PRACTICAL RETURN-ORIENTED PROGRAMMING

RSACONFERENCE2010

SECURITY DECODED

Dino Dai Zovi
Endgame Systems

Session ID: RR-304
Session Classification: Advanced

WHY AM I HERE?

- Show the practical applications of return-oriented programming to exploitation of memory corruption vulnerabilities
  - "Preventing the introduction of malicious *code* is not enough to prevent the execution of malicious *computations*"[1]
- Demonstrate that while exploit mitigations make exploitation of many vulnerabilities impossible or more difficult, they do not prevent all exploitation
  - Modern computing needs more isolation and separation between components (privilege reduction, sandboxing, virtualization)
  - The user-separation security model of modern OS is not ideally suited to the single-user system
  - Why do all of my applications have access to read and write *all of my data*?

1. "The Geometry of Innocent Flesh on the Bone: Return-Into-Libc without Function Calls (on the x86)", Hovav Shacham (ACM CCS 2007)

AGENDA

Current State of Exploitation

Return-Oriented Programming

Bypassing DEP

Exploiting iPhone

Current State of Exploitation

A BRIEF HISTORY OF MEMORY CORRUPTION

- Morris Worm (November 1988)
  - Exploited a stack buffer overflow in BSD in.fingerd on VAX
  - Payload issued execve("/bin/sh", 0, 0) system call directly
- Thomas Lopatic publishes remote stack buffer overflow exploit against NCSA HTTPD for HP-PA (February 1995)
- "Smashing the Stack for Fun and Profit" by Aleph One published in Phrack 49 (August 1996)
- Researchers find and exploit stack buffer overflows in a variety of Unix software throughout the late 90's
- Many security experts thought (incorrectly) that stack buffer overflows were the only exploitable problem

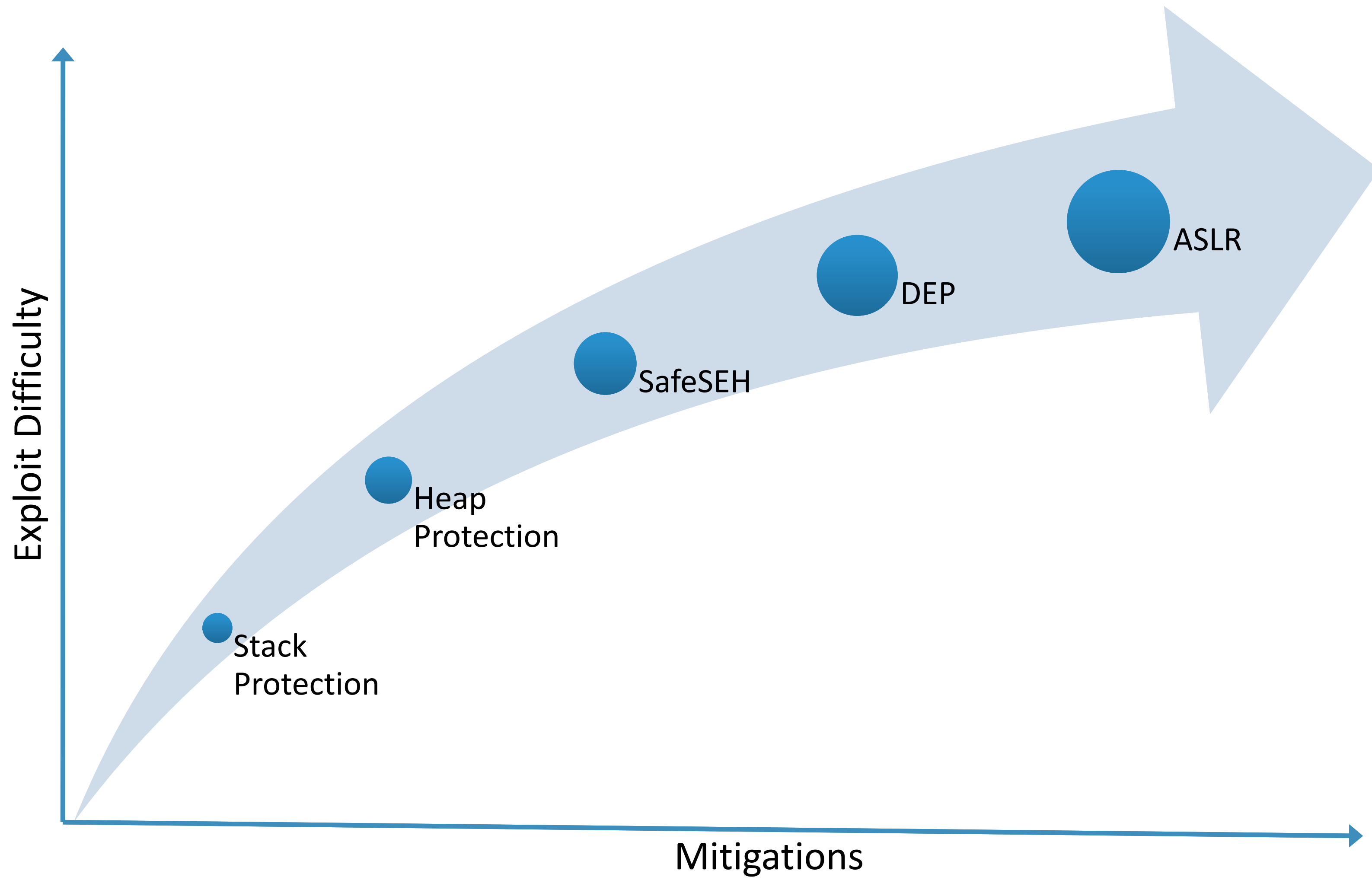
A BRIEF HISTORY OF MEMORY CORRUPTION

- "JPEG COM Marker Processing Vulnerability in Netscape Browsers" by Solar Designer (July 2000)
  - Demonstrates exploitation of heap buffer overflows by overwriting heap free block next/previous linked list pointers
- Apache/IIS Chunked-Encoding Vulnerabilities demonstrate exploitation of integer overflow vulnerabilities
  - Integer overflow => stack of heap memory corruption
- In early 2000's, worm authors took published exploits and unleashed worms that caused widespread damage
  - Exploited stack buffer overflow vulnerabilities in Microsoft operating systems
  - Results in Bill Gates' "Trustworthy Computing" memo
- Microsoft's Secure Development Lifecycle (SDL) combines secure coding, auditing, and **exploit mitigation**

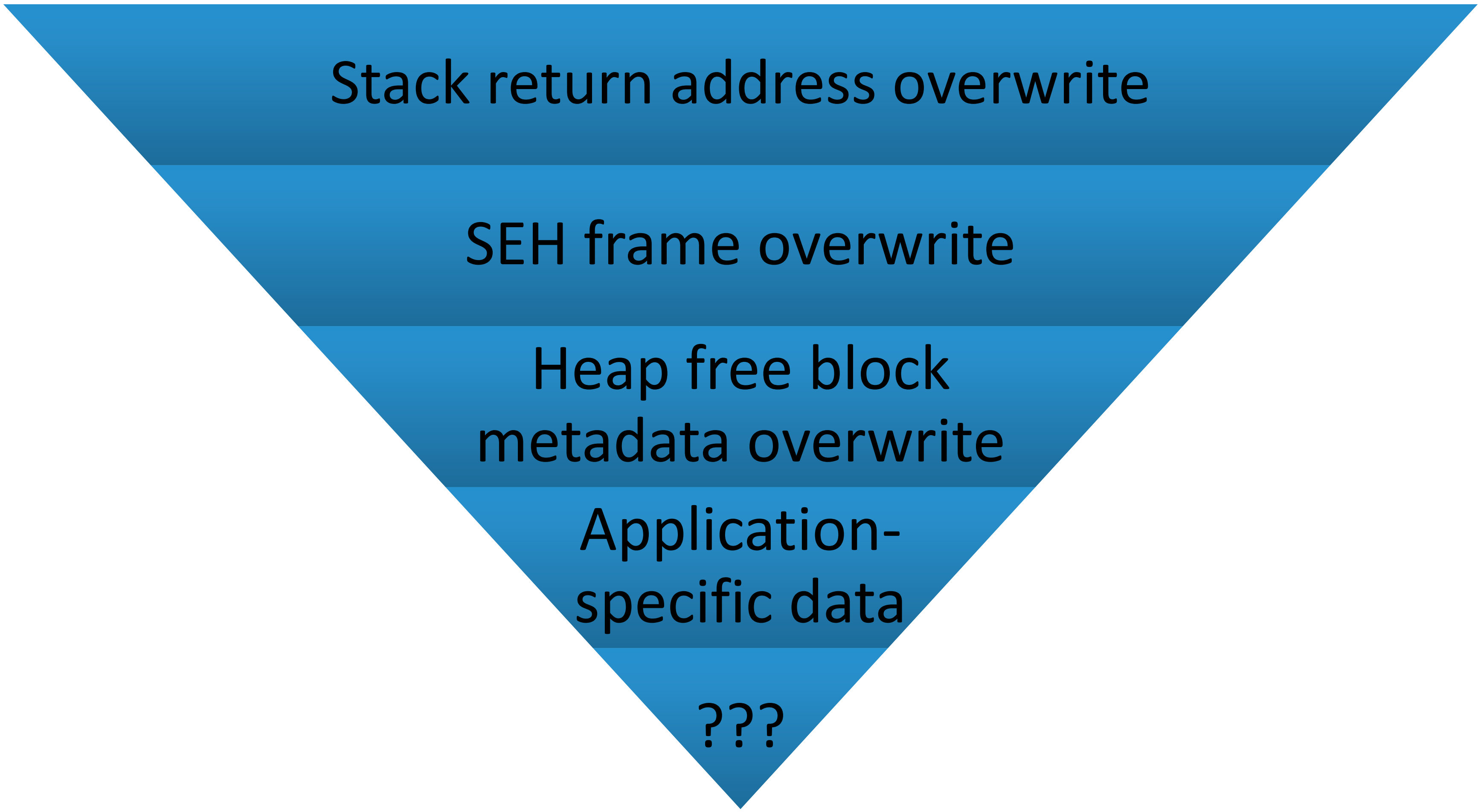
EXPLOIT MITIGATION

- Patching every security vulnerability and writing 100% bug-free code is impossible
  - Exploit mitigations acknowledge this and attempt to make exploitation of remaining vulnerabilities impossible or at least more difficult
- Windows XP SP2 was the first widespread operating system to incorporate exploit mitigations
  - Protected stack metadata (Visual Studio compiler /GS flag)
  - Protected heap metadata (RtlHeap Safe Unlinking)
  - SafeSEH (compile-time exception handler registration)
  - Software, Hardware-enforced Data Execution Prevention (DEP)
- Windows Vista implements Address Space Layout Randomization (ASLR)
  - Invented by and first implemented in PaX project for Linux

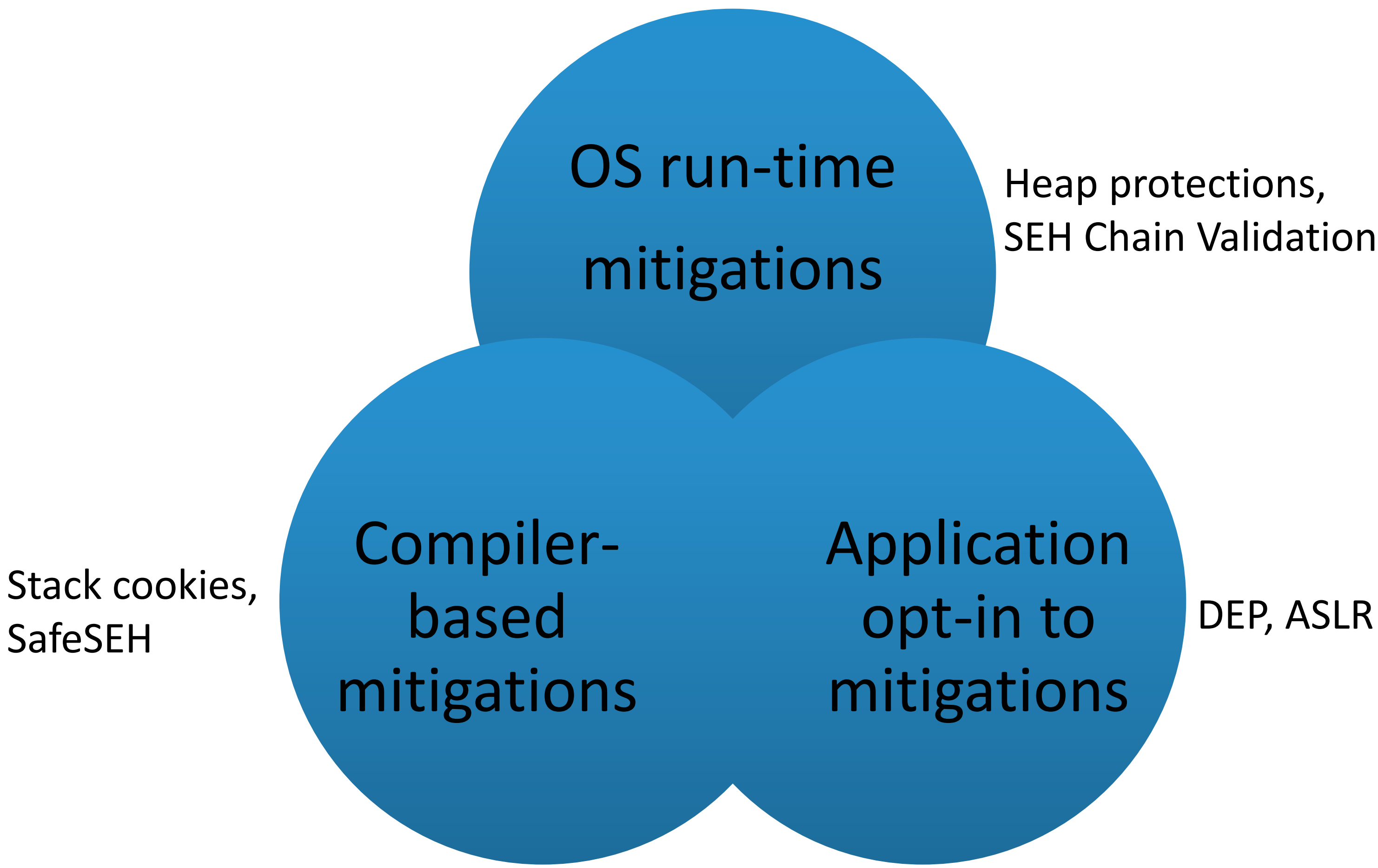
MITIGATIONS MAKING EXPLOITATION HARDER



EXPLOIT TECHNIQUES RENDERED INEFFECTIVE



MITIGATIONS REQUIRES OS, COMPILER, AND APPLICATION PARTICIPATION AND ARE ADDITIVE



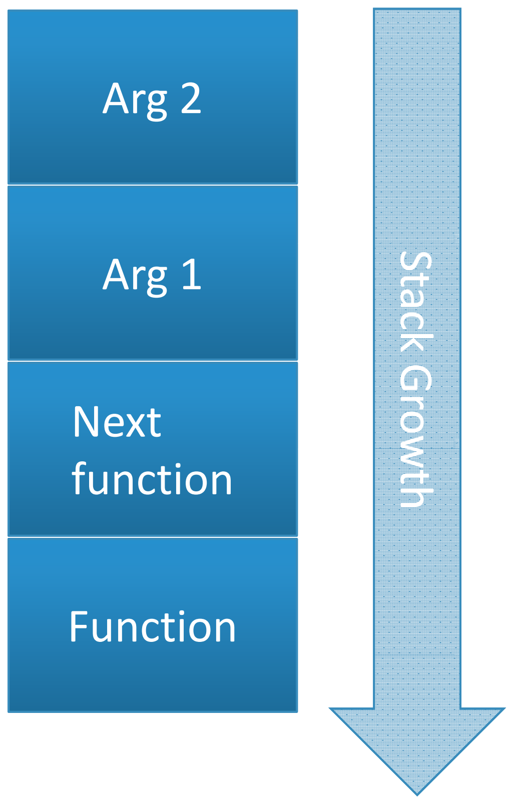
WHAT MITIGATIONS ARE ACTIVE IN MY APP?

- It is difficult for even a knowledgeable user to determine which mitigations are present in their applications
  - Is the application compiled with stack protection?
  - Is the application compiled with SafeSEH?
  - Do all executable modules opt-in to DEP (NXCOMPAT) and ASLR (DYNAMICBASE)?
  - Is the process running with DEP and/or Permanent DEP?
- Internet Explorer 8 on Windows 7 is 100% safe, right?
  - IE8 on Windows 7 uses the complete suite of exploit mitigations
  - … as long as you don't install any 3rd-party plugins or ActiveX controls
- What about Adobe Reader?
  - You don't want to know…

Return-Oriented Programming

RETURN-TO-LIBC

- Return-to-libc (ret2libc)
  - An attack against non-executable memory segments (DEP, W^X, etc)
  - Instead of overwriting return address to return into shellcode, return into a loaded library to simulate a function call
  - Data from attacker's controlled buffer on stack are used as the function's arguments
  - i.e. call system(*cmd*)

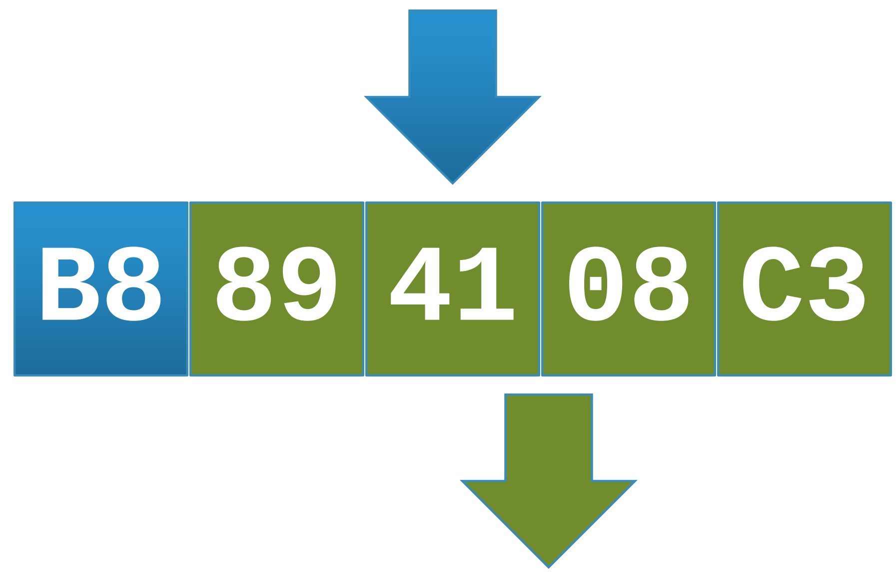


"Getting around non-executable stack (and fix)", Solar Designer (BUGTRAQ, August 1997)

RETURN-ORIENTED PROGRAMMING

- Instead of returning to functions, return to instruction sequences followed by a return instruction
- Can return into middle of existing instructions to simulate different instructions
- All we need are useable byte sequences anywhere in executable memory pages

```
mov eax, 0xc3084189
```

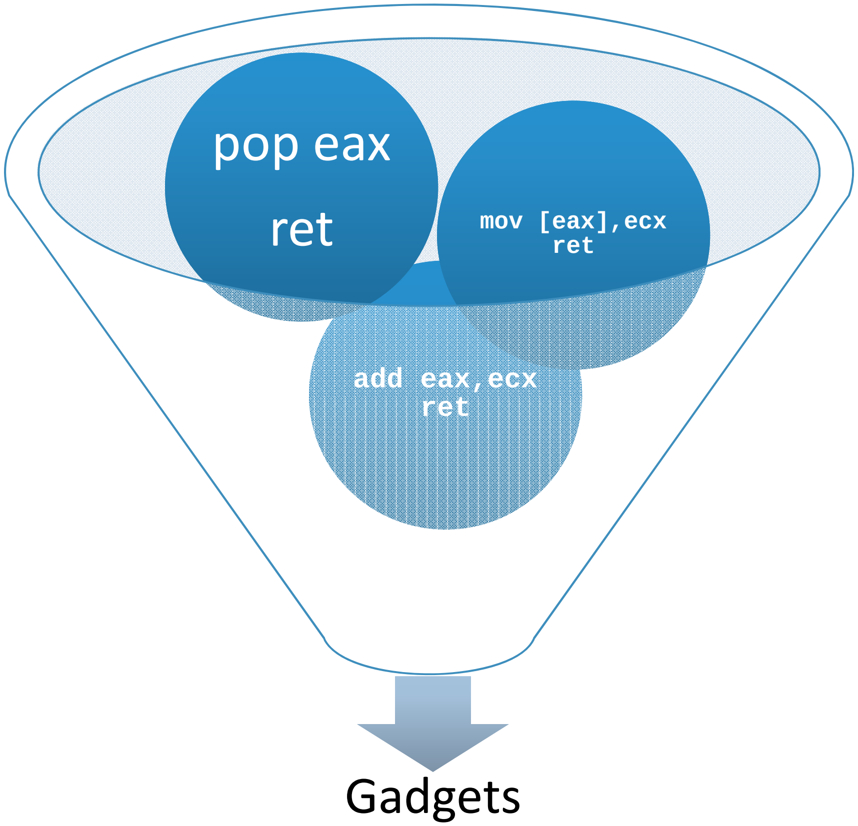



```
mov [ecx+8], eax
ret
```

"The Geometry of Innocent Flesh on the Bone: Return-Into-Libc without Function Calls (on the x86)", Hovav Shacham (ACM CCS 2007)

RETURN-ORIENTED PROGRAMMING

- Various instruction sequences can be combined to form *gadgets*
- Gadgets perform higher-level actions
  - Write specific 32-bit value to specific memory location
  - Add/sub/and/or/xor value at memory location with immediate value
  - Call function in shared library



EXAMPLE GADGET

```
pop eax
ret
```

+

```
pop ecx
ret
```

+

```
mov [ecx],eax
ret
```

=

STORE IMMEDIATE VALUE

GENERATING A RETURN-ORIENTED PROGRAM

- Scan executable memory regions of common shared libraries for useful instruction sequences followed by return instructions
- Chain returns to identified sequences to form all of the desired gadgets from a Turing-complete gadget catalog
- The gadgets can be used as a backend to a C compiler
  - See Hovav Shacham's paper for details on GCC compiler backend and demonstration of return-oriented quicksort
- **Preventing the introduction of malicious *code* is not enough to prevent the execution of malicious *computations***

Bypassing DEP

DATA EXECUTION PREVENTION

- DEP uses the NX/XD bit of x86 processors to enforce the non-execution of memory pages without PROT_EXEC permission
  - On non-PAE processors/kernels, READ => EXEC
  - PaX project cleverly simulated NX by desynchronizing instruction and data TLBs
- Requires every module in the process (EXE and DLLs) to be compiled with **/NXCOMPAT** flag
- DEP can be turned off dynamically for the whole process by calling (or returning into) NtSetInformationProcess()[1]
- XP SP3, Vista SP1, and Windows 7 support "Permanent DEP" that once enabled, cannot be disabled at run-time

1. "Bypassing Windows Hardware-Enforced Data Execution Prevention", skape and Skywing (Uninformed Journal, October 2005)

RETURN-ORIENTED EXPLOITS

- First, attacker must cause stack pointer to point into attacker-controlled data
  - This comes for free in a stack buffer overflow
  - Exploiting other vulnerabilities (i.e. heap overflows) requires using a *stack pivot* sequence to point ESP into attacker data
    - ```
      mov esp, eax
      ret
      ```
    - ```
      xchg eax, esp
      ret
      ```
    - ```
      add esp, <some amount>
      ret
      ```
- Attacker-controlled data contains a return-oriented exploit payload
  - These payloads may be 100% return-oriented programming or simply act as a temporary payload stage that enables subsequent execution of a traditional machine-code payload
      ```

RETURN-ORIENTED PAYLOAD STAGE

- HEAP_CREATE_ENABLE_EXECUTE method[1]

```
hHeap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0);
pfnPayload = HeapAlloc(hHeap, 0, dwPayloadLength);
CopyMemory(pfnPayload, ESP+offset, dwPayloadLength);
(*pfnPayload)();
```

- VirtualAlloc() method

```
VirtualAlloc(lpAddress, dwPayloadSize, MEM_COMMIT,
  PAGE_EXECUTE_READWRITE);
CopyMemory(lpAddress, ESP+offset, dwPayloadSize);
(*lpAddress)();
```

- VirtualProtect(ESP) method

```
VirtualProtect(ESP+offset & ~(4096 – 1),
  dwPayloadSize, PAGE_EXECUTE_READWRITE);
(*ESP+offset)();
```

1. “DEPLIB”, Pablo Sole (H2HC November 2008)

DO THE MATH

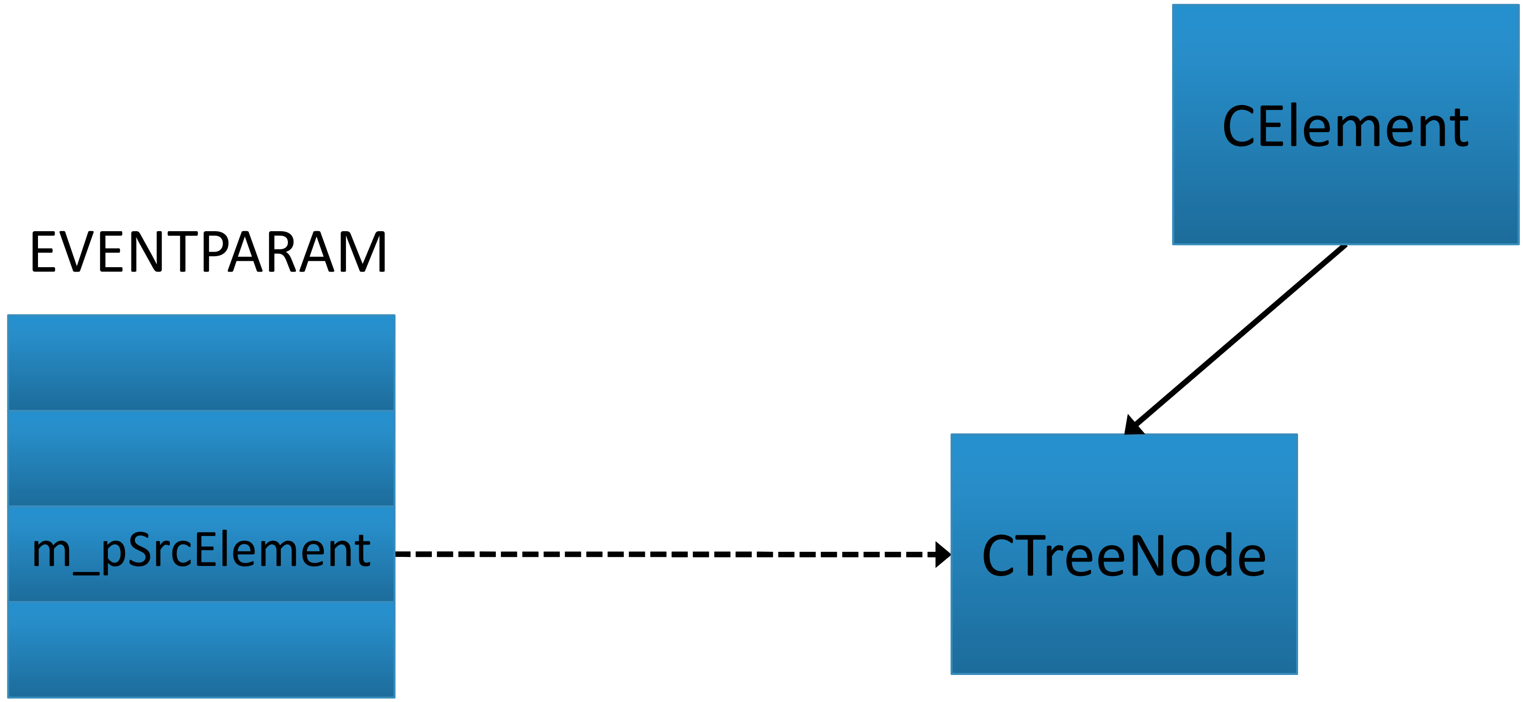
Stack Pivot + Return-Oriented Payload Stage + Traditional Payload = Permanent DEP Bypass Exploit

DEP WITHOUT FULL ASLR IS VERY WEAK SAUCE

- No ASLR:
  - Exploitation requires building a reusable return-oriented payload stage from any common DLL
- One or more modules do not opt-in to ASLR:
  - Exploitation requires building entire return-oriented payload stage from useful instructions found in non-ASLR module(s)
- All executable modules opt-in to ASLR:
  - Exploitation requires exploiting a memory disclosure vulnerability to reveal the load address of one DLL and dynamically building the return-oriented payload stage

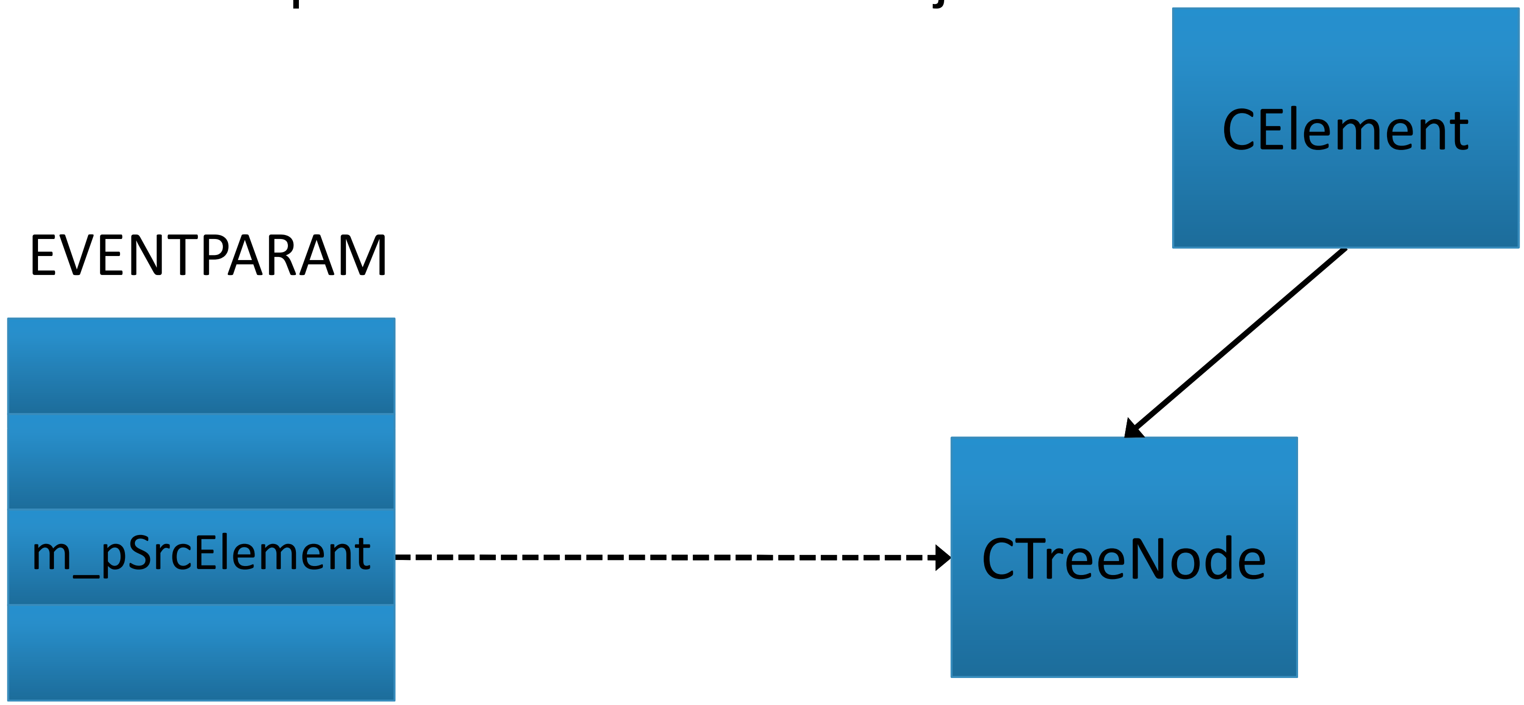
THE "AURORA" IE VULNERABILITY

- EVENTPARAMs copied by createEventObject(oldEvent) don't increment CTreeNode ref count



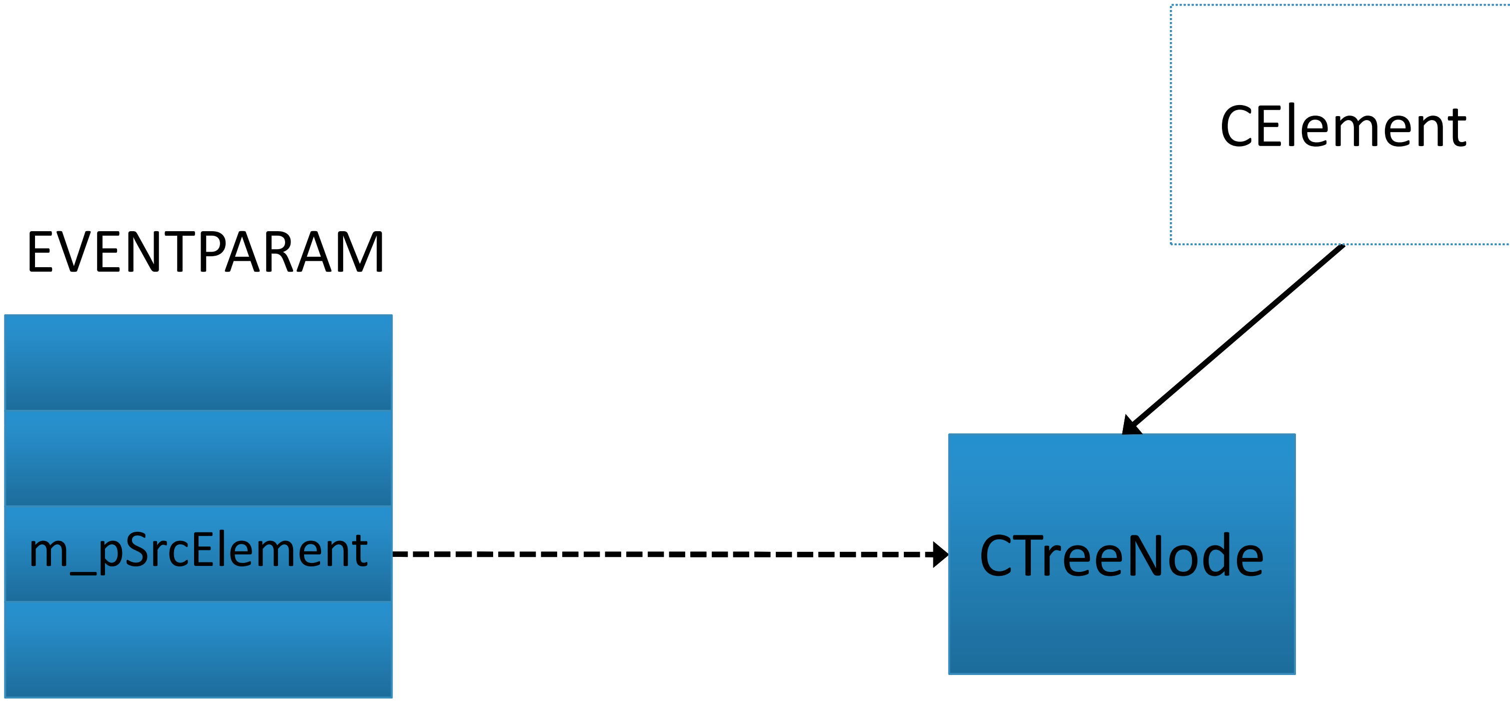
THE "AURORA" IE VULNERABILITY

- EVENTPARAM member variable and CElement member variable both point to CTreeNode object



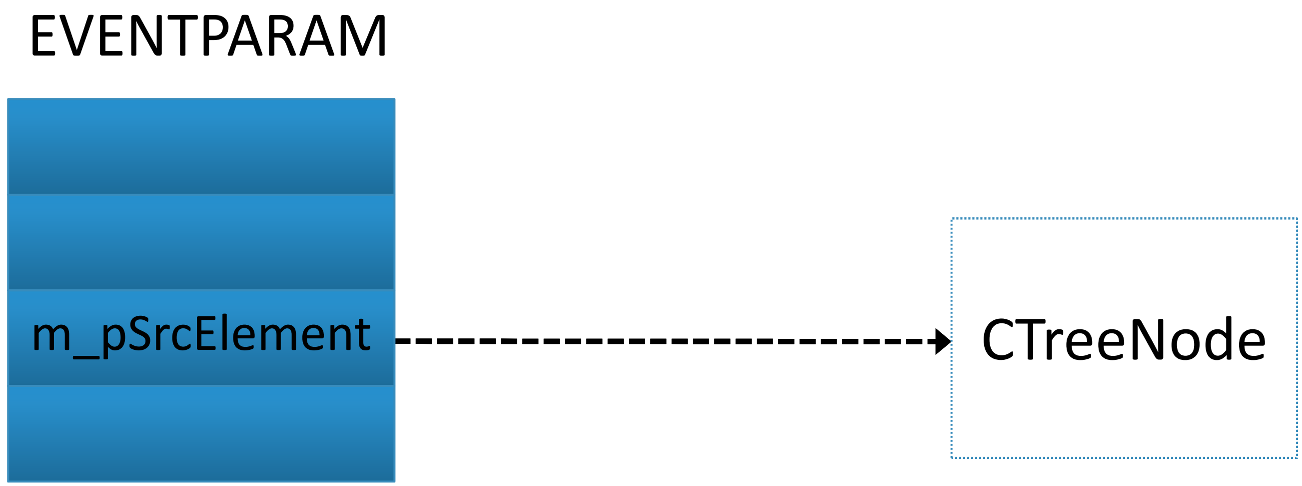
THE "AURORA" IE VULNERABILITY

- When HTML element is removed from DOM, CElement is freed and CTreeNode refcount decremented



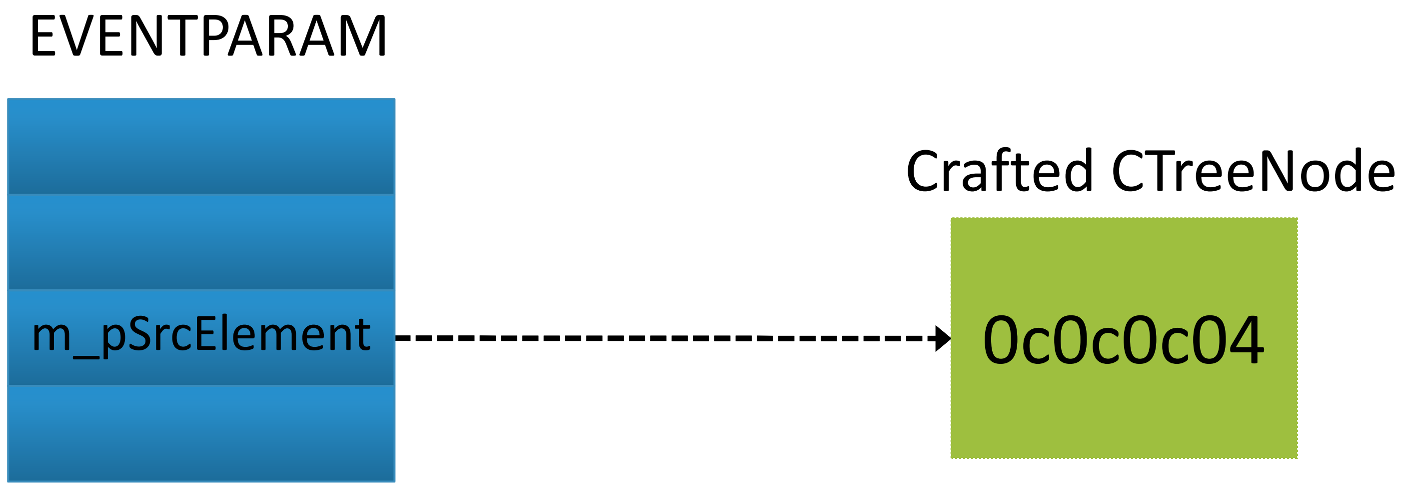
THE “AURORA” IE VULNERABILITY

- If CTreeNode refcount == 0, the object will be freed and EVENTPARAM points free memory



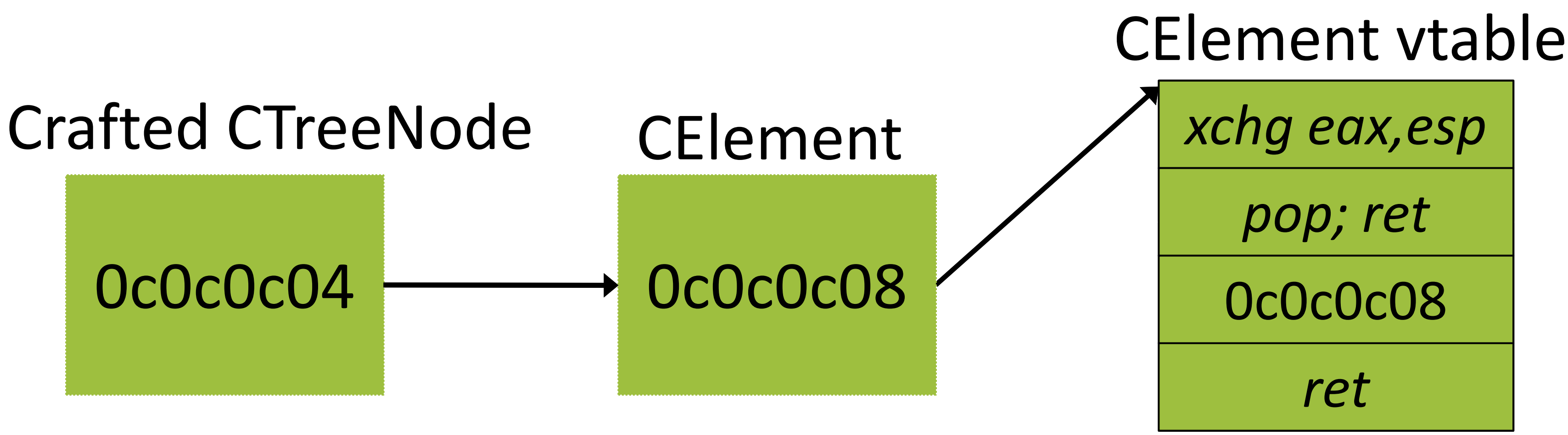
EXPLOITING THE AURORA VULNERABILITY

- Attacker can use controlled heap allocations to replace freed heap block with crafted heap block



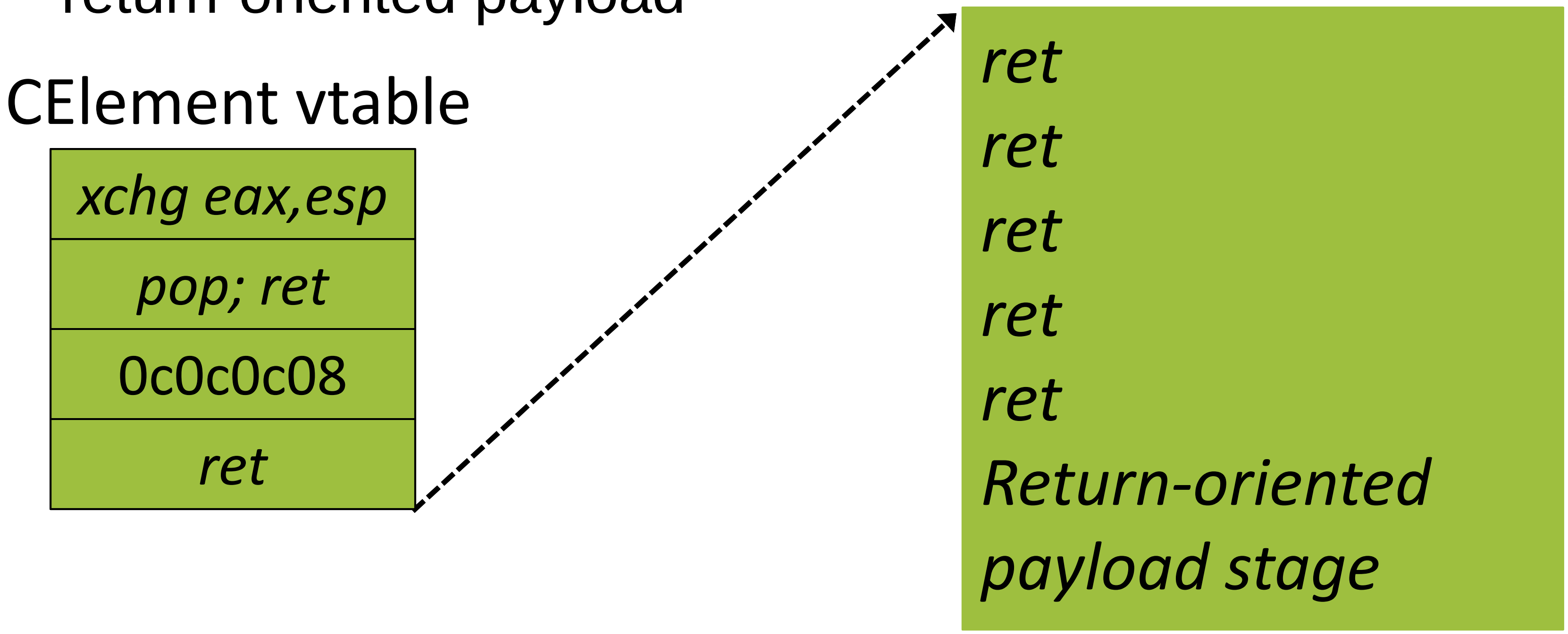
EXPLOITING THE AURORA VULNERABILITY

- The crafted heap block points to a crafted CElement object in the heap spray, which points back to itself as a crafted vtable



EXPLOITING THE AURORA VULNERABILITY

- Attacker triggers virtual function call through crafted CElement vtable, which performs a stack pivot through a return to an ‘xchg eax, esp; ret’ sequence and runs return-oriented payload



Aurora Exploit Demo

Exploiting iPhone

REVIEW OF IPHONE OS SECURITY MECHANISMS

- W^X non-executable memory policy
  - Memory page can't be writable and executable at the same time
- Code-signing enforcement (unless you JailBreak)
  - If a memory page's backing store is not an executable binary signed by Apple, it cannot be marked PROT_EXEC
  - If an executable memory page has been made writable, it cannot later be made executable again
  - Can't execute a binary that has not been signed by Apple
- Sandbox
  - Restricts process behavior at run-time by blocking disallowed system calls
  - Policy against background processes => fork() returns EPERM
- No ASLR

LACK OF ASLR IS WEAK

- Lack of ASLR means that all libraries and frameworks can be used as source material for return-oriented programs
  - dyld
  - libSystem
- Writable .data segments at known locations can be used for scratch data storage
- Return-oriented payloads for iPhone have already been presented[1]
  - Sends contents of file to remote server
  - Still restricted by sandbox policy
- Return-oriented payloads for Windows Mobile on ARM have also recently been developed[2]

1. "Fun and Games with Mac OS X and iPhone Payloads", Miller and Iozzo (BlackHat EU 2009)
2. "Return Oriented Programming for the ARM Architecture", Tim Kornau (Diploma, Dec. 2009)

RETURN-ORIENTED ARM

- ARM (32-bit) vs. Thumb mode (16-bit)
  - Bytes decode to different instructions depending on CPU state
  - BX and BLX instructions can switch modes based on least significant bit of address (0 => ARM, 1 => Thumb)
  - Can also switch modes via LDR/LDM/POP instructions that set PC register
- Scan all executable segments and disassemble as both ARM and Thumb to look for instruction sequences followed by returns (LDM/POP)
- Keep track of CPU state when generating return-oriented program and switch states as necessary in gadgets
- Return-oriented programming also sidesteps exploitation difficulties presented by separate instruction and data caches

ARMV5 GADGETS

- Simulate a function call and store return value
  - ```
    pop {r0, r1, r2, r3, pc}
    …
    pop {r4, r7, pc}
    …
    str r0, [r4]
    pop {r4, r7, pc}
    ```
- Store immediate value to memory
  - ```
    pop {r4, r5, r7, pc}
    …
    str r4, [r5]
    pop {r4, r5, r7, pc}
    ```
- Load value from memory into r0
  - ```
    ldr r0, [r0]
    pop {r7, pc}
    ```
- And so on…
  - For more details, see "Return Oriented Programming for the ARM Architecture", Tim Kornau 2009
    ```

Wrapping Up

OTHER APPLICATIONS OF RETURN-ORIENTED PROGRAMMING

- Embedded processors often have separate instruction and data write-back caches, which make injecting code problematic
  - Return-oriented programming techniques can be used to flush the caches before executing the payload (Dai Zovi, 2003)
- x86-64 ABI requires non-executable (NX) data memory
  - “Borrowed code chunks” exploitation technique (Krahmer 2005)
- Some secure hardware designs keep code in ROM and refuse to execute code from RAM
  - Checkoway et al (Usenix 2008) demonstrated the use of ROP on the Z80-based Sequoia AVC Advantage secure voting machine

CONCLUSIONS

- Return-oriented techniques are increasingly required to exploit vulnerabilities on systems with non-executable data memory protections
- A return-oriented payload stage can be developed to bypass Permanent DEP
- Bypassing DEP under ASLR requires at least one non-ASLR module
- Bypassing DEP under full ASLR requires an executable memory address disclosure vulnerability in addition to memory corruption corruption
- iPhone's code signing enforcement requires attackers to develop fully return-oriented payloads
  - Attacker's actions are still limited by the application sandbox
- **Preventing malicious *actions* is more important than preventing malicious *code***

TAKEAWAYS

- IT Security
  - Malware may eventually use these techniques to exploit DEP-enabled processes
  - Malware analysts must learn how to analyze return-oriented exploit payloads
- Software Vendors
  - Do not assume DEP/ASLR make vulnerabilities non-exploitable
  - Better to assume that all vulnerabilities yield full code execution
  - Restrict the actions that may be performed by application components that parse and handle potentially untrusted data
    - Privilege reduction (i.e. run under Low Integrity on Vista/7)
    - Sandboxing (see Chromium's sandboxed web renderers[1])
    - Virtualization?

1. http://dev.chromium.org/developers/design-documents/sandbox

Questions?