Establishing the TigerRAT and TigerDownloader malware families

threatray.com/blog/establishing-the-tigerrat-and-tigerdownloader-malware-families

Executive Summary

Recent research by Malwarebytes (April 2021), Kaspersky (June 2021) and the Korean CERT (September 2021), reports about attacks on South Korean entities, employing new techniques and malware not previously identified.

The initial report by Malwarebytes attributes the attack to the Lazarus group. Kaspersky refines the attribution to the Andariel APT, a subgroup of Lazarus. Korea CERT (KrCERT) reports a new attack and calls the malware tools seen in this attack TigerDownloader and TigerRAT. The KrCERT report provides a thorough and detailed, indicator-based analysis of the relationship between their malware samples and those previously analyzed by Kaspersky and Malwarebytes. They also employ a proprietary attribution technology to further relate the attacks.

In this report, we focus on the malware tooling from the previously reported attacks. We provide new evidence to attribute these tools to the same downloader and RAT families. We will refer to these families as TigerDownloader and TigerRAT respectively. We've chosen these names in recognition of KrCERT's important work where the names were first introduced to refer to the malware tools they studied in that same work.

We systematically study code reuse as well as functional commonalities between all the samples used in different stages of the previously reported attacks (i.e., packers, downloaders, and RAT payloads). We have also found that while the tools fall into the mentioned families, there are different variants of the tools which have been deployed in the reported attacks. For the RAT payloads, we have found three versions with distinct capabilities. For the downloaders we have found two versions, one with and the other without persistence capabilities.

Apart from these findings, we contribute novel insights and speculations to the existing body of knowledge toward a clearer mapping of the techniques and tools used by this threat actor. Finally, we are making our unpacking and config extraction scripts as well as raw data available to the community (https://github.com/threatray/tigerrat) to facilitate further research and defense capabilities.

Introduction

What is the Andariel APT group?

Andariel group is a state-sponsored threat actor. It is a subgroup of the Lazarus cybercrime group, considered one of the most sophisticated North Korean threat actors to which threat researchers have attributed many attacks from 2009 to 2021. The Andariel group is mostly targeting South Korean entities focusing mainly on financial gain and cyber espionage. This group is known to employ custom tools and new techniques to increase the effectiveness of its attacks.

Previous research

April 19, 2021: Malwarebytes has reported a recent attack targeting South Korea using a malicious Word document. The Malwarebytes report describes the attack and attributes it to the Lazarus group. Malwarebytes discovered a novel downloader component used in the attack. – https://blog.malwarebytes.com/threat-intelligence/2021/04/lazarus-apt-conceals-malicious-code-within-bmp-file-to-drop-its-rat/

June 15, 2021: Kaspersky released a blog post about the same attack, mentioning the Malwarebytes report, saying they detected the Word document in April. Kaspersky refines the attribution to the Andariel APT group, a subgroup of Lazarus. Kaspersky's analysis is based on operational similarities found between the current and past attacks of the Andariel APT group. They also identify novel downloaders and RAT payloads. In addition, they find a new ransomware deployed by the RAT. – https://securelist.com/andariel-evolves-to-target-south-korea-with-ransomware/102811/

September, 2021: KrCERT reports on an operation they call "ByteTiger", a campaign targeting South Korean entities which they have attributed to the Andariel APT group. This report analyses in detail a multistage attack with two unknown pieces of code which they call TigerDownloader and TigerRAT. They link the new attack to the samples previously disclosed by Malwarebytes and Kaspersky using some proprietary tooling. Linkage is apparently done through similarities / re-use of code, rich headers, section hashes and C2 infrastructure, yet no further details are shared in the report. https://www.krcert.or.kr/filedownload.do?attach_file_seq=3277&attach_file_id=EpF3277.pdf

The attack chains in all the reported cases have some structural similarities (see Figure 1). In all three reports a downloader malware has been observed. Kaspersky and KrCERT have additionally seen a third-stage RAT components. Concerning the access methods, malicious documents have been used in the cases reported by Malwarebytes and Kaspersky, whereas a compromised website was used in the KrCERT case.

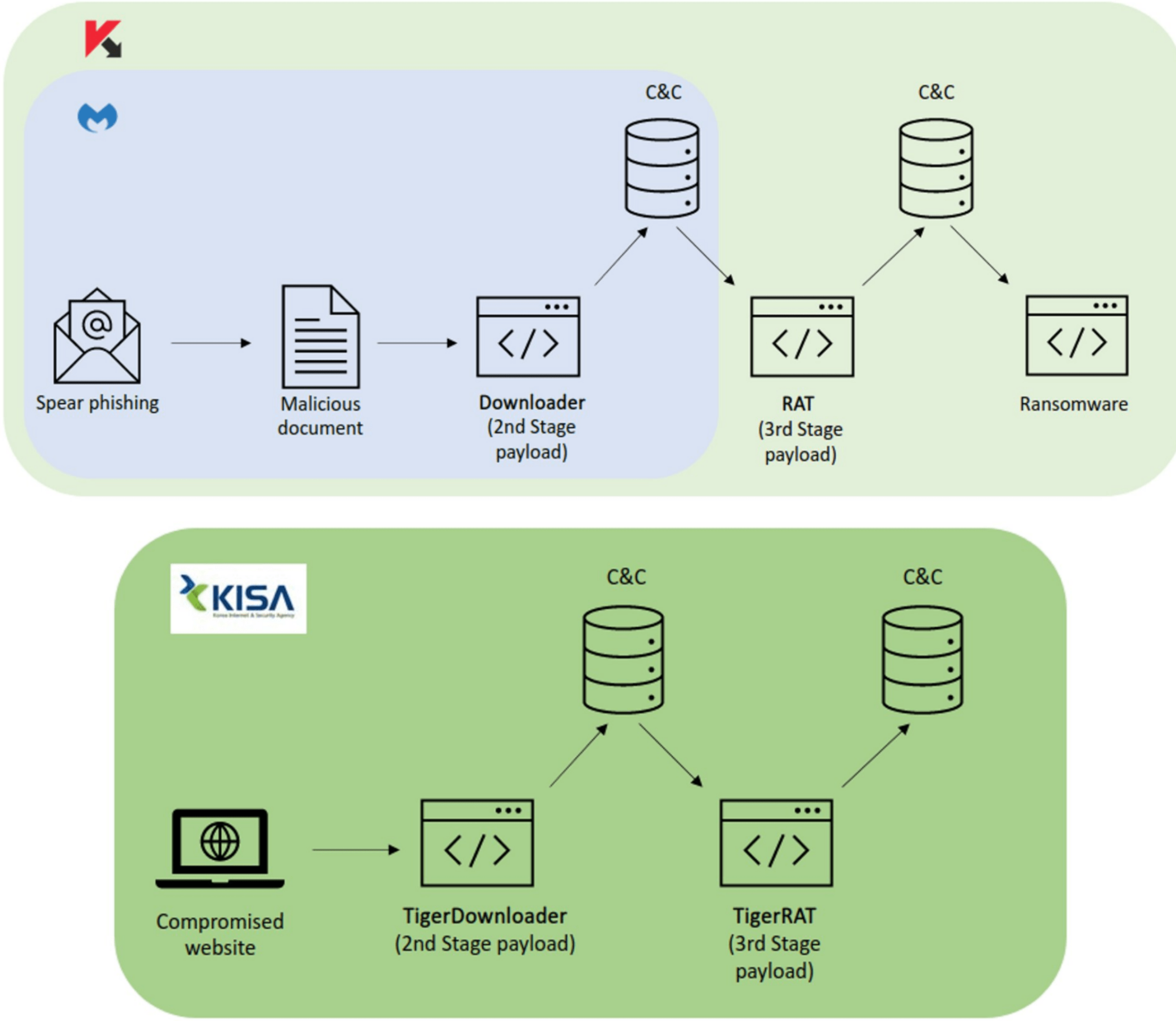

Figure 1: Similarities and differences between the attack chains reported by Malwarebytes, Kaspersky and KrCERT.

Packer analysis

In this section we will first establish that the packed binaries share common code that originates from the unpacking algorithm. Then we show that there is a common packing scheme underlying all the packed samples at our disposal. Our findings thus provide strong evidence that the binaries are related by the same packer. Should the packer be under exclusive control of the attacker (which we don't know) then our findings would allow attribution of all the binaries to the same actor.

Shared code in packed samples

To quickly understand if the packed samples are related, we performed an automated code reuse analysis at the function level. The results forming that analysis are shown in Figure 2. In the table, the numbers in the "function reuse" column measure the number of samples in which a function occurs. As an example, the function at the address 0x140002b70 (first row) appears in 27 out of 27 the packed samples. That is, this is a function that occurs in all packed samples.

```
$ python3 tiger.py --cluster-functions
+----------------------------------------------------------------------------------------------------------+
|                            packed hash                                function address   function reuse |
+==========================================================================================================+
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140002b70          (27/27)          |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140001bf0          (27/27)          |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140002030          (26/27)          |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140002860          (26/27)          |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140001230          (17/27)          |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   403160             (15/27)          |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   4030a0             (14/27)          |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140002360          (12/27)          |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140002a30          (14/27)          |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   402760             (10/27)          |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   401890             (10/27)          |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   4023d0             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   402b00             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   402370             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   401dd0             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   401d60             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   402250             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   4033e0             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   4034d0             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   4034f0             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   403520             (9/27)           |
| 0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01   4035a0             (9/27)           |
| 0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414   140001ef0          (7/27)           |
| 0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414   140002660          (5/27)           |
| 0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414   140003070          (5/27)           |
| 0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414   140001e70          (5/27)           |
| 0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414   140003150          (5/27)           |
| 0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414   140003210          (5/27)           |
| 0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414   1400032f0          (5/27)           |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140003060          (3/27)           |
| ed5fbefd61a72ec9f8a5ebd7fa7bcd632ec55f04bdd4a4e24686edccb0268e05   140001dc0          (2/27)           |
| ed5fbefd61a72ec9f8a5ebd7fa7bcd632ec55f04bdd4a4e24686edccb0268e05   1400036f0          (2/27)           |
| ed5fbefd61a72ec9f8a5ebd7fa7bcd632ec55f04bdd4a4e24686edccb0268e05   1400037a0          (2/27)           |
| ed5fbefd61a72ec9f8a5ebd7fa7bcd632ec55f04bdd4a4e24686edccb0268e05   140003870          (2/27)           |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140001e20          (2/27)           |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140003140          (2/27)           |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140003210          (2/27)           |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   1400032d0          (2/27)           |
| 69bac736f42e37302db7eca68b6fc138c3aa9a5c902c149e46cce8b42b172603   140003380          (2/27)           |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   140003390          (2/27)           |
| 0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c   1400033e0          (2/27)           |
+----------------------------------------------------------------------------------------------------------+
```

Figure 2: Function reuse across the 27 packed binaries which we have analyzed.

There are several other functions (i.e., 0x140001bf0, 0x140002030, 0x140002860) that appear in 27 or 26 samples. From the table, we can establish that the packed samples are clearly related. All of them have two functions in common and there are various subsets of the samples that feature substantial code reuse.

In a nutshell, the automated function reuse analysis gives us a quick understanding about the relations of the packed samples. As we shall see next, it also directs our manual analysis efforts.

Based on the analysis, we suspected that the samples share a few functions for the effective unpacking, while some of the remaining functions are used to avoid detections by antivirus, Yara and related pattern-based detection technologies. We then took a closer look at these stable functions and could confirm that they do, indeed, contain packing functionality. The results of this analysis are shown in Figure 3.

PACKED HASH	FUNCTION ADDRESS	FUNCTION REUSE	FUNCTIONALITY
0996a8e5ec1a41645309…	140002b70	(27/27)	map_decrypted_payload()

0996a8e5ec1a41645309…	140001bf0	(27/27)	anti_analysis_check()
0996a8e5ec1a41645309…	140002030	(26/27)	do_unpacking()
0996a8e5ec1a41645309…	140002860	(26/27)	dynamic_winapi_resolution()
0996a8e5ec1a41645309…	140002360	(12/27)	main_program()
0996a8e5ec1a41645309…	140002a30	(14/27)	relocate_mapped_payload()

Figure 3: Functionality found in the most stable functions.

We could also confirm the presence of junk code to avoid detection technologies. Figure 4 shows the same function decrypt_payload() in two different samples. We can see junk functions like GetFontUnicodeRanges(), GetSysColorBrush() and CreateBitMap() which are called but whose return values are not being used. In the figure, the effective unpacking code, which in this case is the XOR decryption algorithm, is contained within the green boxes shown.

We have found this junk-code strategy in all the packed code and throughout many functions.

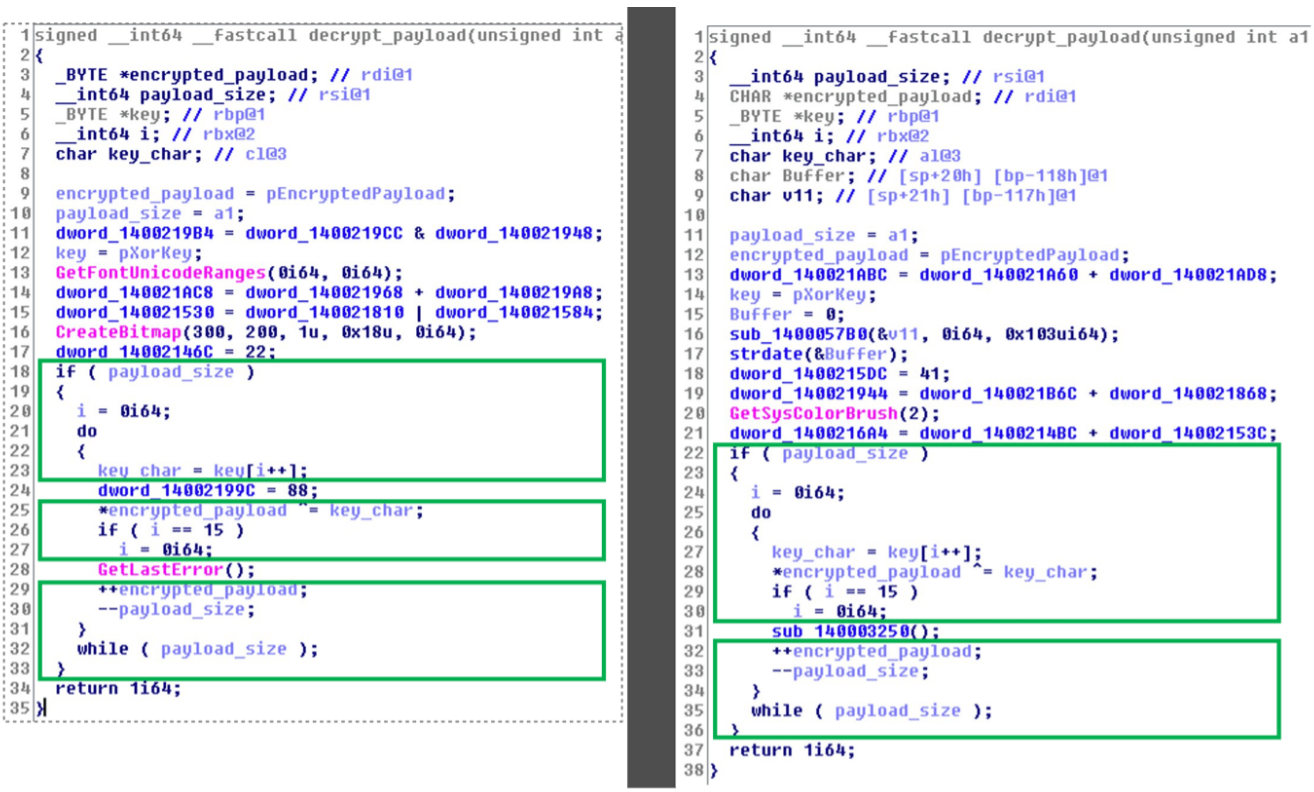

Figure 4: Junk code in packer code to avoid anti-virus and Yara detections.

In summary, we have seen so far that the packed samples are related by a common packer code. The code-wise differences between the packed samples is mainly due to the presence of junk code.

Common packing scheme

The packer is a simple loader, which decrypts and maps the payload into memory. The decryption scheme is a simple XOR using a 16-byte key. This has been established in previous research.

Additionally, we found that all packer variants follow the same common packing scheme, whereas the variants of the scheme are determined by two parameters. One parameter is whether or not the packed payload is Base64 encoded, the other is where the packed payload is stored within a PE file.

The variations concerning encoding of the payload are illustrated in Figure 5.

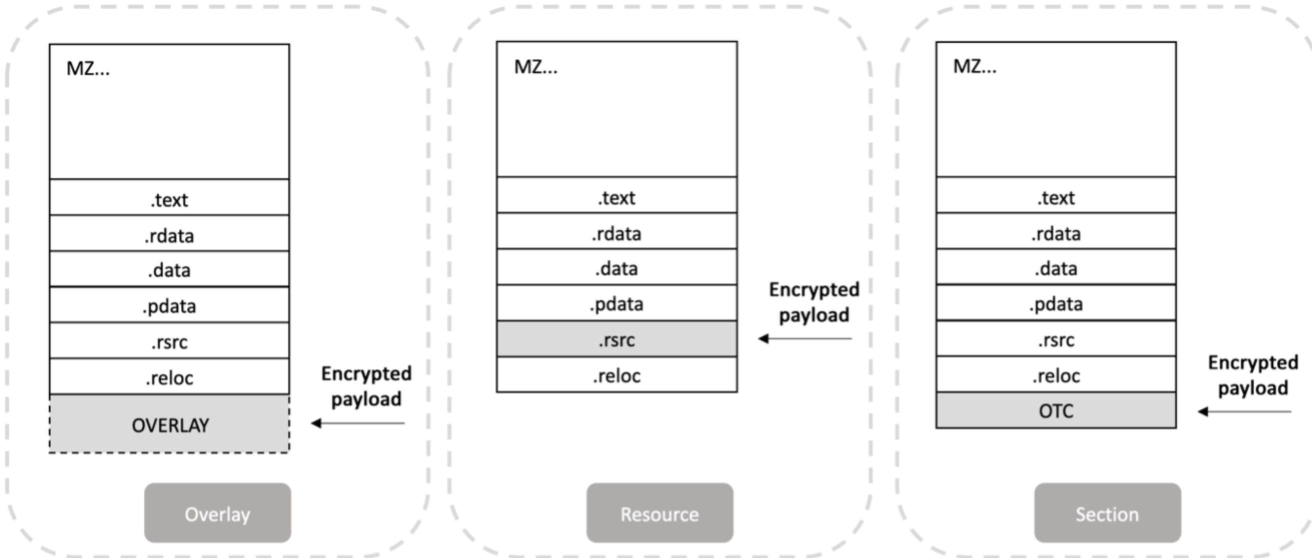

Figure 6: Variations of packed code locations in a PE file. Left to right, packed code in PE overlay, in the PE resource section, or in a dedicated PE section which is named OTC in this example.

For the third variant using a dedicated section, we observed the following section names: “KDATA,” “OTC,” “OTS,” “OTT,” and “data.” We could not identify the significance, if any, underlying these names.

Our findings are summarized in Figure 7, which shows the packing scheme common to all packed downloader and RAT variants we analyzed.

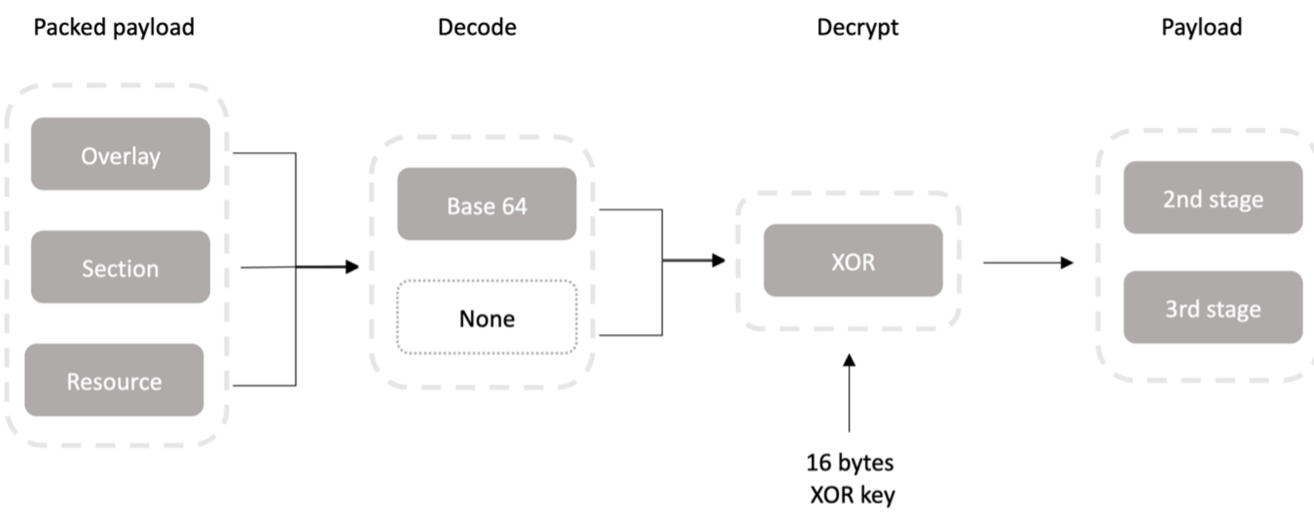

Figure 7: Packing scheme common to all samples.

Malware families and variants

In this section, we will establish through code reuse analysis that all the unpacked binaries fall into a downloader or RAT family. We are calling these families the TigerDownloader and TigerRAT malware family. These names were introduced in the KrCERT report to refer to the downloader and RAT components in their investigation.

To get a quick understanding of the unpacked binaries, we have performed a combined cluster and code-reuse analysis. This analysis allows us to automatically identify malware families and malware variants within a family. The goal of this analysis is to gain a quick understanding of the relationship between binaries and to direct analysts to the relevant samples for further manual analysis to eventually understand the attacker’s tooling and capabilities.

The results of the cluster and code-reuse analysis are shown in Figure 8. The figure confirms that the unpacked binaries either fall into the TigerDownloader (blue) or TigerRAT (orange) family. Moreover, we see that each family has three variants (shown as large circles). We have used a cluster threshold of 97.5%, meaning that binaries which are at least 97.5% similar fall into the same cluster. The clusters in the graph consist of the so-called “cluster representatives” (large circles) and samples (small circles) directly connected to a cluster representative. The underlying idea is that the samples within a cluster are essentially identical and thus well represented by the cluster representative.

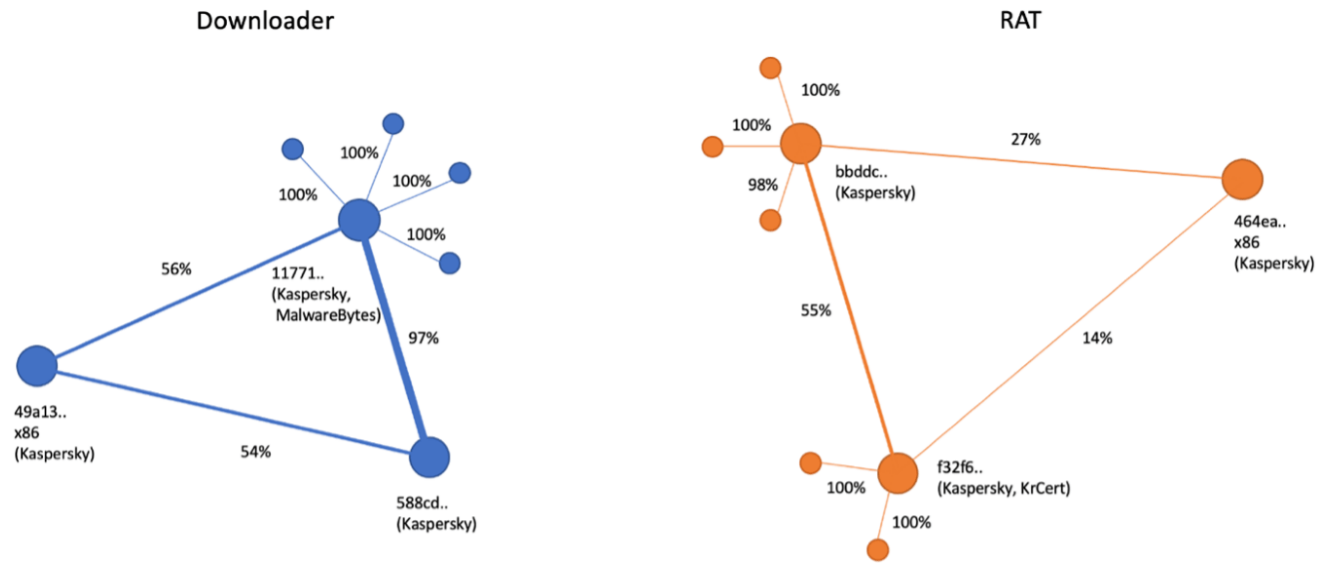

Figure 8: Cluster and code-reuse analysis of the unpacked samples with their abbreviated hashes.

We note that the choice of the cluster threshold has an obvious impact on the variants: A high threshold will reveal minor and more variants, while a low one reveals fewer and only major variants.

We draw the following conclusions from the graphs:

- There is no code reuse between the TigerDownloader and the TigerRAT family. We recall from the packer analysis that although the families are code-wise distinct, they are packed using the same packing scheme.
- Within the downloader family, there are three variants: one x86 and two x64 variants. The two x64 variants are very closely related (i.e., 97% code reuse) and thus are likely variants with minor differences.
- Within the RAT family, we have a similar situation with three variants: one x86 and two x64. However, the two x64 variants only share 55% of their code and thus seem to be substantial RAT variants.

The relations between the x64 and x86 binaries are lower, which is expected due to compiler and CPU architecture differences, but relevant code reuse can still be found.

The table in Figure 9 shows the detailed composition of the clusters from the previous graphs. We also notice that some (hash-wise) unique packed samples result in (hash-wise) identical unpacked sample, reducing the effective diversity of the samples under consideration.

```
$ python3 tiger.py --cluster-samples --threshold 0.975
+---------------------------------------------------------------------------------------------------------------+
| cluster                        unpacked hash                                   packed hash        sample source   |
+===============================================================================================================+
| 0       4aadf767491077ab83c6436cf108b014fc0bf8c3bd01cc6087a0f2b80564bc08   4d03a981...   ThirdStage(Kaspersky)        |
| 0       bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc4bc94f2d71ac   9137e886...   ThirdStage(Kaspersky)        |
| 0       bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc4bc94f2d71ac   d26987b7...   ThirdStage(Kaspersky)        |
| 0       bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc4bc94f2d71ac   b0d6aee3...   ThirdStage(Kaspersky)        |
| 0       bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc4bc94f2d71ac   f13aff9e...   ThirdStage(Kaspersky)        |
| 0       bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc4bc94f2d71ac   0e447797...   ThirdStage(Kaspersky)        |
| 0       868a62feff8b46466e9d63b83135a7987bf6d332c13739aa11b747b3e2ad4bbf   None          ThirdStage(Kaspersky)        |
| 0       8b3c8046fa776b70821b7e50baa772a395d3d245c10bdaa4b6171e0c5ce3f717   69bac736...   ThirdStage(Kaspersky)        |
| 1       f32f6b229913d68daad937cc72a57aa45291a9d623109ed48938815aa7b6005c   1f8dcfae...   ThirdStage(KrCERT)           |
| 1       ed11e94fd9aa3c7d4dd0b4345c106631fe52929c6e26a0daec2ed7d22e47ada0   d231f3b6...   ThirdStage(Kaspersky)        |
| 1       fec82f2542d7f82e9fce3e16bfa4024f253adee7121973bd9d67a3c79441b83c   da787cf1...   ThirdStage(Kaspersky)        |
| 2       63bae252d796bc9ac331fdc13744a72bd85d1065ef41a884dc11c6245ea933e2   6310cd9f...   SecondStage(Kaspersky)       |
| 2       63bae252d796bc9ac331fdc13744a72bd85d1065ef41a884dc11c6245ea933e2   b59e8f44...   SecondStage(Kaspersky)       |
| 2       1177105e51fa02f9977bd435f9066123ace32b991ed54912ece8f3d4fbeeade4   008e906f...   SecondStage(Malwarebytes)    |
| 2       1177105e51fa02f9977bd435f9066123ace32b991ed54912ece8f3d4fbeeade4   ed5fbefd...   SecondStage(Kaspersky)       |
| 2       5c2f339362d0cd8e5a8e3105c9c56971087bea2701ea3b7324771b0ea2c26c6c   f4765f7b...   SecondStage(Kaspersky)       |
| 3       588cdbd3ee3594525eb62fa7bab148f6d7ab000737fc0c311a5588dc96794acc   0996a8e5...   SecondStage(Kaspersky)       |
| 3       588cdbd3ee3594525eb62fa7bab148f6d7ab000737fc0c311a5588dc96794acc   4da0ac4c...   SecondStage(Kaspersky)       |
| 4       49a13bf0aa53990771b7b7a7ab31d6805ed1b547e7d9f114e8e26a98f6fbee28   ab194f2b...   SecondStage(x86)(Kaspersky)  |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   e83f5e0a...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   1892b72c...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   7d7dc812...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   87f389d8...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   0dc3f66f...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   d0fa0bfe...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   ebe4befd...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   f62adc67...   ThirdStage(x86)(Kaspersky)   |
| 5       464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067   2f53109e...   ThirdStage(x86)(Kaspersky)   |
+---------------------------------------------------------------------------------------------------------------+
```

Figure 9: Detailed cluster information.

In the following sections we will analyze the downloader and RAT variants in more detail, limiting our analysis to the cluster representatives. This ability to reduce analysis to cluster representatives is key for the directed and efficient analysis and tracking of malware variants. The choice of cluster representatives and their names used in the following analysis are shown in Figure 10.

CLUSTER	STAGE	SAMPLE NAME	HASH
0	RAT (x64) 3rd stage	RAT-Kaspersky-x64	bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc4bc94f2d71ac
1	RAT (x64) 3rd stage	RAT-KrCERT-x64	32f6b229913d68daad937cc72a57aa45291a9d623109ed48938815aa7b6005c
2	Downloader (x64) 2nd stage	Downloader-Malwarebytes-x64	1177105e51fa02f9977bd435f9066123ace32b991ed54912ece8f3d4fbeeade4
3	Downloader (x64) 2nd stage	Downloader-Kaspersky-x64	588cdbd3ee3594525eb62fa7bab148f6d7ab000737fc0c311a5588dc96794acc
4	Downloader (x86) 2nd stage	Downloader-Kaspersky-x86	49a13bf0aa53990771b7b7a7ab31d6805ed1b547e7d9f114e8e26a98f6fbee28
5	RAT (x86) 3rd stage	RAT-Kaspersky-x86	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067

Figure 10: Cluster representatives used in the subsequent analysis.

TigerDownloader variants

In this section, we take a closer look at the two downloader variants: Downloader-Malwarebytes-x64 and Downloader-Kaspersky-x64. From the cluster and code reuse analysis (see Figure 8) we know that they share 97% of code and thus are minor variants of the TigerDownloader family.

Using the binary diffing capabilities of our analysis toolchain, we see in Figure 11 that the samples are largely made up of the same functions, except for one unique function (the one with the address 0x140001230) in the Kaspersky (Downloader-Kaspersky-x64) sample.

```
$ python3 tiger.py --compare
+-------------------------------------------------------------------------------------------------------+
|            sample 1                address 1             sample 2                       address 2   |
+=======================================================================================================+
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140001030  1177105e51fa02f9977bd435f9066123...  0x140001030 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140001360  1177105e51fa02f9977bd435f9066123...  0x1400011f0 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140001480  1177105e51fa02f9977bd435f9066123...  0x140001310 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x1400016c0  1177105e51fa02f9977bd435f9066123...  0x140001550 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x1400017d0  1177105e51fa02f9977bd435f9066123...  0x140001650 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140001df0  1177105e51fa02f9977bd435f9066123...  0x140001c70 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140001ff0  1177105e51fa02f9977bd435f9066123...  0x140001e70 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x1400022d0  1177105e51fa02f9977bd435f9066123...  0x140002150 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x1400024b0  1177105e51fa02f9977bd435f9066123...  0x140002330 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x1400026b0  1177105e51fa02f9977bd435f9066123...  0x140002530 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140002ed0  1177105e51fa02f9977bd435f9066123...  0x140002d50 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140003220  1177105e51fa02f9977bd435f9066123...  0x1400030a0 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140003380  1177105e51fa02f9977bd435f9066123...  0x140003200 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140003f20  1177105e51fa02f9977bd435f9066123...  0x140003da0 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140003fd0  1177105e51fa02f9977bd435f9066123...  0x140003e50 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140004190  1177105e51fa02f9977bd435f9066123...  0x140004010 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140004420  1177105e51fa02f9977bd435f9066123...  0x140004160 |
| 588cdbd3ee3594525eb62fa7bab148f6...  0x140001230                                                  |
+-------------------------------------------------------------------------------------------------------+
```

Figure 11: Function level diff between Downloader-Kaspersky-x64 and Downloader-Malwarebytes-x64.

Analyzing the downloader sample from Kaspersky (see Figure 12), we see that the unknown function (0x140001230) is called from the main function of the downloader.

```
cnc_index = 1;
result = initialization();
if ( result )
{
  Dst[0] = 0;
  memset(&Dst[1], 0, 0x103ui64);
  GetModuleFileNameA(0i64, Dst, 520i64);
  sub_140001230(Dst);                               // <---- Unique function
  do
  {
    if ( do_communication() )
    {
      ATTEMPTS = 0;
    }
    else
    {
      ++ATTEMPTS;
      ++cnc_index;
      if ( cnc_index == 3 * (cnc_index / 3) )
      {
        cnc_url = COMMAND_AND_CONTROL_URL_2;
      }
      else
      {
        cnc_url = COMMAND_AND_CONTROL_URL_1;
        if ( cnc_index % 3 != 1 )
          cnc_url = COMMAND_AND_CONTROL_URL_3;
      }
      get_domain(cnc_url);
      if ( ATTEMPTS == 6 )
      {
        self_delete();
        exit(0);
      }
    }
    Sleep((1000 * dword_1400238FC));
  }
```

```
cnc_index = 1;
result = initialization();
if ( result )
{
  v3 = 0;
  memset(&Dst, 0, 0x103ui64);
  GetModuleFileNameA(0i64, &v3, 520i64);
  do
  {
    if ( do_communication() )
    {
      ATTEMPTS = 0;
    }
    else
    {
      ++ATTEMPTS;
      ++cnc_index;
      if ( cnc_index == 3 * (cnc_index / 3) )
      {
        cnc_url = &COMMAND_AND_CONTROL_URL_2;
      }
      else
      {
        cnc_url = &COMMAND_AND_CONTROL_URL_1;
        if ( cnc_index % 3 != 1 )
          cnc_url = &COMMAND_AND_CONTROL_URL_3;
      }
      get_domain(cnc_url);
      if ( ATTEMPTS == 6 )
      {
        self_delete();
        exit(0);
      }
    }
    Sleep(1000 * dword_1400238FC);
  }
```

Figure 12: Left, Downloader-Kaspersky-x64; right, Downloader-Malwarebytes-x64.

It turns out that this function is used to achieve persistence. The technique being used is straightforward and consists of creating a link in the current user startup folder to make sure that the downloader is started upon reboot of a victim machine (see Figures 13 and 14).

```
  executing_sample_path = a1;
  encrypted_string = -202377702;
  v6 = -1397834514;
  v7 = -792941395;
  v8 = -857608;
  v9 = -1292961292;
  v10 = 16249584;
  memset(&user_startup_folder, 0, 0x103ui64);
  if ( !get_user_startup_folder(&user_startup_folder) )// SHGetFolderPathA CSIDL_STARTUP
    goto LABEL_11;
  memset(&pszPath, 0, 0x103ui64);
  v2 = &encrypted_string;                          // "Visor 2010 Launcher.lnk"
  if ( encrypted_string )
  {
    do
    {
      *v2 ^= 0x9Cu;
      v3 = *(v2 + 1) == 0;
      v2 = (v2 + 1);
    }
    while ( !v3 );
  }
  sprintf_s(&pszPath, 0x103ui64, "%s\\%s", &user_startup_folder);
  // C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Visor 2010 Launcher.lnk
  if ( PathFileExistsA(&pszPath) )
    DeleteFileA(&pszPath);
  if ( create_shortcut(executing_sample_path, &pszPath) )
    result = 1i64;
  else
LABEL_11:
    result = 0i64;
  return result;
}
```

Figure 13: Function which creates a shortcut for persistence.

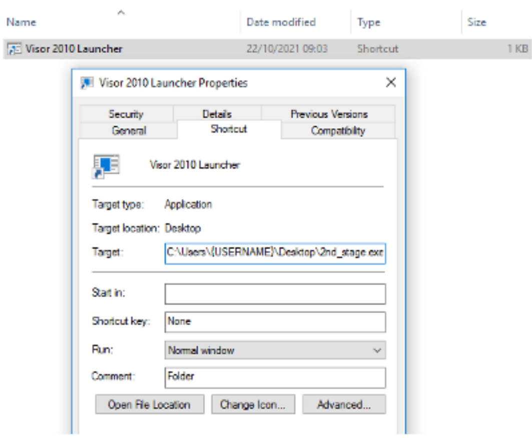

Figure 14: Shortcut to persistent executable.

Finally, we note that we haven't found any persistence techniques in the Downloader-Malwarebytes-x64 sample. The reason is likely to minimize indicators being left on victim machines.

Possible connections to the KrCERT TigerDownloader

Unfortunately, the downloader sample (f0ff67d4d34fe34d52a44b3515c44950) from the KrCERT report is not available publicly, thus we could not include it into our analysis. To nevertheless examine possible relations between KrCERT and the Malwarebytes and Kaspersky downloaders, we attempted to connect them purely based on the artifacts and behaviors publicly reported by KrCERT.

Let's start with a negative result. KrCERT reports a couple of C2 commands which they have found in their downloader (see Figure 15). We couldn't find any of the "Tiger10X" identifiers in the downloaders at our disposal. Neither were we able to find any other identifiers which could be possible C2 commands.

IDENTIFIER	ACTION
Tiger101	Send victim info
Tiger102	Receive command
Tiger103	File upload

Figure 15: TigerDownloader C2 commands reported by KrCERT.

On the other hand, we have found various aspects reported by KrCERT that are also present in the other downloaders:

- The packer in the KrCERT reports fits into the packer scheme which we have established above.
- KrCERT reports that the communication is encoded using Base64, which we have also observed in our samples.
- The 3rd stages (RATs) which are downloaded by the 2nd stages (downloaders) all belong to the same TigerRAT family (as we shall establish in the following section).

In a nutshell, the observations above suggest that the KrCERT Downloader might be related to the downloaders observed by Malwarebytes and Kaspersky. However, this is speculative because we lack hard evidence since we don't have access to the KrCERT sample.

TigerRAT variants

We recall from the code reuse and cluster analysis (see Figure 8) that we could connect all RATs to the same TigerRAT family through code-reuse analysis. We have also seen that there are RAT variants that differ more substantially than the downloader variants. For instance, the variants RAT-Kaspersky-x64 and RAT-KrCERT-x64 share only about 50% of their code.

In this section, we take a closer look at the RAT variants. We present strong new evidence on the functional and design levels that further attributes the RAT variants at our disposal to the same TigerRAT malware family. We also show that variants mainly differ in terms of the C2 commands they implement.

For this analysis, we'll focus on the representatives RAT-Kaspersky-x64, RAT-KrCERT-x64 and RAT-Kaspersky-x86 which we established earlier (see Figure 10).

Commands and capabilities per variant

Let's look at the C2 commands which we have found in the different variants. Figure 16 shows all the C2 commands that we have observed in at least one of the three RAT variants. The absence of the commands with the ids 0x08 and 0x09 lead us to speculate that there are yet unknown samples in the wild which do include these commands.

COMMAND	ID
SelfDelete	0x01
SystemInfo	0x02
Shell	0x03
FileManager	0x04
Keylogger	0x05
SocksTunnel	0x06
ScreenCapture	0x07
PortForwarder	0x0a

Figure 16: Summary of all C2 commands which are available in at least one of the three RAT variants.

Next, we're looking at the C2 commands which are supported by the different variants (see Figure 17).

RAT VARIANTS	COMMANDS
RAT-Kaspersky-x86	FileManager, ScreenCapture, SelfDelete, Shell
RAT-Kaspersky-x64	FileManager, Keylogger, ScreenCapture, SelfDelete, Shell, SocksTunnel, SystemInfo
RAT-KrCERT-x64	FileManager, Keylogger, PortForwarder, ScreenCapture, SelfDelete, Shell, SocksTunnel, SystemInfo

Figure 17: C2 commands found in the different RAT variants.

We see that the three variants which we have automatically identified using cluster analysis are indeed three functionally distinct variants. Apart from these variations in C2 capabilities, the core code of the variants is largely identical. Thus, it is essentially the C2 commands that define the three variants. We also observe that the four commands "FileManager," "ScreenCapture," "SelfDelete" and "Shell" are common to all variants.

A common interface for C2 commands

We have found an interface that is common to all three variants, as follows:

```
struct t_Module_GenericCommandInterface
{
t_GenericCommand *Command;
_DWORD id; // Command id
t_MainStructure *MainStructure;
_BYTE unk_data[0x10];
_BYTE initialized;
};

struct t_GenericCommand
{
void (*init)(t_Module_GenericCommandInterface *a1);
void (*execute)(t_Module_GenericCommandInterface *a1);
void (*enable)(t_Module_GenericCommandInterface *a1);
void (*disable)(t_Module_GenericCommandInterface *a1);
void *enabled;
};
```

The interface provides an abstraction that is implemented by all C2 commands found in the RATs. This common interface establishes a strong relation between the variants within their core C2 functionalities.

New C2 protocol variant in RAT-KrCERT-x64

The C2 protocol is essentially identical across all variants. The exception is a minor protocol change which we spotted in the RAT-KrCERT-x64 variant. The change concerns the registration of the malware with the C2 and consists of an extra check located in the TCP module, which is responsible for all communication with the C2:

```
struct t_TCP
{
void (*constructor)(t_Module_TCP *a1);
void (*set_cncs)(t_Module_TCP *a1);
void (*connect_to_cnc)(t_Module_TCP *a1);
void (*check_response_from_cnc)(t_Module_TCP *a1);
void (*listen_to_new_commands)(t_Module_TCP *a1);
void (*close_socket)(t_Module_TCP *a1);
void (*send_data)(t_Module_TCP *a1, t_EncData *a2, int a3);
void (*process_recv_command)(t_Module_TCP *a1);
void (*enable_commands)(t_Module_TCP *a1);
void *var_1;
};

struct t_TCP_Variant_KrCERT-x64
{
void (*constructor)(t_Module_TCP *a1);
void (*set_cncs)(t_Module_TCP *a1);
void (*connect_to_cnc)(t_Module_TCP *a1);
void (*check_response_from_cnc)(t_Module_TCP *a1);
void (*new_check_from_cnc_response)(t_Module_TCP *a1); // new in RAT-KrCERT-x64 variant
void (*listen_to_new_commands)(t_Module_TCP *a1);
void (*close_socket)(t_Module_TCP *a1);
void (*send_data)(t_Module_TCP *a1, t_EncData *a2, int a3);
void (*process_recv_command)(t_Module_TCP *a1);
void (*enable_commands)(t_Module_TCP *a1);
void *var_1;
};
```

In Figure 18, the red rectangle contains the new protocol check which was added to the RAT-KrCERT-x64 variant.

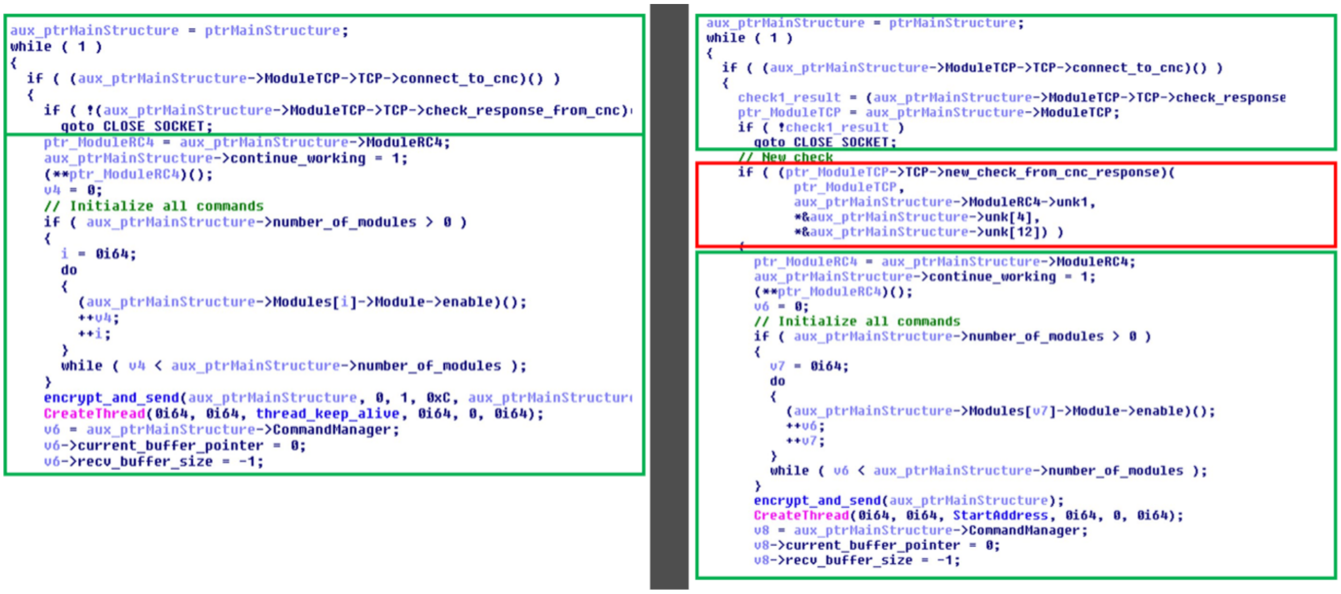

Figure 18: Left, other variants; right, RAT-KrCERT-x64 variant.

The new function essentially sends a 17-byte length chunk to the C2. We have not analyzed what data is sent, but it looks like it could be related to a bot identifier or something similar. Once the data is sent, it checks that the C2 returns the string “n0gyPPx” (see Figure 19).

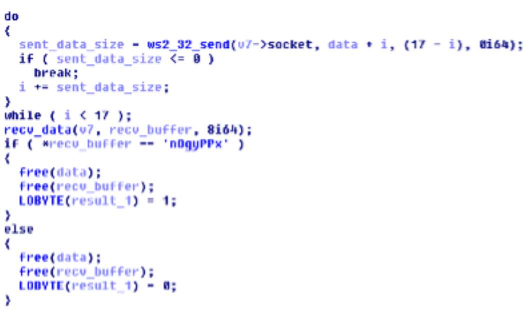

Figure 19: C2 protocol check for “n0gyPPx.”

In addition to this protocol change, we have also observed a change in the HTTP header that is sent at the beginning of the communication in the very first request by the RAT-KrCERT-x64 variant (see Figure 20).

RAT VARIANT	HTTP HEADER
RAT-KrCERT-x64	HTTP 1.1 /index.php?member=sbi2009 SSL3.3.7
RAT-KrCERT-x64, RAT-Kaspersky-x86	HTTP 1.1 /member.php SSL3.4

Figure 20: HTTP header variants.

Based on this protocol analysis, we believe that RAT-KrCERT-x64 is a slightly newer version of the RAT which is at the same time clearly related to the other versions.

Conclusions

Our analysis revealed new evidence and insights enabling us to attribute the previously reported Andariel APT binaries by Malwarebytes, Kaspersky and KrCERT to two new malware families. We call these the TigerDownloader and TigerRAT families, using names originally introduced by KrCERT. We have also seen that all the binaries are related by the same packing scheme. Our results are based on both automated code-reuse analysis and manual analysis of the malware tooling reported in the previous reports.

To facilitate further research and defense capabilities, we are sharing our unpacking and config-extraction scripts as well as data with the community (https://github.com/threatray/tigerrat).

The analysis in this report is based on the malware samples at our disposal at the time of writing. During our analysis, we found indicators suggesting that additional, not yet publicly known, variants may exist. Since threat analysis and attribution is data driven and evolving work, additional samples may complete our current findings or lead to new findings. We invite you to contact us with additional information, particularly if you can share suspected or confirmed TigerDownloader or TigerRAT binaries.

Appendix

Alleged compilation datesWe have looked at the compilation timestamps of the packed samples and concluded that they are randomly chosen. For instance, some timestamps are in the future (e.g., “2024/06/09”) and others many years in the past (e.g., “1996/10/17”).

On the other hand, we have found that the compilation dates of the unpacked samples appear reasonable and likely correspond to the effective compilation dates. In fact, the unpacked compilation timestamp is always before the first seen date. In many cases, it is 1 to 2 days before the first seen date which makes sense due to the time delay between the infection/detection and reporting/submission to platforms like VirusTotal. Also, none of the dates are in the future or unrealistically old. While these still could be false flags, it is reasonable to assume that the compilation dates of the unpacked samples correspond to their effective production date. We also see that most of the 3rd stage (TigerRAT) samples were detected before the 2nd stage (TigerDownloader) samples. This could indicate that until a host becomes infected by the 3rd stage, the 2nd stage samples are not detected. It could also be due to the fact that 2nd stage samples are stealthier and have fewer features/functions. The raw data is shown in the table below.

PACKED HASH	UNPACKED HASH
f4765f7b089d99b1cdcebf3ad7ba7e3e23ce411deab29b7afd782b23352e698f	5c2f339362d0cd8e5a8e3105c9c56971087bea2701ea3b73247
ed5fbefd61a72ec9f8a5ebd7fa7bcd632ec55f04bdd4a4e24686edccb0268e05	1177105e51fa02f9977bd435f9066123ace32b991ed54912ece8
008e906f2727d502f130a549eeebfda23362e24b2f1ac6e2c198ea82acc8a06a	1177105e51fa02f9977bd435f9066123ace32b991ed54912ece8
b59e8f44822ad6bc3b4067bfdfd1ad286b8ba76c1a3faff82a3feb7bdf96b9c5	63bae252d796bc9ac331fdc13744a72bd85d1065ef41a884dc1
6310cd9f8b6ae1fdc1b55fe190026a119f7ea526cd3fc22a215bda51c9c28214	63bae252d796bc9ac331fdc13744a72bd85d1065ef41a884dc1
350082b3f14e130c6337ef88d46d54d353ca6785508264112dfbd20ce4e47b98	63bae252d796bc9ac331fdc13744a72bd85d1065ef41a884dc1
f40d387631ddb0db70128e72239d0cae7a22b2135c0ec0d540e018aa727d4c8e	588cdbd3ee3594525eb62fa7bab148f6d7ab000737fc0c311a55
0996a8e5ec1a41645309e2ca395d3a6b766a7c52784c974c776f258c1b25a76c	588cdbd3ee3594525eb62fa7bab148f6d7ab000737fc0c311a55
4da0ac4c3f47f69c992abb5d6e9803348bf9f3c6028a7214dcabec9a2e729b99	588cdbd3ee3594525eb62fa7bab148f6d7ab000737fc0c311a55
ab194f2bad37bffd32fae9833dafaa04c79c9e117d86aa46432eadef64a43ad6	49a13bf0aa53990771b7b7a7ab31d6805ed1b547e7d9f114e8e
4d03a981bed15a3bd91f36972d7391b39791c582bb2959a9be154a74bd64db31	4aadf767491077ab83c6436cf108b014fc0bf8c3bd01cc6087a0f
1f8dcfaebbcd7e71c2872e0ba2fc6db81d651cf654a21d33c78eae6662e62392	f32f6b229913d68daad937cc72a57aa45291a9d623109ed4893
d231f3b6d6e4c56cb7f149cbc0178f7b80448c24f14dced5a864015512b0ba1f	ed11e94fd9aa3c7d4dd0b4345c106631fe52929c6e26a0daec2
da787cf1f4fd829dd4a7637bec392438b793c5f9c920560197545d20b58691af	fec82f2542d7f82e9fce3e16bfa4024f253adee7121973bd9d67a
69bac736f42e37302db7eca68b6fc138c3aa9a5c902c149e46cce8b42b172603	8b3c8046fa776b70821b7e50baa772a395d3d245c10bdaa4b61

b0d6aee39e988196fdc821895a1f1aa63d1c032ea880c26a15c857068f34bfd9	bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc
0e447797aa20bff416073281adb09b73c15433ab855b5cdb2d883f8c2af9c414	bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc
f13aff9e1192c081c012f974b29bf60487385eed644d506d7f82b3538c2b035f	bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc
9137e886e414b12581852b96a1d90ee875053f16b79be57694df9f93f3ead506	bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc
d26987b705f537b10a11fb9913d0acc0218a0c0ae5f27e6f821d6d987b1cd4c7	bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc
–	868a62feff8b46466e9d63b83135a7987bf6d332c13739aa11b7
87f389d8f3a63f0879aa9d9dfbbd2b2c9cf678b871b704a01b39e1eaa234020c	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
2f53109e01c431c1c1acec667adee07cf907cdc4d36429022f915654c9b7113b	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
ebe4befd2a7f941baa65248d5dea09de809e638ec8e8caffae322aa3b6863c1c	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
1892b72c053ab48edae8305ef449f2b5391921efea8b1d7c37d6d29f59edc92e	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
e83f5e0a51845d7078a3aca8ca7a5b786e8bdf284efd3e08b3472dbf3e098930	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
d0fa0bfef8b199a42f4f33145274576e5a7edeb5522fb342af41fdc16e9021e2	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
f62adc678eaadc019277640e6695143a45336c2f91019f5d9308812db1d07285	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
0dc3f66f4af3250f56a32f8e1b9e772c514f74718358d19c195e3950d370ea01	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d
7d7dc8125a26d9515d90a66bfd20d609820197c879030cb932d39b1c2998e9d4	464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d

Extracted C2s

Another indicator to group these samples could be the C2 used by each sample. To do so, we created a config extractor for these samples. The following table shows the C2s for each sample. The 2nd stage samples use a domain whereas the 3rd stage samples directly use an IP address.

NOTE: In the configuration of the 3rd stage there are 4 hardcoded IPs. In almost all cases, three of them are the same IP which belong to the C2. The remaining IP is empty in some cases, and in others it looks like a network mask (e.g. 1.0.0.0, 2.0.0.1, 4.0.0.0, 16.0.0.0). We omitted these in the following table. You can find the “raw” configuration here:https://github.com/threatray/tigerrat/blob/main/iocs/payload_configs.csv

UNPACKED HASH	ARCH.	STAGE	VARIANT	C2
f32f6b229913d68daad937cc72a57aa45291a9d623109ed48938815aa7b6005c	x64	RAT (3rd stage)	RAT-KrCERT-x64 (TigerRAT)	52.202.193.124

ed11e94fd9aa3c7d4dd0b4345c106631fe52929c6e26a0daec2ed7d22e47ada0	x64	RAT (3rd stage)	RAT-KrCERT-x64 (TigerRAT)	185.208.158.208
fec82f2542d7f82e9fce3e16bfa4024f253adee7121973bd9d67a3c79441b83c	x64	RAT (3rd stage)	RAT-KrCERT-x64 (TigerRAT)	185.208.158.208
4aadf767491077ab83c6436cf108b014fc0bf8c3bd01cc6087a0f2b80564bc08	x64	RAT (3rd stage)	RAT-Kaspersky-x64 (TigerRAT)	10.101.30.127
8b3c8046fa776b70821b7e50baa772a395d3d245c10bdaa4b6171e0c5ce3f717	x64	RAT (3rd stage)	RAT-Kaspersky-x64 (TigerRAT)	23.229.111.197
bbddcb280af742ce10842b18b9d7120632cc042a8fe42eed90fc4bc94f2d71ac	x64	RAT (3rd stage)	RAT-Kaspersky-x64 (TigerRAT)	45.58.112.77
868a62feff8b46466e9d63b83135a7987bf6d332c13739aa11b747b3e2ad4bbf	x64	RAT (3rd stage)	RAT-Kaspersky-x64 (TigerRAT)	45.58.112.77
464eaa82103f6f479e0d62dd48d2dab8ece300458136c03165d20915ee658067	x86	RAT (3rd stage)	RAT-Kaspersky-x86 (TigerRAT)	23.229.111.197
5c2f339362d0cd8e5a8e3105c9c56971087bea2701ea3b7324771b0ea2c26c6c	x64	Downloader (2nd stage)	Downloader-Kaspersky-x64	hxxp://mail.sisnet.co.kr hxxp://mail.neocyon.co
1177105e51fa02f9977bd435f9066123ace32b991ed54912ece8f3d4fbeeade4	x64	Downloader (2nd stage)	Downloader-Kaspersky-x64	hxxp://www.jinjinpig.co hxxp://mail.namusoft.k
63bae252d796bc9ac331fdc13744a72bd85d1065ef41a884dc11c6245ea933e2	x64	Downloader (2nd stage)	Downloader-Malwarebytes-x64	hxxp://snum.or.kr/skin_ hxxp://www.ddjm.co.kr/
588cdbd3ee3594525eb62fa7bab148f6d7ab000737fc0c311a5588dc96794acc	x64	Downloader (2nd stage)	Downloader-Kaspersky-x64 (Persistence)	hxxp://www.jinjinpig.co hxxp://mail.namusoft.k
49a13bf0aa53990771b7b7a7ab31d6805ed1b547e7d9f114e8e26a98f6fbee28	x86	Downloader (2nd stage)	Downloader-Kaspersky-x86	hxxp://www.conkorea.c hxxp://www.allamwith.c

MITRE ATT&CK Mapping

The table below shows the MITRE ATT&CK Mapping after combining all these attacks/campaigns from previous reports and our analysis.

TECHNIQUE	TACTIC	TECHNIQUE NAME
T1584.006	Resource Development	Compromise Infrastructure: Web Services
T1583.003	Resource Development	Acquire Infrastructure: Virtual Private Server
T1566.001	Initial Access	Phishing: Spearphishing Attachment
T1189	Initial Access	Drive-by Compromise
T1204.002	Execution	User Execution: Malicious File
T1059.007	Execution	Command and Scripting Interpreter: JavaScript
T1036.005	Defense Evasion	Masquerading: Match Legitimate Name or Location
T1027.003	Defense Evasion	Obfuscated Files or Information: Steganography
T1497.001	Defense Evasion	Virtualization/Sandbox Evasion: System Checks

T1049	Discovery	System Network Connections Discovery
T1057	Discovery	Process Discovery
T1113	Collection	Screen Capture
T1056.001	Collection	Input Capture: Keylogging
T1071.001	Command and Control	Application Layer Protocol: Web Protocols
T1095	Command and Control	Non-Application Layer Protocol
T1573.001	Command and Control	Encrypted Channel: Symmetric Cryptography
T1041	Exfiltration	Exfiltration Over C2 Channel
T1486	Impact	Data Encrypted for Impact

About Threatray

Threatray is a novel malware analysis and intelligence platform. We support all key malware defense use cases, including identification / detection, hunting, response, and analysis. Threatray helps security teams of all skill levels to effectively identify and analyze ongoing and past compromises.

At the core of Threatray are highly scalable code similarity search algorithms that find code reuse between a new and millions of known samples in seconds. Our core search algorithms do not make use of traditional byte pattern matches and are thus highly resilient to code mutations.

Our user facing features are based on the core search technology. They include best of class threat family identification and detection, easy to use real-time retro-hunting and retro-detection, cluster analysis to quickly find relevant IOCs, and low-level multi-binary analysis capabilities. Some of our binary analysis capabilities have been used for the research presented in this report.

Contact us at https://threatray.com/contact-us or https://twitter.com/threatray

Ready to find out how Threatray can protect your organization?

Subscribe to our Newsletter