Synthesis for Software Security

Jeff Foster
University of Maryland, College Park
Introduction

• Security is hard to get right
  ■ Security is a property of a whole program
    - Components that may be “secure” in isolation in some sense, may not be when combined
  ■ Secure coding practices can be subtle
    - Example: given an API that provides features for security, it is still easy to make mistakes

• Can we use *software synthesis* to help make programs secure?
Symbolic Security Analysis for Rails

- Many web applications built with frameworks
  - Ruby on Rails is one popular framework
  - Uses Ruby, a dynamic, OO scripting language
- Frameworks come with lots of security features
  - But programmers must use those features correctly and carefully
  - And must also worry about application-specific security properties
    - E.g., sensitive operations protected with access control, etc
Symbolic Execution Example

1. int a = α, b = β, c = γ;
2. // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c) { y = 1; }
9.   z = 2;
10.}
11. assert(x+y+z!=3)
Why Is This Possible?

• There are very powerful SMT/SAT solvers today
  ▪ SMT = Satisfiability Modulo Theories = SAT++
  ▪ Can solve very large instances, very quickly
    - Lets us check assertions, prune infeasible paths
  ▪ We’ve used Z3, STP, and Yices

• Recent success: bug finding
  ▪ Heuristic search through space of possible executions
  ▪ Find really interesting bugs

• We’ve also used these for deeper analysis and synthesis
Rubyx: Symbolic Execution for Rails

[Abstract Implementation]
Rails API

[Abstract Implementation]
Browser + Web Server

[Specifications]

Bad states reachable?

[Abstract Implementation]
Analysis Script

[Specifications, incl. roles]
Programmable Specifications

• Rubyx provides four special operations
  ■ `fresh(name)` — returns a fresh symbolic variable
  ■ `assume(p)` — adds `p` to the path condition
  ■ `assert(p)` — checks that path condition implies `p`
  ■ `def invariant() p end` — maintains `p` as an invariant for all objects of the class

• In above, `p` can be any Ruby expression
  ■ As in Ruby, `false` and `nil` are false, everything else true

• Writing specs just like writing Ruby tests
  ■ And testing is heavily used in Ruby community
Example Specification

Only admin can modify database

```ruby
# send login request
response = Browser.exec(UserController, :login, fresh(:PARAMS))

# assume that login is successful
assume Browser.session[:id]

# send request to update user information
response = Browser.exec(UserController, :update, fresh(:PARAMS))

# assert that logged in user is admin
assert(User.admin?(Prin.sender)) if Db.modified?(User)
```
# No XSS: output sent to trusted users has been sanitized
assert (output.trust?) unless (Prin.receiver == Lattice.bot)

# Secrecy: output secrecy level is at most level of receiver
assert (Lattice.leq (output.secrecy?, Prin.receiver))

# No CSRF: messages must be sent from higher to low trust levels, and requests that change state must be POST requests
assert (Lattice.leq (Prin.receiver, Prin.sender)) if params[:post]
assert params[:post] if (Session.modifier? || Db.modified?)

# Authentication: the sender and receiver must be at least as trusted as the logged-in user
assert (Lattice.leq (session[Prin.Id], Prin.receiver))
assert (Lattice.leq (session[Prin.Id], Prin.sender)) if params[:post]
Results

- Analyzed 7 apps, found security vulnerabilities in all
- Common lack of understanding of CSRF, session replay
- Most vulnerabilities could be easily fixed

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pubmgr</td>
<td>√</td>
<td>×(1)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×(2)</td>
</tr>
<tr>
<td>coffee</td>
<td>? (2)</td>
<td>×(1)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>N/A</td>
</tr>
<tr>
<td>depot</td>
<td>×(2)</td>
<td>×(10)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×r (7)</td>
</tr>
<tr>
<td>chuckslist</td>
<td>√</td>
<td>×(1)</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>N/A</td>
</tr>
<tr>
<td>boxroom</td>
<td>√</td>
<td>×(5)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?r (2)</td>
</tr>
<tr>
<td>mystic</td>
<td>×(17)</td>
<td>×(8)</td>
<td>√</td>
<td>√</td>
<td>×r (6)</td>
<td>N/A</td>
</tr>
<tr>
<td>rtplan</td>
<td>√</td>
<td>×(1)</td>
<td>×(16)</td>
<td>√</td>
<td>?r</td>
<td>√</td>
</tr>
</tbody>
</table>

√ = no vuln. found  × = vuln. found  ? = potential vuln. found  (n) = n fixes  (−) = did not fix  r = replay attack
Symbolic Execution for Web Apps

• Web applications are “broad” and “shallow”
  ▪ Program designed to quiesce at request-response boundaries
  ▪ Many possible requests and responses, but each request-response short

• Scripting languages are highly dynamic
  ▪ Challenging to analyze with traditional static analysis
  ▪ But symbolic execution just runs the code to see what it does

• Specs are written as programs
  ▪ Should be easy for programmers to write, understand
Synthesis for Software Security

• Wouldn’t it be great to synthesize the code needed to satisfy security policies?

• We have the pieces to start working on this problem:
  - Synthesis technology
    - We can synthesize code using symbolic execution as a driver
      - PINS: Path-Based Inductive Synthesis
      - E.g., synthesized LZ77 decompressor given compressor
  - Security policies to start from
    - Simple policies: no buffer overflows, no null pointer errors, etc
    - Some richer, application-specific policies
    - Coding practices and frameworks for security
  - Vulnerable software!
Possible Measures of Success

• Performance
  ▪ Has to complete in reasonable time

• Correctness/effectiveness
  ▪ Ask developers to judge whether synthesized code is correct
  ▪ Prevent known attacks against system
  ▪ Red teaming against system with synthesized security

• Usefulness
  ▪ Evaluation by developers
  ▪ Deployment and long-term evaluation
New Advances Required

• Deeper understanding of software security
  ▪ Need more information to do synthesis than to do bug finding

• Scalable synthesis
  ▪ Synthesizing components in a big system harder than synthesizing small examples
    - Large systems don’t necessarily have the right structure and abstraction barriers to give synthesizer a clean interface
  ▪ Need synthesis that can be used by developers, during software development and maintenance
# For and Against

<table>
<thead>
<tr>
<th>Why This May Work</th>
<th>Why This May Not Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>There has been a tremendous amount of work on analyzing programs to find potential vulnerabilities, and this work has been successful.</td>
<td>“Bug finding” tools can be wrong sometimes (false positives or negatives); but synthesized code for security must be at least as good as an expert programmer.</td>
</tr>
<tr>
<td>We understand a wide range of security properties formally and precisely</td>
<td>We need a very thorough attacker model to drive the synthesis search.</td>
</tr>
<tr>
<td>The programmer saves effort by writing down a security policy (no redundancy).</td>
<td>Synthesis could still be too hard.</td>
</tr>
<tr>
<td>Synthesized security code could be resynthesized to respond to new threats as they evolve over time (monotonic improvement?)</td>
<td>The programmer needs to be convinced the synthesized code is correct, but it could change in surprising ways if resynthesized.</td>
</tr>
<tr>
<td>Like program analysis, program synthesis can in theory reason about all possible program runs, including unusual and untested paths</td>
<td>In practice, we can’t reason about all paths of large scale software; though we can still try to reason about many of them.</td>
</tr>
</tbody>
</table>
Discussion Questions

• Would developers and auditors trust that synthesized code actually improves security?

• Is it reasonable to write security specifications that are sufficient for synthesis, or is it easier to write “secure” code?

• Many security vulnerabilities have social aspects (e.g., phishing attacks); are those so prevalent now that we should focus on those, rather than more technical vulnerabilities (e.g., buffer overflows)?

• Could synthesis for security scale to large systems?

• How do we measure the utility of synthesis for programmers?