Memory Injection like a Boss

Posted on 5 January 2018 by Arran Purewal

Our first meetup – Cybersecurity with Countercept – was held on December 6, 2017 and was a roaring success. Aside from mince pies, beer, and pizza, the evening featured three presentations from the Countercept team. Countercept Threat Hunter Arran Purewal presented 'Memory Injection Like a Boss'.

"Why do attackers use in-memory attacks?"

Before I go into that, let's first talk about what an in-memory attack is.

An in-memory attack does not rely on a file written to disk. It lives in a computer's RAM, which we call 'volatile memory'. This means the malicious content is removed once the computer is rebooted.

The reason why attackers are trying to avoid files being written to disk is that most widely-used security software programs (such as anti-virus) concentrate their efforts on inspecting artefacts written to disk. As a result, these tools are proficient at detecting malicious files and preventing them from infecting a computer. By contrast, in-memory attacks are more sophisticated and bypass anti-virus software and forensics. For an attacker wanting to remain undetected, it's currently the best way to evade defences.

There's a lot of jargon around in-memory attacks, so let's break some of that down:

- Shellcodes are a small stub of code used as a payload
- DLL is a shared library of functions that multiple programs can access
- Process is an instance of a program being executed

- Process is an instance of a program being executed
- Thread is a small sequence of instructions or a component of a process
- Windows API protocols allow interaction with the Windows OS
- VirtualAllocEx reserves or changes a region of memory
- WriteProcessMemory writes data to an area of memory in a specified process
- CreateRemoteThread creates a thread in the address space of another process

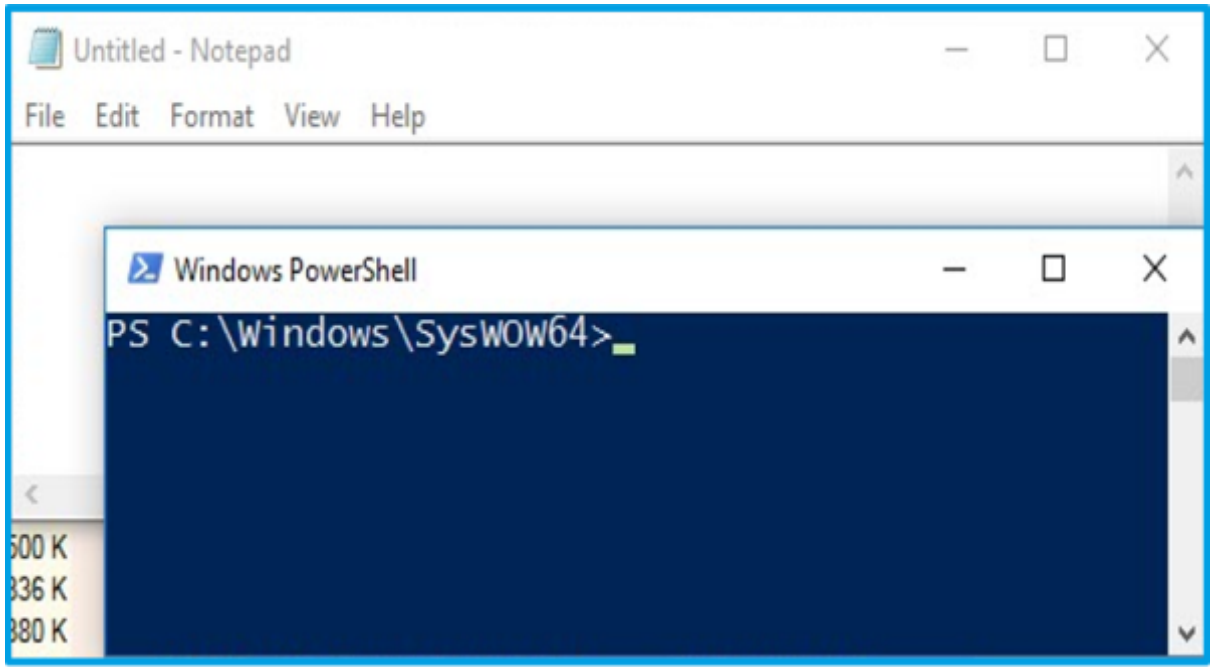
Memory Injection Techniques

To carry out a simple in-memory attack one only needs to use the three functions shown above, which will be looked at alongside a few other types of exploit:

- Shellcode Injection
- Reflective DLL Injection
- Process Hollowing
- AtomBombing
- Inline Hooking

Shellcode Injection

Shellcode is a small piece of code that – when used as a payload – injects malicious code into a running application. In this case, it is used to launch PowerShell, which is regularly used in attempts to execute in-memory attacks.



Although the PowerShell command here is benign, it could also have been appended with malicious arguments to try and launch an exploit on the target system.

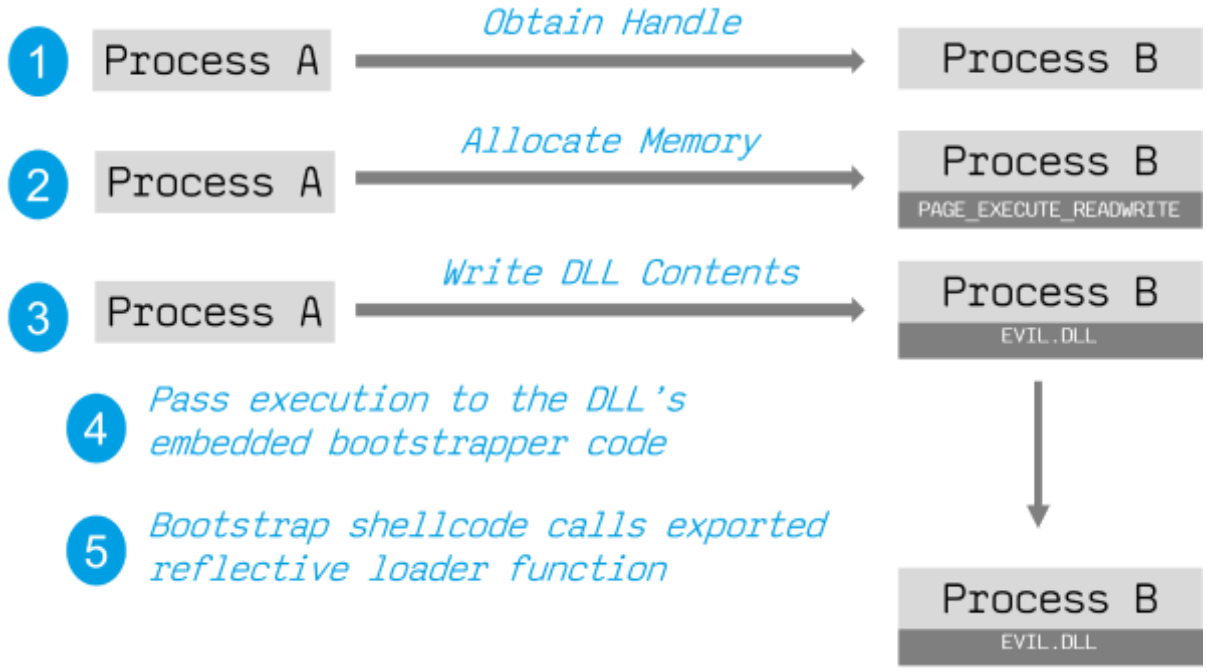
Reflective DLL Injection

Normally, loading a DLL in Windows calls the function LoadLibrary. It takes the path of the file and executes its functions without requiring too much from the user. It requires the DLL to be on disk and will enumerate the DLL with the process.

However, there is a stealthier method called reflective DLL injection, in which the contents of a DLL can be loaded in memory. This requires the usage of a custom loader, as LoadLibrary cannot be used. So, when the contents of the DLL is loaded into memory, the execution will pass to the embedded code (bootstrapper code), which will emulate the tasks

carried out by LoadLibrary (such as mapping the variable memory) and execute the reflectively loaded functions, as seen

carried out by LoadLibrary (such as mapping the variable memory) and execute the reflectively loaded functions, as seen below.



DoublePulsar – which was used alongside EternalBlue in the WannaCry attacks – performed a reflective DLL injection, but did not make use of the standard LoadLibrary call. You can find out more about it in our in-depth analysis of its use.

Process hollowing

This technique starts a legitimate process whose sole purpose is to be a container for malicious code. It delivers the process in a "suspended" state, then rewrites the content with the required code in memory, and continues to execution.

Dridex – the infamous banking malware – and many of its variants use process hollowing to gain initial footholds on

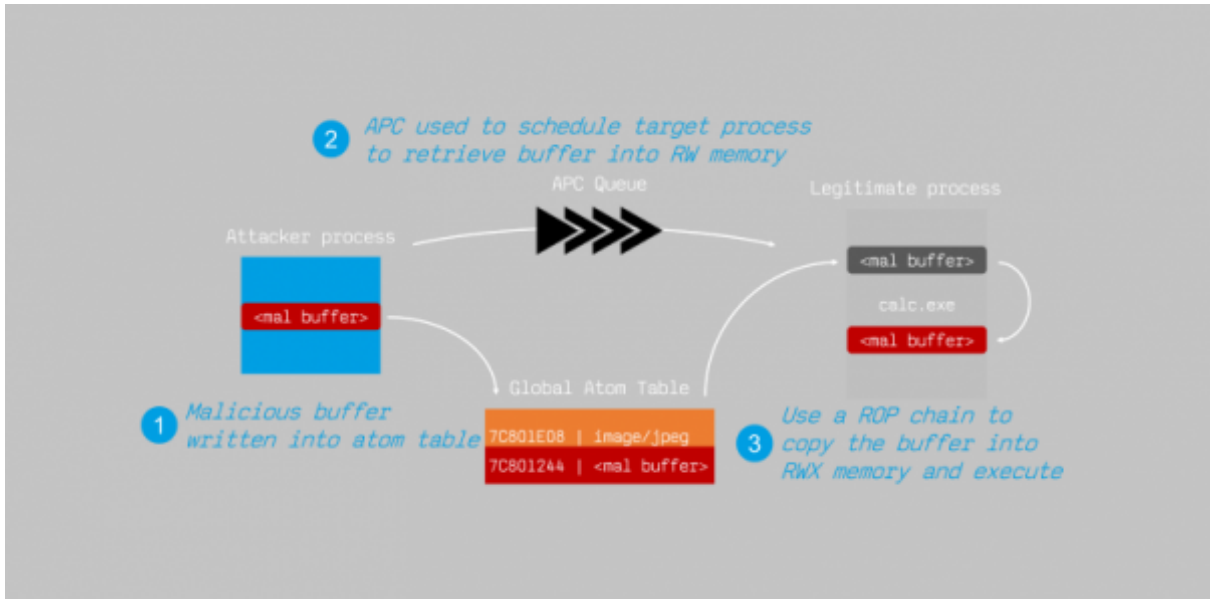
Dridex – the infamous banking malware – and many of its variants use process hollowing to gain initial footholds on machines. Using Dridex as an example for process hollowing, this type of malware typically follows this sequence:

- A phishing email with an embedded macro is opened (demonstrating yet again that people are inadvertently the weakest link in an organization);
- Malicious content is downloaded from the URL;
- Process hollowing extracts the unpacked version of the exploit into memory, which will then run;
- The malware now has a foothold and will snoop on users, waiting for credentials to be entered. It can also upload files, execute files, and inject itself into browser processes to monitor information.

AtomBombing

Attackers, as we know, never sit on their laurels. They regularly change techniques to evade detection. As process hollowing has become more associated with Dridex, its authors have looked for new ways to execute its malware - developing a technique called AtomBombing.

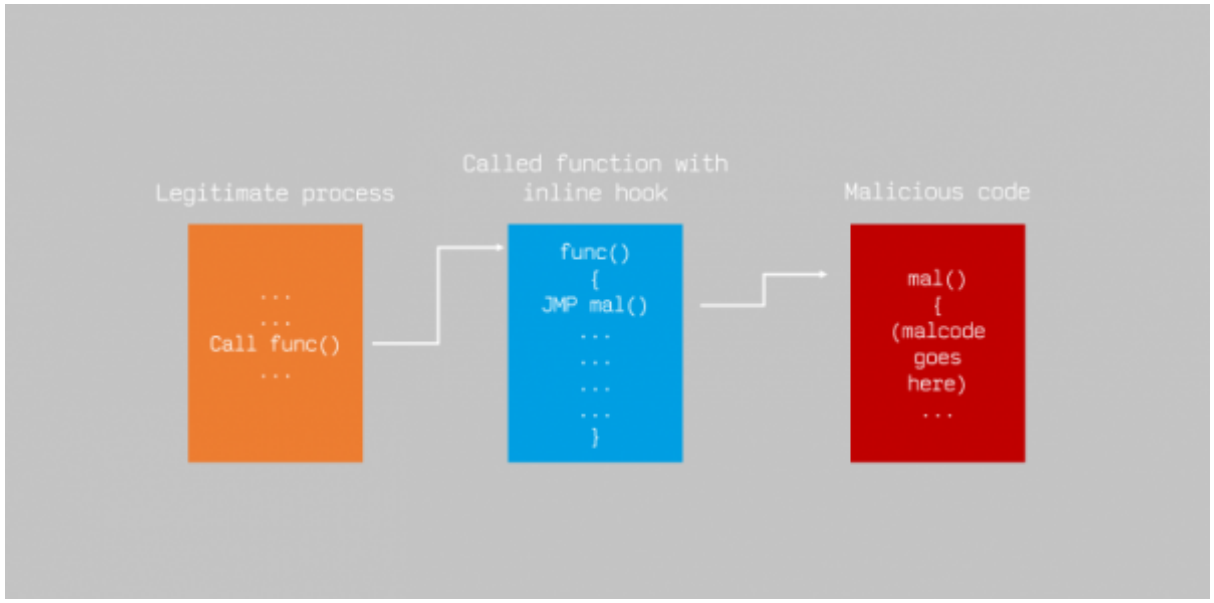
AtomBombing is an exploit where attackers write malicious code into Windows' atom tables, then force a legitimate program to retrieve the code from the table. For Dridex, this meant: a malicious buffer was written into an atom table; asynchronous procedure call (APC) was used to schedule the target process to retrieve the buffer and place it into read-write memory; a return-oriented programming (ROP) chain copied the buffer into RWX memory, where it then executed.



Inline hooking

Inline hooking is just what it sounds like – modifying memory 'inline' to 'hook' functions and redirect execution. It often involves modifying the first few instructions to move execution flow to the malicious code, which will then re-route to the legitimate call.

The Zeus malware – which has had a prolific career due to its stealth techniques – infects via phishing and then hooks functions related to http communications and uses man-in-the-browser keystroke logging to steal passwords and get access to accounts.



Memory forensics at scale

Memory analysis is a crucial component of any attack detection solution, as the signature-based nature of traditional detection would not detect the techniques I've described. In-memory hunting allows us to look in more detail at processes and gauge whether they've been tampered with.

Looking at isolated processes is very different to looking at scale, and there is still work to be done in this field. We need to develop indicators that associates and identifies some of the behavior we see during in-memory attacks. However, the results will still require the discretion and experience of a skilled analyst to understand what has happened, and decide what to do next.

BACK TO BLOG