CYBER SECURITY DIVISION

MALWARE TECHNICAL INSIGHT

TURLA “Penquin_x64”

Last update: May 29th 2020

The information contained in this document is proprietary to Leonardo S.p.a. This document and the information contained herein may not be copied, reproduced, used or disclosed in whole or in part in any form without the prior written consent of Leonardo S.p.a.

© Copyright Leonardo S.p.a. – All rights reserved

LEONARDO

INDEX

EXECUTIVE SUMMARY……………………………………………..…………………………………….....1
1. SCENARIO……..……...………………………………………………………………………….....2
2. EVOLUTION OF THE “PENQUIN”………….....…………………………………………….......4
3. BUILD DATE ESTIMATION……………………………………………………………………...7
4. MALWARE CAPABILITIES………………………………………………………………………10
5. CONCLUSION……………………………………………………………………………………....22
6. MITRE ATT&CK TTPs……………………………………………………………………………..23
7. INDICATORS OF COMPROMISE……………………………………..……………....……..25
REFERENCES…………………………………………………………………………………..........28
APPENDIX A: BUILD DATE ESTIMATION………………………………………………………29

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

EXECUTIVE SUMMARY

The APT group known as Turla (aka Snake, Venomous Bear, Group 88, Uroburos, Waterbug and others) is a well-established collective that has been operational since at least 2004. Turla is one of the most advanced APTs in the world; it is famous for developing new and very advanced techniques to avoid detection and to ensure the persistence on the targeted network. This adversary is known for targeting, among the others, government, defense and education sectors all around the globe.

In December 2014, Kaspersky reported on a tool attributed to the Turla intrusion set used to target the Linux Operating System: they named it “**Penquin**” Turla. In 2017, more information about this threat was discovered[1]. Since then, almost three years have passed with no new information being discovered about this specific threat, until now.

In this technical report we analyse new samples of the toolkit spotted in April 2020, and dubbed “**Penquin_x64**”. We describe in depth the capabilities of this stealth backdoor, comparing it to the older known versions, and we also investigate the possible build dates of these samples. The threat actor put in place a considerable amount of effort to avoid the improper activation of the backdoor. In this report we shed light on the communication protocol, providing a tool to efficiently detect a “Penquin_x64” infection in enterprise networks.

The discovery of this Turla module still raises the same question of the 2014 Kaspersky’s report about how many other unknown Turla variants exist.

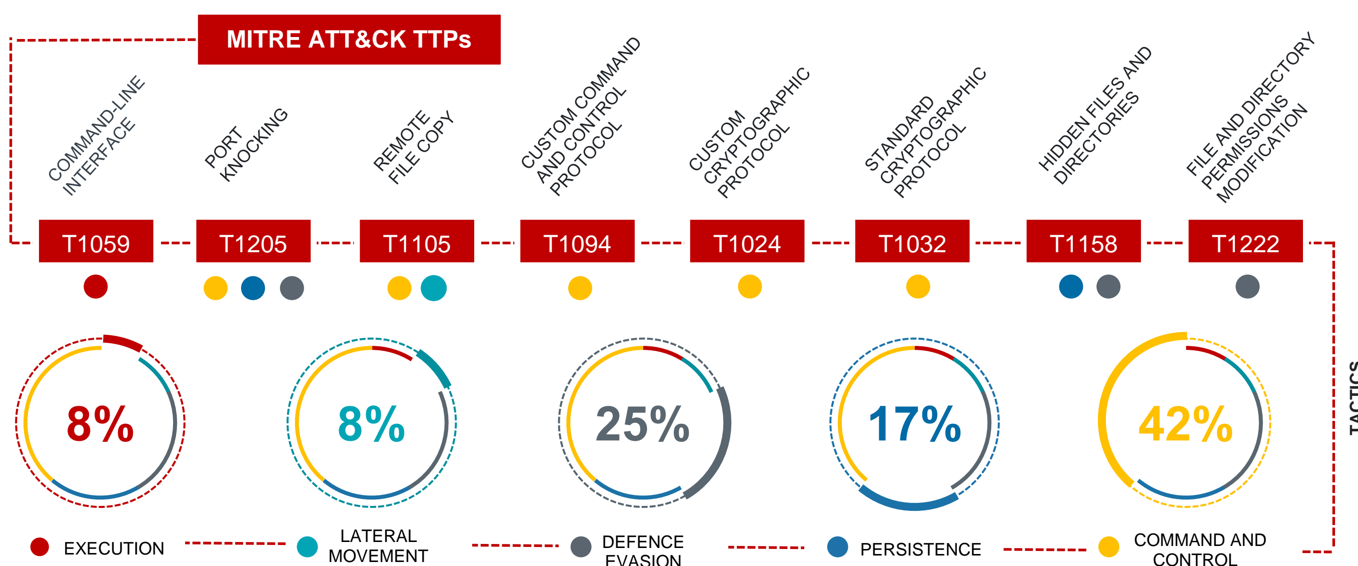

Figure 1 – Evaluation of “Penquin_x64” tactics and techniques

[1] PENQUIN’S MOONLIT MAZE: The Dawn of Nation-State Digital Espionage; by Juan Andres Guerrero-Saade (GReAT), Costin Raiu (GReAT), Daniel Moore (King’s College London), Thomas Rid (King’s College London). April 2017.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

1. SCENARIO

In April 2020, a few other samples similar to those detected in 2014 have been uploaded to a multi-scanner service. They let us discover the existence of an additional version of Turla “Penquin” that has not been analysed yet: we named it “Penquin_x64”.

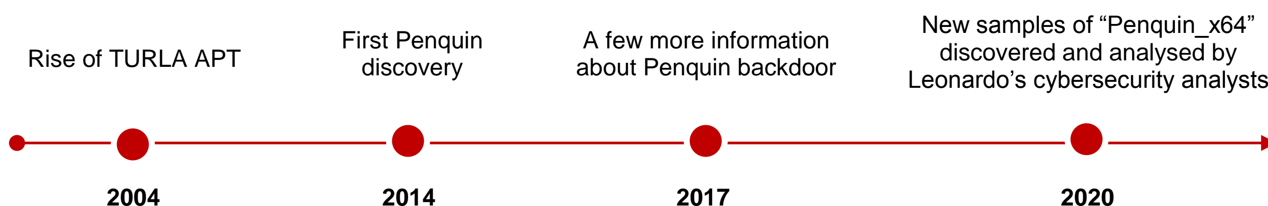

Figure 2 – Timeline of “Penquin” events

“Penquin_x64” is a stealth backdoor that, in addition, tries to hide itself from the eyes of the system administrators mimicking the “cron” binary, that is a system utility used to create pre-scheduled and periodic background jobs on Linux servers and clients.

The analysis we performed allowed us to fully understand all the malware capabilities and the network protocol used to activate the backdoor. Consequently, we built a python code that can be used by system administrators to check the presence of “Penquin_x64” on their networks.

“Penquin_x64” is a brand new piece of the Turla puzzle that remained uncovered for years. It is not possible to identify the exact date of build for this version. However, the analysis performed allowed us to estimate with high confidence that the most recent sample we discovered has been built between 2016 and 2017. We cannot state that this component is still in use by Turla, but there is the possibility that additional versions that target the Linux OS have not been discovered yet. Even though we do not have concrete data to support this statement, we can confirm that the most recent samples we found are still able to run on the current stable version of Ubuntu.

Considering that Turla is one of the most complex APTs in the world, and since it is highly likely that this malware has been used more often than what we currently know, we recommend to verify its presence on Linux OS servers and clients by using the tool we have built and the indicators of compromise attached to the end of this report. Based on our analysis, we have a high degree of confidence that this malware has been used as a post compromise tool on Virtual Private Servers that successively played a role in the Turla network infrastructure (Figure 3).

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

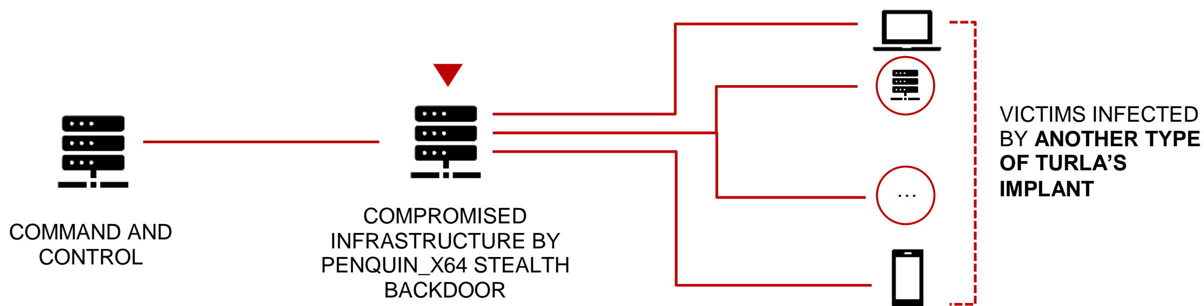

Figure 3 – Hypothesis about the role of the Penquin family within the Turla’s infrastructure

The behavior of this implant suggests that it starts to operate at the “Installation” phase of the Cyber Kill Chain®, and, successively, once the stealth backdoor is activated by the attacker, it operates at the “Command and Control” phase also (Figure 4). At the same time, we do not exclude that the “Actions on Objectives” phase could be up and running on the compromised infrastructure or on other infrastructures that are reachable from the infected one. However, since we did not have the chance to investigate an actual incident involving this implant, it is not possible to be confident about this last statement.

Figure 4 – Penquin killchain phases

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

2. EVOLUTION OF THE "PENQUIN"

The birth of the "Penquin" toolkit goes back years, at the dawn of Nation-State digital espionage. Two important reports of extremely high quality about this threat have been published by Kaspersky GReAT, in 2014 and in 2017 respectively: "The Penquin Turla" [1] and "Penquins Moonlit Maze" [2]. While the former is a technical analysis of a set of samples found in the wild, the latter gives a very broad but specific context about Turla intrusion set, and how it evolved since the early 2000s. A third important report that mentioned features of the "Penquin" family has been published by the Swiss CERT in 2016 detailing a prolonged campaign against the defence contractor Ruag [3].

In April 2020, we detected three new "Penquin" samples. With respect to those already known, these new samples target 64 bits architectures. We analysed them, and we compared them with the old ones. In the rest of this document we will report the result of this deep technical analysis that allowed us to understand much more about this specific threat. In the following section we will describe the three different versions of "Penquin" we have been able to identify. We will provide an estimation of the build date of the new samples in Section 2, while in Section 3 we will report on the capabilities of this malware, also providing a discussion about differences and commonalities of the different versions.

2.1 "Penquin" versions

Having at our disposal new and old samples, we have been able to categorise them in three different sets that have many commonalities, but also a few important differences.

1. The first set, that we will call "Penquin", is the one firstly analysed by Kaspersky in 2014. One of its peculiar features is the sniffing technique initially attributed to the cd00r project (http://www.phenoelit.org/stuff/cd00r.c) [1], and successively to LOKI2 [2]. This feature allows the samples belonging to this set to become stealth backdoors that are difficult to detect, and that can be potentially used as post-compromise tools. Indeed, the sample will connect back to a C2 only after a sniffed packet matches a series of predefined mathematical conditions. Note that the IP address of the C2 is encoded in the activation packet, and therefore it is not possible to foresee it in advance.

2. We named the second set of samples "Penquin_2.0". One sample belonging to this set has been detected by Kaspersky after the first publication of their 2014 report, which has been updated to include it. The other three samples we know about have been first seen in the wild in 2014, 2016 and 2017 respectively. All of them have been included among the indicators of compromise reported in [2]. "Penquin_2.0" shares most of the capabilities with his ancestor, but there is a fundamental difference: it doesn't use the network sniffing technique anymore. On the contrary, it includes a hardcoded

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Command and Control server, specified by IP address and port, that is immediately contacted. This IP address belonged to LunaSat ISP. In 2015, it already came out that several other IP addresses belonging to the same Lebanese satellite Internet Service Provider have been leveraged by Turla.

3 The third set of samples is the most interesting one, as a matter of fact it has not been publicly disclosed yet: we named it “Penquin_x64”. Two samples belonging to this set have been submitted to VirusTotal service in April 2020 from the Czech Republic and Italy respectively. We have reason to believe that the sample submitted from Italy is related to an incident investigation that could reveal more details about Turla TTPs. We are not aware of the identity of who submitted it, even though we are based in the same Country. The two samples submitted in 2020 let us discover a third one that has been submitted to the Virus Total service in December 2017.

All “Penquin_x64” samples are compiled to work on 64 bits architectures and they embed a legitimate linux binary trying to mimic: “cron”. Furthermore, they implement the same sort of port knocking technique[2] of the first “Penquin” version, using a different filter to activate the backdoor. In Table 1, we provide a comparison of the main features of the three sets of samples we analysed, for the STIX 2.1 standard. More details about commonalities and differences of the three malware versions will be reported in Section 3.

Features	**Malware Versions**			**Description**
Name	Penquin	Penquin_2.0	Penquin_x64	
Estimated Date of build	-	-	After mid-2016	The date of build that we estimate
First Seen ITW	November 2014	December 2014	December 2017	Date that the malware instance or family was first seen.
Architecture execution envs	x86	x86	x86-64	The processor architectures that the malware instance or family is executable on.
Implementation languages	c	c	c	The programming language(s) used to implement the malware instance or family.
Capabilities	controls-local-machine	controls-local-machine, communicates-with-c2	controls-local-machine	Specifies any capabilities identified for the malware instance or family.

[2] Since it does not open a local port but it connects back to a C2 specified in the incoming packet, formally it should not be considered an actual port knocking. However, the way it is used recalls it.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Features	**Malware Versions**			**Description**
Name	Penquin	Penquin_2.0	Penquin_x64	
Kill-chain Phases	Installation, C2	Installation, C2	Installation, C2	The list of kill chain phases for which this Malware instance can be used.
Malware Types	backdoor, remote-access-trojan	backdoor, remote-access-trojan	backdoor, remote-access-trojan	The type of malware being described.
OS execution envs	linux	linux	linux	The operating systems that the malware family or malware instance is executable on.
Malware Instances (SHA256)	1d5e4466a6c5723cd30caf8b1c3d33d1a3d4c94c25e2ebe186c02b8b41daf905, 3e138e4e34c6eed3506efc7c805fce19af13bd62aeb35544f81f111e83b5d0d4	8856a68d95e4e79301779770a83e3fad8f122b849a9e9e31cfe06bf3418fa667, 2dabb2c5c04da560a6b56dbaa565d1eab8189d1fa4a85557a22157877065ea08, 5a204263cac112318cd162f1c372437abf7f2092902b05e943e8784869629dd8, d49690ccb82ff9d42d3ee9d7da693fd7d302734562de088e9298413d56b86ed0	8ccc081d4940c5d8aa6b782c16ed82528c0885bbb08210a8d0a8c519c54215bc 67d9556c695ef6c51abf6fbab17acb3466e3149cf4d20cb64d6d34dc969b6502 d9f2467ff11efae921ec83e074e4f8d2eac7881d76bff60a872a801bd45ce3d5	Specific malware samples belonging to the corresponding families
C2 Domains	news-bbc.podzone[.]org			C2 hardcoded domains
C2 IP and Ports		82.146.175[.]43:1773		C2 hardcoded IP addresses and ports

Table 1 - Features of the three different “Penquin” versions discovered

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

3. BUILD DATE ESTIMATION

Since the first confirmed reporting of Turla “Penquin” dates back to the end of 2014, we considered extremely important for the current analysis to estimate the build date of the new samples we found to be extremely important to our analysis. Indeed, when we started this analysis, one of the first hypotheses we considered was that the new samples we detected could have been built before the publication of the first Kaspersky’s report in 2014.

In comparison to the Windows Portable Executable which has a compile timestamp field, the ELF file format does not provide any corresponding field that can be used for such a purpose. Therefore, we considered several different approaches, which are fully described in Appendix A, such as analyzing the EI_ABIVERSION, or the epoch of the statically linked libraries. As for the “Penquin_x64” set, the approach that in our opinion turned out to be the most reliable is based on the analysis of the embedded cron binaries, which is detailed in the following subsection.

3.1 Estimated date of the embedded binaries

Analyzing the “Penquin_x64” malware set, we found that these samples embed and execute the cron binary, used in linux OS to create pre-scheduled and periodic background jobs. Indeed, samples belonging to this set try to mimic cron in order to hide themselves from the eyes of the user or system administrator, and they need the actual version of the binary to run it after their execution. To support this hypothesis, we immediately verified that the cron binary each malware embeds was not trojanized, and that it was launched by the malware as a child process. After extracting the embedded cron from the samples, we also noticed that each sample embeds a different version of the binary.

Therefore, we analysed the ABI field of the cron binaries we recovered, and we found the indicative time windows they have been built. It was discovered that they included three different versions of cron, that have been built in three different time windows, but still the information that we have got was too inexact. Table 2 reports the outcome of this first analysis.

Cron hashes	ABI	GCC	Year
3309e8f29e53d56d177ab2ad4b814cd3d8215944a0bbe233e498766 1d1db5afd	2.6.32	>= 4.9.1 < 7.2.0	>=2014 <2017
dc17065fac8ce24aa6c344a45f12a0d0e3e4928d23b8aa6edad769b2 4f4c7a39	2.6.18	> 4.4.4 < 4.8.2	>2010 <2013
3609f24f314d2b95f9d607be8205ed8722b1457897d1eb222d950e38f 84aa728	2.6.24	>= 4.8.2 < 4.9.1	>=2013 <2014

Table 2 - Cron build date estimation based on the ABI field of the ELF

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

In addition to the analysis of the ABI field of the cron instances, we also tried a different approach that is based on the following simple hypothesis: if these binaries have not been re-compiled by the attacker, then it is probable they have been taken from a Linux distribution. Therefore, we downloaded the cron binary packages of several distributions, and we found with a simple sha256 comparison that those we were interested in have been released in Ubuntu and CentOS.

Furthermore, one of the three cron binaries has been released with Ubuntu 16.04 and 17.04, in April 2016 and April 2017 respectively. Therefore, it is with high confidence that we can state that “Penquin_x64” was still being developed during those years, and that the last version we currently know about was compiled after April 2016. Table 3 summarises the results of the build date estimation based on this analysis.

Cron SHA256	Penquin_x64 SHA256 embedding the cron	Possible Linux Distributions	First known release date
3309e8f29e53d56d177ab2ad4b814cd3d8215944a0bbe233e4987661d1db5afd	d9f2467ff11efae921ec83e074e4f8d2eac7881d76bff60a872a801bd45ce3d5	>= Ubuntu 1604 <= Ubuntu 1704	April 2016 - April 2017
dc17065fac8ce24aa6c344a45f12a0d0e3e4928d23b8aa6edad769b24f4c7a39	67d9556c695ef6c51abf6fbab17acb3466e3149cf4d20cb64d6d34dc969b6502	Centos 6.7 Centos 6.8	Sep 2015- July 2016
3609f24f314d2b95f9d607be8205ed8722b1457897d1eb222d950e38f84aa728	8ccc081d4940c5d8aa6b782c16ed82528c0885bbb08210a8d0a8c519c54215bc	Ubuntu 13.10 Ubuntu 14.04	October 2013 - April 2014

Table 3 - Release date of the Linux distributions that include the cron binaries found in “Penquin_x64”

It can be observed that the results reported in Table 3 do not intersect perfectly those reported in Table 2, but from our analysis we believe this is the most reliable finding to give a lower bound to the build date of each sample belonging to the “Penquin_x64” set (Figure 5).

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

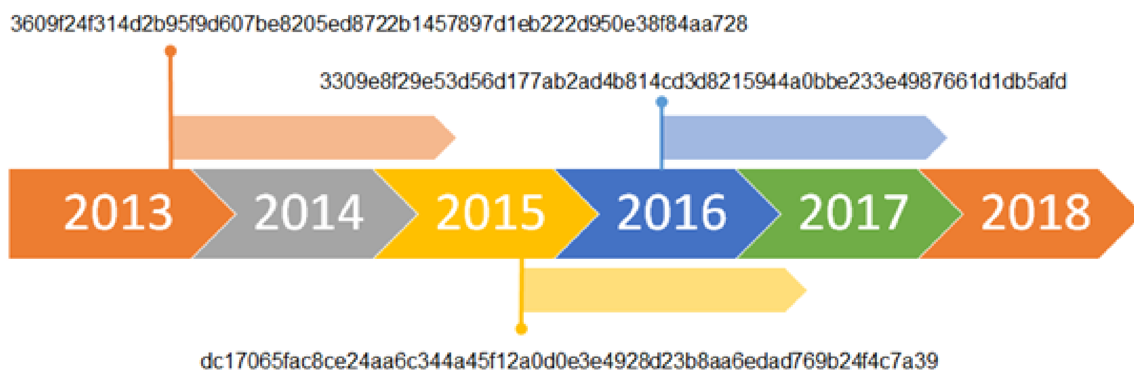

Figure 5 - Build date lower bound estimation for “Penquin_x64” samples, based on the first known date of release of the embedded cron.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

4. MALWARE CAPABILITIES

In order to understand all the differences among the three Turla “Penquin” versions we have identified, we conducted deep analysis of all the available samples, including those already reported in 2014 and 2017. A very important piece of code, that was not previously described in any analysis we know about, is used by the three malware sets:

```
result = trybuiltin(a2, v9, v10);
if ( !result ):
    result = runcmd((int)v4, v9, (int)v10, a4);
```

The runcmd function is the one described by Kaspersky: “[the malware] reports its own PID and IP to the remote address, and starts an endless loop for receiving remote commands. When a command arrives, it is executed with a ‘/bin/sh -c’ script” [1]. However, the code snippet above highlights that before calling the runcmd function (that uses the /bin/sh technique previously described), it checks the result of another function, called trybuiltin in one of the rare samples that didn’t have the symbols stripped.

The trybuiltin function checks the received parameters against a hardcoded data structure, named cmdtab, and calls the corresponding command implemented in the malware, without resorting to the command line. These commands are specified through command codes that are mapped to a list of functions (the full list with a detailed description is reported in the Section 3.3), that include: a do_upload function to send back to the C2 a file, a do_download function to download a remote file from the C2, and a do_exec function to download and execute a file. These commands add to the known stealth backdoor capabilities of this malware also the capabilities of a remote access trojan.

Additional implemented commands are: do_vsupload, do_vsdownlod, do_vslist, do_vsdelete, do_vsshutdown and do_vstat. All the commands preceded by the string do_vs have something very important in common: they can be executed on a remote IP address and port that is received as input parameter, that is potentially, and highly probably, different from the C2. These commands will therefore establish a new encrypted connection with a third peer that is specified by the attacker. The very detailed Ruag report published in 2016 already mentioned this capability of “Penquin”, but it didn’t give too many details. In particular, the report by the GovCERT.ch says:

> It [the malware] even contains a feature to access a linear filesystem at a third IP address (like a file repository), but we never found the corresponding server implementation [3].

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

As for the do_vsdelete and the do_vsshutdown functions, they are used to send two commands to the remote peer. The binary code available in the malware does not allow to confirm what should be the result on the remote peer, but their names suggest that they will respectively delete and shutdown something. It is highly possible that the do_vsdelete deletes a file on the remote peer, and that the do_vsshutdown executes the shutdown of the machine or kills the malware process. The do_vslist function lists the files of the peer and sends back to the C2 a list with the following information: Description, File Name, Size and Status. We presume at this time, that the presence of fields such as Status and Description may be related to some peer to peer file system functionalities.

4.1 “Penquins” differences and commonalities

The three malware sets have the same code base. As a matter of fact, they share a lot of binary code. For example, one of the features that remained unchanged among all the three malware sets is related to the encryption algorithm used to obfuscate the strings included in the binary. This kind of technique is also peculiar of other tools used by Turla:

```
index = 0;
do{
 _decrypted__buffer[index] = _xored__buffer[index] ^ (index + 0x5);
  ++index;
}
while ( index <= _xored__buffer_size);
```

Another commonality of the three sets is related to the encryption algorithm used to communicate through the network. Indeed, all the communications are encrypted with the BlowFish algorithm, and a symmetric key exchanged with Diffie Hellman. Furthermore, the values that are used for the initialisation vector are always the same.

```
int __cdecl bf_crypt(int len, int a2, int key, int a4)
{
  int iv; // eax
  int num = 0; // [esp+20h] [ebp-8h]

  if ( a4 )
    iv = key + 0x386C;
  else
    iv = key + 0x3864;
  BF_ofb64_encrypt((_BYTE *)a2, (_BYTE *)a2, len, key + 10268, iv, &num);
  return len;
}
```

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Commonalities are more noticeable when considering the set of commands they can receive from the network, and the corresponding binary code to handle and execute them. “Penquin” and “Penquin_2.0” share exactly the same set of commands, together with the main function that handles all of them. These commands are organised in a data structure that the malware authors called cmdtab, previously mentioned, which will be fully described in Section 3.2. These two malware versions, after the initialisation phase, basically share the same code to process the received commands:

```
while ( !feof(v10) )
 {
  memset(&v20, 0, 10240);
  receiveData(srvfd, &v20, 10240, (int)&CStruct);
  command(&v20, (int)&cmdtab, 1, srvfd);
  fflush((int)v10);
  sendEndOfData(srvfd, (int)&CStruct);
 }
```

Only a small part of code is slightly different, that is mainly related to the way “Penquin_2.0” recovers the hardcoded IP address and port of the C2 it has to connect to. As a matter of fact, “Penquin_2.0” is the only one that has the C2 IP address and port hardcoded in itself at the end of the file.

As for the stealth backdoor capabilities, “Penquin_x64” and “Penquin” use the same technique to interact with the attacker: they leverage the *libpcap* library to sniff the network traffic, applying a filter to match a ‘*magic’ packet*. The filters they use are slightly different (they are reported in detail in Section 3.3). On the contrary, “Penquin_2.0” malware set does not use the *libpcap* library to sniff the network traffic and it does not require root privileges to run. The IP address of the C2 is hardcoded at the very end of each sample. However, the only IP address and port we have been able to recover is equal to 82.146.175[.]43:1773. This IP address belonged to LunaSat ISP, a Lebanese satellite Internet Service Provider that has been used by Turla also in other cases [4]. We think that other samples belonging to this set have been padded or resubmitted with the last portion of the binary modified, producing therefore additional IP addresses that may not be really used by the attacker. For this reason, we do not report them as indicators of compromise.

“Penquin_x64” is the only one that embeds the cron binary. With respect to the other samples, several strings have been encrypted or removed, probably to avoid detection. Furthermore, some portions of code have been removed. “Penquin_x64” uses the do_start method as the main function. This function was present in the other versions already, available in cmdtab structure, but it has been slightly modified. Indeed, in “Penquin” and “Penquin_2.0”, it was used to download an executable from the network, to save it to /tmp/.xdfg, and to execute and

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

delete it after the execution. “Penquin_x64” uses a different way to parse the incoming commands, and the do_start function seems to be related to this new feature. In particular, the cmdtab data structure containing the codes of the commands to execute and the corresponding functions to call is not present anymore, even though the implementation of the functions is still there. Therefore, the do_start function downloads a file (not an executable anymore) from the network, and it saves it to /root/.session.

The sample expects that the .session file must be encoded with the linux utility named uuencode, generally used to transmit binary files over transmission mediums that do not support other than simple ASCII data. The resulting encoding will include both the binary file and its metadata, such as the original filename and the permissions. “Penquin_x64” embeds the implementation of the counterpart of uuencode, that is the uudecode utility, that is then used to decode the file, leveraging the included metadata. Since the binary code of “Penquin_x64” expects to execute a file named /root/.hsperfdata just after the decoding, we guess that this is the original name of the file that is received from the network, encoded in /root/.session. The file /root/.hsperfdata is then executed through the command ‘/bin/sh -c’, capturing the standard output that is then redirected to a file.

4.2 Commands description

Table 4 reports the full list of commands that are built-in in to the malware code of the three malware versions. This information is directly derived from the cmdtab data structure that is present in “Penquin” and “Penquin_2.0”. Indeed, this data structure contains the command code, the corresponding function to execute when such a command code is received, the minimum and the maximum number of input parameters allowed. As for “Penquin_x64”, the cmdtab data structure has been removed, but all the implemented functions are still present.

Function name	***Command code***	***Description***	***Num of input params***
do_exit	quit	It makes the malware exit returning 0. It doesn’t return any feedback to the C2.	0
do_setenv	setenv	It adds a new environment variable or modifies it if it does not exist. It does not return any feedback to the C2.	2
do_cd	0x80D13E9	It re-implements the ‘cd’ command logic. HOME value is set to /tmp during the malware init code.	0 or 1

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Function name	*Command code*	*Description*	*Num of input params*
do_download	___123!@#	It opens a local file in write mode. Its path is passed by the C2 as an input parameter. If there are no errors, it sends back to the C2 the message "__we_are_happy__", and receives the content of the file to write on the disk.	1 or 2
do_upload	___456$$$	It opens a local file in read mode. If there are no errors, it sends back to the C2 the message “__we_are_happy__”, followed by the file content.	1 or 2
do_start	___789&*(	It downloads an executable in /tmp/.xdfg. It changes its permissions by adding the executable flag, and then it starts it by using a fork and the arguments passed as input parameters. Finally, the executable is deleted from the file system. [This function has been modified in “Penquin_x64”]	1 to 24
do_exec	___243)!#	It downloads an executable in /tmp/.xdfg (“Penquin” and “Penquin_2.0”) or in /tmp/.sync.pid (Penquin_x64). It changes its permissions by adding the executable flag, and then it runs it by using a fork the arguments passed as input parameters. Finally, the executable is deleted from the file system.	0 to 24
do_vslist	___4396#?	It establishes a new encrypted connection to an IP and port passed as argument from the network, and it sends to this peer the message “0x10”. Note that IP and port of the peer can be different from the C2 that sent the input command. Successively, it sends back to the C2 files information received from the remote peer that are organized with the following fields: Description, FileName, Size, and Status. Finally, it sends back to the C2 the message “Done!\n”. The status can have one of the following values: “Ok”, “Size mismatch”,"Unknown err","Remote VS is empty !\n".	2
do_vsupload	___43()*#	It establishes a new encrypted connection to an IP and port passed as argument from the network, and it sends to this peer the message “0x11”. Successively, it reads a local file that is then sent to the peer through the established SSL channel, encrypted with the BlowFish algorithm. It checks different error cases and when the command execution ends, it sends back to the C2 the message “Done!\n”.	3 to 4

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Function name	*Command code*	*Description*	*Num of input params*
do_vsdownlod	____526((@	It establishes a new encrypted connection to an IP and port passed as argument from the network, and it sends to this peer the message “0x12”. Successively, it downloads a remote file in chunks (1024 bytes each) to a local file, which path is passed as argument. If the number of downloaded bytes is zero, the local file is removed. Finally, it sends back to the C2 the message “Done!\n” once the operation is completed.	3 to 4
do_vsdelete	____*32)(2	It establishes a new encrypted connection to an IP and port passed as argument from the network, and it sends to this peer the message “0x13”, followed by another argument. At the end, it sends back to the C2 the message “Done!\n”. We speculate that one of the arguments is the remote path of the file to delete.	3
do_vsshutdown	____(@#*$5	It establishes a new encrypted connection to a remote peer (IP and port passed as arguments), and it sends to it the message “0x14”. Successively, it sends the message “Done!\n” to the C2.	2
do_vsstat	____@ͨ	It establishes a new encrypted connection to a remote peer (IP and port passed as arguments), and it sends to it the message “0x15”. It receives data from the peer and successively, it sends back to the C2 the peer file system type and information about its available disk space.	2

Table 4 - List of commands that can be executed

4.3 Magic packets for remote command execution

Once “Penquin_x64” completes the startup sequence, it is not ready to receive commands from the attacker yet. It does not start a reverse shell, nor does it try to listen for incoming connections on any local port of the infected host. It passively starts sniffing the incoming data on *eth0* interface, searching for the right packet that matches all the required conditions. The threat actor that is behind this tool, has taken considerable effort (and a lot of math) in such a small piece of code in order to make sure that only properly crafted packets can trigger an action on the running sample. This is also due to the fact that, encoded in the activation packet, lies the IP address that the sample will try to contact to if all the conditions are matched. In this section we will describe such conditions and explain how to build the *magic* packet to make the Penquin happy (cit.).

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

The first condition, that we call **pcap_condition**, is the simplest one (it has also been described in previous analysis [1]): it consists of a pcap filter which is implemented through the *libpcap* library that is statically linked in the binary. However, the filter applied by the stealth backdoor capability of “Penquin” and “Penquin_x64” is slightly different as reported below:

“Penquin” magic packet filter

```
(tcp[8:4] & 0xe007ffff = 0xe003bebe) or (udp[12:4] & 0xe007ffff = 0xe003bebe)
(tcp[8:4] & 0xe007ffff = 0x1bebe) or (udp[12:4] & 0xe007ffff = 0x1bebe)
```

“Penquin_x64” magic packet filter

```
(tcp[8:4] & 0xe007ffff = 0x6005bdbd) or (udp[12:4] & 0xe007ffff = 0x6005bdbd)
```

It can be noticed that in both cases the malware checks for a specific four bytes value of the incoming packet. TCP or, alternatively, UDP packets are considered by the filters. However, since both constraints and actions taken by the sample are the same in either case, for the sake of simplicity, in the rest of this analysis we will focus on UDP packets only.

Even though the bitmask remained the same for all versions (0xe007ffff), the result value has changed between different releases (0xe003bebe or 0x1bebe became 0x6005bdbd). As will be detailed below, some bits of that DWORD are used to generate the final IP address that the sample will try to connect to. In particular, this value influences four bits of the two most significant bytes of the C2 IP address. We have no evidence of the motivation that led the threat actor to change such bits.

However, the pcap_condition is only the tip of the iceberg since the constraints also involve the first 8 bytes of the packet (namely the **first dword** and the **second dword**) together with the source port used to send the packet. It is worth noting that this part of “Penquin_x64” didn’t change too much from the first version. A comparison between the control flow graph of “Penquin” and “Penquin_x64” is reported in Figure 6.

In the first part of the control flow graph, highlighted in Figure 6, both versions filter out protocols different from TCP or UDP. The central portion seems to be different, but a closer look shows that they are very similar since the 32 bits version uses a call to an external function, while the 64 bits version uses the inlining of the same function. The bottom part is again very similar, and it is related to the payload and source port constraints.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Transport protocol constraints

Payload and source port constraints

Penquin_x64

Penquin

Figure 6 – Comparison of the control flow graph of “Penquin” and “Penquin_x64” related to the magic packet that activates the backdoor (1d5e4466a6c5723cd30caf8b1c3d33d1a3d4c94c25e2ebe186c02b8b41daf905 vs 67d9556c695ef6c51abf6fbab17acb3466e3149cf4d20cb64d6d34dc969b6502).

The commented C code that we generated during the analysis is reported in the following code snippet. This piece of code replicates the behaviour of Penquin when it receives a packet that passes the pcap_condition. In particular, it is used by the malware to retrieve the IP address it has to connect back to. Note that in case the packet does not satisfy the mathematical constraints reported in this code snippet, then the sample will not open back any connection.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

At first sight, it is not the easiest piece of C code you can find. The relation between the final IP address and the input values depends on several bitwise operations that must be fully understood in order to make further considerations.

```
UINT64 get_final_ip(UINT32 first_dword, UINT32 second_dword, UINT16 src_port)
{
    UINT16 src_port_ror;
    BYTE final_check_1;
    BYTE final_check_2;
    UINT32 final_check_3;
    UINT32 final_check_5;
    UINT32 ip_final;
    UINT64 result;

    src_port_ror = __ROR2__(src_port, 8);

    /* Calculates conditions from input data */
    final_check_1 = ((__ROR2__(second_dword, 8) & 0xE000) >> 10) | __ROR2__(second_dword, 8) & 7;
    final_check_2 = (__ROR2__(second_dword, 8) & 0x18) >> 3;
    final_check_3 = ((first_dword & 0x200) << 6) | 2 * (first_dword & 0x1C0) | src_port_ror & 0x7C7F;
    final_check_5 = (first_dword & 0x3C00000u) >> 22;

    ip_final = ((src_port_ror & 0x8000) >> 6) | ((src_port_ror & 0x380) >> 1) |
((__ROR2__(second_dword, 8) & 0x1E0) << 17) | first_dword & 0xFC3FFC3F;

    /* Verify the conditions on the input data */
    if (
        ((unsigned __int8)final_check_5 | (unsigned __int8)((__ROR2__(second_dword, 8) & 0x1E00) >>
5))
        !=
        ((unsigned __int8)(HIBYTE(ip_final) ^ BYTE1(ip_final) ^ BYTE1(final_check_3)) ^
            (unsigned __int8)(BYTE2(ip_final) ^ ip_final ^ final_check_3 ^
            (final_check_1 | (final_check_2 << 6))))
       )
        result = 0;
    else
        result = ip_final;
    return result;
}
```

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

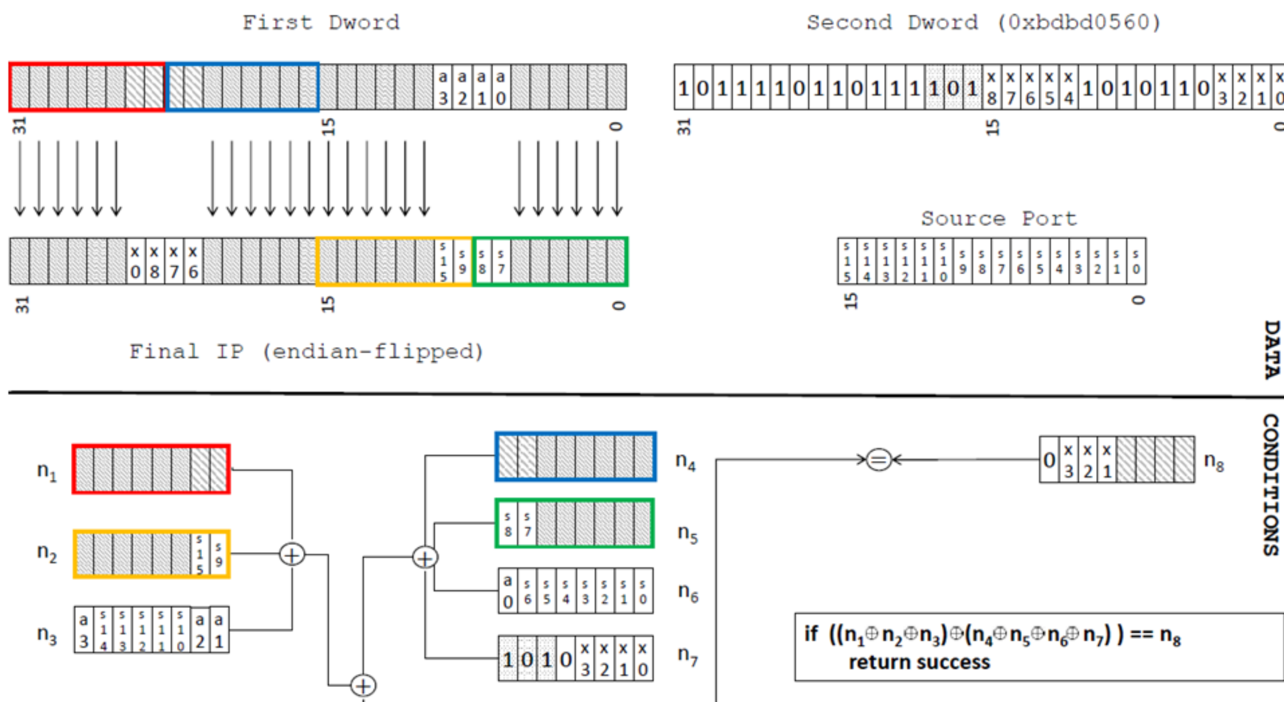

Figure 7 - Representation of the checks made by "Penquin" in order to retrieve the command and control IP address from a sniffed incoming activation packet.

A different representation of the commented C code is reported in Figure 7 where the relation between the input packets and the final destination IP address, as well as the conditions required by Penquin, are summarised. Every field is represented with a bit-level granularity since most of the operations are performed bitwise and do not align to the byte boundaries. On top, there is the input data (that consists of two DWORDS and the source port) and the final IP address. The first DWORD is not considered by the pcap_condition so it can be set to an arbitrary value. The second DWORD should appear familiar since it corresponds to the value specified in the pcap filter (for example, 0xbdbd0560 for "Penquin_x64" samples). This DWORD is subject to the constraints of the pcap_condition. In the figure, the values of "Penquin_x64" are reported; the bits marked with an 'x' can be chosen as desired, as long as they match the conditions represented in the bottom section of the figure.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

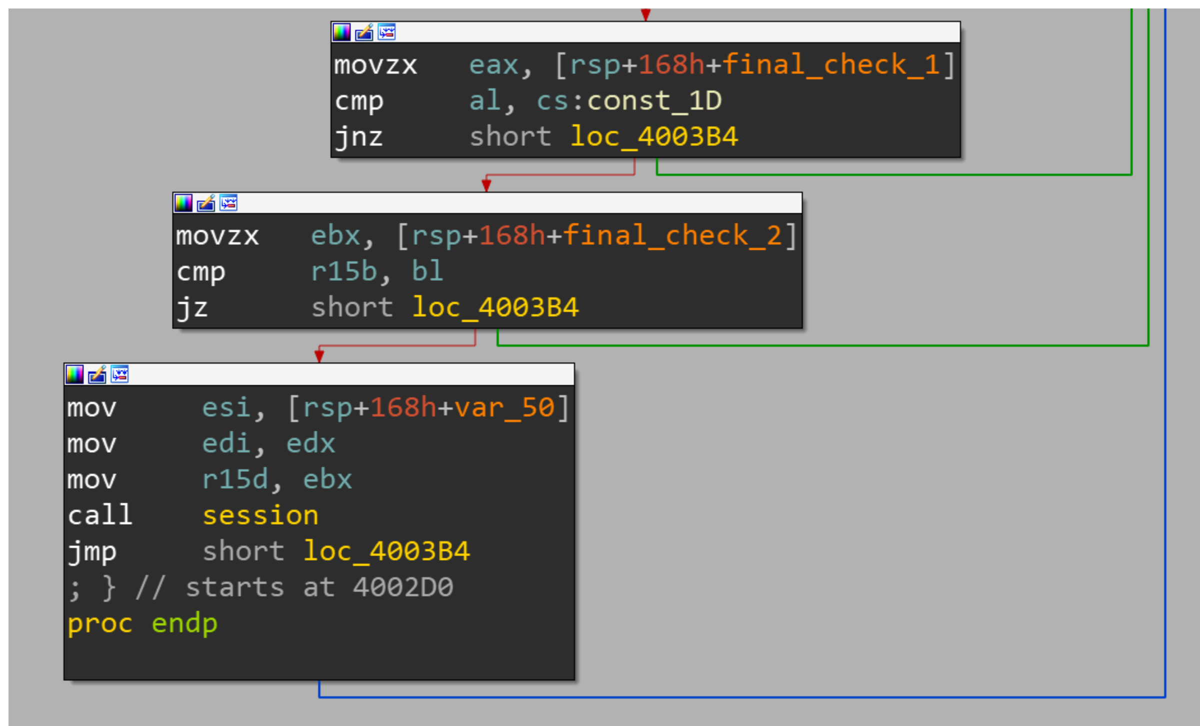

Figure 8 - Portion of code of “Penquin_x64” executed when a valid incoming activation packet is received

Another interesting portion of the code of “Penquin_x64” is reported in Figure 8. These instructions are executed when the conditions we discussed are matched. However, it is important to note that **final_check_1** must be equal to `1Dh` in order to be accepted by this sample. Such value was specified as a command line parameter in “Penquin” version and it was hardcoded in “Penquin_x64”.

Figure 8 also reports an important detail about **final_check_2**. This value is used as a status flag. Whenever the execution reaches the final basic block, the sample changes its internal status according to the value of **final_check_2** and stores it into the register **r15d**. This value depends on the values marked with ‘x’ of the second DWORD of the packet. This means that a correctly formatted activation packet is accepted only if the state of the stored **final_check_2** value is different from the one generated from the incoming packet. However **final_check_2** seems to have only two possible values that are 0 or 2. Since the status is zero-initialized, the first packet to activate Penquin must have **final_check_2** equal to 2.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

4.4 Is the Penquin in the house?

With the knowledge and understanding of the portion of binary code related to the magic packet, we have been able to forge packets that are compliant with all the conditions of the Penquin protocol. The python code reported below triggers the sample to connect back to an IP address that is encoded inside the payload. It can be used by blue teams to check if their Linux servers are infected by this version of “Penquin_x64”.

```
import socket

tc_remote_port = 0x9999
outgoing_if = "192.168.202.1" # local IP
remote_tc_host = "192.168.202.130" # potentially infected machine IP

#send UDP packet with final_check2 equal to 2 and callback IP address equal to 10.11.60.129
tc_local_port =  65105
tc_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
tc_sock.bind((outgoing_if, tc_local_port))
tc_sock.sendto(b"\x4a\x0a\x3c\x83\x60\x95\xbd\xbd", (remote_tc_host,tc_remote_port))

#send UDP packet with final_check2 equal to 0 and callback IP address equal to 10.188.60.129
tc_local_port =  30846
tc_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
tc_sock.bind((outgoing_if, tc_local_port))
tc_sock.sendto(b"\x0a\xbc\x3c\x80\x62\x85\xbd\xbd", (remote_tc_host,tc_remote_port))
```

The variables that can be freely configured are: `tc_remote_port`, `outgoing_if` and `remote_tc_host`. Since we are unable to establish the value of the status flag in advance, two packets are sent in order to trigger the connection to one of the two private IP addresses 10.11.60.129 or 10.188.60.129, that should be used by the system administrators to monitor the call back home traffic of “Penquin_x64”.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

5. CONCLUSION

This report is based on the outcome of a deep technical analysis of a set of malware samples that have been uploaded to a multi-scanner service in the last few years. Since we didn't have the chance to investigate an actual incident involving them, or to analyse a single compromised machine, we have not been able to collect and analyse additional context information that could add important information about the victimology, the TTPs of the threat actor and the timeline of the events.

The analysis we performed proved that Turla continued to develop new versions of "Penquin_x64" at least up to April 2016. Furthermore, it revealed the enormous effort put in place by the threat actor to avoid the activation of the backdoor not controlled by themselves. Nevertheless, we have been able to completely understand the network protocol that is used, and to build a tool that is able to activate the backdoor and call back an IP address that we can control. This tool can be used by system administrators for defensive purposes.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

6. MITRE ATT&CK TTPs

T1059	Command-Line Interface	Command-line interfaces provide a way of interacting with computer systems and is a common feature across many types of operating system platforms. One example command-line interface on Windows systems is cmd, which can be used to perform a number of tasks including execution of other software. Command-line interfaces can be interacted with locally or remotely via a remote desktop application, reverse shell session, etc. Commands that are executed run with the current permission level of the command-line interface process unless the command includes process invocation that changes permissions context for that execution (e.g. Scheduled Task).
T1205	Port Knocking	Port Knocking is a well-established method used by both defenders and adversaries to hide open ports from access. To enable a port, an adversary sends a series of packets with certain characteristics before the port will be opened. Usually this series of packets consists of attempted connections to a predefined sequence of closed ports, but can involve unusual flags, specific strings or other unique characteristics. After the sequence is completed, opening a port is often accomplished by the host based firewall, but could also be implemented by custom software.
T1105	Remote File Copy	Files may be copied from one system to another to stage adversary tools or other files over the course of an operation. Files may be copied from an external adversary-controlled system through the Command and Control channel to bring tools into the victim network or through alternate protocols with another tool such as FTP. Files can also be copied over on Mac and Linux with native tools like scp, rsync, and sftp.
T1094	Custom Command and Control Protocol	Adversaries may communicate using a custom command and control protocol instead of encapsulating commands/data in an existing Standard Application Layer Protocol. Implementations include mimicking well-known protocols or developing custom protocols (including raw sockets) on top of fundamental protocols provided by TCP/IP/another standard network stack.
T1024	Custom Cryptographic Protocol	Adversaries may use a custom cryptographic protocol or algorithm to hide command and control traffic. A simple scheme, such as XOR-ing the plaintext with a fixed key, will produce a very weak ciphertext.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

T1032	Standard Cryptographic Protocol	Adversaries may explicitly employ a known encryption algorithm to conceal command and control traffic rather than relying on any inherent protections provided by a communication protocol. Despite the use of a secure algorithm, these implementations may be vulnerable to reverse engineering if necessary secret keys are encoded and/or generated within malware samples/configuration files.
T1158	Hidden Files and Directories	To prevent normal users from accidentally changing special files on a system, most operating systems have the concept of a ‘hidden’ file. These files don’t show up when a user browses the file system with a GUI or when using normal commands on the command line. Users must explicitly ask to show the hidden files either via a series of Graphical User Interface (GUI) prompts or with command line switches (dir /a for Windows and ls –a for Linux and macOS).
T1222	File and Directory Permissions Modification	File and directory permissions are commonly managed by discretionary access control lists (DACLs) specified by the file or directory owner. File and directory DACL implementations may vary by platform, but generally explicitly designate which users/groups can perform which actions (ex: read, write, execute, etc.).

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

7. INDICATORS OF COMPROMISE

Hashes

1d5e4466a6c5723cd30caf8b1c3d33d1a3d4c94c25e2ebe186c02b8b41daf905
8dc3d053e5008ab92a17dc47fed43213a9873db0
0994d9deb50352e76b0322f48ee576c6

3e138e4e34c6eed3506efc7c805fce19af13bd62aeb35544f81f111e83b5d0d4
04686b0d2fdafa7cb6c17adc551abade334d7b85
14ecd5e6fc8e501037b54ca263896a11

8856a68d95e4e79301779770a83e3fad8f122b849a9e9e31cfe06bf3418fa667
7f043eb95d74d051ac780aee52ebf1c497c43060
19fbd8cbfb12482e8020a887d6427315

2dabb2c5c04da560a6b56dbaa565d1eab8189d1fa4a85557a22157877065ea08
4594453e2e4002101481dc44f203d3ffebe079ae
edf900cebb70c6d1fcab0234062bfc28

5a204263cac112318cd162f1c372437abf7f2092902b05e943e8784869629dd8
09580f1deb096bb50d082bd169271d41756adf73
ea06b213d5924de65407e8931b1e4326

d49690ccb82ff9d42d3ee9d7da693fd7d302734562de088e9298413d56b86ed0
9d133d7e0616573a7d1c822cc878149e7aa7bad6
e079ec947d3d4dacb21e993b760a65dc

8ccc081d4940c5d8aa6b782c16ed82528c0885bbb08210a8d0a8c519c54215bc
0675329cfa4d13ee35f74c1cc236bc630b7de464
ad6731c123c4806f91e1327f35194722

67d9556c695ef6c51abf6fbab17acb3466e3149cf4d20cb64d6d34dc969b6502
f5a1a9180913bbeb1641af48660fbb756325f91e
b4587870ecf51e8ef67d98bb83bc4be7

d9f2467ff11efae921ec83e074e4f8d2eac7881d76bff60a872a801bd45ce3d5
c67abb20ae5100f12ce084279827632fdbcb222a
7533ef5300263eec3a677b3f0636ae73

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Yara Rules

```
rule APT_MAL_LNX_Turla_Apr202004_1 {
  meta:
    description = "Detects Turla Linux malware x64 x32"
    date = "2020-04-24"
    hash1 = "67d9556c695ef6c51abf6fbab17acb3466e3149cf4d20cb64d6d34dc969b6502"
    hash2 = "8ccc081d4940c5d8aa6b782c16ed82528c0885bbb08210a8d0a8c519c54215bc"
    hash3 = "8856a68d95e4e79301779770a83e3fad8f122b849a9e9e31cfe06bf3418fa667"
    hash4 = "1d5e4466a6c5723cd30caf8b1c3d33d1a3d4c94c25e2ebe186c02b8b41daf905"
    hash5 = "2dabb2c5c04da560a6b56dbaa565d1eab8189d1fa4a85557a22157877065ea08"
    hash6 = "3e138e4e34c6eed3506efc7c805fce19af13bd62aeb35544f81f111e83b5d0d4"
    hash7 = "5a204263cac112318cd162f1c372437abf7f2092902b05e943e8784869629dd8"
    hash8 = "8856a68d95e4e79301779770a83e3fad8f122b849a9e9e31cfe06bf3418fa667"
    hash9 = "d49690ccb82ff9d42d3ee9d7da693fd7d302734562de088e9298413d56b86ed0"

  strings:
    $s1 = "/root/.hsperfdata" ascii fullword
    $s2 = "Desc|     Filename     |  size  |state|" ascii fullword
    $s3 = "VS filesystem: %s" ascii fullword
    $s4 = "File already exist on remote filesystem !" ascii fullword
    $s5 = "/tmp/.sync.pid" ascii fullword
    $s6 = "rem_fd: ssl " ascii fullword
    $s7 = "TREX_PID=%u" ascii fullword
    $s8 = "/tmp/.xdfg" ascii fullword
    $s9 = "__we_are_happy__" ascii fullword
    $s10 = "/root/.sess" ascii fullword
    $s11 = "ZYSZLRTS^Z@@NM@@G_Y_FE" ascii fullword
  condition:
    uint16(0) == 0x457f and
    filesize < 5000KB and
    4 of them
}
```

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

```
rule APT_MAL_LNX_Turla_Apr202004_1_opcode {
  meta:
    description = "Detects Turla Linux malware x64 x32"
    date = "2020-04-24"
    hash1 = "67d9556c695ef6c51abf6fbab17acb3466e3149cf4d20cb64d6d34dc969b6502"
    hash2 = "8ccc081d4940c5d8aa6b782c16ed82528c0885bbb08210a8d0a8c519c54215bc"
    hash3 = "8856a68d95e4e79301779770a83e3fad8f122b849a9e9e31cfe06bf3418fa667"
    hash4 = "1d5e4466a6c5723cd30caf8b1c3d33d1a3d4c94c25e2ebe186c02b8b41daf905"
    hash5 = "2dabb2c5c04da560a6b56dbaa565d1eab8189d1fa4a85557a22157877065ea08"
    hash6 = "3e138e4e34c6eed3506efc7c805fce19af13bd62aeb35544f81f111e83b5d0d4"
    hash7 = "5a204263cac112318cd162f1c372437abf7f2092902b05e943e8784869629dd8"
    hash8 = "8856a68d95e4e79301779770a83e3fad8f122b849a9e9e31cfe06bf3418fa667"
    hash9 = "d49690ccb82ff9d42d3ee9d7da693fd7d302734562de088e9298413d56b86ed0"

  strings:
      $op0 = { 8D 41 05 32 06 48 FF C6 88 81 E0 80 69 00 } /* Xor string loop_p1 x32*/
      $op1 = { 48 FF C1 48 83 F9 49 75 E9  } /* Xor string loop_p2 x32*/
      $op2 = { C7 05 9B 7D 29 00 1D 00 00 00 C7 05 2D 7B 29 00 65 74 68 30 C6 05 2A 7B 29 00 00 E8 }
              /* Load eth0 interface*/
      $op3 = { BF FF FF FF FF E8 96 9D 0A 00 90 90 90 90 90 90 90 90 90 90 89 F0}
               /* Opcode exceptions*/
      $op4 = { 88 D3 80 C3 05 32 9A C1 D6 0C 08 88 9A 60 A1 0F 08 42 83 FA 08 76 E9   }
               /* Xor string loop  x64*/
      $op5 = { 8B 8D 50 DF FF FF B8 09 00 00 00 89 44 24 04 89 0C 24 E8 DD E5 02 00 } /* Kill call x32 */
      $op6 = { 8D 5A 05 32 9A 60 26 0C 08 88 9A 20 F4 0E 08 42 83 FA 48 76 EB } /* Decrypt init str */
      $op7 = { 8D 4A 05 32 8A 25 26 0C 08 88 8A 20 F4 0E 08 42 83 FA 08 76 EB} /* Decrypt init str */
  condition:
    uint16(0) == 0x457f and
    filesize < 5000KB and
    2 of them
}
```

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

REFERENCES

[1] The ‘Penquin’ Turla: A Turla/Snake/Uroburos Malware for Linux; by Kurt Baumgartner, Costin Raiu. December 2014.

[2] Penquin’s Moonlit Maze: The Dawn of Nation-State Digital Espionage; by Juan Andres Guerrero-Saade (GReAT), Costin Raiu (GReAT), Daniel Moore (King’s College London), Thomas Rid (King’s College London). April 2017.

[3] APT Case RUAG; by GovCERT.ch: Technical Report about the Espionage Case at RUAG. May 2016.

[4] Satellite Turla: APT Command and Control in the Sky. How the Turla operators hijack satellite Internet links, by Stefan Tanase. September 2015.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

APPENDIX A: BUILD DATE ESTIMATION

The ELF file format does not contain an explicit compile timestamp field, and therefore we had to consider different approaches. Due to these limitations, we have been able to outline a rough timeline of events only, but we have no evidence that such dates have been timestomped in some way or that some behaviour has been deliberately put in place by the threat actor in order to hinder such kind of analysis.

Statically linked libraries and compilers

All the samples we analysed statically include a few libraries, such as *libpcap*, *openssl* and *glibc* which have strings suggesting their version. Furthermore, even though debugging information has been stripped off, the samples still embed GCC compiler versions into the section named *.comment*. Therefore, our first thought was to estimate the release date of these libraries and compilers in such a way to have a lower bound on the build date of the malware. Unfortunately, as for “Penquin” and “Penquin_2.0”, most of the temporal references date back to 2000-2004, as reported in Table 5. These dates may suggest that the samples belonging to these two sets have been built almost ten years before the publication of the first report on this threat. It is also likely that the threat actor included these old libraries for other reasons, such as to preserve compatibility.

The “Penquin_x64” set is far more recent. Indeed, these samples introduced the usage of OpenSSL 1.0.1j that has been released at the end of 2014, more specifically on October 15th, 2014. Therefore, taking into account this information only, “Penquin_x64” must have a build date that is post mid-October 2014. At this point, we almost lost the hope that “Penquin_x64” could be more recent than 2014, but we wanted to look at it closer. Therefore, we tried harder (and did more).

Library Included	Penquin	Penquin_2.0	Penquin_x64	Year (>=)
OpenSSL 0.9.6	✓			2000
OpenSSL 0.9.7e		✓		2004
OpenSSL 1.0.1j			✓	2014
libpcap (several with a date between 2000 and 2001)	✓			2000 - 2001
libpcap version 1.1.1			✓	2010
glibc 2.3.2	✓	✓		2004

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

glibc 2.3.6			✓	2006
GCC: (GNU) 3.3.2 20031022 (Red Hat Linux 3.3.2-1)	✓			2003
GCC: (GNU) 3.3.2 20031218 (Red Hat Linux 3.3.2-5)	✓			2003
GCC: (GNU) 3.3.5 (Debian 1:3.3.5-13)	✓	✓		2004
GCC: (GNU) 3.3.6			✓	2005
GCC: (GNU) 4.1.0			✓	2006

Table 5 - Compilers and Libraries statically included in the three “Penquin” versions

EI_ABIVERSION Analysis

We made a second attempt to estimate the build date using the EI_ABIVERSION embedded within the samples. This field of the ELF binary file format specifies the version of the Application Binary Interface to which the object is targeted. This version is mainly related to what version of the compiler (typically GCC) was used to build the sample. Although a table of the GCC releases can be obtained easily (https://gcc.gnu.org/releases.html), the ABI version associated with each version is not immediately available. To this aim we downloaded several GCC versions and tried to derive which ABI corresponds to each version. Partial results of this preliminary task are reported in Table 6.

GCC	ABI	Release date
4.4.4	2.6.15	April 29, 2010
4.8.2	2.6.24	October 16, 2013
4.9.1	2.6.32	July 16, 2014
6.2.0	2.6.32	August 22, 2016
6.3.0	2.6.32	December 21, 2016
7.2.0	3.2	August 14, 2017

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

7.3.0	3.2	January 25, 2018
7.5	3.2	November 14, 2019
9.2.1	3.2	August 12, 2019

Table 6 - ABI and GCC versions release date

Unfortunately, as reported in Table 6, the ABI remains quite stable over time (e.g. ABI 2.6.32 has been out for 2.5 years at least), as such this analysis didn’t provide us a smaller time window with respect to the analysis of the linked libraries (see Table 7).

ABI Version	Penquin	Penquin_2.0	Penquin_x64	Year (>=)
2.2.0		✓		2004
2.2.5	✓			2003
2.4.18			✓	2006

Table 7 - Penquin build date estimation based on the ABI field of the ELF

Section 3 describes the approach that was taken to estimate the build date of the “Penquin_x64” samples, achieving what we consider the most accurate result.

© Copyright Leonardo S.p.a. – All rights reserved

Company General Use

Piazza Monte Grappa, 4
00195 Rome
T +39 06324731
F +39 063208621

leonardocompany.com

© Copyright Leonardo S.p.a. – All rights reserved