Linux Rekoobe Operating with New, Undetected Malware Samples

intezer.com/blog-linux-rekoobe-operating-with-new-undetected-malware-samples/

January 20, 2020

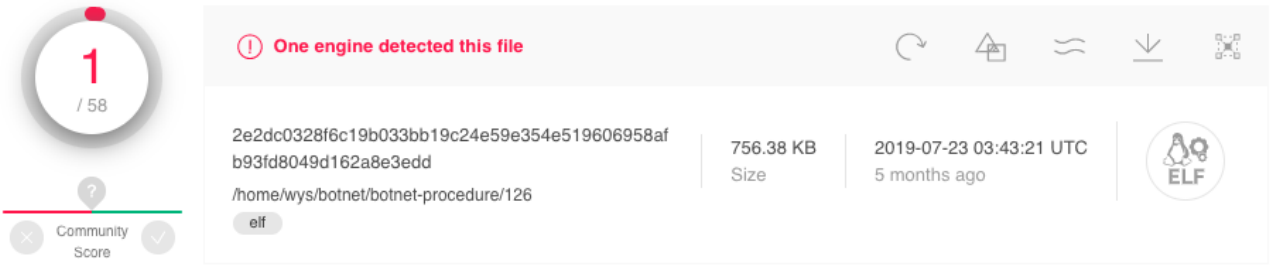
Written by Ignacio Sanmillan - 20 January 2020

Introduction

Our research team has identified new versions of an old Linux malware known as Rekoobe, a minimalistic trojan with a complex CNC authentication protocol originally targeting SPARC and Intel x86, x86-64 systems back in 2015.

The new malware samples have lower detection rates than their predecessors. We believe this malware ceased its operation in 2016 after it was reported, however, based on our findings we can estimate the operators behind Rekoobe have resumed their operations utilizing a newer version of the malware.

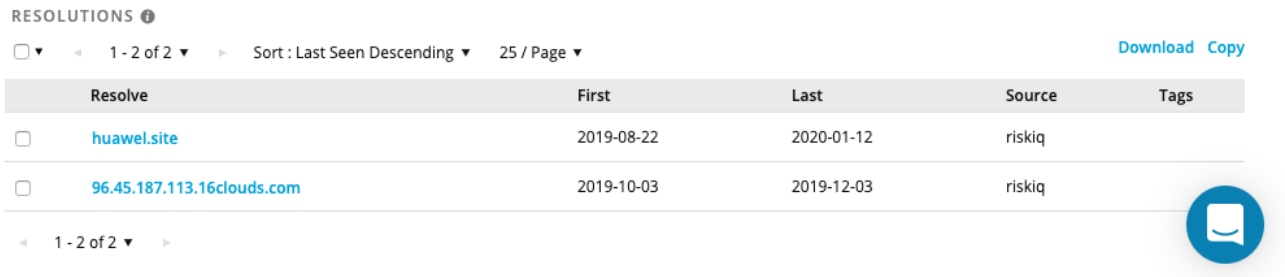
Based on our research, we can estimate the new samples have been in the wild since 2018.



We will present a brief technical analysis of the new Rekoobe samples and explain why we believe the new samples have lower detection rates despite previous versions of Rekoobe being well-detected by different security vendors.

Technical Analysis

Linux Rekoobe was first reported by DrWeb in 2015. We believe this malware resumed its operation some time between 2018 and 2019, based on CNC reverse DNS intelligence from RiskIQ, along with sample information from VirusTotal concerning the new variants.



It's important to mention that the new variants are statically compiled ELF binaries while older variants were dynamically compiled.

This undeniably implies that on a code level these two generations of the same malware are different. Since the compilation flags used by the GCC compiler differ between dynamically and statically linked code, the compiler will generate different code accordingly.

We have also taken into consideration that if indeed these new implants were generated after a three to four year gap, the GCC compiler version would also differ. Consequently, the compiler will generate different code on an assembly level even though additional compilation flags may not have been explicitly used.

It's important to mention that the authors of Linux Rekoobe have removed every attributive string from older variants in their new samples. This explains why string-based signatures have not been able to detect the new version of the malware.

In previous Rekoobe variants, the malware would initially collect some preliminary configuration saved in disk, masquerading this configuration to be a shared object, as previously reported by DrWeb researchers. However, in the new variants, the need for this configuration file has been completely removed, having hard-coded the subject artifacts previously dependent on the configuration file.

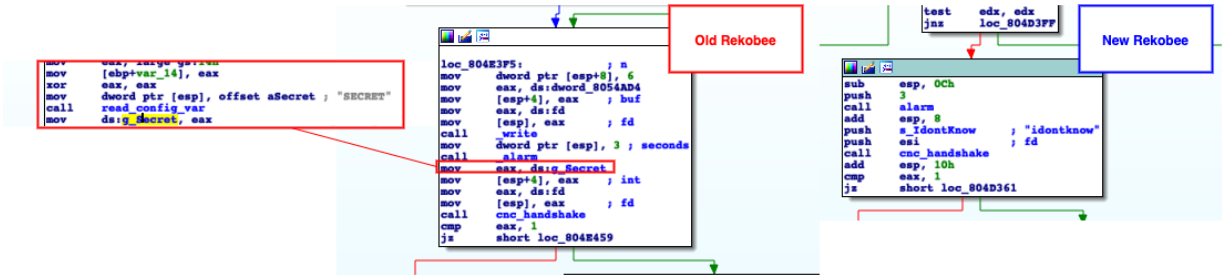
Main function comparison between Rekoobe variants

Among the hardcoded artifacts we can find the CNC IP and port, in addition to the shared 'secret' used to authenticate the CNC and client.

A new feature was added to the new Rekoobe variants to rewrite argv[0], as an attempt to rename the process name (as shown in the picture above) since some forensics tools do retrieve the process name from this location.

The names to rewrite argv[0] chosen by the authors were:

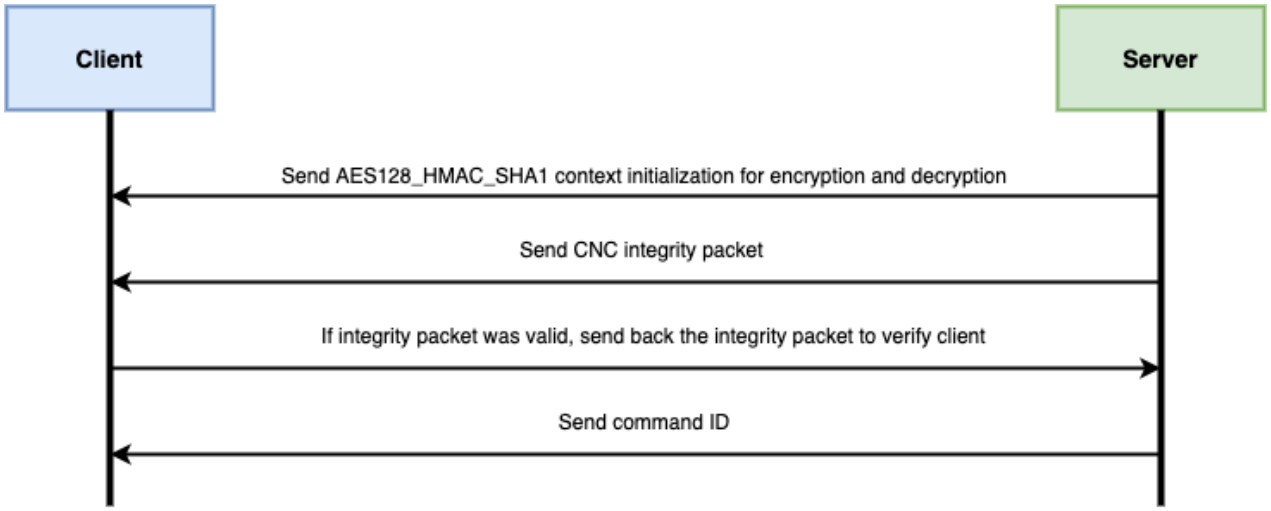
- /lib/sys/temd/sys/temd-udevd
- /bin/bash



Shared secret retrieval comparison between different Rekoobe versions

Another noticeable difference is that in previous variants a magic value was retrieved from the configuration file and sent to the CNC prior to the handshake mechanism. It seems that the authors removed this preliminary packet on the new Rekoobe variants.

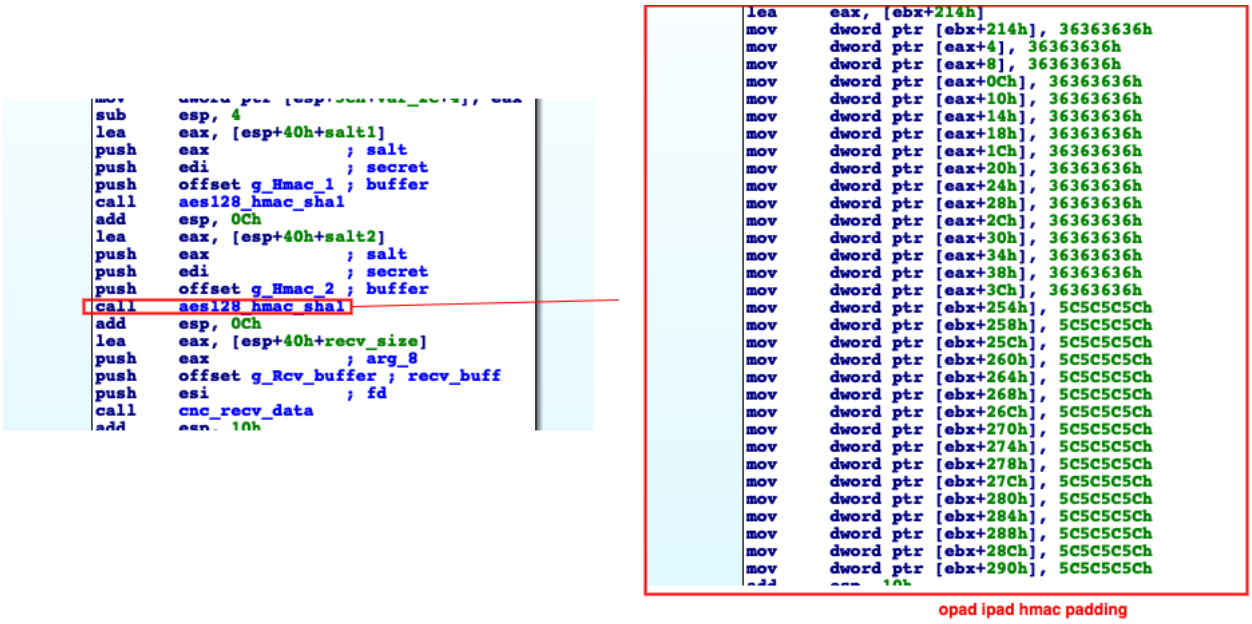
The remaining code resembles previous variants of Rekoobe. The following is a simplified overview of the network protocol used by the new Rekoobe samples, mostly shared with older Rekoobe variants:



The following is a brief description of the most relevant steps in the authentication mechanism:

1. The client will read a stream of 40-bytes from the server. This packet will be divided into two blocks of 20-bytes that will be utilized to initialize two AES128_HMAC_SHA1 contexts. The HMAC SHA1 pair will be generated against each of the 20-byte streams using a given shared secret (“idontknow” hardcoded string in newer variants) and they will be used as AES128 keys for encryption and decryption of future packets.

These computed keys will be used throughout the communication process to decrypt the received packets and to encrypt the packets to be sent by the client with different keys. This implies that if this preliminary packet is not retrieved and the shared secret is not known, then the remaining traffic will unlikely be decryptable.

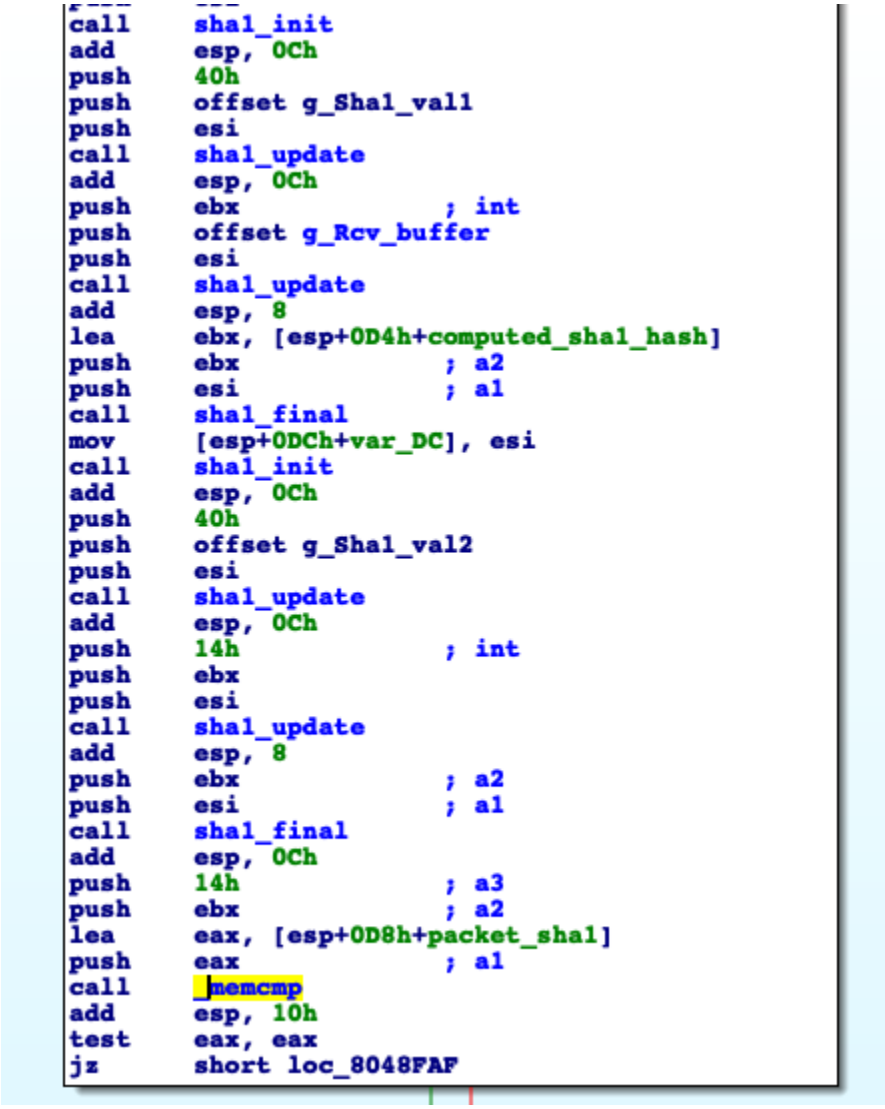


AES128_HMAC_SHA1 context generation

2. After AES128_HMAC_SHA1 context initialization is achieved there will be a CNC authentication procedure similar to a Challenge-Handshake Authentication Protocol (CHAP). The client will read an additional stream of 16-bytes that will then decrypt using AES128 with the corresponding and previously generated SHA1 hash as key. It will xor decode the stream to retrieve information of the next packet to be read. This process will be repeated every time the client and server sends an additional packet.

3. Every other packet will be subject to HMAC integrity checks to verify the integrity of the packets. Two layers of SHA1 will be computed against the subject packet with two additional salts. This computed SHA1 hash will be then compared with the first 20-bytes of the received packet which contains the pre-computed SHA1 of the AES encrypted packet's payload.

If the computed SHA1 hash does not match with the hardcoded hash delivered in the packet, the client will cease execution since it would imply that the integrity of that packet has been compromised; otherwise the client will proceed to AES128 decrypt the packet's payload and then apply an xor layout.



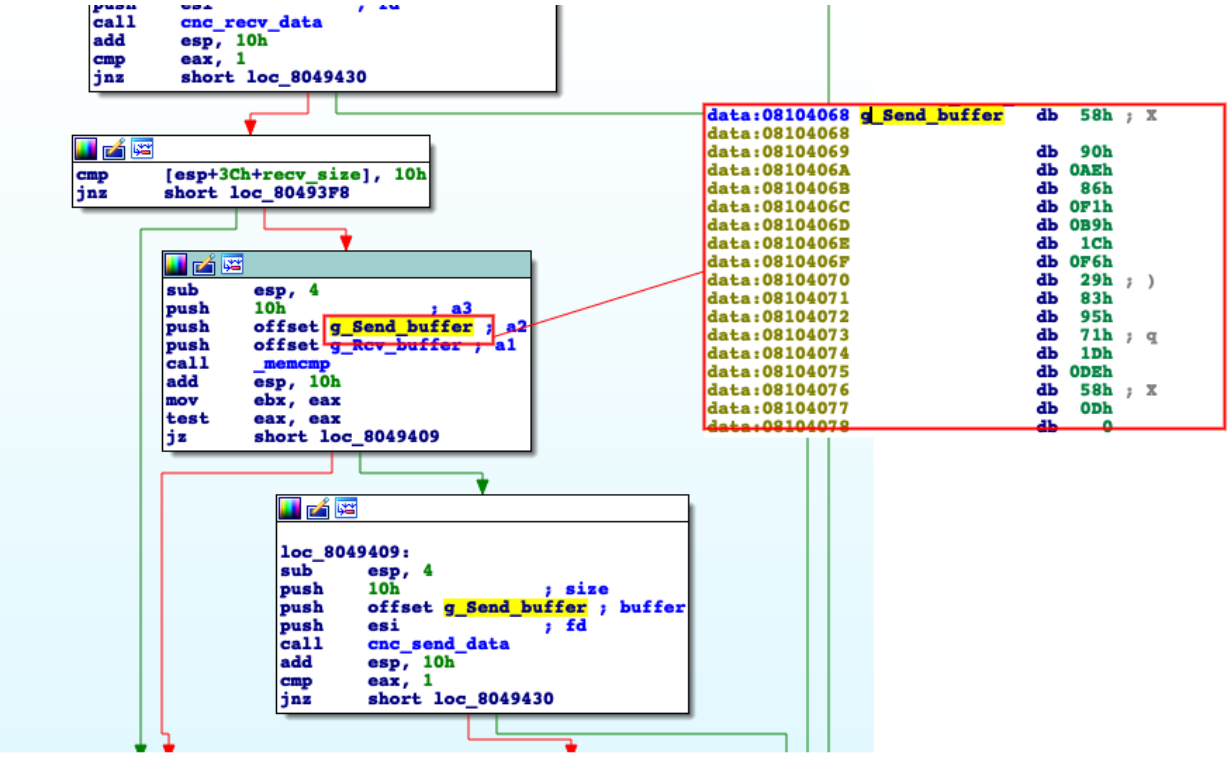
```
call    sha1_init
add     esp, 0Ch
push    40h
push    offset g_Sha1_val1
push    esi
call    sha1_update
add     esp, 0Ch
push    ebx             ; int
push    offset g_Rcv_buffer
push    esi
call    sha1_update
add     esp, 8
lea     ebx, [esp+0D4h+computed_sha1_hash]
push    ebx             ; a2
push    esi             ; a1
call    sha1_final
mov     [esp+0DCh+var_DC], esi
call    sha1_init
add     esp, 0Ch
push    40h
push    offset g_Sha1_val2
push    esi
call    sha1_update
add     esp, 0Ch
push    14h             ; int
push    ebx
push    esi
call    sha1_update
add     esp, 8
push    ebx             ; a2
push    esi             ; a1
call    sha1_final
add     esp, 0Ch
push    14h             ; a3
push    ebx             ; a2
lea     eax, [esp+0D8h+packet_sha1]
push    eax             ; a1
call    _memcmp
add     esp, 10h
test    eax, eax
jz      short loc_8048FAF
```

SHA1 scheme applied to encrypted packet and comparison to hardcoded SHA1

4. After this packet has been decrypted, the contents will be compared against a hard-coded byte sequence of 16-bytes in order to verify the integrity of the server.

Once again, if this sequence does not match, the handshake will fail.

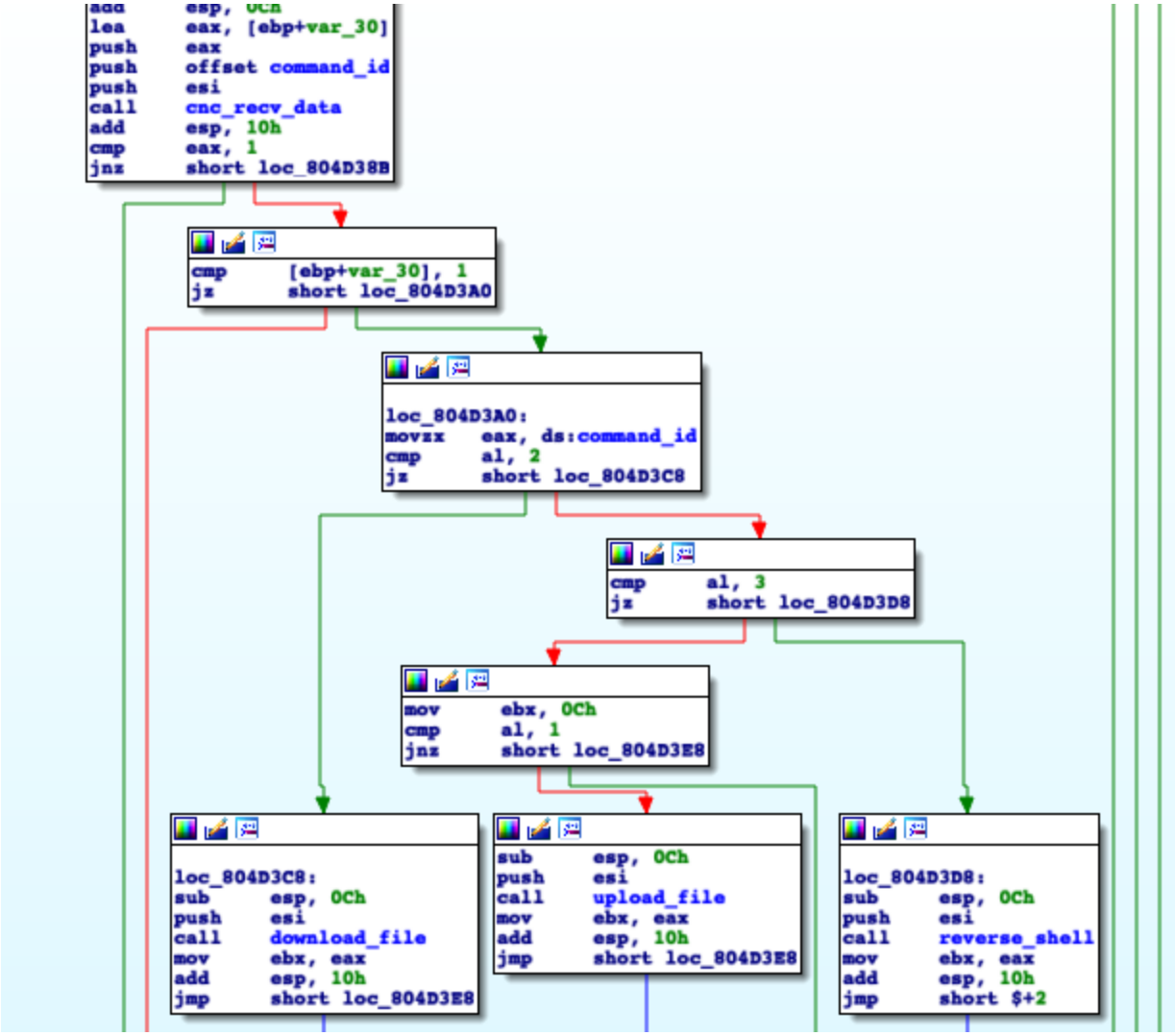
5. If the sequence is correct, then this byte sequence will be AES128 encrypted with the corresponding HMAC SHA1 as key, two layers of SHA1 will be computed with its correspondent salts against the encrypted payload, and this SHA1 hash will be written on the first 20-bytes of the packet itself to then be sent back to the CNC as the last verification step, in this case to verify the integrity of the client.



Hardcoded stream of 16-bytes the CNC and client authentication is based on

We assume this field would be an ideal choice for also identifying the campaign ID by the operators, although it seems the value of this field was shared with previous variants.

6. If the handshake was successful, the client will proceed to listen for a command from the CNC. The malware supports the same three different commands as it did in previous variants, those being file upload, file download, and a reverse shell. The following picture shows the command management implemented by the client after command packet has been successfully decrypted:



Conclusion

We have provided a brief technical analysis of the new Linux Rekoobe samples, highlighting some of the differences between these variants and previous samples. We have also provided an overview of the network protocol.

We have provided several reasons for why the malware has gone undetected, even though the code base of these new variants doesn't appear to have been heavily modified on a source code level from the original Rekoobe samples.

We do not believe this malware has been consistently operational since late 2015. In contrast, we believe the malware may have operated intermittently over small periods of time, since the newly discovered samples appear to be created in recent years, and there appears to be a gap in 2017 where additional Rekoobe samples have not been found.

We have indexed the code from the new Linux Rekoobe variants in our Genetic Malware Analysiss platform Intezer Analyze, and we have published a new YARA rule in order to help the community to detect this threat.

We expect Linux threats to pose a significant challenge to enterprise cloud security in the near future. We have just released our new cloud security product, Intezer Protect, which is based on our Genetic Malware Analysis technology and provides native cloud protection. For more information, visit https://www.intezer.com//intezer-protect/.

IOCs

80e5fec19843c32c6c3fc38aabdeb428c339b0dfce28023529144405b9c72b33
C9eb46d00e11acb354b518f725412b88c69cc511ec8d5bd3cb03c1740f8a2936
2e2dc0328f6c19b033bb19c24e59e354e519606958afb93fd8049d162a8e3edd
E63c2e35a41c51e33b246f5b60c5d1b8da0d8c50bf7ec592383b61818217e8d7
7148ae1ab45e17889915100fdc203fe7941d8e9b946d44a3989ab8baeb6066e1
1d0591049a65db6508a9517f72954541ef6b5a7fe9153c5edcb1bac1b70b991c
4B45E601D480124C38BE06A706F7D8F4
F34119A442651945D5EFB33DB8901D9B
7xin.bitscan[.]win
huawel[.]site
96.45.187[.]113
119.3.22[.]174

Ignacio Sanmillan
Nacho is a security researcher specializing in reverse engineering and malware analysis. Nacho plays a key role in Intezer\'s malware hunting and investigation operations, analyzing and documenting new undetected threats. Some of his latest research involves detecting new Linux malware and finding links between different threat actors. Nacho is an adept ELF researcher, having written numerous papers and conducting projects implementing state-of-the-art obfuscation and anti-analysis techniques in the ELF file format.