SEPTEMBER 19, 2025

MALWARE CLASSIFICATION

ROOTKIT

NGUYỄN HOÀNG THANH PHONG – SE184915
FPT UNIVERSITY

Contents

1. Executive summary

This report classifies high impact malware family – **Rootkit** – and presents an applied analysis of a provided Linux binary sample (***MD5 d1abb8c012cc8864dcc109b5a15003ac***). Static triage of the sample reveals strings and module-like artifacts consistent with a **loader / LVM-style rootkit** (e.g., marker such as ***parasite_loader, parasite_blob, kmatryoshka.c, encrypt.h***. The sample therefore likely functions as a rootkit/loader (Reptile-like behavior), and further dynamic analysis under strongly isolated conditions is recommended to confirm persistence mechanisms, kernel integration, network behavior, and whether it acts as a dropper for trojans or ransomware.

2. Introduction

Malicious software (Malware) presents varied functionality and risk: from cover persistence and system compromise (Rootkit), to credential theft and remote control (Trojan), to disruptive extortion (ransomware). Correct classification is essential for selecting detection techniques, response actions, and forensics procedures. This report ***(1)*** defines the **"Rootkit"** family, ***(2)*** highlights distinguishing behaviors and mitigations, and ***(3)*** applies the classification to a real ELF sample.

3. Scope & Methodology

3.1. Scope:

- Focus: Rootkit (Linux-targeted rootkits included)
- Applied: static triage of the provided ELF binary; recommended dynamic and forensic procedures.

3.2. Methodology:

1. Static triage: computer hash, identify filetype, extract printable strings, inspect for obivous artifacts.
2. Produce IOCs and defensive detection rules (YARA).
3. Recommend step-by-step dynamic analysis runbook to be executed in an isolated sandbox/VM.
4. Provide containment and remediation guidance appropriate.

4. Malware taxonomy overview – Rootkit

- **Definition:** Software designed to hide the presense of processes, files, network sockets, or other artifacts; on Linux, often implemented as a Loadable Kernel Module (LKM) or via kernel-level exploits.
- **Capabilities:** process/file hiding, syscall hooking, backdoor sockets or port-knocking, priviledged remote shells.
- **Risk profile:** extremely high – kernel-level rootkits can evade many detection controls and complicate remediation. Rootkits are commonly combined with trojans to maintain persistence and conceal subsequent payloads (including ransomware).

5. Detailed analysis

5.1. Sample metadata (static triage)

- **MD5:** d1abb8c012cc8864dcc109b5a15003ac
- **SHA-1:** 2ca4787d2cfffac722264a8bdae77abd7f4a2551
- **SHA-256:** d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba

```
┌──(phong184915㉿Vostro3405)-
[~/Downloads/malware/Reptile/Reptile_Rootkit]
└─$ exiftool
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf
ExifTool Version Number         : 13.25
File Name                       :
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf
Directory                       : .
File Size                       : 644 kB
File Modification Date/Time     : 2025:09:16 08:33:36+07:00
File Access Date/Time           : 2025:09:19 15:22:08+07:00
File Inode Change Date/Time     : 2025:09:19 15:21:48+07:00
File Permissions                : -rw-r--r--
File Type                       : ELF object file
File Type Extension             : o
MIME Type                       : application/octet-stream
CPU Architecture                : 64 bit
CPU Byte Order                  : Little endian
Object File Type                : Relocatable file
CPU Type                        : AMD x86-64
```

- **Static strings:**

```
┌──(phong184915㉿Vostro3405)-
[~/Downloads/malware/Reptile/Reptile_Rootkit]
└─$ strings
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf
| grep -E "(.+\.[ch])|(.*key.*)|(system)|(.*domain.*)" | sort
. . .
/check/parasite_loader/kmatryoshka.c
/check/parasite_loader/parasite_loader.mod.c
. . .
encrypt.h
```

```
exec_domain
. . .
kmatryoshka.c
kmatryoshka.c
. . .
parasite_loader.mod.c
parasite_loader.mod.c
. . .
static_key
static_key
static_key_mod
static_key_mod
sysfs.h
sysfs.h
system_freezable_wq
system_freezable_wq
system_wq
system_wq
. . .
vector_allocation_domain
vector_allocation_domain
```

- **Observation:** strings strongly indicate a loader component and encryption-related functionality (e.g., ***encrypt.h***, ***static_key***) plus module-like source file names (e.g., ***kmatryoshka.c***), consistent with an LKM loader or userland loader for kernel components.

5.2. Interpretation

- The sample appears to be a loader/backdoor intended to install or manage a parasite module (a rootkit), possibly implementing encryption or encrypted payloads (hence references to ***encrypt*** and ***static_key***).
- No plaintext C2[1] domain names were observed in extracted strings; this suggests dynamic C2 resolution (encrypted at rest, DGA[2], or C2 loaded at runtime) or out-of-band mechanisms (e.g., port-knocking).

5.3. Likely family / analog

- Markers and structure are consistent with **Reptile-like LKM Rootkits** or a loader for such modules (open-source/modified Reptile variants are known to be used

[1] Command and Control: refers to the infrastructure that attackers use to remotely communicate with compromised systems
[2] Domain Generation Algorithm: is a method by which malware dynamically creates a large number of domain names

in the wild to achieve kernel persistence and concealment). Static artifacts (module filenames, parasite_blob) commonly appear in loader-style rootkits.

5.4. Recommended immediate triage actions in reality.

- Register hash code (MD5, SHA256, ...) in defensive systems (EDR/AV/SIEM) and block/exclude execution.
- Add YARA detection.
- Search file shares and endpoints for files containing ***parasite_loader***/***parasite_blob*** strings or matching hash.

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

1. IOCs (static)

- **SHA-256:** d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba
- **Filetype:** ELF64 executable (Linux x86_64)
- **High-signal strings:**

```
/check/parasite_loader
/check/parasite_loader/encrypt
/check/parasite_loader/kmatryoshka.c
/check/parasite_loader/parasite_loader.mod.c
encrypt.h
kmatryoshka.c
parasite_blob
parasite_loader
parasite_loader.mod.c
static_key
static_key_mod
```

Note: the presence of these strings alone does not prove malicious activity on its own (false positives possible); use combined telemetry to validate.

2. Defensive YARA (static)

```
rule ReptileRootkit {
  strings:
    $a="/check/parasite_loader"
    $b="/check/parasite_loader/encrypt"
    $c="/check/parasite_loader/kmatryoshka.c"
    $d="/check/parasite_loader/parasite_loader.mod.c"
    $e="encrypt.h"
    $f="kmatryoshka.c"
    $g="parasite_blob"
    $h="parasite_loader"
    $i="parasite_loader.mod.c"
    $k="static_key"
    $j="static_key_mod"
    $magik={7F 45 4C 46}
  condition:
    $magik and any of ($a,$b,$c,$d,$e,$f,$g,$h,$i,$k,$j)
}
```

7. Recommended analysis playbook

7.1. Static triage

- Compute hashes: `sha256sum sample.elf`
- Extract strings: `strings sample.elf | less`
- Basic header: `readelf -h sample.elf, readelf -d sample.elf`
- Check entropy/packers: `binwalk`

7.2. Controlled dynamic analysis

- Prepare sandbox: Fresh VM snapshot. No production network access. Optional controlled proxy (logger) for limited outbound connectivity.
- Baseline captures: Pre-run: `lsmod > lsmod.before`, `ps aux > ps.before`, `ls -la /`, `netstat -tulpen > net.before`.
- Run the binary (on copy)
  - Start `tcpdump -w pcap.cap` and `sysdig -w capture.scap`.
  - Run under `strace -ff -o strace.out ./sample.elf` (or with `gdb` if safe).
- Observe
  - Watch kernel module lists: `lsmod`, check `/proc/modules`.
  - Monitor filesystem writes and new files (inotify, `auditd` logs).
  - Monitor for outbound connections or listener sockets (e.g., port 4444).
- Memory and disk capture: If suspicious behavior observed, capture memory image (`liquid`/`dd` or hypervisor snapshot) then preserve disk image.
- Post-run:
  - Collect logs: `strace` output, `sysdig` captures, pcap file, `dmesg`, `journalctl`.
  - Compare `lsmod.before` vs `lsmod.after`, `ps.before` vs `ps.after`.

7.3. Forensic analysis

- Kernel module artifacts: examine `/lib/module/*` for unknown files; `modinfo <module>` if present.
- Search for persistence vectors: systemd unit files, cron entries, injected SSH keys, modified `/etc/ld.so.preload`.
- Inspect network pcap for C2 patterns, port-knocking sequences, and data exfiltration channels.

8. Comparison & operational impact

- **Trojan:** typically initial access; medium compromise lifetime until detected.
- **Ransomware:** high immediate operational impact (data unavailability); incentivizes immediate action.
- **Rootkit:** high stealth and long-term compromise; hardest to detect and eradicate.
- **Worst-case scenario:** rootkit + trojan + ransomware used together — rootkit conceals trojan activity while ransomware performs destructive/encrypting operations. Response sequencing must prioritize containment and evidence preservation.

9. Conclusion

- The supplied sample show static artifacts consistent with a loader/backdoor intended to manage a parasite/rootkit module. This aligns with Reptile-style rootkits (or close variants).
- Immediate actions: block the sample's hash, deploy the YARA rule in monitoring engines, search the environment for matching aritfacts, and plan an isolated dynamic analysis to confirm runtime behavior.
- If kernel-level compromise is confirmed, plan for full system rebuilds from verified backups and full credential rotation.

10. References

[1] Microsoft, “Trojan:Linux/Reptile.C threat description”, *Microsoft Security Intelligence*, 2023. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Linux/Reptile.C. [Accessed: Sep. 19, 2025].

[2] AhnLab Security Emergency response Center (ASEC), “Reptile malware targeting Linux systems”, *ASEC Blog*, 2022. [Online]. Available: https://asec.ahnlab.com/en/55785. [Accessed: Sep. 19, 2025].

[3] Broadcom (Symantec), “Reptile: Linux rootkit malware protection bulletin”, *Reptile - a Linux rootkit malware,* 2021. [Online]. Available: https://www.broadcom.com/support/security-center/protection-bulletin/reptile-a-linux-rootkit-malware. [Accessed: Sep. 19, 2025].

[4] HivePro, “Reptile rootkit targets Linux systems in South Korea,” *Threat Advisory PDF*, 2022. [Online]. Available: https://www.hivepro.com/wp-content/uploads/2023/08/Reptile-Rootkit-Targets-Linux-Systems-in-South-Korea_TA2023326.pdf [Accessed: Sep. 19, 2025].

[5] Sandfly Security, “Detecting Linux stealth rootkits with directory link errors”, *Sandfly Security Blog*, 2021. [Online]. Available: https://sandflysecurity.com/blog/detecting-linux-stealth-rootkits-with-directory-link-errors. [Accessed: Sep. 19, 2025].

[6] abuse.ch, “Sample d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba”, *MalwareBazaar*, 2025. [Online]. Available: https://bazaar.abuse.ch/sample/d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba. [Accessed: Sep. 19, 2025].

[7] VirusTotal, “File d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba”, *VirusTotal*, 2025. [Online]. Available: https://www.virustotal.com/gui/file/d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba. [Accessed: Sep. 19, 2025].

[8]OpenAI, “Danh sách Malware 2020+”, ChatGPT, 2025. [Online]. Available: https://chatgpt.com/share/68cd7695-8d58-800e-b739-7e19b90e75cb [Accessed: Sep. 19, 2025]

11. Appendix A – Deep Static triage Analysis

11.1. Sample metadata & hashes

- Status, and file hash:

```
MD5: d1abb8c012cc8864dcc109b5a15003ac
SHA1: 2ca4787d2cfffac722264a8bdae77abd7f4a2551
SHA256:
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba
File:
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf
  Size: 644496        Blocks: 1264       IO Block: 4096   regular
file
Device: 8,3     Inode: 5767938     Links: 1
Access: (0755/-rwxr-xr-x)  Uid: ( 1000/phong184915)   Gid:
( 1000/phong184915)
Access: 2025-09-19 22:49:59.341945055 +0700
Modify: 2025-09-16 08:33:36.000000000 +0700
Change: 2025-09-19 22:49:55.053846273 +0700
 Birth: 2025-09-19 15:21:48.410103303 +0700
```

- Check ELF Header:

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          642704 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         28
```

```
  Section header string table index: 25
```

- Check file-permission, last-modified:

```
ls -lh
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf
-rwxr-xr-x 1 phong184915 phong184915 630K Sep 16 08:33
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf
```

11.2. Section & Symbol table extraction & Readable strings

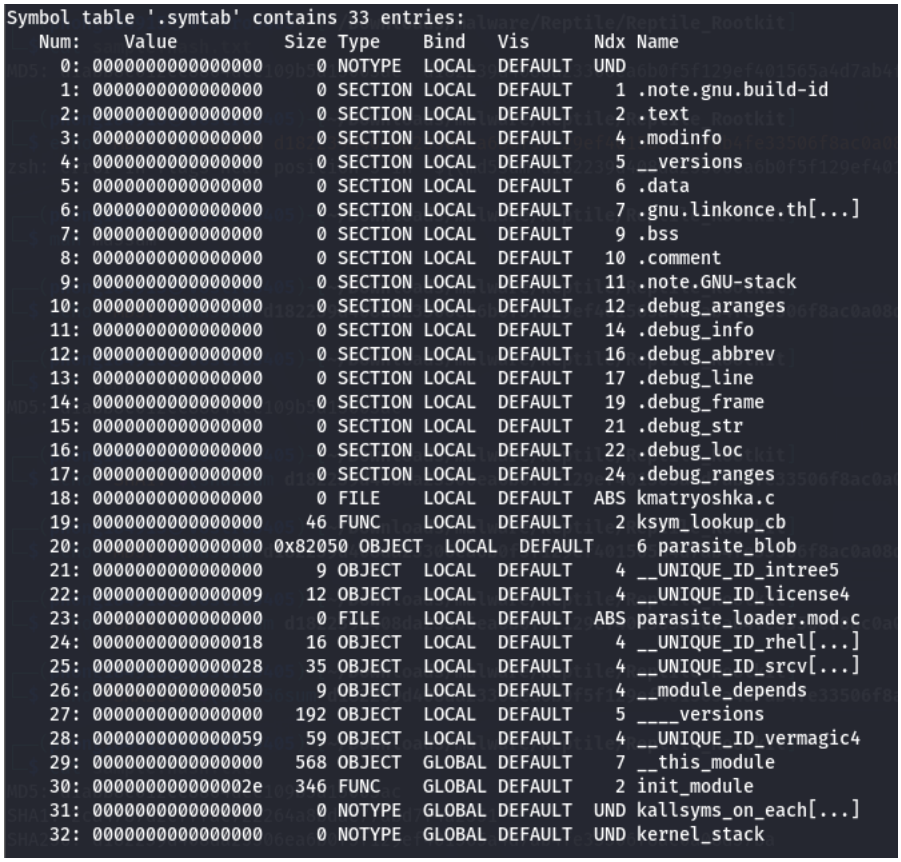
- Extract general strings:

```
$ strings
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf
>
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.str
ings.txt
```

- The rootkit employs encryption to protect its payloads and configuration data.

```
grep -Ei
'encrypt|decrypt|AES|RSA|openssl|key|private|public|pass|password|st
atic_key'
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.str
ings.txt| sort -u
8key
AEs0(c
/check/parasite_loader/encrypt
encrypt.h
static_key
static_key_mod
```

- Symbol table analysis:



```
Symbol table '.symtab' contains 33 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .note.gnu.build-id
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    2 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .modinfo
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 __versions
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    6 .data
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7 .gnu.linkonce.th[...]
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    9 .bss
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT   10 .comment
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT   11 .note.GNU-stack
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT   12 .debug_aranges
    11: 0000000000000000     0 SECTION LOCAL  DEFAULT   14 .debug_info
    12: 0000000000000000     0 SECTION LOCAL  DEFAULT   16 .debug_abbrev
    13: 0000000000000000     0 SECTION LOCAL  DEFAULT   17 .debug_line
    14: 0000000000000000     0 SECTION LOCAL  DEFAULT   19 .debug_frame
    15: 0000000000000000     0 SECTION LOCAL  DEFAULT   21 .debug_str
    16: 0000000000000000     0 SECTION LOCAL  DEFAULT   22 .debug_loc
    17: 0000000000000000     0 SECTION LOCAL  DEFAULT   24 .debug_ranges
    18: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS kmatryoshka.c
    19: 0000000000000000    46 FUNC    LOCAL  DEFAULT    2 ksym_lookup_cb
    20: 0000000000000000 0x82050 OBJECT  LOCAL  DEFAULT    6 parasite_blob
    21: 0000000000000000     9 OBJECT  LOCAL  DEFAULT    4 __UNIQUE_ID_intree5
    22: 0000000000000009    12 OBJECT  LOCAL  DEFAULT    4 __UNIQUE_ID_license4
    23: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS parasite_loader.mod.c
    24: 0000000000000018    16 OBJECT  LOCAL  DEFAULT    4 __UNIQUE_ID_rhel[...]
    25: 0000000000000028    35 OBJECT  LOCAL  DEFAULT    4 __UNIQUE_ID_srcv[...]
    26: 0000000000000050     9 OBJECT  LOCAL  DEFAULT    4 __module_depends
    27: 0000000000000000   192 OBJECT  LOCAL  DEFAULT    5 ____versions
    28: 0000000000000059    59 OBJECT  LOCAL  DEFAULT    4 __UNIQUE_ID_vermagic4
    29: 0000000000000000   568 OBJECT  GLOBAL DEFAULT    7 __this_module
    30: 000000000000002e   346 FUNC    GLOBAL DEFAULT    2 init_module
    31: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND kallsyms_on_each[...]
    32: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND kernel_stack
```

```
18: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS kmatryoshka.c
20: 0000000000000000 0x82050 OBJECT  LOCAL  DEFAULT    6 parasite_blob
23: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS parasite_loader.mod.c
29: 0000000000000000   568 OBJECT  GLOBAL DEFAULT    7 __this_module
30: 000000000000002e   346 FUNC    GLOBAL DEFAULT    2 init_module
31: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND kallsyms_on_each[...]
```

```
0000000000000000 l    df *ABS*  0000000000000000 kmatryoshka.c
0000000000000000 l     O .data  0000000000082050 parasite_blob
0000000000000000 l    df *ABS*  0000000000000000 parasite_loader.mod.c
0000000000000000 g     O .gnu.linkonce.this_module      0000000000000238 __this_module
000000000000002e g     F .text  000000000000015a init_module
0000000000000000       *UND*  0000000000000000 kallsyms_on_each_symbol
```

Intepretation:

1. `init_module` present → **module entry point**. Kernel will call this on module insertion. (Confidence: High)
2. `__this_module` → module metadata; confirms the binary is intended as a Linux kernel module (LKM). (Confidence: High)
3. `parasite_blob` in `.data`, size `0x82050` (~528 KB) → large embedded payload within the module. Typical of loader-style rootkits which carry an encrypted/compressed module blob for runtime deployment. (Confidence: High)
4. Undefined symbol `kallsyms_on_each_symbol` → module intends to enumerate/resolve kernel symbols at runtime (common for rootkits seeking non-exported kernel symbols like `sys_call_table`). (Confidence: High)

11.3. Extract parasite_blob

- Write a python script to extract it

```
from elftools.elf.elffile import ELFFile
import sys
def extract_symbol(filein, sym_name, fileout):
    with open (filein,"rb") as f:
        elf = ELFFile(f)
        symtab = None
        for section in elf.iter_sections():
            if section.name in ('.symtab','.dynsym'):
                symtab = section
                break
        if not symtab:
            print('No symbol table')
            return 1
        sym = None
        for entry in symtab.iter_symbols():
            if entry.name == sym_name:
                sym = entry
                break
        if not sym:
            print('Symbol not found')
            return 2
        addr = sym['st_value']; size = sym['st_size']
        sec = None
        for section in elf.iter_sections():
            if section['sh_addr'] <= addr < section['sh_addr'] +
section['sh_size']:
                sec = section
                break
        if not sec:
            print('Containing Section not found')
            return 3
        f.seek(sec['sh_offset'] + (addr - sec['sh_addr']))
        data = f.read(size)
        open (fileout,'wb').write(data)
        print("Wrote", fileout, "size", len(data))
        return 0
if __name__=='__main__':
    if len(sys.argv)!=4:
        print("Usage: extract_symbol.py <elf> <symbol> <out>")
        sys.exit(1)
    sys.exit(extract_symbol(sys.argv[1], sys.argv[2], sys.argv[3])
```

- The size of `parasite_blob.bin` is same as the size in symbol table:

```
readelf -s
d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf |
egrep "parasite_blob"; python -c "print(int('0x82050',16))"; stat
parasite_blob.bin
    20: 0000000000000000 0x82050 OBJECT  LOCAL  DEFAULT    6 parasite_blob
  532560
  File: parasite_blob.bin
  Size: 532560
```

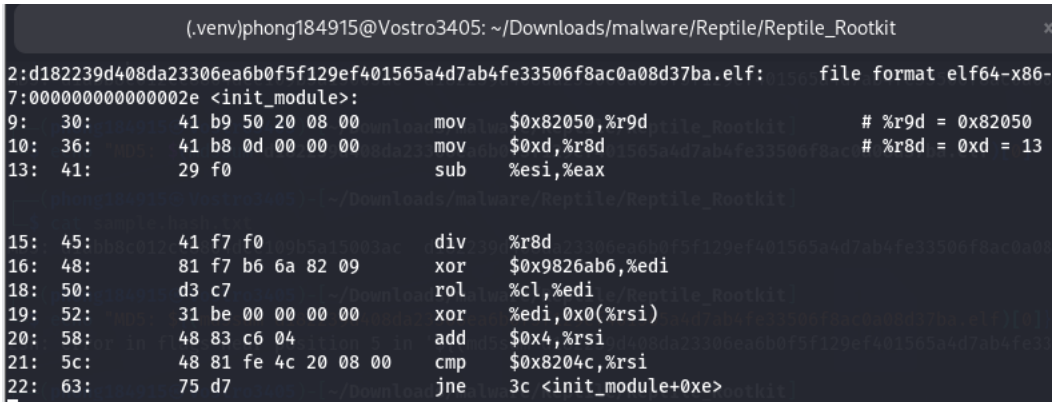
11.4. Hash & entropy of parasite_blob

- SHA256: 5604d20a6a608ccac9245c3d0f998e47a1eb92084a1f44835c7c2e0fbd2ffc8d
- Entropy ≥7.5 (7.6593) → likely encrypted/packed.
- Packer scan:

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

parasite_blob.bin: SIMH tape data
intree=Y
license=GPL
rhelversion=7.3
srcversion=9519A3D590508985462EAE4
depends=
vermagic=3.10.0-514.el7.x86_64 SMP mod_unload modversions
module_layout
kernel_stack
kallsyms_on_each_symbol
```

- `init_module` contains a decoding routine that operates in-place on parasite_blob, then resolves kernel symbols and invokes functions to install the payload — behavior typical of a rootkit loader: decrypt embedded blob.



11.5. Detailed block-by-block explaination in `init_module` section

- The `init_module` routine initializes constants (including `0x82050`, matching the embedded blob size), performs a loop of XOR+ROL transformations over a large memory region (processing 4-byte blocks), constructs ASCII symbol names on the stack, calls helper routines to allocate/verify state, resolves kernel symbols (kallsyms-style), and finally invokes a function pointer (`call *%rax`). This sequence is consistent with an **in-place decryption of an embedded payload** (`parasite_blob`), followed by runtime symbol resolution and installation of the decrypted payload into the kernel (loader-style LKM behavior).

- Confidence (static): High — multiple independent indicators (blob size constants, decrypt loop, symbol construction, kallsyms usage, indirect call) point to loader behavior. Dynamic confirmation is required to recover the decrypted payload and observe actual kernel insertion.

Block A — Parameter initialization: sets `%r9d = 0x82050` and `%r8d = 13`

```
(.venv)phong184915@Vostro3405: ~/Downloads/malware/Reptile/Reptile_Rootkit
2:d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf:     file format elf64-x86-
7:000000000000002e <init_module>:
9:   30:        41 b9 50 20 08 00       mov    $0x82050,%r9d                # %r9d = 0x82050
10:  36:        41 b8 0d 00 00 00       mov    $0xd,%r8d                    # %r8d = 0xd = 13
13:  41:        29 f0                   sub    %esi,%eax
```

⇨ `0x82050` matches the `parasite_blob` size recorded in the symbol table. The small constant `0x0d` is used later in arithmetic/division and likely participates in key derivation or loop control for deobfuscation. This strongly ties the routine to processing the embedded blob.

Block B — Deobfuscation / decryption loop: repeatedly computes a value (XOR with a constant, rotate by `%cl`) and XORs that value into memory at `%rsi`, advancing `%rsi` by 4 bytes each iteration until it reaches the upper bound (~`0x8204c`).

```
13:  41:        29 f0                   sub    %esi,%eax
15:  45:        41 f7 f0                div    %r8d
16:  48:        81 f7 b6 6a 82 09       xor    $0x9826ab6,%edi              # XOR with a constant
18:  50:        d3 c7                   rol    %cl,%edi                     # ROTATE by %cl
19:  52:        31 be 00 00 00 00       xor    %edi,0x0(%rsi)               # XOR that value into memory at %rsi
20:  58:        48 83 c6 04             add    $0x4,%rsi                    # Advancing %rsi by 4 bytes
21:  5c:        48 81 fe 4c 20 08 00    cmp    $0x8204c,%rsi                # each iteration until it reaches the upper bound (0x8204c)
22:  63:        75 d7                   jne    3c <init_module+0xe>
```

⇨ This is a classic in-place decryption/deobfuscation loop operating on 4-byte blocks. The loop bounds and the earlier `mov $0x82050` indicate the code decrypts the entire embedded payload region (`parasite_blob`). The use of XOR+ROL is a simple but effective obfuscation/cryptographic primitive often used to hide payload bytes in static data.

Block C — Build symbol strings on stack: writes ASCII words to memory, assembling a recognizable symbol string

```
35:  90:        c7 00 73 79 73 5f       movl   $0x5f737973,(%rax)
36:  96:        c7 40 04 69 6e 69 74    movl   $0x74696e69,0x4(%rax)
37:  9d:        c7 40 08 5f 6d 6f 64    movl   $0x646f6d5f,0x8(%rax)
38:  a4:        c7 40 0c 75 6c 65 00    movl   $0x656c75,0xc(%rax)
```

```
>>> hex_array=[0x5f737973,0x74696e69,0x646f6d5f,0x656c75]
>>> for hex in hex_array:
...     print(hex.to_bytes(4, byteorder='little').decode('ascii',errors='replace'), end='')
... print()
...
sys_init_module
```

⇨ Constructing symbol names at runtime suggests the code will perform a name-based lookup (kallsyms) to resolve kernel addresses that are not exported. This technique is commonly used by kernel loaders or rootkits to obtain addresses of non-exported kernel routines.

- **Block D — Call helpers and check return values:** invokes helper functions and checks return values to decide whether to proceed.

```
39:  ab:        48 89 45 e0             mov    %rax,-0x20(%rbp)
40:  af:        e8 00 00 00 00          call   b4 <init_module+0x86>                  # Call helper function 1
41:  b4:        48 8b 45 e8             mov    -0x18(%rbp),%rax
42:  b8:        48 85 c0                test   %rax,%rax
43:  bb:        75 5b                   jne    118 <init_module+0xea>                 # If %rax != 0, jump to address 118
44:  bd:        48 83 ec 28             sub    $0x28,%rsp
45:  c1:        48 8d 75 e0             lea    -0x20(%rbp),%rsi
46:  c5:        48 c7 c7 00 00 00 00    mov    $0x0,%rdi
47:  cc:        48 8d 44 24 0f          lea    0xf(%rsp),%rax
48:  d1:        48 c7 45 e8 00 00 00    movq   $0x0,-0x18(%rbp)
49:  d8:        00
50:  d9:        48 83 e0 f0             and    $0xfffffffffffffff0,%rax
51:  dd:        c7 00 5f 5f 64 6f       movl   $0x6f645f5f,(%rax)
52:  e3:        c7 40 04 5f 73 79 73    movl   $0x7379735f,0x4(%rax)
53:  ea:        c7 40 08 5f 69 6e 69    movl   $0x696e695f,0x8(%rax)
54:  f1:        c7 40 0c 74 5f 6d 6f    movl   $0x6f6d5f74,0xc(%rax)
55:  f8:        c7 40 10 64 75 6c 65    movl   $0x656c7564,0x10(%rax)
56:  ff:        c7 40 14 00 00 00 00    movl   $0x0,0x14(%rax)
57: 106:        48 89 45 e0             mov    %rax,-0x20(%rbp)
58: 10a:        e8 00 00 00 00          call   10f <init_module+0xe1>                 # Call helper function 2
59: 10f:        48 8b 45 e8             mov    -0x18(%rbp),%rax
60: 113:        48 85 c0                test   %rax,%rax
61: 116:        74 62                   je     17a <init_module+0x14c>                # If %rax == 0, jump to address 17a
```

  ⇨ Helpers likely perform allocation, verify prerequisites, or prepare kernel memory. The conditional checks indicate robust error handling before payload installation.

- **Block E — Read per-CPU/kernel data:** reads from a GS-based[3] per-CPU structure and loads a value at a negative offset.

```
62: 118:        65 48 8b 14 25 00 00    mov    %gs:0x0,%rdx                           # MOVE value of offset 0 in GS-based per-CPU data area to %rdx
63: 11f:        00 00
64: 121:        4c 8b a2 48 c0 ff ff    mov    -0x3fb8(%rdx),%r12                     # Address in %rdx - 0x3fb8, load into %r12
```

  ⇨ Accessing GS and per-CPU data implies kernel-context operations and interaction with kernel internal structures. It is consistent with kernel-mode code adjusting kernel state or recording loader state into kernel data.

- **Block F — Alignment, write, and indirect call:** re-sets blob size, aligns an address to a page boundary, stores it to a kernel structure, then performs an indirect call through `%rax`.

```
70: 139:        48 c7 c1 00 00 00 00    mov    $0x0,%rcx
71: 140:        be 50 20 08 00          mov    $0x82050,%esi                          # Rewrite the size of parasite_blob into %esi
72: 145:        48 c7 c7 00 00 00 00    mov    $0x0,%rdi
73: 14c:        48 81 e1 00 f0 ff ff    and    $0xfffffffffffff000,%rcx               # Page-alignment, *%rcx & 0xfffffffffffff000 (mask)
74: 153:        65 48 8b 1c 25 00 00    mov    %gs:0x0,%rbx
75: 15a:        00 00
76: 15c:        48 89 8b 48 c0 ff ff    mov    %rcx,-0x3fb8(%rbx)
77: 163:        ff d0                   call   *%rax
```

  ⇨ The indirect call (`call *%rax`) is the critical step where a previously resolved function pointer is invoked. Given prior symbol construction and kallsyms usage, `%rax` likely points to a kernel routine (for example `finit_module` or a resolved install function) used to install or initialize the decrypted payload. The alignment suggests safe memory placement before handing control to kernel code.

- **Block G — Result handling and return:** checks the return from the indirect call, updates state, and returns an error code if necessary.

[3] In Linux x86_64, the GS register points to per-CPU memory. The code's mov %gs:0x0 is dereferencing this region, a common kernel-mode technique for reading CPU-local kernel state.

```
78: 165:        48 83 f8 01             cmp    $0x1,%rax
79: 169:        4c 89 a3 48 c0 ff ff    mov    %r12,-0x3fb8(%rbx)
80: 170:        19 c0                   sbb    %eax,%eax
81: 172:        83 e0 f1                and    $0xfffffff1,%eax
82: 175:        83 e8 16                sub    $0x16,%eax
83: 178:        eb 05                   jmp    17f <init_module+0x151>
84: 17a:        b8 ea ff ff ff          mov    $0xffffffea,%eax
85: 17f:        48 8d 65 f0             lea    -0x10(%rbp),%rsp
```

⇨ The code properly handles success/failure of the invoked kernel routine and cleans up, indicating expected lifecycle behavior for a kernel module init function.