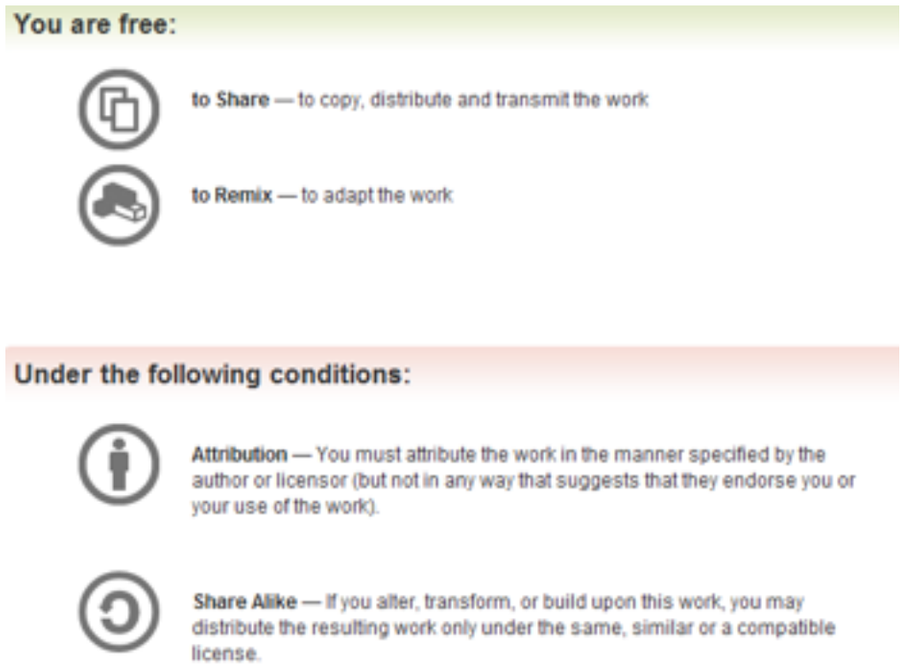
Introduction to Intel x86-64 Assembly, Architecture, Applications, & Alliteration

Xeno Kovah – 2014-2015
xeno@legbacore.com

All materials is licensed under a Creative Commons “Share Alike” license.

- http://creativecommons.org/licenses/by-sa/3.0/



Attribution condition: You must indicate that derivative work
"Is derived from Xeno Kovah's 'Intro x86-64' class, available at http://OpenSecurityTraining.info/IntroX86-64.html"

Attribution condition: You must indicate that derivative work

"Is derived from Xeno Kovah's ‘Intro x86-64’ class, available at http://OpenSecurityTraining.info/IntroX86-64.html"

LetErRIP.c

RIP-relative Addressing

```
//LetErRIP.c
char someGlobal = 0;

short main(){
    char a = 1;
    someGlobal = a;
    return 22;
}
```

```
main:
0000000140001000 48 83 EC 18          sub         rsp,18h
0000000140001004 C6 04 24 01          mov         byte ptr [rsp],1
0000000140001008 0F B6 04 24          movzx       eax,byte ptr [rsp]
000000014000100C 88 05 EE 44 06 00    mov         byte ptr [40065500h],al
0000000140001012 B8 16 00 00 00       mov         eax,16h
0000000140001017 48 83 C4 18          add         rsp,18h
000000014000101B C3                   ret
```

```
main:
0000000140001000 48 83 EC 18          sub         rsp,18h
0000000140001004 C6 04 24 01          mov         byte ptr [rsp],1
0000000140001008 0F B6 04 24          movzx       eax,byte ptr [rsp]
000000014000100C 88 05 EE 44 06 00    mov         byte ptr [someGlobal (0140065500h)],al
0000000140001012 B8 16 00 00 00       mov         eax,16h
0000000140001017 48 83 C4 18          add         rsp,18h
000000014000101B C3                   ret
```

LetErRIP.c

RIP-relative Addressing

```
//LetErRIP.c
char someGlobal = 0;

short main(){
    char a = 1;
    someGlobal = a;
    return 22;
}
```

Takeaways:
Visual Studio 2012 displays RIP-relative addresses ***misleadingly***!
64 bit bug I think. *View with symbols to see the accurate address*
(Some students said this was fixed in VS 2015)

```
main:
0000000140001000 48 83 EC 18          sub         rsp,18h
0000000140001004 C6 04 24 01          mov         byte ptr [rsp],1
0000000140001008 0F B6 04 24          movzx       eax,byte ptr [rsp]
000000014000100C 88 05 EE 44 06 00    mov         byte ptr [40065500h],al
0000000140001012 B8 16 00 00 00       mov         eax,16h
0000000140001017 48 83 C4 18          add         rsp,18h
000000014000101B C3                   ret
```

```
main:
0000000140001000 48 83 EC 18          sub         rsp,18h
0000000140001004 C6 04 24 01          mov         byte ptr [rsp],1
0000000140001008 0F B6 04 24          movzx       eax,byte ptr [rsp]
000000014000100C 88 05 EE 44 06 00    mov         byte ptr [someGlobal (0140065500h)],al
0000000140001012 B8 16 00 00 00       mov         eax,16h
0000000140001017 48 83 C4 18          add         rsp,18h
000000014000101B C3                   ret
```

LetErRIP.c

RIP-relative Addressing

```
//LetErRIP.c
char someGlobal = 0;

short main(){
    char a = 1;
    someGlobal = a;
    return 22;
}
```

On Linux:

```
00000000004004ed <main>:
  4004ed:    55                      push   %rbp
  4004ee:    48 89 e5                mov    %rsp,%rbp
  4004f1:    c6 45 ff 01             movb   $0x1,-0x1(%rbp)
  4004f5:    0f b6 45 ff             movzbl -0x1(%rbp),%eax
  4004f9:    88 05 3a 0b 20 00       mov    %al,0x200b3a(%rip)  # 601039 <someGlobal>
  4004ff:    b8 16 00 00 00          mov    $0x16,%eax
  400504:    5d                      pop    %rbp
  400505:    c3                      retq
```

More clearly RIP-relative

Helpful math of next instruction (0x4004FF)
+ displacement (0x200B3A) = 0x601039