Teaching with angr: A Symbolic Execution Curriculum and CTF

Jacob M. Springer\textsuperscript{1,2}, Wu-chang Feng\textsuperscript{1}

\textsuperscript{1}Portland State University
\textsuperscript{2}Swarthmore College
Outline

- What is symbolic execution?
- How do we teach it?
Symbolic execution: why should you care?
Symbolic execution: why should you care?

- Program analysis and testing
Symbolic execution: why should you care?

- Program analysis and testing
- Microsoft applications (PowerPoint, Word, etc.)
Symbolic execution: why should you care?

- Program analysis and testing
- Microsoft applications (PowerPoint, Word, etc.)
- DARPA's Cyber-Grand Challenge
Symbolic execution: why should you care?

- Program analysis and testing
- Microsoft applications (PowerPoint, Word, etc.)
- DARPA's Cyber-Grand Challenge
- Important for students to understand and apply
What is symbolic execution?
Find input to print “Good Job.”

```c
int check_code(int input) {
    if (input >= SECRET+88) return 0;
    if (input > SECRET+100) return 0;
    if (input == SECRET+68) return 0;
    if (input < SECRET) return 0;
    if (input <= SECRET+78) return 0;
    if (input & 0x1) return 0;
    if (input & 0x2) return 0;
    if (input & 0x4) return 0;
    return 1;
}

int main() {
    int input;
    scanf("%d", &input);
    if (check_code(input))
        printf("Good Job.\n");
    else
        printf("Try again.\n");
}
```
Execution paths can be represented as a tree.
Animation: Building a Set of Paths

```javascript
if (input >= SECRET + 88)
```

Legend:
- **Blue** = already executed
- **Yellow** = active
- **Red** = terminated
Animation: Building a Set of Paths

Legend:
Blue = already executed
Yellow = active
Red = terminated
Animation: Building a Set of Paths

Legend:
Blue = already executed
Yellow = active
Red = terminated

```c
if (input >= SECRET+88)
    return 0;

if (input > SECRET+100)
    return 0;

if (input == SECRET+68)
    return 0;
```
Animation: Building a Set of Paths

Legend:
- Blue = already executed
- Yellow = active
- Red = terminated

We found what we wanted!
Applying symbolic execution

Once we have a path, we can build an equation that can be solved by the computer:

```plaintext
if (input >= SECRET+88)
  if (input > SECRET+100)
    if (input == SECRET+68)
      if (input < SECRET)
        if (input <= SECRET+78)
          if (input & 0x1)
            if (input & 0x2)
              if (input & 0x4)
                return 1;
```

\[
\text{input} \geq \text{SECRET}+88 \quad \land \quad \text{input} > \text{SECRET}+100 \quad \land \quad \text{input} = \text{SECRET}+68 \quad \land \quad \text{input} < \text{SECRET} \quad \land \quad \text{input} \leq \text{SECRET}+78 \quad \land \quad \text{input} & 0x1 \quad \land \quad \text{input} & 0x2 \quad \land \quad \text{input} & 0x4
\]
Angr-y CTF

Goal: Build a curriculum and a set of capture-the-flag (CTF) levels to introduce students to symbolic execution
Our Approach
Our Approach

Modeled after MetaCTF (USENIX 3GSE 2015)

- Find a password that causes a program to print "Good Job."
Our Approach

Modeled after MetaCTF (USENIX 3GSE 2015)

- Find a password that causes a program to print "Good Job."

18 scaffolded levels

- Requires symbolic execution to solve
Our Approach

Modeled after MetaCTF (USENIX 3GSE 2015)

- Find a password that causes a program to print "Good Job."

18 scaffolded levels

- Requires symbolic execution to solve

Uses angr (angr.io)
A typical level
A typical level

- Student receives a binary and a template angr script
A typical level

- Student receives a binary and a template angr script
- Student edits the template to analyze the binary
A typical level

- Student receives a binary and a template angr script
- Student edits the template to analyze the binary
- Student runs the script which prints a password
A typical level

- Student receives a binary and a template angr script
- Student edits the template to analyze the binary
- Student runs the script which prints a password
- Student runs the binary and types in the password to confirm their work
The levels are scaffolded
What does scaffolding mean?

- Support structure, just like a scaffold
- Guided, incremental introduction of concepts
CTF Modules

- Basic symbolic execution
- Symbol injection
- Handling complexity
- Automated exploitation
Scaffolding for pedagogy: not frustrating

- Level 1
- Well documented
- Only need to change two lines
Scaffolding for pedagogy: guided

- Tells student how to get started

```bash
23  # We want to identify a place in the binary, when strncpy is called, when we can:
24  #  1) Control the source contents (not the source pointer!)
25  #       * This will allow us to write arbitrary data to the destination.
26  #  2) Control the destination pointer
27  #       * This will allow us to write to an arbitrary location.
```
Scaffolding: simple

# Explore the binary to attempt to find the address that prints "Good Job."
# You will have to find the address you want to find and insert it here.
# This function will keep executing until it either finds a solution or it
# has explored every possible path through the executable.
# (!)
print_good_address = ???  # :integer (probably in hexadecimal)
simulation.explore(find=print_good_address)
MetaCTF Example

```c
int check_code(int input) {
    if (input >= SECRET+88) return 0;
    if (input > SECRET+100) return 0;
    if (input == SECRET+68) return 0;
    if (input < SECRET) return 0;
    if (input <= SECRET+78) return 0;
    if (input & 0x1) return 0;
    if (input & 0x2) return 0;
    if (input & 0x4) return 0;
    return 1;
}

int main() {
    int input;
    scanf("%d", &input);
    if (check_code(input))
        printf("Good Job.\n");
    else
        printf("Try again.\n");
}
```

```
0x804867a ;[gi]
sub esp, 0xc
; 0x8048760
; "Good Job."
push str.Good_Job.
call sym.imp.puts;[gk]
add esp, 0x10
```
Scaffolding: builds on previous concepts

```c
0x804867a ;[gi]
sub esp, 0xc
; 0x8048760
; "Good Job."
push str.Good_Job.
call sym.imp.puts; [gk]
add esp, 0x10

# Explore the binary to attempt to find the address that prints "Good Job."
# You will have to find the address you want to find and insert it here.
# This function will keep executing until it either finds a solution or it
# has explored every possible path through the executable.
# (!)
print_good_address = 0x804867a # :integer (probably in hexadecimal)
simulation.explore(find=print_good_address)
```
Scaffolding: incremental and reinforcing

- Level 02 (find_condition)

- Level 03 (symbolic_registers)
Scaffolding: incremental and reinforcing

- Level 02 (find_condition)
  - 1. Load binary
  - 2. Define the termination condition (Has the program printed “Good Job.”?)
  - 3. Search binary for condition
- Level 03 (symbolic_registers)
Scaffolding: incremental and reinforcing

- Level 02 (find_condition)
  - 1. Load binary
  - 2. Define the termination condition (Has the program printed “Good Job.”?)
  - 3. Search binary for condition

- Level 03 (symbolic_registers)
  - 1. Load binary
  - 2. Inject symbols
  - 3. Define the termination condition (Has the program printed “Good Job.”?)
  - 4. Search binary for condition
Scaffolding: conceptual

- First glance: seems complicated
Scaffolding: conceptual, part 2

# Get the eip register (review 03_anger_symbolic_registers).
# (!)
eip = unconstrained_state.regs.

# Check if we can set the state to our print_good function.
# (!)
if unconstrained_state.satisfiable(extra_constraints=(eip == ???)):  # We can!
    solution_state = unconstrained_state

# Now, constrain eip to equal the address of the print_good function.
# (!)
...
break
What does metamorphic mean?

- Different SECRET for every student
- Can generate arbitrary C code

```c
int check_code(int input) {
    if (input >= SECRET+88) return 0;
    if (input > SECRET+100) return 0;
    if (input == SECRET+68) return 0;
    if (input < SECRET) return 0;
    if (input <= SECRET+78) return 0;
    if (input & 0x1) return 0;
    if (input & 0x2) return 0;
    if (input & 0x4) return 0;
    return 1;
}

int main() {
    int input;
    scanf("%d", &input);
    if (check_code(input))
        printf("Good Job.\n");
    else
        printf("Try again.\n");
}```
Metamorphic levels

- Reduce cheating
- Allow reuse
- Maintain consistency of difficulty across students
Evaluation

- Offered Winter 2018 in Portland State University's CS 492/592: Malware course
  - Last 2 weeks focused on symbolic execution
- Survey given at the end of two weeks
  - 33 of 42 responded
Results

Curriculum and scaffolding allow students to complete most levels

<table>
<thead>
<tr>
<th>Completion percentage</th>
<th>Number of students</th>
</tr>
</thead>
<tbody>
<tr>
<td>95-100%</td>
<td>25</td>
</tr>
<tr>
<td>85-95%</td>
<td>4</td>
</tr>
<tr>
<td>75-85%</td>
<td>6</td>
</tr>
<tr>
<td>Below 75%</td>
<td>7</td>
</tr>
</tbody>
</table>
Survey

- Ratings evaluate helpfulness of curriculum and CTF
  - Very Unhelpful = 1
  - Very Helpful = 5

- Q1: Rate the lecture material for understanding the concepts

<table>
<thead>
<tr>
<th>Rating</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>17</td>
<td>12</td>
<td>4.15</td>
</tr>
</tbody>
</table>

- Q2: Rate the CTF exercises for understanding the concepts

<table>
<thead>
<tr>
<th>Rating</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>18</td>
<td>9</td>
<td>3.94</td>
</tr>
</tbody>
</table>
Survey

- Q3: Rate the CTF exercises for developing skills in using symbolic execution techniques

<table>
<thead>
<tr>
<th>Rating</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>16</td>
<td>10</td>
<td>3.94</td>
</tr>
</tbody>
</table>
Try the CTF!

https://malware.oregonctf.org

Also on GitHub http://github.com/jakespringer/angr_ctf