ASPIRE: Iterative Specification Synthesis for Security

Kevin Chen, Warren He, Devdatta Akhawe, Vijay D'Silva, Prateek Mittal, Dawn Song

University of California, Berkeley
Princeton University
Security Analysis of Software Systems

- Abstract: From programs to models
- Check: Security properties on the models

Specifications = $\mathbf{\text{\textbf{V}}}$ + $\mathbf{\text{\textbf{S}}}$
The Spectrum of Security Analyses

Completely manual

Fully automatic

CompCert C

BitBlaze

Address-sanitizer
Manual Specification Creation

- Steep learning curve
- Model remains error prone
- Process has to be repeated for different applications
Automatic Program Analysis (Bottom-up)

- Unable to efficiently recognize high-level semantics ("bad at throwing away details")
- Typically requires full code visibility or complex environment models
- Properties often hard-coded into the analyzers

BitBlaze

SLAM address-sanitizer
Is there a middle ground?
Insight: Build from Common Blocks
Preliminary Results: Security Analysis of Web Applications
Web Applications

- Hard to implement the protocols correctly
  - Customized APIs and undocumented behaviors
  - Subtleties of the web’s security model
- Hard to check the protocol implementation
  - Hard to generate models
  - Hard to specify security properties
  - Don’t have all parties’ code
Problem Definition

Do the following:

a. construct a model that is consistent with the application behavior (i.e. the execution traces)
b. check the model against the security policy.

Given reasonable resources:

i. a web application consisting of multiple parties
ii. execution traces of the web application
iii. a security policy
The Security Policy

- **Session integrity:**
  - Any action that an honest server takes should not be directly/indirectly caused by a dishonest/untrusted party
  - e.g. A request caused by robber.com shouldn’t reduce money in my bank account
  - e.g. A request caused by sessionrider.com shouldn’t change my login status on facebook.com

- **Information secrecy:**
  - Secrets shared by the client and the server should not be learned or inferred by any unauthorized third-party
Modeling: Observations

Common web application logics

- Web applications use similar mechanisms to maintain web sessions
- Single sign-on services use similar concepts regardless of the protocols (e.g. Facebook Connect, Google Login, CAS Login, ...)
- E-commerce protocols use similar concepts and steps to process payments (e.g. Paypal, Amazon payment, ...)
Modeling: Our approach

● Middle ground
  ○ Manually construct the basic blocks once
    ■ Application independent
  ○ Use these basic blocks to describe many protocols
    ■ Application dependent

● Representation:
  a domain specific language (DSL)

● Use DSL in program synthesis
  ○ programmatically search for program that passes test cases
  ○ high-level helps search efficiently
ASPIRE Workflow

(i) Application

Synthesizer

Model

Validator

Domain Knowledge and DSL

Policy

(iii)
ASPIRE Workflow

(i) Application

(ii) System Demonstration

Synthesizer

Model

Validator

Domain Knowledge and DSL

(iii) Policy
ASPIRE Workflow

(i) Application

(ii) System Demonstration

Synthesizer

Model

(iv) Domain Knowledge and DSL

Analyst Feedback

Policy

(iii) Validator
Running Example: Synthesis

~GET /login HTTP/1.1
Host: bodgeitstore.com

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=7ffa4512
<form method="post" action="/login">
<input type="hidden" name="csrftoken" value="3eff8527">
<input type="text" name="username">
<input type="password" name="password">
<input type="submit" name="submit" value="login">
</form>

~POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: session=7ffa4512
Host: bodgeitstore.com
csrftoken=3eff8527&username=user1&password=secretpwd&submit=login

HTTP/1.1 200 OK
Content-Type: text/html
<b>Welcome!</b>

servers: bodgeit;
init:
  bodgeit knows t1,t2;
client knows t3,t4;
messages:
  request(server=bodgeit, type=req-helo),
  response(server=bodgeit, type=resp-helo, fields=(jsid in setcookie, csrf in body)),
  request(server=bodgeit, type=req-login, fields=(rcsr in urlparam, rjsid in cookie, username in urlparam, password in urlparam)),
  response(server=bodgeit, type=resp-login);
invariants:
  resp-helo.jsid isa t1;
  resp-helo.csrf isa t2;
  req-login.username isa t3;
  req-login.password isa t4;
forall m1:resp-helo, m2:req-helo {
  m1.jsid == m2.rjsid <=> m1.csrf == m2.rcsr;
Running Example: Synthesis

~GET /login HTTP/1.1
Host: bodgeitstore.com

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=7ff4512
<form method="post" action="/login">
  <input type="hidden" name="csrftoken" value="3eff8527">
  <input type="text" name="username">
  <input type="password" name="password">
  <input type="submit" name="submit" value="login">
</form>

~POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: session=7ff4512
Host: bodgeitstore.com
csrftoken=3eff8527&username=user1&password=secretpwd&submit=login

HTTP/1.1 200 OK
Content-Type: text/html
<b>Welcome!</b>

servers: bodgeit;
init:
  bodgeit knows t1,t2;
  client knows t3,t4;
messages:
  request(server=bodgeit, type=req-helo),
  response(server=bodgeit, type=resp-helo,
    fields=(jsid in setcookie, csrf in body)),
  request(server=bodgeit, type=req-login,
    fields=(rcsrf in urlparam, rjsid in cookie,
      username in urlparam, password in urlparam)),
  response(server=bodgeit, type=resp-login);
invariants:
  resp-helo.jsid isa t1;
  resp-helo.csrf isa t2;
  req-login.username isa t3;
  req-login.password isa t4;
  forall m1:resp-helo, m2:req-helo { m1.jsid == m2.rjsid <=> m1.csrf == m2.rcsrf; }
Running Example: Synthesis

~GET /login HTTP/1.1
Host: bodgeitstore.com

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=7ffa4512
<form method="post" action="/login">
<input type="hidden" name="csrftoken" value="3eff8527">
<input type="text" name="username">
<input type="password" name="password">
<input type="submit" name="submit" value="login">
</form>

~POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: session=7ffa4512
Host: bodgeitstore.com
csrftoken=3eff8527&username=user1&password=secretpwd&submit=login

HTTP/1.1 200 OK
Content-Type: text/html
<b>Welcome!</b>

servers: bodgeit;
init: bodgeit knows t1,t2;
client knows t3,t4;
messages:

request(server=bodgeit, type=req-helo),
response(server=bodgeit, type=resp-helo,
fields=(jsid in setcookie, csrf in body)),
request(server=bodgeit, type=req-login,
fields=(rcsrftoken in urllparam, rjsid in cookie,
username in urllparam, password in urllparam)),
response(server=bodgeit, type=resp-login);
invariants:
resp-helo.jsid isa t1;
resp-helo.csrf isa t2;
req-login.username isa t3;
req-login.password isa t4;
forall m1:resp-helo, m2:req-helo {
    m1.jsid == m2.rjsid <=>
    m1.csrf == m2.rcsrftoken;
}
Running Example: Synthesis

~GET /login HTTP/1.1
Host: bodgeitstore.com

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=7ffa4512
<form method="post" action="/login">
<input type="hidden" name="csrftoken" value="3eff8527">
<input type="text" name="username">
<input type="password" name="password">
<input type="submit" name="submit" value="login">
</form>

~POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: session=7ffa4512
Host: bodgeitstore.com
csrftoken=3eff8527&username=user1&password=secretpwd&submit=login

HTTP/1.1 200 OK
Content-Type: text/html
<b>Welcome!</b>

servers: bodgeit;
init:
    bodgeit knows t1,t2;
    client knows t3,t4;
messages:
    request(server=bodgeit, type=req-helo),
    response(server=bodgeit, type=resp-helo,
        fields=(jsid in setcookie, csrf in body)),
    request(server=bodgeit, type=req-login,
        fields=(rcsrf in urlparam, rjsid in cookie,
            username in urlparam, password in urlparam)),
    response(server=bodgeit, type=resp-login);
invariants:
    resp-helo.jsid isa t1;
    resp-helo.csrf isa t2;
    req-login.username isa t3;
    req-login.password isa t4;
    forall m1:resp-helo, m2:req-helo {
        m1.jsid == m2.rjsid <-> m1.csrf == m2.rcsrf;
    }
Running Example: Synthesis

~GET /login HTTP/1.1
Host: bodgeitstore.com

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=7ffa4512
<form method="post" action="/login">
<input type="hidden" name="csrftoken" value="3eff8527">
<input type="text" name="username">
<input type="password" name="password">
<input type="submit" name="submit" value="login">
</form>

~POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: session=7ffa4512
Host: bodgeitstore.com
csrftoken=3eff8527&username=user1&password=secretpwd&submit=login

HTTP/1.1 200 OK
Content-Type: text/html
<b>Welcome!</b>

servers: bodgeit;
init:
  bodgeit knows t1,t2;
client knows t3,t4;
messages:
  request(server=bodgeit, type=req-helo),
  response(server=bodgeit, type=resp-helo,
  fields=(jsid in setcookie, csrf in body)),
  request(server=bodgeit, type=req-login,
  fields=(rcsrf in urlparam, rjsid in cookie,
  username in urlparam, password in urlparam)),
  response(server=bodgeit, type=resp-login);
invariants:
  resp-helo.jsid isa t1;
  resp-helo.csrf isa t2;
  req-login.username isa t3;
  req-login.password isa t4;
forall m1:resp-helo, m2:req-helo {
  m1.jsid == m2.rjsid <> m1.csrf == m2.rcsrf;
}
# Running Example: Synthesis

~~~
~GET /login HTTP/1.1
Host: bodgeitstore.com

HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: session=7ffa4512
<form method="post" action="/login">
<input type="hidden" name="csrftoken" value="3eff8527">
<input type="text" name="username">
<input type="password" name="password">
<input type="submit" name="submit" value="login">
</form>

~POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: session=7ffa4512
Host: bodgeitstore.com
csrftoken=3eff8527&username=user1&password=secretpwd&submit=login

HTTP/1.1 200 OK
Content-Type: text/html
<br>Welcome!\n
~~~

servers: bodgeit;
init:
  bodgeit knows t1,t2;
  client knows t3,t4;
messages:
  request(server=bodgeit, type=req-helo),
  response(server=bodgeit, type=resp-helo,
    fields=(jsid in setcookie, csrf in body)),
  request(server=bodgeit, type=req-login,
    fields=(rcsrf in urlparam, rjsid in cookie,
    username in urlparam, password in urlparam)),
  response(server=bodgeit, type=resp-login);
invariants:
  resp-helo.jsid isa t1;
  resp-helo.csrf isa t2;
  req-login.username isa t3;
  req-login.password isa t4;
  forall m1:resp-helo, m2:req-helo {
    m1.jsid == m2.rjsid <=< m1.csrf == m2.rcsrf;
  }

```
Running Example: Checking

CSRF:

```
pred isCSRF[r: HTTPRequest] {  
(some r.prev and r.prev in MaliciousRedirectionResponse)  
(r.from = VictimClient)  
(r.to in VictimServer))  
some (r.payload − r.cookies)  
attackerCanLearn(r.payload − r.cookies)  
}
```
Running Example: Checking

**CSRF:**
1. Malicious server serves malicious web page to victim client

```plaintext
pred isCSRF[r: HTTPRequest] {
    (some r.prev and r.prev in MaliciousRedirectionResponse)
    (r.from = VictimClient)
    (r.to in VictimServer))
    some (r.payload − r.cookies)
    attackerCanLearn(r.payload − r.cookies)
}
```
Running Example: Checking

CSRF:
1. Malicious server serves malicious web page to victim client
2. Malicious web page sends request to victim server
   ○ uses existing cookies
   ○ attacker controls the other parameters

```plaintext
pred isCSRF[r: HTTPRequest] {
    (some r.prev and r.prev in MaliciousRedirectionResponse)
    (r.from = VictimClient)
    (r.to in VictimServer))
    some (r.payload - r.cookies)
    attackerCanLearn(r.payload - r.cookies)
}
```
Running Example: Checking

CSRF:
1. Malicious server serves malicious web page to victim client
2. Malicious web page sends request to victim server
3. Victim server performs action and responds to victim client

```plaintext
pred isCSRF[r: HTTPRequest] {
  (some r.prev and r.prev in MaliciousRedirectionResponse)
  (r.from = VictimClient)
  (r.to in VictimServer))
  some (r.payload = r.cookies)
  attackerCanLearn(r.payload = r.cookies)
}
```
Running Example: Checking

**CSRF:**
1. Malicious server serves malicious web page to victim client
2. Malicious web page sends request to victim server
3. Victim server performs action and responds to victim client

**Rule encoding:**
- Request caused by malicious page

```prolog
pred isCSRF(r: HTTPRequest) { 
  (some r.prev and r.prev in MaliciousRedirectionResponse) 
  (r.from = VictimClient) 
  (r.to in VictimServer)) 
  some (r.payload − r.cookies) 
  attackerCanLearn(r.payload − r.cookies) 
}
```
Running Example: Checking

CSRF:
1. Malicious server serves malicious web page to victim client
2. Malicious web page sends request to victim server
3. Victim server performs action and responds to victim client

Rule encoding:
- Request caused by malicious page
- The victim client sent it to the victim server

```latex
pred isCSRF(r: HTTPRequest) {
    (some r.prev and r.prev in MaliciousRedirectionResponse)
    (r.from = VictimClient)
    (r.to in VictimServer))
some (r.payload - r.cookies)
attackerCanLearn(r.payload - r.cookies)
}
```
Running Example: Checking

CSRF:
1. Malicious server serves malicious web page to victim client
2. Malicious web page sends request to victim server
3. Victim server performs action and responds to victim client

Rule encoding:
- Request caused by malicious page
- The victim client sent it to the victim server

```text
pred isCSRF(r: HTTPRequest) {
    (some r.prev and r.prev in MaliciousRedirectionResponse)
    (r.from = VictimClient)
    (r.to in VictimServer))
    some (r.payload − r.cookies)
    attackerCanLearn(r.payload − r.cookies)
}
```
Running Example: CSRF Token

servers: bodgeit;
init:
  bodgeit knows t1,t2;
  client knows t3,t4;
messages:
  request(server=bodgeit, type=req-helo),
  response(server=bodgeit, type=resp-helo,
    fields=(jsid in setcookie, csrf in body)),
  request(server=bodgeit, type=req-login,
    fields=(rcsr in urlparam, rjsid in cookie, 
      username in urlparam, password in urlparam)),
  response(server=bodgeit, type=resp-login);
invariants:
  resp-helo.jsid isa t1;
  resp-helo.csrf isa t2;
  req-login.username isa t3;
  req-login.password isa t4;
  forall m1:resp-helo, m2:req-helo {
    m1.jsid == m2.rjsid <=> m1.csrf == m2.rcsr;
  }

pred isCSRF[r: HTTPRequest] {
  (some r.prev and r.prev in MaliciousRedirectionResponse)
  (r.from = VictimClient)
  (r.to in VictimServer))
  some (r.payload − r.cookies)
  attackerCanLearn(r.payload − r.cookies)
}

UNSAT
Case Study: the CAS Protocol

User

Services
(Relying party)

Identity Provider
The Synthesized Model
The Synthesized Model

/home

redirect, sid

/login?sid

login form, sid&csrf
setcookie:c1

[Diagram showing the flow from home to login form, and then back to home or redirection]
The Synthesized Model

/home

redirect, sid

/login?sid

login form, sid&csrf
setcookie:c1

/auth?user&pwd&sid&csrf
cookie:c1

redirect, sid&ticket
setcookie:c2,c3
The Synthesized Model

/home

redirect, sid

/home

/login?sid

/login?sid

login form, sid&csrf
setcookie:c1

/auth?user&pwn&sid&csrf
cookie:c1

/auth?user&pwn&sid&csrf
cookie:c1

redirect, sid&ticket
setcookie:c2,c3

redirect, sid&ticket
setcookie:c2,c3

/check?sid&ticket

/check?sid&ticket

redirect, setcookie:sid

redirect, setcookie:sid
The Vulnerability

/home

redirect, sid

/login?sid

login form, sid&csrf
setcookie:c1

/auth?user&pwd&sid&csrf
cookie:c1

redirect, sid&ticket
setcookie:c2,c3

/check?sid&ticket

redirect, setcookie:sid
## Preliminary Results

<table>
<thead>
<tr>
<th>Name</th>
<th>#Servers</th>
<th>New Hints</th>
<th>#Msgs</th>
<th>Verif. Time (s)</th>
<th>Vuln.?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>2</td>
<td>None</td>
<td>12</td>
<td>7.17</td>
<td>Y (New)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ignore msg. (-)</td>
<td>12</td>
<td>41.71</td>
<td>Y (Known)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>None</td>
<td>12</td>
<td>&gt;7200</td>
<td>N</td>
</tr>
<tr>
<td>NeedMy Password.com</td>
<td>1</td>
<td>None</td>
<td>8</td>
<td>7.20</td>
<td>Y (New)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ignore msg. (-)</td>
<td>8</td>
<td>9.53</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Input value (+)</td>
<td>8</td>
<td>8.16</td>
<td>Y (Known)</td>
</tr>
<tr>
<td>Govtrak.us</td>
<td>2</td>
<td>None</td>
<td>48</td>
<td>&gt;7200</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ignore URLs (-)</td>
<td>24</td>
<td>699.91</td>
<td>Y (New)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ignore msg. (-)</td>
<td>24</td>
<td>2399.77</td>
<td>Y (New)</td>
</tr>
</tbody>
</table>
ASPIRE’s Architecture

- Core: the encoding of the domain knowledge for a class of applications
- The analyst starts by using examples to demonstrate how the application works
- The synthesizer generates one or more candidate models that
  - conform to the DSL syntax
  - conform to the examples
- The specifications will be inspected and the results will feedback to the synthesizer
Conclusion

● Synthesize models of applications from high-level building blocks
  ○ Constructing the build blocks: manually from observation of common patterns
  ○ Constructing the model: automatically using synthesizers

● Key elements
  ○ The input: execution traces and feedback
  ○ The representation: domain specific languages
  ○ The algorithm: specification synthesis
End of presentation.
Backup slides and graphical resources follow.
∀ ⟨,⟩, security_policy( ⟨,⟩, ⟨⟩) = True

- Session integrity
- Information secrecy
ASPIRE Workflow

System Demonstration

Analyst Feedback

Model and Policy

Results

Backend Solver

Domain Knowledge and DSL

Synthesizer

Validator
Use cases for the generated spec

- Run classic analysis and verification tools
- Translate spec to implementation
- Help better understand the existing systems
ASPIRE for the Web

- Given a multiparty web application and its execution traces
- Generate the specification of the web protocol used by the servers and the client
- Check for session integrity (CSRF) vulnerabilities on the specification
- Return attack traces or refine the specification to reduce false positives and false negatives
Components

- The DSL
- The synthesizer
- The validator