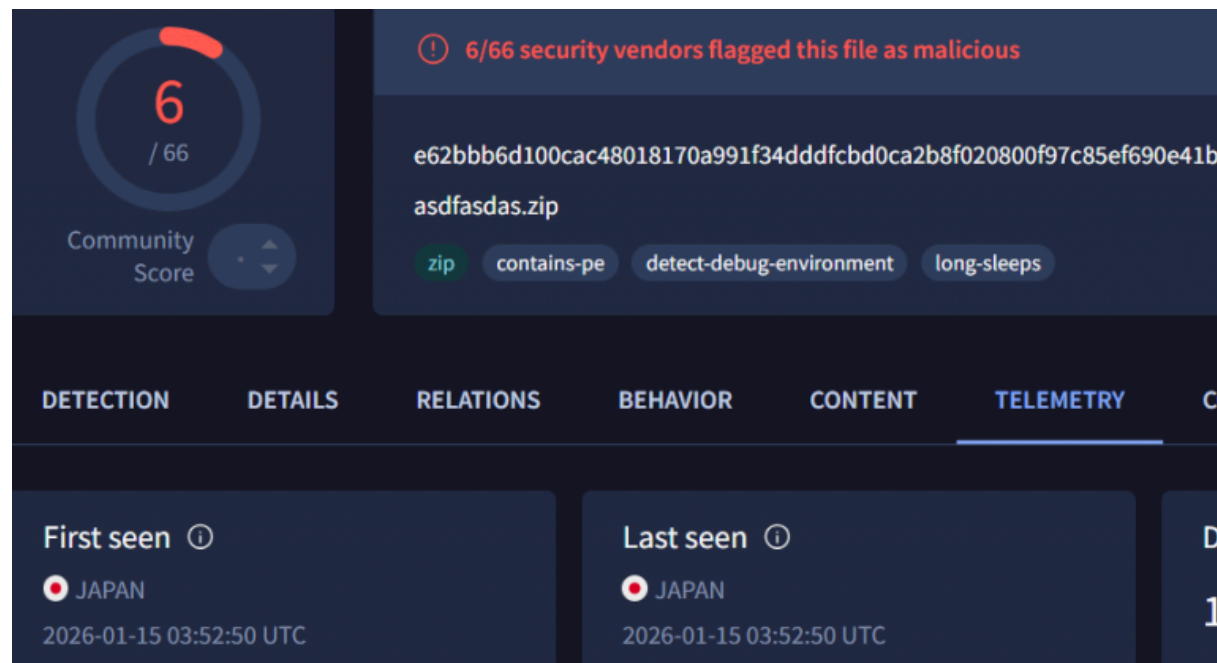


BOFの実行などに対応した新たなマルウェアSLOTAGENT – IIJ Security Diary

sect.iiij.ad.jp/blog/2026/04/slotagent

Naoki Takayama

2026年4月28日



2026年初頭、IIJは日本からパブリックマルウェアリポジトリにアップロードされたZIPファイルに含まれる未知のマルウェアを発見しました。

Name	Size	Packed Size	Modified	Created	Accessed
db.config	120 258	120 298	2026-01-14 02:09	2026-01-14 02:09	2026-01-14 19:52
WindowsOobeAppHost.AOT.dll	11 264	4 297	2026-01-14 02:09	2026-01-14 02:09	2026-01-14 19:52
WindowsOobeAppHost.AOT.exe	104 960	52 517	2026-01-14 02:09	2026-01-14 02:09	2026-01-14 19:52

0 / 3 object(s) selected

図1: ZIPファイルの内容

マルウェア解析を行った結果、これは攻撃者によって発出された様々な命令を実行できる多機能なRATであることが判明しました。特徴的な要素として、Cobalt Strikeを含む複数のPost-Exploitationフレームワークと同じくBOF (Beacon Object File) 形式のペイロードが実行できること、タイムストンピングのようなアンチフォレンジック機能が実装されていることなどが挙げられます。

内部で使用している文字列 (ファイルパス等) から、我々はこのマルウェアをSLOTAGENTと呼んでいます。この記事ではSLOTAGENTの解析結果や検知ルールなどを共有します。

ローダおよびシェルコード

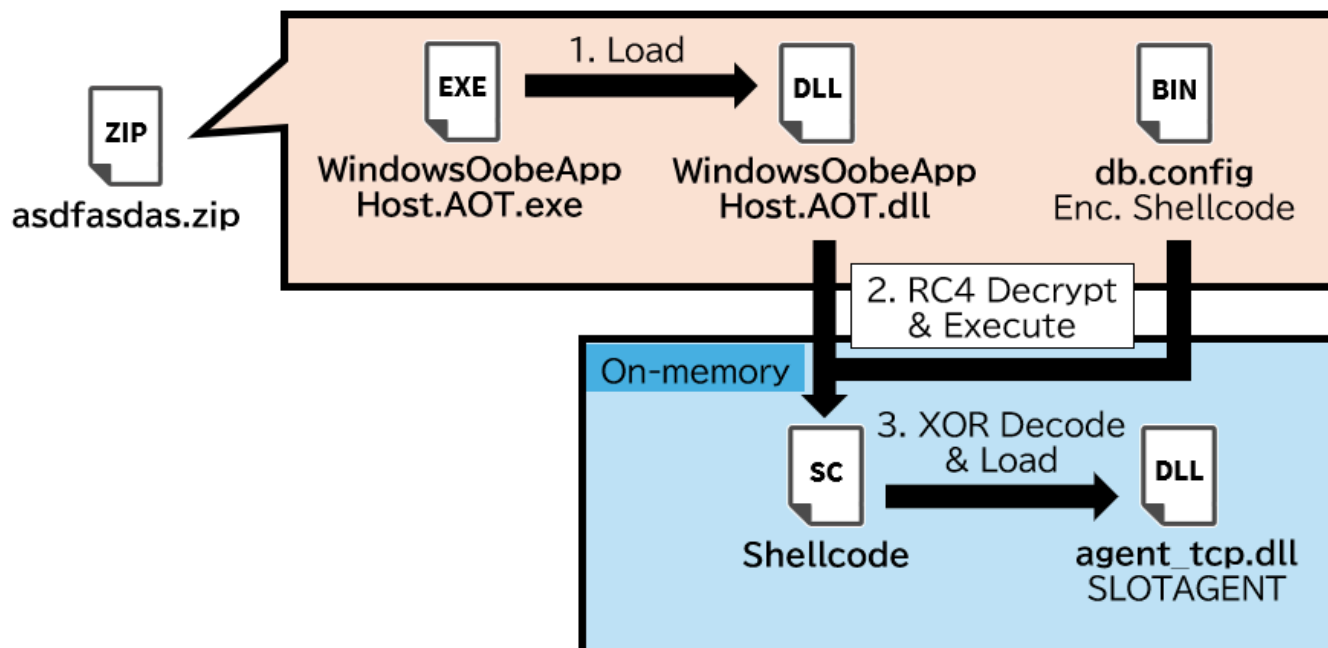


図2: SLOAGENTの実行フロー

ZIPファイルに含まれる実行ファイル `WindowsOobeAppHost.AOT.exe` が実行されると、DLLファイル `WindowsOobeAppHost.AOT.dll` のエクスポート関数 `__managed__Main` が呼び出されます。

Name	Address
DllCanUnloadNow	00000000180002D60
DllGetActivationFactory	00000000180002D60
DllMem	00000000180002D60
DotNetRuntimeDebugHeader	00000000180002D60
<code>__managed__Main</code>	00000000180002D60
DllEntryPoint	00000000180002E00

図3: `WindowsOobeAppHost.AOT.dll` のエクスポート関数の一覧

`__managed__Main` 関数が呼び出されると、まずXORおよびROR11を組み合わせた独自のアルゴリズムを使用したAPI Hashingによって `NtCreateFile` といった複数のWindows API関数アドレスを動的解決します。

```
1 __int64 __fastcall malware_calculate_string_hash(unsigned __int16 *ptrString)
2 {
3     char lowerChar; // [rsp+20h] [rbp-18h]
4     __int64 hashValue; // [rsp+28h] [rbp-10h]
5
6     hashValue = 0xAF6F01BD5B2D7583uLL;
7     while ( *ptrString )
8     {
9         lowerChar = malware_char_to_lower(*ptrString++);
10        hashValue = (lowerChar ^ (unsigned __int64)hashValue)
11                    - 0x4B0D7E8D6E7D8BE3LL
12                    + malware_ROR8(lowerChar ^ (unsigned __int64)hashValue, 11);
13    }
14    return hashValue;
15 }
```

図4: SLOAGENTローダのAPI Hashingで使用するハッシュ関数

その後、ファイル `db.config` 内のデータを鍵 `easdbadshyfab` でRC4復号して、`NtCreateThreadEx` 関数等を使用して復号したデータをシェルコードとして実行します。

```
46 RC4_SboxHelperFunc(encFileBuffer, a1, a2);
47 strcpy(rc4Key, "easdbadshyfab");
48 memset(&rc4Key[14], 0, 0xF2u);
49 memset(sbox, 0, 0x100u);
50 rc4Key_0 = rc4Key;
51 rc4Key_length = -1;
52 do
53     ++rc4Key_length;
54 while ( rc4Key_0[rc4Key_length] );
55 RC4_KSA((__int64)sbox, (__int64)rc4Key, rc4Key_length);
56 RC4_PRGA(sbox, encFileBuffer, (unsigned int)encFileSize);
```

図5: `db.config` ファイル内のデータをRC4復号

```
64 if ( NtProtectVirtualMemory(-1, &baseAddress, &regionSize, PAGE_EXECUTE_READ, oldProtect) )
65     return 0;
66 v24 = NtCreateThreadEx;
67 hThread = 0;
68 if ( NtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, 0, -1, encFileBuffer, 0, 0, 0, 0, 0, 0) || !hThread )
69     return 0;
70 if ( a3_1 && NtResumeThread )
71 {
72     v25 = NtResumeThread;
73     NtResumeThread(hThread, 0, 0);
74 }
```

図6: シェルコードを実行

シェルコード内にはXORエンコードされたSLOTAGENT (DLLファイル)が含まれていて、実行後にオフセット0x297に存在する16バイトのキーを使ってXORデコードした後、リフレクティブロードします。

```
45 v13 = NtProtectVirtualMemory;
46 ptrEncodedData_0 = ptrShellcodeBase + 0x15C2;
47 v17 = ptrShellcodeBase + 0x1000;
48 ptrEncodedData_1 = ptrShellcodeBase + 0x15C2;
49 encodedSize_0 = 0x1C000;
50 NtProtectVirtualMemory(v2, v1, (void **)&ptrEncodedData_1, -1, &encodedSize_0, 4);
51 ptrXorKey = ptrShellcodeBase + 0x297;
52 ptrEncodedChar = ptrEncodedData_0;
53 encodedSize_1 = 0x1C000;
54 cnt = 0;
55 do
56 {
57     *ptrEncodedChar++ ^= ptrXorKey[cnt];
58     cnt = (unsigned int)(cnt + 1);
59     if ( (_DWORD)cnt == 16 )
60         cnt = 0;
61     --encodedSize_1;
62 }
63 while ( encodedSize_1 );
```

図7: シェルコード内のXORデコード処理

キー 値

aid	感染ホスト固有のID
h	コンピュータ名
u	ユーザ名
i	ローカルIPアドレス
m	MACアドレス
o	OSバージョン情報
x	用途不明 (ランダムなバイト列)
p	PID
n	モジュールファイル名
e	プロセスが管理者権限で実行されているか否か
st	スリープタイム (5)
j	ジッター (10)
_d	用途不明 (ランダムなバイト列)

次にコマンドの受信を試行して、成功した場合は対応する処理を実行します。今回観測したSLOTAGENTに実装されていたコマンドは以下の通りです。

コマンド名	説明
screenshot	スクリーンショット (BMP形式) を取得する
download	ファイルをダウンロードする
shell	リモートシェルを実行する
sysinfo	システム情報を取得する
ps	実行中のプロセスの情報を取得する
ls	作業ディレクトリ内のファイル・フォルダのパスを取得する
pwd	現在の作業ディレクトリのパスを取得する
upload	ファイルをアップロードする (1)
sleep	C2サーバとの通信間隔を変更する
kill	taskkill コマンドを使用して任意のプロセスをキルする
cat	ファイルの内容を取得する
write	ファイルへの文字列の書き込み
rm	ファイルを削除する
touch	ファイルの \$SI 属性内のタイムスタンプを改ざんする
bof	BOF (Beacon Object File) を実行する
savememory	プロセスのメモリダンプを取得する
upload_chunked	ファイルをアップロードする (2)
download_url	指定したURLからファイルをダウンロードする
exit	マルウェアの実行を終了する
destroy	マルウェアに関連するファイルを削除して、実行を終了する

コマンドの実行結果は、データの受信時に使用したものと同一プロトコルを使用して (JSONデータにまとめられて) C2サーバに送信されます。

解析妨害

SLOTAGENTには複数の解析 (リバースエンジニアリング) 妨害を目的とした機能が実装されていました。

API Hashing

SLOTAGENTは実行中、複数のWindows API関数を呼び出しますが、これらのAPI関数のアドレスはDJB2ベースのアルゴリズムを用いたAPI Hashingによって実行時に動的解決されます。

```

1  __int64 __fastcall malware_calculate_string_hash(unsigned __int8 *a1)
2  {
3      __int64 result; // rax
4      int v2; // edx
5
6      for ( result = 5218; ; result = (unsigned int)(v2 + 33 * result) )
7      {
8          v2 = *a1++;
9          if ( !v2 )
10             break;
11             if ( (unsigned int)(v2 - 65) <= 0x19 )
12                 v2 += 32; // lower()
13             }
14         return result;
15     }

```

図10: SLOTAGENTのAPI Hashingで使用するハッシュ関数

```

g_GetComputerNameA = (__int64 (__fastcall *)(_QWORD, _QWORD))malware_resolve_winapi_from_hash(v10, 0x9E71D593);

```

```

__int64 __fastcall malware_get_computer_name(_BYTE *computerNameAstr, __int64 bufferSize)
{
    __int64 result; // rax
    char *v5; // rax
    _DWORD nsize[7]; // [rsp+3Ch] [rbp-1Ch] BYREF

    if ( !g_GetComputerNameA
        || (nsize[0] = bufferSize, result = g_GetComputerNameA(computerNameAstr, nsize), !(_DWORD)result) )
    {
        v5 = malware_crypt_data(&_EncryptedUnknown); // Decrypted: b'unknown\x00'
        result = malware_strcpy_like_func(computerNameAstr, (__int64)v5, bufferSize - 1);
        computerNameAstr[bufferSize - 1] = 0;
    }
    return result;
}

```

図11: ハッシュ値をもとにAPI関数のアドレスを呼び出している様子

文字列の暗号化

コマンド名を含むSLOTAGENT内部のほとんどの文字列は、TEAに近いアルゴリズムによって暗号化されていました。これらの文字列はすべて実行時に復号されるため、静的解析では内容を確認することができません。

```

1 char *__fastcall malware_crypt_data(char *a1)
2 {
3     int v1; // edx
4     char *v3; // r8
5     int v4; // ecx
6     char v5; // r10
7     int v6; // eax
8     int v7; // eax
9     __int64 i; // rdx
10    unsigned int v9; // r11d
11    unsigned int v10; // r10d
12    unsigned int v11; // ecx
13    unsigned int v12; // edi
14    char *v13; // r8
15
16    v1 = (unsigned __int8)*a1;
17    if ( (_BYTE)v1 )
18    {
19        v3 = a1 + 1;
20        if ( (v1 & 0x80u) != 0 )
21        {
22            v1 &= 0x7Fu;
23            v4 = 7;
24            do
25            {
26                v5 = *v3++;
27                v6 = (v5 & 0x7F) << v4;
28                v4 += 7;
29                v1 |= v6;
30            }
31            while ( v5 < 0 );
32        }
33        memset(a1, 0, v3 - a1);
34        v7 = v1 / 8;
35        for ( i = 0; v7 > (int)i; ++i )
36        {
37            v9 = *(_DWORD *)&v3[8 * i];
38            v10 = *(_DWORD *)&v3[8 * i + 4];
39            v11 = 0x3D1FA148;
40            do
41            {
42                v12 = v11 + dword_3679FC780[(v11 >> 11) & 3];
43                v11 += 0x481EAE9D;
44                v10 -= v12 ^ (v9 + ((v9 >> 4) ^ (32 * v9)));
45                v9 -= (v11 + dword_3679FC780[v11 & 3]) ^ (v10 + ((v10 >> 4) ^ (32 * v10)));
46            }
47            while ( v11 );
48            *(_DWORD *)&v3[8 * i] = v9;
49            *(_DWORD *)&v3[8 * i + 4] = v10;
50        }
51        return v3;
52    }
53    else
54    {
55        v13 = a1;
56        do
57            ++v13;
58        while ( !*v13 );
59        return v13;
60    }
61 }

```

図12: 暗号化アルゴリズムが実装された関数

私たちは解析の効率化のために、SLOTAGENT内の暗号化された文字列を静的に復号するIDAPythonスクリプトを作成・公開しました。ぜひインシデント対応や脅威ハンティング等でお役立てください。

Output	
0x3679F4980	-> 'port':
0x3679F4980	-> 'port':
0x3679F49A0	-> '%s{"pid":%lu,"ppid":%lu,"name": "%s", "threads":%lu}'
0x3679F49E0	-> "psk":
0x3679F4A00	-> '{"success":true,"width":%d,"height":%d,"format":"bmp","data": "%s"}'
0x3679F4A60	-> '{"t": "%s", "s": "%s", "r": "%s"}'
0x3679F4AA0	-> "token":
0x3679F4AC0	-> '{"token": "%s", "ts": %u}'
0x3679F4AE0	-> '{"token": "%s", "ts": %u}'
0x3679F4B00	-> "ts":
0x3679F4B20	-> '{"success":true,"filename": "%s", "size": %lu}'
0x3679F4B60	-> "url":
0x3679F4B70	-> 'kernel32.dll'
0x3679F4B90	-> 'kill'
0x3679F4BA0	-> 'kill:'
0x3679F4BB0	-> 'kill:'
0x3679F4BC0	-> '127.0.0.1'
0x3679F4BE0	-> 'ls'
0x3679F4BF0	-> 'ls'
0x3679F4C00	-> 'ls:'
0x3679F4C10	-> 'ls:'
0x3679F4C20	-> '%02X:%02X:%02X:%02X:%02X:%02X'
0x3679F4C60	-> 'Memory allocation failed'
0x3679F4CA0	-> 'Global\\\\AgentSlot_%d'
0x3679F4CC0	-> '[No output captured]'
0x3679F4CE0	-> 'ntdll.dll'
0x3679F4D00	-> 'Path conversion failed'
0x3679F4D20	-> 'Failed to create process snapshot'
0x3679F4D60	-> 'ps'
0x3679F4D70	-> 'ps'
0x3679F4D80	-> 'pwd'
0x3679F4D90	-> 'pwd'
0x3679F4DA0	-> 'Failed to get current directory'
0x3679F4DE0	-> 'rm'
0x3679F4DF0	-> 'rm'
0x3679F4E00	-> 'rm:'
0x3679F4E10	-> 'rm:'
0x3679F4E20	-> 'savememory'
0x3679F4E40	-> 'savememory'
0x3679F4E60	-> 'screenshot'
0x3679F4E80	-> 'screenshot'
0x3679F4EA0	-> 'shell'
0x3679F4EB0	-> 'shell'
0x3679F4EC0	-> 'shell:'
0x3679F4ED0	-> 'shell:'
0x3679F4EE0	-> 'sleep:'
0x3679F4EF0	-> 'sleep:'
0x3679F4F00	-> 'Sleep updated'
0x3679F4F20	-> 'Sleep updated'
0x3679F4F40	-> 'C:\\\\ProgramData\\\\.agent_slot_%d.id'
0x3679F4F80	-> 'Windows Server 10.0.%lu'
0x3679F4FA0	-> 'Windows Server 2008 (Build %lu)'
0x3679F4FE0	-> 'Windows Server 2008 R2 (Build %lu)'
0x3679F5020	-> 'Windows Server 2012 (Build %lu)'
0x3679F5060	-> 'Windows Server 2012 R2 (Build %lu)'
0x3679F50A0	-> 'Windows Server 2016 (Build %lu)'
0x3679F50E0	-> 'Windows Server 2019 (Build %lu)'

Python

図13: スクリプトの実行結果

さいごに

SLOTAGENTは非常に多機能なRATであり、今後もサイバー攻撃で使用される可能性があります。にてファイルハッシュや通信先を、にてSLOTAGENTの検知が可能なYARAルールを共有します。よろしければご活用ください。

Appendix A: IoCs

ファイル

SHA256	説明
e62bbb6d100cac48018170a991f34dddfcbcd0ca2b8f020800f97c85ef690e41b	実行に必要なファイルを含むZIPアーカイブ
97e0714ee7279feb558aa38ab9d4c279731d3000c501aff7ad5c2967c3cb987f	暗号化されたSLOTAGENT (db.config)
c1681d3aae736585c1dd656fe3ad66dafd3712ad4125e09fc97a4f1e5f367548	SLOTAGENTローダー (windows0obeAppHost.AOT.dll)
a9c46b67ff938930b16b377df9ddf86f3a56ef9876267387f30299a069c98472	SLOTAGENT (メモリ上で実行される)

ネットワーク

43.156.59.[110 (SG; AS 132203; Tencent Building, Kejizhongyi Avenue)

Appendix B: YARAルール

SLOTAGENTローダ

```
rule SLOTAGENT_Loader {
  meta:
    description = "SLOTAGENT loader"
    author = "Internet Initiative Japan Inc."
    hash = "c1681d3aae736585c1dd656fe3ad66dafd3712ad4125e09fc97a4f1e5f367548"

  strings:
    $xorKey = { C6 84 24 ?? 01 00 00 65
                C6 84 24 ?? 01 00 00 61
                C6 84 24 ?? 01 00 00 73
                C6 84 24 ?? 01 00 00 64
                C6 84 24 ?? 01 00 00 62
                C6 84 24 ?? 01 00 00 61
                C6 84 24 ?? 01 00 00 64
                C6 84 24 ?? 01 00 00 73
                C6 84 24 ?? 01 00 00 68
                C6 84 24 ?? 01 00 00 79
                C6 84 24 ?? 01 00 00 66
                C6 84 24 ?? 01 00 00 61
                C6 84 24 ?? 01 00 00 62
                C6 84 24 ?? 01 00 00 00 }
    $xorLoop = { B9 00 01 00 00 F3 AA    }

  condition:
    all of ($xor*)
}Code language: plaintext (plaintext)
```

SLOTAGENT

```
rule SLOTAGENT_RAT {
  meta:
    description = "SLOTAGENT paths and internal strings (in plaintext)"
    author = "Internet Initiative Japan Inc."
    hash = "a9c46b67ff938930b16b377df9ddf86f3a56ef9876267387f30299a069c98472"

  strings:
    $path0 = "/api/v1/users/profile"
    $path1 = "/api/v1/session/refresh"
    $path2 = "/api/v1/analytics/event"
    $path3 = "/api/v1/stream/data"

    $str0 = "{\"error\":\"Failed to get drives\"}"
    $str1 = "
{\\name\\":\\\"%c:\\\",\\\"type\\\":\\\"dir\\\",\\\"size\\\":0,\\\"modified\\\":0,\\\"accessed\\\":0,\\\"changed\\\":0,\\\"drive_type\\\"
\\\":\\\"%s\\\"}"

  condition:
    all of ($path*) or all of ($str*)
}Code language: plaintext (plaintext)
```

カテゴリー:[マルウェア解析](#)

タグ: [Malware](#)