STORM: Refinement Types for Secure Web Applications
Top 10 Web Application Security Risks

**A1: 2017- Injection:** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

**A2: 2017- Broken Authentication:** Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.

**A3: 2017- Sensitive Data Exposure:** Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PI. Attacker may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

**A4: 2017- XML External Entities (XXE):** Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

**A5: 2017- Broken Access Control:** Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users’ accounts, view sensitive files, modify other users’ data, change access rights, etc.

**A6: 2017- Security Misconfiguration:** Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

**A7: 2017- Cross-Site Scripting XSS:** XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
Security checks mixed into business logic

how do we guarantee they satisfy the application’s security policies?
IFC Frameworks

- application code
- policies

- policies are much smaller than application code
Existing IFC Frameworks

- application code
- policies

IFC framework

static
dynamic
Existing IFC Frameworks

- predictable behavior
- no run-time overhead
- no rich policies

- crashes/unpredictable behavior
- run-time overhead
+ support for rich policies
Our solution: **STORM**

- + predictable behavior
- + no run-time overhead
- - no rich policies
- + automatic checking of rich policies

Diagram:

- **application code**
- **policies**
- **STORM**
Our solution: **STORM**

- **Model**
  - read/update

- **Controller**
  - policy
  - information flow
  - sends

- **View**
In this Talk

1. Overview: Wishlist App
2. How does Storm enforce IFC automatically
3. Evaluation
Ranjit Jhala

Godzilla vs. Kong: Special Edition (DVD) $27.99

Throw Throw Burrito by Exploding Kittens - A Dodgeball Card Game - Family-Friendly Party Games - Card Games for Adults, Teens & Kids $19.56

Coding For Dummies (For Dummies (Computers)) $15.19

KaroKing Karaoke Machine for Kids & Adults Wireless Microphone Speaker with Disco Ball, 2 Wireless Bluetooth Microphones & Phone/Tablet Holder - Karaoke Bluetooth Toys for Kids (G100) $199.99

private
Wishlist App

Policy:

Only the owner of a wish should have access to their private wishes.
Wishlist App

User
  name String
  email String

Wish
  owner UserId
  description String
Wishlist App

User
  name  String
  email  String

Wish
  owner  UserId
  description  String
  accessLevel  String

- "public"
- "private"
Wishlist App

User
  name  String
  email String

Wish
  owner  UserId
  description String
  accessLevel String

def OnlyPublic(wish, viewer):
    wish.accessLevel = "public" ||
    wish.owner = viewer
Wishlist App

User
- name: String
- email: String

Wish
- owner: UserId
- description: String
- accessLevel: String

```python
@OnlyPublic

def OnlyPublic(wish, viewer):
    wish.accessLevel = "public" ||
    wish.owner = viewer
```
def OnlyPublic(wish, viewer):
    wish.accessLevel == "public" ||
    wish.owner == viewer

User
    name String
    email String

Wish
    owner UserId
    description String
    accessLevel String

showWishes uid = do {
    wishes <- select (Owner == . uid);
    descrs <- project Description wishes;
    respond descrs;
}
def OnlyPublic(wish, viewer):
    wish.accessLevel == "public" ||
    wish.owner == viewer

showWishes uid = do {
    wishes ← select (Owner == . uid);
    descrs ← project Description wishes;
    respond descrs;
}
Wishlist App

```python
def OnlyPublic(wish, viewer):
    wish.accessLevel = "public" ||
    wish.owner == viewer
```

```
showWishes uid = do {
    wishes ← select (Owner == . uid);
    descrs ← project Description wishes;
    respond descrs;
}
```
def OnlyPublic(wish, viewer):
    wish.accessLevel = "public" ||
    wish.owner = viewer

showWishes uid = do {
    wishes ← select (Owner == . uid);
    descrs ← project Description wishes;
    respond descrs;
}
Wishlist App

def OnlyPublic(wish, viewer):
    wish.accessLevel == "public" || wish.owner == viewer

User
    name  String
    email String

Wish
    owner   UserId
    description String
    accessLevel String

read [description] @OnlyPublic

showWishes uid = do {
    wishes ← select (Owner == . uid);
    descrs ← project Description wishes;
    respond descrs;
}

sessionUser ≠ uid

all wishes owned by uid

only viewable by uid
Wishlist App

```python
User
    name   String
    email  String

Wish
    owner   UserId
    description  String
    accessLevel  String

    read [description] @OnlyPublic

def OnlyPublic(wish, viewer):
    wish.accessLevel == "public" ||
    wish.owner == viewer

showWishes uid = do {
    viewer ← requireSessionUser;
    wishes ← select (Owner == .uid);
    descrs ← project Description wishes;
    respond descrs;
}
```
**Wishlist App**

```python
def OnlyPublic(wish, viewer):
    wish.accessLevel = "public" ||
    wish.owner == viewer
```

```sql
showWishes uid = do {
    viewer ← requireSessionUser;
    wishes ←
        if viewer == uid
        then select (Owner == uid)
        else select (Owner == uid &&
                        AccessLevel == "public")
    descrs ← project Description wishes;
    respond descrs;
}
```
Ranjit's Wishes

Godzilla vs. Kong: Special Edition (DVD)
Price: $27.99
Rank: 1

Throw Throw Burrito by Exploding Kittens - A Dodgeball Card Game - Family-Friendly Party Games - Card Games for Adults, Teens & Kids
Price: $19.56
Rank: 2

Coding For Dummies (For Dummies (Computers))
Price: $15.19
Rank: 3

KaraoKing Karaoke Machine for Kids & Adults Wireless Microphone Speaker with Disco Ball, 2 Wireless Bluetooth Microphones & Phone/Tablet Holder - Karaoke Bluetooth Toys
Price: $49.99
Rank: 4
Wishlist App

Policy:

Only the owner of a wish should have access to their private wishes and public wishes should only be viewable by followers.
Wishlist App

User
- name: String
- email: String

Wish
- owner: UserId
- description: String
- accessLevel: String

Follows
- follower: UserId
- followee: UserId
- status: String
  - "pending"
  - "accepted"
  - "rejected"
Wishlist App

User
  name  String
  email String

Wish
  owner  UserId
  description String
  accessLevel String

Follows*
  follower  UserId
  followee  UserId
  status    String

*witness that follower follows followee
Wishlist App

User
  name  String
  email String

Wish
  owner  UserId
  description  String
  accessLevel  String

Follows
  follower  UserId
  followee  UserId
  status  String

declare follows :: UserId → UserId → Bool
Wishlist App

User
- name: String
- email: String

Wish
- owner: UserId
- description: String
- accessLevel: String

Follows
- follower: UserId
- followee: UserId
- status: String

assert @AcceptedFollows

declare follows :: UserId → UserId → Bool

def AcceptedFollows(f):
    if f.status == "accepted":
        follows(f.follower, f.followee)
Wishlist App

```python
User
    name String
    email String

Wish
    owner UserId
    description String
    accessLevel String

    read [description] @OwnerOrFollower

Follows
    follower UserId
    followee UserId
    status String

    assert @Accepted

declare follows :: UserId → UserId → Bool

def AcceptedFollows(f):
    f.status == "accepted" ⇒
    follows(f.follower, f.followee)

def OwnerOrFollower(wish, viewer):
    wish.owner == viewer ||
    ( wish.accessLevel == "public" &&
      follows(viewer, wish.owner) )
```
Wishlist App

```haskell
notifyFollowers uid = do {
    -- Get list of wishes
    wishes <- select (Owner == uid && AccessLevel == "public");
    descrs <- project Description wishes;

    -- Get list of followers
    follows <- select (Followee == uid && Status == "accepted");
    followerIds <- project Follower follows;
    followers <- select (UserId <- followerIds);

    -- Notify followers
    sendMail followers descrs;
}
```
Wishlist App

```
notifyFollowers uid = do {
  -- Get list of wishes
  wishes ← select (Owner == uid && AccessLevel == "public");
  descrs ← project Description wishes;

  -- Get list of followers
  follows ← select (Followee == uid && Status == "accepted");
  followerIds ← project Follower follows;
  followers ← select (UserId ← followerIds);

  -- Notify followers
  sendMail followers descrs;
}
```
Wishlist App

```
notifyFollowers uid = do {
  -- Get list of wishes
  wishes <- select (Owner == uid && AccessLevel == "public");
  descrs <- project Description wishes;

  -- Get list of followers
  follows <- select (Followee == uid && Status == "accepted");
  followerIds <- project Follower follows;
  followers <- select (UserId <- followerIds);

  -- Notify followers
  sendMail followers descrs;
}
```
Wishlist App

```haskell
notifyFollowers uid = do {
    -- Get list of wishes
    wishes ← select (Owner ==. uid &&. AccessLevel == "public");
    descrs ← project Description wishes;

    -- Get list of followers
    follows ← select (Followee ==. uid &&. Status == "accepted");
    followerIds ← project Follower follows;
    followers ← select (UserId ←. followerIds);

    -- Notify followers
    sendMail followers descrs;
}
```
In this Talk

1. Overview: Wishlist App

2. How does Storm enforce IFC automatically

3. Evaluation
Authorizees and Observers

```haskell
notifyFollowers uid = do 
  -- Get list of wishes
  wishes ← select (Owner == uid && AccessLevel == "public");
  descrs ← project Description wishes;

  -- Get list of followers
  follows ← select (Followee == user && Status == "accepted");
  followerIds ← project Follower follows;
  followers ← select (UserId ← followerIds);

  -- Notify followers
  sendMail followers descrs;
```
Compositional IFC

controller = s1 ; s2 ; s3 ; s4

s :: \langle auth, obs \rangle

under-approximates    over-approximates
Compositional IFC

\[ \text{obs}_2 \subseteq \text{auth}_1 \]

\[ \text{s}_1 \quad ; \quad \text{s}_2 \]

\[ \langle \text{auth}_1, \text{obs}_1 \rangle \quad ; \quad \langle \text{auth}_2, \text{obs}_2 \rangle \]

\[ \langle \text{auth}_1 \cap \text{auth}_2, \text{obs}_1 \cup \text{obs}_2 \rangle \]
Sets as Predicates

\[ \lambda u \rightarrow true \]

“The set of all users”

\[ \lambda u \rightarrow false \]

“The empty set”

\[ \lambda u \rightarrow \text{follows}(u, \text{uid}) \]

“The set of followers of uid”

\[ \lambda u \rightarrow u = \text{sessionUser} \]

“The singleton set \{\text{sessionUser}\}”
Sets as Predicates

\[ u = \text{sessionUser} \implies \text{follows}(u, uid) \]

\[ s_1; \langle \lambda u \rightarrow \text{follows}(u, uid), \lambda u \rightarrow \text{false} \rangle \]

\[ s_2 \langle \lambda u \rightarrow \text{true}, \lambda u \rightarrow u = \text{sessionUser} \rangle \]
notifyFollowers uid = do {
    -- Get list of wishes
    wishes ← select (Owner == uid && AccessLevel == "public");
descrs ← project Description wishes;

    -- Get list of followers
    follows ← select (Followee == uid && Status == "accepted");
    followerIds ← project Follower follows;
    followers ← select (UserId ← followerIds);

    -- Notify followers
    sendMail followers descrs;
}
notifyFollowers uid = do {
    -- Get list of wishes
    wishes ← select (Owner ==. uid &&. AccessLevel ==. "public");
    descrs ← project Description wishes;

    -- Get list of followers
    follows ← select (Followee ==. uid &&. Status ==. "accepted");
    followerIds ← project Follower follows;
    followers ← select (UserId ←. followerIds);

    -- Notify followers
    sendMail followers descrs;
}

notifyFollowers uid = do {
    -- Get list of wishes
    wishes ← select (Owner == uid && AccessLevel == "public");
    descrs ← project Description wishes;

    -- Get list of followers
    follows ← select (Followee == uid && Status == "accepted");
    followerIds ← project Follower follows;
    followers ← select (UserId ← followerIds);  

    -- Notify followers
    sendMail followers descrs;
}
notifyFollowers uid = do {
    -- Get list of wishes
    wishes ← select (Owner == uid && AccessLevel == "public");
    descrs ← project Description wishes;
    
    -- Get list of followers
    follows ← select (Followee == uid && Status == "accepted");
    followerIds ← project Follower follows;
    followers ← select (UserId ← followerIds);
    
    -- Notify followers
    sendMail followers descrs;
}
notifyFollowers uid = do {

-- Get list of wishes
    wishes ← select (Owner == uid && AccessLevel == "public");
    descrs ← project Description wishes;

-- Get list of followers
    follows ← select (Followee == uid && Status == "accepted");
    followerIds ← project Follower follows;
    followers ← select (UserId ← followerIds);

-- Notify followers
    sendMail followers descrs;
}
notifyFollowers uid = do {
  -- Get list of wishes
  wishes ← select (Owner =. uid &&. AccessLevel =. "public");
  descrs ← project Description wishes;

  -- Get list of followers
  follows ← select (Followee =. uid &&. Status =. "accepted");
  followerIds ← project Follower follows;
  followers ← select (UserId ←. followerIds);

  -- Notify followers
  sendMail followers descrs;
}
notifyFollowers uid = do {
  -- Get list of wishes
  wishes <- select (Owner == uid && AccessLevel == "public" );
  descrs <- project Description wishes;

  -- Get list of followers
  follows <- select (Followee == uid && Status == "accepted" );
  followerIds <- project Follower follows;
  followers <- select (UserId <- followerIds);

  -- Notify followers
  sendMail followers descrs;
}
notifyFollowers uid = do {
  -- Get list of wishes
  wishes <- select (Owner == uid && AccessLevel == "public");
  descrs <- project Description wishes;

  -- Get list of followers
  follows <- select (Followee == uid && Status == "accepted");
  followerIds <- project Follower follows;
  followers <- select (UserId <- followerIds);

  -- Notify followers
  sendMail followers descrs;
}
notifyFollowers uid = do {
    -- Get list of wishes
    wishes ← select (Owner == uid &&. AccessLevel == "public");
    descrs ← project Description wishes;

    -- Get list of followers
    follows ← select (Followee == uid &&. Status == "accepted");
    followerIds ← project Follower follows;
    followers ← select (UserId == followerIds);

    -- Notify followers
    sendMail followers descrs;}}\(\lambda u \rightarrow true, \lambda u \rightarrow \text{follows}(u, \text{uid})\)
notifyFollowers uid = do {

-- Get list of wishes
wishes ← select (Owner == uid && AccessLevel == "public");
descrs ← project Description wishes;

-- Get list of followers
follows ← select (Followee == uid && Status == "accepted");
followerIds ← project Follower follows;
followers ← select (UserId ← followerIds);

-- Notify followers
sendMail followers descrs;
}
In this Talk

1. Overview: Wishlist App

2. How does Storm enforce IFC automatically

3. Evaluation
Evaluation

1. Expressiveness
2. Effort
3. Auditability
# Expressiveness

<table>
<thead>
<tr>
<th>System</th>
<th>Benchmark</th>
<th>Model</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>UrFLOW</td>
<td>secret</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>poll</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>calendar</td>
<td>15</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>gradebook</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>forum</td>
<td>19</td>
<td>34</td>
</tr>
<tr>
<td>JACQUELINE</td>
<td>conference</td>
<td>42</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>course</td>
<td>32</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>health</td>
<td>79</td>
<td>23</td>
</tr>
<tr>
<td>HAILS</td>
<td>gitsar</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>LWEB</td>
<td>bibifi</td>
<td>312</td>
<td>101</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>555</td>
<td>314</td>
</tr>
</tbody>
</table>

*all but one policy
Evaluation

1. Expressiveness
2. Effort
3. Auditability
Typing Annotations

{-@ showWishes :: _ -> TaggedT<{\_ -> False}, {\_ -> True}> _ _ _ @-}
showWishes :: UserId -> Controller ()
showWishes user = do {
    wishes <- select (Owner ==. user);
    descrs <- project Description wishes;
    respond descrs;
}
Typing Annotations

{-@ getAuthors :: p: _ \rightarrow\ TaggedT{{\\{u \rightarrow\ PcoAuthorOrAccepted~p~u\},\ \{\_ \rightarrow\ False\}\_\_\_@-}\}

getAuthors :: Entity~Paper \rightarrow\ Controller\ [\text{Text}]

getAuthors~paper = do

  (paperId, authorId) \leftarrow\ project2\ (\text{PaperId, PaperAuthor})\ paper;

  author \leftarrow\ select\ (\text{UserId} ==.\ authorId);

  authors \leftarrow\ \text{mapT}\ (\text{project}\ \text{UserName})\ author;

  coauthors \leftarrow\ select\ (\text{PaperCoauthorPaper} ==.\ paperId);

  coauthorNames \leftarrow\ \text{mapT}\ (\text{project}\ \text{PaperCoauthorAuthor})\ coauthors;

  return\ $\ authors\ ++\ coauthorNames;

}
## Effort

<table>
<thead>
<tr>
<th>Application</th>
<th>LOC</th>
<th>Annot.</th>
<th>Annot./Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>conference</td>
<td>644</td>
<td>43</td>
<td>0.07</td>
</tr>
<tr>
<td>course</td>
<td>198</td>
<td>5</td>
<td>0.03</td>
</tr>
<tr>
<td>wishlist</td>
<td>334</td>
<td>20</td>
<td>0.6</td>
</tr>
</tbody>
</table>

*1 line of annotation per 19 lines of code*
Evaluation

1. Expressiveness

2. Effort

3. Auditability
- PLMW (Jun 2020)
- VMW (Jul 2020)
- 100 Users
- 4 Classes F20-S21
- 50-200 Students
Disco and Voltron Architecture

Storm server

Vue.js client

Sensitive data
## Auditability

<table>
<thead>
<tr>
<th>Application</th>
<th>LOC</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Server</td>
<td>Models</td>
<td>Policy</td>
<td>Client</td>
</tr>
<tr>
<td>voltron</td>
<td>756</td>
<td>32</td>
<td>37</td>
<td>1012</td>
</tr>
<tr>
<td>disco</td>
<td>859</td>
<td>42</td>
<td>32</td>
<td>4630</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1615</td>
<td>74</td>
<td>69</td>
<td>5642</td>
</tr>
</tbody>
</table>

*policy code $<$ 4% of server code*  
(<2% with client code)
STORM

Controller

Model

View

read/update

information flow

sends

policy