The return of the spoof part 2: Command line spoofing

By NVISO

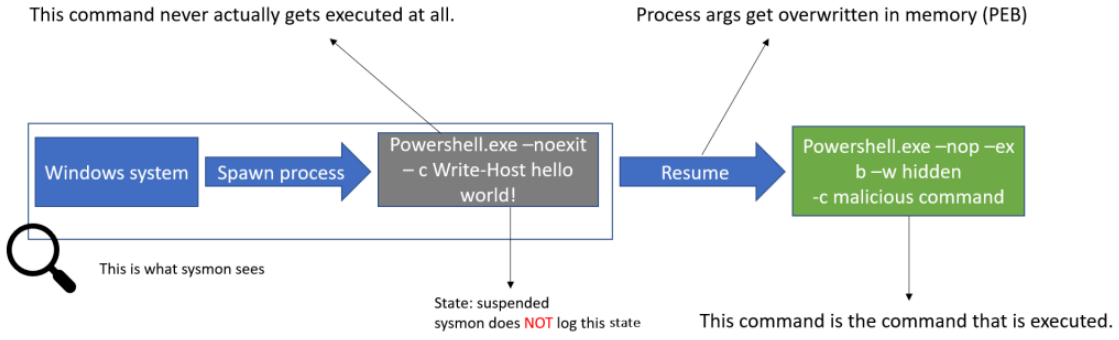
Published: 2020-02-04 · Archived: 2026-04-05 14:13:01 UTC

A few days ago I wrote a blog post about the evolving landscape of threat detection and how attackers need to adapt their techniques. In the previous post, I talked about one of the deception techniques that attackers are now using, called parent process ID spoofing. In this blog post, I'll talk about another deception technique that goes hand in hand with the parent PID spoofing technique, which is called "command line spoofing".

What is command line spoofing and why is it deceptive?

Command line spoofing is a technique that spawns a process with fake arguments and overrides those arguments at execution time. The way this works is an attacker will use a technique called process hollowing. Process hollowing spawns a process in a suspended state, which is done using legitimate Windows API calls.

Tell me more about these Windows API calls!
Specifically, the CreateProcessA function is used for this. One of the parameters in this function is the dwCreationFlags argument, which has a value called "CREATE_SUSPENDED". If a process is spawned with this flag, it gets launched in a suspended state, and will not run until the ResumeThread fuction is called. This provides the unique opportunity for any malicious actor to spawn a completely harmless process, without it actually executing. This in itself is enough to fool any endpoint protection that checks process creation for malicious behavior. Sysmon for example will be completely fooled with this technique.



A high level overview of command line argument spoofing

So, how does an attacker abuse this suspended process?
Once the process is spawned in a suspended state, an attacker will look for the process in memory. The way this usually happens is by querying the PEB (Process Environment Block). The PEB is a data structure that holds all the information of the current process (every process has a PEB). Once the correct PEB is found in the Windows memory, the arguments get updated to the malicious arguments and the process is resumed, resulting in executing the malicious code.

That sounds serious once again! How can we detect it?
Good news! If an attacker only did the above steps, tools like process hacker and process explorer will catch this technique. This is because process hacker and process explorer actually retrieve a copy of the PEB each time the process is inspected, meaning that our spoofed arguments are revealed.

Adam Chester Found a way to bypass process explorer and process hacker monitoring mechanisms. I'll explain it here briefly:

PEB stores command line arguments in a UNICODE_STRING structure.

```
typedef struct _RTL_USER_PROCESS_PARAMETERS {
    BYTE           Reserved1[16];
    PVOID          Reserved2[10];
    UNICODE_STRING ImagePathName;
    UNICODE_STRING CommandLine;
} RTL_USER_PROCESS_PARAMETERS, *PRTL_USER_PROCESS_PARAMETERS;
```

The UNICODE_STRING structure looks like this:

```
typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
```

When an attacker sets a length that is less than the buffer, process hacker and process explorer will terminate the "commandline" argument at the length of the unicode_string. The actual process however, will use the buffer

parameter and will thus execute the full buffer. This allows attackers to spawn processes that will fool even process hacker and process explorer!

So how do we detect this process explorer/ process hacker bypassing behavior?
At the time of writing, I have not found a foolproof way of detecting this behavior. Sysmon (and most likely other EDR solutions) still logs the fact that the process is making a network connection and, shortly after, spawning another process (most of the time a reverse shell, or a malicious executable). This could also be considered as a suspicious behavior to raise alerts on.
Note: sysmonx claims to have detection capabilities for this technique. However when I tried to compile sysmonx, a lot of build errors appeared. I can only assume sysmonx is still a work in progress, but it might be worth keeping an eye on.

Conclusion

This concludes my two blog posts about deception techniques used in the wild. As you can see when these two concepts are paired with each other, they can make the defender's life a lot more difficult. It's pretty hard to detect these deception techniques at scale because of the enormous amount of logging that would be required. The detection techniques are also prone to false positives so it's hard to distinguish the malicious activities from the false positives.
I have written four example scenarios, available over at GitHub, should you want to try them out for yourself. It is worth mentioning that cmd and powershell are just example scenarios, any valid command can get executed using this method.

Should you have knowledge of a robust and mature tool that can detect this technique from a blue perspective, feel free to reach out by commenting on this blog post!

About the author

Jean-François Maes is a red teaming and social engineering expert working in the NVISO Cyber Resilience team. When he is not working, you can probably find Jean-François in the Gym or conducting research. You can find Jean-François on LinkedIn.

Published February 4, 2020December 8, 2021

Post navigation