Static Program Analysis Meets Test Case Generation

Maria Christakis

MPI-SWS, Germany

About me

About me

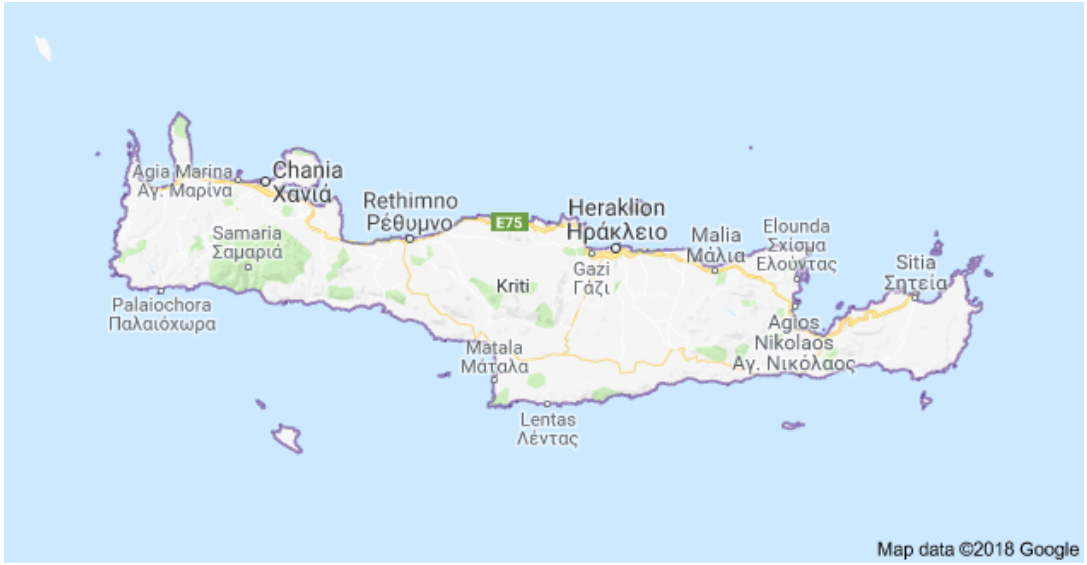
1986
Born in Crete,
Greece

About me

1986
Born in Crete, Greece

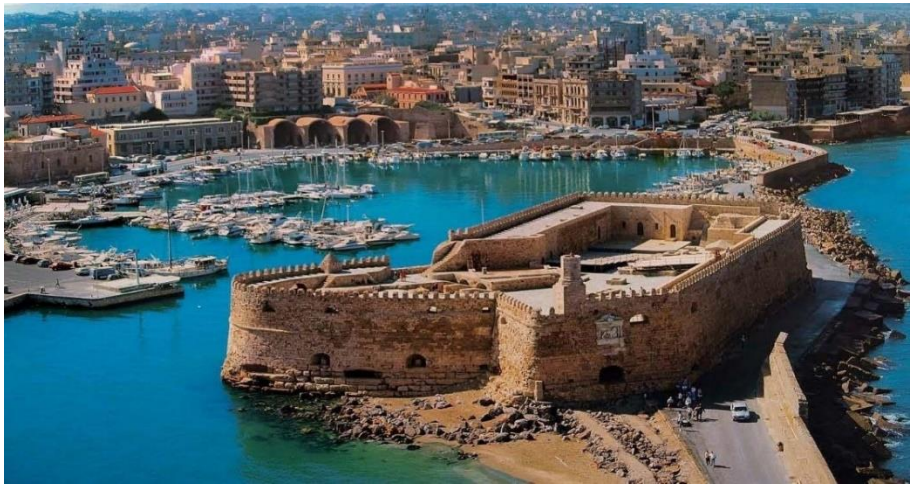
About me

1986
Born in Crete, Greece

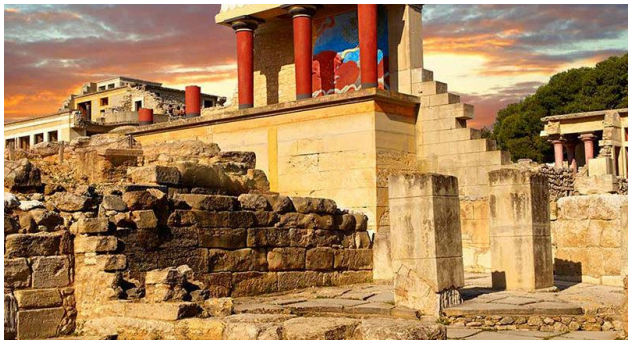


About me

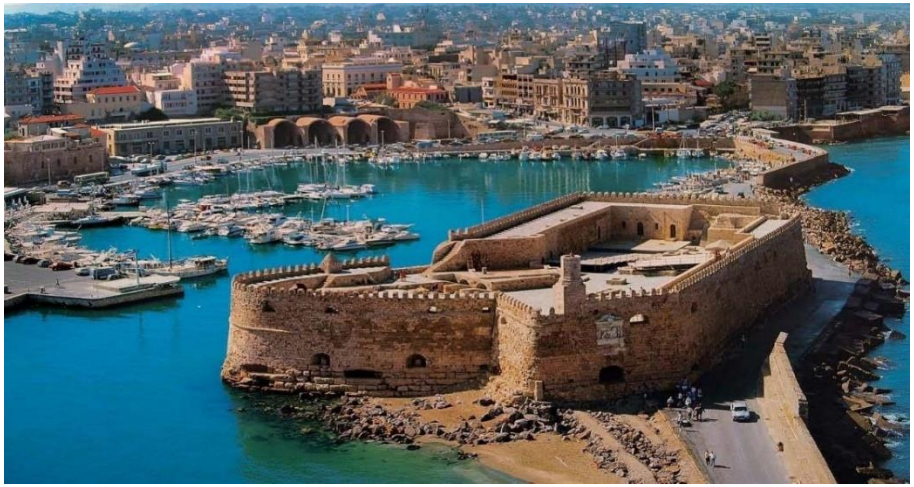
1986
Born in Crete, Greece



About me



1986
Born in Crete, Greece



About me

2003 – 2009
Dipl. in Electrical and Computer Engineering
N.T.U.A., Greece

1986
Born in Crete,
Greece

About me

2003 – 2009
Dipl. in Electrical and Computer Engineering
N.T.U.A., Greece

1986
Born in Crete, Greece

2011 – 2015
Ph.D in Computer Science
ETH Zurich, Switzerland

About me

1986
Born in Crete, Greece

2003 – 2009
Dipl. in Electrical and Computer Engineering
N.T.U.A., Greece

2011 – 2015
Ph.D in Computer Science
ETH Zurich, Switzerland

2015 – 2016
Post-doctoral researcher
MSR, Redmond, WA, U.S.A.

About me

2015 – 2016
Post-doctoral researcher
MSR, Redmond, WA, U.S.A.

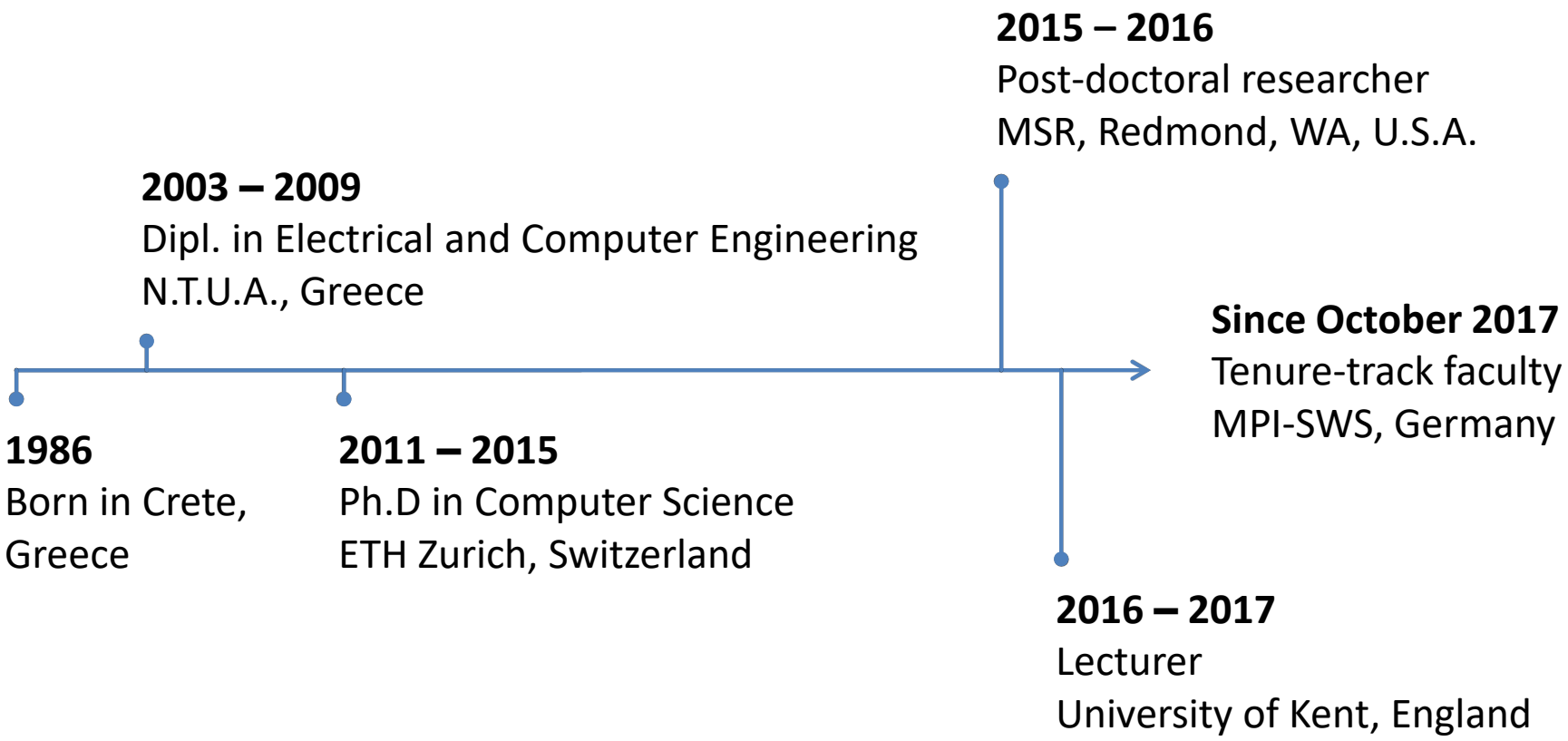
2003 – 2009
Dipl. in Electrical and Computer Engineering
N.T.U.A., Greece

1986
Born in Crete, Greece

2011 – 2015
Ph.D in Computer Science
ETH Zurich, Switzerland

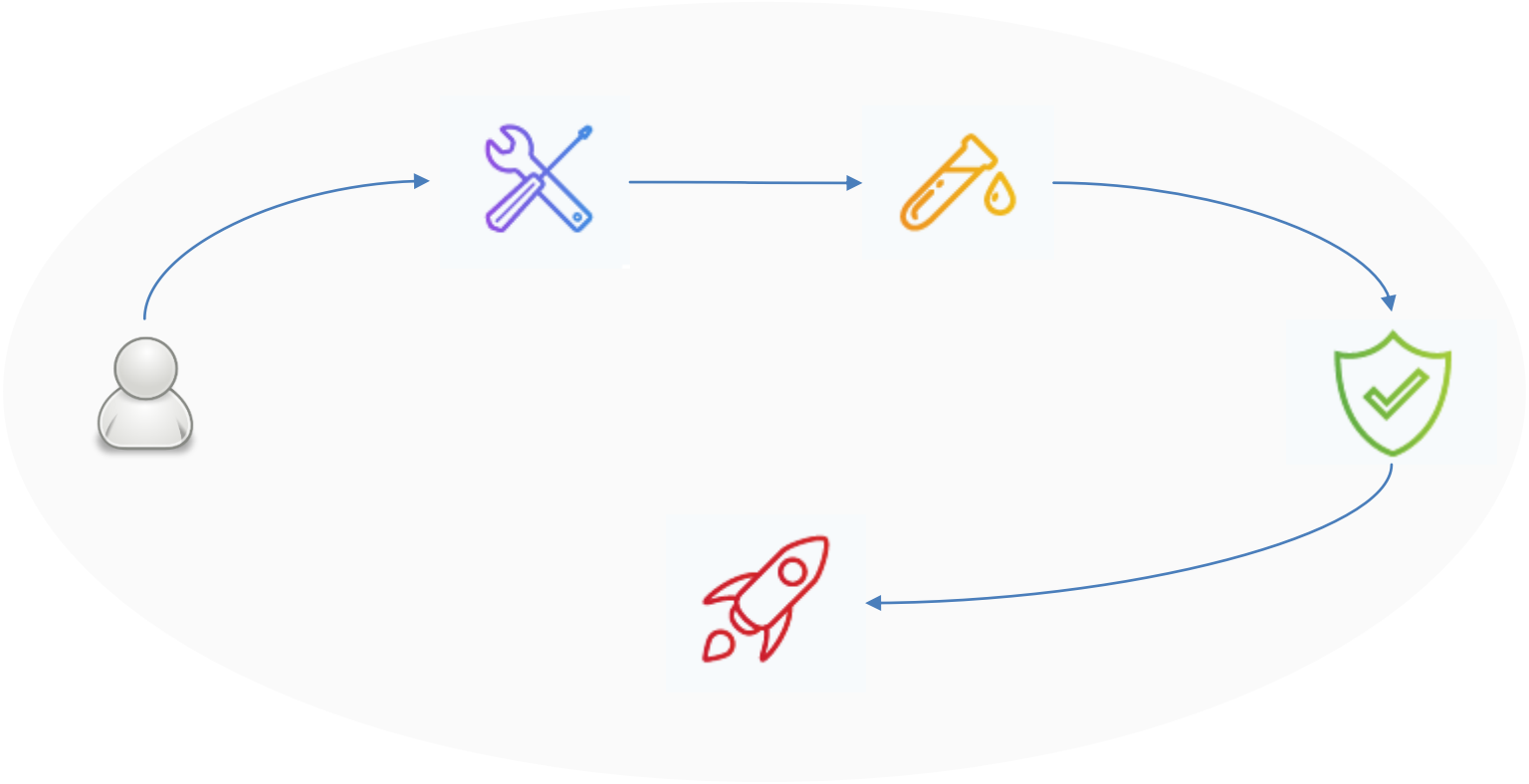
2016 – 2017
Lecturer
University of Kent, England

About me

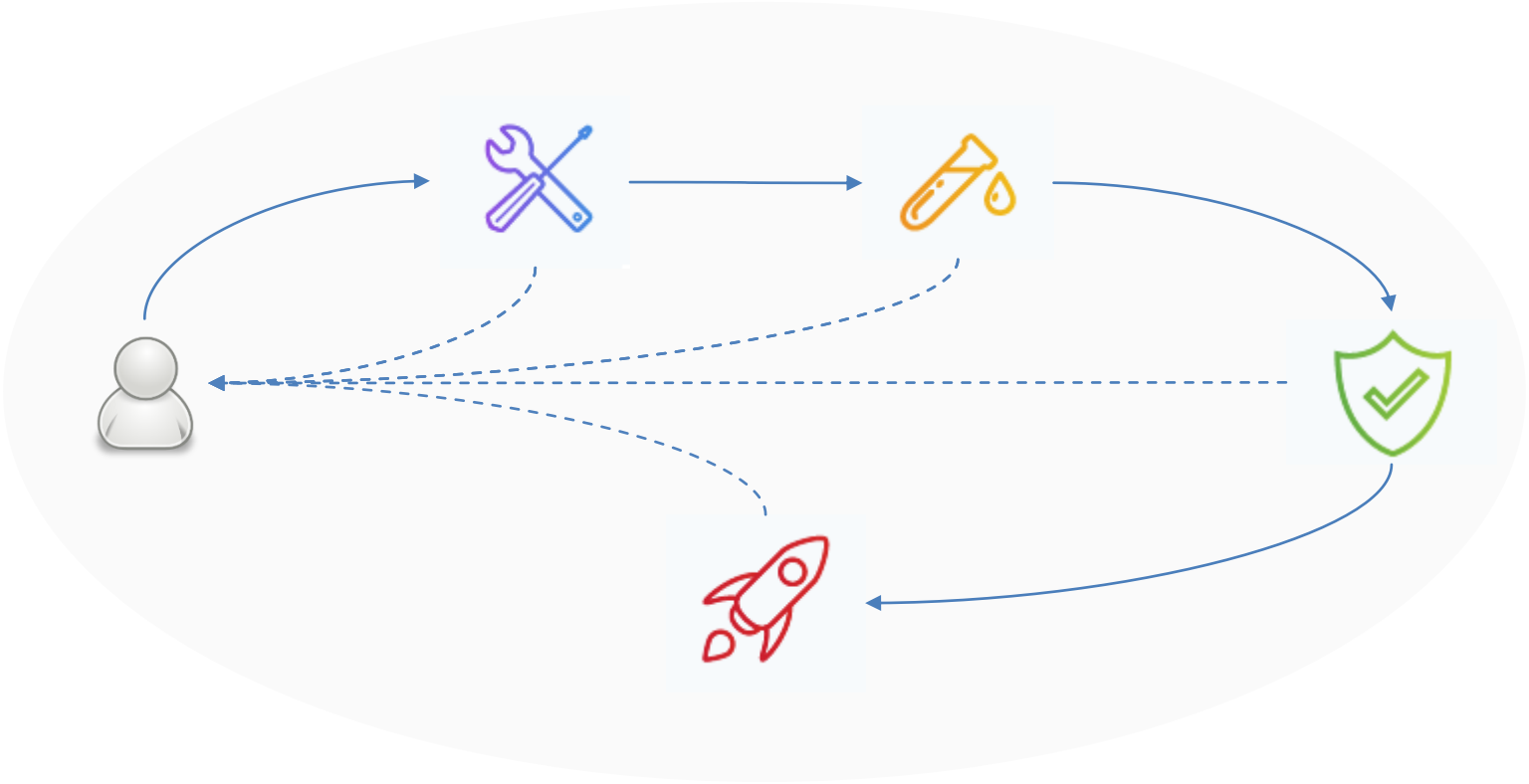

2015 – 2016
Post-doctoral researcher
MSR, Redmond, WA, U.S.A.
2003 – 2009
Dipl. in Electrical and Computer Engineering
N.T.U.A., Greece
Since October 2017
Tenure-track faculty
MPI-SWS, Germany
1986
Born in Crete, Greece
2011 – 2015
Ph.D in Computer Science
ETH Zurich, Switzerland
2016 – 2017
Lecturer
University of Kent, England

Practical formal methods

Practical formal methods



Practical formal methods



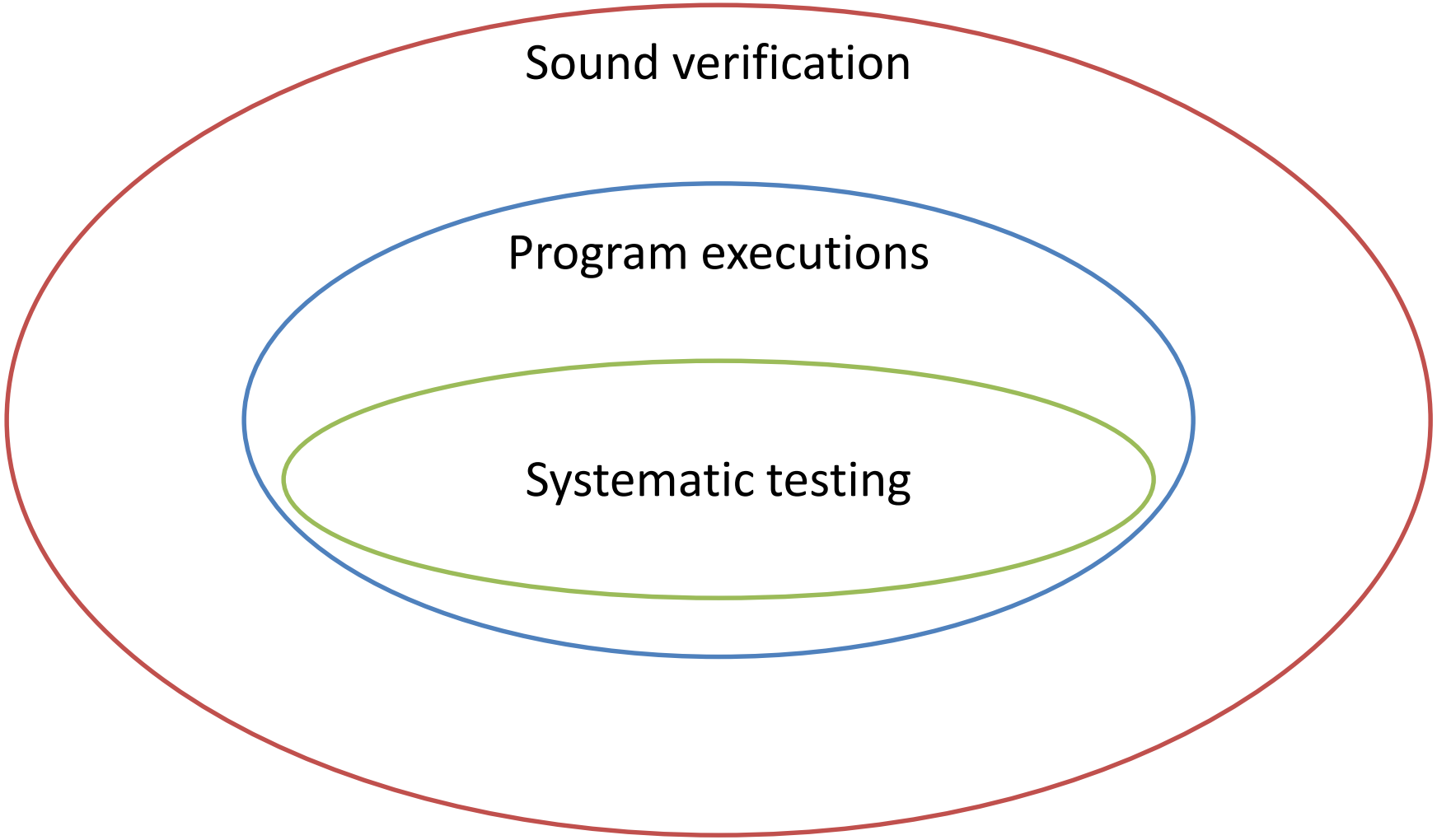
Our research mission

- Build more reliable software while increasing developer productivity
  - Formal foundation
  - Wide applicability
  - Strong practical impact

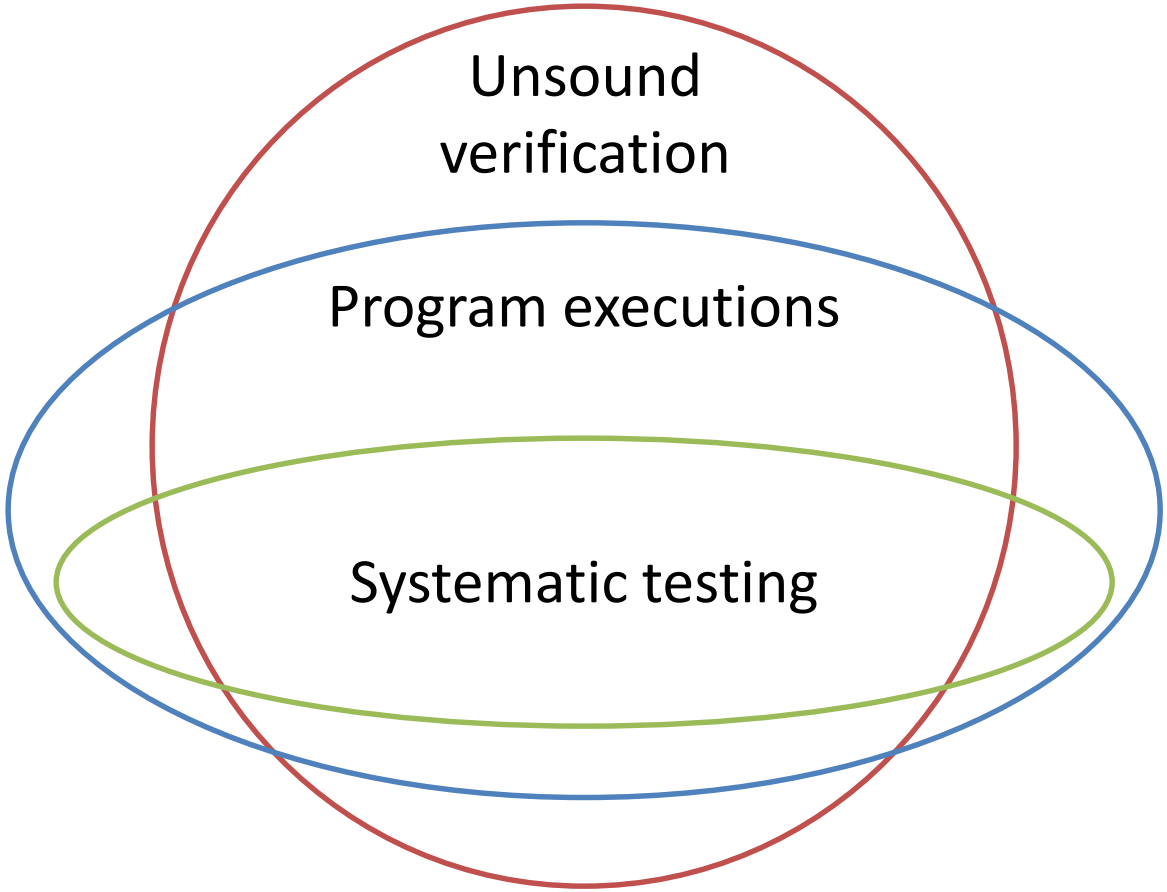
Our methodology

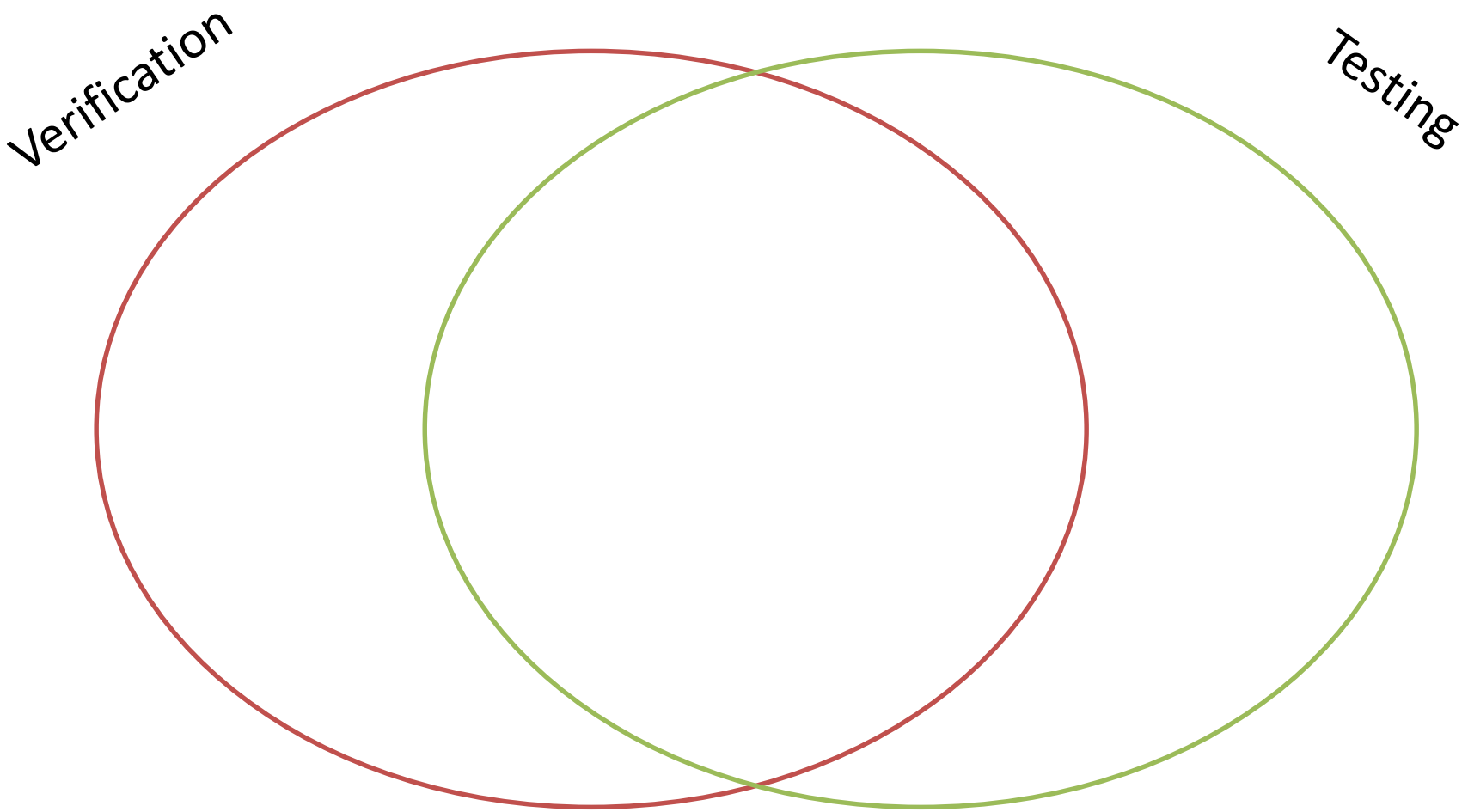
- Software engineering and formal methods
  - Automatic test generation
  - Program analysis and verification
  - Human-computer interaction
  - Empirical evaluation

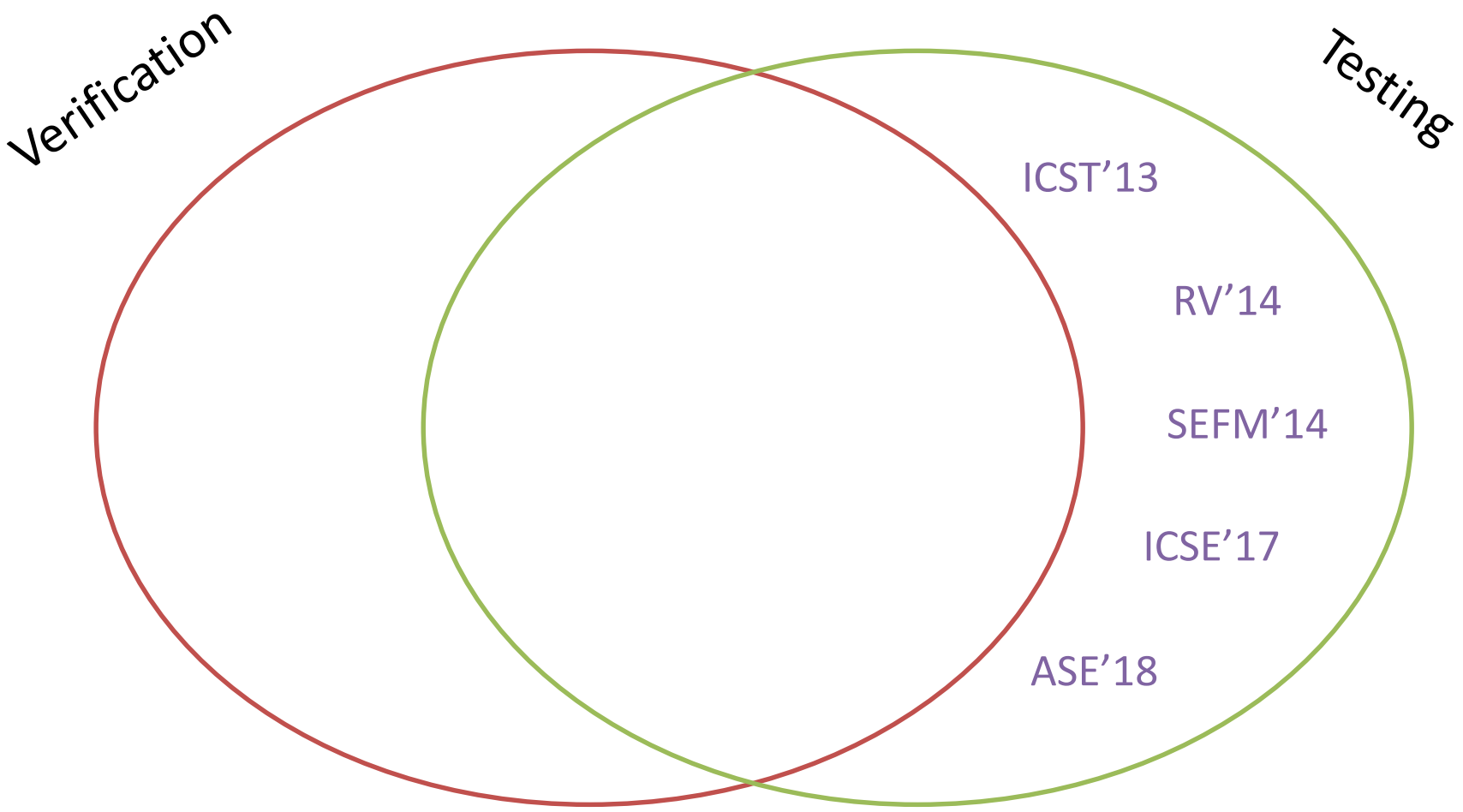
Verification and testing

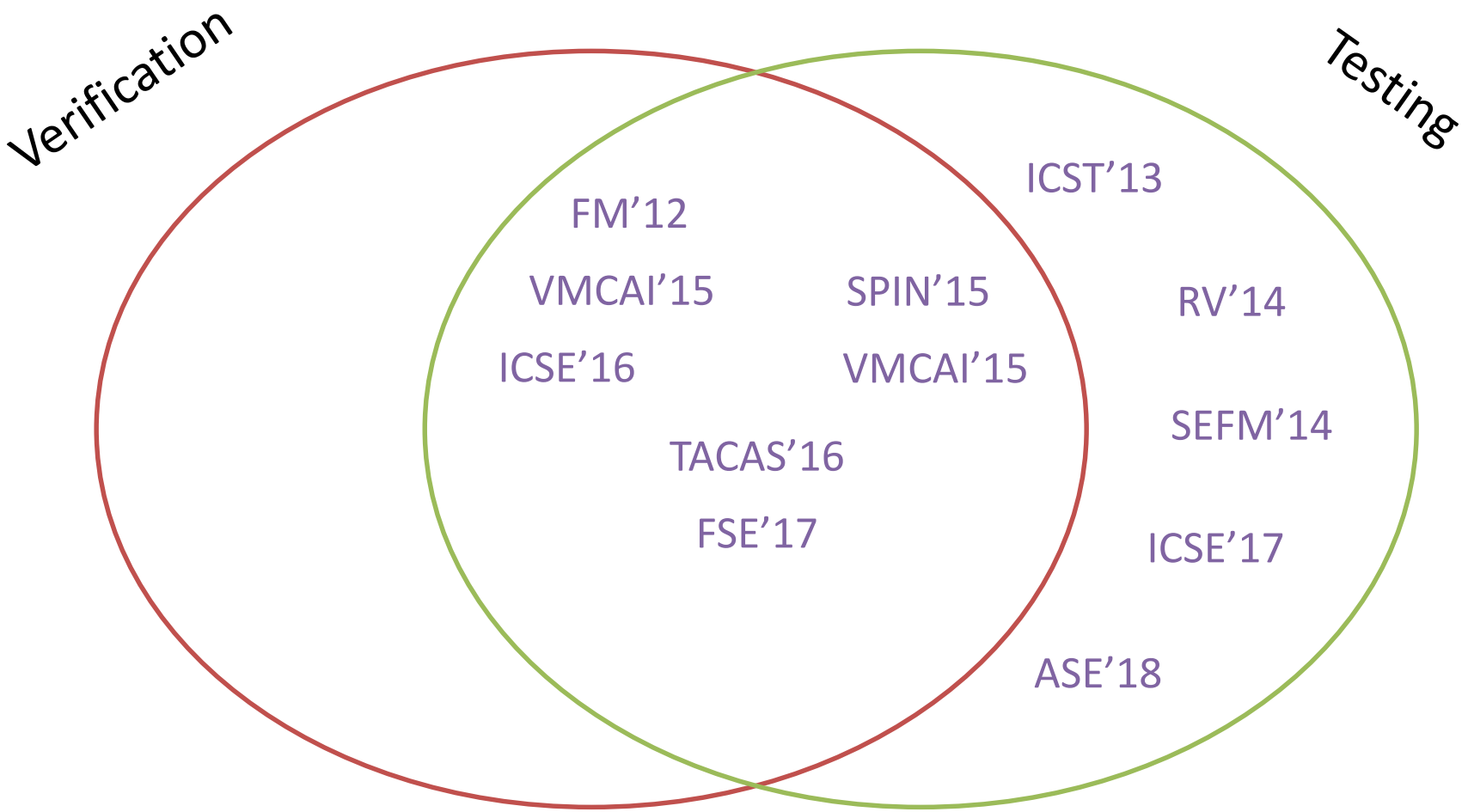


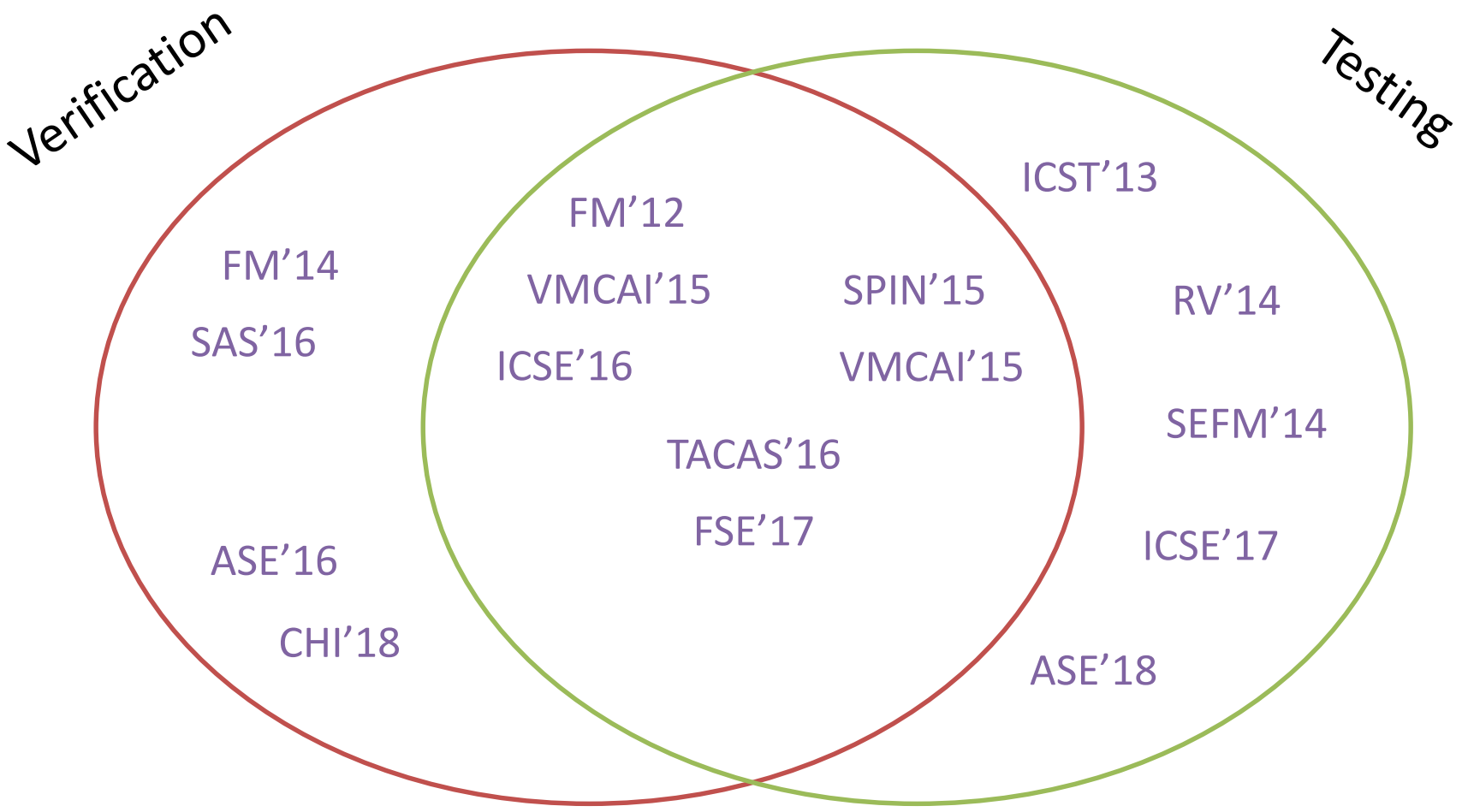
Verification and testing

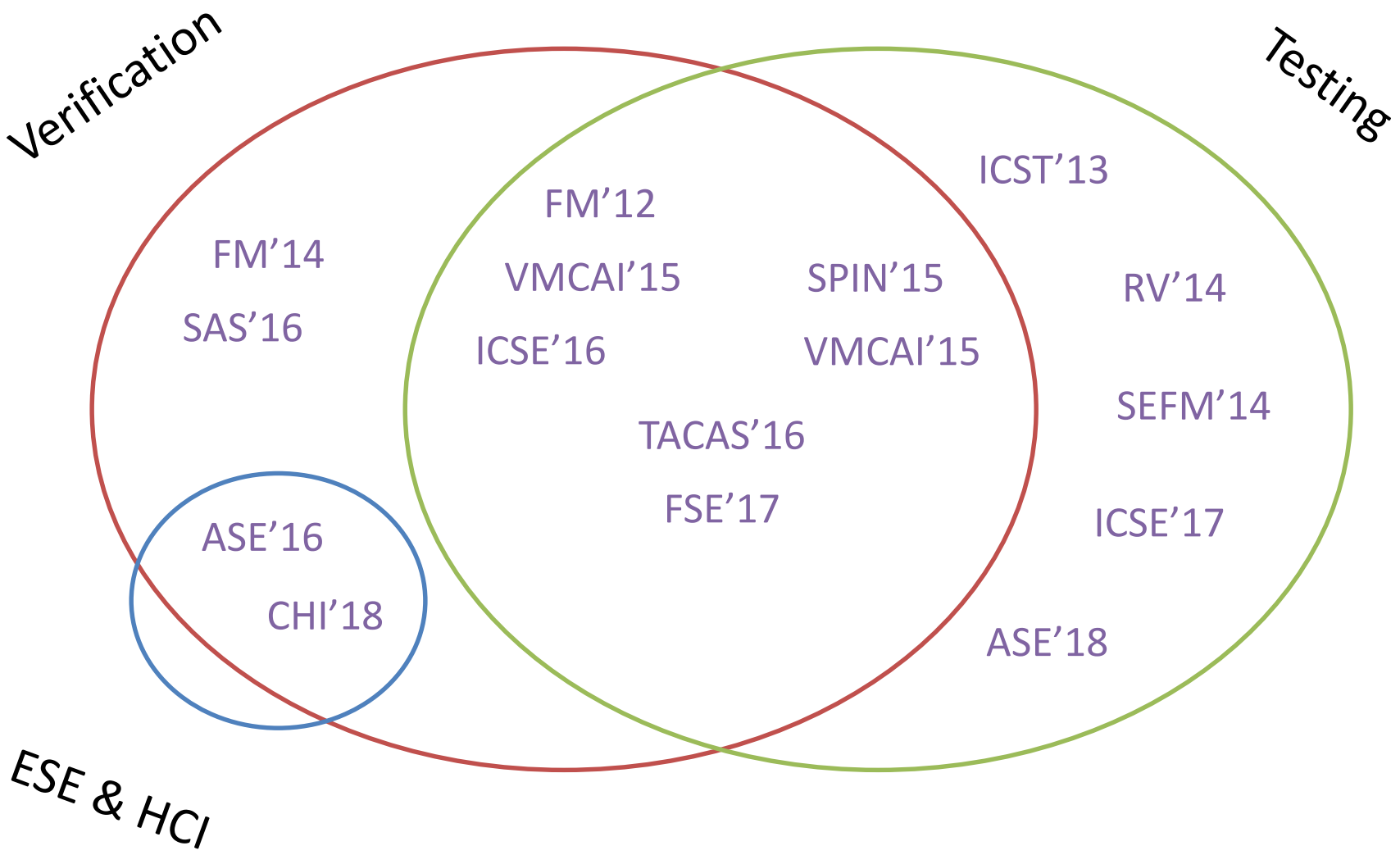



Verification
Testing


Verification
Testing
ICST'13
RV'14
SEFM'14
ICSE'17
ASE'18


Verification
Testing
FM'12
VMCAI'15
ICSE'16
SPIN'15
VMCAI'15
TACAS'16
FSE'17
ICST'13
RV'14
SEFM'14
ICSE'17
ASE'18


Verification
Testing
FM'14
SAS'16
ASE'16
CHI'18
FM'12
VMCAI'15
ICSE'16
SPIN'15
VMCAI'15
TACAS'16
FSE'17
ICST'13
RV'14
SEFM'14
ICSE'17
ASE'18


Verification
Testing
ESE & HCI
FM'14
SAS'16
FM'12
VMCAI'15
ICSE'16
SPIN'15
VMCAI'15
TACAS'16
FSE'17
ICST'13
RV'14
SEFM'14
ICSE'17
ASE'18
ASE'16
CHI'18

Static Program Analysis Meets *Test Case Generation*
Lecture 1

Textbook:
Test Case Generation

Test case generation

- Detects bugs and security vulnerabilities
- Improves software quality
- Is cumbersome when done manually

Test case generation

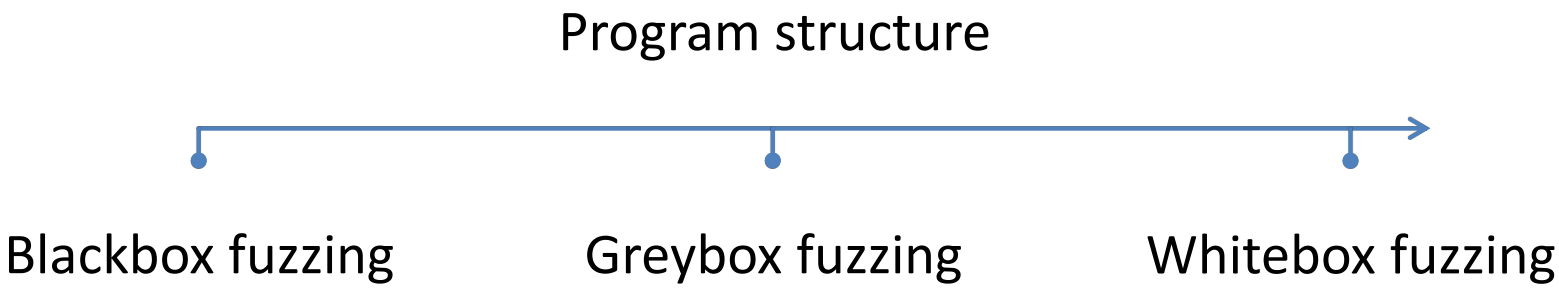
- Detects bugs and security vulnerabilities
- Improves software quality
- Is cumbersome when done manually

Characteristics

- Does not cover all program executions
- Does not generate false positives

Test case generation

- Detects bugs and security vulnerabilities
- Improves software quality
- Is cumbersome when done manually



Textbook:
Blackbox Fuzzing

Blackbox fuzzing

- Also called random testing

Blackbox fuzzing

- Also called random testing

Algorithm

- Generates random inputs to a program
- Runs the program with these inputs
- Checks whether any bugs were found

Blackbox fuzzing: Discussion

Blackbox fuzzing: Discussion

- Practical
- Very efficient

Blackbox fuzzing: Discussion

- Practical
- Very efficient

but…

- Very ineffective in exploring new paths

Exercise

What is the probability of a blackbox fuzzer detecting the bug?

```
void foo(int x) {
  if x == 42 { bug(); }
}
```

Exercise

What is the probability of a blackbox fuzzer detecting the bug?

```
void foo(int x) {
  if x == 42 { bug(); }
}
```

$2^{32}!$

Fun:
A General Framework for Dynamic Stub Injection

Background: Stub testing

- A technique to simulate dependencies of an application

Background: Stub testing

- A technique to simulate dependencies of an application

1. Decide which functions to replace by stubs
2. For each such function, implement one or more stubs
3. For each call to such functions, decide whether to call a stub
4. For each chosen call, perform the stub injection

Our framework

- Provides a domain specific language
  - Arbitrary stub behavior and flexible injection strategies
  - Stub reuse across programming languages
- Dynamically alters any executable program
  - No need for source code
- Detected 176 crashes in 10 commercial applications

A stub injection rule

```
kernel32.dll!GetModuleFileNameW(
    hModule, lpFileName, nSize)
```

A stub injection rule

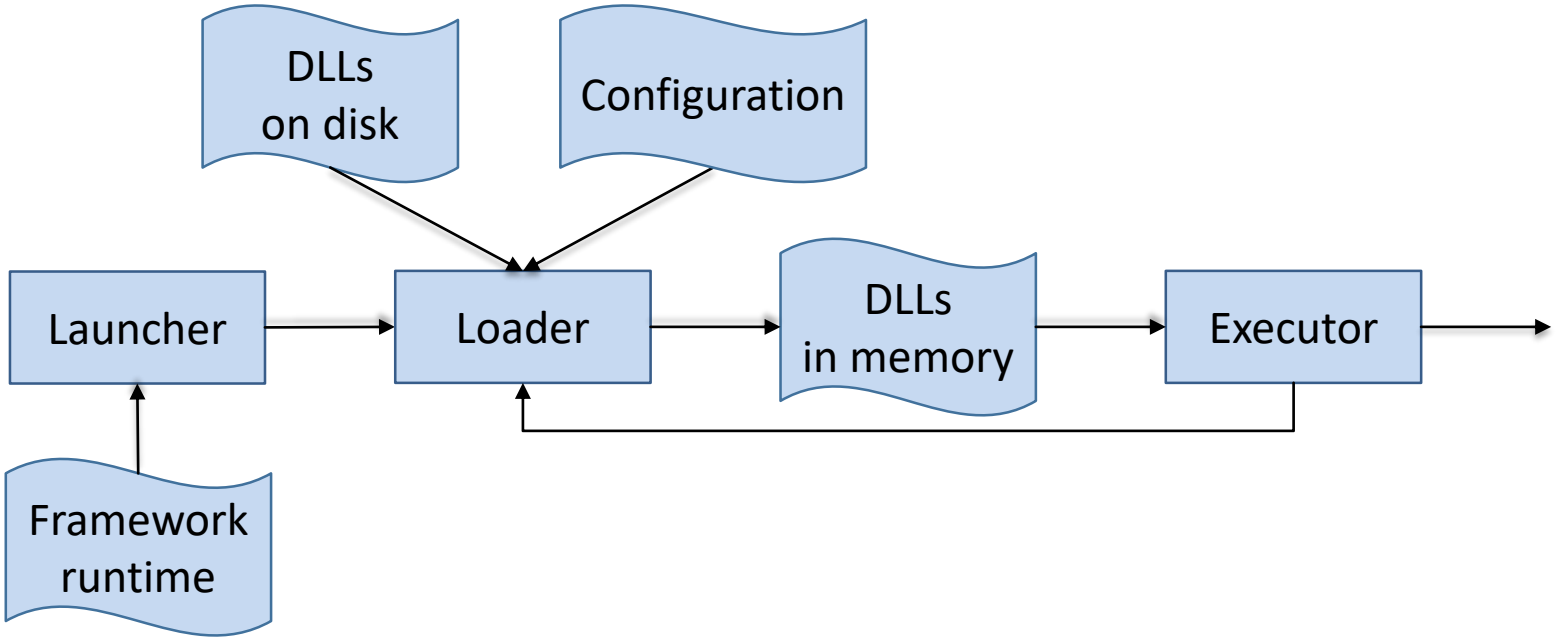
```
rule kernel32.dll!GetModuleFileNameW(
          hModule, lpFileName, nSize)
  frequency every(2);
  after {
    if (nSize < 32767) {
      SetLastError(ERROR_INSUFFICIENT_BUFFER);
      result = nSize;
    }
  }
```

A stub injection rule

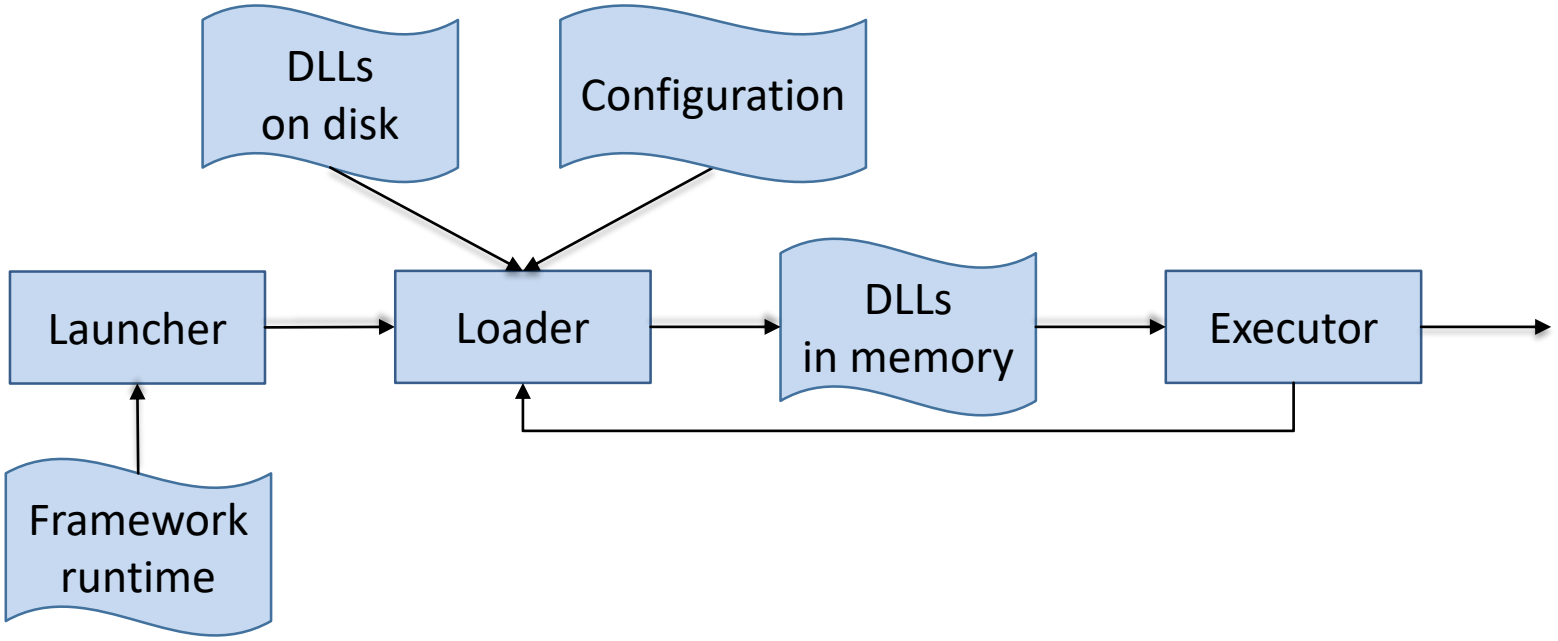
```
rule kernel32.dll!GetModuleFileNameW(
          hModule, lpFileName, nSize)
  frequency every(2);
  after {
    if (nSize < 32767) {
      SetLastError(ERROR_INSUFFICIENT_BUFFER);
      result = nSize;
    }
  }
```

- A stub can have non-trivial behavior
- A rule is language independent
- A stub can replace or augment the function behavior

The stub injection workflow



The stub injection workflow



- The instrumentation happens on the deployed code
- No need to know in advance which DLLs will be loaded
- Complex invocation schemes are easily handled

Experiment design: Fault injection

- Applications under test
- Test scenarios
- Stub injection rules

Experiment design: Fault injection

- Applications under test
- Test scenarios
- Stub injection rules

Internet Explorer Microsoft Excel

Microsoft Notepad

Microsoft OneNote Launcher

Microsoft Outlook Microsoft Paint

Microsoft PowerPoint

Microsoft Publisher

Microsoft Visio Microsoft Word

Notepad++ Skype for Business

Experiment design: Fault injection

- Applications under test
- Test scenarios

Start → Wait → Close

- Stub injection rules

Design: Stub injection rules

- Target functions
  - Requesting new resources
  - Accessing external resources
- Injection strategy
- Stub behavior

Design: Stub injection rules

- Target functions
  - Requesting new resources
  - Accessing external resources
- Injection strategy
- Stub behavior

Microsoft Notepad Notepad++

Design: Stub injection rules

- Target functions
  - Requesting new resources
  - Accessing external resources
- Injection strategy
- Stub behavior

Always
Once
EveryOtherCall
FiftyFifty

Design: Stub injection rules

- Target functions
  - Requesting new resources
  - Accessing external resources
- Injection strategy
- Stub behavior

MSDN StackOverflow

Result highlights

- 160 rules (40 target functions x 4 injection strategies)
- 1,920 runs (160 rules x 12 applications)

Result highlights

- 160 rules (40 target functions x 4 injection strategies)
- 1,920 runs (160 rules x 12 applications)
- 176 crashes in 10 applications
- 390,571 instrumented calls
- 140,621 stubbed calls

Results: Injection strategies

- All injection strategies detected crashes
- Always detects the most crashes, and Once the least
- Always may inject fewer stubs than EveryOtherCall and FiftyFifty
- Once detects by far the most crashes per 1,000 stubbed calls

Results: From crashes to bugs

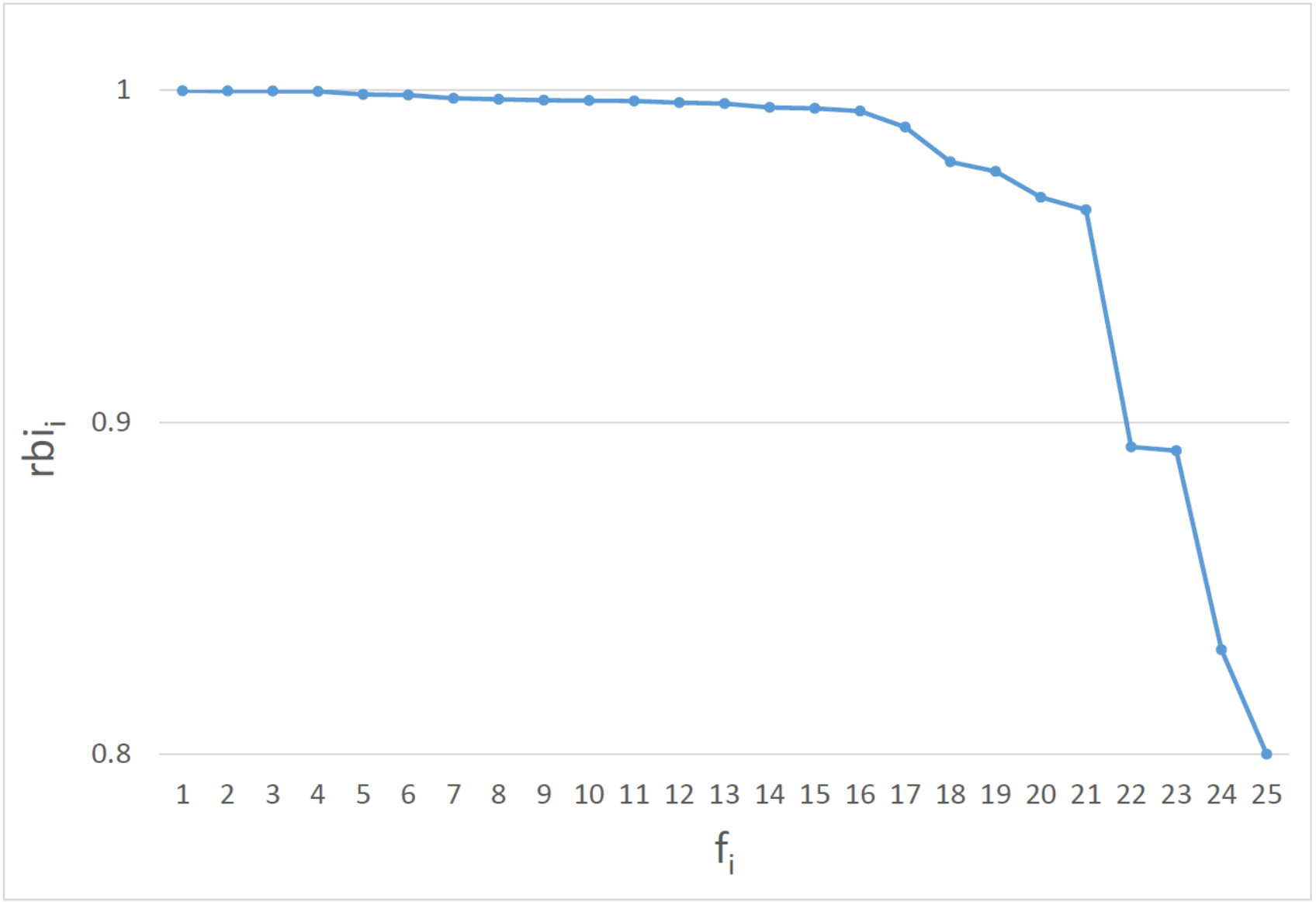
$$rbi_i = 1 - \frac{\text{number of crashes by stubbing } f_i}{\text{number of stubbed calls to } f_i}$$

Results: From crashes to bugs

$$rbi_i = 1 - \frac{\text{number of crashes by stubbing } f_i}{\text{number of stubbed calls to } f_i}$$

- Rules with a low rbi_i should be revised
- Developers should prioritize crashes by rules with high rbi_i
- New applications should be tested with rules with high rbi_i

Results: The real-bug indicator



Results: Bug in Excel

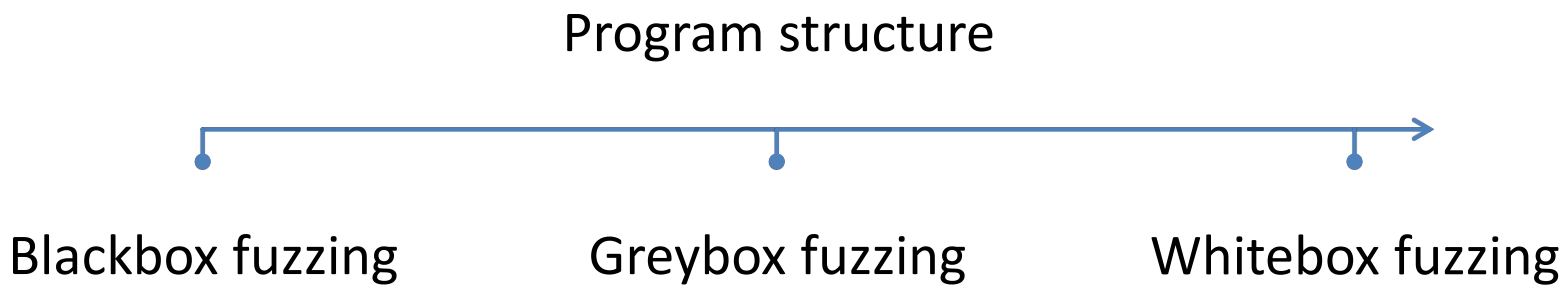
```
for (int c = 0; c < 2; c++) {
  error = RegQueryValueExW(key, /* ... */,
                           buffer, &buffer_size);
  if (ERROR_SUCCESS == error) {
    value_ptr = buffer;
    break;
  }
  else if (ERROR_MORE_DATA == error) {
    buffer.resize(buffer_size)
  }
}
assert(error == ERROR_SUCCESS);
```

Results: Rule for bug in Excel

```
rule ADVAPI32.dll!RegQueryValueExW(hKey,
            /* ... */, lpData, lpcbData)
  before {
    if (last_key != hKey) {
      last_key = hKey;
      counter = 0;
    } else {
      counter++;
    }
  }
  after {
    if (counter < 2) {
      (*lpcbData)++;
      SetLastError(ERROR_MORE_DATA);
      result = ERROR_MORE_DATA;
    }
  }
```

Textbook:
Greybox Fuzzing

Greybox fuzzing

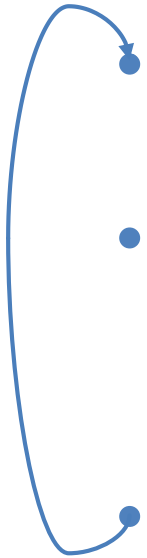


Greybox fuzzing: Algorithm

- Runs the program with an input
- Computes the identifier of the explored path
- Mutates the input and runs the program again
- Adds the new input to the test suite,
  if a new path was explored
- Picks a new input from the test suite

Greybox fuzzing: Algorithm

- Runs the program with an input
- Computes the identifier of the explored path
- Mutates the input and runs the program again
- Adds the new input to the test suite,
  if a new path was explored
- Picks a new input from the test suite



Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

a = -1, b = 0, c = -5

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
a =  0, b = 0, c = -5
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
a =  0, b = 0, c = -5
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
a =  0, b = 0, c =  5
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
a =  0, b = 0, c =  5
a = -1, b = 7, c = -5
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
a =  0, b = 0, c =  5
a = -1, b = 7, c = -5
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
a =  0, b = 0, c =  5
a = -1, b = 7, c = -5
a = -1, b = 7, c = 64
```

Greybox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b = 0, c = -5
a =  0, b = 0, c =  5
a = -1, b = 7, c = -5
a = -1, b = 7, c = 64
```

4 out of 5 paths
covered in 12 hours!

Greybox fuzzing: Discussion

Greybox fuzzing: Discussion

- Very practical
- Efficient

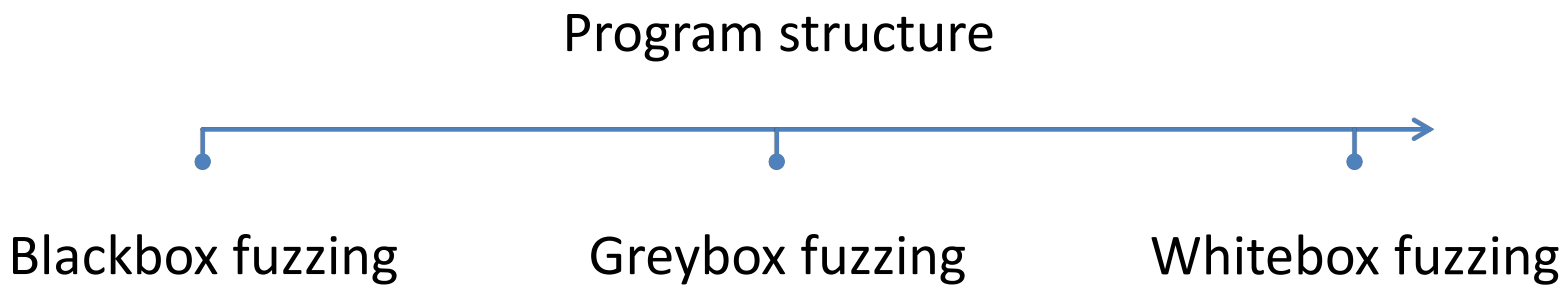
Greybox fuzzing: Discussion

- Very practical
- Efficient

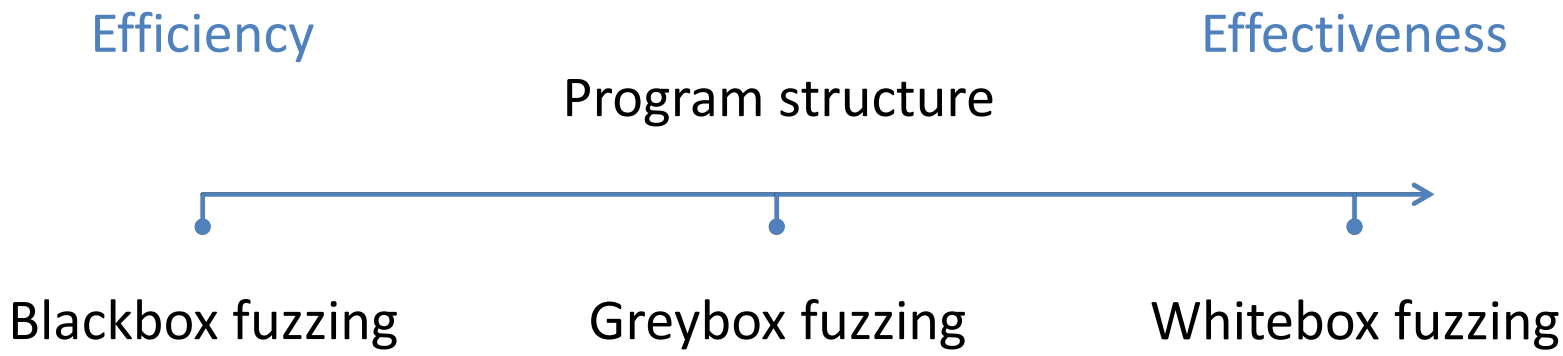
but…

- Ineffective in exploring new paths
  (Inputs are still randomly mutated)

Greybox fuzzing: Discussion

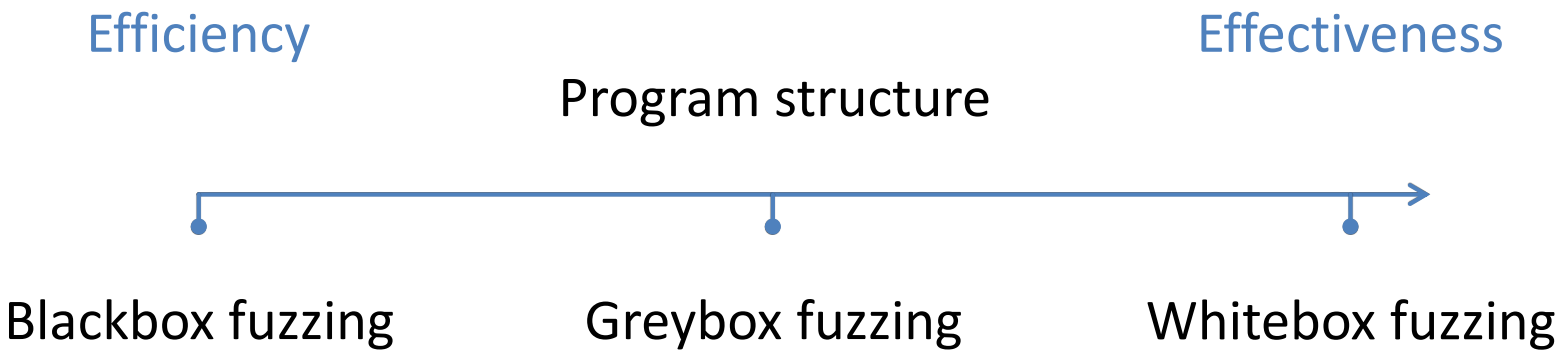


Greybox fuzzing: Discussion

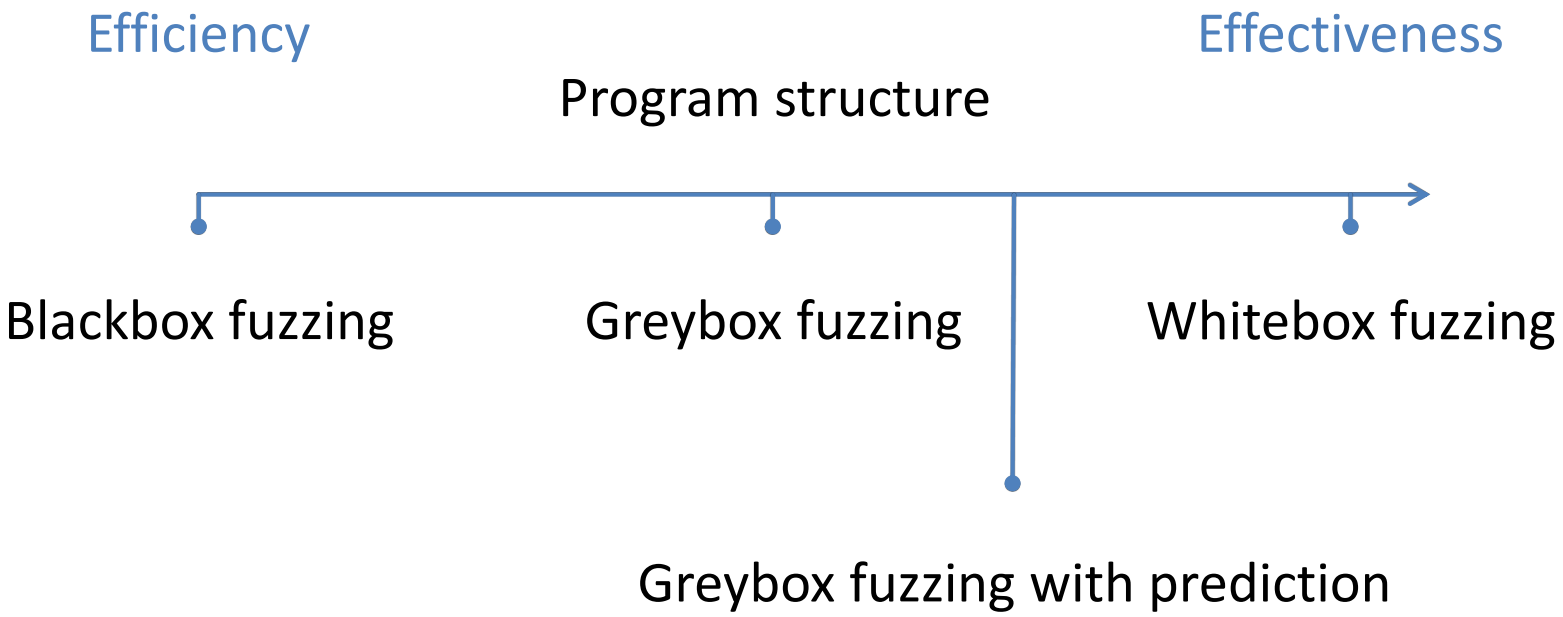


Fun:
Greybox Fuzzing with Cost-Directed Input Prediction

Greybox fuzzing



Greybox fuzzing: Input prediction



Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d

d

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d

3 - b

d

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d

3 - b

b - 2

d

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d

3 - b

b - 2

d

1) a = -1, b = 0, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d

3 - b

1) a = -1, b = 0, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6

3 - b

1) a = -1, b = 0, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6

3 - b c1:3

1) a = -1, b = 0, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

`1 - d` c1:6

`3 - b` c1:3

```
1) a = -1, b =  0, c = -5
2) a = -1, b = -3, c = -5
```

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6

3 - b c1:3

1) a = -1, b = 0, c = -5
2) a = -1, b = -3, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6 c2:9

3 - b c1:3

1) a = -1, b = 0, c = -5
2) a = -1, b = -3, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6 c2:9

3 - b c1:3 c2:6

```
1) a = -1, b =  0, c = -5
2) a = -1, b = -3, c = -5
```

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6 c2:9

3 - b c1:3 c2:6

1) a = -1, b = 0, c = -5

~~2) a = -1, b = -3, c = -5~~

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

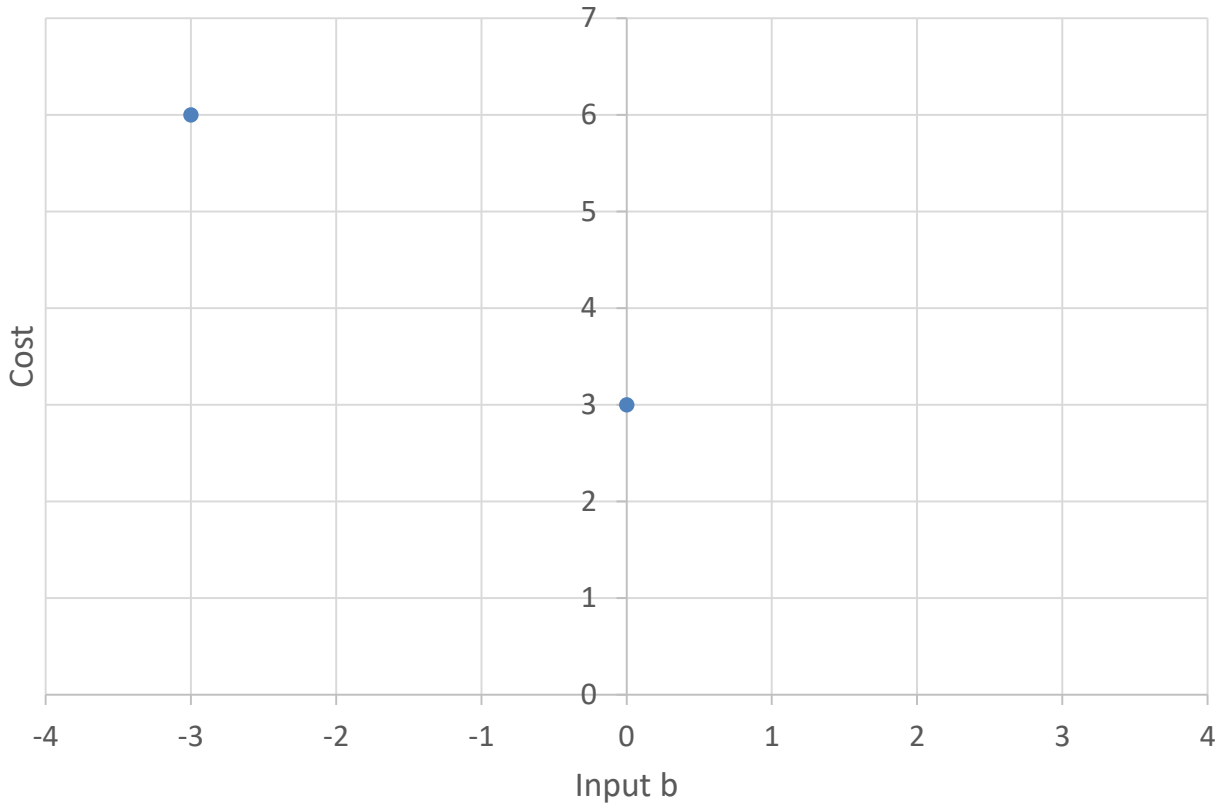
1 - d c1:6 c2:9

3 - b c1:3 c2:6

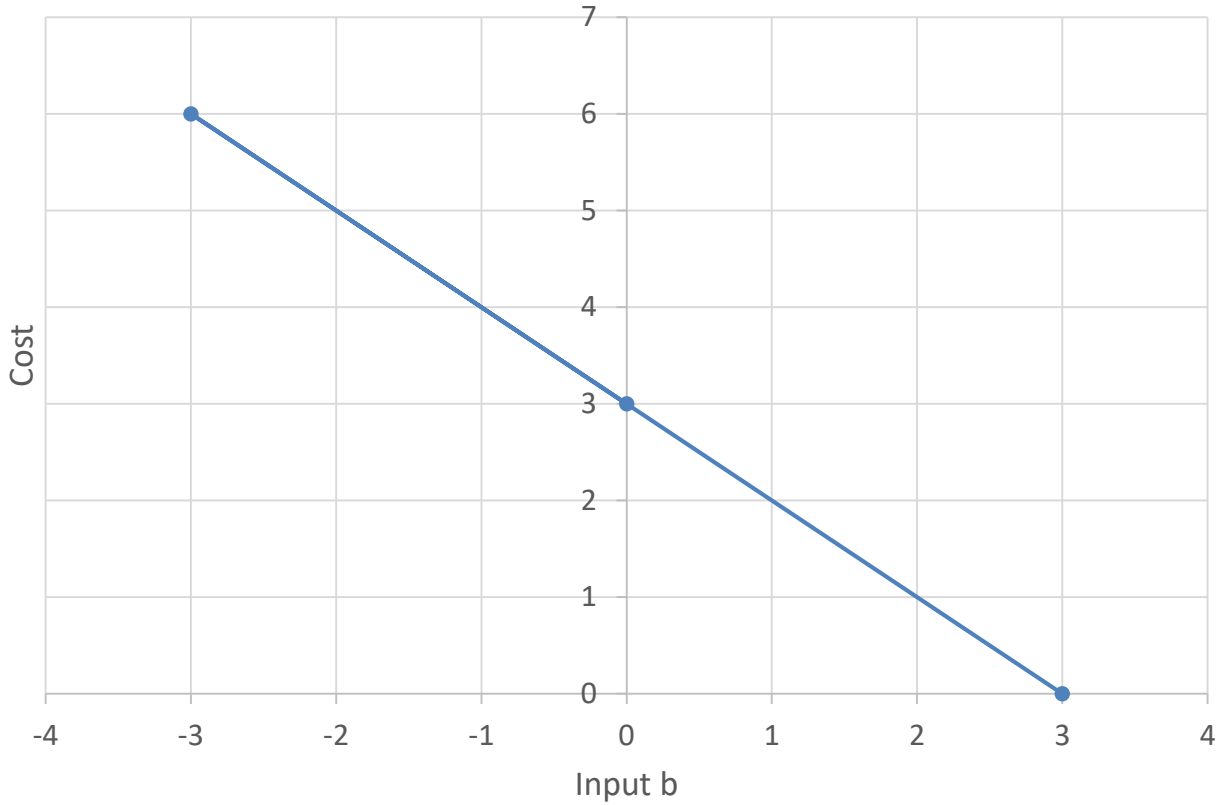
1) a = -1, b = 0, c = -5

~~2) a = -1, b = -3, c = -5~~

Input prediction: Example



Input prediction: Example



Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6 c2:9

3 - b c1:3 c2:6

1) a = -1, b = 0, c = -5

~~2) a = -1, b = -3, c = -5~~

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6 c2:9

3 - b c1:3 c2:6

1) a = -1, b = 0, c = -5

~~2) a = -1, b = -3, c = -5~~

3) a = -1, b = 3, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6 c2:9

3 - b c1:3 c2:6

1) a = -1, b = 0, c = -5

~~2) a = -1, b = -3, c = -5~~

3) a = -1, b = 3, c = -5

Input prediction: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

1 - d c1:6 c2:9

3 - b c1:3 c2:6

1) a = -1, b = 0, c = -5

~~2) a = -1, b = -3, c = -5~~

3) a = -1, b = 3, c = -5

Only 8 tests
for full path coverage!

Evaluation highlights

- Run on 26 applications, totaling 12’000 LOC
- Increased path coverage by up to 4X
- Found up to 77% more vulnerabilites, often orders-of-magnitude faster

Input prediction: Discussion

Input prediction: Discussion

- Very practical
- Efficient
- Effective in exploring new paths

Input prediction: Discussion

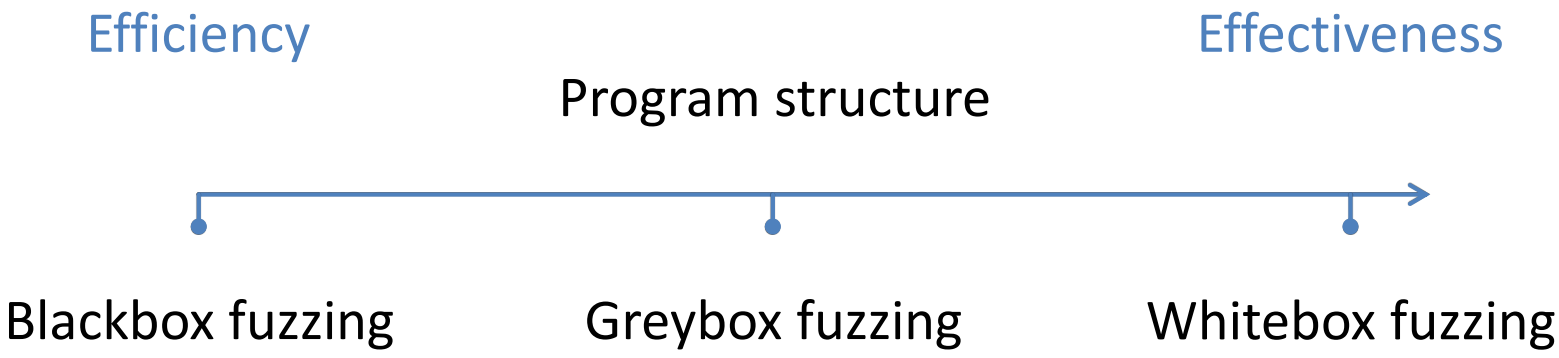
- Very practical
- Efficient
- Effective in exploring new paths

but…

- Approximate!

Textbook:
Whitebox Fuzzing

Whitebox fuzzing



Whitebox fuzzing

- Also called dynamic symbolic execution

Whitebox fuzzing

- Also called dynamic symbolic execution

Algorithm

- Runs the program concretely and symbolically
- Collects constraints on inputs at branches
- Uses a constraint solver to generate new inputs

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

a = -1, b = 0, c = -5

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

a = -1, b = 0, c = -5

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

a = -1, b = 0, c = -5

$b + c < 1 \wedge b < 3$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

a = -1, b = 0, c = -5

$b + c < 1 \wedge b \geq 3$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
```

$b + c < 1 \wedge b \geq 3$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
```

$b + c < 1 \wedge b \geq 3$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
```

$$b + c < 1 \wedge b \geq 3 \wedge \neg(a = 42)$$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
```

$b + c < 1 \wedge b \geq 3 \wedge a = 42$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
```

$b + c < 1 \wedge b \geq 3 \wedge a = 42$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
```

$b + c < 1 \wedge b \geq 3 \wedge a = 42$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
```

$b + c \geq 1$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
a = 42, b =   1, c =  0
```

$b + c \geq 1$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
a = 42, b =   1, c =  0
```

$b + c \geq 1$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
a = 42, b =   1, c =  0
```

$b + c \geq 1 \wedge c < 42$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
a = 42, b =   1, c =  0
```

$b + c \geq 1 \wedge c \geq 42$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
a = 42, b =   1, c =  0
a = 42, b = -41, c = 42
```

$b + c \geq 1 \wedge c \geq 42$

Whitebox fuzzing: Example

```
int bar(int a, int b, int c) {
  var d = b + c;
  if d < 1 {
    if b < 3 {
      return 1;
    }
    if a == 42 {
      return 2;
    }
    return 3;
  } else {
    if c < 42 {
      return 4;
    }
    return 5;
  }
}
```

```
a = -1, b =   0, c = -5
a = -1, b =   3, c = -3
a = 42, b =   3, c = -3
a = 42, b =   1, c =  0
a = 42, b = -41, c = 42
```

$b + c \geq 1 \wedge c \geq 42$

Whitebox fuzzing: Discussion

Whitebox fuzzing: Discussion

- Practical
- Inefficient

Whitebox fuzzing: Discussion

- Practical
- Inefficient

but…

- Very effective in exploring new paths

Exercise

Encode the path constraints to determine the feasible paths.

```
int bar(int a, int b) {
  var c = a * b;
  if c >= 0 && b > 0 {
    var d = c / 3 + a;
    if d < 0 {
      return 1;
    }
    return 2;
  }
  return 3;
}
```

Exercise

Encode the path constraints to determine the feasible paths.

```
int bar(int a, int b) {
  var c = a * b;
  if c >= 0 && b > 0 {
    var d = c / 3 + a;
    if d < 0 {
      return 1;
    }
    return 2;
  }
  return 3;
}
```

2 feasible paths!

Exercise

How many paths are feasible?

```
void bar(int c) {
  var i = 0;
  while i < c {
    i++;
  }
}
```

Exercise

How many paths are feasible?

```
void bar(int c) {
  var i = 0;
  while i < c {
    i++;
  }
}
```

2,147,483,647 + 1!

Fun:
Proving Memory Safety of the ANI Image Parser

Contributions

- First application of systematic testing for proving attacker memory safety
- of a complex and security-critical image parser
- using only compositional exhaustive testing
- How big is the gap between systematic testing and verification?

Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses

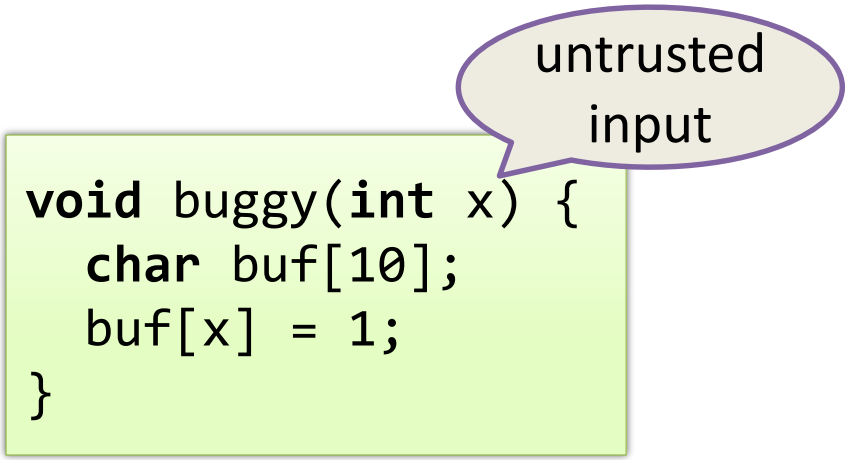
Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses

```
void buggy(int x) {
  char buf[10];
  buf[x] = 1;
}
```

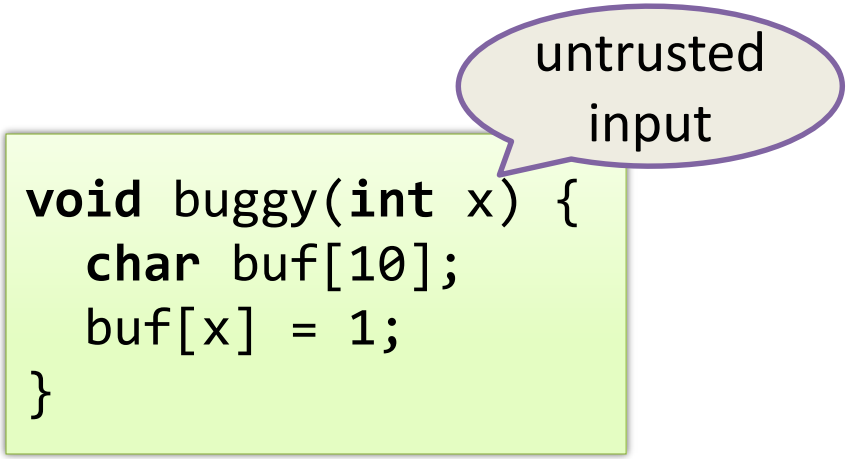
Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses



Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses



attacker memory unsafe

Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses

untrusted input

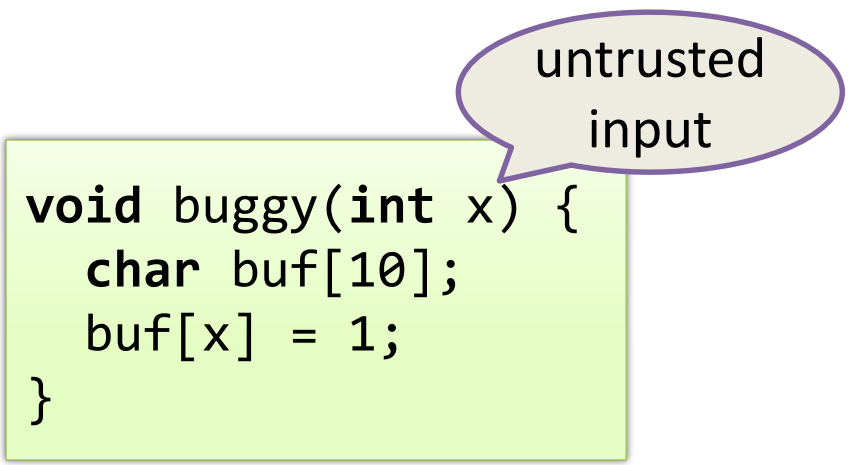
```
void buggy(int x) {
  char buf[10];
  buf[x] = 1;
}
```

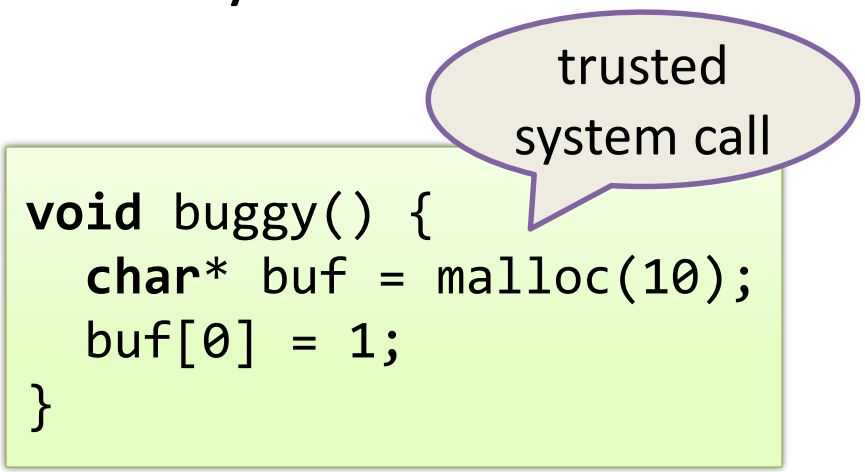
attacker memory unsafe

```
void buggy() {
  char* buf = malloc(10);
  buf[0] = 1;
}
```

Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses

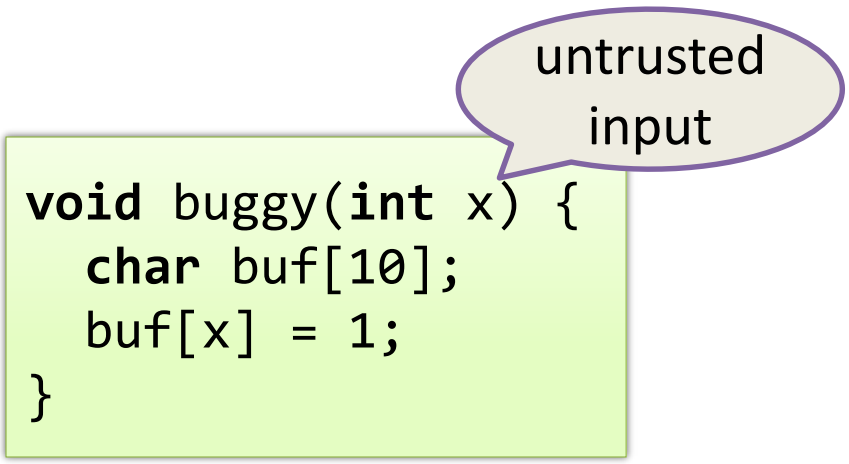




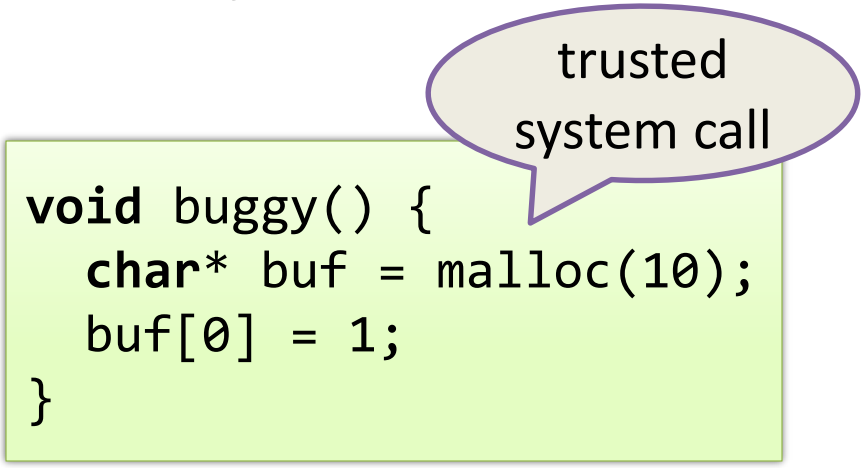
attacker memory unsafe

Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses



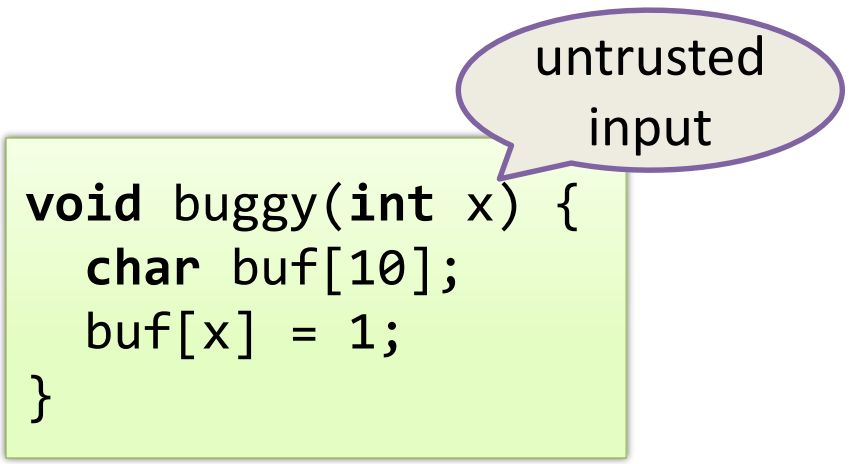
attacker memory unsafe



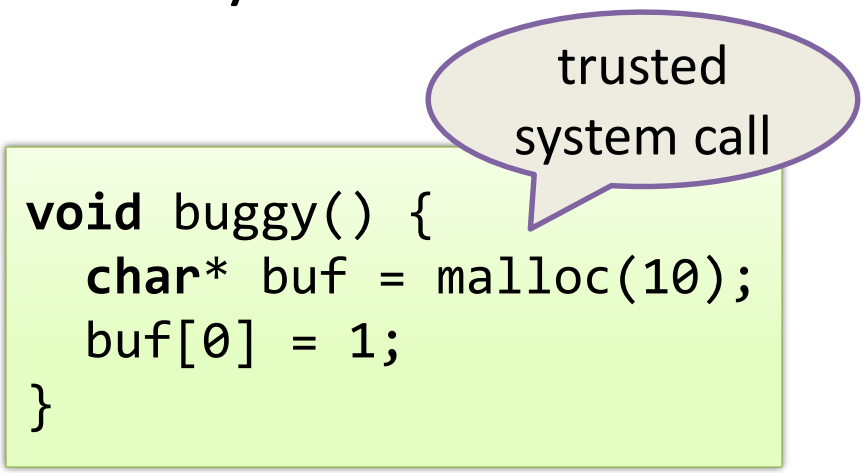
attacker memory safe

Attacker memory safety

- Means no attacker-controllable buffer overflows
- Is weaker than traditional memory safety
- Ignores input-independent memory accesses



attacker memory unsafe



attacker memory safe
but memory unsafe

SAGE

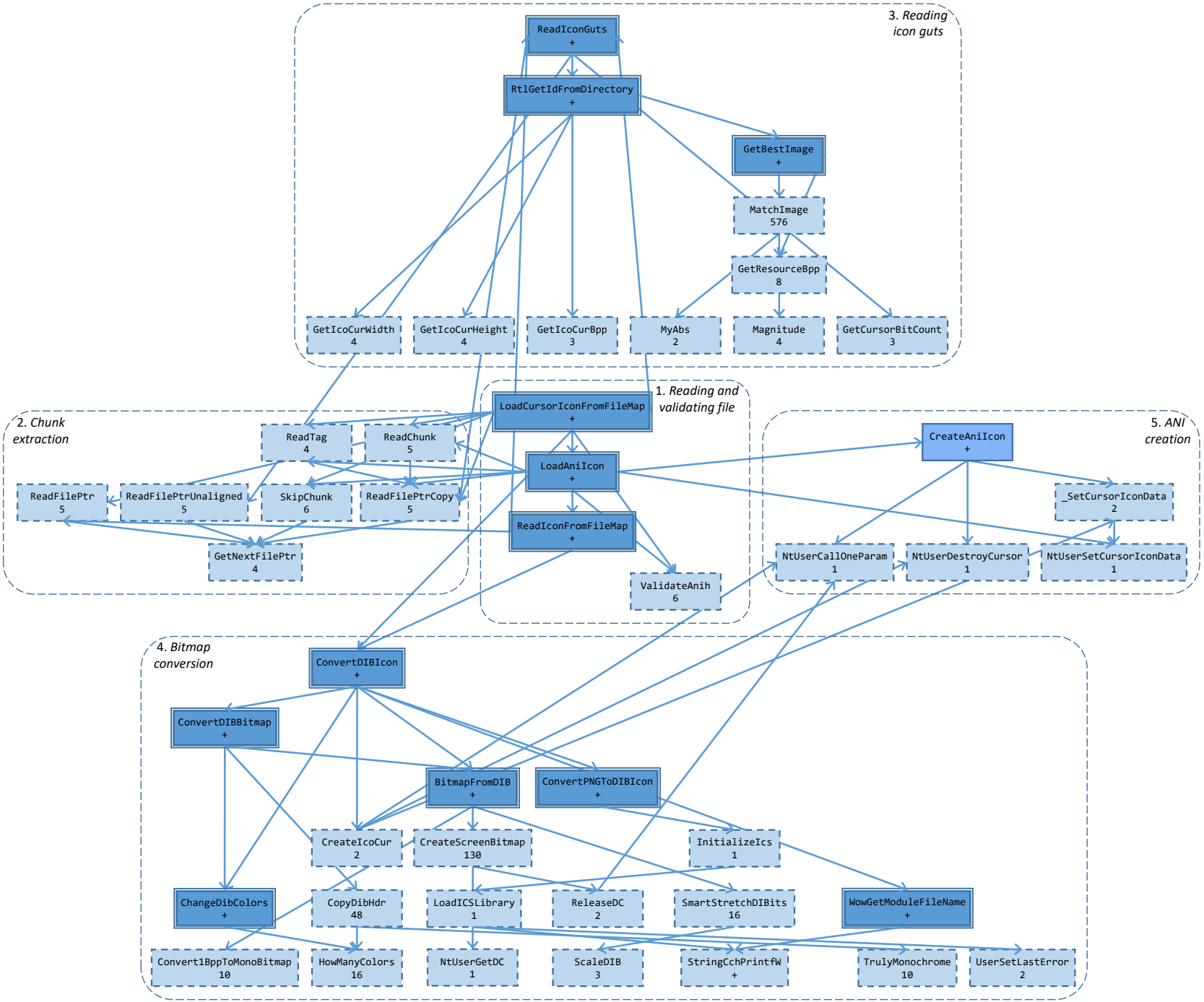
- Is a whitebox fuzzer for security testing
- Applies dynamic symbolic execution at the x86 level
- Is extensively used to fuzz several Microsoft products

Verification with testing?

- All program paths may be enumerated with dynamic symbolic execution when
  - there are finitely many paths
  - all feasible branches are explored
  - the constraint solver is sound and complete
- We turned maximum precision of SAGE on and any unsound pruning off

The ANI Windows image parser

- Implemented in C and x86 assembly
- At least 350 functions in 5 DLLs
- Testing untrusted inputs in at least 110 functions
- 47 top-level functions in `user32.dll`
- No recursion and no complex data structures


3. Reading icon guts
ReadIconGuts +
RtlGetIdFromDirectory +
GetBestImage +
MatchImage 576
GetResourceBpp 8
GetIcoCurWidth 4
GetIcoCurHeight 4
GetIcoCurBpp 3
MyAbs 2
Magnitude 4
GetCursorBitCount 3
1. Reading and validating file
LoadCursorIconFromFileMap +
LoadAniIcon +
ReadIconFromFileMap +
ValidateAnih 6
2. Chunk extraction
ReadTag 4
ReadChunk 5
ReadFilePtr 5
ReadFilePtrUnaligned 5
SkipChunk 6
ReadFilePtrCopy 5
GetNextFilePtr 4
5. ANI creation
CreateAniIcon +
_SetCursorIconData 2
NtUserCallOneParam 1
NtUserDestroyCursor 1
NtUserSetCursorIconData 1
4. Bitmap conversion
ConvertDIBIcon +
ConvertDIBBitmap +
BitmapFromDIB +
ConvertPNGToDIBIcon +
CreateIcoCur 2
CreateScreenBitmap 130
InitializeIcs 1
ChangeDibColors +
CopyDibHdr 48
LoadICSLibrary 1
ReleaseDC 2
SmartStretchDIBits 16
WowGetModuleFileName +
Convert1BppToMonoBitmap 10
HowManyColors 16
NtUserGetDC 1
ScaleDIB 3
StringCchPrintfW +
TrulyMonochrome 10
UserSetLastError 2

Strategies

- Stage 1: Bottom-up strategy
  - Exhaustive path enumeration and inlining
- Stage 2: Input-dependent loops
  - Restricting input-dependent loop bounds
- Stage 3: Top-down strategy
  - Summarization

Strategies

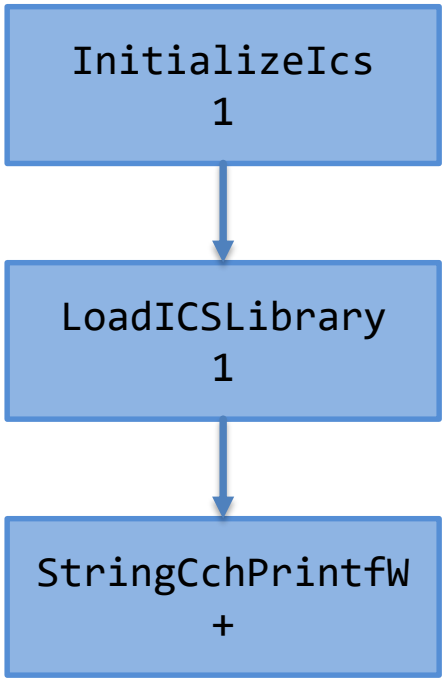
- Stage 1: Bottom-up strategy
  - Exhaustive path enumeration and inlining
- Stage 2: Input-dependent loops
  - Restricting input-dependent loop bounds
- Stage 3: Top-down strategy
  - Summarization
- Fixing or ignoring attacker-memory-safety bugs

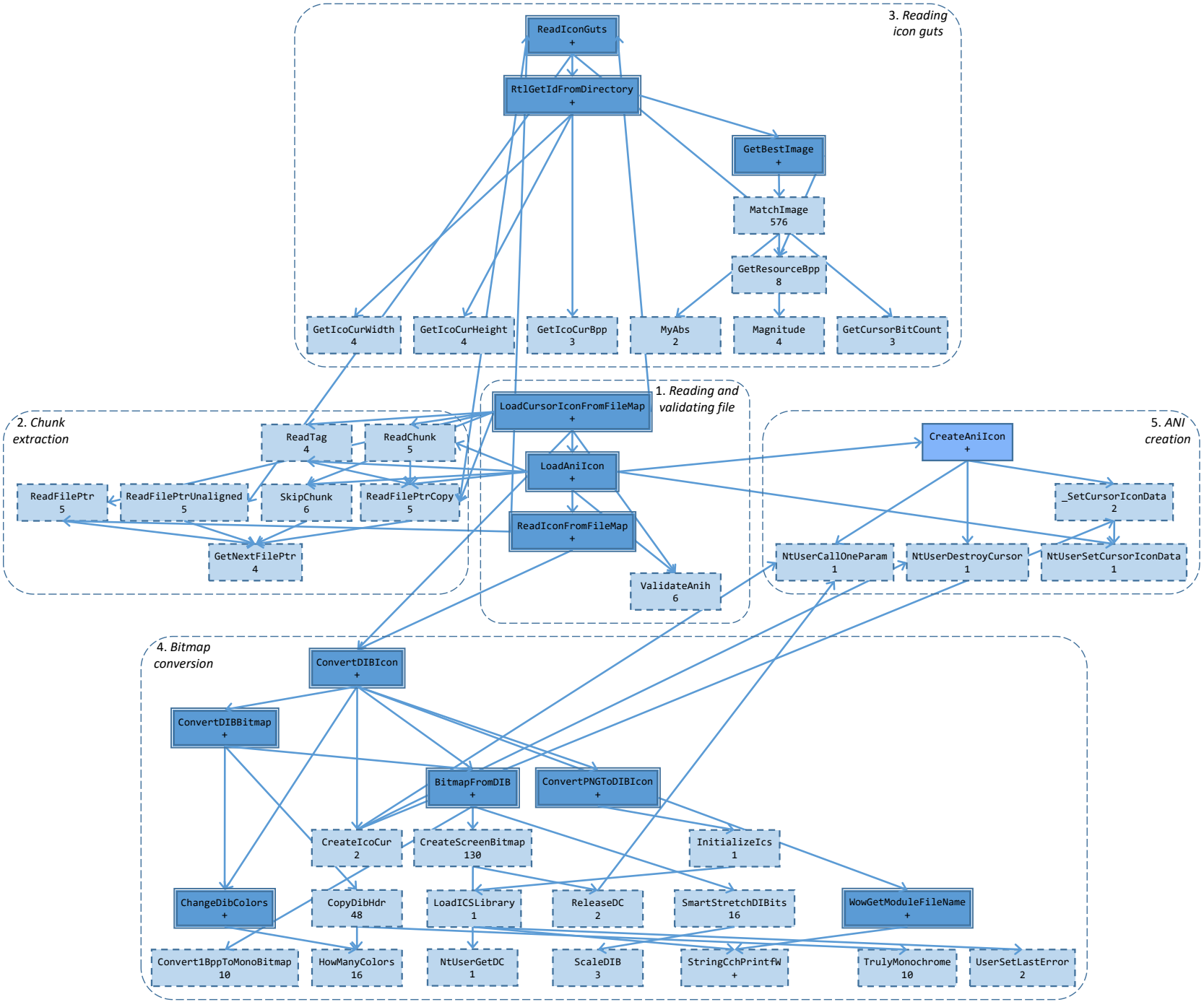
Stage 1: Bottom-up strategy

- Exhaustive path enumeration for all calling contexts in less than 12 hours

Stage 1: Bottom-up strategy

- Exhaustive path enumeration for all calling contexts in less than 12 hours




3. Reading icon guts
ReadIconGuts +
RtlGetIdFromDirectory +
GetBestImage +
MatchImage 576
GetResourceBpp 8
GetIcoCurWidth 4
GetIcoCurHeight 4
GetIcoCurBpp 3
MyAbs 2
Magnitude 4
GetCursorBitCount 3
1. Reading and validating file
LoadCursorIconFromFileMap +
LoadAniIcon +
ReadIconFromFileMap +
ValidateAnih 6
2. Chunk extraction
ReadTag 4
ReadChunk 5
ReadFilePtr 5
ReadFilePtrUnaligned 5
SkipChunk 6
ReadFilePtrCopy 5
GetNextFilePtr 4
5. ANI creation
CreateAniIcon +
_SetCursorIconData 2
NtUserCallOneParam 1
NtUserDestroyCursor 1
NtUserSetCursorIconData 1
4. Bitmap conversion
ConvertDIBIcon +
ConvertDIBBitmap +
BitmapFromDIB +
ConvertPNGToDIBIcon +
CreateIcoCur 2
CreateScreenBitmap 130
InitializeIcs 1
ChangeDibColors +
CopyDibHdr 48
LoadICSLibrary 1
ReleaseDC 2
SmartStretchDIBits 16
WowGetModuleFileName +
Convert1BppToMonoBitmap 10
HowManyColors 16
NtUserGetDC 1
ScaleDIB 3
StringCchPrintfW +
TrulyMonochrome 10
UserSetLastError 2

Stage 1: Bottom-up strategy

- Exhaustive path enumeration for all calling contexts in less than 12 hours

34 out of 47 functions were verified!

Stage 1: Bottom-up strategy

- Exhaustive path enumeration for all calling contexts in less than 12 hours

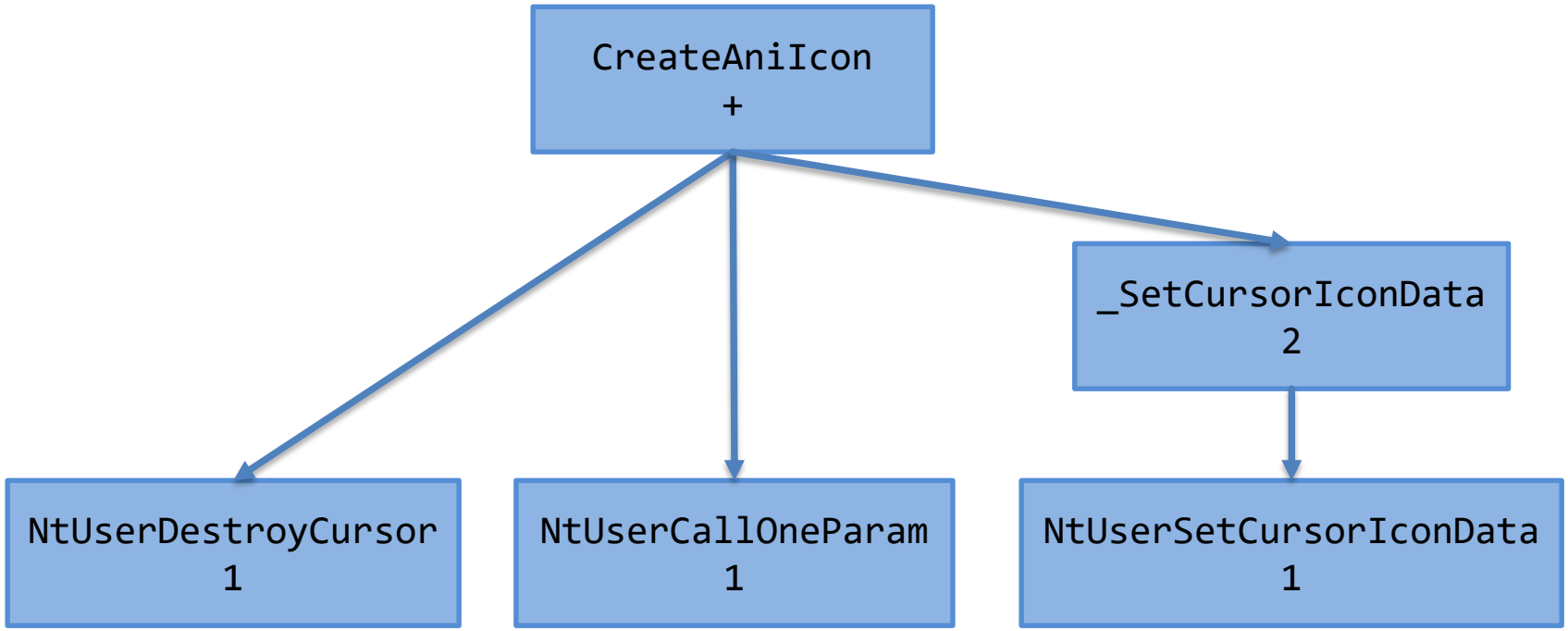
34 out of 47 functions were verified!

Stage 2: Input-dependent loops

- Manually restricting input-dependent loop bounds

Stage 2: Input-dependent loops

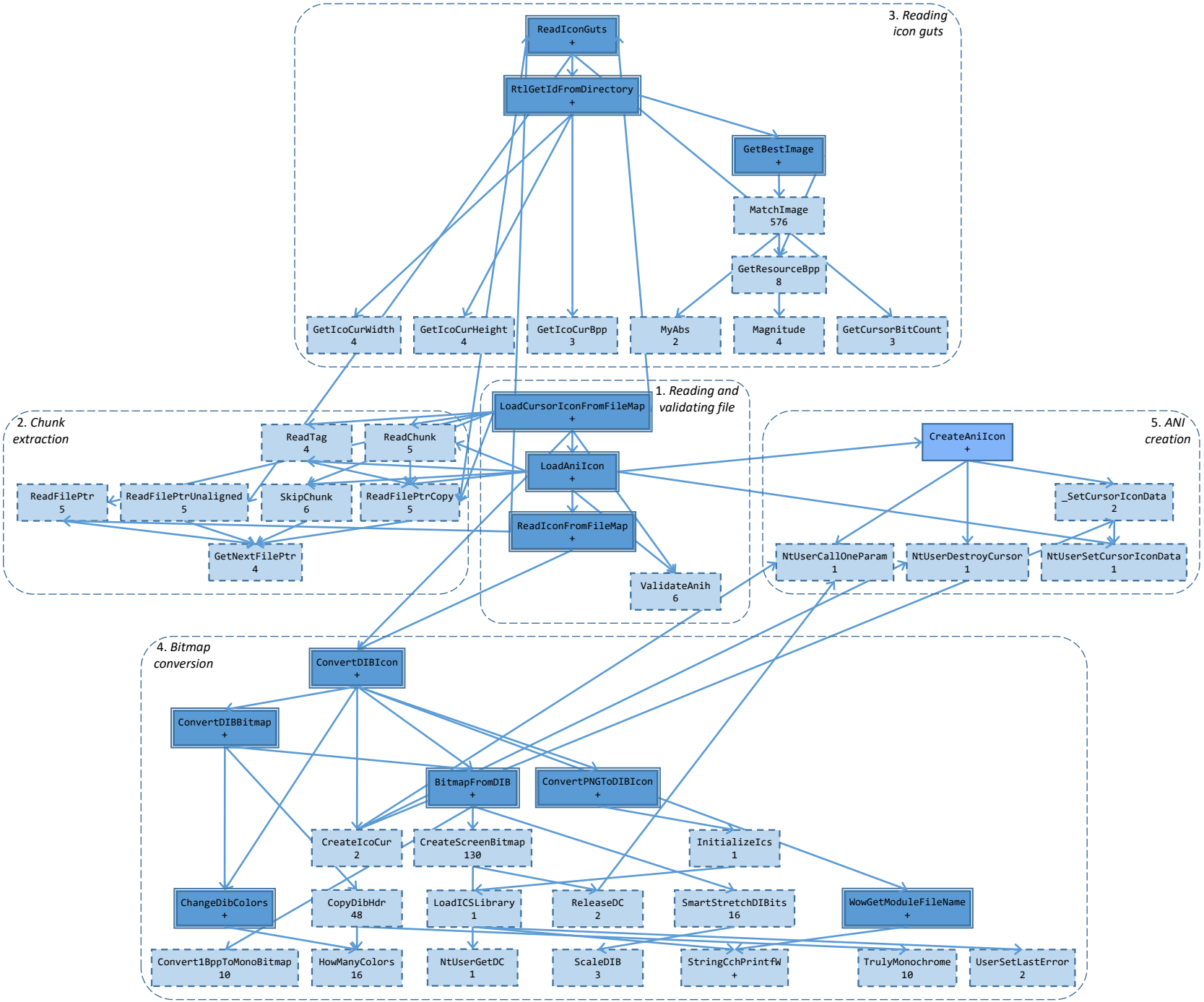
- Manually restricting input-dependent loop bounds



Stage 2: Input-dependent loops

- Manually restricting input-dependent loop bounds

```
for (i = 0; i < frames; i++)
  frameArrT[i] = frameArr[i];
for (i = 0; i < steps; i++) {
  if (rateArr == NULL)
    rateArrT[i] = rate;
  else
    rateArrT[i] = rateArr[i];
  if (stepArr == NULL)
    stepArrT[i] = i;
  else
    stepArrT[i] = stepArr[i];
}
```


3. Reading icon guts
ReadIconGuts +
RtlGetIdFromDirectory +
GetBestImage +
MatchImage 576
GetResourceBpp 8
GetIcoCurWidth 4
GetIcoCurHeight 4
GetIcoCurBpp 3
MyAbs 2
Magnitude 4
GetCursorBitCount 3
1. Reading and validating file
LoadCursorIconFromFileMap +
LoadAniIcon +
ReadIconFromFileMap +
ValidateAnih 6
2. Chunk extraction
ReadTag 4
ReadChunk 5
ReadFilePtr 5
ReadFilePtrUnaligned 5
SkipChunk 6
ReadFilePtrCopy 5
GetNextFilePtr 4
5. ANI creation
CreateAniIcon +
_SetCursorIconData 2
NtUserCallOneParam 1
NtUserDestroyCursor 1
NtUserSetCursorIconData 1
4. Bitmap conversion
ConvertDIBIcon +
ConvertDIBBitmap +
BitmapFromDIB +
ConvertPNGToDIBIcon +
CreateIcoCur 2
CreateScreenBitmap 130
InitializeIcs 1
ChangeDibColors +
CopyDibHdr 48
LoadICSLibrary 1
ReleaseDC 2
SmartStretchDIBits 16
WowGetModuleFileName +
Convert1BppToMonoBitmap 10
HowManyColors 16
NtUserGetDC 1
ScaleDIB 3
StringCchPrintfW +
TrulyMonochrome 10
UserSetLastError 2

Stage 2: Input-dependent loops

- Manually restricting input-dependent loop bounds

34 out of 47 functions were verified!

1 out of 47 functions was tested!

Stage 2: Input-dependent loops

- Manually restricting input-dependent loop bounds

34 out of 47 functions were verified!

1 out of 47 functions was tested!

All input-dependent loop bounds are controlled by 10 input bytes plus the file size!

Stage 2: Input-dependent loops

- Manually restricting input-dependent loop bounds

34 out of 47 functions were verified!

1 out of 47 functions was tested!

All input-dependent loop bounds are controlled by 10 input bytes plus the file size!

Stage 3: Top-down strategy

- Manual program decomposition and summarization

Summarization

- Alleviates path explosion at the expense of computing logic representations of functions

Summarization

- Alleviates path explosion at the expense of computing logic representations of functions
- Is attractive only for functions
  - that contain many execution paths, and
  - whose inputs/outputs are easy to represent logically

Compositional memory safety

```
void bar(char* buf, int x) {
  if (0 <= x && x < 10) buf[x] = 1;
}
```

- Summary for bar:

$$\big((0 \le x) \wedge (x < 10) \wedge (0 \le x < \mathsf{buf}.size) \wedge (\mathsf{buf[x]=1})\big) \vee \big((x < 0) \wedge (10 \le x)\big)$$

Compositional memory safety

```
void bar(char* buf, int x) {
  if (0 <= x && x < 10) buf[x] = 1;
}
```

- Summary for bar:

$$\big((0 \le x) \wedge (x < 10) \wedge (0 \le x < \text{buf}.size) \wedge (\text{buf[x]=1})\big) \vee \big((x < 0) \wedge (10 \le x)\big)$$

```
void foo(int x) {
  char* buf = malloc(5);
  bar(buf, x);
}
```

Compositional memory safety

```
void bar(char* buf, int x) {
  if (0 <= x && x < 10) buf[x] = 1;
}
```

- Summary for bar:

$$\big((0 \le x) \wedge (x < 10) \wedge (0 \le x < \text{buf}.\cancel{size}) \wedge (\text{buf[x]=1})\big) \vee \big((x < 0) \wedge (10 \le x)\big)$$

5

```
void foo(int x) {
  char* buf = malloc(5);
  bar(buf, x);
}
```

Stage 3: Top-down strategy

- Manual program decomposition and summarization

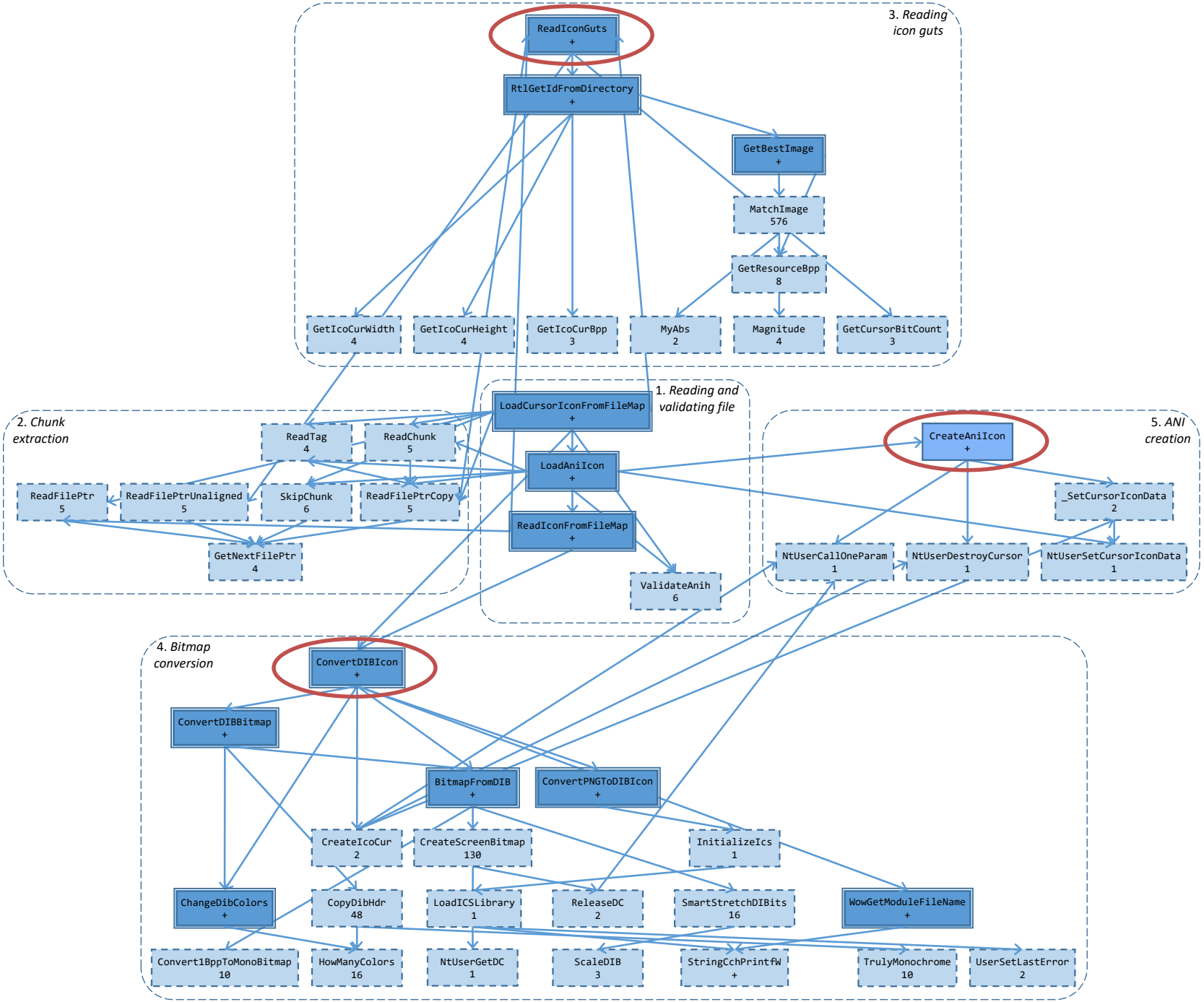
Stage 3: Top-down strategy

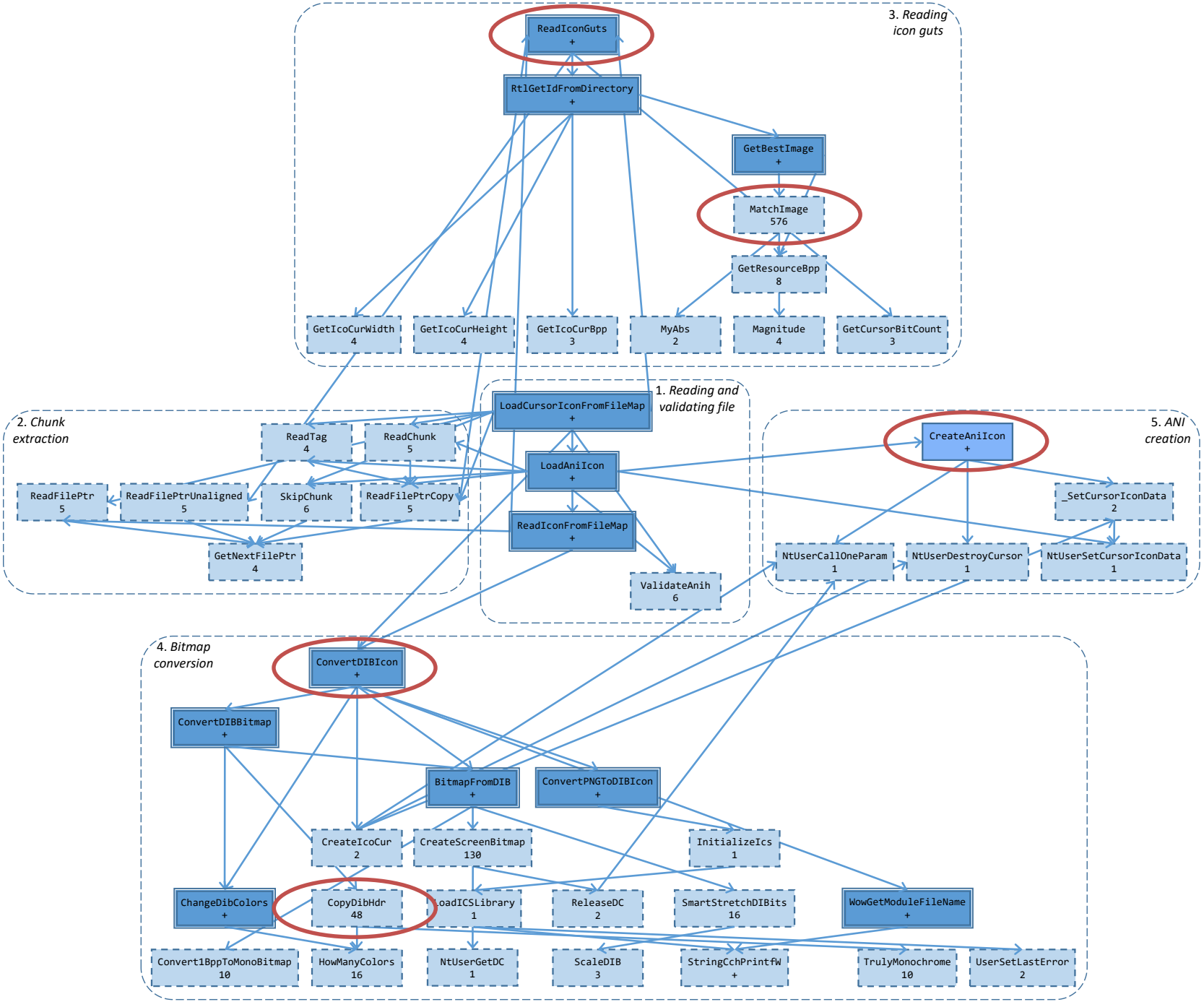
- Manual program decomposition and summarization

- Key observation:
  Architectural components of the parser typically return a single “success” or “failure” value

  - If “failure”, parsing aborts

  - If “success”, parsing proceeds with unconstrained inputs


3. Reading icon guts
ReadIconGuts +
RtlGetIdFromDirectory +
GetBestImage +
MatchImage 576
GetResourceBpp 8
GetIcoCurWidth 4
GetIcoCurHeight 4
GetIcoCurBpp 3
MyAbs 2
Magnitude 4
GetCursorBitCount 3
1. Reading and validating file
LoadCursorIconFromFileMap +
LoadAniIcon +
ReadIconFromFileMap +
ValidateAnih 6
2. Chunk extraction
ReadTag 4
ReadChunk 5
ReadFilePtr 5
ReadFilePtrUnaligned 5
SkipChunk 6
ReadFilePtrCopy 5
GetNextFilePtr 4
5. ANI creation
CreateAniIcon +
_SetCursorIconData 2
NtUserCallOneParam 1
NtUserDestroyCursor 1
NtUserSetCursorIconData 1
4. Bitmap conversion
ConvertDIBIcon +
ConvertDIBBitmap +
BitmapFromDIB +
ConvertPNGToDIBIcon +
CreateIcoCur 2
CreateScreenBitmap 130
InitializeIcs 1
ChangeDibColors +
CopyDibHdr 48
LoadICSLibrary 1
ReleaseDC 2
SmartStretchDIBits 16
WowGetModuleFileName +
Convert1BppToMonoBitmap 10
HowManyColors 16
NtUserGetDC 1
ScaleDIB 3
StringCchPrintfW +
TrulyMonochrome 10
UserSetLastError 2


3. Reading icon guts
ReadIconGuts +
RtlGetIdFromDirectory +
GetBestImage +
MatchImage 576
GetResourceBpp 8
GetIcoCurWidth 4
GetIcoCurHeight 4
GetIcoCurBpp 3
MyAbs 2
Magnitude 4
GetCursorBitCount 3
1. Reading and validating file
LoadCursorIconFromFileMap +
LoadAniIcon +
ReadIconFromFileMap +
ValidateAnih 6
2. Chunk extraction
ReadTag 4
ReadChunk 5
ReadFilePtr 5
ReadFilePtrUnaligned 5
SkipChunk 6
ReadFilePtrCopy 5
GetNextFilePtr 4
5. ANI creation
CreateAniIcon +
_SetCursorIconData 2
NtUserCallOneParam 1
NtUserDestroyCursor 1
NtUserSetCursorIconData 1
4. Bitmap conversion
ConvertDIBIcon +
ConvertDIBBitmap +
BitmapFromDIB +
ConvertPNGToDIBIcon +
CreateIcoCur 2
CreateScreenBitmap 130
InitializeIcs 1
ChangeDibColors +
CopyDibHdr 48
LoadICSLibrary 1
ReleaseDC 2
SmartStretchDIBits 16
WowGetModuleFileName +
Convert1BppToMonoBitmap 10
HowManyColors 16
NtUserGetDC 1
ScaleDIB 3
StringCchPrintfW +
TrulyMonochrome 10
UserSetLastError 2

Stage 3: Top-down strategy

- Manual program decomposition and summarization

34 out of 47 functions were verified!

13 out of 47 functions were tested!

Stage 3: Top-down strategy

- Manual program decomposition and summarization

34 out of 47 functions were verified!

13 out of 47 functions were tested!

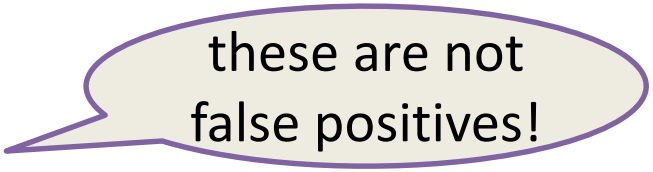
Only 5 summaries were used!

Attacker-memory-safety bugs

- Real bugs (fixed in the latest version of Windows)
- Harmless (off-by-one) bugs
- Code parts not memory safe by design

Attacker-memory-safety bugs

- Real bugs (fixed in the latest version of Windows)
- Harmless (off-by-one) bugs
- Code parts not memory safe by design



Shrinking the gap

- Interesting findings:
  - Many loop-free and easy-to-verify functions
  - Input-dependent loops controlled by about 10 bytes plus the file size
  - Path explosion controlled by 5 simple summaries

- Can we automate the manual steps?

Static Program Analysis Meets Test Case Generation Lecture 1

Maria Christakis

MPI-SWS, Germany