presented by

UEFI Security Defenses

UEFI Spring Plugfest – May 8-10, 2012

Presented by Dick Wilkins

Dick_Wilkins@phoenix.com

Phoenix Technologies Ltd.

Agenda

- Introduction
- Defensive Security Goals
- Stack Buffer Overrun Detection (/GS, /RTC)
- Heap Corruption Detection
- Data Execution Prevention (DEP) / No eXecute (NX)
- Address Space Location Randomization (ASLR)
- Conclusion
- Q & A

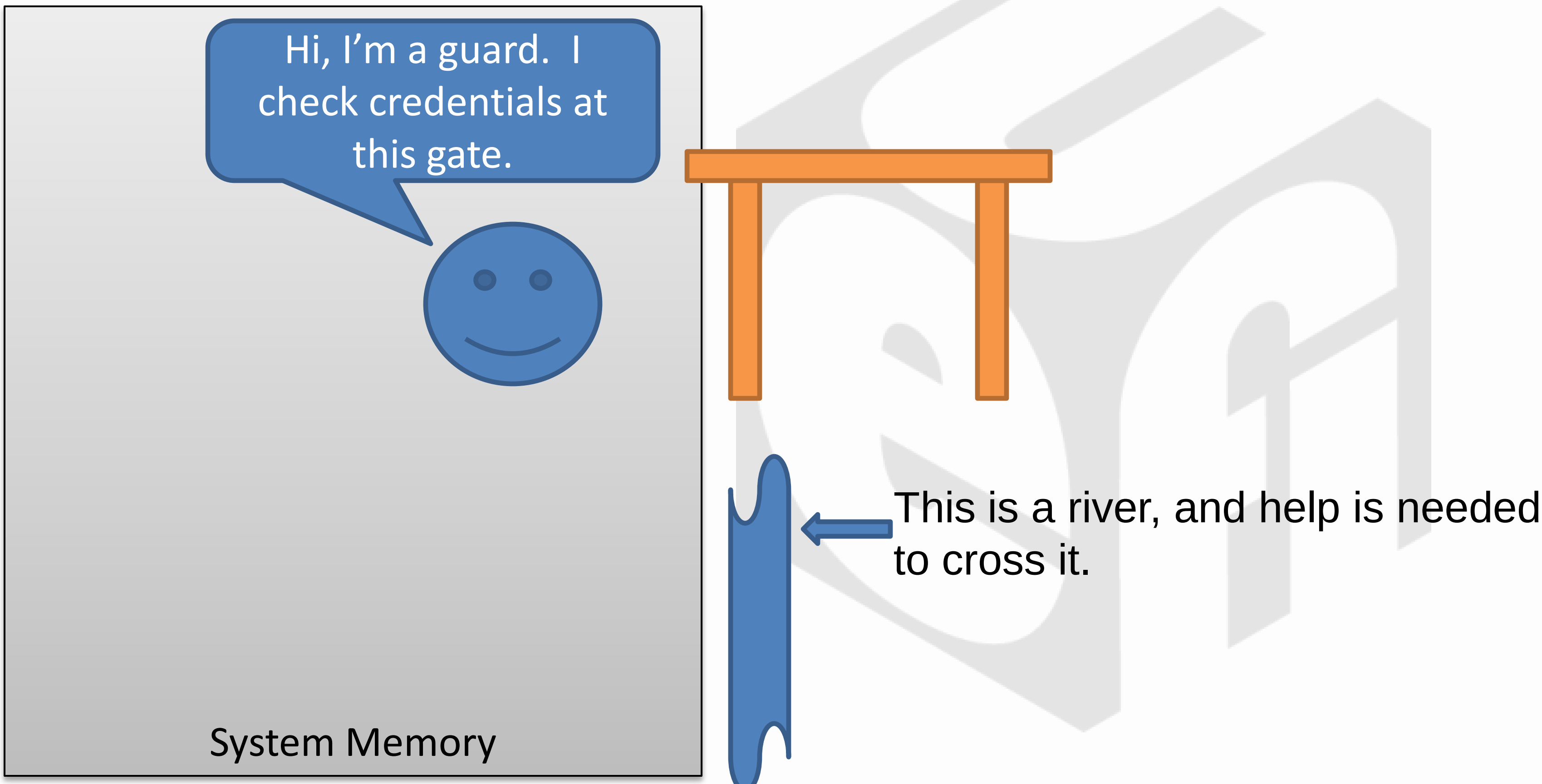
Introduction

We will be discussing security defenses that harden UEFI BIOS implementations against attacks

The defenses discussed here have been added to EDK 2 as part of a collaboration between Microsoft and Phoenix Technologies Ltd.

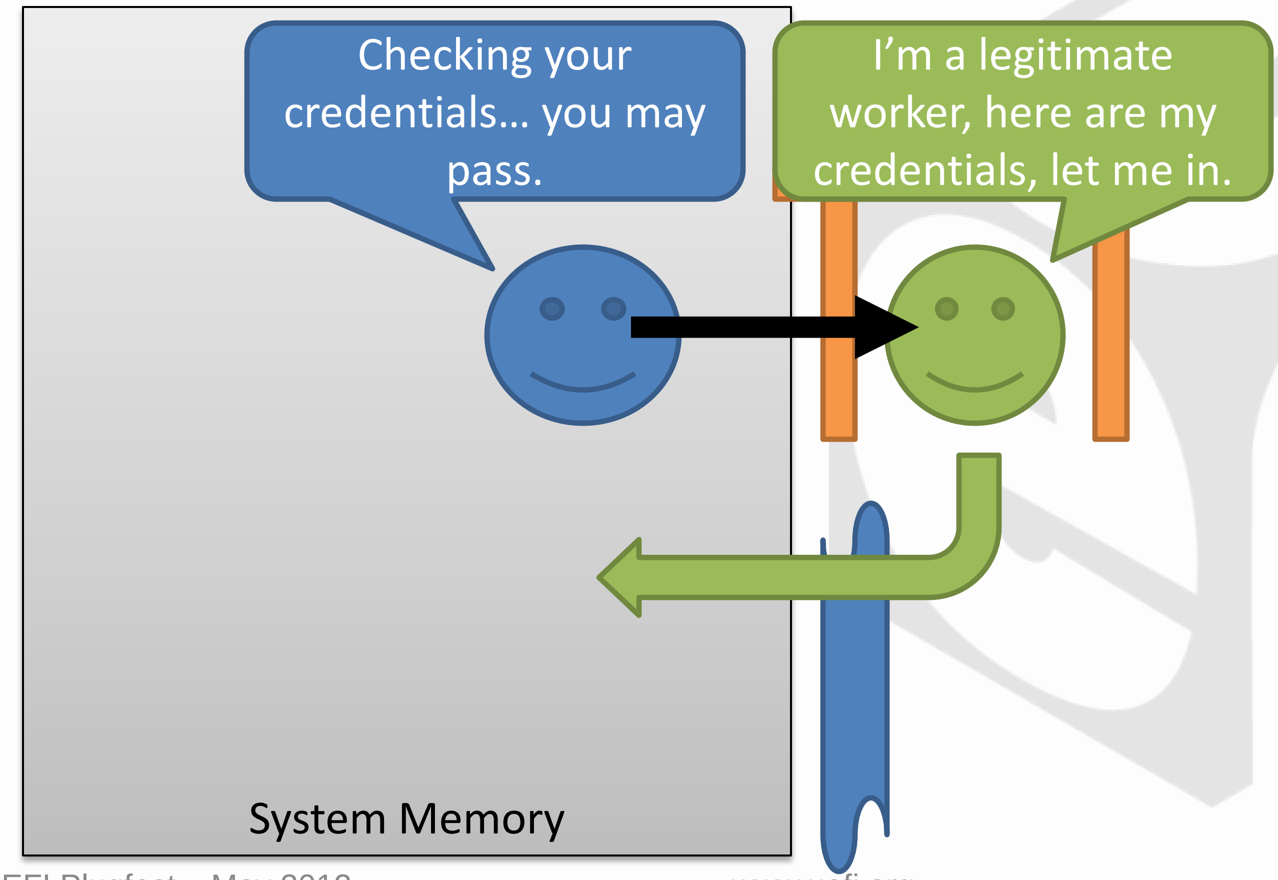
Defensive Security Goals

- Imagine the BIOS as a guarded gateway



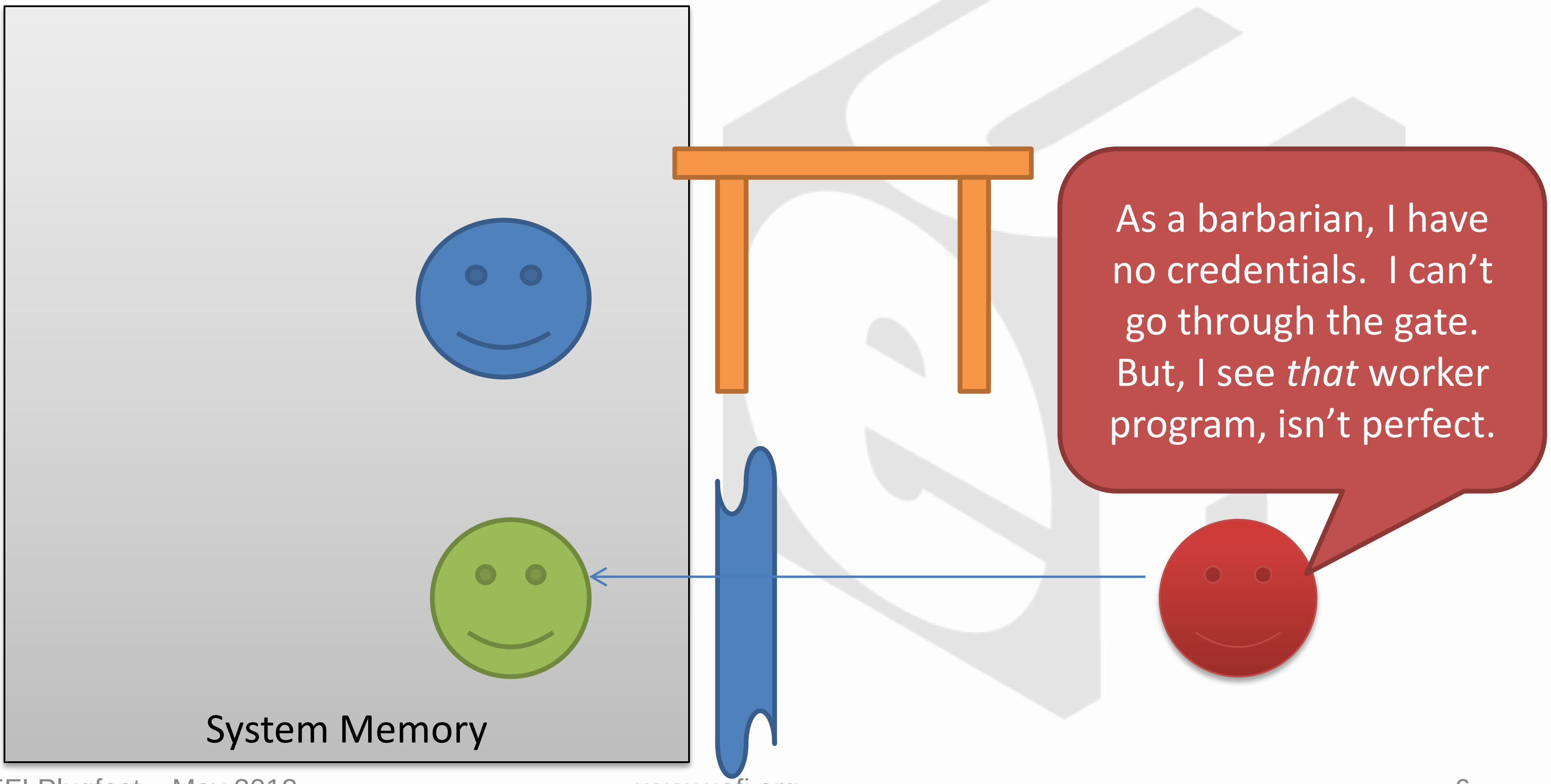
Defensive Security Goals

- Guards are good at checking credentials



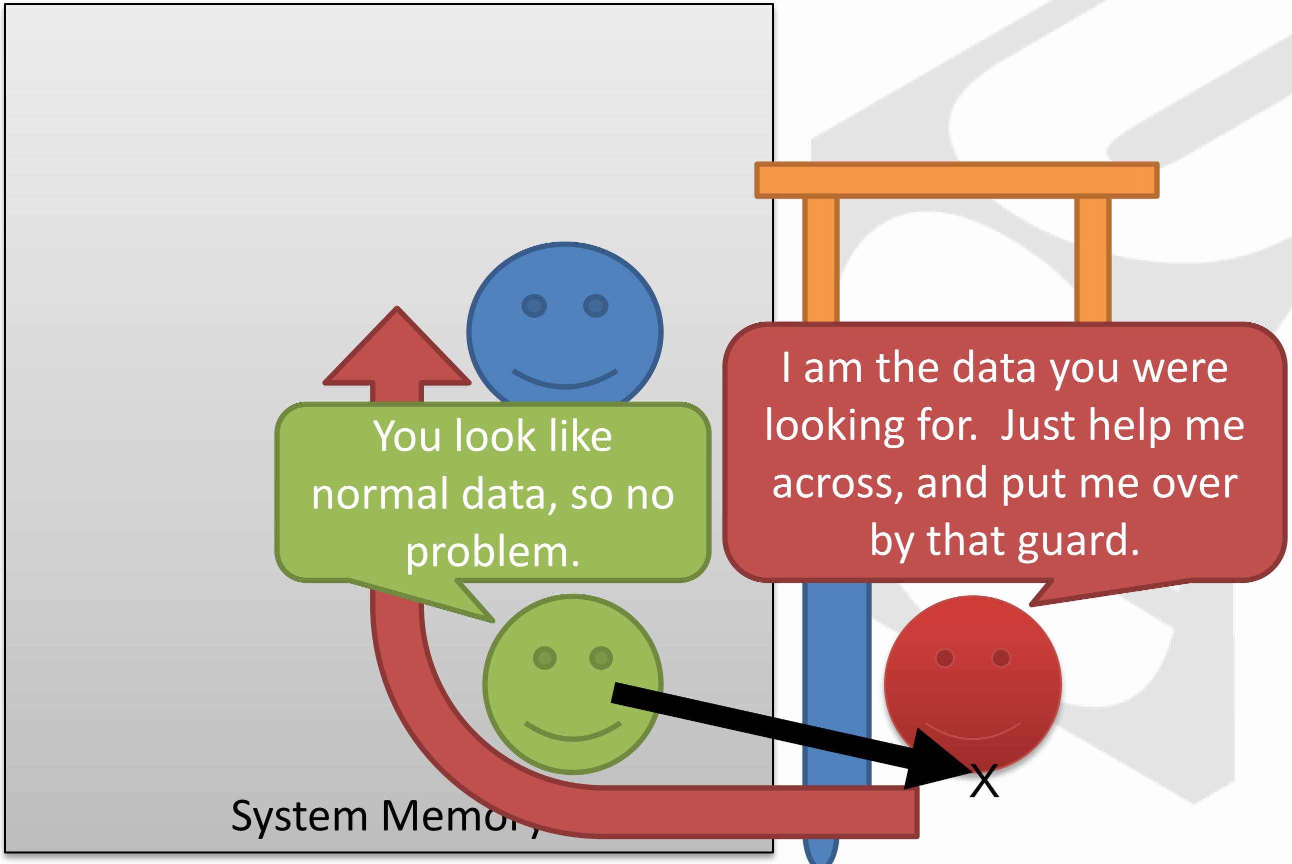
Defensive Security Goals

- Attackers want to get past the guard



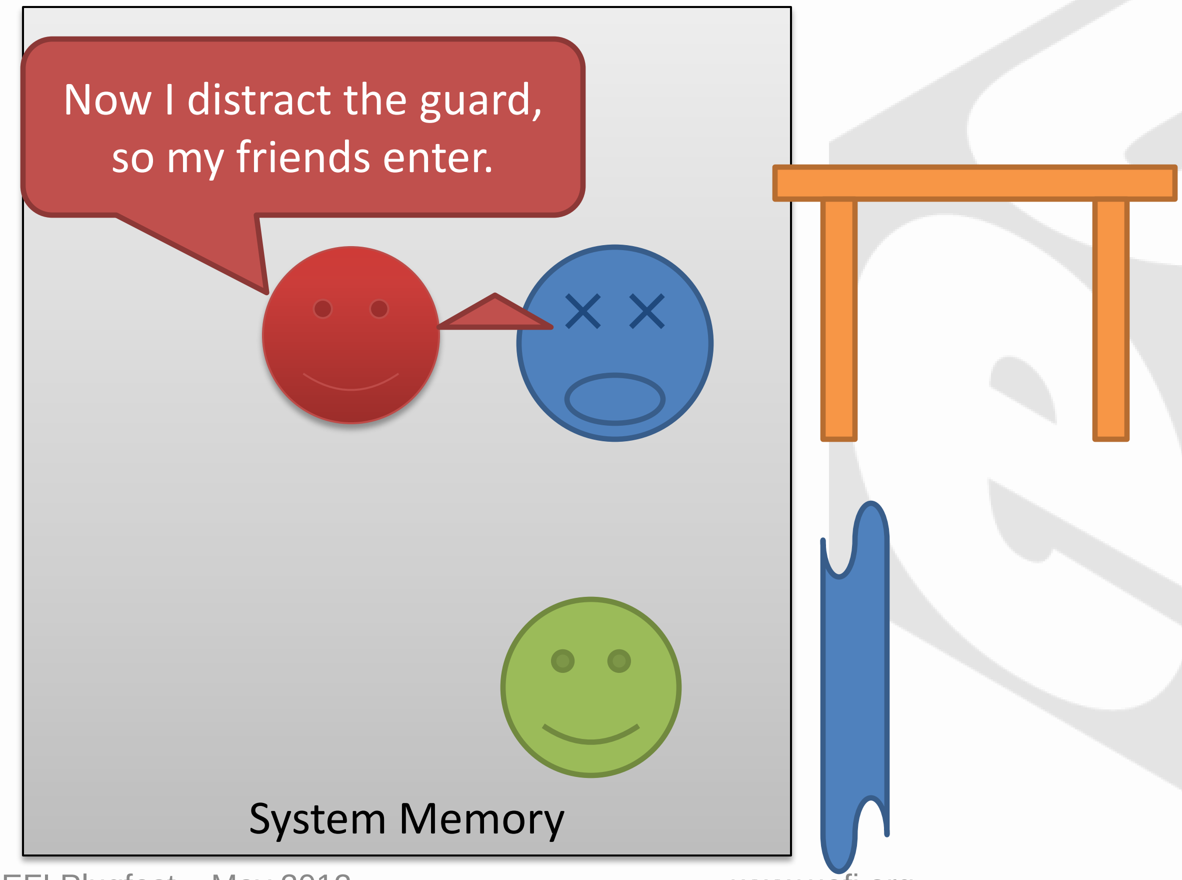
Defensive Security Goals

- Workers do not always check credentials



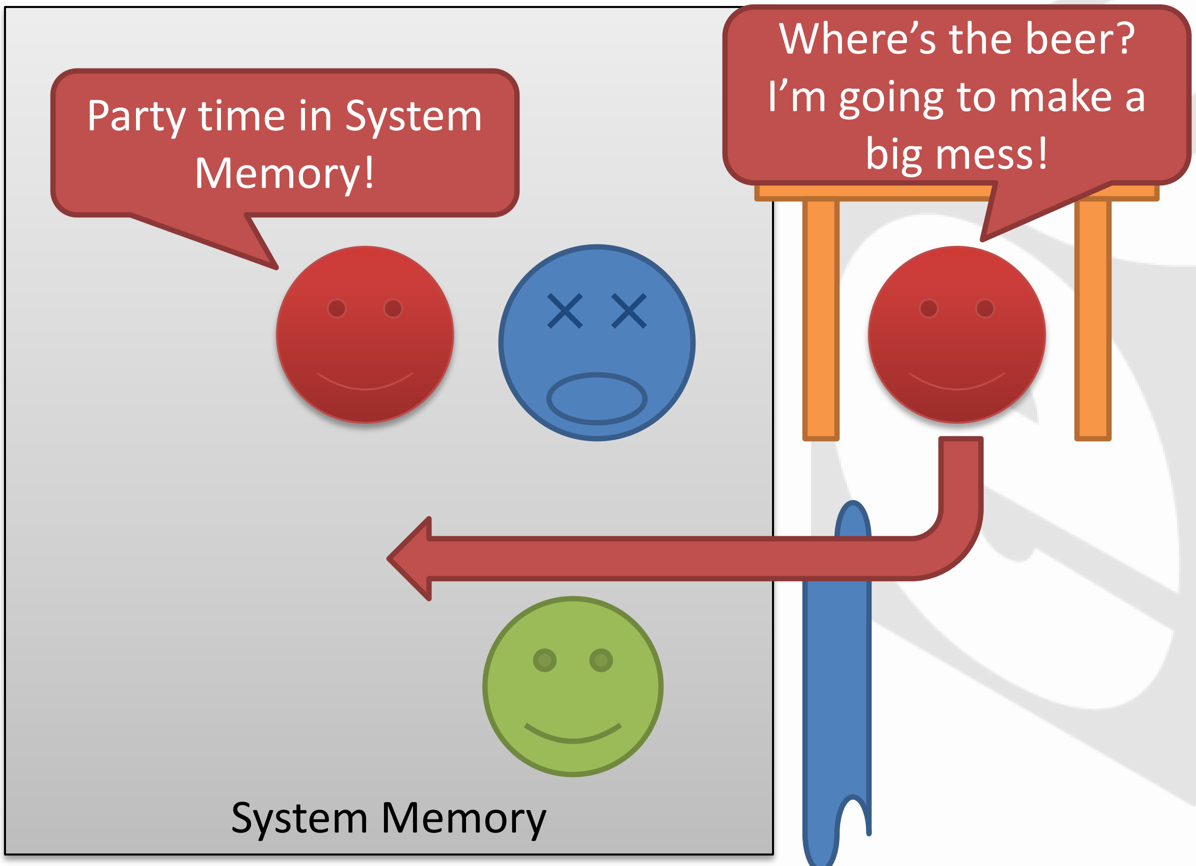
Defensive Security Goals

- If the attacker distracts the guard…



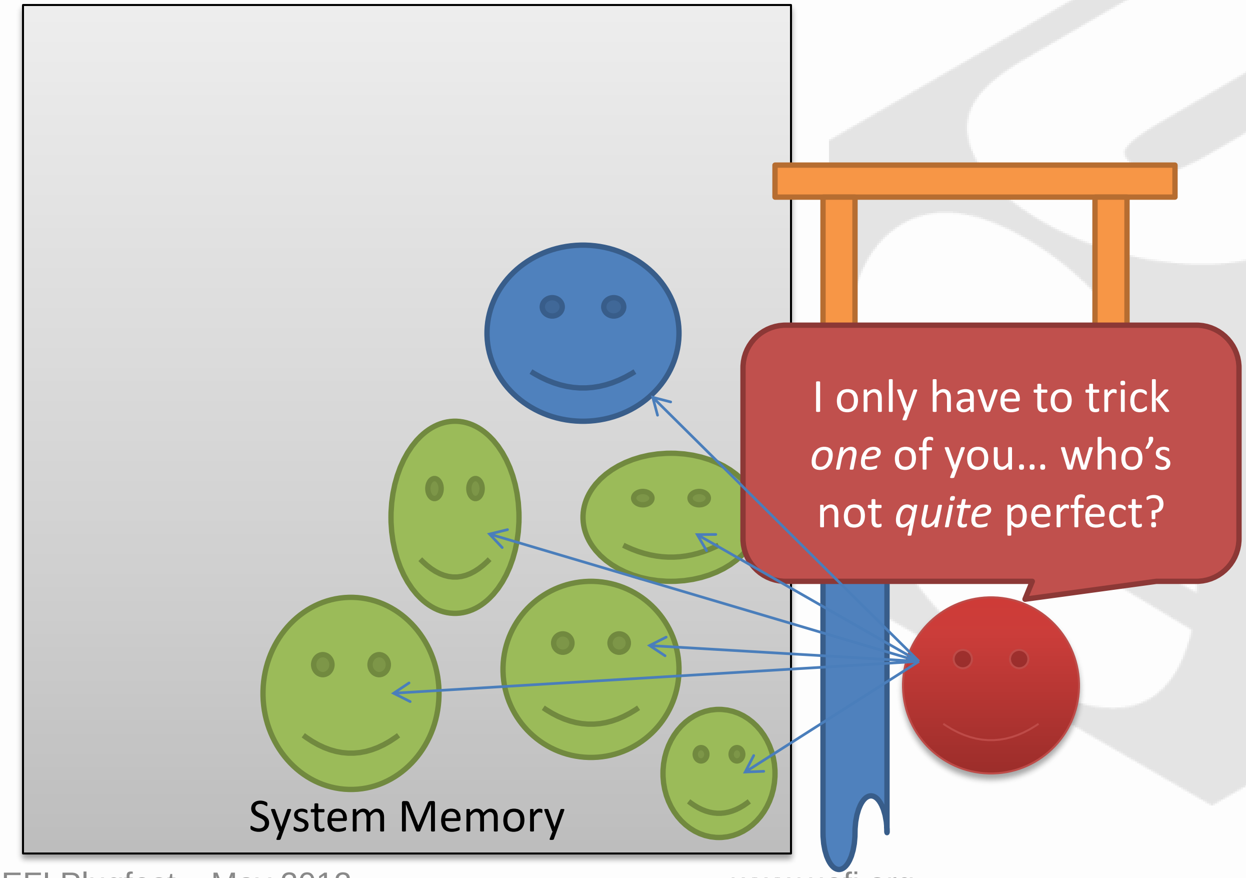
Defensive Security Goals

- We have to prevent this from happening!



Defensive Security Goals

- Too much code to be sure it's all perfect



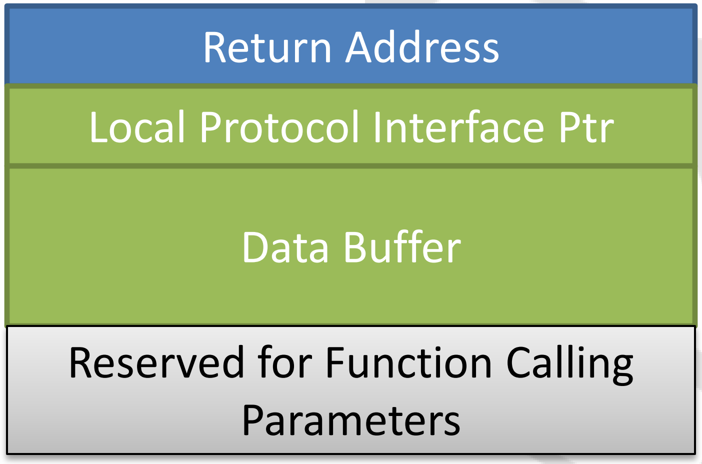
Protecting from Stack Buffer Overruns

Stack Buffer Overrun Detection

- Goal: Detect Buffer Overruns on the Stack
  - Local Variables are stored on the stack
  - Function return addresses are stored on the stack
- Note that the intent of buffer overrun detection is to expose coding errors at runtime during testing that could compromise security, not to provide perfect protection from all possible buffer overrun attacks

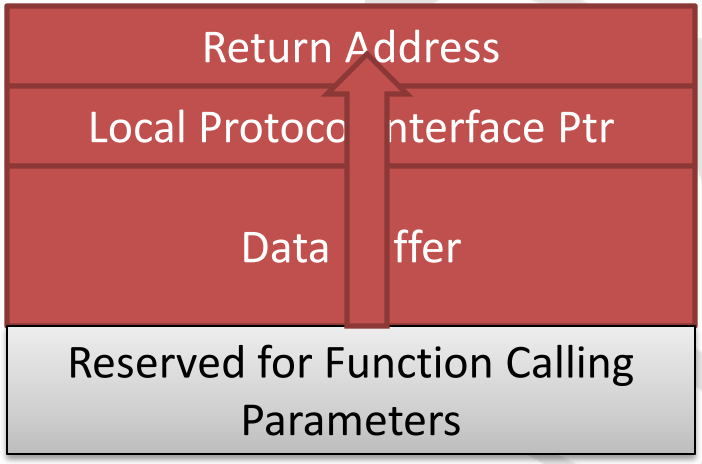
Stack Buffer Overrun Detection

- Illustration of a vulnerable stack frame



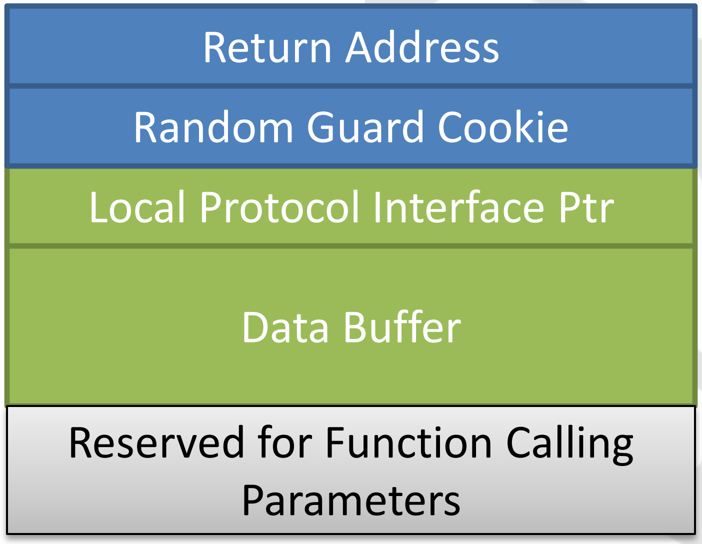
Stack Buffer Overrun Detection

- Buffer overflows occur when a function does not correctly check the amount of data being transferred into a buffer



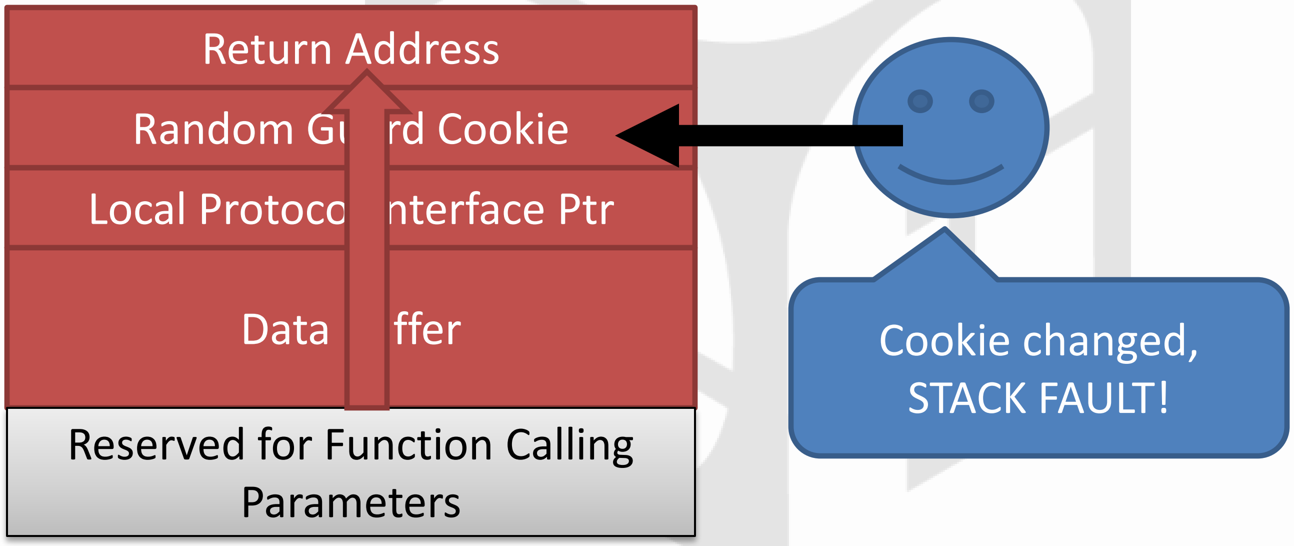
Stack Buffer Overrun Detection

- The MSVC /GS compiler switch inserts a randomized guard cookie onto the stack between the return address and locals



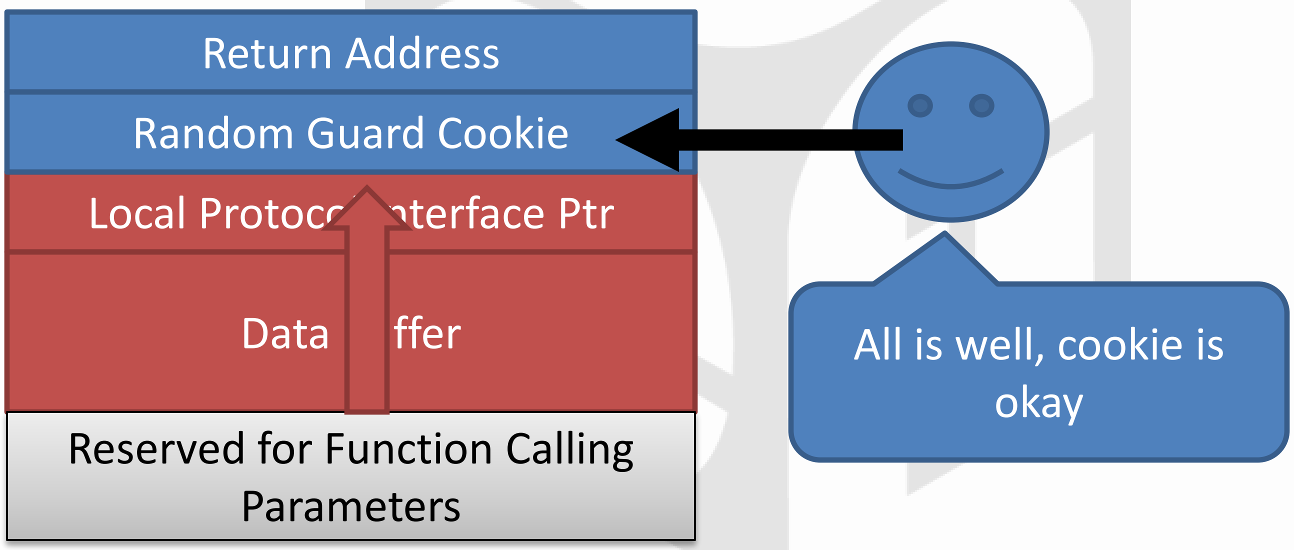
Stack Buffer Overrun Detection

- Changing the return address with a buffer overflow requires changing the guard cookie, so such overflows are detected



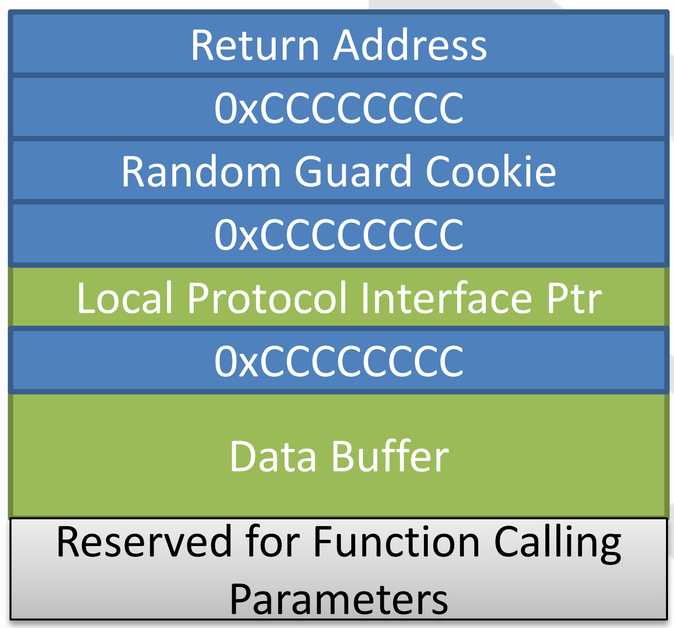
Stack Buffer Overrun Detection

- /GS does NOT detect changes to locals if the buffer overrun doesn't reach the guard cookie



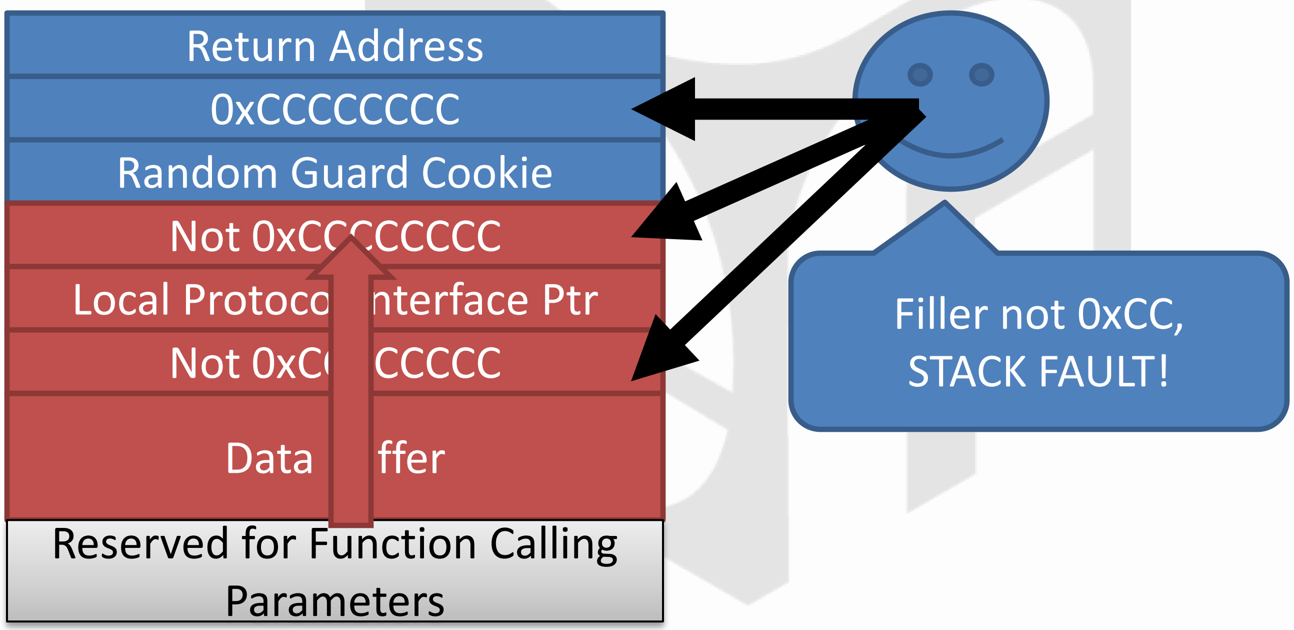
Stack Buffer Overrun Detection

- The MSVC /RTCs compiler switch inserts 0xCC onto the stack between local variables



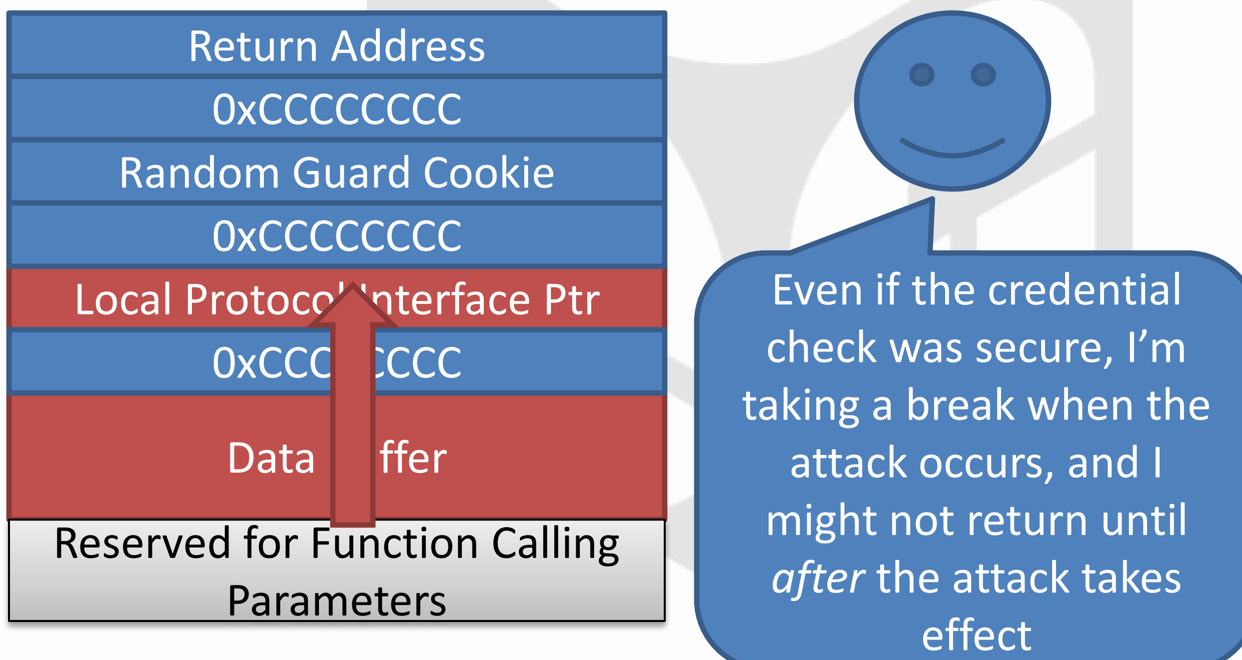
Stack Buffer Overrun Detection

- If a buffer overflow changes the filler between locals, that overflow is detected



Stack Buffer Overrun Detection

- However, 0xCC is easy to forge, and the check comes just before return, not immediately after the buffer overflow



Stack Buffer Overrun Detection

- Note that /RTC switches require that optimizations be disabled
- Because of the insecure signature and the optimization disable requirement, the current /RTC implementation should be considered a debugging feature meant to help identify buffer overflows, and can not provide more security in a release build than is already provided by /GS

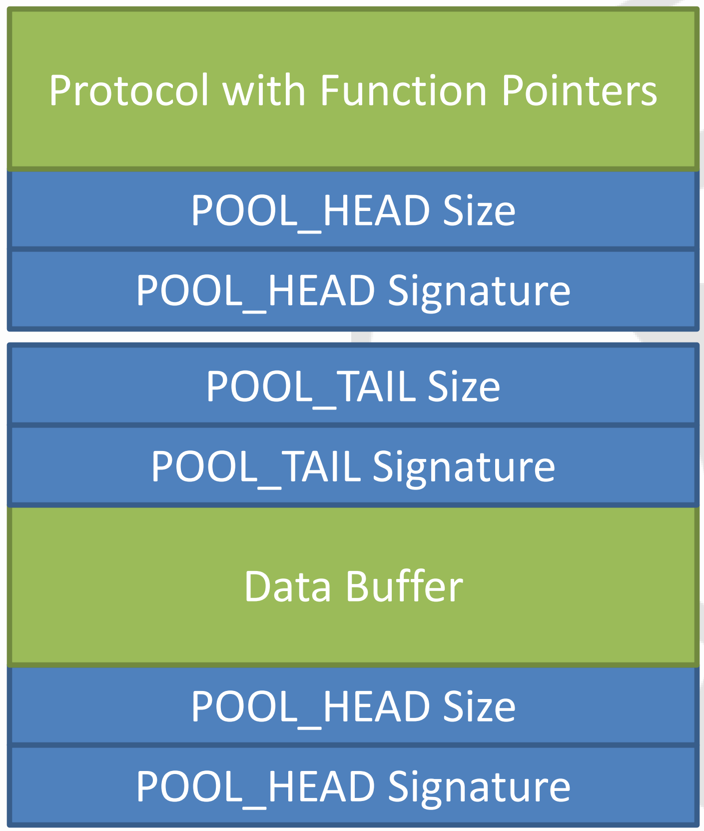
Heap Corruption Detection

Heap corruption detection

- Goal: Detect Buffer Overruns on the Heap
  - Protocols with function pointers are typically stored on the heap
  - Dynamically sized buffers are also usually allocated and stored on the heap

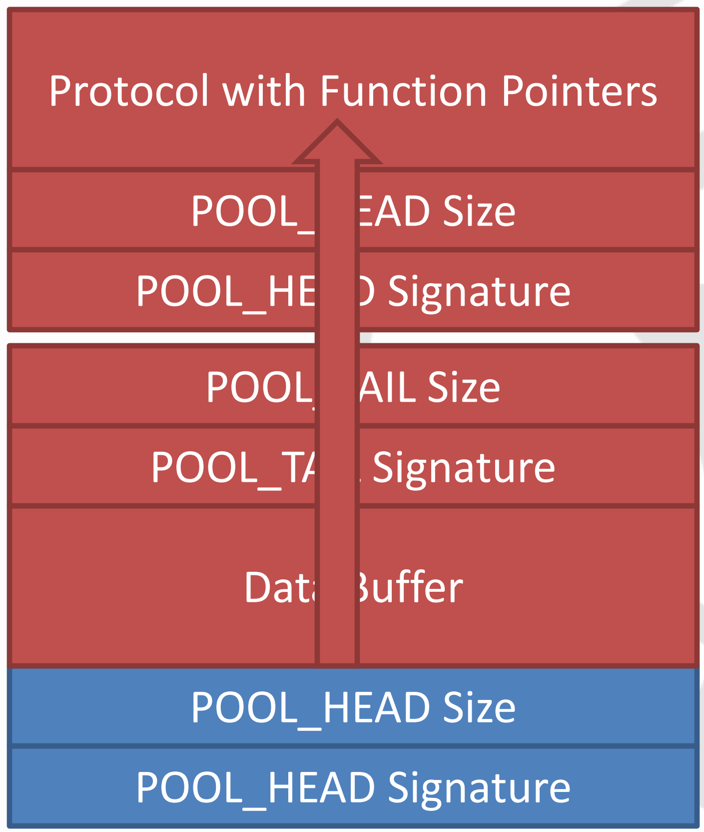
Heap corruption detection

- Heap is dynamically allocated memory



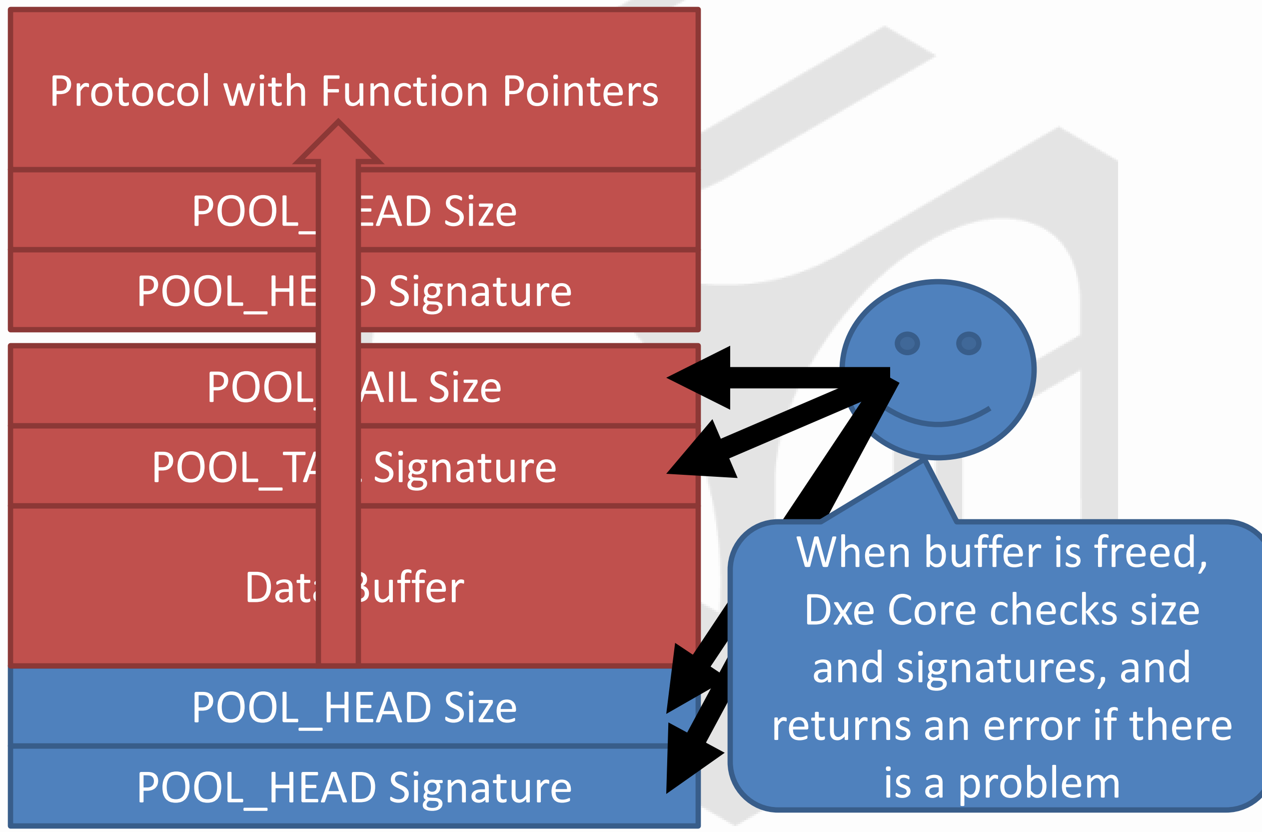
Heap corruption detection

- Buffer Overflow attack on Heap looks like



Heap corruption detection

- Existing signature checks should catch



Heap corruption detection

- Remaining ways to improve heap corruption detection
  - More of the heap should be verified in addition to those signatures around specific memory under consideration during heap free or allocate calls
  - Full validation of heap should occur periodically, outside the context of allocate and free calls
  - Heap signatures should be encrypted at run-time using XOR with a random number to prevent signature forgery by attackers
  - Failed signature checks should throw an exception, rather than returning an error, as few clients of "free" function calls check for or handle error conditions returned by free
  - Guard pages (which are write protected or for which page presence bit is clear) should be placed between code and data pages

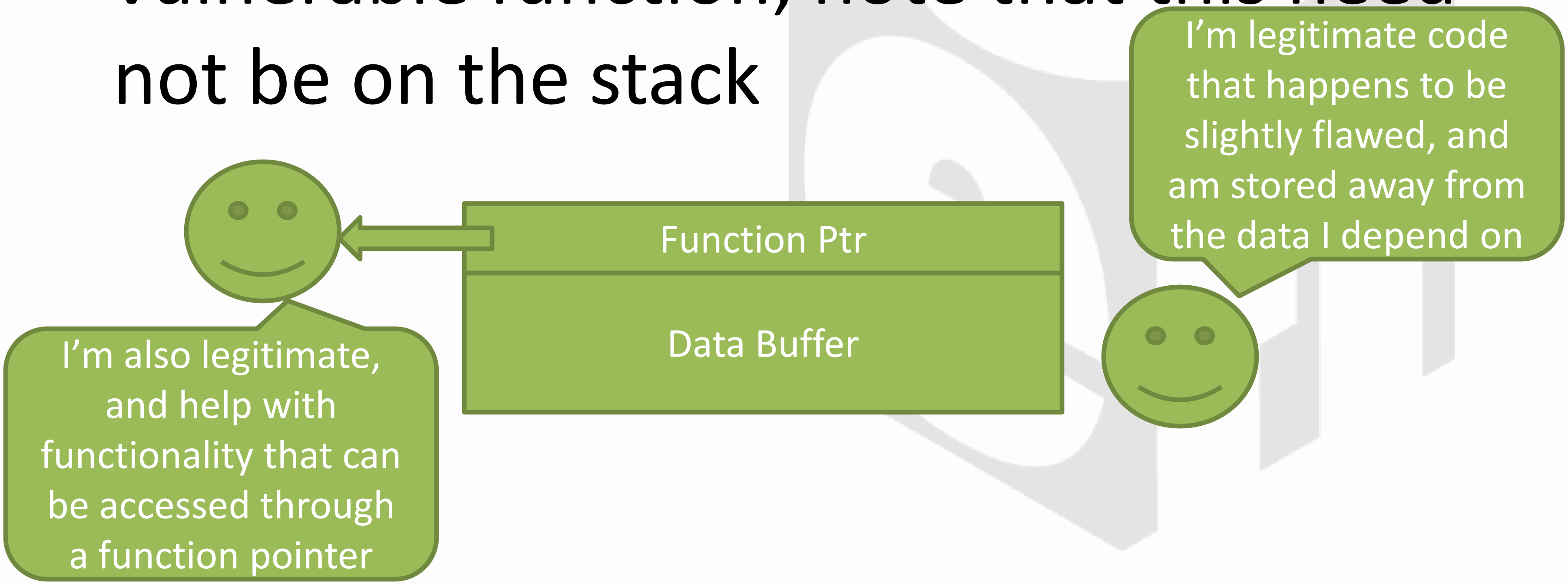
Prevention of the Execution of Data

Data Execution Prevention

- Goal: Prevent usage of data buffers as storage for exploit code
- Enabling CPU Technology
  - Modern x86 CPUs provide support for NX as a bit that can be set in PAE and IA32E page tables (called XD in Intel Volume 3)
  - Setting the execute disable bit in a page table entry causes the processor to page fault when fetching code from the associated page

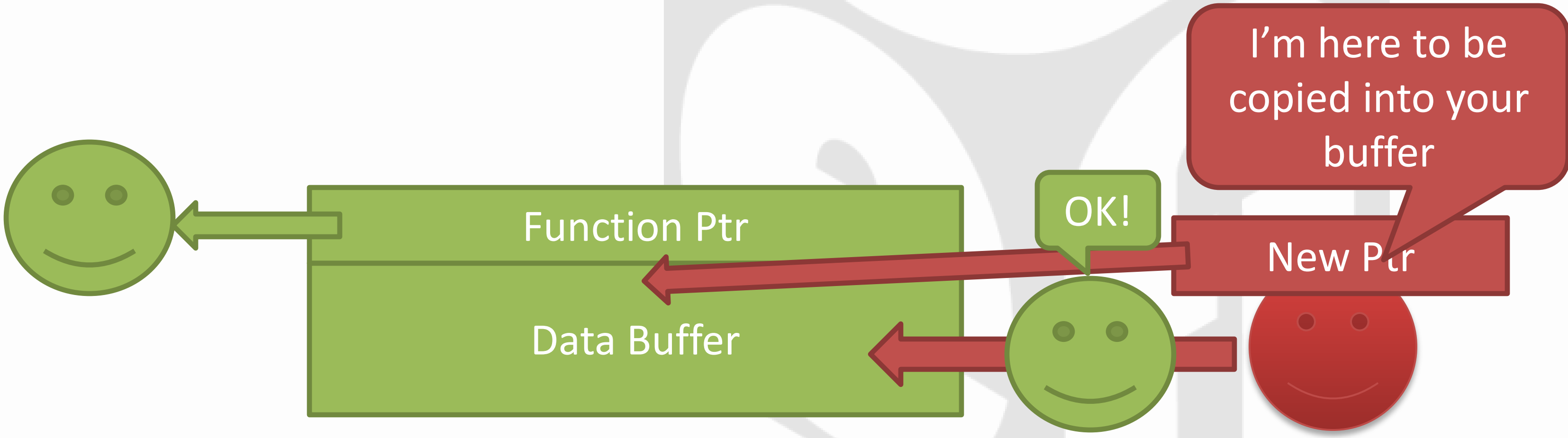
Data Execution Prevention

- Illustration of a vulnerable variable arrangement in the context of a vulnerable function; note that this need not be on the stack



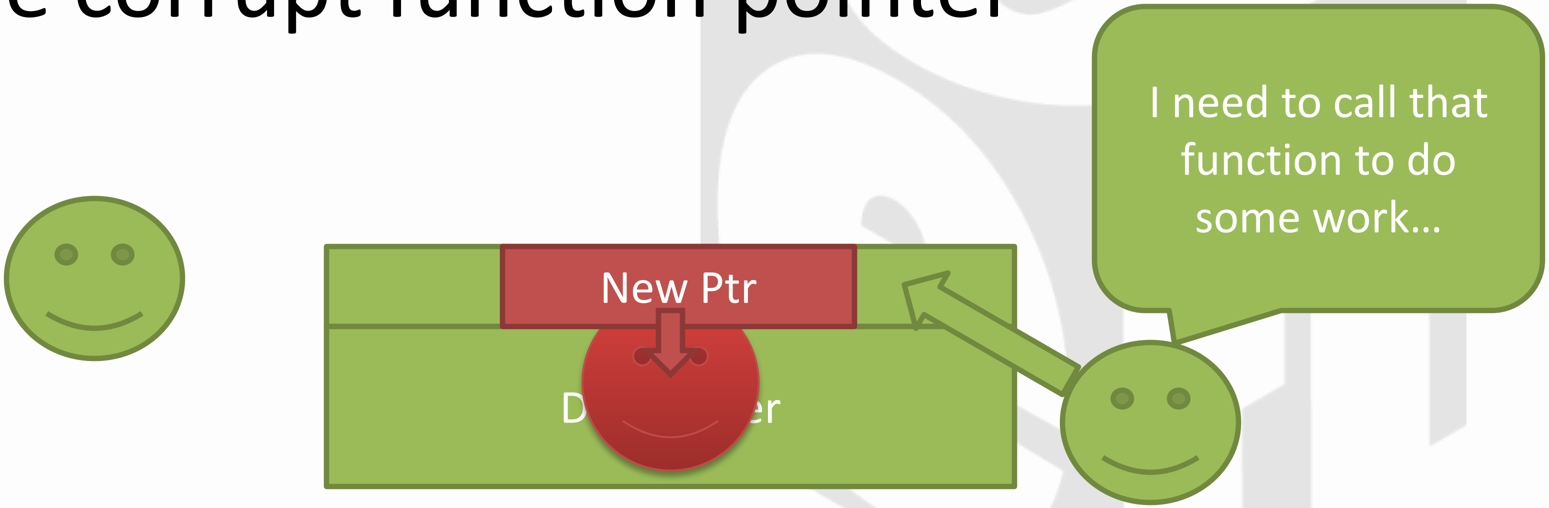
Data Execution Prevention

- The attacker tricks vulnerable code into copying an exploit into the buffer and altering the function pointer to point to it



Data Execution Prevention

- Some function, perhaps the vulnerable function, calls the exploit code through the corrupt function pointer

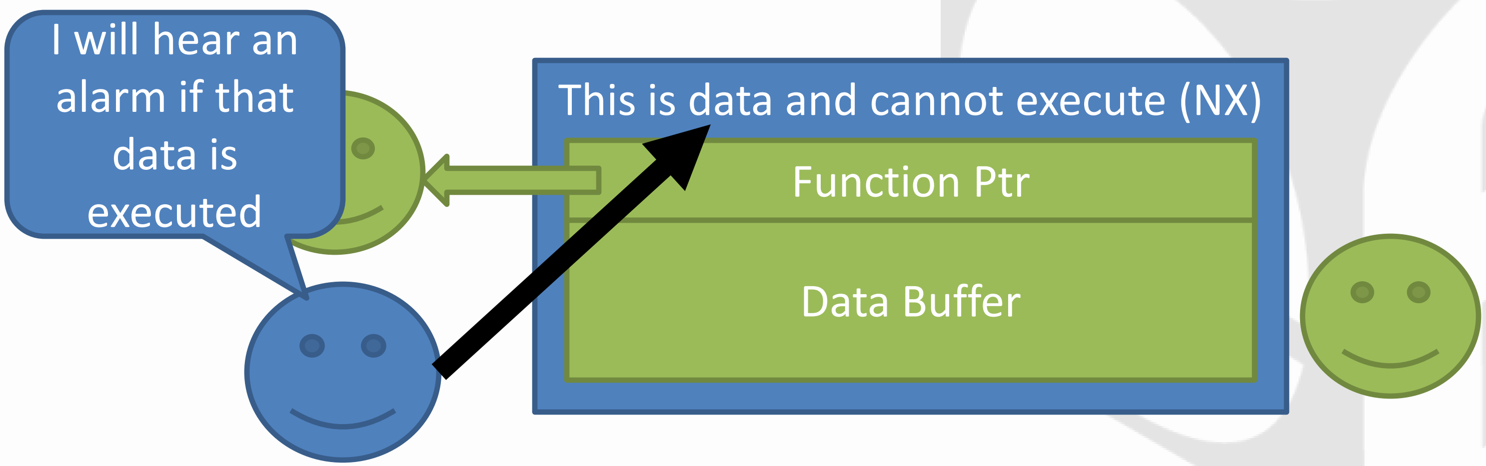


Data Execution Prevention

- The exploit code now has control and can do nearly anything

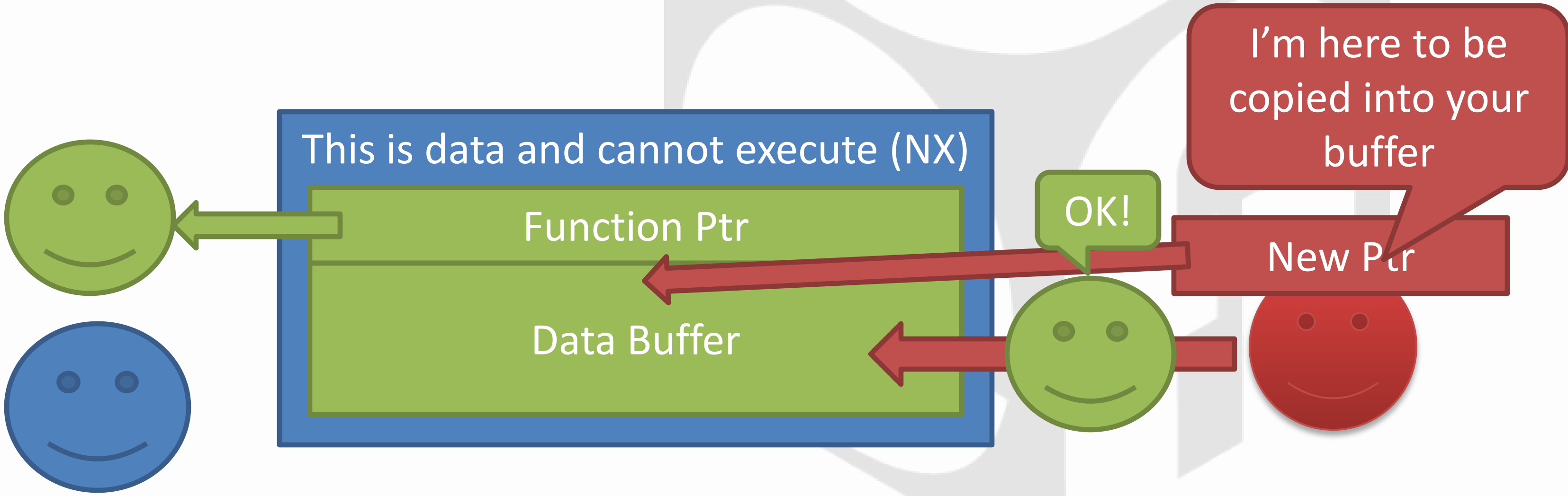
Data Execution Prevention

- With data execution prevention, DXE Core marks all memory that is definitely data as “NX” or No eXecute



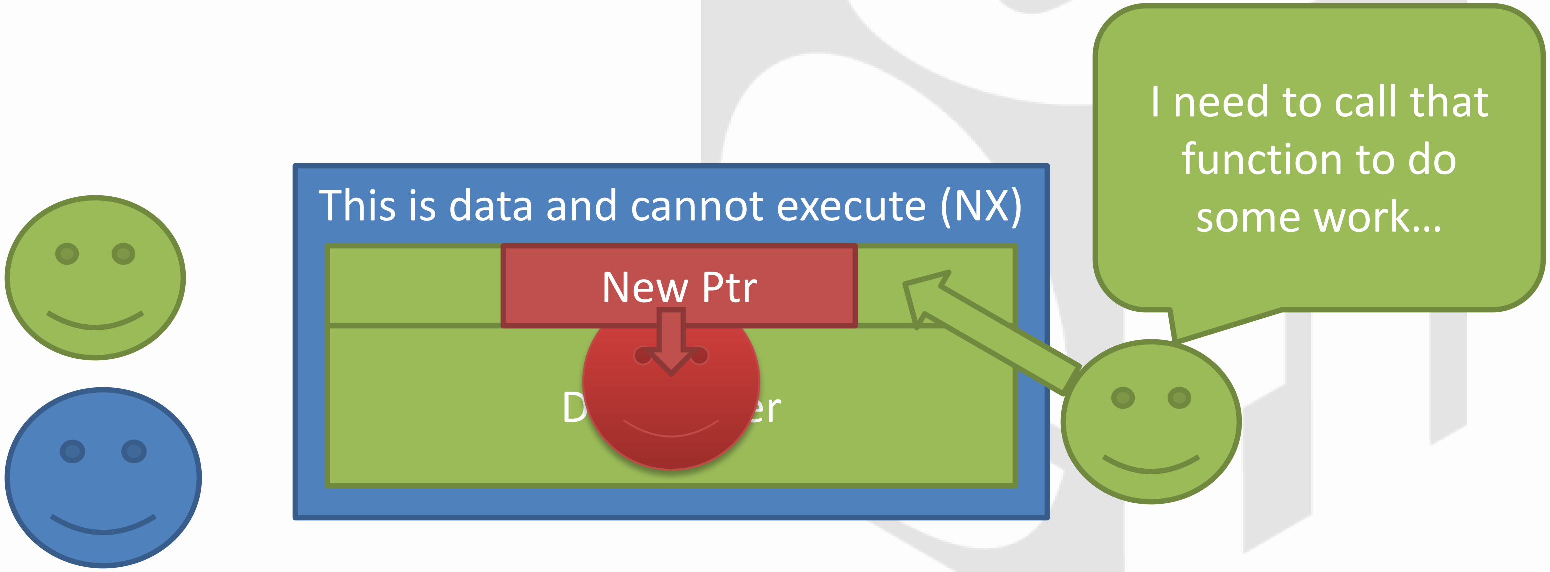
Data Execution Prevention

- The exploit code is still copied into the buffer by the vulnerable code



Data Execution Prevention

- Somewhere, the exploit is still called through the modified function pointer



Data Execution Prevention

- But the (NX) protection on the data pages acts as an alarm, and the page fault handler in DXE core is called before the exploit can execute

Data Execution Prevention

- Pre-requisites
  - Memory from which code is to be executed must be allocated as one of the following types
    - EfiReservedMemoryType
    - EfiLoaderCode
    - EfiBootServicesCode
    - EfiRuntimeServicesCode
    - EfiACPIMemoryNVS
  - IA32_EFER.NXE (MSR 0xC0000080 bit 11) must be set for the BSP and all APs when IA32_EFER.LME (MSR 0xC0000080 bit 8) is set

Data Execution Prevention

- Phoenix NX Implementation
  - New BasePageTableLib MdePkg library
    - BasePageTableLib contains stub functions
    - BasePageTableLibIA32E contains IA32E page table support (used for X64)
  - Enabling PcdPageTableNxSupport causes DxeIplPeim to enable IA32_EFER.NXE
  - Enabling PcdPageTableLibrarySupport causes DxeCore to call the Page Table Library functions when pages are allocated

Data Execution Prevention

- Platform and Silicon Considerations
  - All application processor (AP) entry vector setup code that sets bit 8 of MSR 0xC0000080 must also set bit 11 before enabling paging using the boot processors (BP) page tables
  - Very early SMM initialization code re-uses the boot processors page tables
    - AllocatePages must be used to set memory from 0x38000 to 0x40000 to EfiReservedMemoryType during the first SMI
    - The code that sets bit 8 of MSR 0xC0000080 must also set bit 11 when the first SMI occurs
- You will know if you missed anything
  - System will reboot or lock up, depending on current IDT and the conditions of the fault
  - A fetch from data address space will cause a page fault

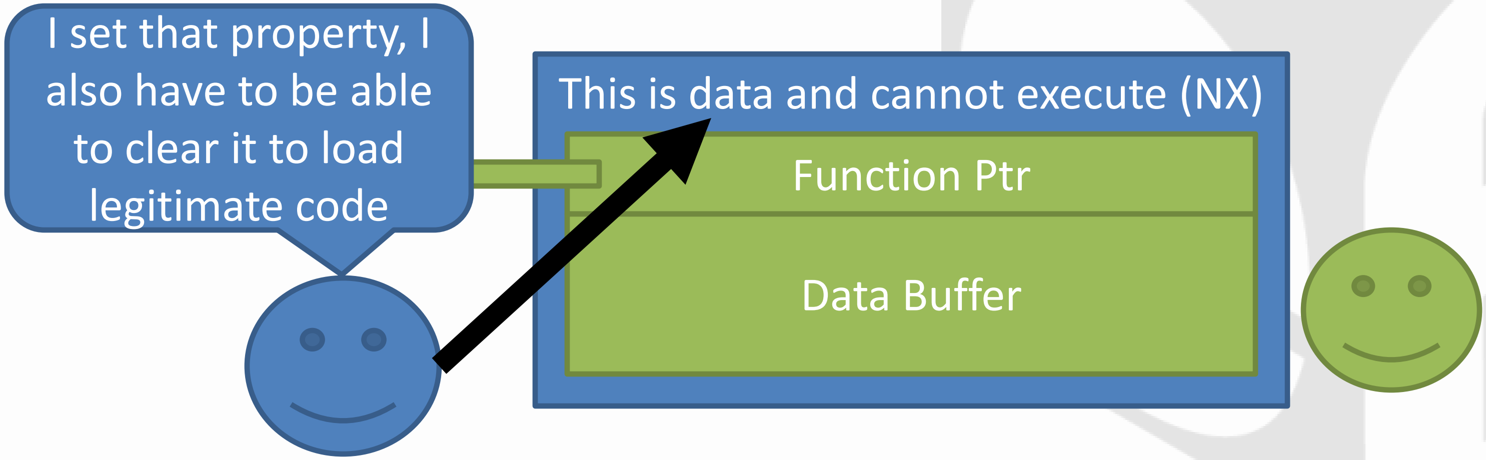
Randomization of the Execution Address Space

Address Space Layout Randomization (ASLR)

- Goal: Prevent Attacker from Exploiting Valid Code Loaded at a Known Address
- Enabling Technology
  - A good source of random numbers is needed that varies on every boot
  - Random numbers can come from a TPM or the CPU's time stamp counter can be used to seed a random number generator

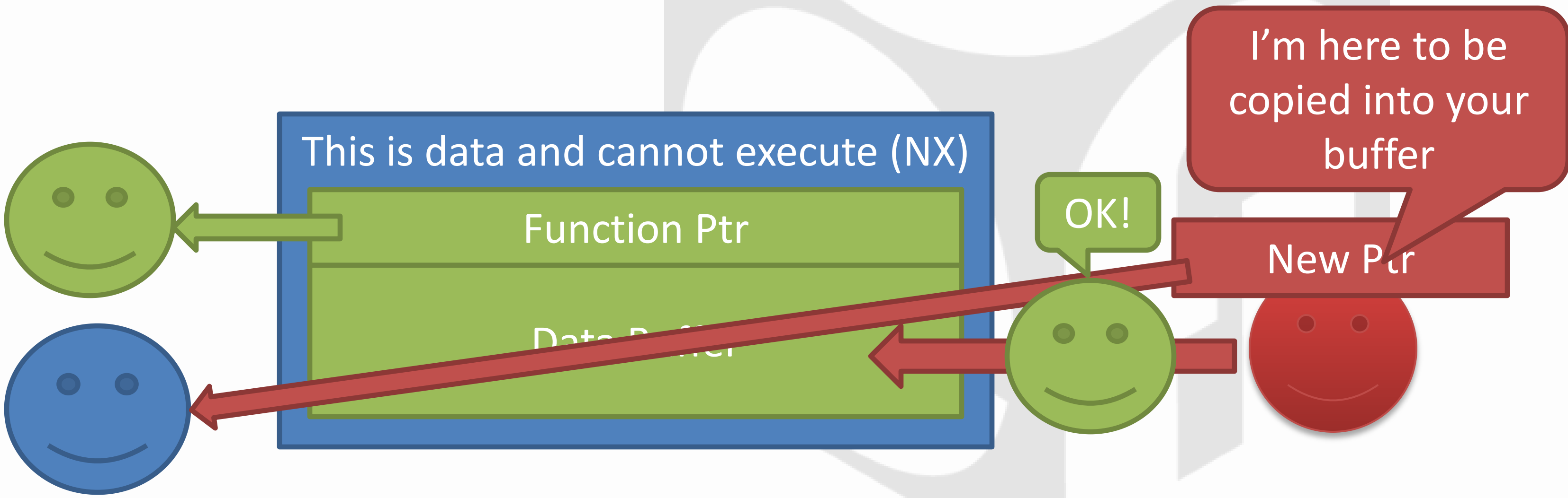
Address Space Layout Randomization (ASLR)

- DXE core contains code to set page table properties, such as NX, as well as to handle page faults



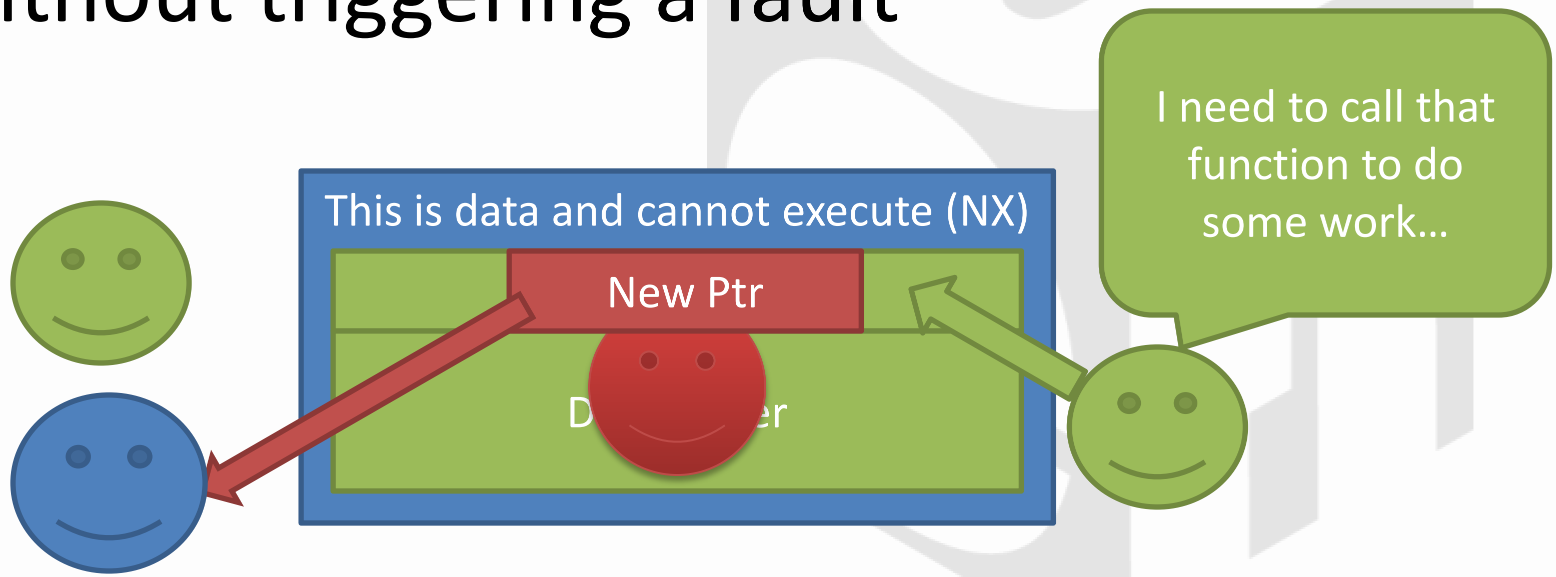
Address Space Layout Randomization (ASLR)

- The attacker once again overflows the buffer, and changes a function pointer



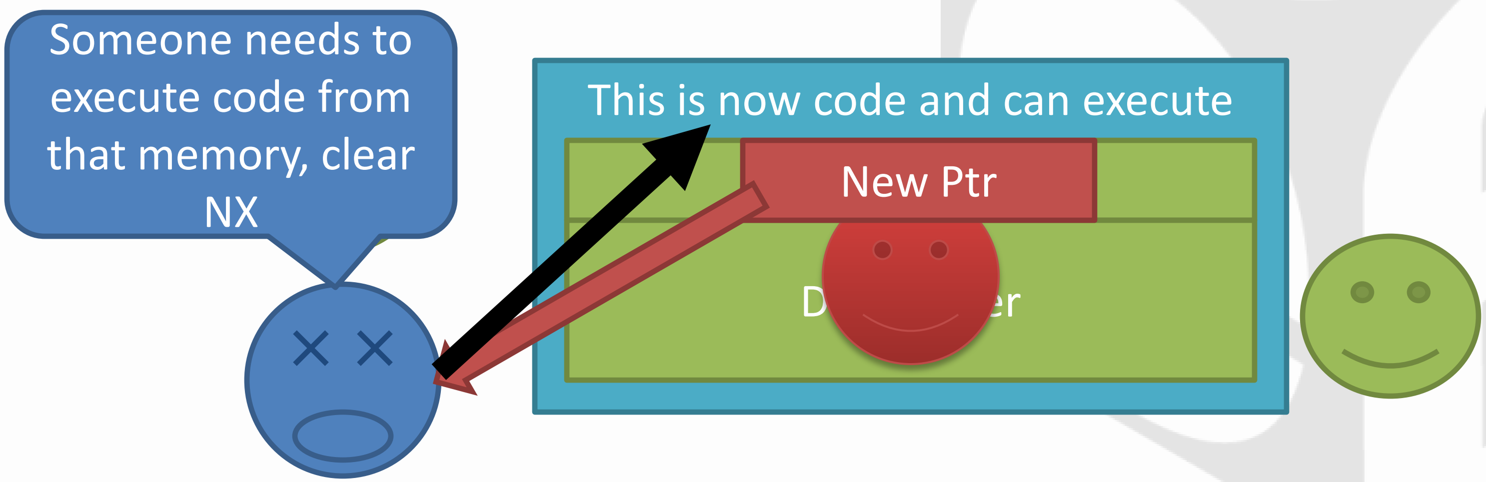
Address Space Layout Randomization (ASLR)

- The altered function pointer is data, so it is referenced to make a call to DXE core without triggering a fault



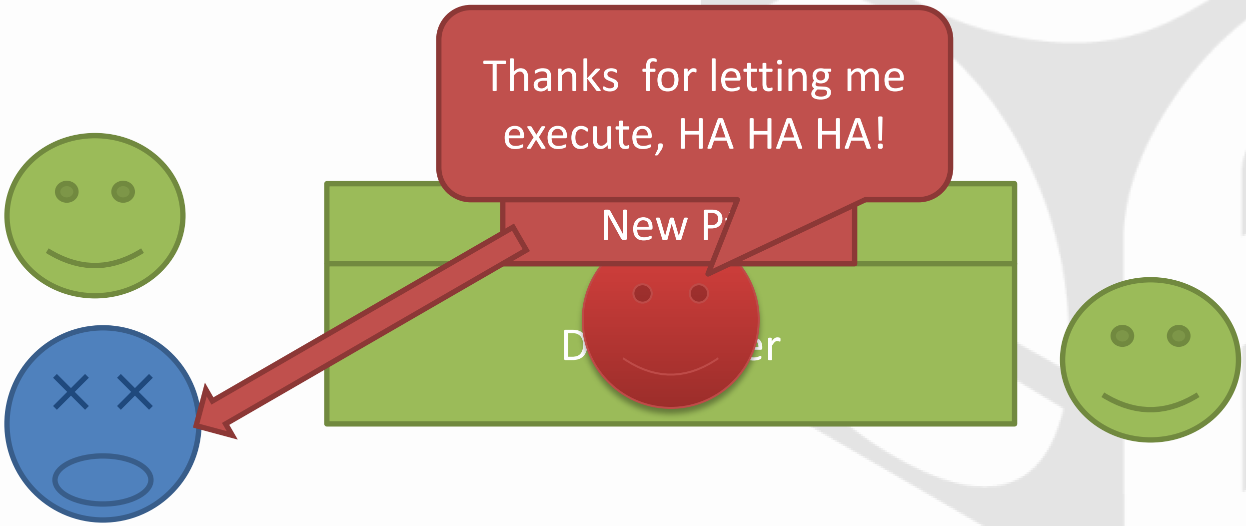
Address Space Layout Randomization (ASLR)

- Legitimate code in DXE core is exploited to disable NX on the memory where the exploit is currently stored



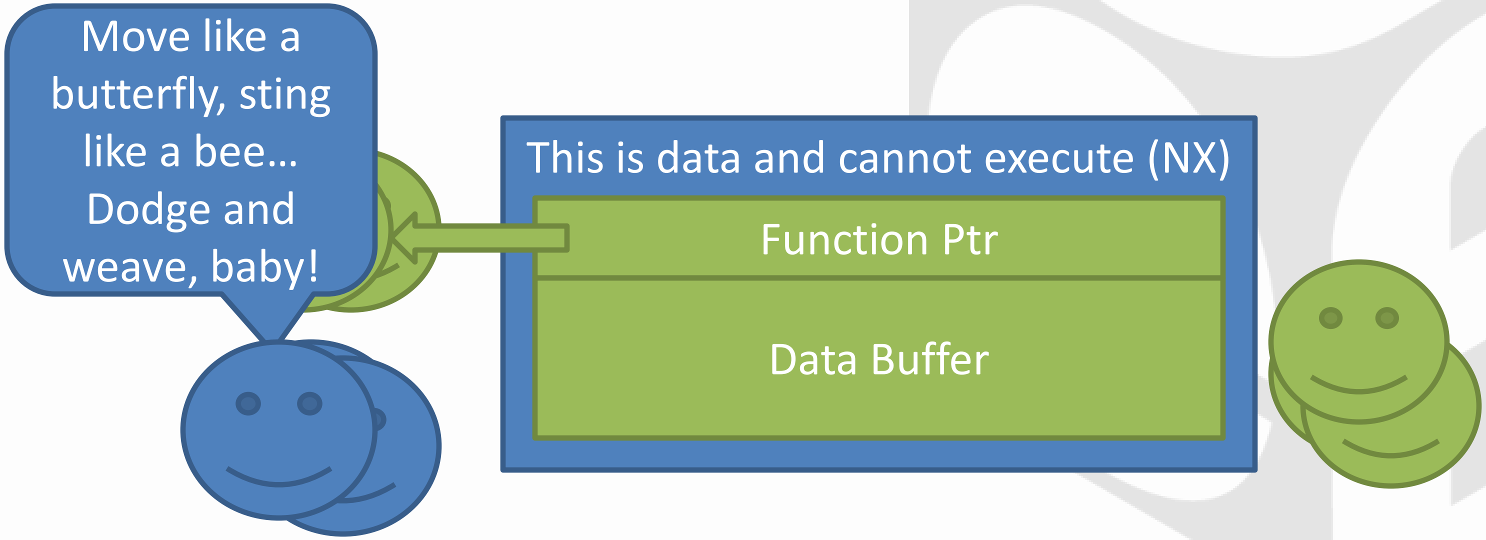
Address Space Layout Randomization (ASLR)

- As a result, the exploit code can now be executed at any time, and NX no longer triggers a page fault



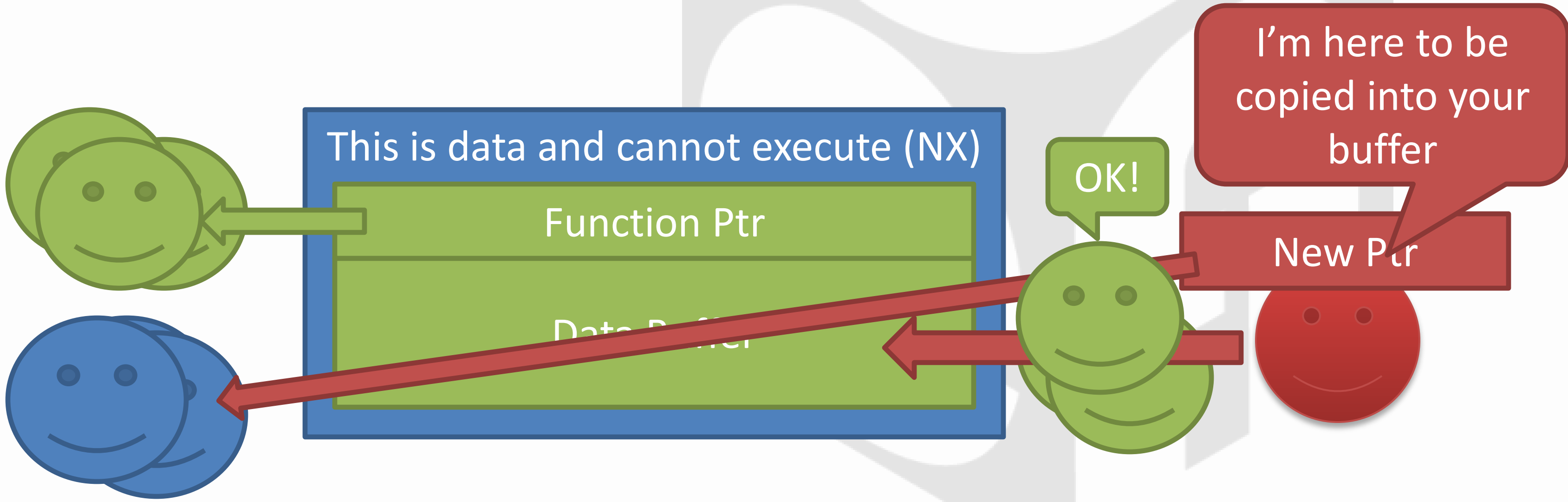
Address Space Layout Randomization (ASLR)

- Address space location randomization causes code to be loaded at different random addresses on every boot



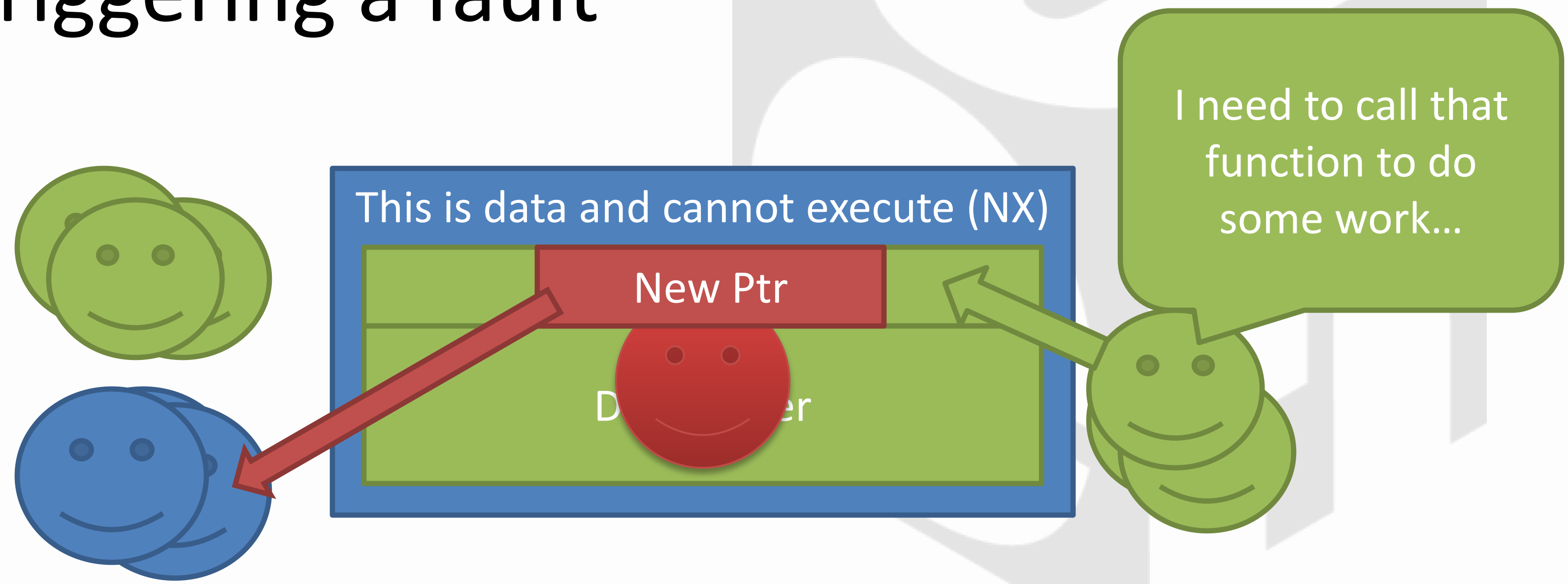
Address Space Layout Randomization (ASLR)

- The exploit can still trick the target into loading it and can change the function pointer to point to a new address



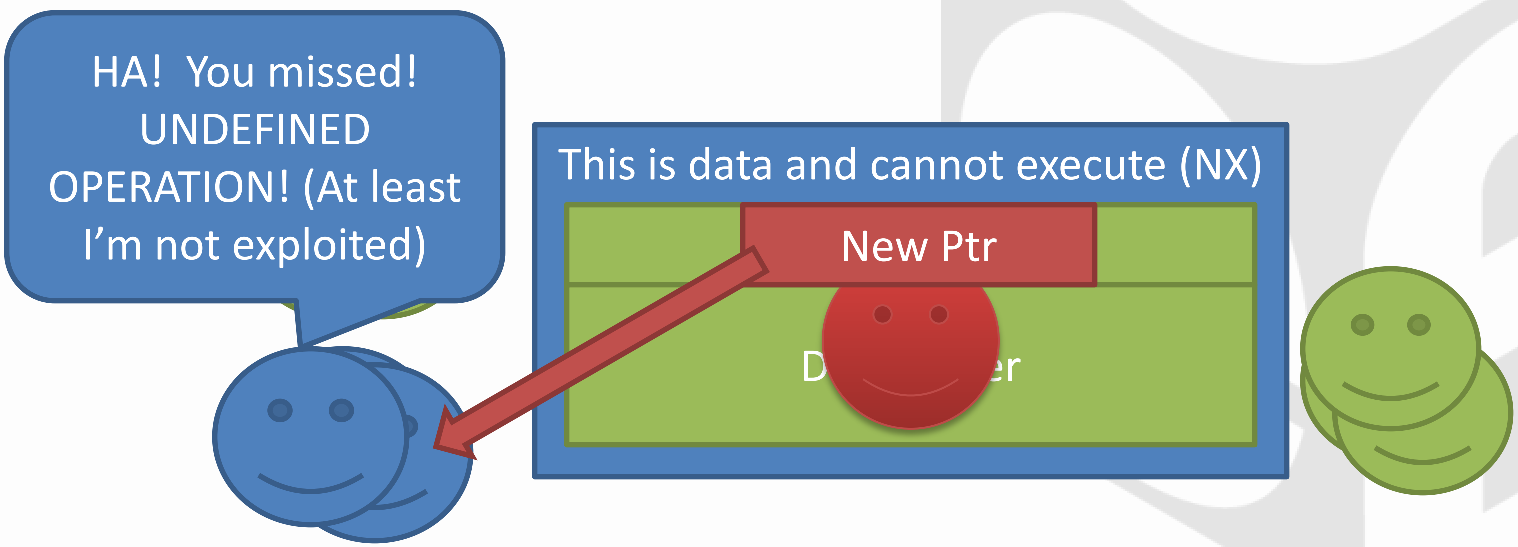
Address Space Layout Randomization (ASLR)

- The altered function pointer is still data, so it is referenced properly without triggering a fault



Address Space Layout Randomization (ASLR)

- But the altered pointer can’t point to code at a known location, because all code is loaded at random addresses



Address Space Layout Randomization (ASLR)

- Phoenix ASLR Implementation
  - New BaseBinSecurityLib MdePkg library
    - Contains code underlying support for /GS, /RTC
    - Also contains random number generation for ASLR
    - Used by PE loaders to randomize load addresses
    - May be replaced to change random number source
  - Enabling PcdAddressSpaceLocRandomizationSupport causes PE loaders to randomize addresses
  - Minimum code alignment can be defined using PcdASLRDefaultAlignmentShift; normally alignment requirements comes directly from PE file format

Thanks for attending the UEFI Spring Plugfest 2012

For more information on the Unified EFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by