RP Stack Protection

At RP enterprises we are so smart about how we protect our memory from buffer overflow attacks we pride ourselves in our elite-ness. We introduce what's called a **canary.**

Canaries are random 3-byte values that are inserted onto the stack at random locations and managed invisible to the program. The program runs as usual but once the frame has been allocated the operating system will insert the canaries at random locations in the stack frame. The OS will keep track of the values inserted, and if upon return of a function, it detects that any canary values have been changed, it'll prevent the continued execution of the program.

Before

High Memory Address		**Description**
|	Function Parameters	
| 500	Return Address	Change to 100
|	Previous Frame Pointer	
| 100	Local Variables	Overflow this way
V	…	Inject code here
Low Memory Address		

After

High Memory Address		Description
|	Function Parameters	
|	Return Address	
|	Previous Frame Pointer	
|	Canary1	1000
V	Local Variables	
	…	
	Canary2	1009
	Local Variables	
Low Memory Address		

Basic canary values are **seeded.** This means we start with a **random** canary value at the first insertion of canary on the stack, and the next insertion of a canary value will add 9 to it, and this value becomes the new seed. Canaries are only inserted before local variables. This is the first step to preventing buffer overflow attacks. At RP Enterprise, we are so confident that this can prevent 99% of buffer overflow attacks. Yes we are that good.