RED TEAM, RESEARCH, X-C3LL

Don't use commands, use code: the tale of Netsh & PortProxy

Jun 11, 2021 _ Adepts of 0xCC

Dear Fell**owl**ship, today's homily is a call to an (un)holy crusade: we have to banish the usage of commands in compromised machines and start to embrace coding. Please, take a seat and listen to the story of netsh and PortProxy.

Prayers at the foot of the Altar a.k.a. disclaimer

The intention of this short article is to encourage people to improve their tradecraft. We use netsh here as a mere example to transmit the core idea: we need to move from commands to tasks coded in our implants/tools.

Introduction

There are tons of ways to tunnel your traffic through a compromised machine. Probably the most common can be dropping an implant that implements a SOCKS4/5 proxy, so you can route your traffic through that computer and run your tools against other network segments previously inaccessible. But in some scenarios we can't just deploy our socks proxy listening to an arbitrary port and we need to rely on native tools, like the well-known **netsh**.

Forwarding traffic from one port to another machine is trivial with netsh. For example, if we want to connect to the RDP service exposed by a server (let's call it C) at 10.2.0.12 and we need to use B (10.1.0.233) as pivot, the command line would look like:

```
netsh interface portproxy add v4tov4 listenport=1337 listenaddress=0.0.0.0 con
```

Then we only need to use our favorite RDP client and point it to B (10.1.0.233) at port 1337. Easy peachy.

But… how netsh works and **what is happening under the hood?** Can we implement this functionality by ourselves so we can avoid the use of the well-known netsh?

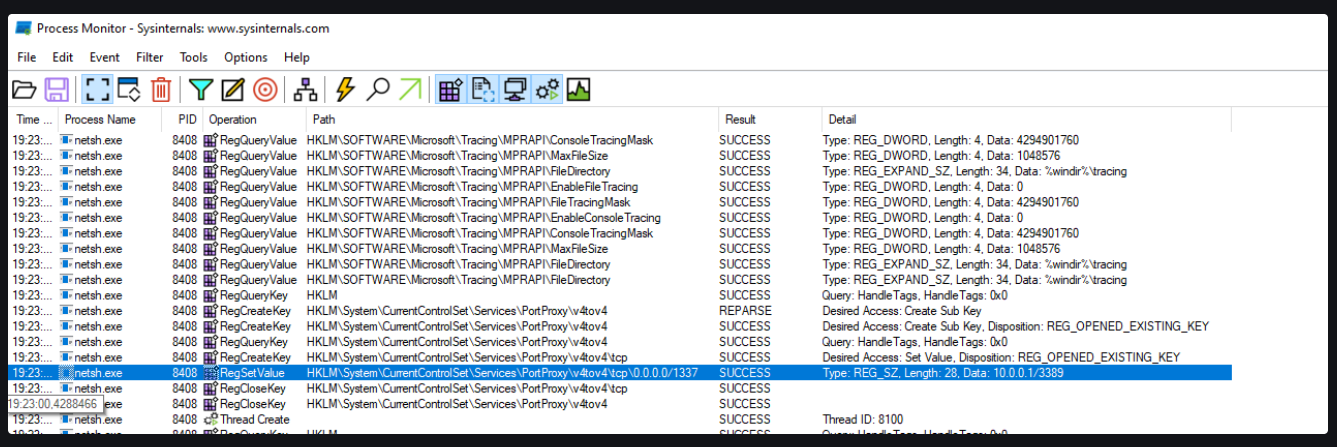
Shedding light

The first thing to do (after googling) when we have to play with something in Windows is to take a look at ReactOS and Wine projects (usually both are a goldmine) but this time we were

unlucky:

```
#include "wine/debug.h"

WINE_DEFAULT_DEBUG_CHANNEL(netsh);

int __cdecl wmain(int argc, WCHAR *argv[])
{
    int i;

    WINE_FIXME("stub:");
    for (i = 0; i < argc; i++)
        WINE_FIXME(" %s", wine_dbgstr_w(argv[i]));
    WINE_FIXME("\n");

    return 0;
}
```

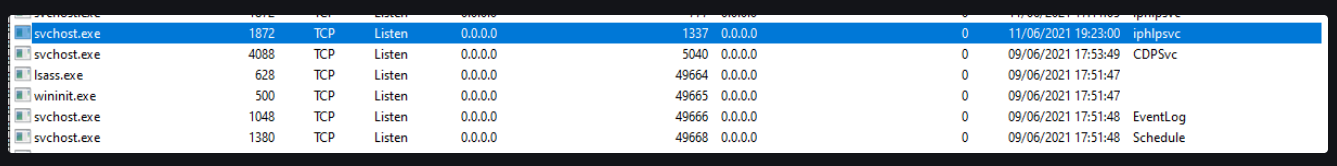
So let's try to execute netsh and take a look at it with Process Monitor:



Netsh setting a registry value.

In Process Monitor the only thing that is related to "PortProxy" is the creation of a value with the forwarding info (source an destination) inside the key `HKLM\SYSTEM\ControlSet001\Services\PortProxy\v4tov4\tcp`. If we google this key we can find a lot of articles talking about DFIR and how this key can be used to detect this particular TTP in forensic analysis (for example: Port Proxy detection - How can we see port proxy configurations in DFIR?).

If we create manually this registry value nothing happens, so we need something more to trigger the proxy creation. What are we missing? Well, that question is easy to answer. Let's see what happened with our previous netsh execution with TCPView:



Svchost and iphlpsvc reference.

As we can see iphlpsvc (IP Helper Service) is in charge to create the "portproxy". So netsh should "contact" this service in order to trigger the proxy creation, but how is this done? We should open iphlpsvc.dll inside Binary Ninja and look for references to "PortProxy". (Spoiler: it is using the `paramchange` control code, so we can trigger it with `sc` easily)

```
18007d2ae                                                   00 00                                      ..

18007d2b0  char const data_18007d2b0[0x2e] = "onecoreuap\\net\\netio\\iphlpsvc\\service\\proxy.c", 0

18007d2de                                                                                 00 00                                     ..
18007d2e0  data_18007d2e0:
18007d2e0  6f 00 6e 00 65 00 63 00-6f 00 72 00 65 00 75 00-61 00 70 00 5c 00 6e 00-65 00 74 00 5c 00 6e 00  o.n.e.c.o.r.e.u.a.p.\.n.e.t.\.n.
18007d300  65 00 74 00 69 00 6f 00-5c 00 69 00 70 00 68 00-6c 00 70 00 73 00 76 00-63 00 5c 00 73 00 65 00  e.t.i.o.\.i.p.h.l.p.s.v.c.\.s.e.
18007d320  72 00 76 00 69 00 63 00-65 00 5c 00 70 00 72 00-6f 00 78 00 79 00 2e 00-63 00 00 00 00 00 00 00  r.v.i.c.e.\.p.r.o.x.y...c.......
18007d340  data_18007d340:
18007d340  53 00 79 00 73 00 74 00-65 00 6d 00 5c 00 43 00-75 00 72 00 72 00 65 00-6e 00 74 00 43 00 6f 00  S.y.s.t.e.m.\.C.u.r.r.e.n.t.C.o.
18007d360  6e 00 74 00 72 00 6f 00-6c 00 53 00 65 00 74 00-5c 00 53 00 65 00 72 00-76 00 69 00 63 00 65 00  n.t.r.o.l.S.e.t.\.S.e.r.v.i.c.e.
18007d380  73 00 5c 00 50 00 6f 00-72 00 74 00 50 00 72 00-6f 00 78 00 79 00 00 00                          s.\.P.o.r.t.P.r.o.x.y...
18007d398  data_18007d398:
18007d398                                                              55 00 6e 00 64 00 6f 00                          U.n.d.o.
18007d3a0  4f 00 6e 00 53 00 74 00-6f 00 70 00 00 00 00 00                                                  O.n.S.t.o.p.....
18007d3b0  data_18007d3b0:
18007d3b0                                                  53 00 79 00 73 00 74 00-65 00 6d 00 5c 00 43 00                  S.y.s.t.e.m.\.C.
18007d3c0  75 00 72 00 72 00 65 00-6e 00 74 00 43 00 6f 00-6e 00 74 00 72 00 6f 00-6c 00 53 00 65 00 74 00  u.r.r.e.n.t.C.o.n.t.r.o.l.S.e.t.
18007d3e0  5c 00 53 00 65 00 72 00-76 00 69 00 63 00 65 00-73 00 5c 00 69 00 70 00-68 00 6c 00 70 00 73 00  \.S.e.r.v.i.c.e.s.\.i.p.h.l.p.s.
18007d400  76 00 63 00 5c 00 43 00-6f 00 6e 00 66 00 69 00-67 00 00 00 00 00 00 00                          v.c.\.C.o.n.f.i.g.......
18007d418  data_18007d418:
18007d418                                                              49 00 70 00 48 00 6c 00                          I.p.H.l.
18007d420  70 00 53 00 76 00 63 00-00 00 00 00 00 00 00 00                                                  p.S.v.c.........
18007d430  data_18007d430:
18007d430                                                  59 9a 3e 5d d5 e9 00 4b-a6 bd ff 34 ff 51 65 48                  Y.>]...K...4.QeH
```

Reference to a registry key with 'PortProxy' word inside

We have a hit with a registry key similar to the one that we were looking for…

```
sub_180014310:
mov     qword [rsp+0x8 {__saved_rbx}], rbx
mov     qword [rsp+0x10 {__saved_rdi}], rdi
push    rbp {__saved_rbp}
mov     rbp, rsp {__saved_rbp}
sub     rsp, 0x50
lea     rax, [rbp-0x10 {var_18}]
mov     ebx, 0x1
mov     qword [rbp-0x8 {var_10}], rax {var_18}
lea     rdx, [rel data_18007d340]  {"System\CurrentControlSet\Service…"}
lea     rax, [rbp-0x10 {var_18}]
mov     r9d, ebx  {0x1}
mov     qword [rbp-0x10 {var_18}], rax {var_18}
xor     r8d, r8d  {0x0}
lea     rax, [rbp+0x20 {arg_18}]
mov     rcx, 0xffffffff80000002
mov     qword [rsp+0x20 {var_38}], rax {arg_18}
call    qword [rel RegOpenKeyExW]
nop     dword [rax+rax], eax
test    eax, eax
jne     0x1800143b6
```

Function chunk that references the registry key found

…so we can start the old and dirty game of following the call cascade (cross-reference party!) until we reach something really interesting (Note: `OnConfigChange` is a function renamed by us):

```
lea     rdx, [rel data_1800988b0]  {"ServiceHandler: Got a SERVICE_CO…"}
mov     ecx, 0x40000
call    sub_180001420
call    OnConfigChange
nop
jmp     0x180001344
```

String with the words ServiceHandler and SERVICE_CONTROL_PARAMCHANGE

We got it! If a `paramchange` control code arrives to the `iphlpsvc`, it is going to read again the PortProxy configuration from the registry and act according to the info retrieved.

We can translate `netsh` PortProxy into the creation of a registry key and then sending a `paramchange` control code to the IP Helper service, or in other words we can execute these commands:

```
reg add HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\PortProxy\v4tov4\tcp
sc control iphlpsvc paramchange
reg delete HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\PortProxy\v4tov4 /
```

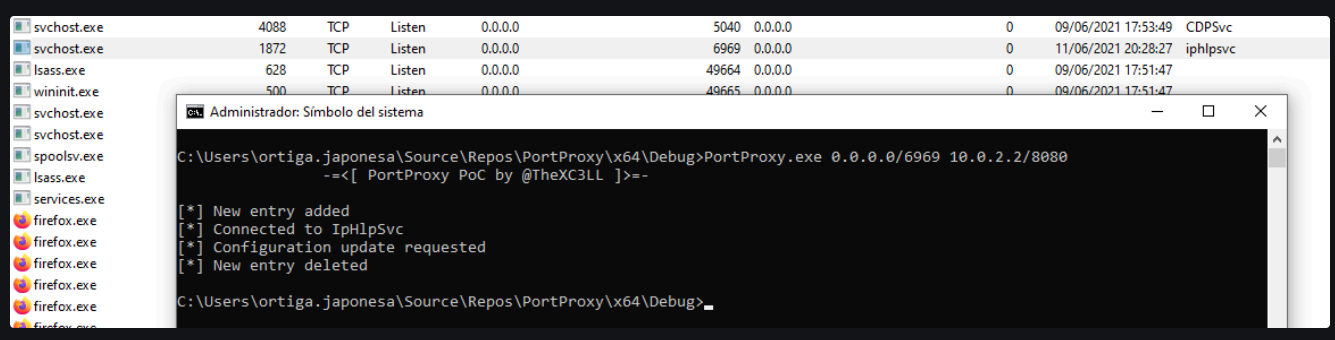
From stone to steel

It's time to translate our commands into a shitty PoC in C:

```c
// PortProxy PoC
// @TheXC3LL

#include <Windows.h>
#include <stdio.h>


DWORD iphlpsvcUpdate(void) {
	SC_HANDLE hManager;
	SC_HANDLE hService;
	SERVICE_STATUS serviceStatus;
	DWORD retStatus = 0;
	DWORD ret = -1;

	hManager = OpenSCManagerA(NULL, NULL, GENERIC_READ);
	if (hManager) {
		hService = OpenServiceA(hManager, "IpHlpSvc", SERVICE_PAUSE_CO
		if (hService) {
			printf("[*] Connected to IpHlpSvc\n");
			retStatus = ControlService(hService, SERVICE_CONTROL_P
			if (retStatus) {
				printf("[*] Configuration update requested\n")
				ret = 0;
			}
			else {
				printf("[!] ControlService() failed!\n");
			}
			CloseServiceHandle(hService);
			CloseServiceHandle(hManager);
			return ret;
		}
		CloseServiceHandle(hManager);
		printf("[!] OpenServiceA() failed!\n");
		return ret;
	}
	printf("[!] OpenSCManager() failed!\n");
	return ret;
}

DWORD addEntry(LPSTR source, LPSTR destination) {
	LPCSTR v4tov4 = "SYSTEM\\ControlSet001\\Services\\PortProxy\\v4tov4\\t
	HKEY hKey = NULL;
	LSTATUS retStatus = 0;
	DWORD ret = -1;

	retStatus = RegCreateKeyExA(HKEY_LOCAL_MACHINE, v4tov4, 0, NULL, REG_O
	if (retStatus == ERROR_SUCCESS) {
		retStatus = (RegSetValueExA(hKey, source, 0, REG_SZ, (LPBYTE)d
		if (retStatus == ERROR_SUCCESS) {
			printf("[*] New entry added\n");
			ret = 0;
		}
```

```
                else {
                        printf("[!] RegSetValueExA() failed!\n");
                }
                RegCloseKey(hKey);
                return ret;
        }
        printf("[!] RegCreateKeyExA() failed!\n");
        return ret;
}

DWORD deleteEntry(LPSTR source) {
        LPCSTR v4tov4 = "SYSTEM\\ControlSet001\\Services\\PortProxy\\v4tov4\\t
        HKEY hKey = NULL;
        LSTATUS retStatus = 0;
        DWORD ret = -1;

        retStatus = RegCreateKeyExA(HKEY_LOCAL_MACHINE, v4tov4, 0, NULL, REG_O
        if (retStatus == ERROR_SUCCESS) {
                retStatus = RegDeleteKeyValueA(HKEY_LOCAL_MACHINE, v4tov4, sou
                if (retStatus == ERROR_SUCCESS) {
                        printf("[*] New entry deleted\n");
                        ret = 0;
                }
                else {
                        printf("[!] RegDeleteKeyValueA() failed!\n");
                }
                RegCloseKey(hKey);
                return ret;
        }
        printf("[!] RegCreateKeyExA() failed!\n");
        return ret;
}

int main(int argc, char** argv) {
        printf("\t\t-=<[ PortProxy PoC by @TheXC3LL ]>=-\n\n");
        if (argc <= 2) {
                printf("[!] Invalid syntax! Usage: PortProxy.exe SOURCE_IP/POR
        }
        if (addEntry(argv[1], argv[2]) != -1) {
                if (iphlpsvcUpdate() == -1) {
                        printf("[!] Something went wrong :S\n");
                }
                if (deleteEntry(argv[1]) == -1) {
                                printf("[!] Troubles deleting the entry, pleas
                }
        }
        return 0;
}
```

Fire in the hole!



Proof of Concept working like a charm

EDIT (2021/06/19): A reader pointed us that “Control001” is the “normal” controlset, but in some scenarios the number can change (002, 003, etc.) so instead of using it directly we should use HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet before.

EoF

As we stated at the beginning this short article is not about “netsh” or the “PortProxy” functionality. We aim higher: we want to encourage you to stop using commands blindly and to start to dig inside what is doing your machine. Explore and learn the internals of everything you do on an red team operation or a pentest.

We hope you enjoyed this reading! Feel free to give us feedback at our twitter @AdeptsOf0xCC.

updated_at 11-06-2021

Previous
From theory to practice: analysis and PoC development for CVE-2020-28018 (Use-After-Free in Exim)

Next
Knock! Knock! The postman is here! (abusing Mailslots and PortKnocking for connectionless shells)

rss © 2024

klisé theme on jekyll