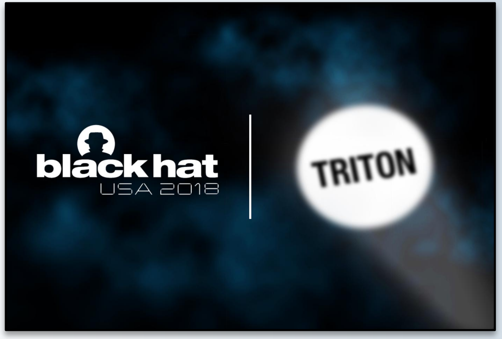
Cybersecurity Fundamentals

Will Hedgecock
Sandeep Neema
Bryan Ward

High-Profile Attacks

Solarwinds software compromised

Remote hacker hijacked jeep



Triton attack against industrial control system safety controllers

Equifax data leaked by attackers

Russia hacked Viasat satellites used by Ukraine

Meat processing plant ransomware attack

ISIS
Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu
VANDERBILT UNIVERSITY

Threat Models

- Many ways for attackers to achieve malicious goals
- Operating systems are integral to system security
  - Kernel has increased privileges
  - OS controls how many resources are accessed and protected
- System security is a product of the security of all levels of the compute stack
- Security should not be an afterthought!
- A system is only as secure as its weakest link so don’t let your software be it!
- Attacks have been demonstrated at all levels
  - Hardware – e.g., Spectre/Meltdown, Rowhammer
  - Operating System – e.g., Dirty Pipe allowed privilege escalation
  - Libraries – e.g., Heartbleed – leaks sensitive memory (e.g., credit cards and keys)
  - Applications – Myriad ransomware examples
  - Networks, social engineering attacks, and other threat vectors as well

Threat Models – Continued

- An attacker can take many forms and have various degrees of access

Masquerader

An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account

Misfeasor

A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges

Clandestine user

An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection

Threat Models – Continued

- Malicious Software – programs that exploit vulnerabilities in computing systems
- Also referred to as malware
- Can take many forms:
  - Parasitic – Fragments of programs that cannot exist independently of some actual application program
    - E.g., Viruses, logic bombs, backdoors, etc.
  - Independent – Self-contained programs that can be scheduled and run by the OS
    - E.g., Application downloaded and run by unsuspecting user
- Malware can have different intended effects, for example:
  - Ransomware
  - Leak data
  - Corrupt data or compromise programs

Security Goals – CIA Triad

- Confidentiality
  - If some piece of information is supposed to be hidden from others, don't allow them to find it out.
- Integrity
  - If some piece of information or system component is supposed to be in a particular state, don't allow an adversary to change it.
- Availability
  - If some information or service is supposed to be available for your own or others' use, make sure an attacker cannot prevent its use

CIA Triad

Security Goals – Principle of Least Privilege

"Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job."

J. Saltzer and M. Schroeder, 1975

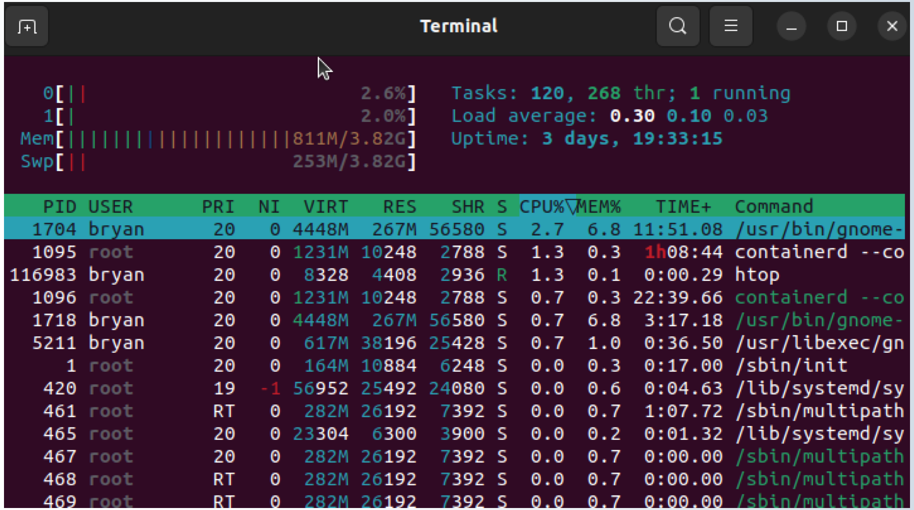
(Counter) Example: Windows XP default account named "Administrator." Millions of people used the administrator account for their everyday computing, while running with administrative privileges!

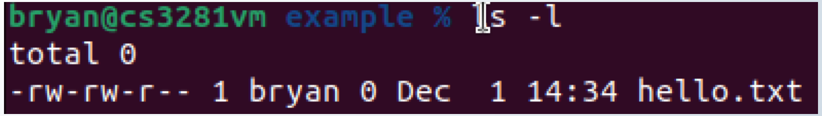
Authentication

- Authentication is the OS mechanism to identify the <u>principle</u> or user
- Many ways of implementing authentication
  - Password
  - 2FA
  - Cryptographic keys (e.g., ssh keys, as we used for github)
  - Biometrics (e.g., fingerprint)
  - Physical thing (e.g., yubikey, RFID tag, Vanderbilt ID card)
  - Etc.
- When implementing authentication mechanisms, it is best not to store secrets
  - It's particularly bad if an attacker can find them
  - Example: Don't store passwords in plain text, use hashing and salting

Users in the OS

- The OS keeps track of which processes are associated with what user
- When a process is forked, it is associated with the same user as the parent
  - Unless setuid bit is set, in which case it runs as the user that owns the file
- System calls such as setuid() used to change user id of a process if it should be different than the parent (e.g., sudo, ssh server, login process, etc.)
- Processes have permissions to <u>objects</u> based upon process user





Malicious Software

- Malicious software can be run by a well-intentioned benign user
  - Downloaded from untrusted internet source (don’t click the link!)
  - Software supply chain compromised (e.g., solar winds)
  - Etc.
- User could have malicious intents
  - Insider threat
  - Stolen credentials (e.g., password) and logged in and ran malicious software
- Benign software can be hijacked, even by remote attackers
- Systems designed for security minimize trust based on the assumption that other software could be malicious or compromised

Trust vs. Trustworthiness

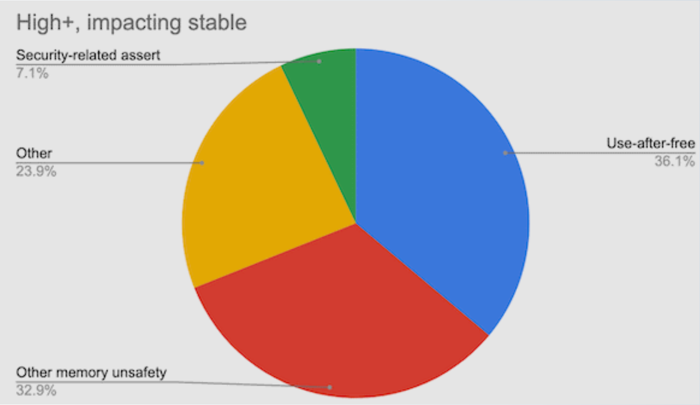
- Trusted software: software that must operate correctly for the system to behave properly or be secure.
  - Want to <u>minimize</u> the amount of trusted software
  - For example: kernel is trusted. If kernel is compromised, it can read or corrupt all of memory
  - Firefox does not need to trust Word. The kernel provides isolation from Word and other processes
- Trust<u>worthy</u> software: software that is correct, or very likely to be correct, and therefore is worthy of being trusted
  - Want to <u>maximize</u> the amount of trustworthy software
  - Challenge: making software trustworthy is very difficult and expensive
  - Example: seL4 microkernel has been formally verified to be correct. It is highly trustworthy

Memory Corruption

Example: Buffer Overflow

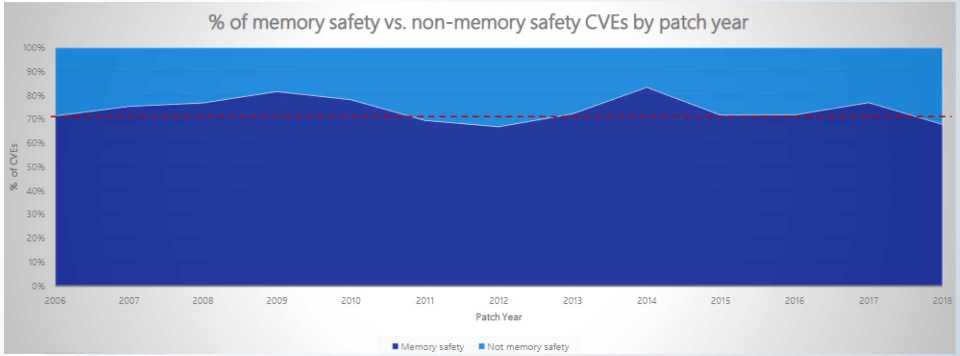
```
char buffer[10] = “”;
strcpy(A, “Something Very Malicious”);
```

Something	Very Malicious
buffer[10]	Other data



Other memory corruption examples:

- Stack/heap overflow
- Use after free
- Format string
- Etc.

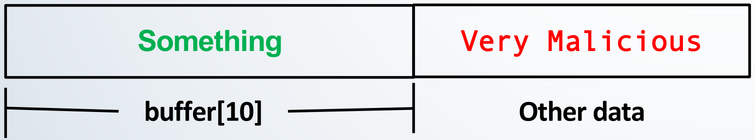


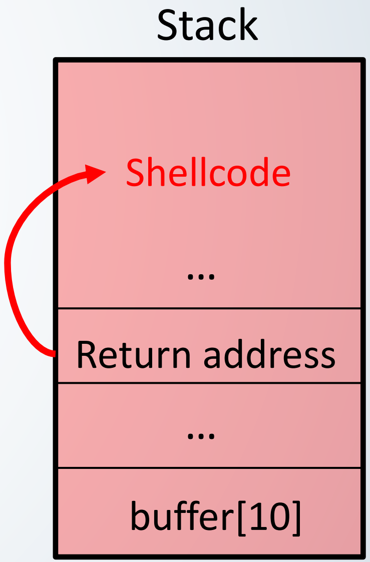
Memory-corruption vulnerabilities represent ~70% of all reported vulnerabilities year over year according to both Microsoft and Google

Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

Stack Overflow and Shellcode

- Attacker could overwrite parts of the stack and corrupt a return address to point to an arbitrary location
- What if that arbitrary location is somewhere the attacker controls (e.g., could write to)?
- Shellcode: code to "pop a shell" or fork and exec a shell process allowing the attacker to run arbitrary commands as the victim user





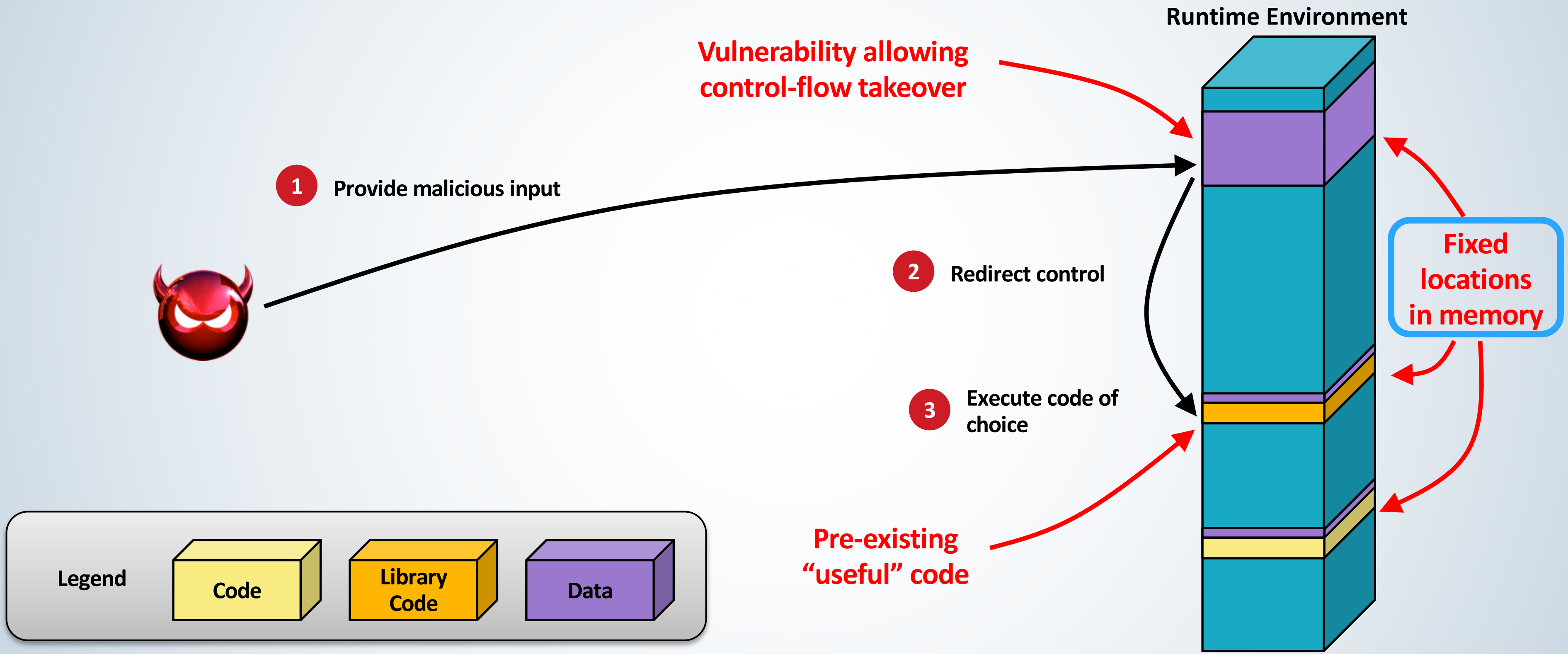
Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

Simple Defenses

- Data Execution Prevention (DEP or W xor X)
  - Do not allow execution of memory that is in a data segment
  - Prevents previous attack by not allowing the execution of data on the stack
  - Deployed in all modern OSes (Windows, Linux, Mac, Android, etc.)
- Stack Canaries
  - "Canary in the coal mine"
  - Place "canary" or special value on the stack between stack frame
  - Check the integrity of the canary before every function return
    - If it is as expected, great!
    - If it has been corrupted, a stack overflow has potentially corrupted the return address
  - Weak if the attacker can figure out the canary value(s)
  - Only protects against stack over/underflows (e.g., not the heap)

Memory Corruption

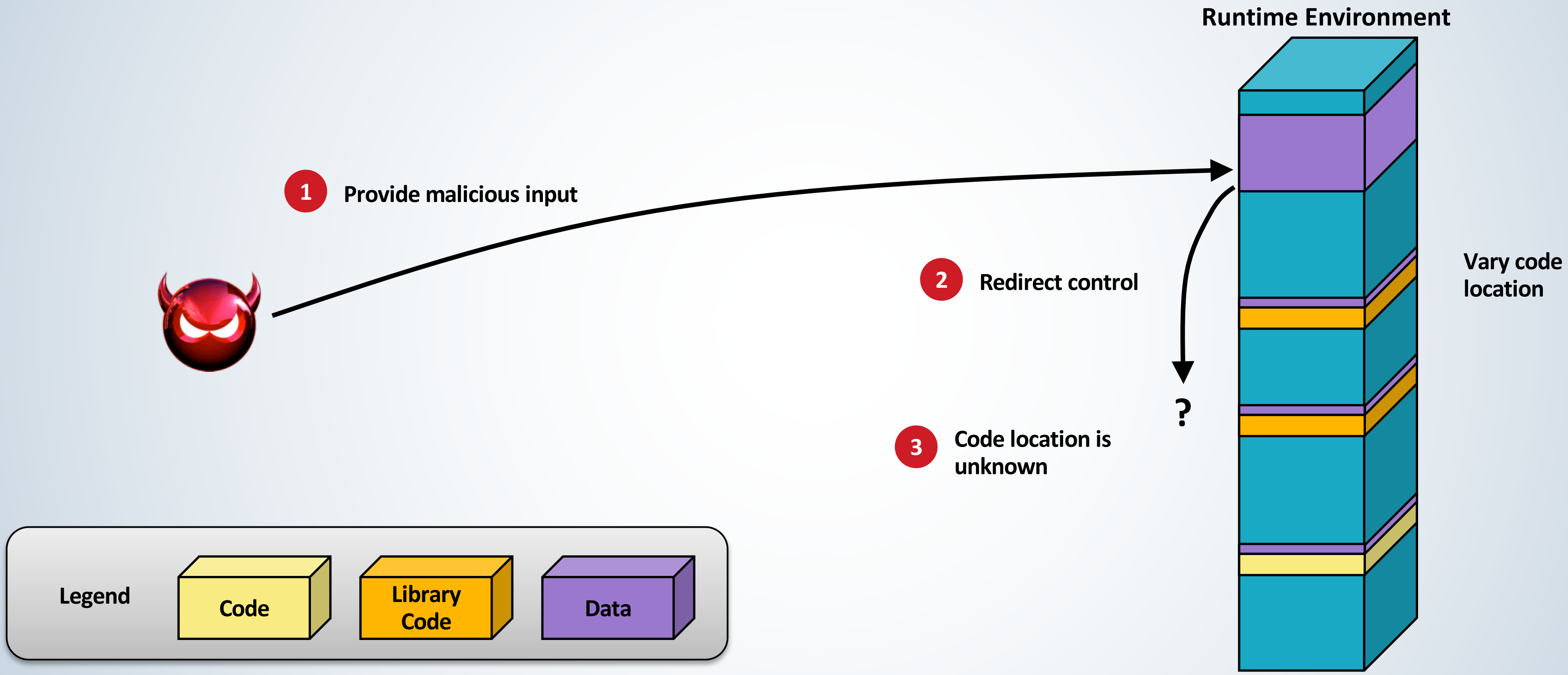
Control-Flow Hijacking



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

Moving-Target Defense

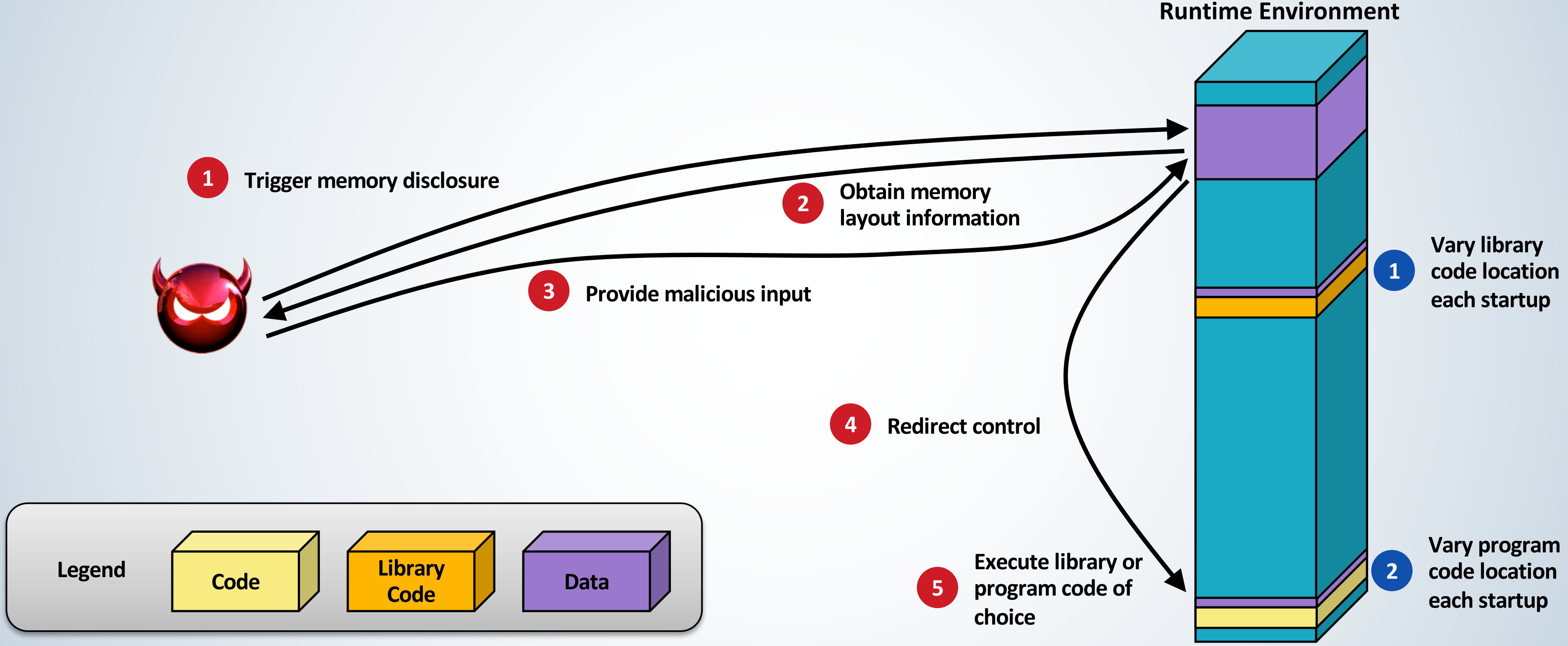
Address-Space Randomization



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

VANDERBILT UNIVERSITY

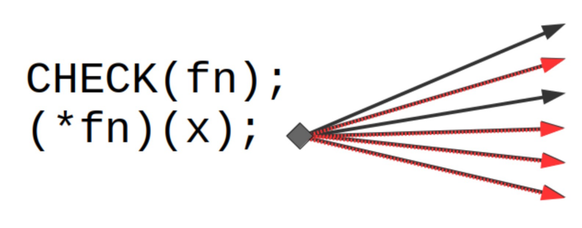
Bypassing Randomization



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu

Control-Flow Integrity (CFI)

- Control-Flow Integrity protects applications against control-flow hijack attacks by ensuring that the control-flow of the application never leave the predetermined, valid control-flow that is defined at the source code/application level. This means that an attacker cannot redirect control-flow to alternate or new locations.



Summary

- Deployed mitigations do not stop all attacks
- DEP stops code injection but not code reuse
- Memory randomization is useful, but vulnerable to information leaks
- CFI is strong, but not always precise
  - Does not protect against data-only attacks
- Lesson learned: Be careful when writing software to ensure you do not introduce vulnerabilities that can be exploited
- Use memory-safe languages where possible to eliminate many types of problematic vulnerabilities