Windows Internals: Advanced Exploit Mitigation

Fabian Nguyen

Abstract

We take a look at general techniques used by attackers to compromise Windows systems and some fundamental defense mechanisms against them. This paper will provide an overview of Microsoft's latest additions to the security concept of the Windows Operating System [OS] , analyze inherent flaws in their design and take a brief look at already existing attacks.

1 Introduction

In an increasingly digitalized world an overwhelming amount of private and/or safety-critical information and data is stored on computers. Windows is by far the most used operating system and therefore the main target of attackers to compromise data or computer systems. One common intent of attackers is to steal an individual's passcode for a website, e.g an online-banking website. Naturally, as the amount and complexity of attacks rises, OS vendors are forced to put an increasingly high amount of effort into mitigating existing weaknesses and deny attackers of further possibilities to compromise their OS. Even though this is the case, the amount of potentially abusable vulnerabilities in Windows has been increasing, instead of decreasing. **?**

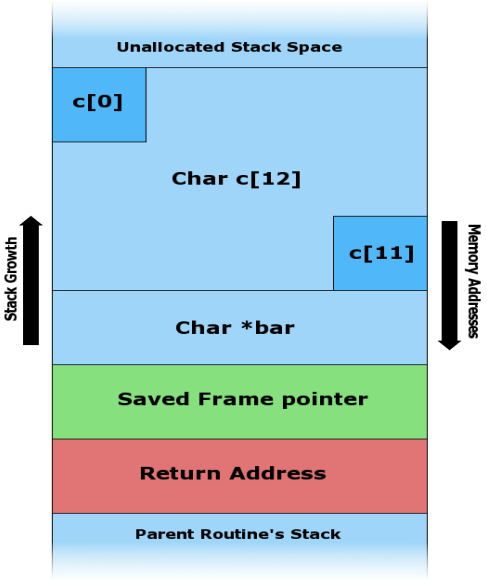
Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	XSS	Bypass something	Gain Information	Gain Privileges
2015	57	4	19	6	6		10	5	26
2016	172	6	47	23	7		19	31	82
2017	268	32	50	16	2	1	18	108	19
2018	257	21	45	19	1	1	39	72	1
2019	357	28	124	101	6	1	10	73	2
Total	1111	91	285	165	22	3	96	289	130
% Of All		8.2	25.7	14.9	2.0	0.3	8.6	26.0	11.7

Figure 1: Amount of documented vulnerabilities in the Windows Operating System.
Note that the spike in "Gain Information" vulnerabilities in 2017 is inflated by a family of attacks widely known as Meltdown/Spectre

We can see that the three largest groups of vulnerabilities consist of "Gain Information", "Code Execution" and "Overflow". Of course these three categories aren't entirely seperated from each other. An attacker that is able to execute Code on a machine often does so in order to gain information and overflows are often the reason why an attacker can execute code in the first place.

2 Overflows

An overflow happens when a program writes data to memory beyond the limits of the intended data structure. One common example of this is a stack buffer overflow caused by an incorrect use of the function *strcpy*.More precisely, an overflow can occur when the given input is longer than the buffer one writes too. Relatively small in size overflows like this in particular are not always easy to spot and often remain unidentified if they don't cause immediate errors. Besides causing faulty program behaviour, this also provides a critical attack surface. To see why this is the case, let's take a look at a typical stack buffer layout with only one buffer present right at the beginning.



As we can see, the stack will usually contain a return adress right at the bottom, a frame pointer on top of it and then local data that is used by the current function. Obviously, one will also need to keep the parent function's data saved below (stack grows upwards). In our example there is only one character buffer of size 12 and a pointer to it present.

July 17, 2020, 14:19
Revision: unknown

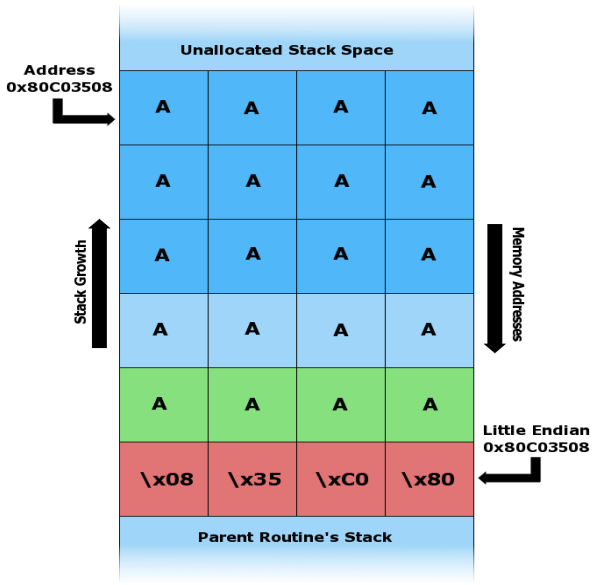


Figure 2: The user-input is too long for the given buffer

Suppose one wants to copy user-input string into this buffer, e.g by using strcpy. The user may now unknowingly or perhaps purposely overflow this buffer by entering a string thats longer than 12 characters. As we observed earlier, this will result in the memory after the buffer being written. Note that data is written from top to bottom. We see that our pointer to the buffer is overwritten, as well as the stack frame we had saved on the stack earlier. Most importantly though, the return adress is also corrupted. In the given figure, the input is constructed so that the part that's written over the return adress resembles an adress itself. In this case, its just the adress of the buffer again. At some point, our given function will attempt to return to this adress. However, once it does so, all it will be able to read from this adress is a swarm of 'A's which certainly doesnt resemble a valid sequence of code. The programm will fault and terminate.

3 Related Work

4 Approach

5 Conclusion

References