decoded.avast.io /davidalvarez/linux-threat-hunting-syslogk-a-kernel-rootkit-found-under-development-in-the-wild/

Linux Threat Hunting: ‘Syslogk’ a kernel rootkit found under development in the wild

: 6/13/2022

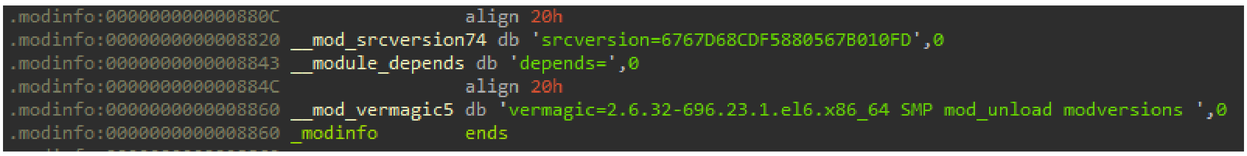
by David Álvarez and Jan Neduchal
June 13, 202213 min read

Introduction

Rootkits are dangerous pieces of malware. Once in place, they are usually really hard to detect. Their code is typically more challenging to write than other malware, so developers resort to code reuse from open source projects. As rootkits are very interesting to analyze, we are always looking out for these kinds of samples in the wild.

Adore-Ng is a relatively old, open-source, well-known kernel rootkit for Linux, which initially targeted kernel 2.x but is currently updated to target kernel 3.x. It enables hiding processes, files, and even the kernel module, making it harder to detect. It also allows authenticated user-mode processes to interact with the rootkit to control it, allowing the attacker to hide many custom malicious artifacts by using a single rootkit.

In `early 2022`, we were analyzing a rootkit mostly based on `Adore-Ng` that we found in the wild, apparently under development. After obtaining the sample, we examined the `.modinfo` section and noticed it is compiled for a specific kernel version.



```
.modinfo:000000000000880C                      align 20h
.modinfo:0000000000008820 __mod_srcversion74 db 'srcversion=6767D68CDF5880567B010FD',0
.modinfo:0000000000008843 __module_depends db 'depends=',0
.modinfo:000000000000884C                      align 20h
.modinfo:0000000000008860 __mod_vermagic5 db 'vermagic=2.6.32-696.23.1.el6.x86_64 SMP mod_unload modversions ',0
.modinfo:0000000000008860 _modinfo           ends
```

As you may know, even if it is possible to ‘force load’ the module into the kernel by using the `--force` flag of the insmod Linux command, this operation can fail if the required symbols are not found in the kernel; this can often lead to a system crash.

```
insmod -f {module}
```

We discovered that the kernel module could be successfully loaded without forcing into a default Centos 6.10 distribution, as the rootkit we found is compiled for a similar kernel version.

While looking at the file's strings, we quickly identified the `PgSD93ql` hardcoded file name in the kernel rootkit to reference the payload. This payload file name is likely used to make it less obvious for the sysadmin, for instance, it can look like a legitimate PostgreSQL file.

Address	Length	Type	String
.rodata.str1.1:00000000000084AD	00000009	C	PgSD93ql
.rodata.str1.1:00000000000084DC	0000001A	C	/etc/rc-Zobk0jpi/PgSD93ql
.rodata.str1.1:0000000000008556	00000009	C	PgSD93ql

Using this hardcoded file name, we extracted the file hidden by the rootkit. It is a compiled backdoor trojan written in C programming language; Avast's antivirus engine detects and classifies this file as `ELF:Rekoob` – which is widely known as the Rekoobe malware family. `Rekoobe` is a piece of code implanted in legitimate servers. In this case it is embedded in a fake SMTP server, which spawns a shell when it receives a specially crafted command. In this post, we refer to this rootkit as `Syslogk` rootkit, due to how it 'reveals' itself when specially crafted data is written to the file `/proc/syslogk` .

Analyzing the Syslogk rootkit

The `Syslogk` rootkit is heavily based on `Adore-Ng` but incorporates new functionalities making the user-mode application and the kernel rootkit hard to detect.

Loading the kernel module

To load the rootkit into kernel space, it is necessary to approximately match the kernel version used for compiling; it does not have to be strictly the same.

```
vermagic=2.6.32-696.23.1.el6.x86_64 SMP mod_unload modversions
```

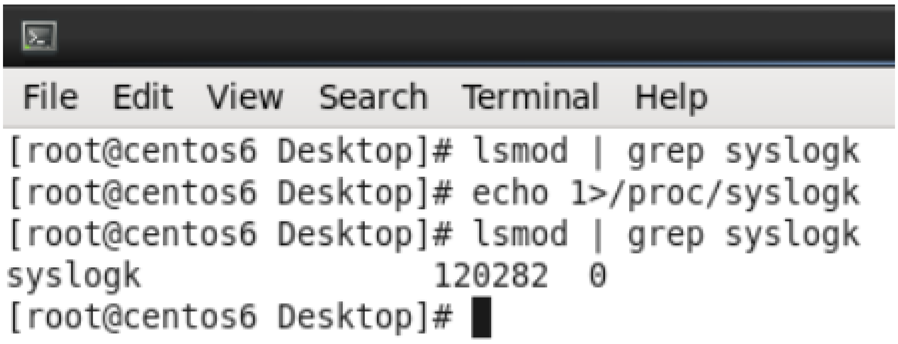
For example, we were able to load the rootkit without any effort in a Centos 6.10 virtual machine by using the insmod Linux command.

After loading it, you will notice that the malicious driver does not appear in the list of loaded kernel modules when using the lsmod command.

Revealing the rootkit

The rootkit has a `hide_module` function which uses the list_del function of the kernel API to remove the module from the linked list of kernel modules. Next, it also accordingly updates its internal `module_hidden` flag.

Fortunately, the rootkit has a functionality implemented in the `proc_write` function that exposes an interface in the /proc file system which reveals the rootkit when the value `1` is written into the file *`/proc/syslogk`*.



Once the rootkit is revealed, it is possible to remove it from memory using the rmmod Linux command. The Files section of this post has additional details that will be useful for programmatically uncloaking the rootkit.

Overview of the Syslogk rootkit features

Apart from hiding itself, making itself harder to detect when implanted, `Syslogk` can completely hide the malicious payload by taking the following actions:

- The `hk_proc_readdir` function of the rootkit hides directories containing malicious files, effectively hiding them from the operating system.
- The malicious processes are hidden via `hk_getpr` – a mix of Adore-Ng functions for hiding processes.
- The malicious payload is hidden from tools like `Netstat`; when running, it will not appear in the list of services. For this purpose, the rootkit uses the function `hk_t4_seq_show`.
- The malicious payload is not continuously running. The attacker remotely executes it on demand when a specially crafted TCP packet (details below) is sent to the infected machine, which inspects the traffic by installing a `netfilter hook`.
- It is also possible for the attacker to remotely stop the payload. This requires using a `hardcoded key` in the rootkit and knowledge of some fields of the `magic packet` used for remotely starting the payload.

We observed that the `Syslogk` rootkit (and Rekoobe payload) perfectly align when used covertly in conjunction with a fake SMTP server. Consider how stealthy this could be; a backdoor that does not load until some magic packets are sent to the machine. When queried, it appears to be a legitimate service hidden in memory, hidden on disk, remotely 'magically' executed, hidden on the network. Even if it is found during a network port scan, it still seems to be a legitimate SMTP server.

For compromising the operating system and placing the mentioned hiding functions, `Syslogk` uses the already known set_addr_rw and set_addr_ro rootkit functions, which adds or removes writing permissions to the `Page Table Entry` (PTE) structure.

After adding writing permissions to the PTE, the rootkit can hook the functions declared in the `hks` internal rootkit structure.

PTE Hooks

Type of the function	Offset	Name of the function
Original	hks+(0x38) * 0	proc_root_readdir

Hook	hks+(0x38) * 0 + 0x10	hk_proc_readdir
Original	hks+(0x38) * 1	tcp4_seq_show
Hook	hks+(0x38) * 1 + 0x10	hk_t4_seq_show
Original	hks+(0x38) * 2	sys_getpriority
Hook	hks+(0x38) * 2 + 0x10	hk_getpr

The mechanism for placing the hooks consists of identifying the hookable kernel symbols via `/proc/kallsyms` as implemented in the `get_symbol_address` function of the rootkit (code reused from this repository). After getting the address of the symbol, the `Syslogk` rootkit uses the udis86 project for hooking the function.

Understanding the directory hiding mechanism

The Virtual File System (VFS) is an abstraction layer that allows for FS-like operation over something that is typically not a traditional FS. As it is the entry point for all the File System queries, it is a good candidate for the rootkits to hook.

It is not surprising that the Syslogk rootkit hooks the VFS functions for hiding the Rekoobe payload stored in the file `/etc/rc-Zobk0jpi/PgSD93ql`.

The hook is done by `hk_root_readdir` which calls to `nw_root_filldir` where the directory filtering takes place.

```
mov     rsi, offset aZobk0jpi ; "-Zobk0jpi"
mov     rdi, rbx         ; haystack
mov     r12d, edx
mov     [rbp+var_38], r9d
mov     r14, rcx
mov     r15, r8
call    strstr
```

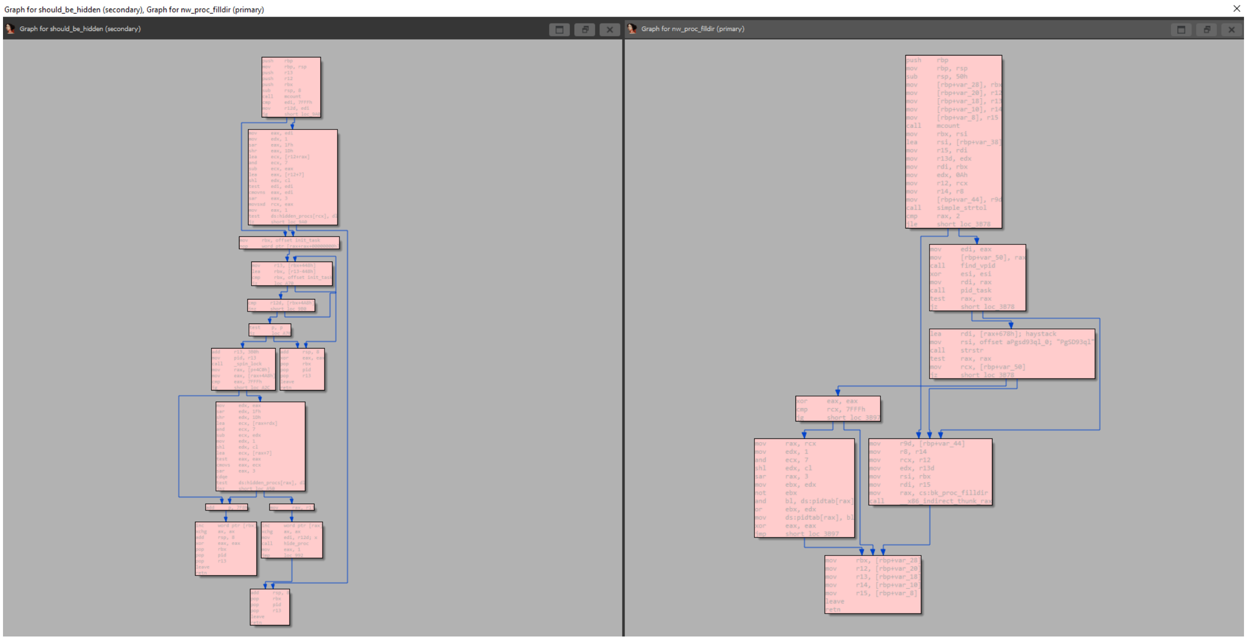
As you can see, any directory containing the substring `-Zobk0jpi` will be hidden.

The function `hk_get_vfs` opens the root of the file system by using filp_open. This kernel function returns a pointer to the structure file, which contains a `file_operations` structure called f_op that finally stores the readdir function hooked via `hk_root_readdir`.

Of course, this feature is not new at all. You can check the source code of `Adore-Ng` and see how it is implemented on your own.

Understanding the process hiding mechanism

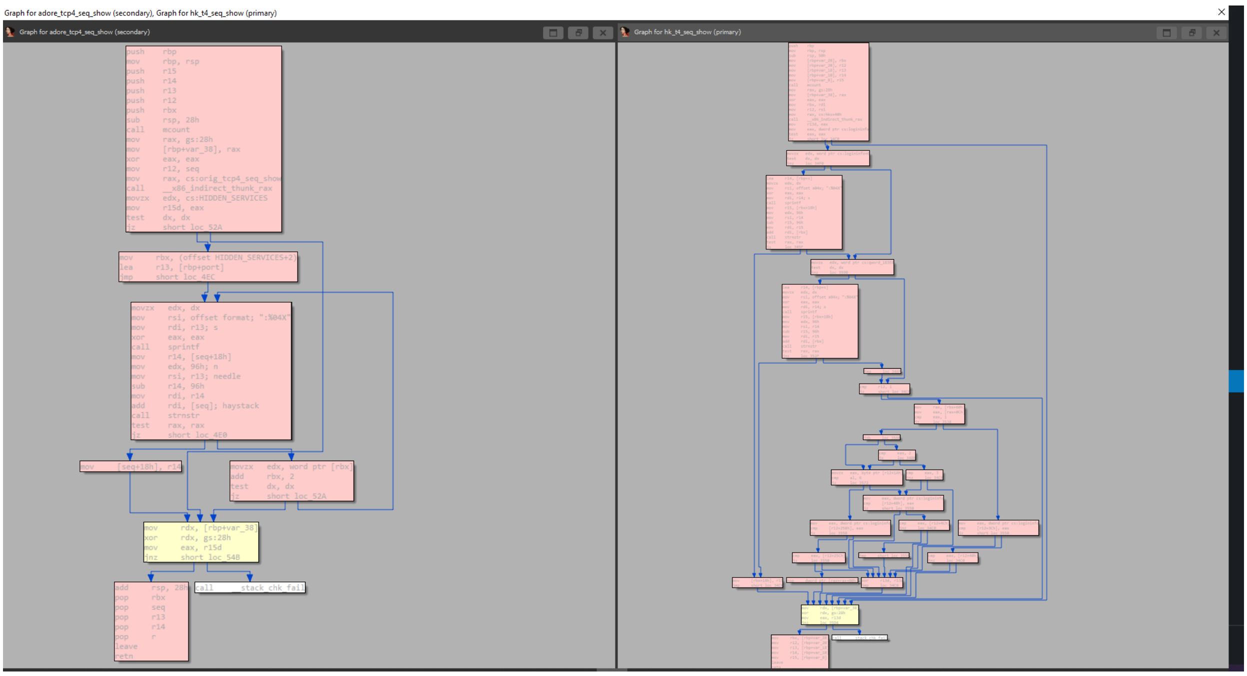
In the following screenshot, you can see that the `Syslogk` rootkit (code at the right margin of the screenshot) is prepared for hiding a process called `PgSD93ql`. Therefore, the rootkit seems more straightforward than the original version (see Adore-Ng at the left margin of the screenshot). Furthermore, the process to hide can be selected after authenticating with the rootkit.



The `Syslogk` rootkit function `hk_getpr` explained above, is a mix of adore_find_task and should_be_hidden functions but it uses the same mechanism for hiding processes.

Understanding the network traffic hiding mechanism

The `Adore-Ng` rootkit allows hiding a given set of listening services from Linux programs like `Netstat`. It uses the exported proc_net structure to change the tcp4_seq_show( ) handler, which is invoked by the kernel when `Netstat` queries for listening connections. Within the adore_tcp4_seq_show() function, strnstr( ) is used to look in `seq->buf` for a substring that contains the hexadecimal representation of the port it is trying to hide. If this is found, the string is deleted.



In this way, the backdoor will not appear when listing the connections in an infected machine. The following section describes other interesting capabilities of this rootkit.

Understanding the magic packets

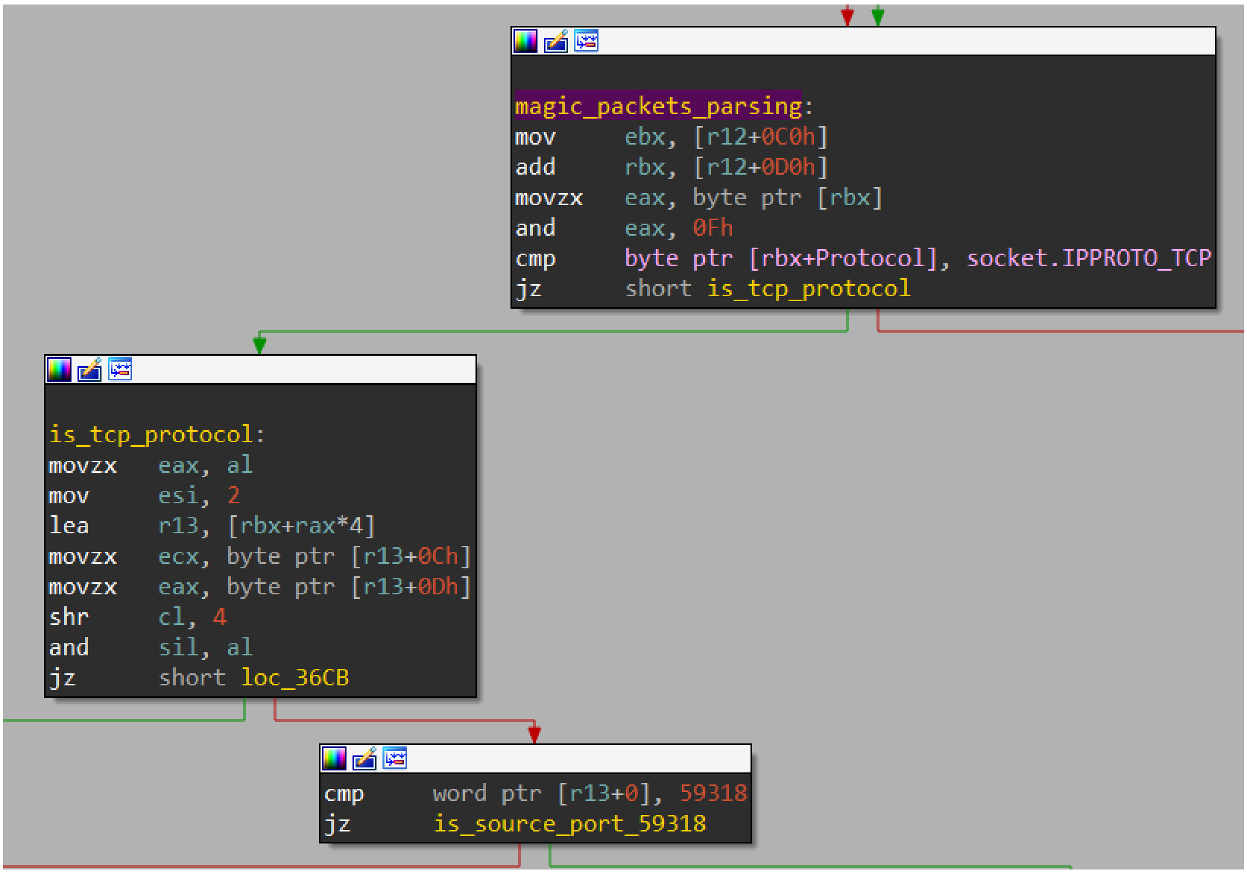
Instead of continuously running the payload, it is remotely started or stopped on demand by sending specially crafted network traffic packets.

These are known as `magic packets` because they have a special format and special powers. In this implementation, an attacker can trigger actions without having a listening port in the infected machine such that the commands are, in some way, 'magically' executed in the system.

Starting the Rekoobe payload

The `magic packet` inspected by the `Syslogk`rootkit for starting the `Rekoobe` fake SMTP server is straightforward. First, it checks whether the packet is a TCP packet and, in that case, it also checks the `source port`, which is expected to be `59318`.

`Rekobee` will be executed by the rootkit if the magic packet fits the mentioned criteria.



Of course, before executing the fake service, the rootkit terminates all existing instances of the program by calling the rootkit function `pkill_clone_0`. This function contains the hardcoded process name `PgSD93ql`; it only kills the `Rekoobe` process by sending the `KILL` signal via send_sig.

To execute the command that starts the `Rekoobe` fake service in user mode, the rootkit executes the following command by combining the kernel APIs: call_usermodehelper_setup, call_usermodehelper_setfns, and call_usermodehelper_exec.

```
/bin/sh -c /etc/rc-Zobk0jpi/PgSD93ql
```

The Files section of this post demonstrates how to manually craft (using Python) the TCP `magic packet` for starting the `Rekoobe` payload.

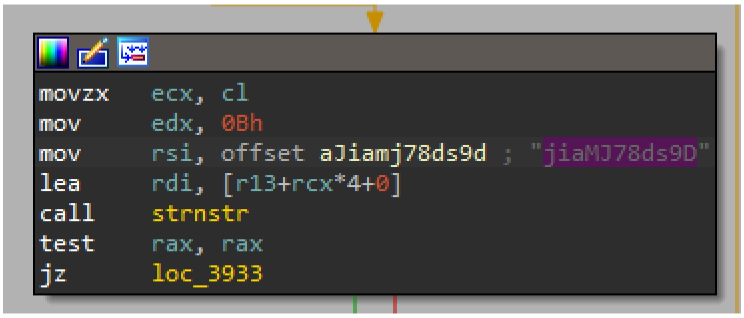
In the next section we describe a more complex form of the `magic packet`.

Stopping the Rekoobe payload

Since the attacker doesn't want any other person in the network to be able to kill `Rekoobe`, the `magic packet` for killing `Rekoobe` must match some fields in the previous `magic packet` used for starting `Rekoobe`. Additionally, the packet must satisfy additional requirements – it must contain a key that is hardcoded in the rootkit and located in a variable offset of the `magic packet`. The conditions that are checked:

1. It checks a flag enabled when the rootkit executes `Rekoobe` via `magic packets`. It will only continue if the flag is enabled.
2. It checks the `Reserved` field of the TCP header to see that it is `0x08`.
3. The `Source Port` must be between `63400` and `63411` inclusive.
4. Both the `Destination Port` and the `Source Address`, must to be the same that were used when sending the `magic packet` for starting `Rekoobe`.
5. Finally, it looks for the `hardcoded key`. In this case, it is: `D9sd87JMaij`

The offset of the hardcoded key is also set in the packet and not in a hardcoded offset; it is calculated instead. To be more precise, it is set in the `data offset` byte (TCP header) such that after shifting the byte `4 bits` to the right and multiplying it by `4`, it points to the offset of where the `Key` is expected to be (as shown in the following screenshot, notice that the rootkit compares the `Key` in reverse order).



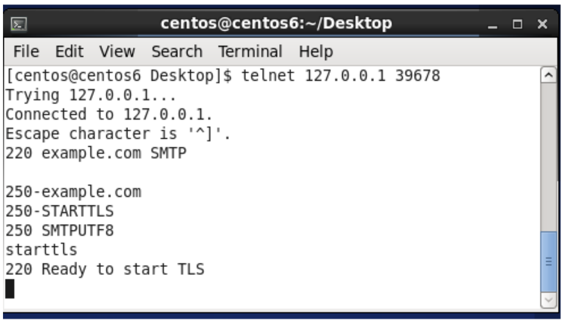
In our experiments, we used the value `0x50` for the `data offset` (TCP header) because after shifting it 4 bits, you get 5 which multiplied by 4 is equal to `20`. Since 20 is precisely the size of the TCP Header, by using this value, we were able to put the key at the start of the data section of the packet.

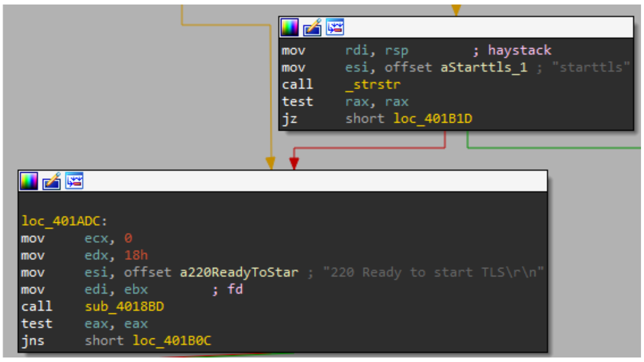
If you are curious about how we implemented this `magic packet` from scratch, then please see the Files section of this blog post.

Analyzing Rekoobe

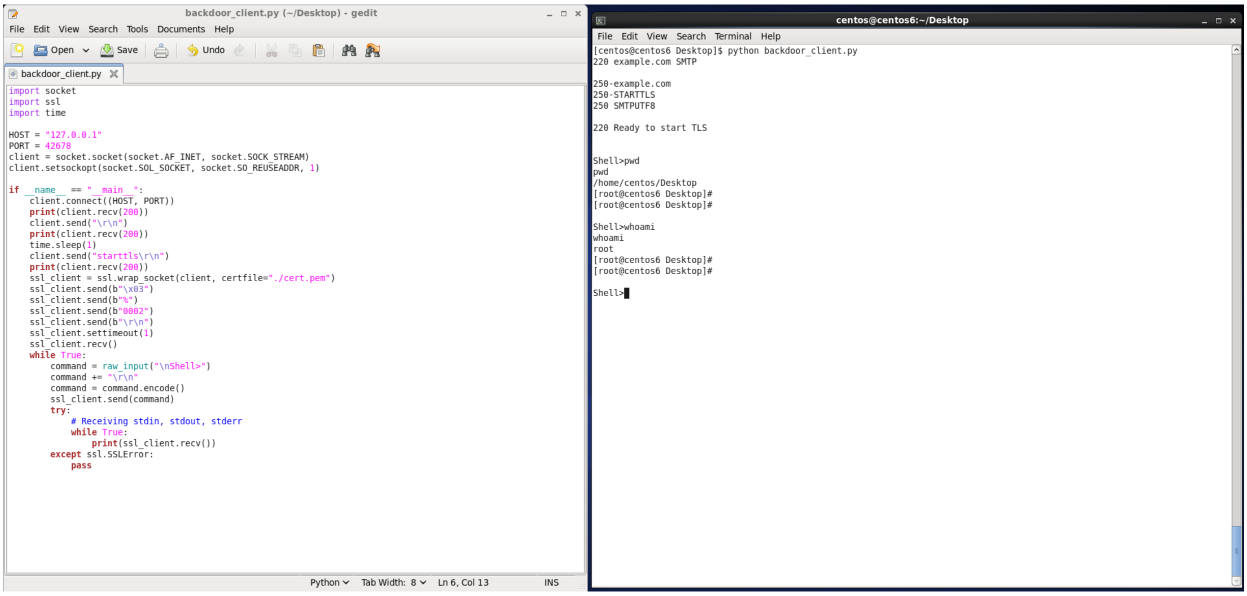
When the infected machine receives the appropriate `magic packet`, the rootkit starts the hidden `Rekoobe` malware in user mode space.

It looks like an innocent SMTP server, but there is a backdoor command on it that can be executed when handling the `starttls` command. In a legitimate service, this command is sent by the client to the server to advise that it wants to start TLS negotiation.





For triggering the `Rekoobe` backdoor command (spawning a shell), the attacker must send the byte `0x03` via TLS, followed by a `Tag Length Value` (TLV) encoded data. Here, the tag is the symbol `%`, the length is specified in four numeric characters, and the value (notice that the length and value are arbitrary but can not be zero).

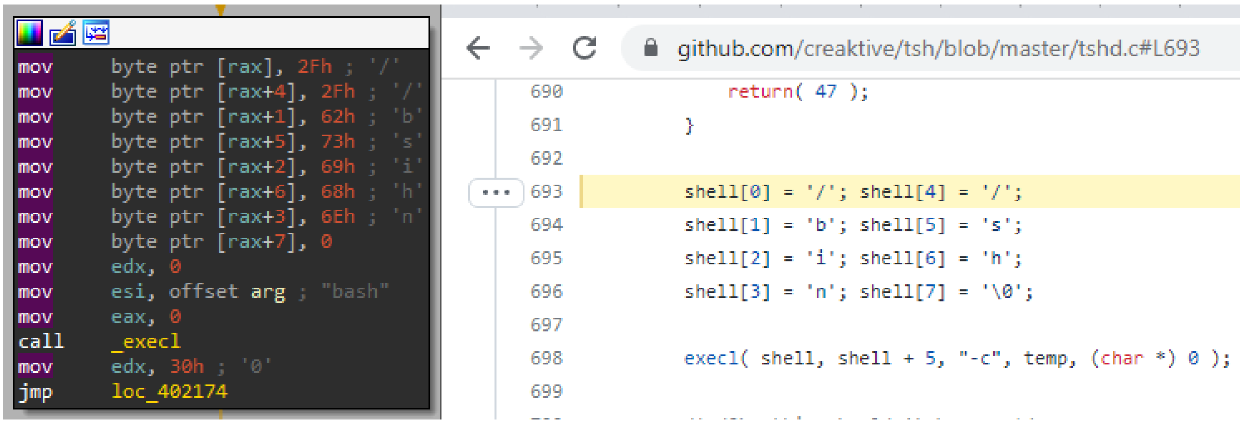


Additionally, to establish the TLS connection, you will need the certificate embedded in `Rekoobe`.

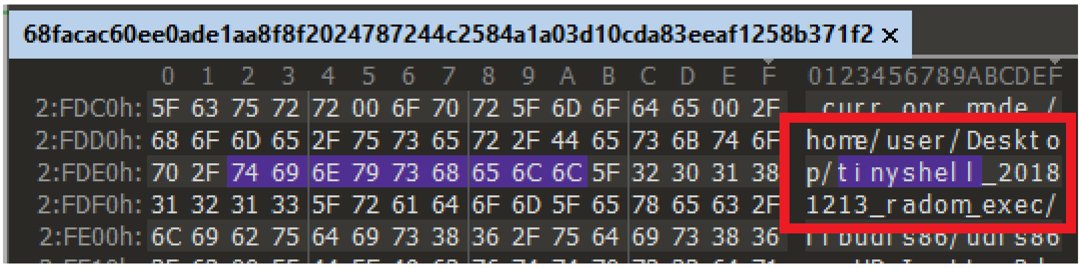
See the Files section below for the certificate and a Python script we developed to connect with `Rekoobe`.

The origin of Rekoobe payload and Syslogk rootkit

`Rekoobe` is clearly based on the TinySHell open source project; this is based on ordering observed in character and variables assignment taking place in the same order multiple times.



On the other hand, if you take a look at the `Syslogk` rootkit, even if it is new, you will notice that there are also references to `TinySHell` dating back to December 13, 2018.



The evidence suggests that the threat actor developed `Rekoobe` and `Syslogk` to run them together. We are pleased to say that our users are protected and hope that this research assists others.

Conclusions

One of the architectural advantages of security software is that it usually has components running in different privilege levels; malware running on less-privileged levels cannot easily interfere with processes running on higher privilege levels, thus allowing more straightforward dealing with malware.

On the other hand, kernel rootkits can be hard to detect and remove because these pieces of malware run in a privileged layer. This is why it is essential for system administrators and security companies to be aware of this kind of malware and write protections for their users as soon as possible.

IoCs

Syslogk sample

- `68facac60ee0ade1aa8f8f2024787244c2584a1a03d10cda83eeaf1258b371f2`

Rekoobe sample

- `11edf80f2918da818f3862246206b569d5dcebdc2a7ed791663ca3254ede772d`

Other Rekoobe samples

- `fa94282e34901eba45720c4f89a0c820d32840ae49e53de8e75b2d6e78326074`

- fd92e34675e5b0b8bfbc6b1f3a00a7652e67a162f1ea612f6e86cca846df76c5
- 12c1b1e48effe60eef7486b3ae3e458da403cd04c88c88fab7fca84d849ee3f5
- 06778bddd457aafbc93d384f96ead3eb8476dc1bc8a6fbd0cd7a4d3337ddce1e
- f1a592208723a66fa51ce1bc35cbd6864e24011c6dc3bcd056346428e4e1c55d
- 55dbdb84c40d9dc8c5aaf83226ca00a3395292cc8f884bdc523a44c2fd431c7b
- df90558a84cfcf80639f32b31aec187b813df556e3c155a05af91dedfd2d7429
- 160cfb90b81f369f5ba929aba0b3130cb38d3c90d629fe91b31fdef176752421
- b4d0f0d652f907e4e77a9453dcce7810b75e1dc5867deb69bea1e4ecdd02d877
- 3a6f339df95e138a436a4feff64df312975a262fa16b75117521b7d6e7115d65
- 74699b0964a2cbdc2bc2d9ca0b2b6f5828b638de7c73b1d41e7fe26cfc2f3441
- 7a599ff4a58cb0672a1b5e912a57fcdc4b0e2445ec9bc653f7f3e7a7d1dc627f
- f4e3cfeeb4e10f61049a88527321af8c77d95349caf616e86d7ff4f5ba203e5f
- 31330c0409337592e9de7ac981cecb7f37ce0235f96e459fefbd585e35c11a1a
- c6d735b7a4656a52f3cd1d24265e4f2a91652f1a775877129b322114c9547deb
- 2e81517ee4172c43a2084be1d584841704b3f602cafc2365de3bcb3d899e4fb8
- b22f55e476209adb43929077be83481ebda7e804d117d77266b186665e4b1845
- a93b9333a203e7eed197d0603e78413013bd5d8132109bbef5ef93b36b83957c
- 870d6c202fcc72088ff5d8e71cc0990777a7621851df10ba74d0e07d19174887
- ca2ee3f30e1c997cc9d8e8f13ec94134cdb378c4eb03232f5ed1df74c0a0a1f0
- 9d2e25ec0208a55fba97ac70b23d3d3753e9b906b4546d1b14d8c92f8d8eb03d
- 29058d4cee84565335eafdf2d4a239afc0a73f1b89d3c2149346a4c6f10f3962
- 7e0b340815351dab035b28b16ca66a2c1c7eaf22edf9ead73d2276fe7d92bab4
- af9a19f99e0dcd82a31e0c8fc68e89d104ef2039b7288a203f6d2e4f63ae4d5c
- 6f27de574ad79eb24d93beb00e29496d8cfe22529fc8ee5010a820f3865336a9
- d690d471b513c5d40caef9f1e37c94db20e6492b34ea6a3cddcc22058f842cf3
- e08e241d6823efedf81d141cc8fd5587e13df08aeda9e1793f754871521da226
- da641f86f81f6333f2730795de93ad2a25ab279a527b8b9e9122b934a730ab08
- e3d64a128e9267640f8fc3e6ba5399f75f6f0aca6a8db48bf989fe67a7ee1a71
- d3e2e002574fb810ac5e456f122c30f232c5899534019d28e0e6822e426ed9d3
- 7b88fa41d6a03aeda120627d3363b739a30fe00008ce8d848c2cbb5b4473d8bc
- 50b73742726b0b7e00856e288e758412c74371ea2f0eaf75b957d73dfb396fd7
- 8b036e5e96ab980df3dca44390d6f447d4ca662a7eddac9f52d172efff4c58f8
- 8b18c1336770fcddc6fe78d9220386bce565f98cc8ada5a90ce69ce3ddf36043
- f04dc3c62b305cdb4d83d8df2caa2d37feeb0a86fb5a745df416bac62a3b9731
- 72f200e3444bb4e81e58112111482e8175610dc45c6e0c6dcd1d2251bacf7897
- d129481955f24430247d6cc4af975e4571b5af7c16e36814371575be07e72299
- 6fc03c92dee363dd88e50e89062dd8a22fe88998aff7de723594ec916c348d0a
- fca2ea3e471a0d612ce50abc8738085f076ad022f70f78c3f8c83d1b2ff7896b
- 2fea3bc88c8142fa299a4ad9169f8879fc76726c71e4b3e06a04d568086d3470
- 178b23e7eded2a671fa396dd0bac5d790bca77ec4b2cf4b464d76509ed12c51a
- 3bff2c5bfc24fc99d925126ec6beb95d395a85bc736a395aaf4719c301cbbfd4
- 14a33415e95d104cf5cf1acaff9586f78f7ec3ffb26efd0683c468edeaf98fd7

- 8bb7842991afe86b97def19f226cb7e0a9f9527a75981f5e24a70444a7299809
- 020a6b7edcff7764f2aac1860142775edef1bc057bedd49b575477105267fc67
- 6711d5d42b54e2d261bb48aa7997fa9191aec059fd081c6f6e496d8db17a372a
- 48671bc6dbc786940ede3a83cc18c2d124d595a47fb20bc40d47ec9d5e8b85dc
- b0d69e260a44054999baa348748cf4b2d1eaab3dd3385bb6ad5931ff47a920de
- e1999a3e5a611312e16bb65bb5a880dfedbab8d4d2c0a5d3ed1ed926a3f63e94
- fa0ea232ab160a652fcbd8d6db8ffa09fd64bcb3228f000434d6a8e340aaf4cb
- 11edf80f2918da818f3862246206b569d5dcebdc2a7ed791663ca3254ede772d
- 73bbabc65f884f89653a156e432788b5541a169036d364c2d769f6053960351f
- 8ec87dee13de3281d55f7d1d3b48115a0f5e4a41bfbef1ea08e496ac529829c8
- 8285ee3115e8c71c24ca3bdce313d3cfadead283c31a116180d4c2611efb610d
- 958bce41371b68706feae0f929a18fa84d4a8a199262c2110a7c1c12d2b1dce2
- 38f357c32f2c5a5e56ea40592e339bac3b0cabd6a903072b9d35093a2ed1cb75
- bcc3d47940ae280c63b229d21c50d25128b2a15ea42fe8572026f88f32ed0628
- 08a1273ac9d6476e9a9b356b261fdc17352401065e2fc2ad3739e3f82e68705a
- cf525918cb648c81543d9603ac75bc63332627d0ec070c355a86e3595986cbb3
- 42bc744b22173ff12477e57f85fa58450933e1c4294023334b54373f6f63ee42
- 337674d6349c21d3c66a4245c82cb454fea1c4e9c9d6e3578634804793e3a6d6
- 4effa5035fe6bbafd283ffae544a5e4353eb568770421738b4b0bb835dad573b
- 5b8059ea30c8665d2c36da024a170b31689c4671374b5b9b1a93c7ca47477448
- bd07a4ccc8fa67e2e80b9c308dec140ca1ae9c027fa03f2828e4b5bdba6c7391
- bf09a1a7896e05b18c033d2d62f70ea4cac85e2d72dbd8869e12b61571c0327e
- 79916343b93a5a7ac7b7133a26b77b8d7d0471b3204eae78a8e8091bfe19dc8c
- c32e559568d2f6960bc41ca0560ac8f459947e170339811804011802d2f87d69
- 864c261555fce40d022a68d0b0eadb7ab69da6af52af081fd1d9e3eced4aee46
- 275d63587f3ac511d7cca5ff85af2914e74d8b68edd5a7a8a1609426d5b7f6a9
- 031183e9450ad8283486621c4cdc556e1025127971c15053a3bf202c132fe8f9

Files

Syslogk research tools

Rekoobe research tool

- rekoobe_backdoor_client.py
- cert.pem

IoC repository

The Syslogk and Rekoobe rootkit research tools and IoCs are in our IoC repository.