Sednit reloaded: Back in the trenches

By ESET Research

Archived: 2026-04-05 21:32:56 UTC

Since April 2024, Sednit's advanced development team has reemerged with a modern toolkit centered on two paired implants, BeardShell and Covenant, each using a different cloud provider for resilience. This dual-implant approach enabled long-term surveillance of Ukrainian military personnel. Interestingly, these current toolsets show a direct code lineage to the group's 2010-era implants.

> **Key points of this blogpost:**
>
> - ESET researchers traced the reactivation of Sednit's advanced implant team to a 2024 case in Ukraine, where a keylogger named SlimAgent was deployed.
> - SlimAgent code was derived from Xagent, Sednit's flagship backdoor from the 2010s.
> - During that operation, BeardShell, a second Sednit-developed implant, was deployed. It executes PowerShell commands via a legitimate cloud provider used as its C&C channel.
> - BeardShell uses a distinctive obfuscation technique also found in Xtunnel, Sednit's network-pivoting tool from the 2010s.
> - Across 2025 and 2026, Sednit repeatedly deployed BeardShell together with Covenant, a third major piece of its modern toolkit.
> - Sednit heavily reworked this open-source implant to support long-term espionage and to implement a new network protocol based on yet another legitimate cloud provider.

Sednit profile

The Sednit group – also known as APT28, Fancy Bear, Forest Blizzard, or Sofacy – has been operating since at least 2004. The US Department of Justice named the group as one of those responsible for the Democratic National Committee (DNC) hack just before the 2016 US elections and linked the group to Unit 26165 of the GRU, a Russian Federation intelligence agency within the Main Intelligence Directorate of the Russian military. The group is also presumed to be behind the hacking of global television network TV5Monde, the World Anti-Doping Agency (WADA) email leak, and many other incidents.

What became of Sednit's advanced implant team?

The Sednit group is arguably one of the APT groups with the most impressive record of compromised targets. Notable among its known compromises are the German parliament (2015), the French television network TV5Monde (2015), and the United States Democratic National Committee (2016).

During those years of high-profile attacks, Sednit relied on an extensive set of custom implants, ranging from full-fledged espionage backdoors such as Xagent and Sedreco, to specialized toolkits such as the network-pivoting

tool Xtunnel and the data stealer for air-gapped machines USBStealer. In 2016, we extensively documented this sophisticated arsenal in our white paper En Route with Sednit.

However, in 2019, a shift occurred. Since then, and until recently, Sednit's high-end implants have rarely been observed in the wild (with only a few exceptions, such as the Graphite malware documented by Trellix in 2021), while the group simultaneously ramped up its phishing operations. The custom malware used in these phishing attacks consisted mostly of simple script-based implants. The reasons behind that technical shift remain a mystery to us.

This blogpost documents the reappearance of Sednit's high-end custom arsenal since 2024. Here we focus on attributing its modern toolsets, as prior publications by CERT-UA and Sekoia have covered their internal workings.

A boutique developer shop

Sednit maintains in-house development of its espionage implants, a distinctive trait that supports an attribution approach based on shared code artifacts.

To illustrate this capability, consider Xagent, the group's flagship backdoor during the 2010s. In 2015, we found the Xagent source code on a Linux server in Ukraine, left in an unprotected archive after the attackers had compiled it. Figure 1 shows that plugins and C&C channels were enabled or disabled by commenting code in or out – selected per target according to operational requirements – leaving little doubt that developers and operators worked in close coordination.

```
int startXAgent(wstring path)
{
    [...REDACTED...]

    AgentKernel krnl( (wchar_t *)path.c_str() );

    IAgentChannel* http_channel = new HttpChannel();
    //IAgentChannel* smtp_channel = new MailChannel();
    IAgentModule* remote_shell = new RemoteShell();
    IAgentModule* file_system = new FSModule();
    //IAgentModule* key_log = new RemoteKeylogger();

    krnl.registerChannel(http_channel);
   // krnl.registerChannel(smtp_channel);
    krnl.registerModule(remote_shell);
    krnl.registerModule(file_system);
    //krnl.registerModule(key_log);

    [...REDACTED...]
}
```

Figure 1. Xagent source code with hardcoded instantiations of plugins and communication channels (2015)

In addition, the 2018 US DOJ indictment explicitly states that Xagent was developed in-house, accusing specific members of GRU Unit 26165 of being its developers.

In this blogpost, we leverage that development footprint as an attribution mechanism. By tracking shared code artifacts across different implants, we link the group's 2010-era toolsets to those currently in use.

SlimAgent

Our account of modern Sednit activities begins with SlimAgent, an espionage implant discovered on a Ukrainian governmental machine by CERT-UA in April 2024. SlimAgent is a simple yet efficient spying tool capable of logging keystrokes, capturing screenshots, and collecting clipboard data.

Ancestors

Interestingly, we identified in ESET telemetry previously unknown samples with code similar to SlimAgent, which were deployed as early as 2018 – six years before the Ukrainian case – against governmental entities in two European countries. These samples exhibit strong code-level similarities with SlimAgent, including an identical six-step data-collection loop, shown in Figure 2. Each step is implemented in a nearly identical manner, as illustrated in Figure 3 with the routine responsible for logging the foreground window's executable; the only differences lie in the layout of the internal data structures.

```
while ( v_struct_keylog->enabled_spying )
{
  // ...
  // [REDACTED]
  // ...
  FN_sleep(&v_random_value);
  FN_log_window(v_struct_keylog);
  FN_log_typed_keys(v_struct_keylog);
  FN_log_window_text(v_struct_keylog);
  FN_log_typed_keys(v_struct_keylog);
  FN_log_clipboard(v_struct_keylog);
  FN_log_typed_keys(v_struct_keylog);
}
```

```
while ( 1 )
{
  Sleep(5u);
  FN_log_window(v_struct_keylog);
  FN_log_typed_keys(v_struct_keylog);
  FN_get_window_text(v_struct_keylog);
  FN_log_typed_keys(v_struct_keylog);
  FN_log_clipboard(v_struct_keylog);
  FN_log_typed_keys(v_struct_keylog);
}
```

Figure 2. Spying loop of 2024 SlimAgent (left) and 2018 samples (right)

```
ForegroundWindow = GetForegroundWindow();
if ( *&a1->field_150 != ForegroundWindow )
{
  *&a1->field_150 = ForegroundWindow;
  WindowThreadProcessId = GetWindowThreadProcessId(
  v4 = *&a1->field_13c;
  if ( WindowThreadProcessId != v4 )
  {
    if ( v4 )
    {
      AttachThreadInput(v4, *&a1->thread_id, 0);
      memset(*&a1->field_10, 0, 0x100u);
      *&a1->field_13c = 0;
    }
    if ( AttachThreadInput(WindowThreadProcessId, *
      *&a1->field_13c = WindowThreadProcessId;
    v5 = dwProcessId;
    if ( dwProcessId != *&a1->pid )
    {
      *&a1->pid = dwProcessId;
      a1->field_158 = 1;
      v6 = OpenProcess(0x410u, 0, v5);
```

```
ForegroundWindow = GetForegroundWindow();
if ( *&a1->field_1f0 != ForegroundWindow )
{
  *&a1->field_1f0 = ForegroundWindow;
  WindowThreadProcessId = GetWindowThreadProcessId(
  v5 = *&a1->field_1dc;
  if ( WindowThreadProcessId != v5 )
  {
    if ( v5 )
    {
      AttachThreadInput(v5, *&a1->thread_id, 0);
      memset(*&a1->field_18, 0, 0x100u);
      *&a1->field_1dc = 0;
    }
    if ( AttachThreadInput(WindowThreadProcessId, *
      *&a1->field_1dc = WindowThreadProcessId;
    v6 = dwProcessId;
    if ( dwProcessId != *&a1->pid )
    {
      *&a1->pid = dwProcessId;
      a1->field_1F8 = 0;
      v7 = OpenProcess(0x410u, 0, v6);
```

Figure 3. Logging foreground window in 2024 SlimAgent (left) and 2018 samples (right)

SlimAgent includes several features that were absent from the 2018 samples, such as encryption of the collected logs. Nevertheless, it is remarkable that samples deployed six years apart exhibit such strong code similarities.

We therefore assess with high confidence that both the 2018 samples and the 2024 SlimAgent sample were built from the same codebase. The remaining question is: where did the 2018 samples originate?

An infamous lineage

The 2018 samples have an internal name that may resonate with fellow analysts: RemoteKeyLogger.dll. This is the name of the keylogging module of Xagent, Sednit's flagship espionage backdoor from 2012 to 2018 (documented in our white paper En Route with Sednit).

Digging into some old Xagent samples (e.g., SHA-1: D0DB619A7A160949528D46D20FC0151BF9775C32), we were indeed able to find some striking similarities, such as the one shown in Figure 4. In this code, the keylogging logic is executed only if the mouse cursor has not moved more than 10 pixels (by comparing the square of the distance between the last and the current position with 0x64, i.e., 100), and it is implemented with the same API calls.

```
if ( *&a1->distance_from_last_cursor_field_164 < 0x64u
    && GetKeyboardState(*&a1->keyboard_state_field_8) )
{
   // ...
   // [REDACTED]
   // ...
   dwhkl = GetKeyboardLayout(*&a1->id_attach_field_13c);
   *v_unicode_char = 0;
   v74 = 0;
   v_scan_code = MapVirtualKeyExW(v2, 0, dwhkl);
   if ( v_scan_code
     && ToUnicodeEx(v2, v_scan_code, *&a1->keyboard_state_fie
```

```
if ( v_distance_from_last_cursor < 0x64
    && GetKeyboardState(*(this + 6)) )
{
   // ...
   // [REDACTED]
   // ...
   KeyboardLayout = GetKeyboardLayout(idAttach);
   v_scan_code = MapVirtualKeyExW(v_cur_key_idx,
   if ( v_scan_code
     && ToUnicodeEx(v_cur_key_idx, v_scan_code,
```

Figure 4. Code comparison between SlimAgent (left) and Xagent (right)

As another example, SlimAgent emits its espionage logs in the HTML format, with the application name, the logged keystrokes, and the window name in blue, red, and green, respectively. Figure 5 shows an example generated while typing and copying text in a newly created TXT file using notepad.exe. The Xagent keylogger also produces HTML logs using the same color scheme. This is illustrated in Figure 6 with the definition of the corresponding color HTML tags in the 2015 Xagent source code.

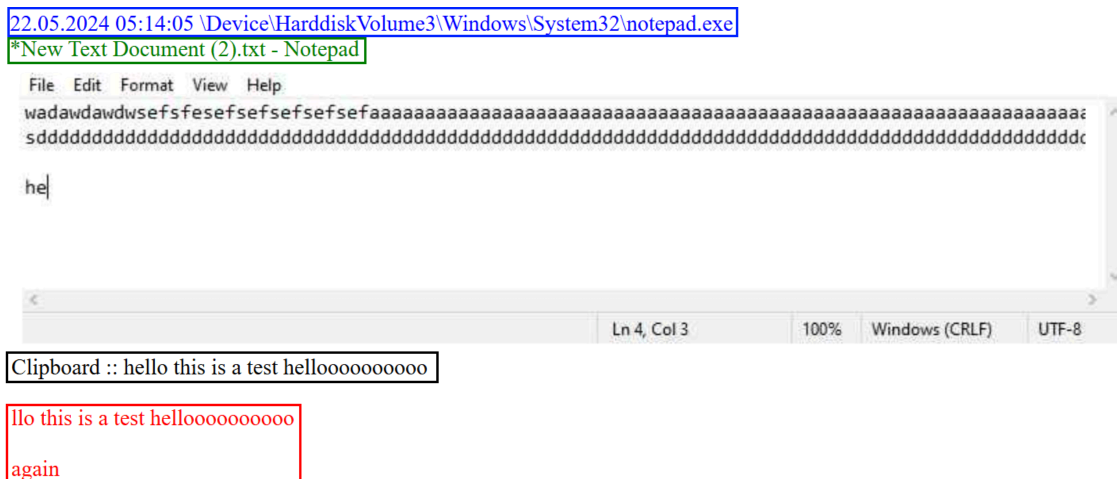

Figure 5. Example of an HTML report produced by SlimAgent

```
namespace threadkeyloggervars
{
/* <xmlblok config="MESSAGE" type="u_char"><![CDATA[ */  static /* ]]> */
        /* <type><![CDATA[ */ char /* ]]> */ /* </type> */
        /* <static><![CDATA[ */  TAG_APPLICATION [] = /* ]]></static> */
        /* <config operation="'ascii'={byte}"><![CDATA[ */ "<font size=4 color=blue><pre>" /* ]]> */ ; /* </config> */
    /* </xmlblok> */

    /* <xmlblok config="MESSAGE" type="u_char"><![CDATA[ */  static /* ]]> */
        /* <type><![CDATA[ */ char /* ]]> */ /* </type> */
        /* <static><![CDATA[ */  TAG_KEYLOG [] = /* ]]></static> */
        /* <config operation="'ascii'={byte}"><![CDATA[ */ "<font color=red><pre>" /* ]]> */ ; /* </config> */
    /* </xmlblok> */

    /* <xmlblok config="MESSAGE" type="u_char"><![CDATA[ */  static /* ]]> */
        /* <type><![CDATA[ */ char /* ]]> */ /* </type> */
        /* <static><![CDATA[ */  TAG_WINDOW [] = /* ]]></static> */
        /* <config operation="'ascii'={byte}"><![CDATA[ */ "<font color=green><pre>" /* ]]> */ ; /* </config> */
    /* </xmlblok> */
```

Figure 6. Xagent source code with definitions of the log colors (2015)

Based on these similarities, we believe that SlimAgent is an evolution of the Xagent keylogger module, which has been deployed as a standalone component since at least 2018. Moreover, because Xagent is a custom toolset used exclusively by the Sednit group for more than six years, we attribute SlimAgent to Sednit with high confidence.

This raises a question: why would Sednit reuse an implant derived from such a well-known codebase? One possible explanation is reduced development capacity. However, SlimAgent was not the only implant found on the Ukrainian machine in 2024; BeardShell – a much more recent addition to Sednit's custom arsenal – was deployed there as well.

BeardShell

BeardShell is a sophisticated implant capable of executing PowerShell commands within a .NET runtime environment, while leveraging the legitimate cloud storage service Icedrive as its C&C channel.

This component bears the marks of intense development efforts and is the primary reason we believe that Sednit's advanced development team is once again active. For example, because Icedrive does not provide a publicly documented API, the developers reimplemented the requests made by the official Icedrive client. Whenever changes to Icedrive's private API disrupt BeardShell communications, Sednit developers produce an updated version *within hours* to restore access.

A mathematical blast from the past

While we could not find other malware families directly related to BeardShell, we uncovered a surprising similarity with past Sednit tooling, starting with a C++ static initializer executed at the very start of BeardShell. This routine's purpose, whose code is shown in Figure 7, is to decrypt the authentication token for the Icedrive cloud storage.

```
int FN_decrypt_icedrive_token()
{
  while ( 2u / (x * x + 1) + 2 == y * y + 5 )
  {
    FN_decrypt_token(encrypted_token, token);
    atexit(FN_clear_decrypted_token);
    y = 6 - y + 6;
  }
  FN_decrypt_token(encrypted_token, token);
  return atexit(FN_clear_decrypted_token);
}
```

Figure 7. Static initializer to decrypt Icedrive authentication token

The routine contains a textbook example of the obfuscation technique known as opaque predicate insertion (highlighted in the red box in Figure 7):

- An arithmetic expression evaluating to zero for all possible inputs – named x and y in Figure 7 – is used as a condition for a while loop. In practice, the loop body is never executed, because the predicate $2\ (x^2 + 1) + 2 = y^2 + 5$ has no integer solution.
- The body of this artificial loop consists of two original instructions (shown in the yellow box in Figure 7), plus a dummy update of the input variable y to mimic a real loop body structure.
- Following the fake loop are the two original instructions that will be executed: a call to the Icedrive token decryption routine and the registration of a cleaner routine.

Opaque predicates are typically used to hinder static analysis but are not particularly useful in such a small routine. Note that other BeardShell static initializers – which are not handling important data – are protected with the same technique, so it seems that the developers simply applied the protection to all of them indiscriminately.

Now, the predicate formula can be simplified as (by subtracting 2 on both sides) $2\ (x^2 + 1) = y^2 + 3$. Interestingly, that *same* opaque predicate was used in Xtunnel, a network-pivoting tool used exclusively by Sednit, from 2013 to 2016, and documented in our white paper En Route with Sednit. Figure 8 shows an example of obfuscated code from Xtunnel (SHA-1: 99B454262DC26B081600E844371982A49D334E5E), with an if statement whose predicate cannot be true.

```
if ( 2u / (x * x + 1) == y * y + 3 )
{
  *(v30 + 4) = 0x8235;
  v31[3] = 10;
  x = ((x - 9) ^ x) + (((x - 9) ^ x) & (x - 9));
}
*(v30 + 4) = 0x8235;
v31[3] = 10;
```

Figure 8. Xtunnel opaque predicate (2015)

Not only is the predicate identical to the one used in BeardShell, but the never-executed block is built in a similar fashion, by duplicating the two original instructions (in the yellow box) and doing a dummy update of one of the predicate inputs (here, x).

To the best of our knowledge, this opaque predicate has not been observed anywhere else except in Xtunnel. One might even wonder if it could not have been used as a false flag, especially since it was publicly mentioned as being unique to Xtunnel, for example in a BlackHat Europe 2016 presentation. Nevertheless, a false flag operation would have likely used the identical predicate, not the variant with +2 on both sides of the equation.

The shared use of this rare obfuscation technique, combined with its co-location with SlimAgent, leads us to assess with high confidence that BeardShell is part of Sednit's custom arsenal.

Since the initial 2024 case, Sednit has continued deploying BeardShell through 2025 and into 2026, primarily in long-term espionage operations targeting Ukrainian military personnel. To maintain persistent access to these high-value targets, Sednit systematically deploys *another* implant alongside BeardShell: Covenant, the final component of its modern arsenal.

Covenant

Covenant is an open-source .NET post exploitation framework first released in February 2019. It enables the creation and management of .NET implants through a web-based dashboard – see the example in Figure 9 – and provides over 90 built-in tasks, supporting capabilities such as data exfiltration, target monitoring, and network pivoting.

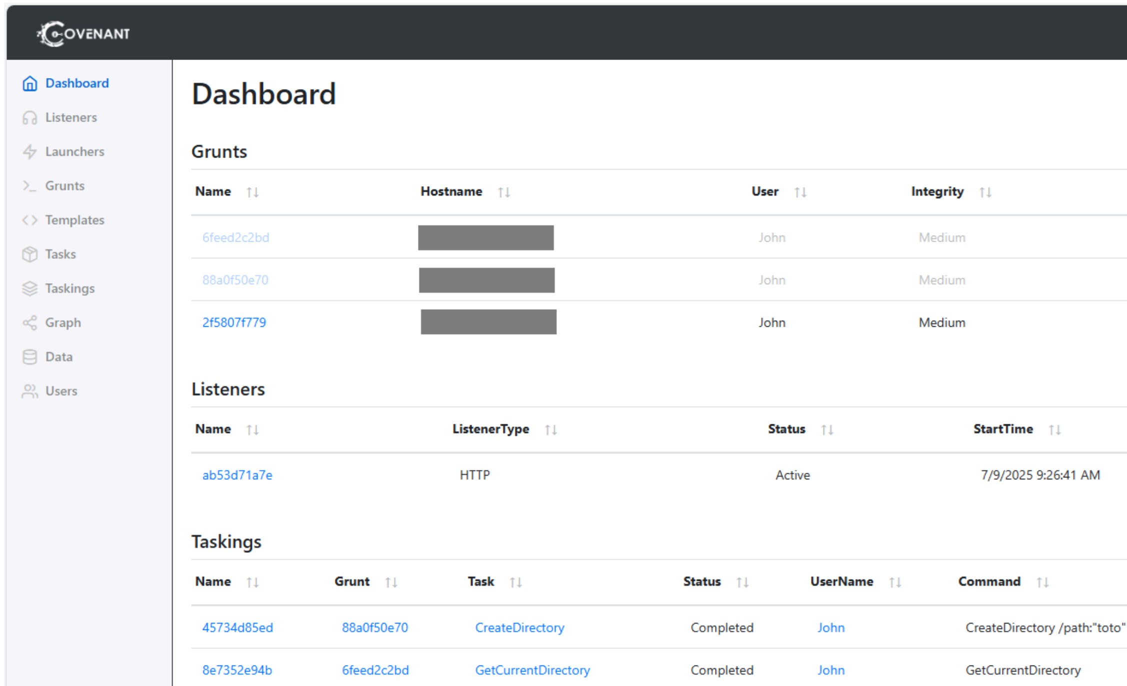

Figure 9. Covenant dashboard

Since 2023, Sednit developers have made a number of modifications and experiments with Covenant to establish it as their primary espionage implant, keeping BeardShell mainly as a fallback in case Covenant encounters operational issues, such as the takedown of its cloud-based infrastructure.

For example, Sednit replaced Covenant's original implant name-generation mechanism with a deterministic method (see Figure 10), producing identifiers derived from machine characteristics rather than generating a new random value at each execution (see the Name column in the Grunts section in Figure 9). This modification illustrates how Sednit adapted Covenant for long-term espionage rather than for short-term, post-exploitation activity: in long-running operations, having the same machine appear under different identifiers after each reboot would clutter the dashboard and reduce operational efficiency.

```
private static string GenerateUniqueId(string shared_secret_password)
{
    string result = Guid.NewGuid().ToString().Replace("-", "").Substring(0, 10);
    string directoryRoot = Directory.GetDirectoryRoot(Environment.SystemDirectory);
    try
    {
        //...[REDACTED]...
        if (GruntStager.GetVolumeInformation(directoryRoot, ...)
        {
            text = string.Format("{0:X4}{1:X4}", num >> 16, num & 65535U);
        }
        text += Environment.UserDomainName;
        text += Environment.UserName;
        text += shared_secret_password;
        byte[] b = MD5.Create().ComputeHash(Encoding.Default.GetBytes(text));
        Guid guid = new Guid(b);
        result = guid.ToString().Replace("-", "").Substring(0, 10);
    }
    catch (Exception)
    {
    }
    return result;
}
```

Figure 10. Grunt ID generation routine added by Sednit

Sednit also changed Covenant's execution flow, which is a two-stage implant, probably to avoid behavioral detection. Instead of having the first-stage downloader invoke the first method of the second-stage .NET assembly using a fixed index (as originally implemented), they introduced a DisplayName attribute and iterated over method attributes to find the entry point. In early 2023 variants, Sednit developers even experimented with embedding both stages into a single binary.

Covenant officially supports only HTTP and SMB, which leads to Sednit's most significant Covenant modification: the addition of a cloud-based network protocol. To achieve this, Sednit developers leveraged the C2Bridge project, a standalone framework created by Covenant's original author to facilitate integration of new communication protocols. With C2Bridge, developers need only implement a class conforming to the IMessenger interface on the implant side, providing Read and Write methods to manage low-level communications. C2Bridge can then run as a standalone component on the controller to relay messages, while new implants created by the controller use the implemented communication methods.

Figure 11 shows the classes introduced by Sednit developers to communicate with the Filen cloud provider, used since July 2025. The FilenMessenger class implements IMessenger and relies on FilenClient to interact with the Filen API. Previously, in 2023, Sednit's Covenant abused the legitimate cloud service pCloud, and in 2024–2025, Koofr, using similar implementations.

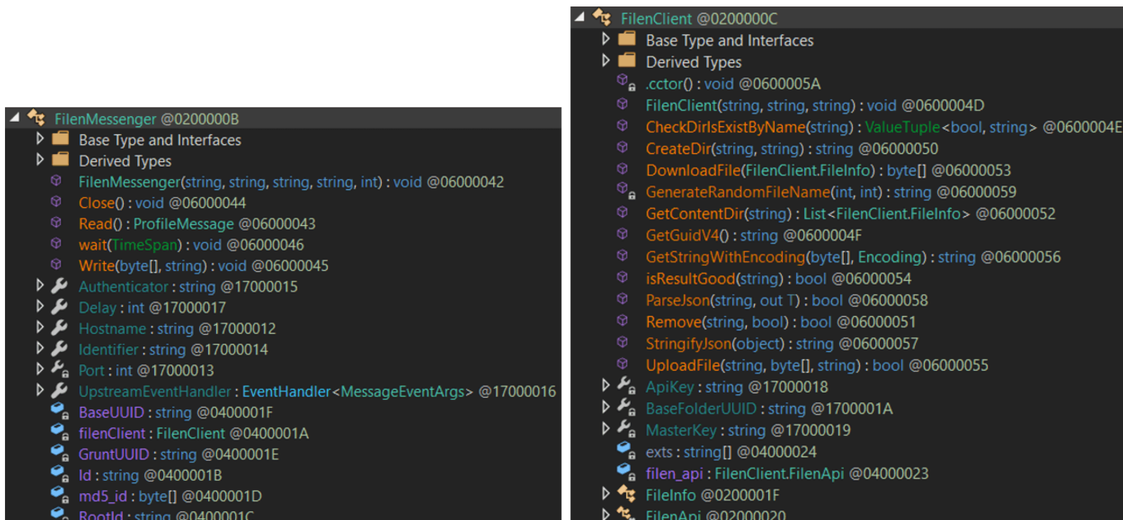

Figure 11. Additional Covenant classes handling communications with a Filen cloud drive

These adaptations show that Sednit developers acquired deep expertise in Covenant – an implant whose official development ceased in April 2021 and may have been considered unused by defenders. This surprising operational choice appears to have paid off: Sednit has successfully relied on Covenant for several years, particularly against selected targets in Ukraine. For instance, in 2025, our analysis of Sednit-controlled Covenant cloud drives revealed machines that had been monitored for more than six months. In January 2026, Sednit also deployed Covenant in a series of spearphishing campaigns exploiting the CVE-2026-21509 vulnerability, as reported by CERT-UA.

Conclusion

In this blogpost, we have shown that Sednit's advanced development team is active once again, operating an arsenal centered on two implants – BeardShell and Covenant – deployed in tandem and each leveraging a different cloud provider. This setup enables operators to reestablish access quickly if the infrastructure for one is taken down. We believe that this dual-implant strategy is not new. For example, in the 2021 campaign documented by Trellix, Sednit deployed two implants in parallel: Graphite, which used OneDrive as its C&C channel, and PowerShell Empire, which relied on separate dedicated infrastructure.

The sophistication of BeardShell and the extensive modifications made to Covenant demonstrate that Sednit's developers remain fully capable of producing advanced custom implants. Furthermore, the shared code and techniques linking these tools to their 2010-era predecessors strongly suggest continuity within the development team.

This raises the question of what these developers were doing during all these years, when the security community primarily observed phishing activity from Sednit. One possibility is that advanced development efforts were reactivated following the Russian invasion of Ukraine. Another is that they never stopped working, but instead became more cautious.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*

> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

Files

A comprehensive list of indicators of compromise (IoCs) and samples can be found in our GitHub repository.

SHA-1	Filename	Detection	Description
5603E99151F8803C13D4 8D83B8A64D071542F01B	eapphost.dll	Win64/Spy.KeyLogger.LS	SlimAgent.
6D39F49AA11CE0574D58 1F10DB0F9BAE423CE3D5	tcpiphlpsvc.dll	Win64/BeardShell.A	BeardShell.

MITRE ATT&CK techniques

This table was built using version 18 of the MITRE ATT&CK framework**.**

Tactic	ID	Name	Description
Resource Development	*T1583.006*	Acquire Infrastructure: Web Services	BeardShell relies on Icedrive cloud storage. Covenant relies on Filen cloud storage.
	T1587.001	Develop Capabilities: Malware	BeardShell and SlimAgent are custom malware.
Execution	*T1059.001*	Command and Scripting Interpreter: PowerShell	BeardShell executes PowerShell commands.
	T1129	Shared Modules	BeardShell and SlimAgent are full-fledged DLL files.
Privilege Escalation	*T1546.015*	Event Triggered Execution: Component Object Model Hijacking	BeardShell and SlimAgent are made persistent by hijacking COM objects.
Defense Evasion	*T1027*	Obfuscated Files or Information	BeardShell Icedrive token decryption is obfuscated.
	T1140	Deobfuscate/Decode Files or Information	BeardShell decrypts its strings.

Tactic	ID	Name	Description
	T1480	Execution Guardrails	BeardShell only executes in taskhost.exe or taskhostw.exe. SlimAgent only executes in explorer.exe.
	T1564	Hide Artifacts	SlimAgent logs are written into a hidden file.
Discovery	T1082	System Information Discovery	BeardShell sends a fingerprint of the compromised machine.
Collection	T1005	Data from Local System	BeardShell, Covenant, and SlimAgent collect data from a compromised machine.
	T1056.001	Input Capture: Keylogging	SlimAgent performs keylogging.
	T1113	Screen Capture	SlimAgent captures screenshots of the compromised machine.
	T1115	Clipboard Data	SlimAgent collects clipboard data.
Command and Control	T1001	Data Obfuscation	BeardShell exfiltrates data in fake images.
	T1071.001	Application Layer Protocol: Web Protocols	BeardShell and Covenant use HTTPS for C&C.
	T1102	Web Service	BeardShell gets commands from Icedrive. Covenant gets commands from Filen.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	BeardShell communications with Icedrive are encrypted using HTTPS. Covenant communications with its controller uses RSA-encrypted session keys.
Exfiltration	T1567	Exfiltration Over Web Service	BeardShell exfiltrates data to Icedrive. Covenant exfiltrates data to Filen.

Source: https://www.welivesecurity.com/en/eset-research/sednit-reloaded-back-trenches/