Han Wang (wang2786, 0028451697)

```
1.
1.main process before appl1():
Base of the stack: Address - 0x0EFD8FFC, Value - 0x0A0AAAA9
Top of the stack: Address - 0x0EFD8FC0, Value - 0x00000000

2. after appl1()

is created before fun1() is called:
Base of the stack: Address - 0x0EFC8FFC, Value - 0x0A0AAAA9
Top of the stack: Address - 0x0EFC8FCC, Value - 0x00000000

3. after appl1() calls fun1() and before fun1() returns:
Base of the stack: Address - 0x0EFC8FFC, Value - 0x0A0AAAA9
Top of the stack: Address - 0x0EFC8F9C, Value - 0x00114273

fun1: (a)13 + (b)21 = 34

4. after appl1() calls fun1() and after fun1() has returned:
Base of the stack: Address - 0x0EFC8FFC, Value - 0x0A0AAAA9
Top of the stack: Address - 0x0EFC8FCC, Value - 0x00000000

appl1: func result is 34
```

After color code all the same value, we can clearly see the context switch between appl1() created to call fun1() to fun1() return. The base and top of the stack changed because the context switch from main to new process. And during the fun1() call, the base of the stack never changed, but the top of the stack got smaller for storing new info from the function. And the memory was freed after fun1() was returned. Also note that the top of the stack's value also changed in step 3, indicating it is storing the appl1() function, so that once fun1() finish, it can find its way back to the caller.

5.
Attack strategy: Overflow the stack by recursively calling a custom function with local variable, which will eventually get access to victim's stack and causing OS to panic.

Impact: The OS will trap the harmful process from keep functioning, and dump the register memory to stdout if you have a kprintf() in the function(which I removed before submitting).