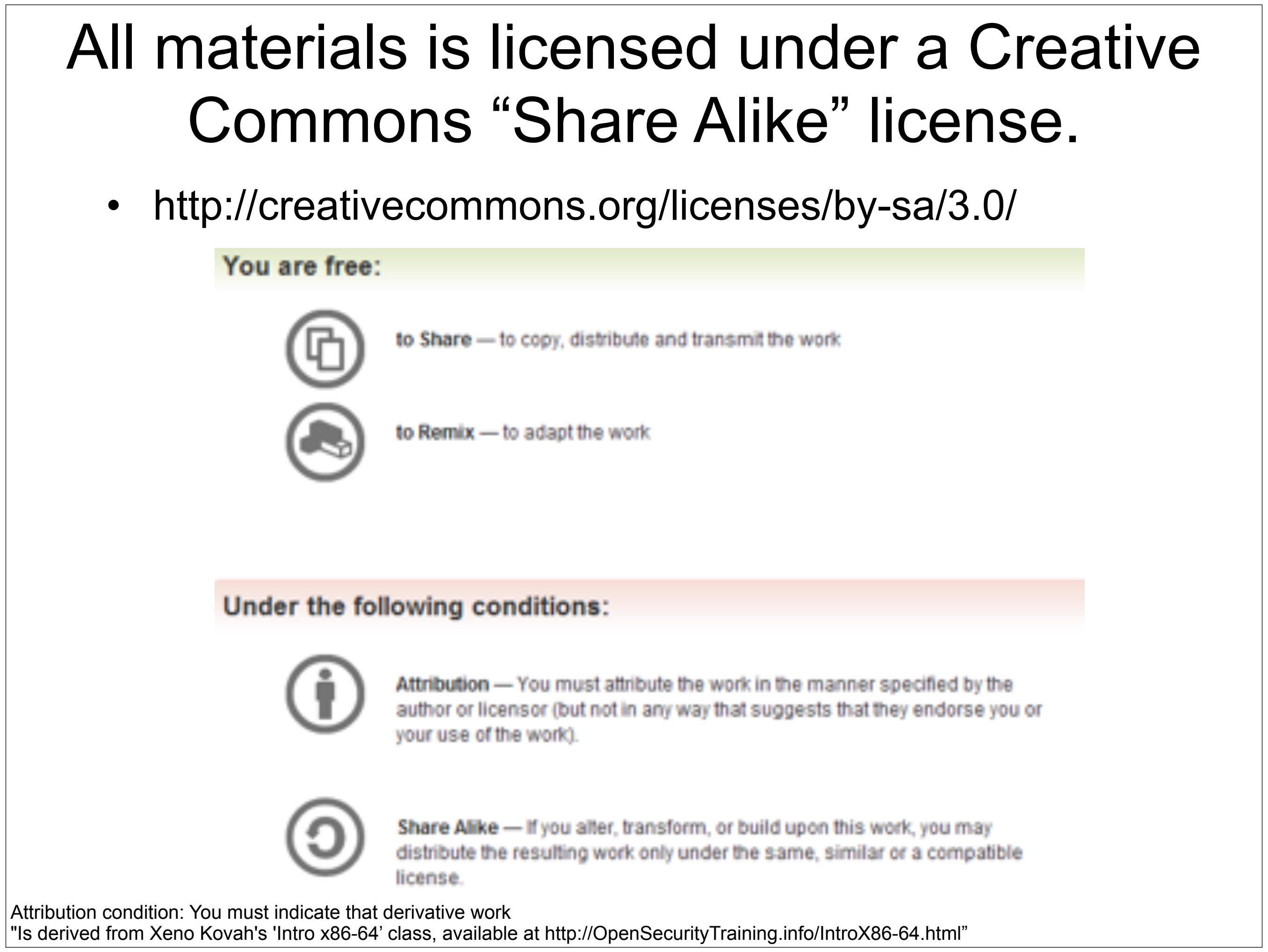
Introduction to Intel x86-64 Assembly, Architecture, Applications, & Alliteration

Xeno Kovah – 2014

xkovah at gmail



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"

ArrayLocalVariable2.c

Zero-initializing the array

```c
//ArrayLocalVariable2.c:
short main(){
    int a;
    short b[6] = {0};
    a = 0x100d;
    b[1] = (short)a;
    return b[1];
}
```

```
main:
0000000140001000  push      rdi
0000000140001002  sub       rsp,20h
0000000140001006  xor       eax,eax
0000000140001008  mov       word ptr [rsp+8],ax
000000014000100D  lea       rax,[rsp+0Ah]
0000000140001012  mov       rdi,rax
0000000140001015  xor       eax,eax
0000000140001017  mov       ecx,0Ah
000000014000101C  rep stos  byte ptr [rdi]
000000014000101E  mov       dword ptr [rsp],100Dh
0000000140001025  mov       eax,2
000000014000102A  imul      rax,rax,1
000000014000102E  movzx     ecx,word ptr [rsp]
0000000140001032  mov       word ptr [rsp+rax+8],cx
0000000140001037  mov       eax,2
000000014000103C  imul      rax,rax,1
0000000140001040  movzx     eax,word ptr [rsp+rax+8]
0000000140001045  add       rsp,20h
0000000140001049  pop       rdi
000000014000104A  ret
```

22

REP STOS - Repeat Store String

- STOS is one of number of instructions that can have the “rep” prefix added to it, which repeat a single instruction multiple times.
- All rep operations use *cx register as a “counter” to determine how many times to loop through the instruction. Each time it executes, it decrements *cx. Once *cx == 0, it continues to the next instruction.
- Either stores 1, 2, 4, or 8 bytes at a time
- Either fill 1 byte at [di] with al or fill 2/4/8 bytes at [*di] with *ax.
- Moves the *di register forward 1/2/4/8 bytes at a time, so that the repeated store operation is storing into consecutive locations.
- So there are 3 pieces which must happen before the actual rep stos occurs: set *di to the start destination, *ax/al to the value to store, and *cx to the number of times to store

Book p. 284

As with other instructions prefixes like “LOCK”, “REP” can only be used with certain instructions - as defined in the manual.

ArrayLocalVariable2.c takeaways

- If you're manually coding asm, REP STOS is functionally a memset()
- Sometimes when you use memset() from C, the compiler may turn it into a REP STOS

```c
//ArrayLocalVariable2.c:
short main(){
    int a;
    short b[6] = {0};
    a = 0x100d;
    b[1] = (short)a;
    return b[1];
}
```

```
main:
 push        rdi
 sub         rsp,20h
 xor         eax,eax
 mov         word ptr [rsp+8],ax
 lea         rax,[rsp+0Ah]
 mov         rdi,rax
 xor         eax,eax
 mov         ecx,0Ah
 rep stos    byte ptr [rdi]
 mov         dword ptr [rsp],100Dh
 mov         eax,2
 imul        rax,rax,1
 movzx       ecx,word ptr [rsp]
 mov         word ptr [rsp+rax+8],cx
 mov         eax,2
 imul        rax,rax,1
 movzx       eax,word ptr [rsp+rax+8]
 add         rsp,20h
 pop         rdi
 ret
```

ThereWillBe0xb100d.c

```
int main(){
        char buf[40];
        buf[39] = 42;
        return 0xb100d;
}
```

ThereWillBe0xb100d.c

```
main:
0000000140001010  push      rdi
0000000140001012  sub       rsp,60h
0000000140001016  mov       rdi,rsp
0000000140001019  mov       ecx,18h
000000014000101E  mov       eax,0CCCCCCCCh
0000000140001023  rep stos  dword ptr [rdi]
0000000140001025  mov       eax,1
000000014000102A  imul      rax,rax,27h
000000014000102E  mov       byte ptr buf[rax],2Ah
0000000140001033  mov       eax,0xb100d
0000000140001038  mov       edi,eax
000000014000103A  mov       rcx,rsp
000000014000103D  lea       rdx,[__xi_z+1A0h (0140006910h)]
0000000140001044  call      _RTC_CheckStackVars (01400010B0h)
0000000140001049  mov       eax,edi
000000014000104B  add       rsp,60h
000000014000104F  pop       rdi
0000000140001050  ret
```

rep stos setup

```
0000000140001016  mov        rdi,rsp
```
Set rdi - the destination

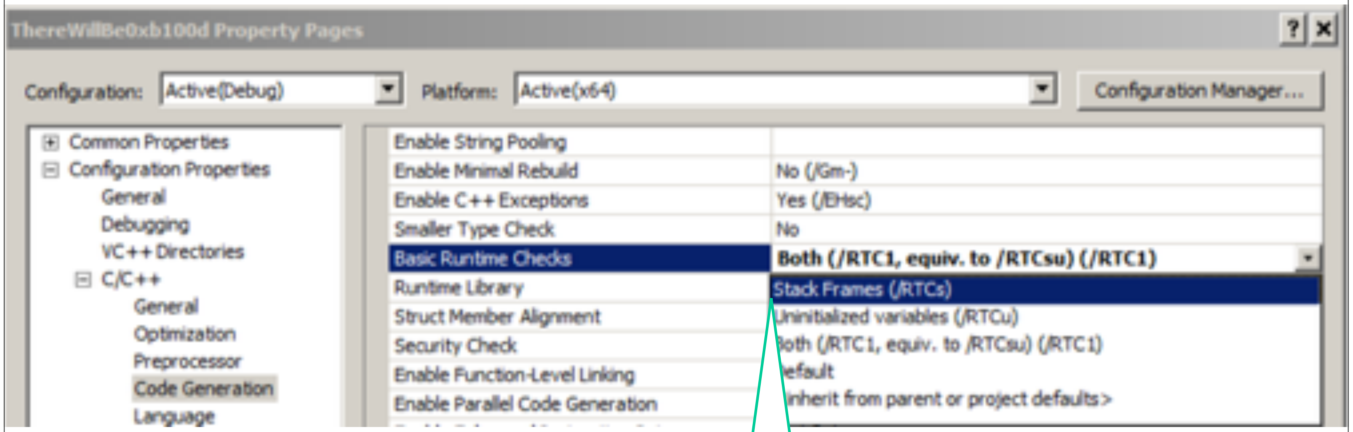
```
0000000140001019  mov        ecx,18h
```
Set ecx - the count

```
000000014000101E  mov        eax,0CCCCCCCCh
```
Set eax - the value

```
0000000140001023  rep stos   dword ptr [rdi]
```
Start the repeated store

- So what's this going to do? Store 0x18 copies of the dword 0xCCCCCCCC starting at rsp
- And that just happens to be 0x60 bytes of 0xCC, the entire reserved stack space!

Q: Where does the rep stos come from in this example?



ThereWillBe0xb100d Property Pages

Configuration: Active(Debug) Platform: Active(x64) Configuration Manager...

Enable String Pooling	
Enable Minimal Rebuild	No (/Gm-)
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Runtime Library	Stack Frames (/RTCs)
Struct Member Alignment	Uninitialized variables (/RTCu)
Security Check	Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Enable Function-Level Linking	Default
Enable Parallel Code Generation	<inherit from parent or project defaults>

A: Compiler-auto-generated code. From the stack frames runtime check option. This is enabled by default in the debug build. Disabling this option removes the compiler-generated code.

More straightforward without the runtime check

```
main:
0000000140001010  sub        rsp,38h
0000000140001014  mov         eax,1
0000000140001019  imul       rax,rax,27h
000000014000101D  mov         byte ptr [rsp+rax],2Ah
0000000140001021  mov         eax,0B100Dh
0000000140001026  add        rsp,38h
000000014000102A  ret
```

But still not entirely clear :)

Instructions we now know (29)

- NOP
- PUSH/POP
- CALL/RET
- MOV
- ADD/SUB
- IMUL
- MOVZX/MOVSX
- LEA
- JMP/Jcc (family)
- CMP/TEST
- AND/OR/XOR/NOT
- INC/DEC
- SHR/SHL/SAR/SAL
- DIV/IDIV
- REP STOS