OCTOBER 23, 2025

STATIC ANALYSIS REPORT

REPTILE ROOTKIT

NGUYỄN HOÀNG THANH PHONG – SE184915
FPT University

CONTENTS

1 EXECUTIVE SUMMARY

This report continues the forensic workflow initiated in Report1 (static triage) and Report2 (dynamic validation) for the *Reptile rootkit*.
Unlike the first static sample (`d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba.elf`), the current analysis targets the **newly compiled kernel module** `reptile.ko` — a locally built variant from the public Reptile source code.

The substitution was necessary because the original sample failed kernel module verification (`parasite_loader: module verification failed: signature and/or required key missing - tainting kernel`) and could not load under the CentOS 7 analysis kernel. The compiled sample was therefore introduced to ensure controlled execution and reproducible results, as demonstrated in Report2.

This document statically re-analyzes the new sample to identify its internal structure, code segments, obfuscation mechanisms, and embedded payloads prior to dynamic runtime decryption.

2 OBJECTIVES & SCOPE

2.1 OBJECTIVES:

- Examine the compiled Reptile rootkit module (`reptile.ko`) at rest (without execution).
- Identify imported/exported kernel symbols, ELF sections, and decryption routines.
- Compare static artifacts with those from Report1 to understand codebase changes.
- Locate and classify embedded data blobs or encrypted payloads.

2.2 SCOPE AND LIMITATIONS:

- Analysis limited to ELF-level disassembly, symbol, and string inspection.
- Dynamic behaviors (C2, persistence, etc.) already confirmed in Report2 — not revalidated here.
- Focus: code-level signatures, symbol table anomalies, and hidden payload embedding.

3 SAMPLE METADATA

Attribute	Value
File name	`reptile.ko`
File size	555,800 bytes
File type	ELF 64-bit LSB relocatable, x86-64, SYSV ABI, not stripped
Build ID	46b52a008da7f78ac6060e9a4e3f6020fed071a0
Hash (MD5)	44aa3745cf62f068218a9b4d0fc21e4e
Hash (SHA256)	9573f414bedbd78464c9d9e4c214c37aea9a1b25dd3a6b4b512e65bbec9f9d51
Vermagic	3.10.0-1160.119.1.el7.x86_64 SMP mod_unload modversions
Compiler	GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)
Compilation date	22-10-2025
Original source	Reptile public GitHub repository (2018–2019 branch)
Author	F0rb1dd3n - ighor@intruder-security.com
Description	"Reptile - A linux LKM rootkit"

4 TOOLS AND ENVIRONMENT

Tool	Version	Purpose
readelf, objdump	binutils 2.27	ELF header, section, relocation inspection
strings, xxd	coreutils 8.22	Static string and hex dump
IDA Free	Version 9.2.250908 Windows x64 (64-bit address size)	Disassembly and decompilation
hashdeep	4.4	Hash verification
binwalk	2.3.3	Embedded data extraction
file, hexdump	coreutils	Format identification
Host system	Windows 11 Home Single Language	

5 FILE STRUCTURE OVERVIEW

ELF Section map:

Section name	Type	Size (bytes)	Notes
.text	PROGBITS	4,488	Main code (non-init).
.rela.text	RELA	4,776	Relocations for .text.
.altinstr_replacement	PROGBITS	4,600	Alternative instruction replacement area.
.init.text	PROGBITS	4,630	Initialization code (init_module related).
.rela.init.text	RELA	984	Relocations for .init.text.
.exit.text	PROGBITS	5,101	Exit/cleanup code (cleanup_module).
.rela.exit.text	RELA	1,056	Relocations for .exit.text.
.rodata.str1.1	PROGBITS	110	Read-only constant strings (short).
.smp_locks	PROGBITS	4	SMP lock data.
.parainstructions	PROGBITS	412	Para-instruction data (arch-specific).
.altinstructions	PROGBITS	72	Alternative instruction templates.
.rodata.str1.8	PROGBITS	6,152	Read-only data (longer strings).
.modinfo	PROGBITS	239	Module metadata (description/author/etc.).
.return_sites	PROGBITS	136	Return site table (CFI/retpoline support).
__mcount_loc	PROGBITS	192	Profiling/linking helpers.
__versions	PROGBITS	2,816	Versioning info for kernel symbols.
.data	PROGBITS	20	Small initialized data.
.gnu.linkonce.this_module	PROGBITS	568	Linkonce module-specific data.
.bss	NOBITS	176	Zero-initialized data size (bss).
.debug_info	PROGBITS	153,598	Substantial debug information (DWARF).

.debug_abbrev	PROGBITS	2,930	DWARF abbrev section.
.debug_loc	PROGBITS	15,220	DWARF location lists.
.debug_line	PROGBITS	7,373	DWARF line number info.
.debug_str	PROGBITS	96,261	Large debug strings section.
.comment	PROGBITS	92	Compiler comment (often GCC version).
.debug_frame	PROGBITS	1,408	DWARF frame info.
.symtab	SYMTAB	3,000	Symbol table (static).
.strtab	STRTAB	1,347	String table for symbols.
.shstrtab	STRTAB	468	Section name string table.

****Observation****

- The module contains substantial DWARF debug information (`.debug_*` sections). This indicates the binary was **compiled with debug info enabled** (useful for analysis and symbolic names).
- The `.init.text` and `.exit.text` sections are non-trivial in size, which is consistent with a loader-style module that performs meaningful initialization and cleanup work (e.g., decryption loops, symbol resolution, hooking).
- There is **no obvious large, contiguous static data section** labeled `parasite_blob` in this compiled variant (unlike the older internet-sourced sample). This supports the dynamic-analysis finding that the payload is produced (decrypted or reconstructed) at runtime rather than stored as a single giant static blob in the module.

6 STATIC INDICATORS & STRING ANALYSIS

A focused string extraction and inspection of `reptile.ko` reveals a number of high-signal markers that tie the compiled module to the publicly-known Reptile family and to the artifacts described in "*Report1*". Notable strings discovered in `.rodata` and symbol tables include: *(IDA Free 9.2)*

```
.rodata.str1.1:000000000000153F  00000008   C     reptile
.rodata.str1.1:0000000000001547  0000000B   C     #<reptile>
.rodata.str1.1:0000000000001552  0000000C   C     #</reptile>
.rodata.str1.1:0000000000001560  00000017   C
     /reptile/reptile_shell
.rodata.str1.1:000000000000157B  0000001A   C
     /reptile/reptile_start.sh
.rodata.str1.1:0000000000001595  0000000F   C     /reptile_shell
.rodata.str1.8:0000000000001798  00000023   C
     PATH=/sbin:/bin:/usr/sbin:/usr/bin
.modinfo:00000000000017BB  0000002A   C     description=Reptile - A
linux LKM rootkit
.modinfo:00000000000017E5  0000002F   C     author=F0rb1dd3n -
ighor@intruder-security.com
.modinfo:0000000000001814  0000000C   C     license=GPL
.modinfo:0000000000001820  0000000C   C     retpoline=Y
.modinfo:000000000000182C  00000010   C     rhelversion=7.9
.modinfo:000000000000183C  00000023
     srcversion=B799CC57720B553D4EFD535
.modinfo:000000000000185F  00000009   C     depends=
.modinfo:0000000000001868  00000042   C     vermagic=3.10.0-
1160.119.1.el7.x86_64 SMP mod_unload modversions
.gnu.linkonce.this_module:0000000000002538  00000008   C     rep_mod
```

Readable strings:

```
reptile
#<reptile>
#</reptile>
/reptile/reptile_shell
/reptile/reptile_start.sh
/reptile_shell
description=Reptile - A linux LKM rootkit
/analysis/f0rb1dd3n-Reptile_-_2018-05-19_20-00-44/
/analysis/f0rb1dd3n-Reptile_-_2018-05-19_20-00-44/
/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
reptile_init
reptile_exit
```

These tokens indicate explicit naming and intent (loader, shell, start script) and provide direct linkage to the source tree used to build the compiled module. Importantly, while Report1's original internet-sourced sample exposed a statically embedded data blob named `parasite_blob` (~532560 bytes) visible in the ELF symbol table, the compiled variant analyzed here **does not** present a similarly named large static data section. Instead, the compiled module contains frequent textual references to the runtime shell and loader components, consistent with a loader-style module that reconstructs or decrypts the payload at runtime.

7 SYMBOL TABLE & FUNCTION ANALYSIS

Analysis of symbol tables (static `symtab` / `strtab`) and exported symbols shows clear initialization and cleanup functions typical of kernel modules, as well as identifiable function names associated with loader behavior.

- Exported / visible symbols of interest: `reptile_init`, `reptile_exit`, `rep_mod_init`, `rep_mod_cleanup`, `__versions` entries mapping to kernel symbol dependencies.
- The module also references kernel helper routines such as `memcpy`, `memmove`, `memset` and list / allocation helpers, which are used in typical decryption and payload relocation code.
- Static relocation tables (`.rela.*`) are populated for `.text`, `.init.text` and `.exit.text` which is consistent with runtime symbol resolution and position-independent code patterns.

Evidence:

```
└──# nm reptile.ko | egrep "\bT\b"
0000000000000c60 T atoi
0000000000000000 T cleanup_module
0000000000000ca0 T decode_n_spawn
00000000000008a0 T e_fget_light
00000000000007d0 T exec
0000000000000970 T f_check
0000000000001030 T find_sys_call_table
0000000000000740 T find_task
0000000000001140 T generic_find_sys_call_table
0000000000000560 T hide
0000000000000a30 T hide_content
00000000000010d0 T ia32_find_sys_call_table
0000000000000000 T init_module
00000000000007b0 T is_invisible
0000000000000390 T l33t_getdents
00000000000001c0 T l33t_getdents64
0000000000000610 T l33t_kill
0000000000000b40 T l33t_read
00000000000000e0 T l33t_setreuid
0000000000000e40 T magic_packet_hook
0000000000000fc0 T memmem
0000000000000000 T shell_execer
0000000000000800 T shell_exec_queue
00000000000005c0 T show
0000000000000c30 T s_xor
```

8 CODE PATTERNS & DEOBFUSCATION ROUTINES

Static disassembly and light decompilation of `.init.text` indicate the presence of non-trivial initialization code consistent with a decryption loop and dynamic resolution of kernel symbols:

- The `.init.text` section exhibits a loop structure that, when compared with Report1's static findings, matches the signature of an XOR + ROL style decryption routine (size and control-flow patterns align with the pre-dynamic hypothesis).
- `.exit.text` contains cleanup paths that appear to undo or remove hooks and to revoke allocated resources — typical for a loader that attempts to conceal or restore state upon removal.
- Several relocation and `kallsyms` -style resolution patterns (string construction and `kallsyms_on_each_symbol` -like callbacks) are present as inferred from relocation entries and identified call-sites; these are consistent with a module that resolves non-exported kernel symbols at runtime.

Actionable artifact:

```
.text:0000000000000C30 s_xor:
     ; DATA XREF: __mcount_loc:00000000000019B8↓o
.text:0000000000000C30                 call    __fentry__
.text:0000000000000C35                 push    rbp
.text:0000000000000C36                 test    edx, edx
.text:0000000000000C38                 mov     rbp, rsp
.text:0000000000000C3B                 jle     short loc_C54
.text:0000000000000C3D                 lea     eax, [rdx-1]
.text:0000000000000C40                 lea     rax, [rdi+rax+1]
.text:0000000000000C45                 nop     dword ptr [rax]
.text:0000000000000C48
.text:0000000000000C48 loc_C48:
     ; CODE XREF: .text:0000000000000C52↓j
.text:0000000000000C48                 xor     [rdi], sil
.text:0000000000000C4B                 add     rdi, 1
.text:0000000000000C4F                 cmp     rdi, rax
.text:0000000000000C52                 jnz     short loc_C48
.text:0000000000000C54
.text:0000000000000C54 loc_C54:
     ; CODE XREF: .text:0000000000000C3B↑j
.text:0000000000000C54                 pop     rbp
.text:0000000000000C55                 jmp     __x86_return_thunk
;;=============================================================
.init.text:00000000000011A6 _init_text      segment byte public
'CODE' use64
.init.text:00000000000011A6                 assume cs:_init_text
.init.text:00000000000011A6                 ;org 11A6h
```

```
.init.text:00000000000011A6                 assume es:nothing,
ss:nothing, ds:_data, fs:nothing, gs:nothing
.init.text:00000000000011A6
.init.text:00000000000011A6 ; =============== S U B R O U T I N E
=======================================
.init.text:00000000000011A6
.init.text:00000000000011A6 ; Alternative name is 'init_module'
.init.text:00000000000011A6 ; Attributes: static bp-based frame
.init.text:00000000000011A6
.init.text:00000000000011A6 ; int __cdecl reptile_init()
.init.text:00000000000011A6                 public reptile_init
.init.text:00000000000011A6 reptile_init    proc
near               ; DATA XREF: .return_sites:000000000000192A↓o
.init.text:00000000000011A6
; .gnu.linkonce.this_module:0000000000002648↓o
.init.text:00000000000011A6
.init.text:00000000000011A6 argv            = qword ptr -20h
.init.text:00000000000011A6 var_8           = qword ptr -8
.init.text:00000000000011A6
.init.text:00000000000011A6                 push    rbp
.init.text:00000000000011A7                 mov     rbp, rsp
.init.text:00000000000011AA                 sub     rsp, 20h
.init.text:00000000000011AE                 mov
cs:read_on.counter, 0
.init.text:00000000000011B8                 mov     rax, gs:28h
.init.text:00000000000011C1                 mov     [rbp+var_8], rax
.init.text:00000000000011C5                 xor     eax, eax
.init.text:00000000000011C7                 mov     [rbp+argv],
offset aReptileReptile_0 ; "/reptile/reptile_start.sh"
.init.text:00000000000011CF                 mov     [rbp+argv+8], 0
.init.text:00000000000011D7                 mov     [rbp+argv+10h],
0
.init.text:00000000000011DF                 call
find_sys_call_table
.init.text:00000000000011E4                 test    rax, rax
.init.text:00000000000011E7                 mov     cs:sct, rax
.init.text:00000000000011EE                 jnz     short loc_11FC
.init.text:00000000000011F0                 call
ia32_find_sys_call_table
.init.text:00000000000011F5                 mov     cs:sct, rax
.init.text:00000000000011FC
.init.text:00000000000011FC
loc_11FC:                               ; CODE XREF: reptile_init+48
↑j
.init.text:00000000000011FC                 cmp     cs:sct, 0
.init.text:0000000000001204                 mov     rax,
cs:page_offset_base
.init.text:000000000000120B                 jnz     short loc_122C
```

```
.init.text:000000000000120D
.init.text:000000000000120D
loc_120D:                               ; CODE XREF: reptile_init+7B
↓j
.init.text:000000000000120D                 cmp     rax,
0FFFFFFFFFFFFFFFFh
.init.text:0000000000001211                 jz      short loc_1223
.init.text:0000000000001213                 cmp     qword ptr
[rax+18h], offset sys_close
.init.text:000000000000121B                 jz      short loc_1225
.init.text:000000000000121D                 add     rax, 8
.init.text:0000000000001221                 jmp     short loc_120D
.init.text:0000000000001223 ; ---------------------------------------
-----------------------------------
.init.text:0000000000001223
.init.text:0000000000001223
loc_1223:                               ; CODE XREF: reptile_init+6B
↑j
.init.text:0000000000001223                 xor     eax, eax
.init.text:0000000000001225
.init.text:0000000000001225
loc_1225:                               ; CODE XREF: reptile_init+75
↑j
.init.text:0000000000001225                 mov     cs:sct, rax
.init.text:000000000000122C
.init.text:000000000000122C
loc_122C:                               ; CODE XREF: reptile_init+65
↑j
.init.text:000000000000122C                 mov     rax, cs:sct
.init.text:0000000000001233                 test    rax, rax
.init.text:0000000000001236                 jz      loc_1360
.init.text:000000000000123C                 mov     rdx, [rax+388h]
.init.text:0000000000001243                 mov     cs:o_setreuid,
rdx
.init.text:000000000000124A                 mov     rdx, [rax+1F0h]
.init.text:0000000000001251                 mov     cs:o_kill, rdx
.init.text:0000000000001258                 mov     rdx, [rax+6C8h]
.init.text:000000000000125F                 mov     cs:o_getdents64,
rdx
.init.text:0000000000001266                 mov     rdx, [rax+270h]
.init.text:000000000000126D                 mov     cs:o_getdents,
rdx
.init.text:0000000000001274                 mov     rax, [rax]
.init.text:0000000000001277                 mov     cs:o_read, rax
.init.text:000000000000127E
```

```
.init.text:000000000000127E
loc_127E:                               ; DATA
XREF: .parainstructions:00000000000015D0↓o
.init.text:000000000000127E                 call
ds:(pv_cpu_ops_0+18h)
.init.text:0000000000001285                 and     rax,
0FFFFFFFFFFFEFFFFh
.init.text:000000000000128B                 mov     rdi, rax
.init.text:000000000000128E
.init.text:000000000000128E
loc_128E:                               ; DATA
XREF: .parainstructions:00000000000015E0↓o
.init.text:000000000000128E                 call
ds:(pv_cpu_ops_0+20h)
.init.text:0000000000001295                 mov     rax, cs:sct
.init.text:000000000000129C                 mov     qword ptr
[rax+388h], offset l33t_setreuid
.init.text:00000000000012A7                 mov     qword ptr
[rax+1F0h], offset l33t_kill
.init.text:00000000000012B2                 mov     qword ptr
[rax+6C8h], offset l33t_getdents64
.init.text:00000000000012BD                 mov     qword ptr
[rax+270h], offset l33t_getdents
.init.text:00000000000012C8                 mov     qword ptr [rax],
offset l33t_read
.init.text:00000000000012CF
.init.text:00000000000012CF
loc_12CF:                               ; DATA
XREF: .parainstructions:00000000000015F0↓o
.init.text:00000000000012CF                 call
ds:(pv_cpu_ops_0+18h)
.init.text:00000000000012D6                 or      rax, 10000h
.init.text:00000000000012DC                 mov     rdi, rax
.init.text:00000000000012DF
.init.text:00000000000012DF
loc_12DF:                               ; DATA
XREF: .parainstructions:0000000000001600↓o
.init.text:00000000000012DF                 call
ds:(pv_cpu_ops_0+20h)
.init.text:00000000000012E6                 mov     rdi, offset
magic_packet_hook_options
.init.text:00000000000012ED                 mov
cs:magic_packet_hook_options.hook, offset magic_packet_hook
.init.text:00000000000012F8                 mov
cs:magic_packet_hook_options.hooknum, 0
.init.text:0000000000001302                 mov
cs:magic_packet_hook_options.pf, 2
```

```
.init.text:0000000000001309                 mov
cs:magic_packet_hook_options.priority, 80000000h
.init.text:0000000000001313                 call    nf_register_hook
.init.text:0000000000001318                 mov     r9, offset
aReptileShell ; "/reptile_shell"
.init.text:000000000000131F                 xor     r8d, r8d
.init.text:0000000000001322                 xor     ecx, ecx
.init.text:0000000000001324                 mov     edx, 1
.init.text:0000000000001329                 mov     esi, 8
.init.text:000000000000132E                 mov     rdi, offset
aS  ; "%s"
.init.text:0000000000001335                 xor     eax, eax
.init.text:0000000000001337                 call
__alloc_workqueue_key
.init.text:000000000000133C                 mov     rdi, [rbp+argv]
.init.text:0000000000001340                 lea     rsi, [rbp+argv]
.init.text:0000000000001344                 mov     ecx, 2
.init.text:0000000000001349                 mov     rdx, offset
envp_61383
.init.text:0000000000001350                 mov     cs:work_queue,
rax
.init.text:0000000000001357                 call
call_usermodehelper
.init.text:000000000000135C                 xor     eax, eax
.init.text:000000000000135E                 jmp     short loc_1363
.init.text:0000000000001360 ; ---------------------------------------
-----------------------------------
.init.text:0000000000001360
.init.text:0000000000001360
loc_1360:                               ; CODE XREF: reptile_init+90
↑j
.init.text:0000000000001360                 or      eax, 0FFFFFFFFh
.init.text:0000000000001363
.init.text:0000000000001363
loc_1363:                               ; CODE XREF:
reptile_init+1B8↑j
.init.text:0000000000001363                 mov     rcx, [rbp+var_8]
.init.text:0000000000001367                 xor     rcx, gs:28h
.init.text:0000000000001370                 jz      short
locret_1377
.init.text:0000000000001372                 call    __stack_chk_fail
.init.text:0000000000001377 ; ---------------------------------------
-----------------------------------
.init.text:0000000000001377
.init.text:0000000000001377
locret_1377:                            ; CODE XREF:
reptile_init+1CA↑j
.init.text:0000000000001377                 leave
```

```
.init.text:0000000000001378                 jmp
__x86_return_thunk
.init.text:0000000000001378 reptile_init    endp
.init.text:0000000000001378
.init.text:0000000000001378 _init_text      ends
;;=============================================
exit.text:000000000000137D _exit_text      segment byte public
'CODE' use64
.exit.text:000000000000137D                 assume cs:_exit_text
.exit.text:000000000000137D                 ;org 137Dh
.exit.text:000000000000137D                 assume es:nothing,
ss:nothing, ds:_data, fs:nothing, gs:nothing
.exit.text:000000000000137D
.exit.text:000000000000137D ; =============== S U B R O U T I N E
=======================================
.exit.text:000000000000137D
.exit.text:000000000000137D ; Alternative name is 'cleanup_module'
.exit.text:000000000000137D ; Attributes: static bp-based frame
.exit.text:000000000000137D
.exit.text:000000000000137D ; void __cdecl reptile_exit()
.exit.text:000000000000137D                 public reptile_exit
.exit.text:000000000000137D reptile_exit    proc
near               ; DATA XREF: .return_sites:000000000000192E↓o
.exit.text:000000000000137D
; .gnu.linkonce.this_module:0000000000002748↓o
.exit.text:000000000000137D                 push    rbp
.exit.text:000000000000137E                 cmp     cs:o_setreuid, 0
.exit.text:0000000000001386                 mov     rbp, rsp
.exit.text:0000000000001389                 jz      short loc_13CE
.exit.text:000000000000138B
.exit.text:000000000000138B
loc_138B:                               ; DATA
XREF: .parainstructions:0000000000001610↓o
.exit.text:000000000000138B                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:0000000000001392                 and     rax,
0FFFFFFFFFFFEFFFFh
.exit.text:0000000000001398                 mov     rdi, rax
.exit.text:000000000000139B
.exit.text:000000000000139B
loc_139B:                               ; DATA
XREF: .parainstructions:0000000000001620↓o
.exit.text:000000000000139B                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:00000000000013A2                 mov     rdx,
cs:o_setreuid
.exit.text:00000000000013A9                 mov     rax, cs:sct
```

```
.exit.text:00000000000013B0                 mov     [rax+388h], rdx
.exit.text:00000000000013B7
.exit.text:00000000000013B7
loc_13B7:                               ; DATA
XREF: .parainstructions:0000000000001630↓o
.exit.text:00000000000013B7                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:00000000000013BE                 or      rax, 10000h
.exit.text:00000000000013C4                 mov     rdi, rax
.exit.text:00000000000013C7
.exit.text:00000000000013C7
loc_13C7:                               ; DATA
XREF: .parainstructions:0000000000001640↓o
.exit.text:00000000000013C7                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:00000000000013CE
.exit.text:00000000000013CE
loc_13CE:                               ; CODE XREF: reptile_exit+C
↑j
.exit.text:00000000000013CE                 cmp     cs:o_kill, 0
.exit.text:00000000000013D6                 jz      short loc_141B
.exit.text:00000000000013D8
.exit.text:00000000000013D8
loc_13D8:                               ; DATA
XREF: .parainstructions:0000000000001650↓o
.exit.text:00000000000013D8                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:00000000000013DF                 and     rax,
0FFFFFFFFFFEFFFFh
.exit.text:00000000000013E5                 mov     rdi, rax
.exit.text:00000000000013E8
.exit.text:00000000000013E8
loc_13E8:                               ; DATA
XREF: .parainstructions:0000000000001660↓o
.exit.text:00000000000013E8                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:00000000000013EF                 mov     rdx, cs:o_kill
.exit.text:00000000000013F6                 mov     rax, cs:sct
.exit.text:00000000000013FD                 mov     [rax+1F0h], rdx
.exit.text:0000000000001404
.exit.text:0000000000001404
loc_1404:                               ; DATA
XREF: .parainstructions:0000000000001670↓o
.exit.text:0000000000001404                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:000000000000140B                 or      rax, 10000h
.exit.text:0000000000001411                 mov     rdi, rax
```

```
.exit.text:0000000000001414
.exit.text:0000000000001414
loc_1414:                               ; DATA
XREF: .parainstructions:0000000000001680↓o
.exit.text:0000000000001414                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:000000000000141B
.exit.text:000000000000141B
loc_141B:                               ; CODE XREF: reptile_exit+59
↑j
.exit.text:000000000000141B                 cmp     cs:o_getdents64,
0
.exit.text:0000000000001423                 jz      short loc_1468
.exit.text:0000000000001425
.exit.text:0000000000001425
loc_1425:                               ; DATA
XREF: .parainstructions:0000000000001690↓o
.exit.text:0000000000001425                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:000000000000142C                 and     rax,
0FFFFFFFFFFEFFFFh
.exit.text:0000000000001432                 mov     rdi, rax
.exit.text:0000000000001435
.exit.text:0000000000001435
loc_1435:                               ; DATA
XREF: .parainstructions:00000000000016A0↓o
.exit.text:0000000000001435                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:000000000000143C                 mov     rdx,
cs:o_getdents64
.exit.text:0000000000001443                 mov     rax, cs:sct
.exit.text:000000000000144A                 mov     [rax+6C8h], rdx
.exit.text:0000000000001451
.exit.text:0000000000001451
loc_1451:                               ; DATA
XREF: .parainstructions:00000000000016B0↓o
.exit.text:0000000000001451                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:0000000000001458                 or      rax, 10000h
.exit.text:000000000000145E                 mov     rdi, rax
.exit.text:0000000000001461
.exit.text:0000000000001461
loc_1461:                               ; DATA
XREF: .parainstructions:00000000000016C0↓o
.exit.text:0000000000001461                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:0000000000001468
```

```
.exit.text:0000000000001468
loc_1468:                               ; CODE XREF: reptile_exit+A6
↑j
.exit.text:0000000000001468                 cmp     cs:o_getdents, 0
.exit.text:0000000000001470                 jz      short loc_14B5
.exit.text:0000000000001472
.exit.text:0000000000001472
loc_1472:                               ; DATA
XREF: .parainstructions:00000000000016D0↓o
.exit.text:0000000000001472                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:0000000000001479                 and     rax,
0FFFFFFFFFFFEFFFFh
.exit.text:000000000000147F                 mov     rdi, rax
.exit.text:0000000000001482
.exit.text:0000000000001482
loc_1482:                               ; DATA
XREF: .parainstructions:00000000000016E0↓o
.exit.text:0000000000001482                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:0000000000001489                 mov     rdx,
cs:o_getdents
.exit.text:0000000000001490                 mov     rax, cs:sct
.exit.text:0000000000001497                 mov     [rax+270h], rdx
.exit.text:000000000000149E
.exit.text:000000000000149E
loc_149E:                               ; DATA
XREF: .parainstructions:00000000000016F0↓o
.exit.text:000000000000149E                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:00000000000014A5                 or      rax, 10000h
.exit.text:00000000000014AB                 mov     rdi, rax
.exit.text:00000000000014AE
.exit.text:00000000000014AE
loc_14AE:                               ; DATA
XREF: .parainstructions:0000000000001700↓o
.exit.text:00000000000014AE                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:00000000000014B5
.exit.text:00000000000014B5
loc_14B5:                               ; CODE XREF: reptile_exit+F3
↑j
.exit.text:00000000000014B5                 cmp     cs:o_read, 0
.exit.text:00000000000014BD                 jz      short loc_150F
.exit.text:00000000000014BF
```

```
.exit.text:00000000000014BF
loc_14BF:                               ; CODE XREF:
reptile_exit+151↓j
.exit.text:00000000000014BF                 mov     eax,
cs:read_on.counter
.exit.text:00000000000014C5                 test    eax, eax
.exit.text:00000000000014C7                 jz      short loc_14D0
.exit.text:00000000000014C9                 call    schedule
.exit.text:00000000000014CE                 jmp     short loc_14BF
.exit.text:00000000000014D0 ; ---------------------------------------
-----------------------------------
.exit.text:00000000000014D0
.exit.text:00000000000014D0
loc_14D0:                               ; CODE XREF:
reptile_exit+14A↑j
.exit.text:00000000000014D0
; DATA XREF: .parainstructions:0000000000001710↓o
.exit.text:00000000000014D0                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:00000000000014D7                 and     rax,
0FFFFFFFFFFFEFFFFh
.exit.text:00000000000014DD                 mov     rdi, rax
.exit.text:00000000000014E0
.exit.text:00000000000014E0
loc_14E0:                               ; DATA
XREF: .parainstructions:0000000000001720↓o
.exit.text:00000000000014E0                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:00000000000014E7                 mov     rdx, cs:o_read
.exit.text:00000000000014EE                 mov     rax, cs:sct
.exit.text:00000000000014F5                 mov     [rax], rdx
.exit.text:00000000000014F8
.exit.text:00000000000014F8
loc_14F8:                               ; DATA
XREF: .parainstructions:0000000000001730↓o
.exit.text:00000000000014F8                 call
ds:(pv_cpu_ops_0+18h)
.exit.text:00000000000014FF                 or      rax, 10000h
.exit.text:0000000000001505                 mov     rdi, rax
.exit.text:0000000000001508
.exit.text:0000000000001508
loc_1508:                               ; DATA
XREF: .parainstructions:0000000000001740↓o
.exit.text:0000000000001508                 call
ds:(pv_cpu_ops_0+20h)
.exit.text:000000000000150F
```

```
.exit.text:000000000000150F
loc_150F:                               ; CODE XREF:
reptile_exit+140↑j
.exit.text:000000000000150F                 mov     rdi, offset
magic_packet_hook_options
.exit.text:0000000000001516                 call
nf_unregister_hook
.exit.text:000000000000151B                 mov     rdi,
cs:work_queue
.exit.text:0000000000001522                 call    flush_workqueue
.exit.text:0000000000001527                 mov     rdi,
cs:work_queue
.exit.text:000000000000152E                 call
destroy_workqueue
.exit.text:0000000000001533                 pop     rbp
.exit.text:0000000000001534                 jmp
__x86_return_thunk
.exit.text:0000000000001534 reptile_exit    endp
.exit.text:0000000000001534
.exit.text:0000000000001534 _exit_text      ends
```

1) The small routine beginning at `.text:0000000000000C30` is a byte-wise XOR loop. Key observations:
   - Prologue shows standard frame setup and a length test: `test edx, edx` → if `edx` (length) ≤ 0 then return.
   - Parameter usage (x86_64 SysV): `rdi` = destination pointer, `rdx` = length, `rsi`/`sil` (lower 8 bits) used as key byte.
     Evidence: `xor [rdi], sil` uses the `sil` register as the XOR key; `add rdi, 1` increments the destination pointer; the loop compares `rdi` against `rax` (computed end pointer) and repeats with `jnz`.
   - Loop structure: compute end pointer (`lea eax, [rdx-1]` then `lea rax, [rdi+rax+1]`), then loop over bytes doing `xor [rdi], sil` until `rdi == rax`.
   - High-level semantics: `for i in 0..len-1: dest[i] ^= key_byte`. This is a single-byte XOR transformer applied in-place.

   Implication: This is a minimal, fast decryption/obfuscation primitive. It strongly indicates the module performs a simple bytewise XOR transform on an in-memory region — very likely the decrypted payload region observed at runtime. Note: your previous mention of XOR+ROL may still be true elsewhere, but this excerpt shows a pure XOR loop. Search other routines for rol/rcl/ror instructions if you expect rotated-key schemes.

2) The `reptile_init` routine (module init) implements the principal loader behavior. Key actions observed and their forensic/behavioral implications:
   a) **Syscall table discovery**:
   - Calls `find_sys_call_table`; if that fails, calls `ia32_find_sys_call_table`. The code then sets `cs:sct` to the discovered syscall table pointer.

- An alternate scanning approach is present (looping `page_offset_base` and comparing `qword ptr [rax+18h]` to `sys_close`) — this is a standard kernel-scan technique when `kallsyms` is not available or when the symbol is not exported.

b) **Retrieval and storage of original syscall handlers:** After locating `sct`, the module reads specific offsets (e.g., `[rax+388h]`, `[rax+1F0h]`, `[rax+6C8h]`, `[rax+270h]`) and stores them into module-local `cs:o_setreuid`, `cs:o_kill`, `cs:o_getdents64`, `cs:o_getdents`, `cs:o_read`. These are the *original* syscall addresses saved for later restoration or use

c) **Overwriting syscall table entries with malicious handlers:** It writes its own handler addresses (`offset l33t_setreuid`, `l33t_kill`, `l33t_getdents64`, `l33t_getdents`, `l33t_read`) into the syscall table (`mov qword ptr [rax+...] , offset l33t_*`). This is a textbook syscall-hooking technique that redirects kernel syscalls to attacker-controlled routines (to hide processes/files, intercept reads, etc.).

d) **Netfilter (NF) hook registration:** The code populates `magic_packet_hook_options` and calls `nf_register_hook`. This registers a netfilter hook (likely to intercept or inspect network traffic, filter packets, or implement covert C2 callbacks).

e) **Paravirtualization & CPU ops:** Calls via `pv_cpu_ops` are used to interact with CPU-specific operations (possibly to maintain atomicity or to interface with paravirtualized environments). This indicates the module attempts to be portable on various kernel configurations (Xen/KVM paravirt-friendly)

f) **Workqueue and call_usermodehelper:** The module creates a workqueue via `__alloc_workqueue_key`, stores it in `cs:work_queue`, and then calls `call_usermodehelper` with `argv` pointing to `"/reptile/reptile_start.sh"`. This is how the kernel invokes a user-space program from kernel context. The presence of `alloc_workqueue` and `call_usermodehelper` shows synchronization/async launch semantics (spawn from kernel safely in workqueue context).

Implication: `reptile_init` performs several classic rootkit bootstrap actions: locate syscall table (often via scanning if not exported), save original handlers, install hooks, register network hooks, and spawn a user-mode component. This is the core of the runtime compromise.

3) The `reptile_exit` disassembly shows careful cleanup logic:
   - Checks if original syscall handlers (`cs:o_setreuid`, `cs:o_kill`, etc.) are not zero; if present, it restores them into the syscall table (`mov [rax+...] , rdx` with `rdx` holding the saved original).
   - Calls `nf_unregister_hook` to remove the netfilter hook.
   - Calls `flush_workqueue` and `destroy_workqueue` to remove queued kernel work.
   - Calls `schedule` in a loop while `read_on.counter != 0` — this suggests coordinated shutdown waiting for active I/O to finish, or to ensure cleanup safe state prior to returning.
   - End: `pop rbp` / return thunk.

Implication: The module implements graceful uninstall behavior — it restores patched syscalls and removes hooks. This is typical for an attacker who wants to avoid leaving obvious indicators or for test/debug builds; it also complicates runtime forensics (if an analyst triggers `cleanup` code, evidence can be removed).

9 EMBEDED DATA & ENTROPY ANALYSIS

9.1 BINWALK / ENTROPY MODULE NOTE

An attempt to use `binwalk -E reptile.ko` on the analysis host resulted in a numba-related exception originating from binwalk's entropy module (stack trace indicates `numba.core.errors.TypingError` associated with `str2bytes`). This is a known compatibility problem between binwalk's entropy routine and certain combinations of Python/numba/llvmlite. The failure is environmental and does not indicate anything about the sample itself. To obtain entropy information, the analysis employed a standalone Shannon entropy scanner (no numba) and produced the results included below.

9.2 ENTROPY SCAN RESULT

A block-based Shannon entropy scan was performed on `reptile.ko` with a block size of 4096 bytes and a high-entropy threshold set to 7.5 (the same threshold used by many entropy-based analysis tools). The scanner output shows the per-block entropy values for the entire file. Key observations:

- No block exceeded the threshold of 7.5; the scan therefore reports **no high-entropy blocks** indicative of large, contiguous encrypted or compressed payloads.
- Entropy values across the file are predominantly in the 2.0–5.5 range. There are localized regions of elevated entropy (e.g., offsets around 188,416 — entropy ~5.89; offsets in the low-to-mid 200k range show entropies in the 5.0–5.3 band), but these values remain far below the 7.5 threshold typically used to indicate compacted or encrypted payloads.
- The largest contributors to higher entropy are the DWARF/debug sections (`.debug_info`, `.debug_str`, etc.), which together constitute a substantial portion of the file and naturally produce higher entropy values due to dense symbolic and string content.

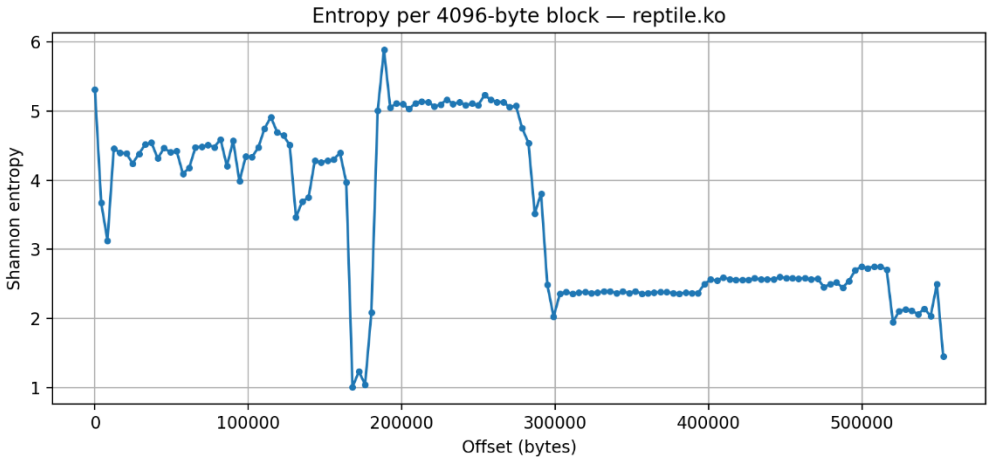
Interpretation: The entropy profile is consistent with a debug-built kernel module that contains sizable DWARF information and string tables rather than a single, embedded encrypted payload. This static finding corroborates the dynamic evidence presented in Report2: the runtime payload is produced or decrypted in memory and does not appear as a single, high-entropy static blob within `reptile.ko`.

```
Scanning reptile.ko with block size 4096 bytes

      Offset  Entropy  High?
------------------------------
         0   5.3157
      4096   3.6723
      8192   3.1289
     12288   4.4592
```

```
 16384   4.3991
 20480   4.3871
 24576   4.2441
 28672   4.3817
 32768   4.5174
 36864   4.5426
 40960   4.3158
 45056   4.4668
 49152   4.4034
 53248   4.4260
 57344   4.0948
 61440   4.1789
 65536   4.4761
 69632   4.4878
 73728   4.5154
 77824   4.4809
 81920   4.5926
 86016   4.2040
 90112   4.5707
 94208   3.9848
 98304   4.3437
102400   4.3359
106496   4.4734
110592   4.7496
114688   4.9149
118784   4.6959
122880   4.6490
126976   4.5083
131072   3.4629
135168   3.6943
139264   3.7492
143360   4.2867
147456   4.2591
151552   4.2872
155648   4.3020
159744   4.3965
163840   3.9706
167936   1.0065
172032   1.2325
176128   1.0452
180224   2.0913
184320   5.0046
188416   5.8890
192512   5.0550
196608   5.1096
200704   5.1015
204800   5.0325
208896   5.1160
212992   5.1402
```

```
217088   5.1316
221184   5.0722
225280   5.0928
229376   5.1696
233472   5.1054
237568   5.1271
241664   5.0873
245760   5.1090
249856   5.0894
253952   5.2308
258048   5.1658
262144   5.1340
266240   5.1330
270336   5.0566
274432   5.0801
278528   4.7513
282624   4.5413
286720   3.5195
290816   3.8089
294912   2.4871
299008   2.0263
303104   2.3565
307200   2.3880
311296   2.3546
315392   2.3721
319488   2.3867
323584   2.3661
327680   2.3796
331776   2.3898
335872   2.3932
339968   2.3673
344064   2.3897
348160   2.3695
352256   2.3954
356352   2.3585
360448   2.3680
364544   2.3793
368640   2.3807
372736   2.3803
376832   2.3657
380928   2.3617
385024   2.3714
389120   2.3703
393216   2.3703
397312   2.4999
401408   2.5634
405504   2.5517
409600   2.5918
413696   2.5695
```

```
    417792    2.5587
    421888    2.5600
    425984    2.5585
    430080    2.5824
    434176    2.5639
    438272    2.5651
    442368    2.5711
    446464    2.5993
    450560    2.5860
    454656    2.5882
    458752    2.5792
    462848    2.5872
    466944    2.5678
    471040    2.5768
    475136    2.4575
    479232    2.4948
    483328    2.5201
    487424    2.4473
    491520    2.5428
    495616    2.6974
    499712    2.7533
    503808    2.7284
    507904    2.7496
    512000    2.7511
    516096    2.7105
    520192    1.9485
    524288    2.1056
    528384    2.1335
    532480    2.1141
    536576    2.0619
    540672    2.1434
    544768    2.0340
    548864    2.4990
    552960    1.4499

No high-entropy blocks found above threshold.
```



10 COMPARISON WITH THE ORIGINAL REPORT1 SAMPLE

A concise comparison between the original Report1 artifact and the current compiled variant:

- **Original sample (Report1)**: SHA256 `d182239d…37ba`. This artifact contained a statically embedded `parasite_blob` (~0x82050 bytes) visible in the symbol table and in static analysis; attempts to execute it in the CentOS7 analysis environment triggered kernel verification failures and a kernel taint message (`module verification failed: signature and/or required key missing`) preventing safe and reproducible dynamic execution.
- **Compiled variant (this report)**: `reptile.ko` — SHA256 `9573f414…f9d51`. Built from the available Reptile source with debug symbols included. The compiled binary does **not** expose a visible `parasite_blob` in the static symbol table. Instead, static relocations and code patterns indicate that the payload is constructed or decrypted in memory at runtime. Dynamic evidence in Report2 confirms execution of a decrypted in-memory payload, a transient `/reptile_shell` entity and syscall table anomalies.

Analytical implication: substituting the compiled variant was necessary to safely reproduce the runtime behavior in a controlled environment. Static comparisons retain lineage and behavioral parity (decryption routine, symbol resolution and hooking strategy) while acknowledging byte-level differences introduced by compilation and debug information.

11 INDICATORS OF COMPROMISE (IOCs) & YARA RULE(S) — STATIC & CORROBORATED DYNAMIC

11.1 INDICATORS OF COMPROMISE (IOCs)

File/Hashes: `reptile.ko` — SHA256:
`9573f414bedbd78464c9d9e4c214c37aea9a1b25dd3a6b4b512e65bbec9f9d51`

High – signal strings: `/reptile/reptile_shell`, `reptile_init`, `reptile_exit`, `#<reptile>`.

Symbolic / behavioral IOCs:

- Patterns consistent with `kallsyms_on_each_symbol`-style resolution and runtime symbol construction.
- Static relocation structures and call patterns consistent with later in-memory hooking of syscalls; dynamic memory dumps show altered syscall targets corroborating this behavior.

Network (from Report 2):

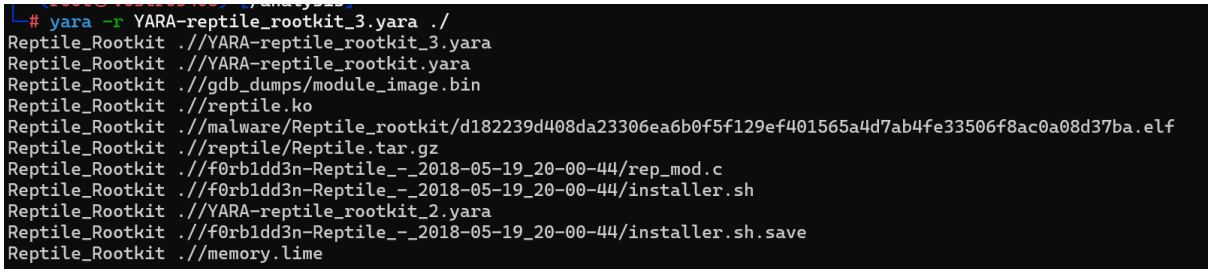
- Attacker/victim simulation IPs: `192.168.1.130 ↔ 192.168.1.150` with captured encrypted sessions.
- Evidence of file transfer activity observed in PCAP (e.g., exfiltration indicators).

11.2 YARA RULE:

```
rule Reptile_Rootkit
{
    strings:
        /* core module markers / high-signal strings */
        $core1 = "reptile" ascii
        $core2 = "rep_mod" ascii
        $core3 = "invisible" ascii
        $core4 = "mod_list" ascii

        /* parasite/blob / static-key indicators */
        $par1  = "parasite" ascii
        $par2  = "static_key" ascii
        $par3  = "kmatryoshka" ascii
        $par4  = "encrypt" ascii

        /* kernel-resolution / hooking indicators */
        $k1    = "kallsyms_on_each_symbol" ascii
        $k2    = "kallsyms_lookup_name" ascii
        $k3    = "sys_call_table" ascii
        $k4    = "init_module" ascii
        $k5    = "insmod" ascii

    condition:
        (
```

```
            /* Primary branch: file-like artifact with ELF header
present */
            uint32(0) == 0x464c457f and
          ( (2 of ($core*) or                /* all core markers must
be present */
            3 of ($par*) ) and                 /* at least one
parasite-style token */
            4 of ($k*) )                     /* at least one
kallsyms/syscall token */
         )
         or
         (
            /* Fallback branch: memory dumps or stripped fragments
without ELF header:
               require the same logical sets but omit the ELF header
and vermagic check */
            ( 2 of ($core*) or
            3 of ($par*) ) and
            any of ($k*)
         )
}
```

Evidence:



12 LIMITATIONS AND CAVEATS

- Artifact differences: `reptile.ko` (compiled test module) and the original internet-sourced sample are not bitwise identical. Differences arise from compilation flags, embedded debug information, build timestamps, and potential local modifications. Therefore, file hashes differ and byte-level IOCs cannot be used interchangeably. The analysis emphasizes behavioral parity (decryption routine, symbol resolution, hooking behavior) rather than exact binary equivalence.
- **Module signing & kernel taint:** the original sample produced a kernel verification failure and kernel tainting on the analysis host. Kernel module verification policy (`CONFIG_MODULE_SIG`, `CONFIG_MODULE_SIG_ALL`) can alter whether a module is loaded, partially loaded, or rejected, and it can introduce additional log noise that must be accounted for when interpreting dynamic runs. Preserve relevant `dmesg` lines and kernel configurations when reproducing experiments.
- **Debug symbols present:** the compiled variant contains sizable DWARF information. While this facilitates static analysis and mapping between source and binary, it may expose internal names that would not appear in a stripped production sample. Analysts should account for this when generalizing detection strategies.

13 CONCLUSION

The static re-analysis of the locally compiled `reptile.ko` confirms the module's identity as a loader-style LKM derived from the Reptile source tree. Static evidence — symbol names, relocation patterns and a decryption-like control flow observed in `.init.text` — aligns with Report2's dynamic findings: the payload is reconstructed or decrypted at runtime, appears in memory as executable code, manifests as a transient `/reptile_shell` entity, and results in kernel-level hooking and syscall anomalies.

Because the compiled variant was used as a controlled instrument to enable reproducible dynamic observation (necessitated by signing/tainting constraints on the original sample), the combined static + dynamic chain provides a robust forensic narrative. Defenders should prioritize behavioral detection of kernel module loading and runtime symbol resolution, and treat kernel taint and unexpected module load events as high-priority incidents.

14 REFERENCES

1. Report1 — Static Triage and ELF-level Findings (original sample: SHA256 `d182239d408da23306ea6b0f5f129ef401565a4d7ab4fe33506f8ac0a08d37ba`)
2. Report2 — Dynamic Analysis and Runtime Evidence for `reptile.ko` (compiled variant)
3. Public Reptile source tree (author: F0rb1dd3n) and associated blog advisories (ASEC, vendor technical notes)
4. Linux kernel module programming documentation and kernel hardening guides