Systems and Internet Infrastructure Security

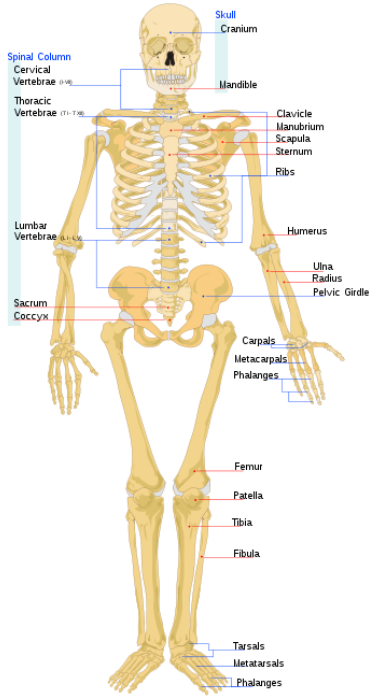
Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

Advanced Systems Security: Program Diversity

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

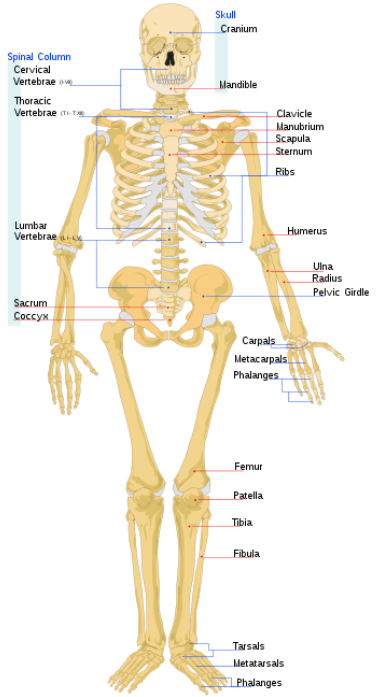
Anatomy of Control Flow Attacks

- Two steps
- First, the attacker changes the control flow of the program
  - In buffer overflow, overwrite the return address on the stack
- Second, the attacker uses this change to run code of their choice
  - In buffer overflow, inject code on stack
  - Or use existing code in ROP attack
- CFI prevents exploitation (incomplete)



Anatomy of Control Flow Attacks

- Two steps
- First, the attacker changes the control flow of the program
  - In buffer overflow, overwrite the return address on the stack
- Second, the attacker uses this change to run code of their choice
  - In buffer overflow, inject code on stack
  - Or use existing code in ROP attack
- Another way to prevent both?



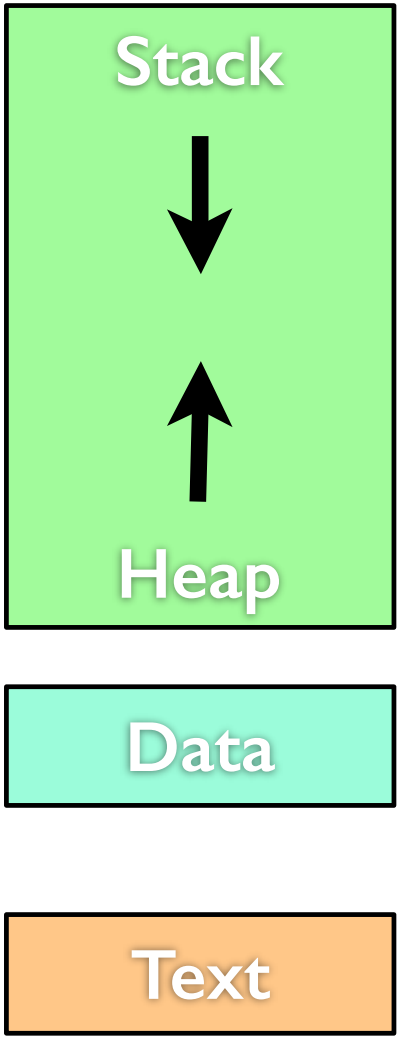
Apply Crypto to Code

PENNSTATE 1855

- Can we randomize the program's execution in such a way that an adversary cannot select gadgets?
- Given a secret key and a program address space, encrypt the address space such that
  - the probability that an adversary can locate a particular instruction (start of gadget or flawed code) is sufficiently low
  - and the program still runs correctly and efficiently
- Called address space randomization

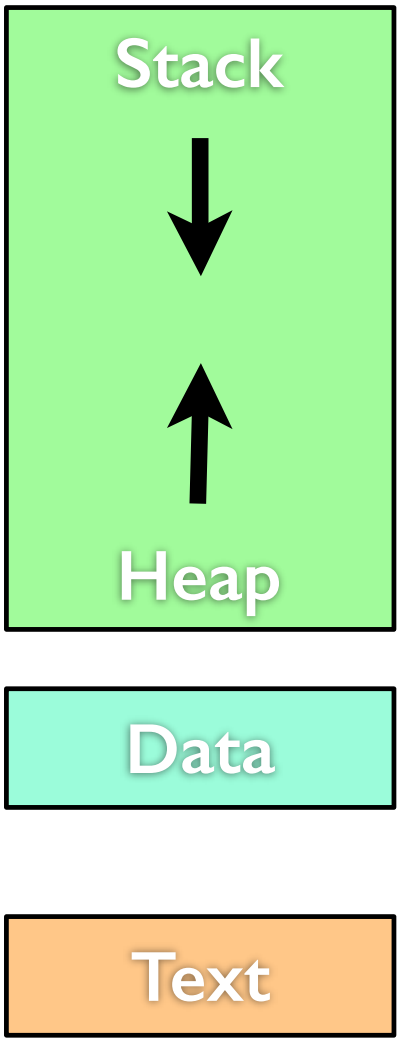
Goal

- Move the code and data so that you cannot predict where gadgets will be
  - What is the best way to make unpredictable?
  - What is the easiest way to make unpredictable?



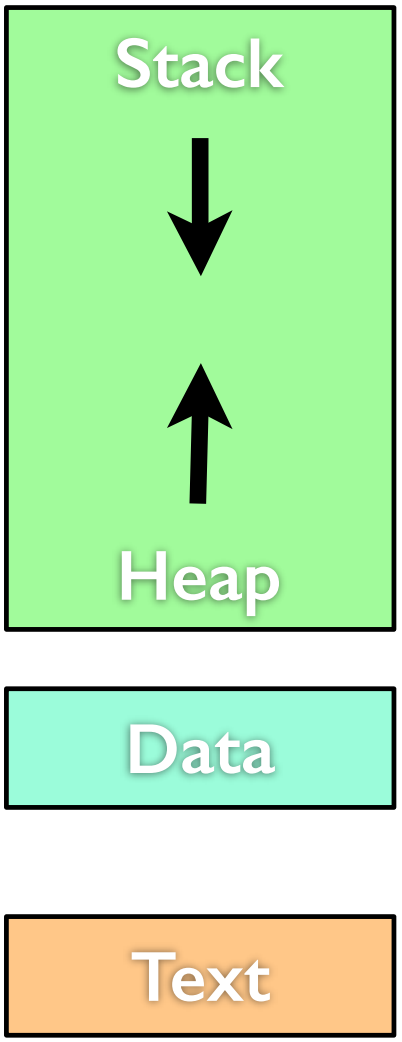
Goal

- Move the code and data so that you cannot predict where gadgets will be
  - What is the best way to make unpredictable?
    - Randomize code and data location for each instruction and variable
  - What is the easiest way to make unpredictable?



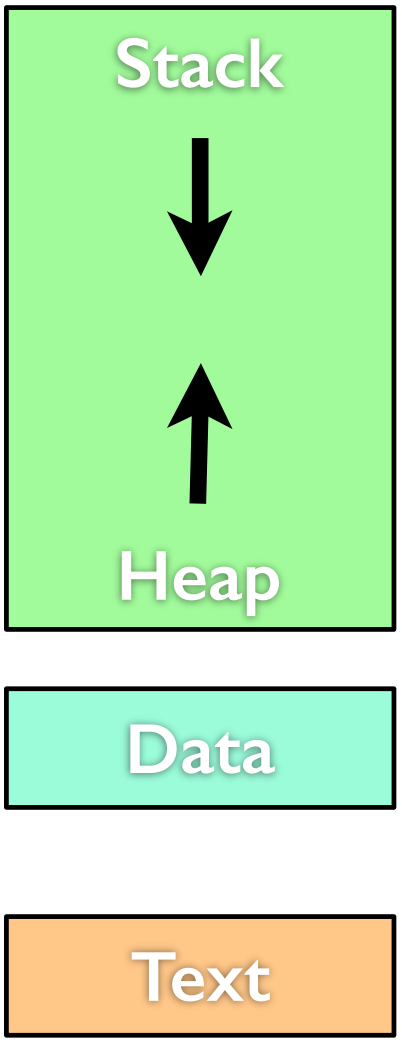
Goal

- Move the code and data so that you cannot predict where gadgets will be
  - What is the best way to make unpredictable?
    - Randomize code and data location for each instruction and variable
  - What is the easiest way to make unpredictable?
    - Just move the base address of the segment
    - Called Address Space Layout Randomization



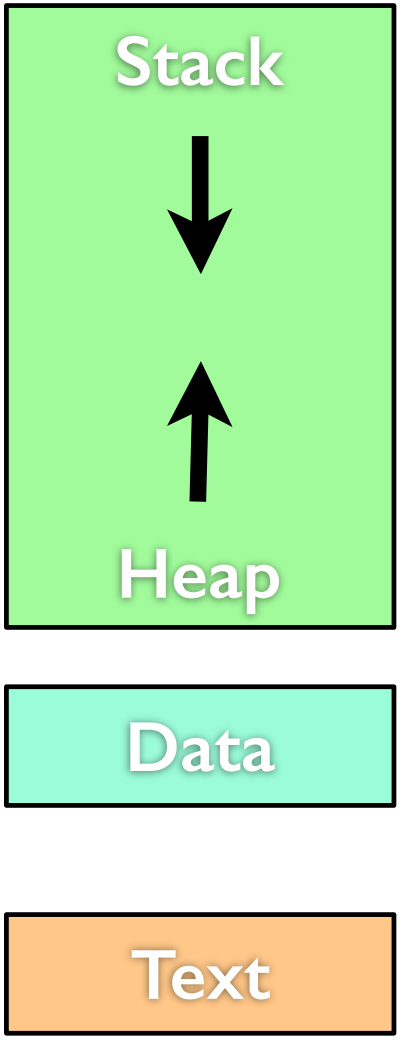
ASLR Impact

- How does it prevent exploitation of attacks?
- Suppose you find a buffer overflow flaw
  - You insert shellcode onto the stack
  - And jump to the stack address
- With ASLR on the stack segment
  - Cannot predict the target stack address
  - Can you overflow return address?



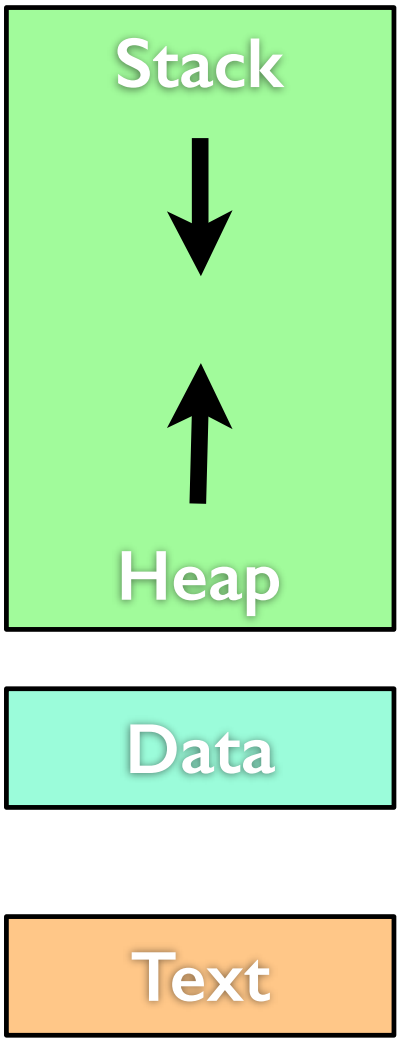
ASLR Impact

- How does it prevent finding of attacks?
- Suppose you find a heap overflow flaw
  - You want to modify a function pointer
    - At known offset – oops, still works
    - At unknown offset – cannot predict
- With ASLR on the heap segment
  - Cannot predict absolute addresses
  - Why not?



ASLR Impact

- How does it prevent exploitation of attacks?
- Suppose you find a buffer overflow flaw
  - You launch an ROP attack
  - And jump to the code address of first gadget
- With ASLR on the code segment
  - Cannot predict the target code address
  - Why not?



Relationship to DEP

PENNSTATE 1855

- ASLR is a complementary defense relative to DEP/CFI
- DEP restricts what may be executed as code
- CFI restricts control flow paths that may be executed
- ASLR prevents some memory attacks
  - Absolute writes over memory (e.g., global)
  - Relative writes are still possible
- Also, ASLR makes gadgets harder to find

What Makes Good ASLR?

PENNSTATE 1855

- Symantec paper investigates ASLR in Windows
- What are choices regarding ASLR use?

What Makes Good ASLR?

PENNSTATE 1855

- Symantec paper investigates ASLR in Windows
- What are choices regarding ASLR use?
  - How many offsets?
    - Limits?
    - Impact on libraries?

What Makes Good ASLR?

PENNSTATE 1855

- Symantec paper investigates ASLR in Windows
- What are choices regarding ASLR use?
  - How many offsets?
    - Limits?
    - Impact on libraries?
  - Distribution?
    - Impact of an uneven distribution?

What Makes Good ASLR?

PENNSTATE 1855

- Symantec paper investigates ASLR in Windows
- What are choices regarding ASLR use?
  - How many offsets?
    - Limits?
    - Impact on libraries?
  - Distribution?
    - Impact of an uneven distribution?
  - Sequence?
    - What should the next offset be?

Risks

PENNSTATE 1855

- How would you attack ASLR?

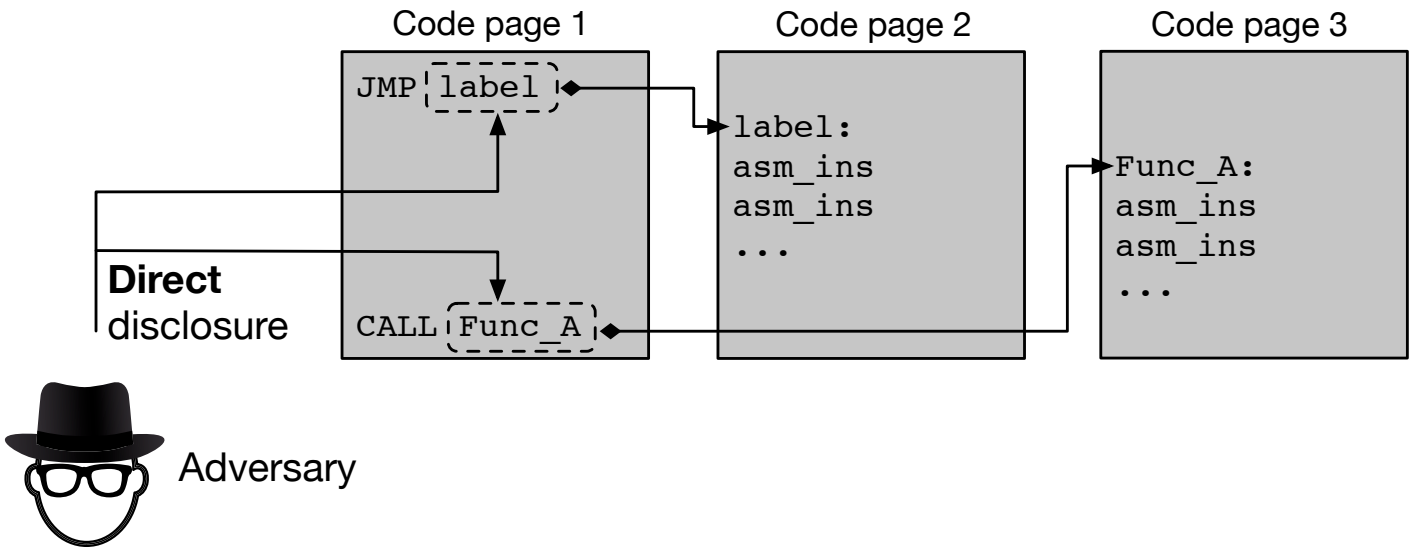
Memory Disclosure Attacks

PENNSTATE 1855

- What is the risk to ASLR?
  - Memory Disclosure
- Consider a buffer overread
  - E.g., Heartbleed
- Instead of reading a key value
  - What would you read to attack ASLR?

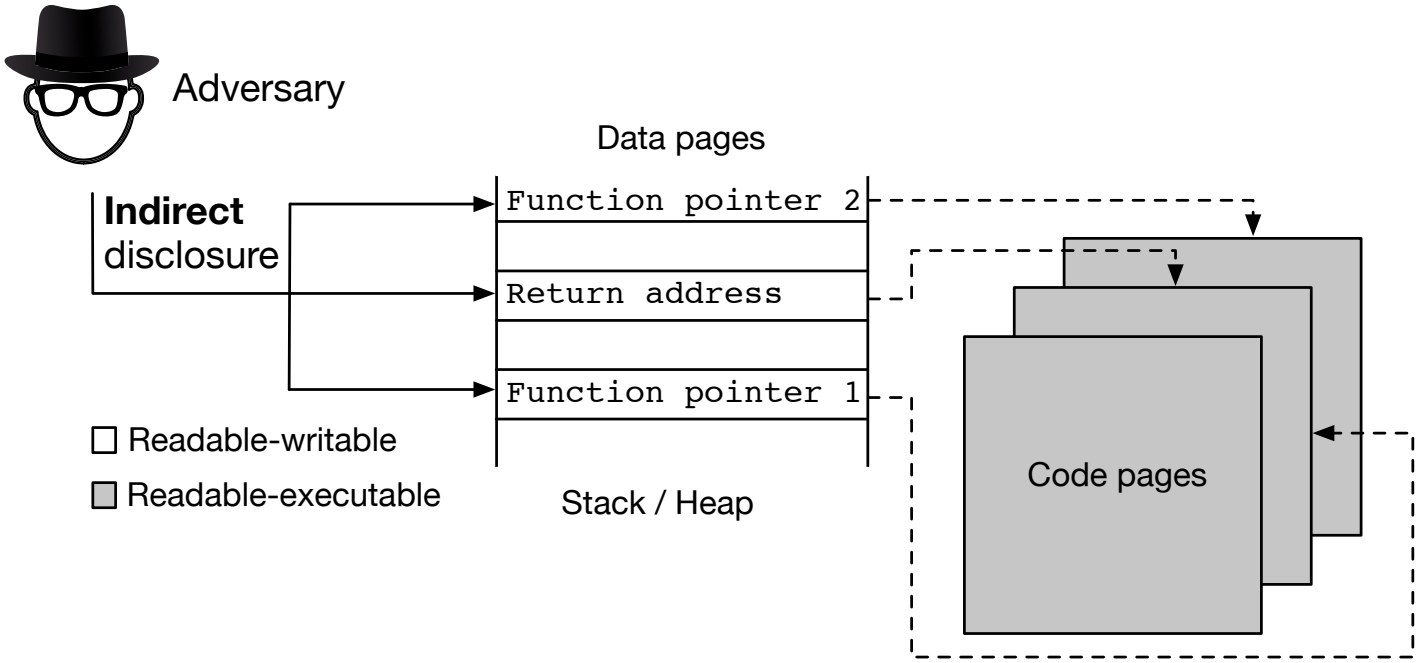
Direct Disclosure

- Adversary is able to directly read code pointers from code pages



Indirect Disclosure

- Adversary harvests code pointers stored on the data pages of the application that are necessarily readable



Fine-Grained Randomization

PENNSTATE 1855

- Can we make harvesting more difficult with fine-grained randomization of code and data?
  - Yes, but at a significant cost
    - E.g., cache locality is completely lost
- See, P. Larsen, A. Homescu, S. Brunthaler, and M. Franz. SoK: Automated software diversity. In 35th IEEE Symposium on Security and Privacy, S&P, 2014.

Other Alternatives

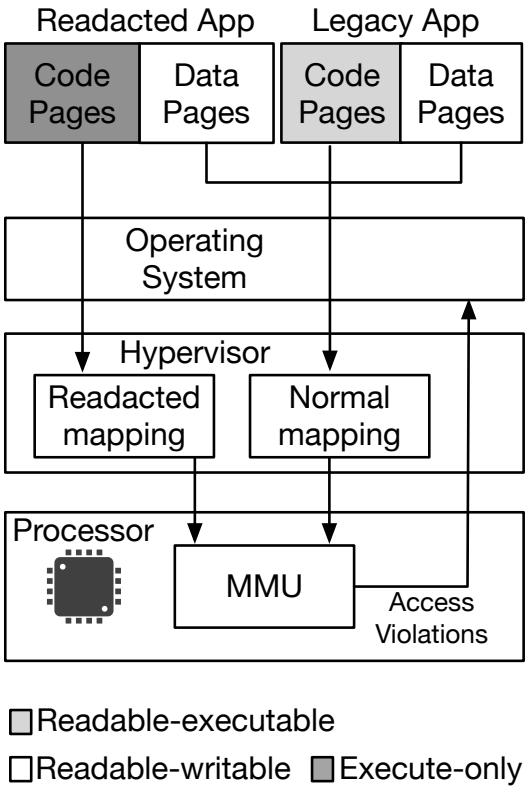
- Prevent read access to code pages that are not currently executing
  - Prevents only direct disclosures
- Adversary can bypass this countermeasures using indirect disclosures
  - E.g., virtual tables for C++
  - Doing disassembly on the fly

Readactor Solution

- To prevent attacks based on direct disclosure,
  - Leverage virtualization capabilities in commodity x86 processors to map code pages with execute-only permissions at all times
- To prevent attacks based on indirect disclosure,
  - Hide the targets of all function pointers and return addresses
- Use compiler-based solution to obtain more precise control-flow information for indirect targets

Memory Management

- Readacted applications use virtualization hardware to map pages differently than legacy applications
  - Can work together, however



Memory Management

- See difference between how code and data pages are mapped
  - Why does this prevent direct disclosure?

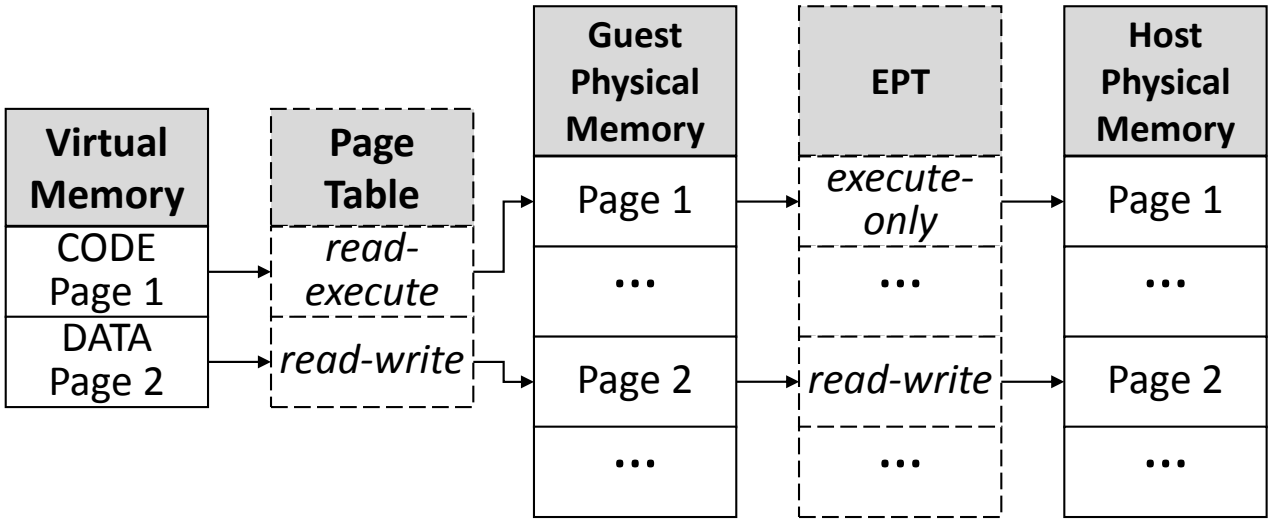
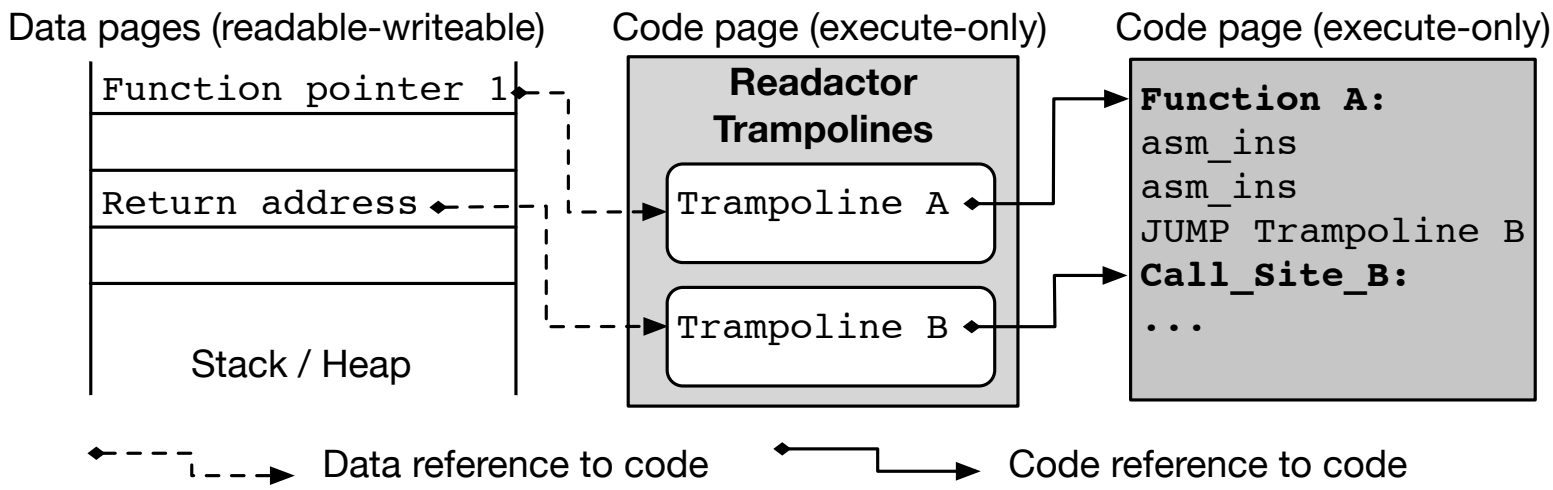


Figure 4: Relation between virtual, guest physical, and host physical memory. Page tables and the EPT contain the access permissions that are enforced during the address translation.

Trampoline

- Point function pointers and return addresses (indirect control transfers) to trampoline code



- Works because the trampoline layout is not correlated with the layout of functions
- I.e., trampoline addresses do not leak information about non-trampoline code

Trampoline for Calls

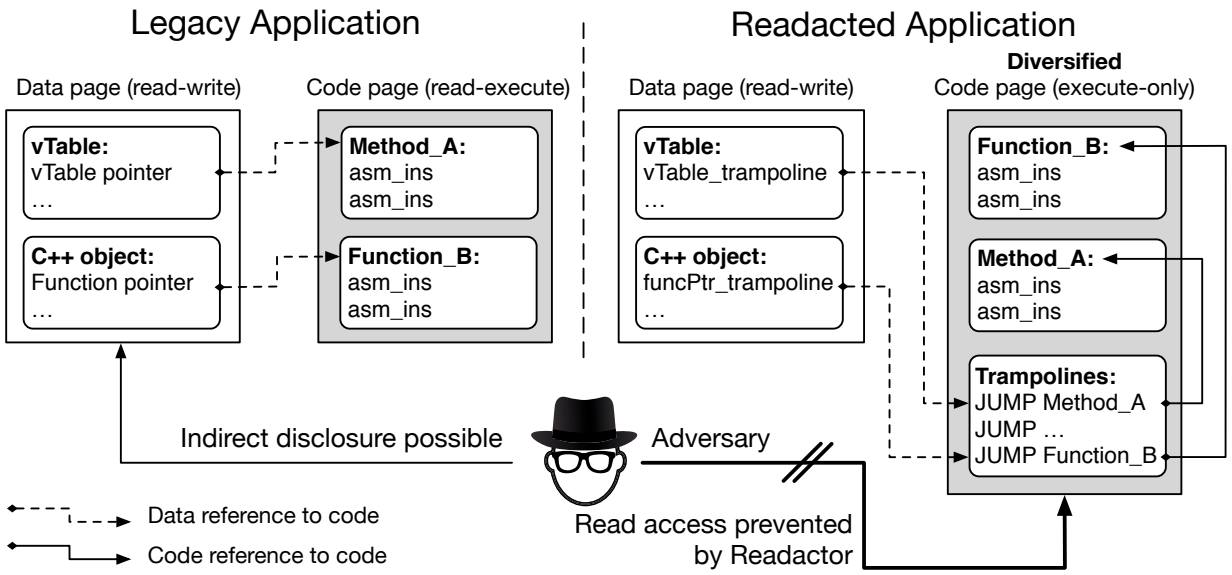


Figure 7: Hiding code pointers stored in the heap and in C++ vtables. Without Readactor, pointers to functions and methods may leak (left). With Readactor, only pointers to jump trampolines may leak and the layouts of functions and jump trampolines are randomized (right).

Trampoline for Returns

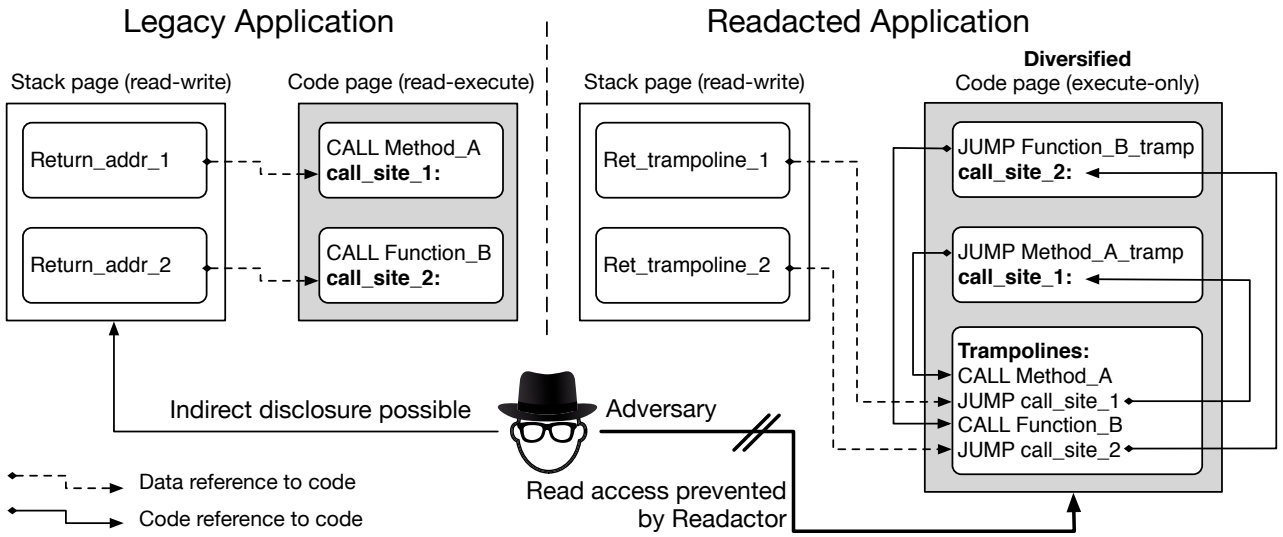


Figure 8: Hiding return addresses stored on the machine stack. Without Readactor, each activation frame on the stack leaks the location of a function (left). With Readactor, calls go through call trampolines so the return addresses pushed on the stack can only leak trampoline locations – not return sites (right).

Readactor Limitations

- Are there any limitations for the Readactor approach?

Other Forms of Diversity

- N-version programming
  - Run multiple versions of same program – presumably with different flaws
- Moving target defense
  - Change the system configuration incrementally (IP addrs)
  - After N time units or when under attack
- Deception
  - Build false versions of legitimate behaviors (honeypot)
  - Build inconsistent versions of legitimate behaviors

Inconsistent Syscall Behavior

PENNSTATE 1855

- Hypothesis: malware is more sensitive to inconsistent system call behavior than normal software
- Experiment
  - Vary execution of some system calls
  - Not all can be varied without breaking programs
- Strategies
  - Silence system calls (return bogus value)
  - Change response size or some bytes of file offset
  - Change delays

Inconsistent Syscall Behavior

- Hypothesis: malware is more sensitive to inconsistent system call behavior than normal software
- Experimental Results
  - Malware behavior degraded performance
  - Only want to apply to unknown software

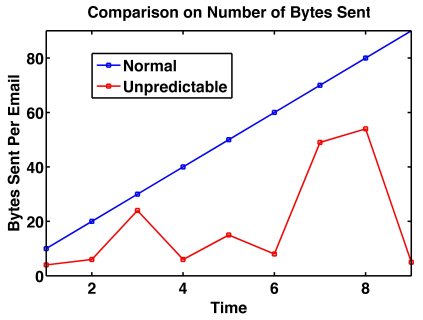


Figure 1: Comparison of email bytes sent from bots in predictable and unpredictable environments.

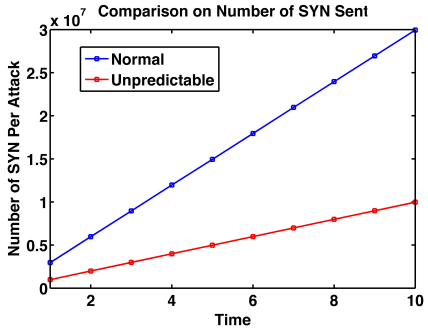


Figure 2: Comparison of SYN-flood attacks in standard and unpredictable environments. Unpredictability can increase the DDoS resource requirements.

Take Away

- Memory errors are the classic vulnerabilities in C programs (buffer overflow)
  - Despite years of exploration into defenses a Turing-complete approach to exploitation remains given an appropriate memory error (return-oriented programming)
- ASLR has been suggested as the way to block memory attacks, such as ROP
  - May be victimized by disclosure attacks
  - Readactor aims to eliminate disclosure
- Alternatives, such as deception, could be applied