**Security Implementation**

**Security Implementations**

Security implementations that have developed during past several years to prevent, or impede the exploitation of vulnerabilities such as Buffer Overflow.

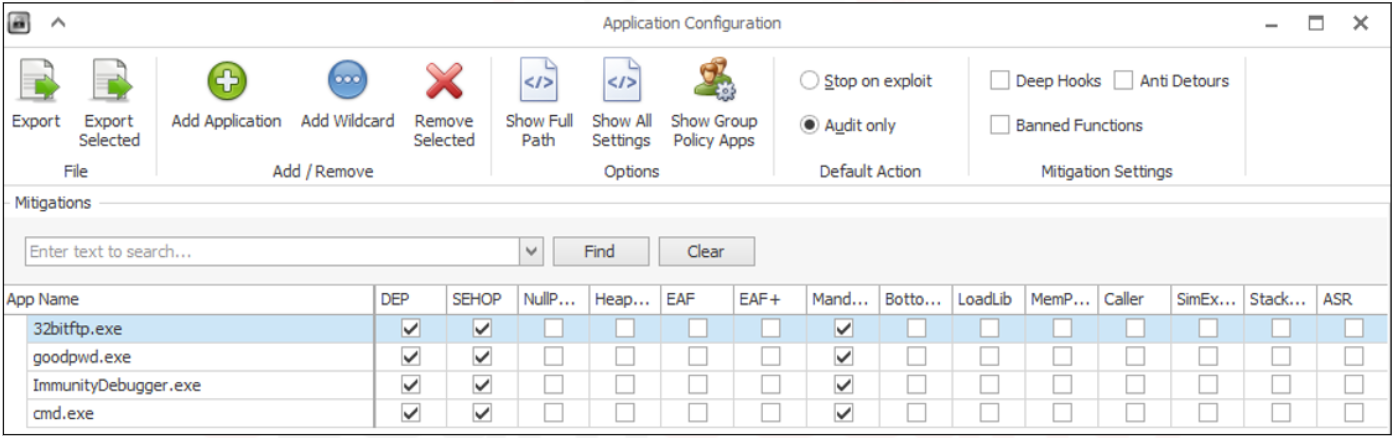
* Address Space Layout Randomization (**ASLR**).
* Data Execution Prevention (**DEP**).
* Stack Cookies (**Canary**).
* SafeSEH

In order to explain how these security features van be bypassed would require a very good understanding and experience with OS architectures, assembly code, reverse engineering and more.

**Note:** Tool that will be extremely useful is **EMET** (Enhanced mitigation experience toolkit). EMET is utility that helps prevent vulnerabilities in software from being successfully exploited. EMET offers many different mitigation technologies, such as **DEP, ALSR, SEHOP and more.** Download [EMET](https://www.microsoft.com/en-us/download/details.aspx?id=50766) and [EMET ‘s Guide](https://www.microsoft.com/en-us/download/details.aspx?id=50802).

Why do we need EMET?

Although I t can be sued to enhance the security of our system, it can also be used to disable them.



ASLR:

ASLR is to introduce randomness for executables, libraries and stacks in the memory address space; this makes it more difficult for an attacker to predict memory addresses and causes exploits to fail and crash the process.

When ASLR is activated, the OS loads the same executable at different locations in memory every time.

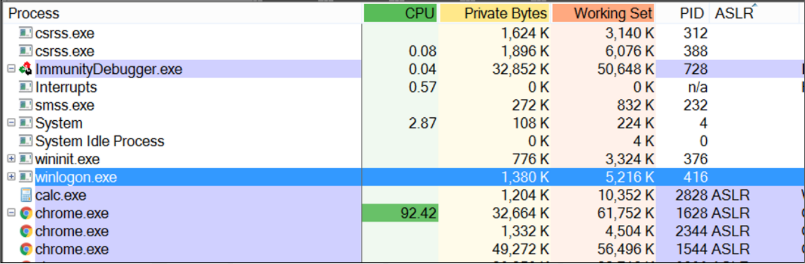
**Note:** even if a process a process has **ASLR** enabled, there could be a DLL in the address space without this protection which could make the process vulnerable to the **ASLR** bypass attack.

**Note****:** [Process explorer tool](http://technet.microsoft.com/en-us/sysinternals/bb896653) to verify the status of ASLR on different programs.

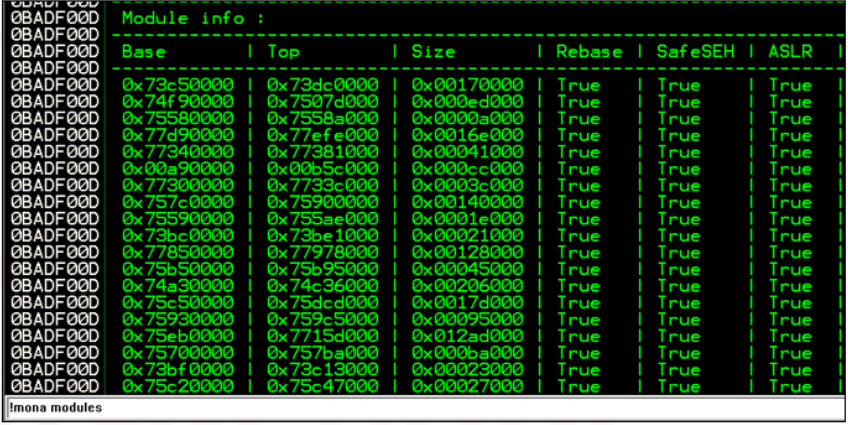
**Note:** when ASLR activated, the OS loads the same executable at different locations in memory every time (at every reboot).

Check ASLR activation:

* [Process explorer tool](http://technet.microsoft.com/en-us/sysinternals/bb896653) to verify the status of ASLR on different programs.



* Mona script
  + Command: !mona modules [Enter]. **OR** !mona noaslr [Enter].

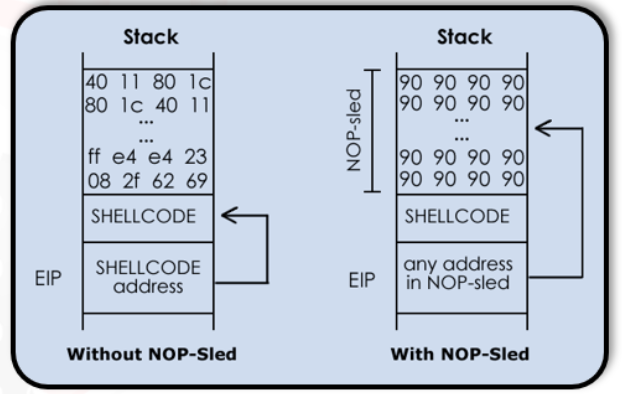


ASLR bypassing techniques:

* Non-randomized modules:
  + This technique aims to find a module that does not have ASLR enabled and then use a simple **JMP/CALL ESP** from that module.
* Brute force:
  + With this mothod, ASLR can be forced by overwriting the return pointer with plausible addresses until, at some point we reach the shellcode.
  + The success of pure brute-force depends on how tolerant an exploit is to variations in the address space layout. (e.g., how many NOPs can be placed in the buffer), and how many exploitations attempted on can perform.

**Note:** This Brute-force applied against those services that configured to be automatically restarted after a crash.

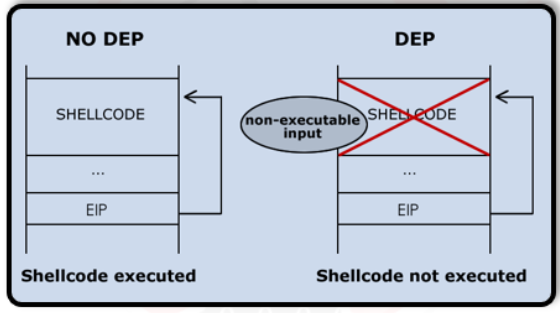
* NOPs-Sled:
  + We create a big area of NOPs in order to increase the chances to jump to this area, as you already know, NOP stand for No Operation, and as the name suggests it is instruction that effectively does nothing at all.



**Note:** we achieve max defense when ASLR is correctly implemented and DEP is enabled.

DEP:

Data Execution Prevention is a defensive hardware and software measure that prevents the execution of code from pages in memory that are not explicitly marked as executable. The code injected into the memory cannot be run from that region; harder.



While DEP makes the exploit development process more complex and time-consuming, it is possible to disable it before executing the actual shellcode.

DEP bypassing techniques:

Bypassing DEP is possible by using a very smart technique called [Return-oriented programming](https://hovav.net/ucsd/talks/blackhat08.html) (ROP), ROP consists of finding multiple machine instructions in the program (called **gadget**), in order to create a chain of instructions that do something.

Gadgets are small groups of instructions that perform some operations (arithmetical operations on registers, check for conditional jump, store or load data and so on) and that end with a RET instructions.

**Note:** The RET is important since it will allow the chain to work and keep jumping to the next address after executing the small set of instructions.

The purposes of the entire chain are different. We can use ROP gadget to call a memory protection function (kernel API such as **VirtualProtect**) that can be used to mark the stack as executable; this will allow us to run our shellcode.

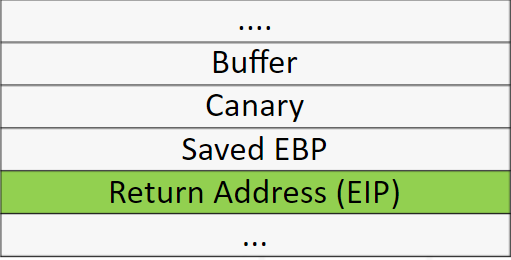
**Note:** we can also use ROP gadgets to execute direct commands or copy data into executable regions and then jump to it.

ROP resources:

* List of ROP gadgets from different libraries and .dll files: [link](https://www.corelan.be/index.php/security/rop-gadgets/)
* Articles that goes deeper in ROP gadgets: [link](https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/#buildingblocks)

Canary:

Canary or stack cookie, is a security implementation that places a value next to the return address on the stack. Prologue loads a value into this location, while the epilogue makes sure that the value is intact. As a result, when the epilogue runs, it checks that the value is still there and that it is correct.



As we already know, most buffer overflows overwrite memory address locations in the stack right before the return pointer; this means that the canary value will be overwritten too.

When the prologue function returns, the value is checked to make sure that it was not changed. If so, it means that a stack buffer flow occurred.

The function prologue loads the random value in the canary location and the epilogue makes sure that the value is not corrupted/modified.

Canary bypassing techniques:

* Guess the canary value and add it to the payload.
* Try to retrieve canary value and dd it to the payload.
* [David Litchfield](https://www.blackhat.com/presentations/bh-asia-03/bh-asia-03-litchfield.pdf) developed a method that does not require any of these. If the canary does not match, the exception handler will be triggered. If the attacker can overwrite the exception handler structure (**SEH**) and trigger an exception before the canary value is checked, the buffer overflow could still be executed, the previous technique called **SEHOP** (Preventing the Exploitation of Structured Exception Handler).

SafeSEH:

The **SEHOP** introduced a new security measures called SafeSEH.

SafeSEH Resources:

* Read more about SafeSEH: [link](https://docs.microsoft.com/en-us/cpp/build/reference/safeseh-image-has-safe-exception-handlers?redirectedfrom=MSDN&view=vs-2019)
* Article on how on bypass stack canary: [link](https://www.corelan.be/index.php/2009/09/21/exploit-writing-tutorial-part-6-bypassing-stack-cookies-safeseh-hw-dep-and-aslr/)