Lecture 11 –
Defense Strategies

[COSE451] Software Security

Instructor: Seunghoon Woo

Spring 2024

Overview

- **Defense strategies**

Defense strategies

- **Vulnerability detection strategies**
  - **Static analysis**
    - Examining source code without executing it
      - To identify potential security vulnerabilities
    - Also called whitebox testing
  - **Dynamic analysis**
    - Running the program and analyzing its behavior during execution
      - To identify potential security vulnerabilities
    - Also called blackbox testing

Defense strategies

- **Static analysis: symbolic execution**
  - Evaluate the program on **symbolic input values**
    - Rather than using a concrete input
    - Use an automated theorem prover (e.g., SMT solver) to check whether there are corresponding concrete input values that make the program fail
    - Returning a result that is expressed in symbolic constants that represent input values

Defense strategies

- **Static analysis: symbolic execution**
  - **Advantage?**
    - Symbolic execution can avoid giving false warnings!
      - Any error found by symbolic execution represents a real
      - Providing feasible path through the program
      - Presenting a test case that illustrates the error
  - **Disadvantages?**
    - Incomplete theorem prover
    - Limited scalability

Defense strategies

- **Static analysis: symbolic execution**
  - Example: random testing

```
void main(int x, int y)
{
  z = 2*y;
  if (z == x)
  {
    if (x > y + 10)
      ERROR!
  }
}
```

Defense strategies

- **Static analysis: symbolic execution**
  - Example: random testing

```
void main(int x, int y)
{
  z = 2*y;
  if (z == x)
  {
    if (x > y + 10)
      ERROR!
  }
}
```

Q. What is the probability of reaching an error with random testing? $(1 \leq x, y \leq 100)$

① 0.04% ② 0.4% ③ 4% ④ 40%

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
int x=0, y=0, z=0;
if(a) {
  x = -2;
}
if (b < 5) {
  if (!a && c) { y = 1; }
  z = 2;
}
assert(x + y + z != 3);
//assert: Error detection code
//        planted in a place
//        where an error will be fatal
```

a	b	c	x	y	z	x+y+z

if x+y+z !=3:
  pass (no problem)

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
int x=0, y=0, z=0;
if(a) {
  x = -2;
}
if (b < 5) {
  if (!a && c) { y = 1; }
  z = 2;
}
assert(x + y + z != 3);
//assert: Error detection code
//        planted in a place
//        where an error will be fatal
```

a	b	c	x	y	z	x+y+z
1	2	1				

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
int x=0, y=0, z=0;
if(a) {
  x = -2;
}
if (b < 5) {
  if (!a && c) { y = 1; }
  z = 2;
}
assert(x + y + z != 3);
//assert: Error detection code
//        planted in a place
//        where an error will be fatal
```

a	b	c	x	y	z	x+y+z
1	2	1	-2	0	2	0

pass

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
int x=0, y=0, z=0;
if(a) {
  x = -2;
}
if (b < 5) {
  if (!a && c) { y = 1; }
  z = 2;
}
assert(x + y + z != 3);
//assert: Error detection code
//        planted in a place
//        where an error will be fatal
```

a	b	c	x	y	z	x+y+z
α	β	γ				

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```c
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

a	b	c	x	y	z	x+y+z
α	β	γ				

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```c
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

a	b	c	x	y	z	x+y+z
α	β	γ				

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

a	b	c	x	y	z	x+y+z
α	β	γ				

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg\alpha \wedge \beta \geq 5$	$\dots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

assert is not violated!

```
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

a	b	c	x	y	z	x+y+z
α	β	γ				

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg\alpha \land \beta \geq 5$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
9	$\neg\alpha \land \beta \geq 5 \land 0 + 0 + 0 \neq 3$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg\alpha \wedge \beta < 5$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg\alpha \land \beta < 5$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
6	$\neg\alpha \land \beta < 5 \land \gamma$	$\ldots, x \mapsto 0, y \mapsto 1, z \mapsto 0$
6	$\neg\alpha \land \beta < 5 \land \gamma$	$\ldots, x \mapsto 0, y \mapsto 1, z \mapsto 2$

Defense strategies

- **Static analysis: symbolic execution**
  - **Example**

```
1  int x=0, y=0, z=0;
2  if(a) {
3    x = -2;
4  }
5  if (b < 5) {
6    if (!a && c) { y = 1; }
7    z = 2;
8  }
9  assert(x + y + z != 3);
10 //assert: Error detection code
11 //        planted in a place
12 //        where an error will be fatal
```

Error detected!

line	Path condition	Symoblic environment
0	True	$a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma$
1	True	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
2	$\neg\alpha$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
5	$\neg\alpha \wedge \beta < 5$	$\ldots, x \mapsto 0, y \mapsto 0, z \mapsto 0$
6	$\neg\alpha \wedge \beta < 5 \wedge \gamma$	$\ldots, x \mapsto 0, y \mapsto 1, z \mapsto 0$
6	$\neg\alpha \wedge \beta < 5 \wedge \gamma$	$\ldots, x \mapsto 0, y \mapsto 1, z \mapsto 2$
9	$\neg\alpha \wedge \beta < 5 \wedge \gamma$ $\wedge \neg(0 + 1 + 2 \neq 3)$	$\ldots, x \mapsto 0, y \mapsto 1, z \mapsto 2$

Defense strategies

- **Limitation of symbolic execution**

```
int foo (int v){
    return secure_hash(v);
}

void test(int x, int y){
    z = foo (y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

Defense strategies

- **Limitation of symbolic execution**

```c
int foo (int v){
    return secure_hash(v);
}

void test(int x, int y){
    z = foo (y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

hash(y) cannot be solved by theorem prover..

Defense strategies

- **Static analysis: concolic execution**
  - Store program state concretely and symbolically
    - Use concrete values to simplify symbolic constraints
    - Solve constraints to guide execution at branch points
    - Explore all execution paths of the unit tested

Defense strategies

- **Static analysis: concolic execution**
  - Example

```c
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

Concrete state	Symbolic state
x = 22, y = 7	x = α, y = β

1st iteration

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

1st iteration

Concrete state	Symbolic state
x = 22, y = 7, z = 14	x = α, y = β, z = 2 * β

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

1st iteration

Concrete state	Symbolic state
x = 22, y = 7, z = 14	$x = \alpha, y = \beta, z = 2 * \beta$ $2 * \beta \neq \alpha$

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

Concrete state	Symbolic state
$x = 22, y = 7, z = 14$	$x = \alpha, y = \beta, z = 2 * \beta$
	$2 * \beta \neq \alpha$

To reach the opposite branch..

Constraint: $2 * \beta = \alpha$
Solution: $\alpha = 2, \ \beta = 1$

1st iteration

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

Concrete state	Symbolic state
x = 2, y = 1	$x = \alpha, y = \beta$

2nd iteration

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

2nd iteration

Concrete state	Symbolic state
x = 2, y = 1, z = 2	x = α, y = β, z = 2 * β

Defense strategies

- **Static analysis: concolic execution**
  - Example

```c
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

2nd iteration

Concrete state

$x = 2, y = 1, z = 2$

Symbolic state

$x = \alpha, y = \beta, z = 2 * \beta$

True $(2 * \beta = \alpha)$

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

2nd iteration

Concrete state

$x = 2, y = 1, z = 2$

Symbolic state

$x = \alpha, y = \beta, z = 2 * \beta$

$2 * \beta = \alpha \wedge$
$\alpha \leq \beta + 10$

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

2nd iteration

Concrete state | Symbolic state

To reach the opposite branch..

Constraint: $2 * \beta = \alpha \wedge \alpha > \beta + 10$
Solution: $\alpha = 30, \ \beta = 15$

$x = 2, y = 1, z = 2$

$x = \alpha, y = \beta, z = 2 * \beta$

$2 * \beta = \alpha \wedge$
$\alpha \leq \beta + 10$

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

3rd iteration

Concrete state	Symbolic state
x = 30, y = 15	x = α, y = β

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

3rd iteration

Concrete state	Symbolic state
x = 30, y = 15, z = 30	$x = \alpha, y = \beta, z = 2 * \beta$

Defense strategies

- **Static analysis: concolic execution**
  - Example

```c
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

Concrete state

x = 30, y = 15, z = 30

Symbolic state

$x = \alpha, y = \beta, z = 2 * \beta$

True $(2 * \beta = \alpha)$

3rd iteration

Defense strategies

- **Static analysis: concolic execution**
  - Example

```c
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

3rd iteration

Concrete state	Symbolic state
x = 30, y = 15, z = 30	x = α, y = β, z = 2 * β
	True (2 * β = α)
	True (α > β + 10)

Defense strategies

- **Static analysis: concolic execution**
  - Example

```
int double (int v){
    return 2*v;
}

void test(int x, int y){
    z = double(y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

3rd iteration

Concrete state

x = 30, y = 15, z = 30

Crashing input!

Symbolic state

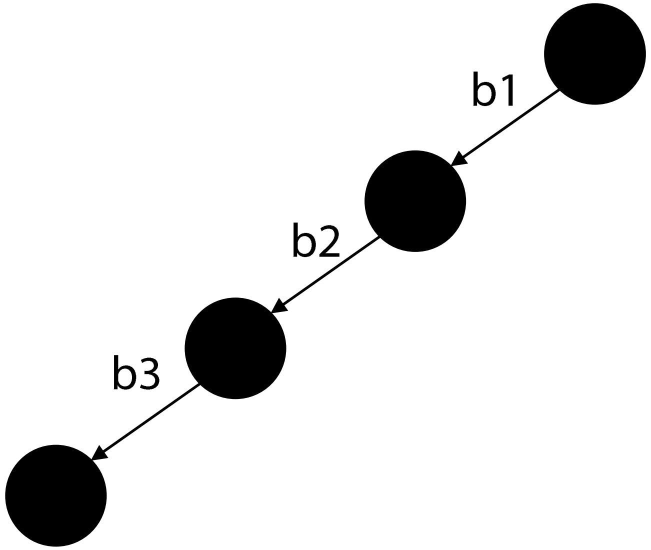
$x = \alpha, y = \beta, z = 2 * \beta$

True ($2 * \beta = \alpha$)

True ($\alpha > \beta + 10$)

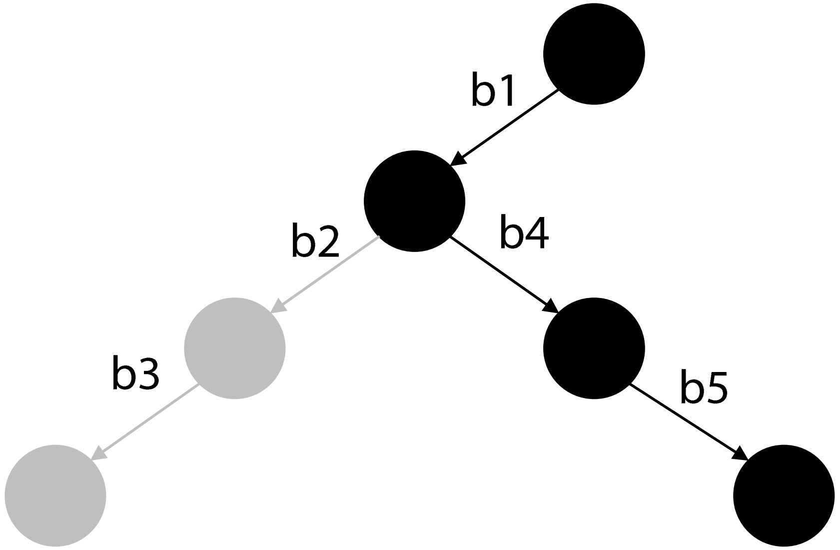
Defense strategies

- **Static analysis: concolic testing algorithm**



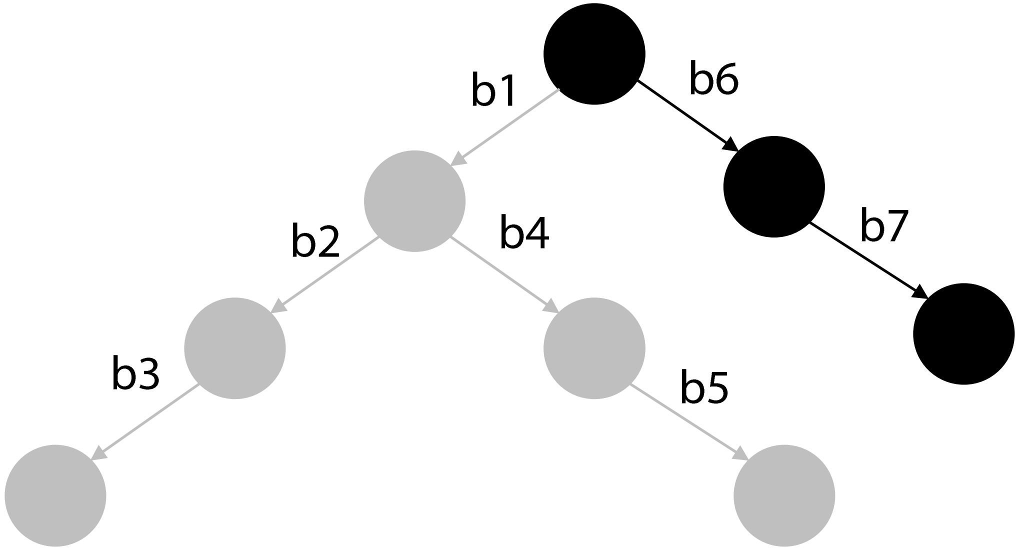
Defense strategies

- **Static analysis: concolic testing algorithm**



Defense strategies

- **Static analysis: concolic testing algorithm**



Defense strategies

- **Static analysis: concolic execution**
  - Another example

```
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

Concrete state	Symbolic state
x = 236, p = NULL	$x = \alpha$, $p = \beta$

1st iteration

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

Concrete state	Symbolic state
x = 236, p = NULL	x = α, p = β True (α > 0)

1st iteration

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

1st iteration

Concrete state	Symbolic state
x = 236, p = NULL	$x = \alpha, p = \beta$ $\alpha > 0 \land \beta = NULL$

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

1st iteration

Concrete state | Symbolic state

To reach the opposite branch..

Constraint: $\alpha > 0 \wedge \beta \neq NULL$
Solution: $\alpha = 236$, $\beta = [\,634 \mid NULL\,]$

Concrete state: x = 236, p = NULL

Symbolic state: $x = \alpha$, $p = \beta$
$\alpha > 0 \wedge \beta = NULL$

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

Concrete state	Symbolic state
x = 236, p = [634 | NULL]	$x = \alpha$, $p = \beta$, $p\text{->}data = \gamma$, $p\text{->}next = \delta$

2nd iteration

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

2^{nd} iteration

Concrete state

x = 236, p = [634 | NULL]

Symbolic state

$x = \alpha$, $p = \beta$,
$p\text{->}data = \gamma$,
$p\text{->}next = \delta$
True ($\alpha > 0$)
True ($\beta \neq NULL$)

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

2nd iteration

Concrete state	Symbolic state
x = 236, p = [634 | NULL]	x = α, p = β, p->data = γ, p->next = δ $\alpha > 0 \wedge \beta \neq NULL \wedge 2 * \alpha + 1 \neq \gamma$

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

2nd iteration

Concrete state | Symbolic state

To reach the opposite branch..

Constraint: $\alpha > 0 \wedge \beta \neq NULL \wedge 2^* \alpha + 1 = \gamma$

Solution: $\alpha = 1$, $\beta = [\, 3 \mid NULL \,]$

Concrete state: x = 236, p = [634 | NULL]

Symbolic state:

x = α, p = β,

p->data = γ,

p->next = δ

$\alpha > 0 \wedge \beta \neq NULL \wedge 2^* \alpha + 1 \neq \gamma$

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

3rd iteration

Concrete state

x = 1, p = [3 | NULL]

Symbolic state

x = α, p = β,
p->data = γ,
p->next = δ

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

Concrete state

x = 1, p = [3 | NULL]

Symbolic state

$x = \alpha$, $p = \beta$,
$p\text{->}data = \gamma$,
$p\text{->}next = \delta$
True ($\alpha > 0$)
True ($\beta \neq NULL$)

3rd iteration

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

3rd iteration

Concrete state	Symbolic state
x = 1, p = [3 | NULL]	x = α, p = β, p->data = γ, p->next = δ True (α > 0) True (β ≠NULL) True(2* α+1 = γ)

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

3rd iteration

Concrete state	Symbolic state
x = 1, p = [3 | NULL]	x = α, p = β, p->data = γ, p->next = δ $\alpha > 0 \land \beta \neq NULL \land 2 * \alpha + 1 = \gamma \land \delta \neq \beta$

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

3rd iteration

Concrete state | Symbolic state

To reach the opposite branch..

Constraint: $\alpha > 0 \wedge \beta \neq NULL \wedge 2^* \alpha + 1 = \gamma \wedge \delta = \beta$
Solution: $\alpha = 1, \beta = [\,3 \mid \quad]$

Concrete state: $x = 1, p = [\,3 \mid NULL\,]$

Symbolic state: $x = \alpha, p = \beta,$
$p\text{->}data = \gamma,$
$p\text{->}next = \delta$

$\alpha > 0 \wedge \beta \neq NULL \wedge$
$2^* \alpha + 1 = \gamma \wedge \delta \neq \beta$

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

Concrete state

x = 1, p = [3 |]

Symbolic state

$x = \alpha, p = \beta,$
$p\text{->}data = \gamma,$
$p\text{->}next = \delta$

4^{th} iteration

Defense strategies

- **Static analysis: concolic execution**
  - Another example

```c
typedef struct cell {
    int data;
    struct cell *next;
} cell;

int foo(int v) { return 2 * v + 1; }

void test(int x, cell *p){
    if (x > 0)
        if (p != NULL)
            if (foo(x) == p->data)
                if (p->next == p)
                    assert();
    return 0;
}
```

4th iteration

Concrete state

x = 1, p = [3 |]

Crashing input!

Symbolic state

$x = \alpha$, $p = \beta$,
p->data = γ,
p->next = δ
True ($\alpha > 0$)
True ($\beta \neq$ NULL)
True ($2 * \alpha + 1 = \gamma$)
True ($\delta = \beta$)

Defense strategies

- **Static analysis: concolic execution**
  - It can do things that symbolic execution cannot

```
int foo (int v){
    return secure_hash(v);
}

void test(int x, int y){
    z = foo (y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```

Concrete state	Symbolic state
x = 22, y = 7, z = 601…129	x = α, y = β, z = hash(β)
	hash(β) ≠ α

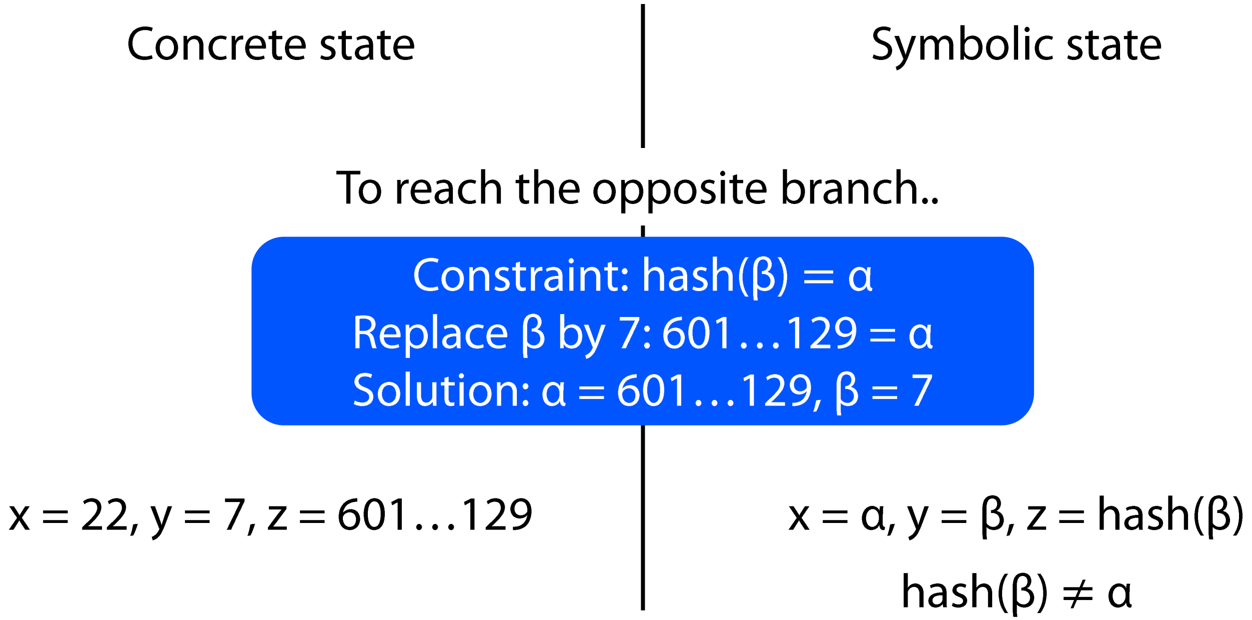
1st iteration

Defense strategies

- **Static analysis: concolic execution**
  - It can do things that symbolic execution cannot

```c
int foo (int v){
    return secure_hash(v);
}

void test(int x, int y){
    z = foo (y);
    if (z == x){
        if (x > y + 10){
            assert();
        } else { }
    }
}
```



1st iteration

Defense strategies

- **Limitation of concolic execution**

```c
int foo (int v){
    return secure_hash(v);
}

void test(int x, int y){
    if (x != y){
        if (foo(x) == foo(y)){
            assert();
        }
    }
}
```

Defense strategies

- **Limitation of concolic execution**

```
int foo (int v){
    return secure_hash(v);
}

void test(int x, int y){
    if (x != y){
        if (foo(x) == foo(y)){
            assert();
        }
    }
}
```

Concrete state	Symbolic state
x = 22, y = 7	x = α, y = β
	α ≠ β ∧ hash(α) ≠ hash(β)

To reach the opposite branch..

Constraint: $\alpha \neq \beta \wedge hash(\alpha) = hash(\beta)$
Replace α and β by 22, 7: $22 \neq 7 \wedge 438\ldots861 = 601\ldots129$
Unsatisfiable!!

1st iteration

Defense strategies

- **Static analysis: code clone detection**
  - Code clone
    - Syntactically or semantically similar code fragments
  - This can be used for 1-day vulnerability detection!
    - Idea
      - Detect code fragments that are syntactically/semantically similar to vulnerable code
    - Considerations
      - What units (e.g., function, file, line, block) will we use to detect vulnerable code clones?
      - How to create a signature?

Defense strategies

- **Static analysis: code clone detection**
  - Example: vulnerable code clone detection (using function unit)

```
DLLEXPORT unsigned char *tjLoadImage(const char *filename, int *width,
        int align, int *height, int *pixelFormat,
        int flags)
{
  int retval = 0, tempc, pitch;
  tjhandle handle = NULL;
  ...
  pitch = PAD((*width) * tjPixelSize[*pixelFormat], align);
  if ((dstBuf = (unsigned char *)malloc(pitch * (*height))) == NULL)
    _throwg("tjLoadImage(): Memory allocation failure");
  ...
}
```

Vulnerable function for CVE-2018-20330 (discovered in Libjpeg-turbo)

Defense strategies

- **Static analysis: code clone detection**
  - Example: vulnerable code clone detection (using function unit)

```c
DLLEXPORT unsigned char *tjLoadImage(const char *filename, int *width,
        int align, int *height, int *pixelFormat,
        int flags)
{
  int retval = 0, tempc, pitch;
  tjhandle handle = NULL;
  ...
  pitch = PAD((*width) * tjPixelSize[*pixelFormat], align);
  if ((dstBuf = (unsigned char *)malloc(pitch * (*height))) == NULL)
    _throwg("tjLoadImage(): Memory allocation failure");
  ...
}
```

Vulnerable function for CVE-2018-20330 (discovered in Libjpeg-turbo)

```diff
@@ -1960,7 +1960,8 @@ DLLEXPORT unsigned char *tjLoadImage(const char *filename, int *width,
                                     int align, int *height, int *pixelFormat,
                                     int flags)
 {
-  int retval = 0, tempc, pitch;
+  int retval = 0, tempc;
+  size_t pitch;
   tjhandle handle = NULL;
   tjinstance *this;
   j_compress_ptr cinfo = NULL;
@@ -2013,7 +2014,9 @@ DLLEXPORT unsigned char *tjLoadImage(const char *filename, int *width,
   *pixelFormat = cs2pf[cinfo->in_color_space];

   pitch = PAD((*width) * tjPixelSize[*pixelFormat], align);
-  if ((dstBuf = (unsigned char *)malloc(pitch * (*height))) == NULL)
+  if ((unsigned long long)pitch * (unsigned long long)(*height) >
+      (unsigned long long)((size_t)-1) ||
+      (dstBuf = (unsigned char *)malloc(pitch * (*height))) == NULL)
     _throwg("tjLoadImage(): Memory allocation failure");

   if (setjmp(this->jerr.setjmp_buffer)) {
```

Defense strategies

- **Static analysis: code clone detection**
  - Example: vulnerable code clone detection (using function unit)

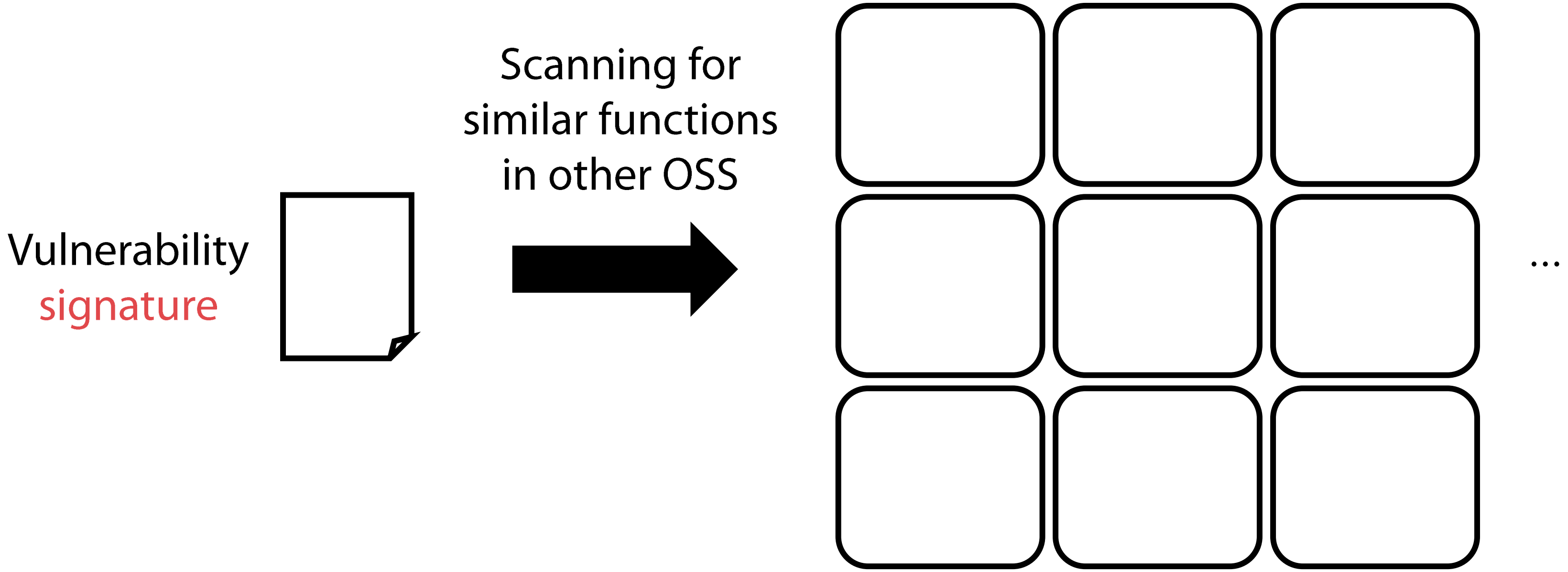
Consider the entire vulnerable function as a signature for clone detection

```c
DLLEXPORT unsigned char *tjLoadImage(const char *filename, int *width,
        int align, int *height, int *pixelFormat,
        int flags)
{
  int retval = 0, tempc, pitch;
  tjhandle handle = NULL;
  ...
  pitch = PAD((*width) * tjPixelSize[*pixelFormat], align);
  if ((dstBuf = (unsigned char *)malloc(pitch * (*height))) == NULL)
    _throwg("tjLoadImage(): Memory allocation failure");
  ...
}
```

Vulnerable function for CVE-2018-20330 (discovered in Libjpeg-turbo)

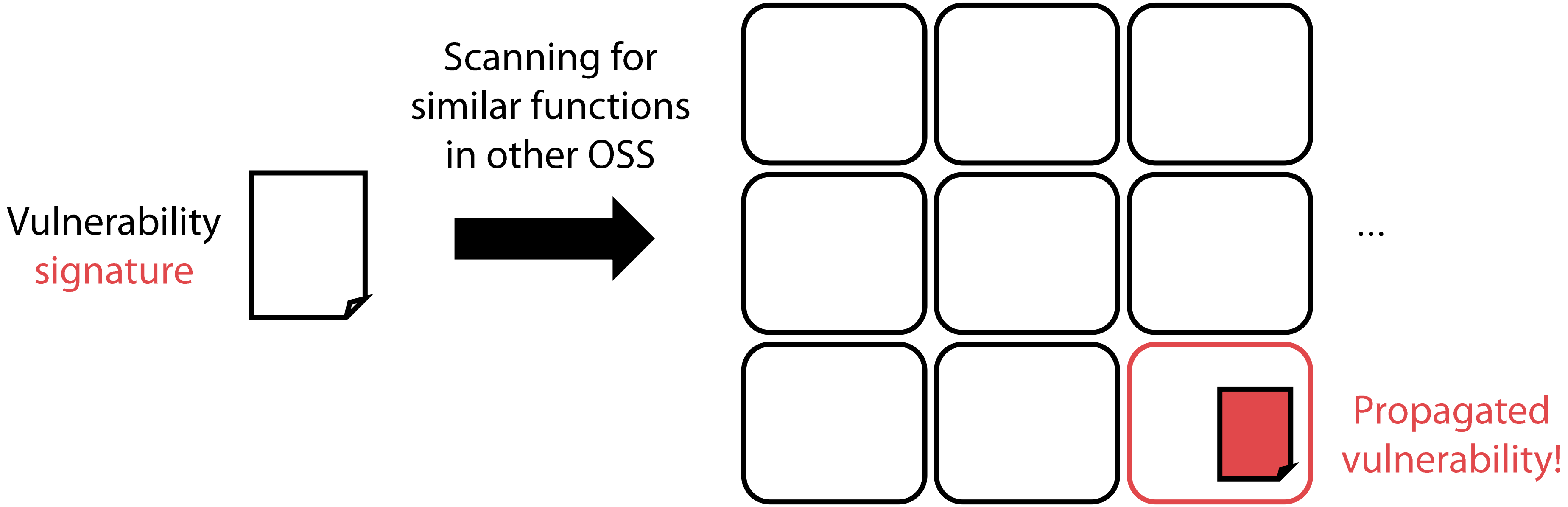
Defense strategies

- **Static analysis: code clone detection**
  - Example: vulnerable code clone detection (using function unit)



Defense strategies

- **Static analysis: code clone detection**
  - Example: vulnerable code clone detection (using function unit)



Defense strategies

- **Static analysis: code clone detection**
  - Example: vulnerable code clone detection (using function unit)

```
DLLEXPORT unsigned char* DLLCALL tjLoadImage(const char *filename, int *width,
        int align, int *height, int *pixelFormat, int flags)
{
        int retval=0, tempc, pitch;
        tjhandle handle=NULL;
        tjinstance *this;
        j_compress_ptr cinfo=NULL;
        cjpeg_source_ptr src;
        unsigned char *dstBuf=NULL;
        FILE *file=NULL;
        boolean invert;
        ...
        pitch=PAD((*width)*tjPixelSize[*pixelFormat], align);
        if((dstBuf=(unsigned char *)malloc(pitch*(*height)))==NULL)
                _throwg("tjLoadImage(): Memory allocation failure");
```

Vulnerable function discovered in the latest version of Mozjpeg (as of 2020)

Defense strategies

- **Static analysis: code clone detection**
  - Recent code clone detection techniques consider semantics more
    - Control flow, data dependency, etc.
    - Will be introduced in the "Advanced topics" lecture

Next Lecture

- **Defense strategies: dynamic analysis**