CS161 Midterm 1 Review

Midterm 1: March 4, 18:30-20:00

Same room as lecture

Security Analysis and Threat Model

- Basic security properties
  - CIA
- Threat model

  A. We want perfect security

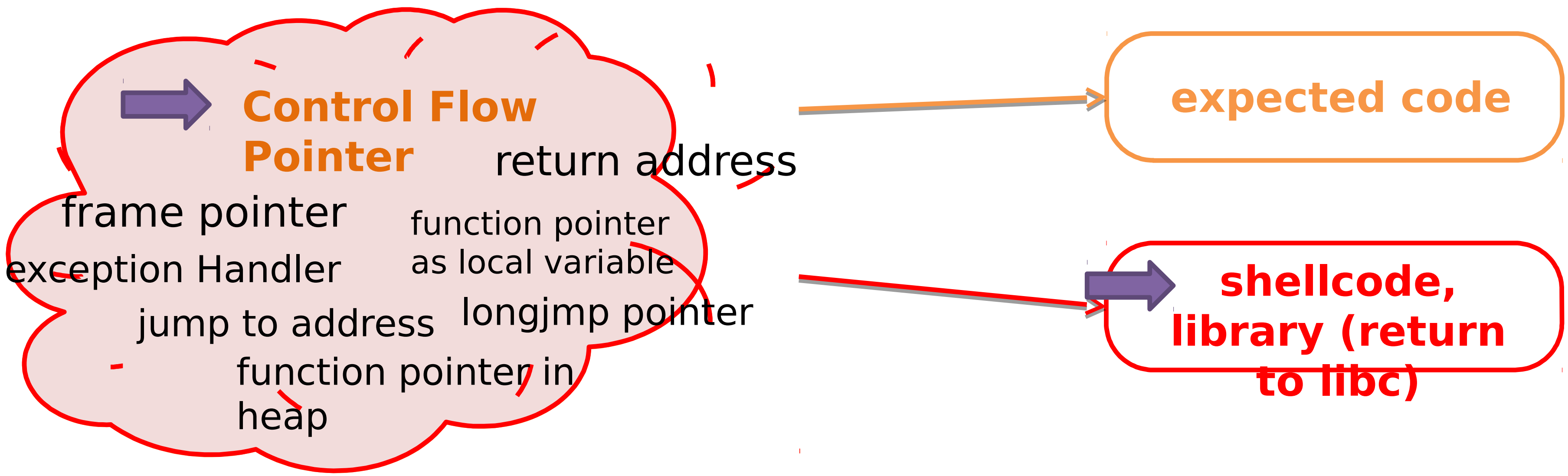
  B. Security is about risk analysis and economics

  Answer is B.

Software Vulnerabilities

- Buffer overflow vulnerabilities and attacks
- Integer overflow vulnerabilities and attacks
- Format string vulnerabilities and attacks
- Arc injection/return-to-libc/ROP vulnerabilities and attacks
- General control hijacking attacks
- Data hijacking attacks

General Control Hijacking



Overwrite Step:

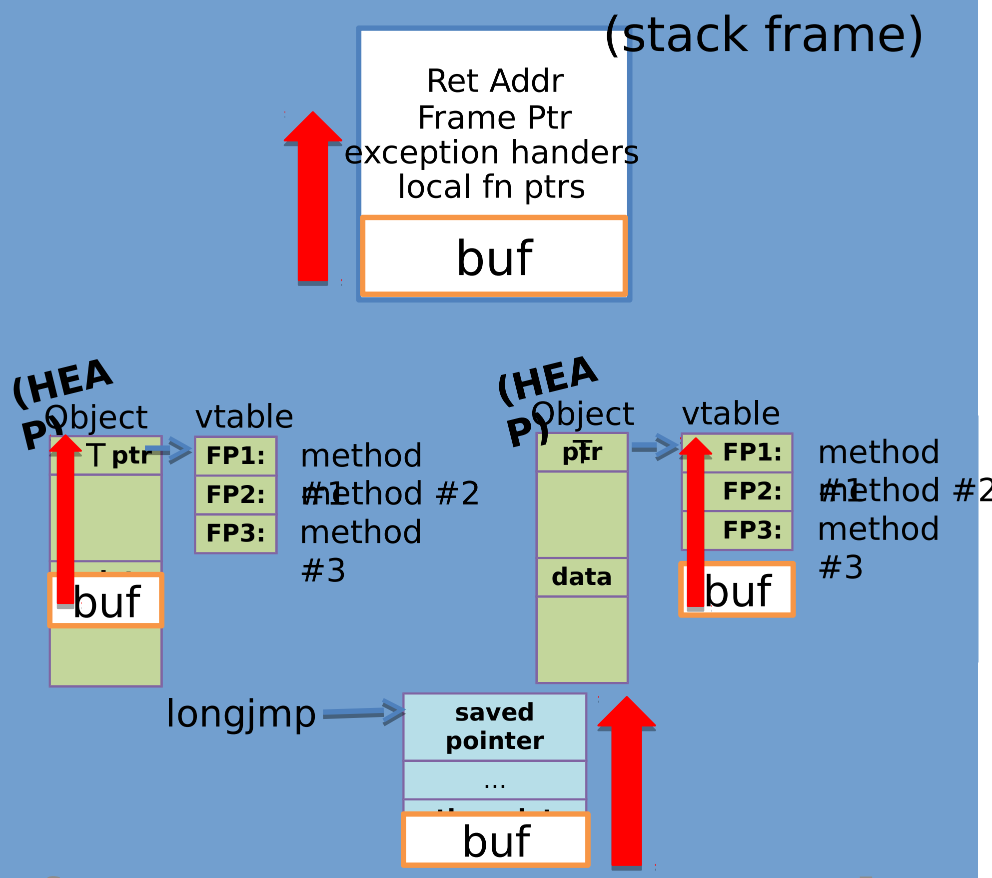
Find some way to **modify** a Control Flow Pointer to point to your shellcode, library entry point, or other code of interest.

Activate Step:

Find some way to **activate** that modified Control Flow Pointer.

Instances of Control Hijacking

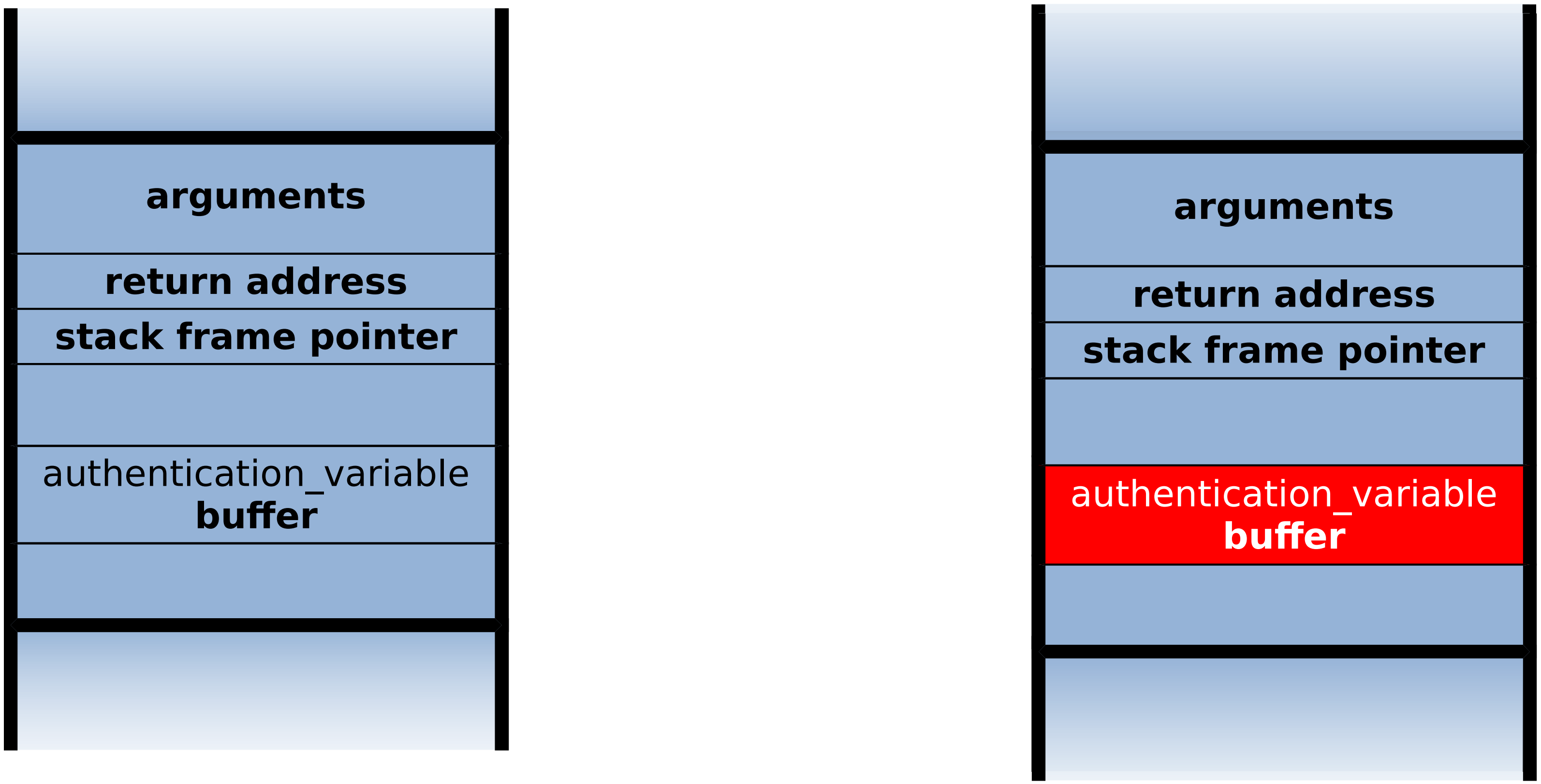
Location in Memory	Control Flow Pointer	How to activate
Stack	Return Address	Return from function
Stack	Frame Pointer	Return from function
Stack	Function Pointers as local variables	Reference and call function pointer
Stack	Exception Handler	Trigger Exception
Heap	Function pointer in heap (i.e. method of an object)	Reference and call function pointer
Anywhere	setjmp and longjmp program state buffer	Call longjmp



Data Hijacking

odifying data in a way not intended

Example: Authentication variab



Exploited Situation:
User types in a password which is long enough to overflow buffer and into the authentication_variable. The user is now unintentionally authenticated

Stack and Format Strings

- Function behavior is controlled by the format string
- Retrieves parameters from stack as requested: “%”
- Example:

printf [illegible]r %d has no address, number %d has: %08x\n”, I, a, &a)

stack top
…
<&a>
<a>
<i>
A
…
stack bottom

A	Address of the format string
I	Value of variable I
a	Value of variable a
&a	Address of variable a

SW Vuln. Defenses

- Non-execute (NX)
- Stack canaries
- ASLR
- Bounds check
- Which defenses are effective against what attacks?

Effectiveness and Limitations

- Defense against buffer overflow attacks

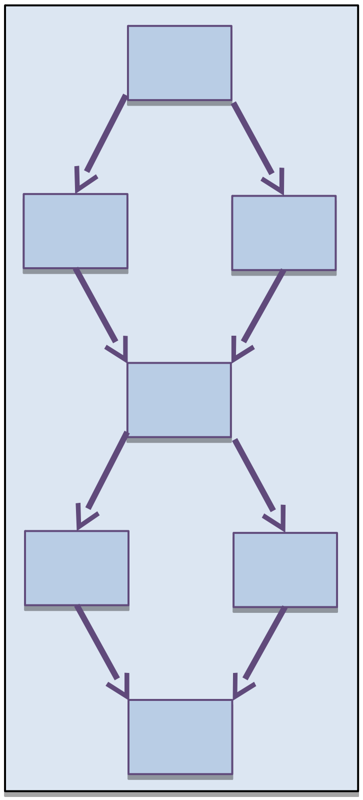
* When Applicable

Defenses/Mitigations	Code Injection	Arc Injection
Stack	Non-Execute (NX)* ASLR StacKGuard(Canaries)	ASLR StacKGuard(Canaries)
Heap	Non-Execute (NX)* ASLR	ASLR
Exception Handlers	Non-Execute (NX)* ASLR	ASLR

Fuzzing

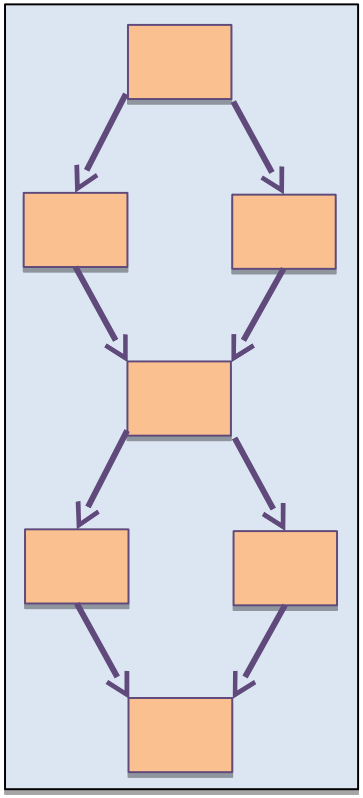
- Random fuzzing
- Mutation-based fuzzing
- Generation-based fuzzing
- Code coverage
  - line, branch and path coverage
- Example problem: given a program, calculate how many inputs can achieve a full line/branch/path coverage (e.g., Discussion 5)

Coverage Metrics



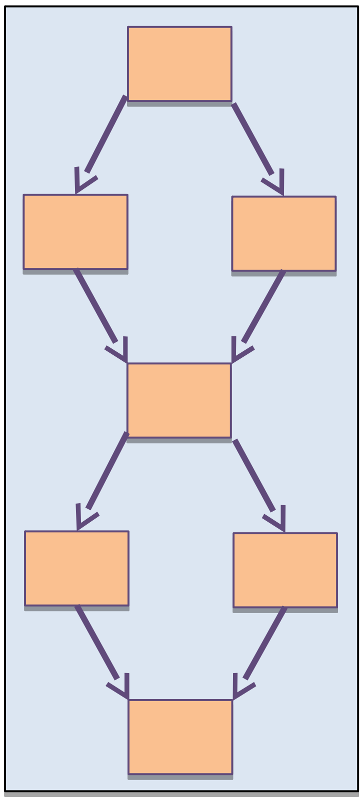
Lines

Coverage Metrics

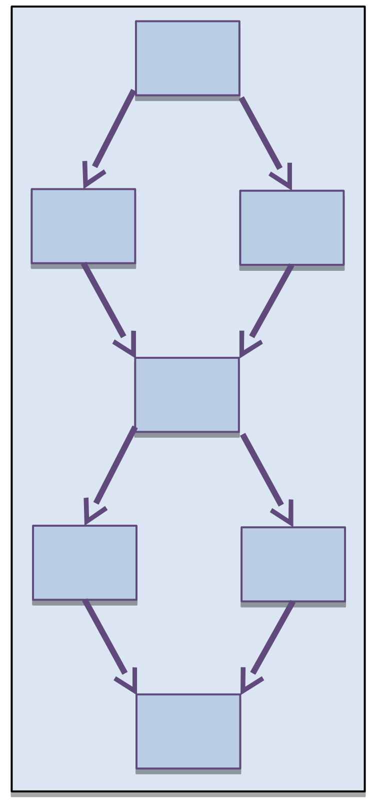


Lines

Coverage Metrics

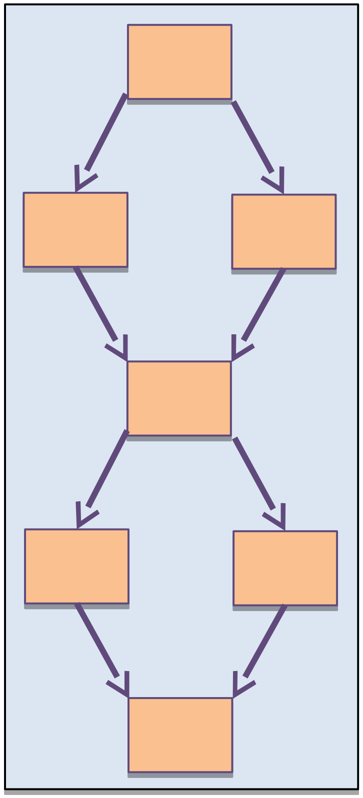


Lines

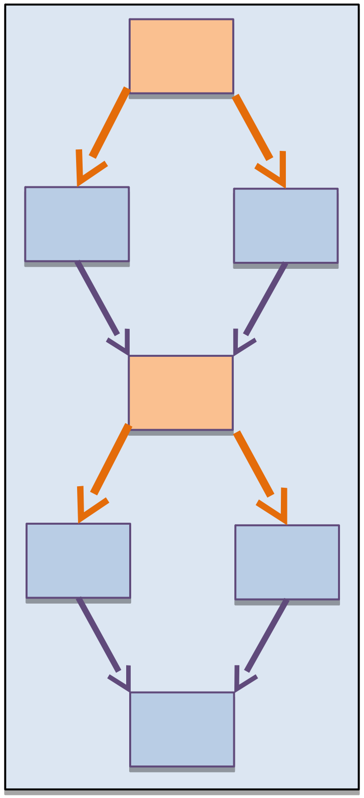


Branche
s

Coverage Metrics

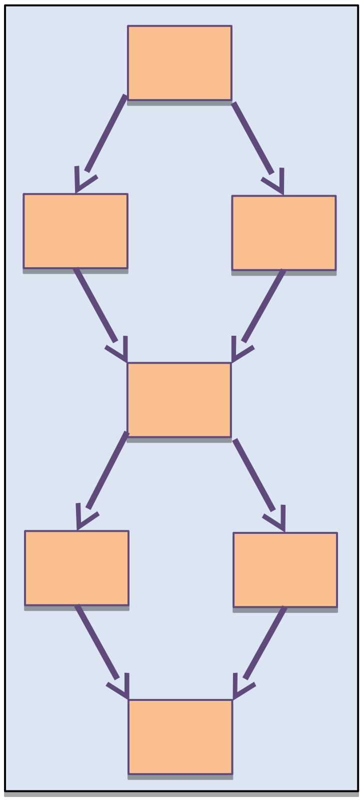


Lines

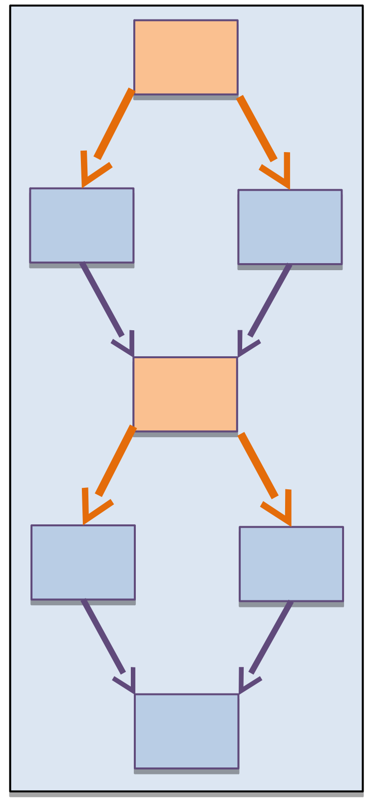


Branche
s

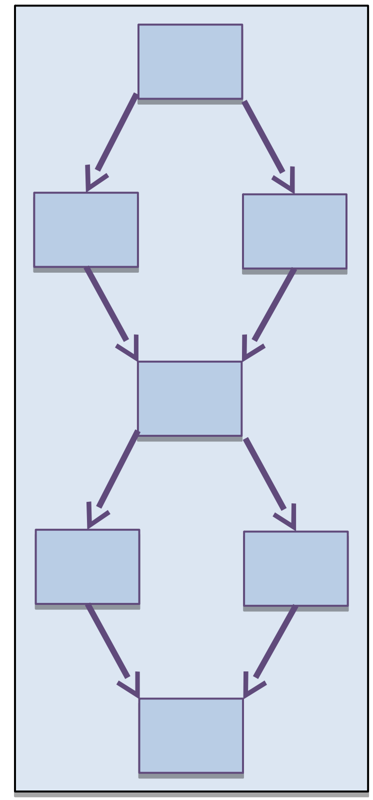
Coverage Metrics



Lines

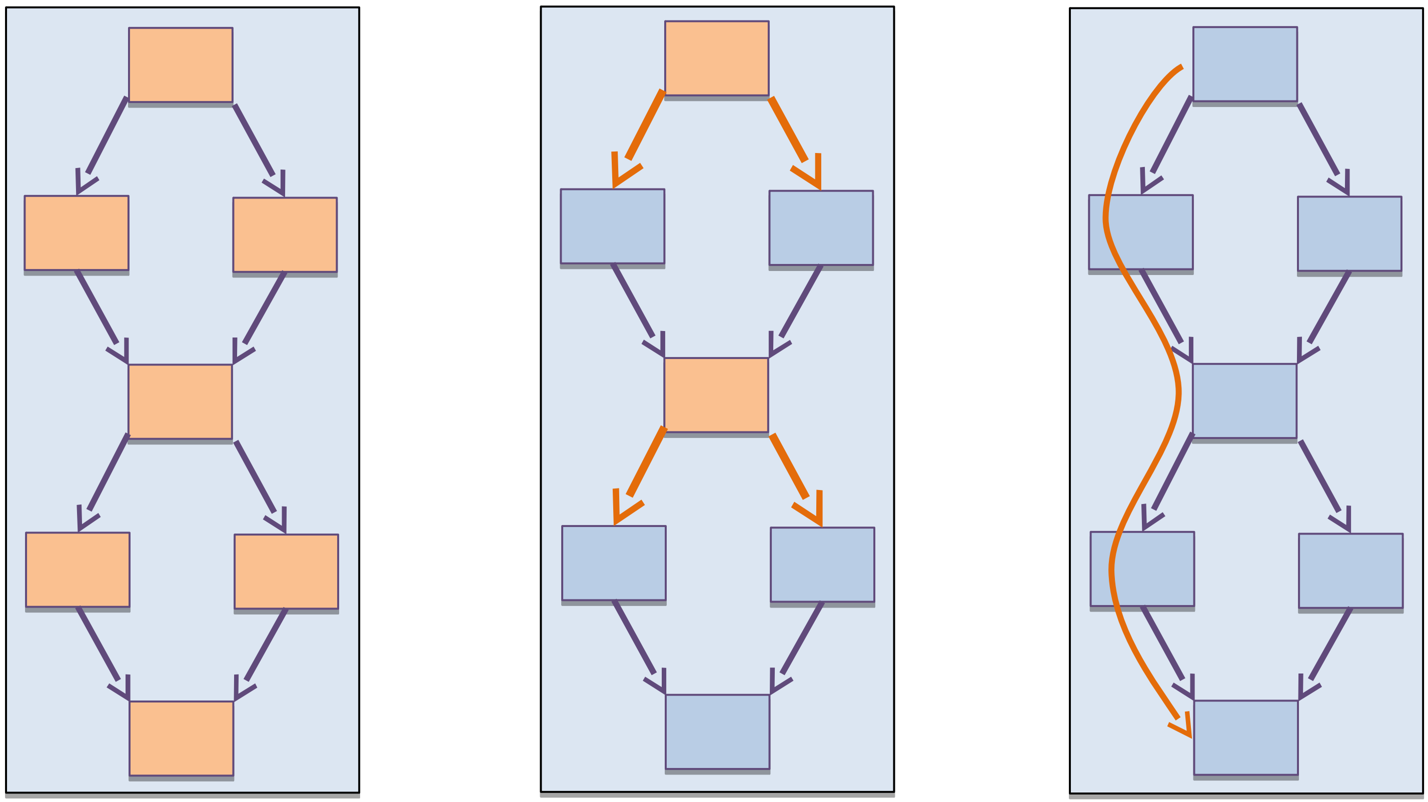


Branche
s



Paths

Coverage Metrics

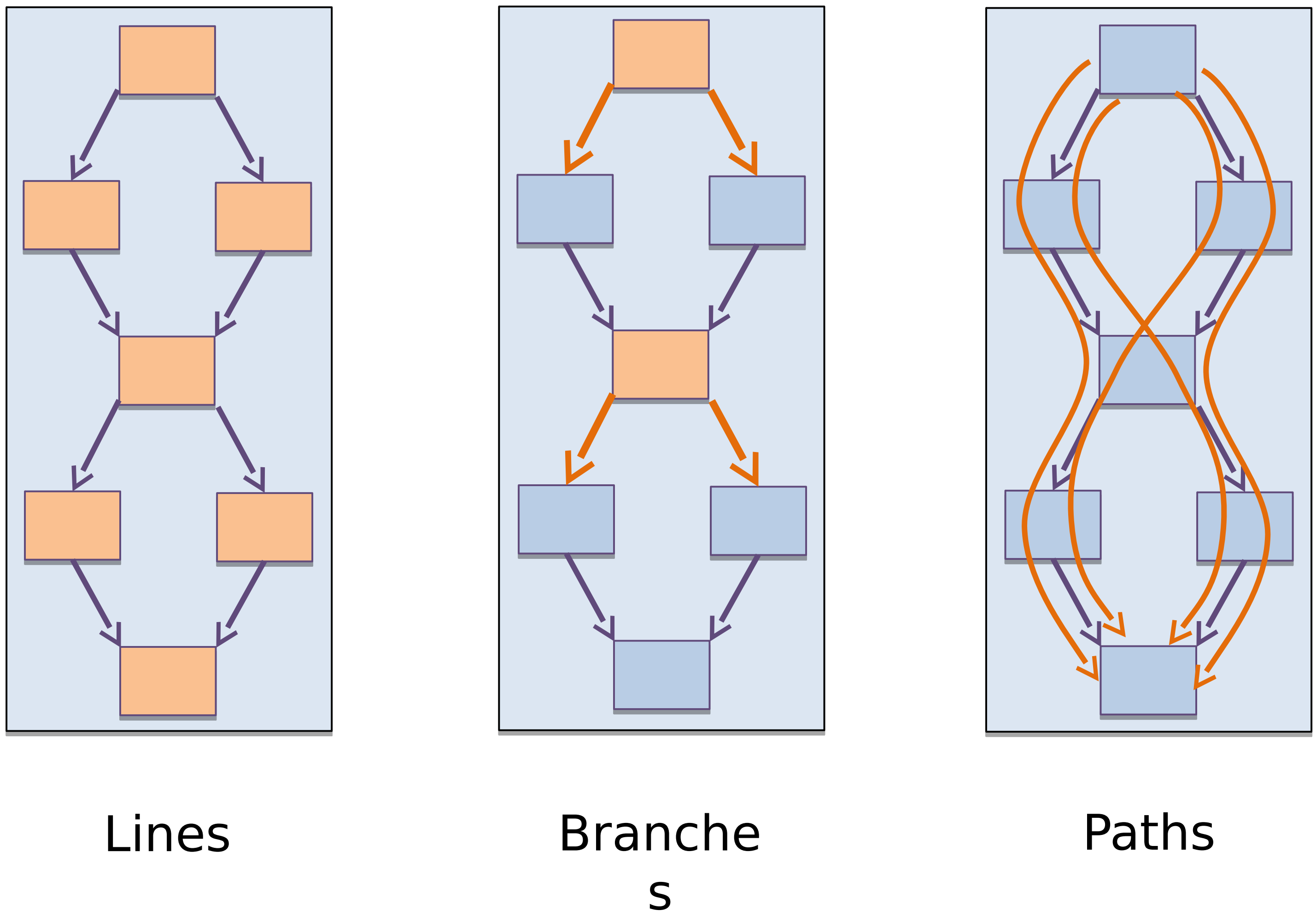


Lines

Branche
s

Paths

Coverage Metrics



Quiz on Line Coverage

How many lines are in this code?

- 1
- 2
- 3
- 4

```
if (a > 2)
  a = 2;
if (b > 2)
  b = 2;
```

How many test cases (pairs of values for (a,b)) are needed to achieve 100% line coverage?

- 1
- 2
- 3
- 4

Quiz on Branch Coverage

How many branches are in this code?

- 1
- 2
- 3
- 4

```
if (a > 2)
  a = 2;
if (b > 2)
  b = 2;
```

How many test cases (pairs of values for (a,b) are needed to achieve 100% branch coverage?

- 1
- 2
- 3
- 4

Quiz on Path Coverage

How many paths are in this code?

- 1
- 2
- 3
- 4

```
if (a > 2)
  a = 2;
if (b > 2)
  b = 2;
```

How many test cases (pairs of values for (a,b) are needed to achieve 100% path coverage?

- 1
- 2
- 3
- 4

Completeness of Coverage Metrics

```
my_copy(char* dst, char* src){

  if (dst && src)
    strcpy(dst, src);

}
```

Which of the following coverage results guarantee the bug will be found?

- 100% line coverage
- 100% branch coverage
- 100% path coverage
- None of the above

Properties of Coverage Metrics

- A numeric measure of an analysis
- An objective basis for comparing different analyses
- A way to evaluate if no progress is made (no coverage metrics are increasing)

Important: Metrics are not sufficient conditions for completeness.

100% coverage does not mean all sources of vulnerabilities have been evaluated.

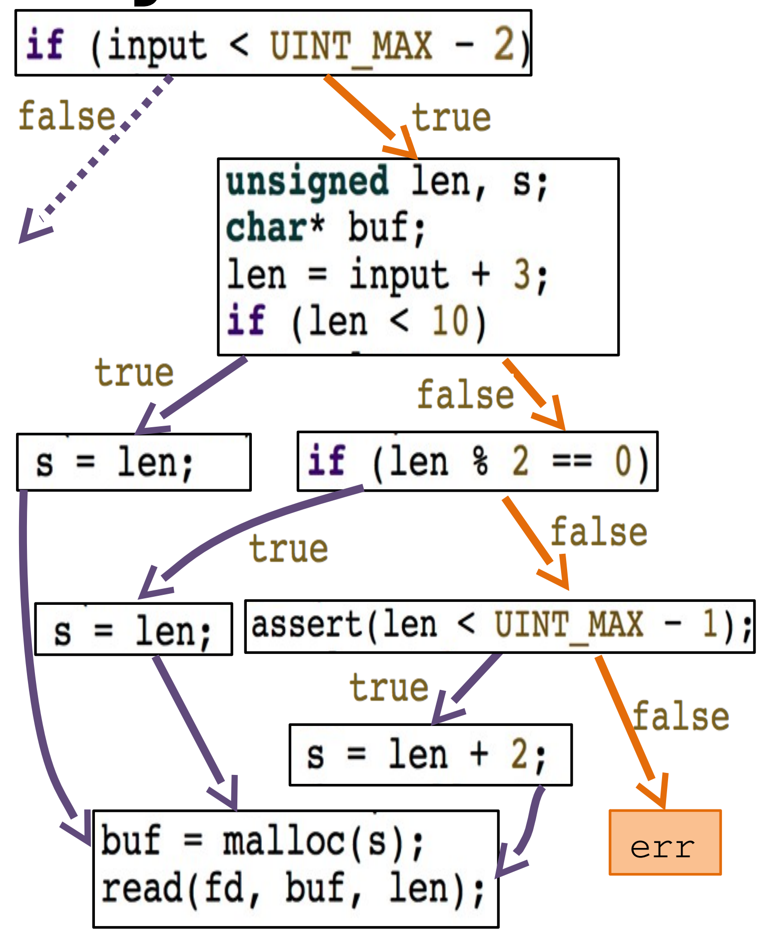
Symbolic Execution

- Path predicates
- Security vulnerabilities as assertion violations
- How to use symbolic execution to find bugs
- Constraint-based automatic test case generation
- Challenges for symbolic execution

Assertion Violation as Satisfiability

In the appropriate theory, the formula

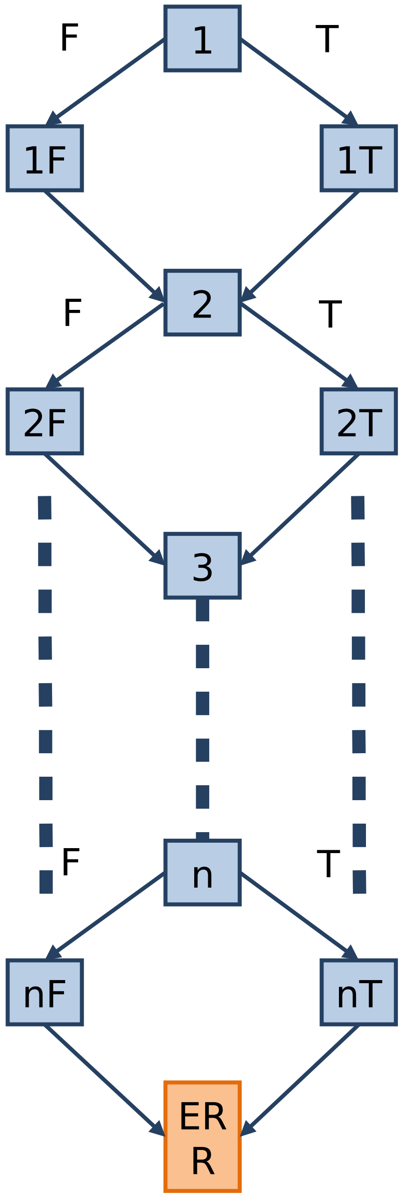
	input < UINT_MAX - 2
&&	len == input + 3
&&	! (len < 10)
&&	! (len % 2 == 0)
&&	!(len < UINT_MAX – 1)



is satisfied by the assignment

input	UINT_MAX - 3
len	UINT_MAX

Quiz: Branches and Paths

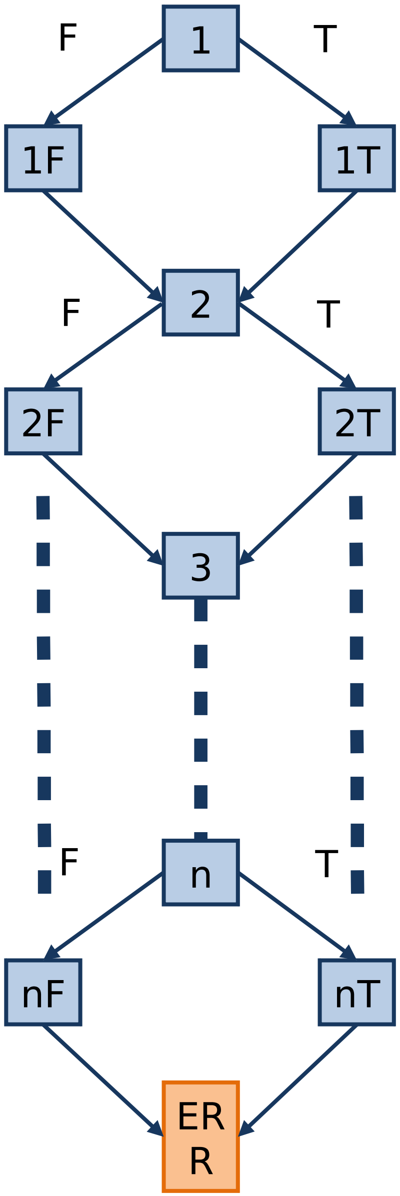


Suppose we want to know if there is a feasible path to the location ERR in this program.

Suppose we generate one path predicate for each path through this program.

How many path predicates are generated?

Quiz: Branches and Paths



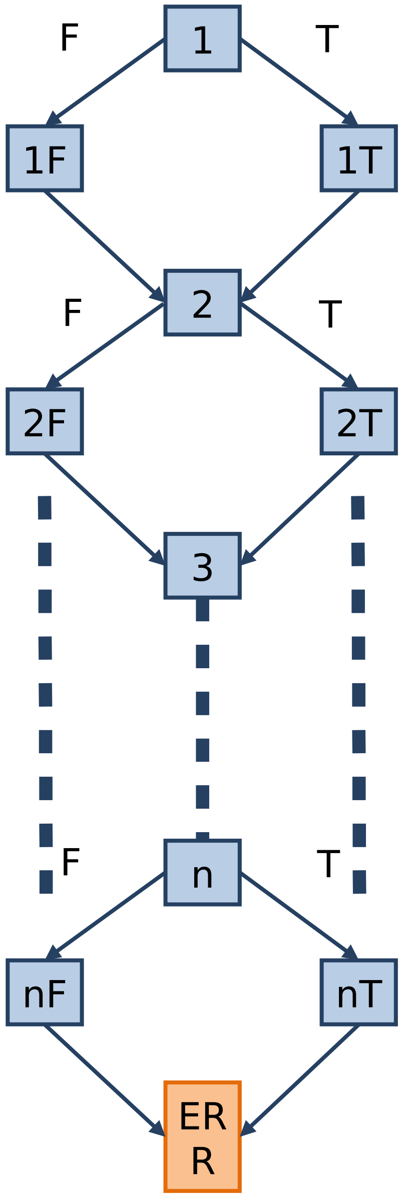
Suppose we want to know if there is a feasible path to the location ERR in this program.

Suppose we generate one path predicate for each path through this program.

How many path predicates are generated?

2^n

Quiz: Branches and Paths



Suppose we want to know if there is a feasible path to the location ERR in this program.

Suppose we generate one path predicate for each path through this program.

How many path predicates are generated?

2^n

Number of predicates can be *exponential in the number of branches.*

Topics Covered in Midterm 2

- Static analysis
- Program Verification
- Security principles and architectures
- Malware
- Other topics after midterm 2