Yasanka Sameera Horawalavithana

Heap Spray

This type of exploits caused due to the usual programming error which allocates sufficiently large data to an array without checking for index boundaries. This leads to crash the user program, overwrite the instruction pointers and enable the program to execute any arbitrary instruction at an arbitrary location. An adversary could hijack this flow of instructions to do whatever he wants, usually exploiting to an interactive shell with root privileges.

Think like an attacker

In this exploit, the ultimate goal is to get into a remote shell. What is the approach of an adversary? Here the adversary is located in a remote server, and the victim would be the client who access a web-page hosted in this remote server. The adversary mounts a malicious code in the web-page that is triggered when the victim clicks a button in the web page.

Seriously How? We use a vulnerability exposed in IE browser plugin called jnlp. Adversary construct a “sufficiently large” buffer that overflow the docbase parameter in the plugin code, and redirect the flow of instruction to a shell code which passes the control back to the remote server (local shell access in this case).

Memory Layout? We have to place our shell code in a known memory location to redirect, which in fact need to be deterministic. In this version, we use “heap spray” approach that layout a pattern of memory chunks (with shell code inside) in the heap. We need to find a good size of chunks to mitigate fragmentation, and good number of chunks to overlay the pattern to higher memory addresses.

We spread the heap with 1000 memory chunks of size 0.25 mb, where we end up having 250 mb in the heap allocated to our malicious pattern. We use “slice()” method to fool the javascript engine to blow up new objects in the heap without having references stored as indices. Once we overwrite EIP value with the overflow (e.g 0x08080808 as the reliable address to place upon NOP), the EIP slides through NOP until it hits an executable shell code. We generate a remote shell code via msfconsole (attached shell code is for 127.0.0.1 and port **4444**).