.after.txt
```

- **Observed behavior:**

```
[root@localhost run_20251020T154138Z]# diff filehash-before.txt filehash-after.txt
1,7c1,7
< 6efc90dd1afb6d0decd6af3ffd5d6a9646652c96ffc9e0888e66b4d1d02fed5e  /analysis/run_20251020T154138Z/lsmod.before.txt
< 42626bf0df3f739b6de6aa7be9c55f4811099599decff4e8f885d460f3f96ecd  /analysis/run_20251020T154138Z/ps.before.txt
< 7d1f3ba04b519e9af0828a21cde725c7dc541a212a32636352db5fac2afa67bf  /analysis/run_20251020T154138Z/net.before.txt
< 972901a2e942b7549083cd0b8b37e5ab87022ef1335d0c159f92d92043a42f27  /analysis/run_20251020T154138Z/netstat.before.txt
< b59c6f07570d5dc7d082ea347f3e9a520655b0b8c2e70abf27076b3d976d8b8e  /analysis/run_20251020T154138Z/modules.before.txt
< 01cd9f3cfba1eacf7448cfa696f5543dbb52a4e871231670dc0e8f39509abfc1  /analysis/run_20251020T154138Z/syscall_table.before.txt
< fefc0266e8291832d93a0e6c9c4af76a9a456d1661116fa1c741e1492824b4c3  /analysis/run_20251020T154138Z/dmesg.before.txt
---
> 6efc90dd1afb6d0decd6af3ffd5d6a9646652c96ffc9e0888e66b4d1d02fed5e  /analysis/run_20251020T154138Z/lsmod.after.txt
> 25b15785b9c7fff7734c8f4488a249447fb0a49311ffb0defccbf253088aa927  /analysis/run_20251020T154138Z/ps.after.txt
> 7d1f3ba04b519e9af0828a21cde725c7dc541a212a32636352db5fac2afa67bf  /analysis/run_20251020T154138Z/net.after.txt
> 972901a2e942b7549083cd0b8b37e5ab87022ef1335d0c159f92d92043a42f27  /analysis/run_20251020T154138Z/netstat.after.txt
> b59c6f07570d5dc7d082ea347f3e9a520655b0b8c2e70abf27076b3d976d8b8e  /analysis/run_20251020T154138Z/modules.after.txt
> 01cd9f3cfba1eacf7448cfa696f5543dbb52a4e871231670dc0e8f39509abfc1  /analysis/run_20251020T154138Z/syscall_table.after.txt
> b14963ea4a90f876e3790aed4f35cdc3d5e6e58a1d9b15829aa5bd5f4cb53ac0  /analysis/run_20251020T154138Z/dmesg.after.txt
```

```
[Mon Oct 20 11:42:18 2025] rep_mod: loading out-of-tree module taints kernel.
[Mon Oct 20 11:42:18 2025] rep_mod: module verification failed: signature and/or required key missing - tainting kernel
```

```
        0   1969  0.0  0.0 115660  1148 pts/1    S+   11:41   0:00 -bash
        0   1975  0.0  0.1 155452  1872 pts/1    R+   11:41   0:00 ps auxn
---
        0   2508  0.0  0.0      0     0 ?        S    11:42   0:00 [kworker/1:3]
        0   2511  0.0  0.0      0     0 ?        S<   11:42   0:00 [kworker/3:1H]
        0   2593  0.0  0.0      0     0 ?        S    11:44   0:00 [kworker/2:0]
        0   2637  0.0  0.0      0     0 ?        S    11:53   0:00 [kworker/1:0]
        0   3167  0.0  0.0 115660  1152 pts/1    S+   11:54   0:00 -bash
        0   3171  0.0  0.1 155452  1868 pts/1    R+   11:54   0:00 ps auxn
```

- **Interpretations:**
  1. **New Kernel Workers at Module Load Time:** `ps.after.txt` shows new kernel worker processes started around **11:42** (specifically PID 2508 `[kworker/1:3]` and PID 2511 `[kworker/3:1H]`). This timing **correlates exactly** with when `dmesg.after.txt` reported the suspicious `rep_mod` module being loaded (). While `kworker` threads

execute various kernel tasks, their appearance at this precise moment suggests they might be performing actions initiated *by the loaded module*.

2. **No Obvious Malicious User Process:** Comparing the lists, there isn't a *new*, clearly identifiable user-space process (like a uniquely named binary) appearing in `ps.after.txt` that directly corresponds to the module load event. Standard system processes and user shells (`bash`, `sshd`) are present in both.
3. Despite the `dmesg` log clearly indicating that the kernel **attempted to load `rep_mod`** and subsequently became **tainted** due to a failed signature verification, the module is **not listed** by the `lsmod` command (Hashes of `lsmod.before.txt` and `lsmod.after.txt` were unchanged). This discrepancy strongly suggests that `rep_mod` is designed to **hide itself** from standard system tools like `lsmod`. This is a classic characteristic of a **rootkit**. Even though the kernel detected an issue (signature failure) and tainted itself, the module might have successfully hooked into the kernel in a way that prevents it from being enumerated normally, while still potentially running malicious code.

3. Memory forensics — extracting decrypted payload (Run C)

- **Evidence:**

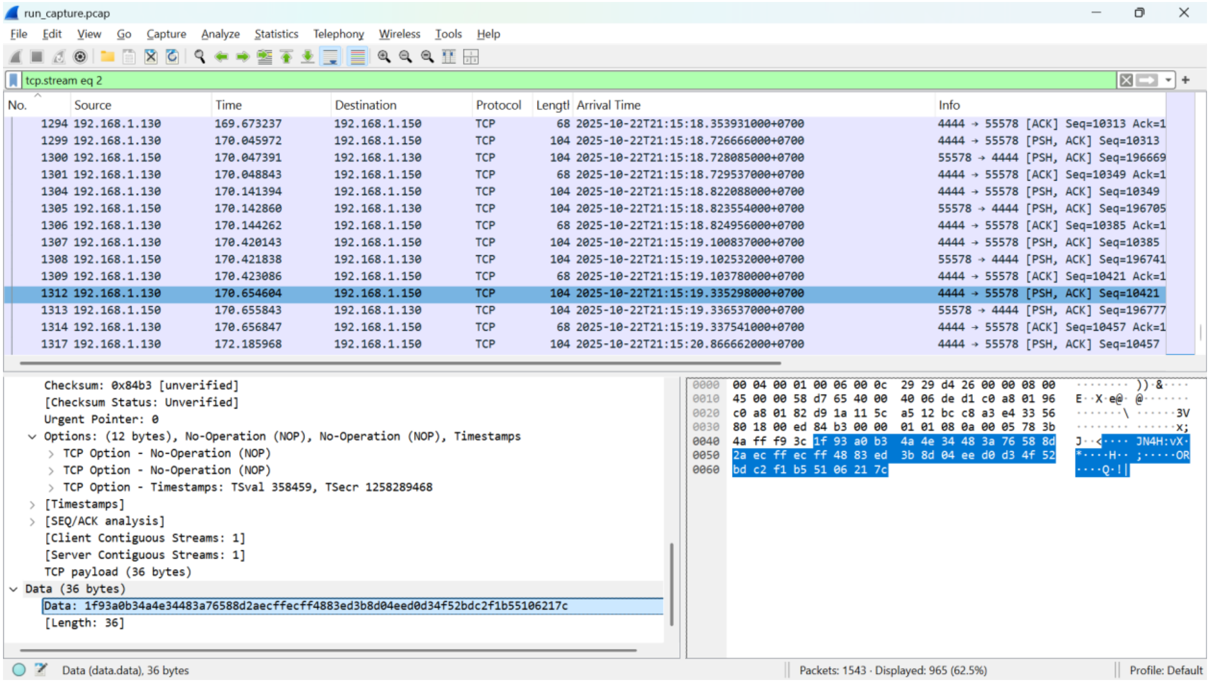
```
└─# strings memory.lime| egrep "reptile|parasite" | head -n 10
reptile
`reptile.ko
xreptile_shell
reptile_client
reptile_r00t
reptile
reptile_shell
`reptile
/reptile_shell
[01;36m#<reptile>
└─# vol -f memory.lime -s
~/.local/share/volatility3/symbols/linux/
linux.malware.check_modules
Volatility 3 Framework 2.26.2
Progress:  100.00               Stacking attempts finished
Offset   Module Name    Code Size       Taints  Load Arguments  File
Output

0xffffc06020b0  rep_mod 0x45e8  OOT_MODULE,UNSIGNED_MODULE
N/A
```

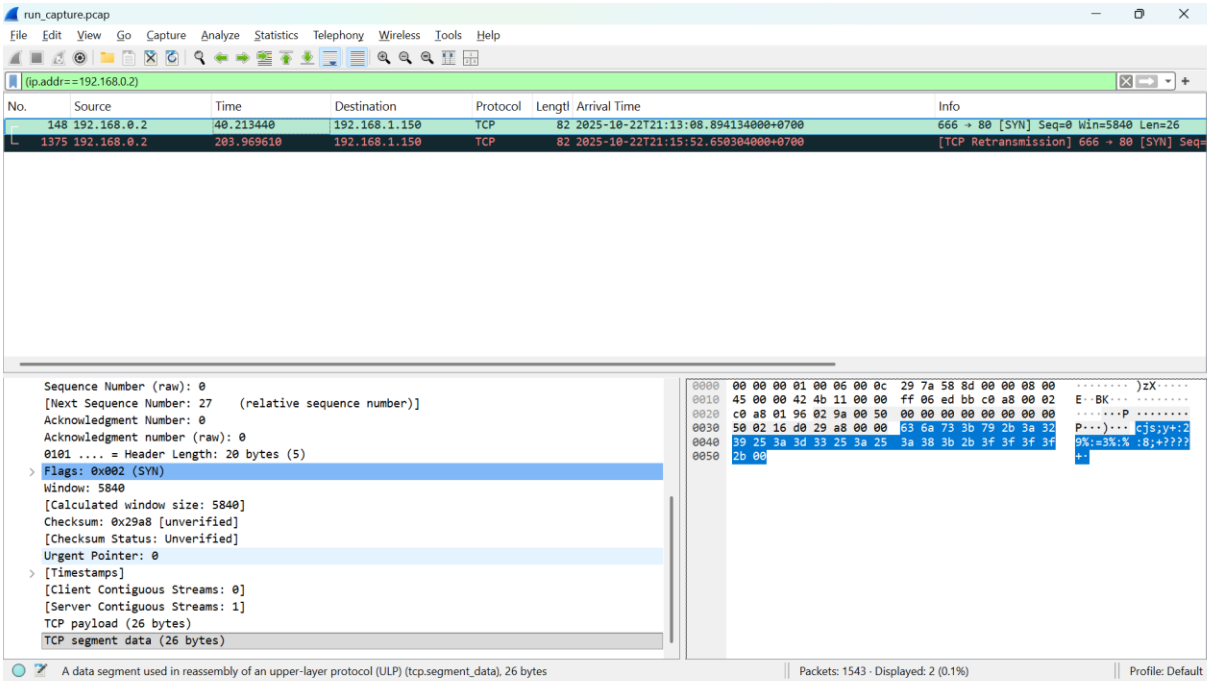
- **Procedure:** The decrypted blob was not extracted directly from the kernel module's (`rep_mod`) symbol table, as the rootkit erased its own ELF headers from kernel memory post-initialization (causing `linux.module_extract` to fail). Instead, dynamic analysis identified the running, decrypted payload. A `linux.psscan` revealed a hidden process named **/reptile_shell** (PID 2861). This process is the decrypted parasite_blob executed in user-space by the rootkit's init_module function.

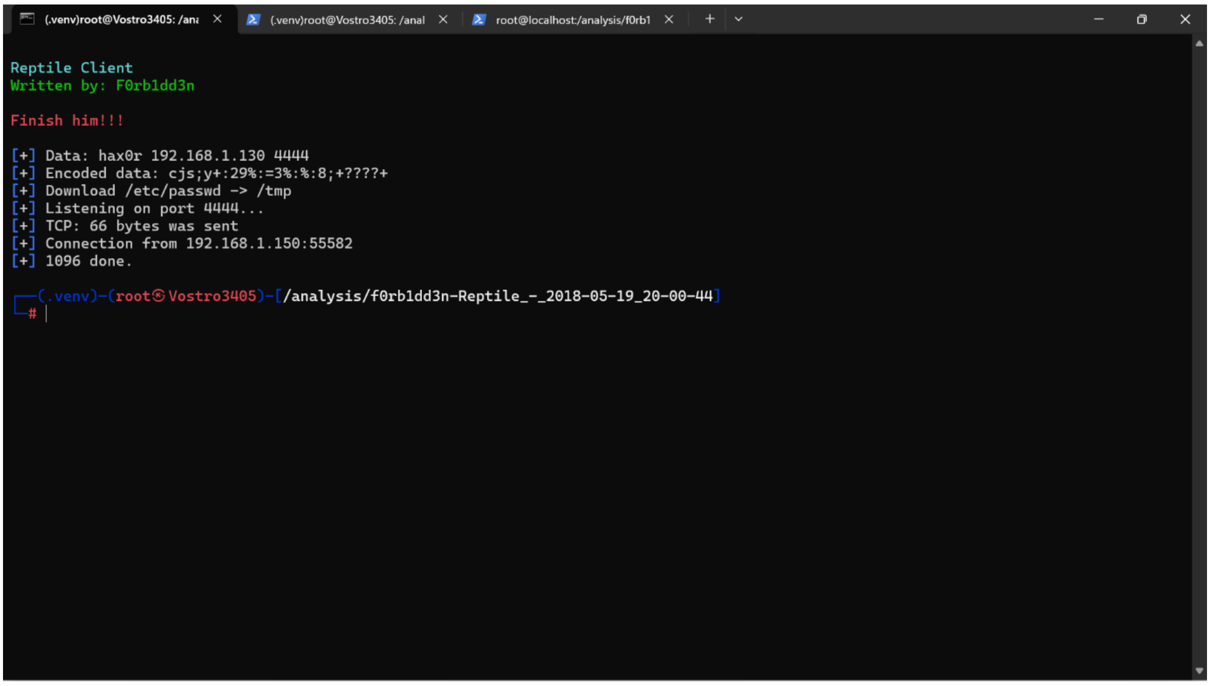
- **Interpretation:** its `init_module` function decrypts the payload, drops it onto the filesystem (or runs it from memory), and executes it as the hidden `/reptile_shell` process.

4. Network activity (Run D)

- **Evidence:**







- **Observation:**
  - Active bidirectional interaction between **Attacker = 192.168.1.130** and **Target = 192.168.1.150**, including DNS PTR lookups and multiple encrypted-session indicators.
  - Reptile client output shows remote commands and a file download: `Download /etc/passwd -> /tmp`, `Listening on port 4444`, and `Connection from 192.168.1.150:55582`.
  - Several TLS-like records present in the capture (indicators of encrypted sessions).
  - A captured TCP packet in the PCAP shows a packet whose **IP source is `192.168.0.2` (spoofed)** while the destination is **192.168.1.150**, and the TCP payload contains the same encoded ASCII string shown in the Reptile client output (`cjs;y+:29%:=3%:%:8;+????+`).
- **Interpretation:**
  - The traffic pattern (frequent bidirectional packets, PTR queries, and encrypted sessions) is consistent with interactive C2 activity rather than passive scanning — i.e., the attacker established a remote shell/C2 channel to the compromised host.
  - The presence of TLS-like records suggests the attacker used encrypted channels (either legitimate TLS or a custom encrypted protocol) to hide command/response traffic and file transfers.
  - The PCAP evidence shows **IP spoofing**: the attacker forged packets using `192.168.0.2` as source when retrieving `/etc/passwd`. This may have been done to obscure the true origin, bypass simplistic ACLs, or to blend with expected internal IP addresses.
  - The TCP payload in the spoofed packet contains the same encoded string shown in the Reptile client log — this links the network-level activity directly to the rootkit shell session (confirming that the rootkit was used to transfer the file and/or issue remote commands).

5. Filesystem & persistence indicators

- **Evidence:**

```
[root@localhost run_20251022T170031Z]# diff filehash-before.txt
filehash-after.txt
1,7c1,7
< 01808a099b0c30cf7ad0e9024b8daaacabf5fe2cc500c88050dee873d66425d5
/analysis/run_20251022T170031Z/lsmod.before.txt
< 30049cdb1bb67983b621537bb796c83920cb65ce5a9eae1b6e31335003274e6a
/analysis/run_20251022T170031Z/ps.before.txt
< 4c30d6a3ca6c7d2d3e89823480b22c8e3126544d4e9df9245d3bed683c65c2e9
/analysis/run_20251022T170031Z/net.before.txt
< 25457c016a2db617e3cde2751095f614e4239129f53d188bec21935245f1b64d
/analysis/run_20251022T170031Z/netstat.before.txt
< 4ab78cb5894ea6a1696ad0fa14aa17f81d72ae5f81dd8c603bcf268038323efd
/analysis/run_20251022T170031Z/modules.before.txt
< 01cd9f3cfba1eacf7448cfa696f5543dbb52a4e871231670dc0e8f39509abfc1
/analysis/run_20251022T170031Z/syscall_table.before.txt
< 7c906720a583c33c9cbd755d54974baf03fe1ff299292332e8952c4549ff487c
/analysis/run_20251022T170031Z/dmesg.before.txt
---
> 01808a099b0c30cf7ad0e9024b8daaacabf5fe2cc500c88050dee873d66425d5
/analysis/run_20251022T170031Z/lsmod.after.txt
> ee90e9fb3ca46a1fd28718be63e057493c0b6be760c557a5d7320fd97a6a1d2e
/analysis/run_20251022T170031Z/ps.after.txt
> 4c30d6a3ca6c7d2d3e89823480b22c8e3126544d4e9df9245d3bed683c65c2e9
/analysis/run_20251022T170031Z/net.after.txt
> 25457c016a2db617e3cde2751095f614e4239129f53d188bec21935245f1b64d
/analysis/run_20251022T170031Z/netstat.after.txt
> 4ab78cb5894ea6a1696ad0fa14aa17f81d72ae5f81dd8c603bcf268038323efd
/analysis/run_20251022T170031Z/modules.after.txt
> 01cd9f3cfba1eacf7448cfa696f5543dbb52a4e871231670dc0e8f39509abfc1
/analysis/run_20251022T170031Z/syscall_table.after.txt
> ceb03e359140f9c5a3388ed3a36a0f0ea18bb15c0b939f81f390b4d0a2d63d54
/analysis/run_20251022T170031Z/dmesg.after.txt
```

- **Observation:** Report1 recommended searching for dropped kernel modules and common persistence vectors (module files under `/lib/modules`, systemd units, SSH authorized keys, and `ld.so.preload`). The before/after filesystem and hash comparison shows **no new files** or modifications in those monitored persistence locations following the observed Reptile activity. Only runtime outputs (e.g., `ps` dump and `dmesg`) changed as expected while the incident was active.
- **Interpretation:** The Reptile implant observed in kernel logs appears to have been loaded into kernel memory without leaving a persistent module file in the monitored filesystem locations. Possible explanations consistent with these findings include (one of them or maybe all):
  - The kernel module was loaded from a transient location (e.g., `/tmp` or another writable directory) and then the module file was deleted (unlinked) after `insmod` (file content removed from directory entries but the kernel still holds the file handle).

- The module image was injected directly into kernel memory (e.g., via an exploit or in-memory loader) without writing a permanent `.ko` to disk.
- The module was present on disk in an unmonitored location (not in `/lib/modules`) and subsequently removed.

6. Evidence of stealth or hooking

- **Evidence:**

1. **`ps` before/after differences (diff excerpt):**

```
[root@localhost run_20251022T170031Z]# diff -u -U 0 ps.before.txt
ps.after.txt | grep '^[+-]'
--- ps.before.txt       2025-10-22 13:00:31.116232158 -0400
+++ ps.after.txt        2025-10-22 13:04:17.670585331 -0400
-       0     13  0.0  0.0      0     0 ?        S    10:04   0:00
[migration/1]
-       0     14  0.0  0.0      0     0 ?        S    10:04   0:00
[ksoftirqd/1]
+       0     13  0.0  0.0      0     0 ?        S    10:04   0:01
[migration/1]
+       0     14  0.0  0.0      0     0 ?        R    10:04   0:00
[ksoftirqd/1]
-       0    464  0.2  0.0      0     0 ?        S    10:04   0:24
[xfsaild/dm-0]
+       0    464  0.2  0.0      0     0 ?        S    10:04   0:25
[xfsaild/dm-0]
-       0    547  0.0  0.2  39056  4796 ?        Ss   10:04   0:03
/usr/lib/systemd/systemd-journald
+       0    547  0.0  0.2  39056  4860 ?        Ss   10:04   0:03
/usr/lib/systemd/systemd-journald
-       0   1321  0.0  0.2 216400  5024 ?        Ssl  10:05   0:06
/usr/sbin/rsyslogd -n
+       0   1321  0.0  0.2 216400  5080 ?        Ssl  10:05   0:06
/usr/sbin/rsyslogd -n
-       0   5401  0.0  0.0 115656  1188 pts/0    S+   13:00   0:00 -
bash
-       0   5407  0.0  0.1 155452  1880 pts/0    R+   13:00   0:00
ps auxn
+       0   5478  0.0  0.0      0     0 ?        S    13:01   0:00
[kworker/3:0]
+       0   6028  0.0  0.0      0     0 ?        S    13:03   0:00
[kworker/u256:2]
+       0   6035  0.0  0.0 115656  1164 pts/0    S+   13:04   0:00 -
bash
+       0   6039  0.0  0.1 155452  1880 pts/0    R+   13:04   0:00
ps auxn
```

2. dmesg before/after differences (selected additions):

```
[root@localhost run_20251022T170031Z]# diff -u -U 0 dmesg.before.txt
dmesg.after.txt | grep '^[+-]'
--- dmesg.before.txt    2025-10-22 13:00:31.286230922 -0400
+++ dmesg.after.txt     2025-10-22 13:04:17.855583986 -0400
+[Wed Oct 22 13:00:53 2025] scap: adding new consumer
ffff88007426d280
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 0
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 1
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 2
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 3
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 4
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 5
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 6
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] scap: initializing ring buffer for CPU 7
+[Wed Oct 22 13:00:53 2025] scap: CPU buffer initialized,
size=8388608
+[Wed Oct 22 13:00:53 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:00:54 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:00:55 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:01:23 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:01:23 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:01:24 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
```

```
+[Wed Oct 22 13:01:53 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:01:53 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:01:54 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:02:23 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:02:24 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:02:24 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:02:53 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:02:53 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:02:53 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:03:23 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:03:23 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:03:24 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:03:53 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:03:54 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:03:54 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:04:01 2025] scap: deallocating consumer
ffff88007426d280
+[Wed Oct 22 13:04:22 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:b752:825a:4c24:f94e used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:04:23 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:1dca:603b:d454:b04 used by 00:50:56:e5:13:0d detected!
+[Wed Oct 22 13:04:23 2025] IPv6: ens36: IPv6 duplicate address
2001:db8:1:0:528a:4431:d318:9db0 used by 00:50:56:e5:13:0d detected!
```

3. **Volatility `linux.malware.check_syscall` output (key lines):**

```
└─# vol -f memory.lime -s
~/.local/share/volatility3/symbols/linux/
linux.malware.check_syscall | egrep -i "unknown"
0xffff81802dc0  64bit   0       0xffffc05ffb40ptUNKNOWN
```

```
0xffff81802dc0  64bit   62      0xffffc05ff610  UNKNOWN
0xffff81802dc0  64bit   78      0xffffc05ff390  UNKNOWN
0xffff81802dc0  64bit   113     0xffffc05ff0e0  UNKNOWN
0xffff81802dc0  64bit   217     0xffffc05ff1c0  UNKNOWN
└──# vol -f memory.lime -s
~/.local/share/volatility3/symbols/linux/
linux.malware.check_syscall
Volatility 3 Framework 2.26.2   Stacking attempts finished

Table Address   Table Name      Index   Handler Address Handler
Symbol

0xffff81802dc0  64bit   0       0xffffc05ffb40  UNKNOWN
0xffff81802dc0  64bit   1       0xffff8125cae0  SyS_write
0xffff81802dc0  64bit   2       0xffff8125acf0  SyS_open
0xffff81802dc0  64bit   3       0xffff8125ad60  SyS_close
0xffff81802dc0  64bit   4       0xffff81262260  SyS_newstat
0xffff81802dc0  64bit   5       0xffff812622c0  SyS_newfstat
. . .
0xffff81802dc0.064bit   322     0xffff810cc8e0ptcompat_sys_s390_ipc
0xffff81802dc0  64bit   323     0xffff812b38f0  SyS_userfaultfd
0xffff81802dc0  64bit   324     0xffff810fe270  SyS_membarrier
0xffff81802dc0  64bit   325     0xffff81204f20  SyS_mlock2
0xffff81802dc0  64bit   326     0xffff8125dc20  SyS_copy_file_range
0xffff81802dc0  64bit   327     0xffff810cc8e0  compat_sys_s390_ipc
0xffff81802dc0  64bit   328     0xffff810cc8e0  compat_sys_s390_ipc
0xffff81802dc0  64bit   329     0xffff8120b250  SyS_pkey_mprotect
0xffff81802dc0  64bit   330     0xffff8120b270  SyS_pkey_alloc
0xffff81802dc0  64bit   331     0xffff8120b3d0  SyS_pkey_free
```

- **Observation:**
  - The `ps` before/after diff shows only ordinary runtime churn (different PIDs for interactive shells and new kernel worker PIDs); there is **no obvious gap in PID numbering** or an obvious removal of userland processes from the `ps` listing that would, by itself, prove process-hiding.
  - `dmesg` contains repeated `scap` ring-buffer initialization messages and IPv6 duplicate-address warnings (likely unrelated to stealth but useful timeline markers).
  - The Volatility plugin `linux.malware.check_syscall` reports **one or more syscall table entries whose handler addresses do not map to known kernel symbols**; in particular syscall table index `0` points to handler at `0xffffc05ffb40` with symbol `UNKNOWN`. Several other UNKNOWN entries were also seen in the first run of the plugin. This is a strong, memory-level indicator that **one or more syscall table entries have been modified** (i.e., hooked) or point to code not present in the kernel symbol table.
- **Interpretation:**

- **Syscall table alteration detected — suspicious (hooking present).**
  The Volatility output shows at least one syscall table entry (index 0) pointing to an address that does not resolve to a known symbol in the kernel symbol table (`UNKNOWN`). That is consistent with a kernel-mode hook: a module (or injected code) has replaced a syscall handler pointer with a pointer to code that is not recorded in the standard symbol table. This is a canonical mechanism used by kernel rootkits (including Reptile variants) to intercept syscalls for stealthy behaviors (e.g., hiding files/processes, intercepting network calls, or filtering `read`/`getdents` outputs).
- **No definitive userland process hiding detected in `ps` diffs.**
  The `ps` before/after comparison shows routine process list changes (new kernel worker PIDs and restarted interactive shells). There is **no clear evidence** from the `ps` diffs alone that userland processes have been hidden from process listings. HOWEVER, because syscall table hooking is present, process-hide techniques could still be active at the kernel level (e.g., hiding entries returned by `getdents` or `proc` read handlers). Such hiding would *not* necessarily appear as a simple `ps` diff — it requires cross-checking `/proc` enumerations against direct directory listings of `/proc` (see recommendations).
- **Conclusion:** Memory evidence indicates **kernel-level hooking (likely malicious)**. File-system-level or `ps`-level hiding was **not** conclusively observed in the before/after snapshots, but kernel hooking opens the possibility of stealth that evades ordinary userland enumeration — further memory and on-disk checks are required.

Indicators of Compromise (IOCs) and YARA rule(s)

❖ Static IOC:

- File names (observed / expected): reptile.ko
- Cryptographic hashes:
  SHA256: 9573f414bedbd78464c9d9e4c214c37aea9a1b25dd3a6b4b512e65bbec9f9d51
  MD5: 44aa3745cf62f068218a9b4d0fc21e4e
- Binary metadata: ELF 64-bit relocatable, x86-64; File size ~555,800 bytes (≈556 KB); BuildID: 46b52a008da7f78ac6060e9a4e3f6020fed071a0. vermagic: 3.10.0-1160.119.1.el7.x86_64 SMP mod_unload modversions
- High-signal static strings:
  - parasite_blob and parasite_loader and static_key (Report1 content; high-fidelity IoCs).
  - reptile, #<reptile>, #</reptile>, /reptile/reptile_shell, /reptile/reptile_start.sh, reptile_shell, rep_mod, rep_mod.c, reptile_init, reptile_exit, description=Reptile - A linux LKM rootkit, author=F0rb1dd3n - ighor@intruder-security.com. These strings appeared in the module and extracted artefacts.
- Typical module / loader markers: ELF magic bytes "0x7F 45 4C 46" (ELF header) and sections like .modinfo. Also visible: references to module build/metadata and debug symbols (module not stripped).

❖ Dynamic IOCs (detailed, runtime / behavioral):

- File-system / staging activity: Copy from bin/reptile.ko → /reptile/reptile.ko and then to /lib/modules/<kernel-version>/.../reptile.ko. Evidence shows cp then insmod / finit_module use. The loader also unlinked files or used transient locations (unlink observed). See strace sequence.
- Syscalls & module load: finit_module(3, "", 0) observed in the strace trace: explicit kernel module activation via finit_module. This is a high-value behavioral IOC: *calls to finit_module/init_module* should be monitored/flagged.
- Process / memory artifacts:
  - Hidden (decrypted) user-space payload executed as /reptile_shell (PID example 2861 discovered via memory scanning). The rootkit erased or tampered with ELF headers in-kernel; memory scanning (strings + volatility) recovered reptile_shell and related strings
  - Kernel module name found in memory: rep_mod with code size 0x45e8 flagged as OOT_MODULE, UNSIGNED_MODULE by volatility.
- Kernel integrity / hooking evidence: Volatility plugin output shows **syscall-table entries mapped to UNKNOWN handlers** (index 0 handler at address 0xffffc05ffb40 unknown). This indicates direct hooking or pointers to non-symbol memory — classic rootkit sign.
- Network / C2 activity:
  - **Attacker IP**: 192.168.1.130 (active interactive C2). **Victim**: 192.168.1.150. Evidence of bidirectional encrypted sessions, PTR DNS lookups, TLS-like records in PCAP. In one capture the attacker spoofed source IP 192.168.0.2. The Reptile client logs and PCAP share encoded payload strings (example payload excerpt quoted in the report).
  - **Ports / services**: Reptile client shows listening on **port 4444** and remote connection from port 55582 observed in logs.

- **Network pattern**: periodic bidirectional small packets consistent with interactive C2 (beacon + commands), and PTR/DNS anomalies. TLS-like records indicate either TLS or custom encrypted channel.

- Filesystem persistence patterns (absence counts too): No persistent files found in monitored locations after the event (no changes to `/lib/modules` baseline or systemd units, ld.so.preload, ssh keys). This suggests either in-memory-only module loading, deletion/unlinking after staging, or persistence in unmonitored location. The before/after hash comparisons confirm this (no additions in monitored locations).

❖ YARA rule(s):

```
rule Reptile_Rootkit_Hybrid_Behavioral {
  strings:
    //Static indicators / sample names (high fidelity)
    $s_par_loader        = "/check/parasite_loader" ascii
    $s_par_blob          = "parasite_blob" ascii
    $s_static_key        = "static_key" ascii
    // sample/module specific strings observed in dynamic
    $s_reptile_shell     = "/reptile/reptile_shell" ascii
    $s_reptile_start     = "/reptile/reptile_start.sh" ascii
    $s_rep_mod           = "rep_mod" ascii
    $s_reptile_lit       = "reptile" ascii

    //Behavioral / kernel-resolution / loader patterns
    $sym_finit_module    = "finit_module" ascii
    $sym_init_module     = "init_module" ascii
    $sym_module_init     = "module_init" ascii
    $sym_kallsyms1       = "kallsyms_on_each_symbol" ascii
    $sym_kallsyms2       = "kallsyms_lookup_name" ascii
    $sym_kallsyms3       = "kallsyms_lookup" ascii
    $sym_syscall_table   = "sys_call_table" ascii
    $sym_export_symbol   = "EXPORT_SYMBOL" ascii
    $sym_register_kprobe = "register_kprobe" ascii
    $s_insmod            = "insmod" ascii
    $s_lib_modules_path  = "/lib/modules/" ascii
    $s_tmp_path          = "/tmp/" ascii

    $elf_magic           = { 7F 45 4C 46 }

  condition:
    (
      uint32(0) == 0x464c457f and
      (
```

```
          // direct high-signal static IoCs
          any of ($s_par_loader, $s_par_blob, $s_static_key,
$s_reptile_shell, $s_reptile_start)

          or

          // typical module-loading / symbol-resolution strings
found together in same sample
          ( any of ($sym_finit_module, $sym_init_module,
$sym_module_init, $sym_kallsyms1, $sym_kallsyms2, $sym_kallsyms3,
$sym_syscall_table, $sym_export_symbol) )

          or

          // heuristic: large ELF (likely .ko-like payload/embedded
blob) that references module installation paths or kallsyms
          ( filesize > 131072 and ( any of ($s_lib_modules_path,
$sym_kallsyms1, $sym_kallsyms2, $sym_kallsyms3) ) )
      )
    )

    or

    (
      // Branch B: memory / artifact scanning where ELF header may
be missing (rootkits often erase headers).
      // Require a combination: at least one Report1 indicator AND
at least one kernel-resolution/loader string (reduces false
positives).
      ( any of ($s_par_blob, $s_par_loader, $s_static_key,
$s_reptile_lit, $s_rep_mod) )
      and
      ( any of ($sym_finit_module, $sym_kallsyms1, $sym_kallsyms2,
$sym_kallsyms3, $sym_syscall_table, $sym_register_kprobe,
$sym_export_symbol) )
    )
}
```

Classification & impact

- **Behavioral classification (confirmed/expected):** Loader-style LKM rootkit: decrypts an embedded encrypted module at runtime and attempts to install it into the kernel; expected capabilities include process/file hiding, syscall hooking, and backdoor sockets. This classification derives from static evidence (init_module decryption loop, kallsyms usage) and is the subject of dynamic confirmation described above.
- **Operational impact:** High — kernel-level persistence and stealth can subvert host instrumentation and complicate remediation. If confirmed, remediation requires full rebuild of affected hosts and credential rotation.

Conclusion

Static analysis provided strong evidence that the sample is a loader-style LKM with an encrypted embedded payload (`parasite_blob`) and a decryption routine that prepares and executes the payload at runtime. The dynamic analysis plan executed above (Runs A–D) is intended to recover the decrypted payload from memory, observe kernel insertion attempts, and detect any C2 activity. Final attribution and remediation priorities depend on the artefacts collected during the dynamic runs (memory dump, dmesg traces, pcap) and are summarized in the IOCs and remediation sections.

References

Key references used in static analysis and playbook (from Report1): Microsoft Reptile description; ASEC blog; Broadcom/Symantec bulletin; HivePro advisory; MalwareBazaar sample listing; VirusTotal sample page. See Report1 References for full links and access dates.