Why 64-bit Malware

- AMD64 known as x64 or x86-64
  - Similar to 32-bit x86
  - Not all tools support it
  - Backwards compatibility support for 32-bit ensures that 32-bit malware can run on both 32-bit and 64-bit machines

- Reasons for 64-bit malware
  - Malware runs with 64-bit process also needs to be 64 bit
  - Kernel rootkits must be compiled in 64-bit for machines to run on a 64-bit OS
    - Microsoft enhanced security in 64 bit – made unauthorized modification of kernel code difficult
  - **Malware and shellcode being injected into a 64-bit process must be 64-bit (target process/application is 64 bit)** – IE Explorer
Differences in x64 Architecture

- 64-bit vs. 32-bit x86
  - All addresses and pointers 64-bit
  - All general purpose registers 64-bit
  - Special purpose registers also 64-bit
  - Double the general purpose registers (new registers: %r8-%r15)
    - %r8 = 64-bits
    - %r8d = 32-bit DWORD
    - %r8w = 16-bit WORD
    - %r8l = 8-bit LOW byte

- Point-relative data addressing (64 bit)
  - x86 absolute addressing – instruction must store entire address
  - x64 RIP-relative data addressing
Differences in x64 Architecture

● Example:

```
00401004 A1 17 4 2 3 3 4 0 0 mov eax, dword_403374
```

32 bit – actual address recorded, not position independent

```
0000000140001058 8B 05 1A 2 D 3 0 0 0 mov eax, dword_14000E400
```

0x0000D3A2 offset (opcode), 14000E400 address (automatically solved by IDApro)

64 bit – decrease the number of addresses to be loaded when DLL is loaded

● Make easy for shellcode, make them hard to detect
Differences in x64 Architecture

- 64-bit vs. 32-bit x86 calling convention and stack usage
  - 64 bit similar to “fastcall” – recall “fastcall” first four arguments of the function passed to `rdx, %rcx, %r8, %r9`, the rest ones pushed to the stack, caller cleans the stack
  - Stack space is allocated at beginning of function call for the duration of the call (i.e. no push/pop within function) (Figure 21-1, p. 445)
  - Microsoft's 64-bit exception handling model assumes a static stack (flat stack size in function call)

Figure 21-1: Stack size in the same function compiled for 32-bit and 64-bit architectures
Leaf and Nonleaf Functions

- Leaf and Nonleaf Functions
  - Functions that call other functions are non-leaf or frame functions as they require a stack frame to be allocated
  - Nonleaf functions required to allocate at least 0x20 bytes of stack space when calling another function to save register parameters (local variables increase allocation beyond 0x20)
  - Why 0x20 bytes (256 bytes)? RCX, RDX, R8, R9 registers – each 64 bits
  - Leaf functions no need to maintain stack.
Windows 32-Bit on Windows 64-Bit

- Notice 32-bit program can also run on 64-bit windows
- **WOW64 subsystem allows 32-bit Windows to run on 64-bit Windows – backwards compatibility**
  - Separate DLLs for both 32- and 64-bit processes to avoid conflicts
  - 64-bit stored in `C:\Windows\System32`; 32-bit stored in `C:\Windows\WOW64` – counterintuitive
  - So if 32-bit malware running on 64-bit Windows writes a file to `C:\Windows\System32`, it shows up in `C:\WINDOWS\WOW64`
- Similar redirection for registry keys
- 32-bit malware wishing to “break-out” of WOW64 to infect 64-bit system can disable redirection (`IsWow64Process`, access `C:\Windows\Sysnative`, `Wow64DisableWow64FsRedirection` disable redirection)
64-Bit Hints at Malware Functionality

- Easier to differentiate pointer and data values in 64-bit
  - Integers often stored in 32-bit values
  - Pointers are always 64-bit
  - Can differentiate non-pointers from pointers via size of register used (Table 21-1, p. 448)

<table>
<thead>
<tr>
<th>32-bit assembly listing</th>
<th>64-bit assembly listing</th>
</tr>
</thead>
<tbody>
<tr>
<td>004114F2 mov eax, [ebp+var_8]</td>
<td>0000000140001148 mov rdx, [rsp+38h+var_18]</td>
</tr>
<tr>
<td>004114F5 push eax</td>
<td>000000014000114D mov ecx, [rsp+38h+var_10]</td>
</tr>
<tr>
<td>004114F6 mov ecx, [ebp+var_14]</td>
<td>0000000140001151 call sub_411186</td>
</tr>
<tr>
<td>004114F9 push ecx</td>
<td></td>
</tr>
<tr>
<td>004114FA call sub_411186</td>
<td></td>
</tr>
</tbody>
</table>

32 bit – eax ecx all 32 bit knows nothing
Rdx 64 bit, ecx 32 bit version of Rdx -> knows this parameter is not a pointer