STORM: Refinement Types for Secure Web Applications

Nico Lehmann
UC San Diego

Rose Kunkel
UC San Diego

Jordan Brown
Independent

Jean Yang
Akita Software

Niki Vazou
IMDEA Software Institute

Nadia Polikarpova
UC San Diego

Deian Stefan
UC San Diego

Ranjit Jhala
UC San Diego

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 PII. Attackers 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

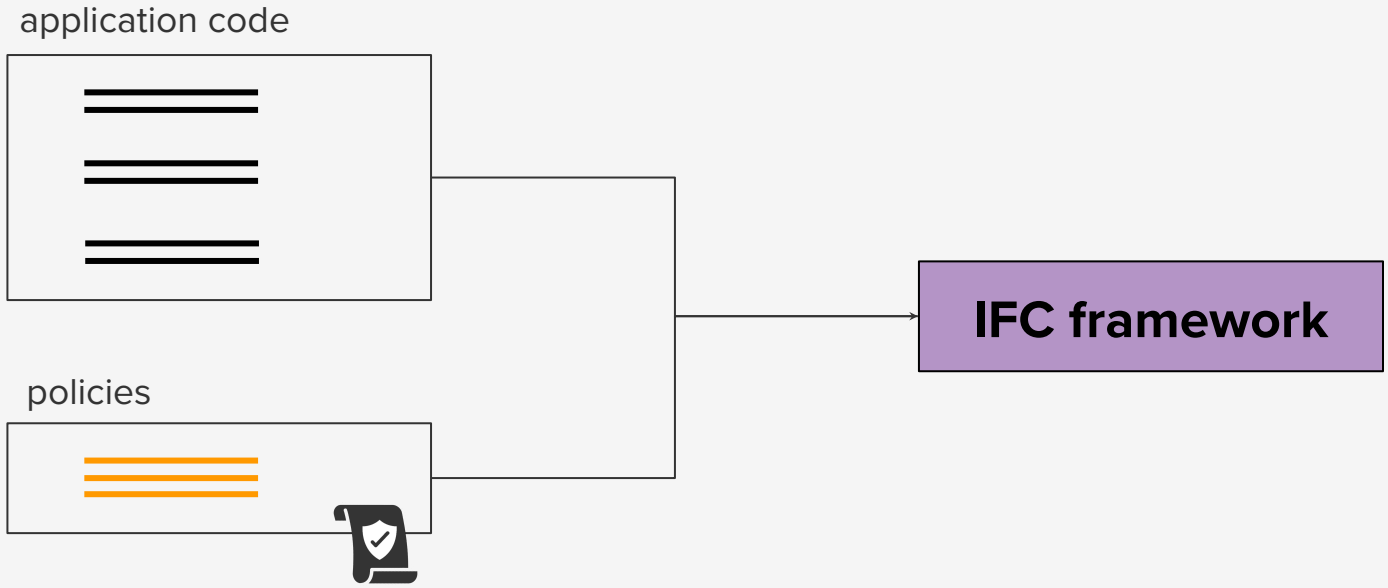
```php
$this->pt = $pt = new PaperTable($this->user, $this->qreq, $this->prow);
$pt->resolve_comments();
if ($pt->can_view_reviews()
    || $pt->mode === "re"
    || ($this->prow->paperId > 0 && $this->user->can_edit_review($this->prow))) {
    $pt->resolve_review(false);
    $pt->fix_mode();
}

// prepare paper table
if ($pt->mode == "edit") {
    $old_overrides = $this->user->remove_overrides(Contact::OVERRIDE_CHECK_TIME);
    $editable = $this->user->can_edit_paper($this->prow);
    if ($this->user->can_edit_final_paper($this->prow)) {
        $editable = "f";
    }
    $this->user->set_overrides($old_overrides);
} else {
    $editable = false;
}

$pt->initialize($editable, $editable && $this->useRequest);
if ($pt->mode === "edit" && !$this->ps) {
    if ($this->prow->paperId <= 0) {
        $this->prow->set_allow_absent(true);
    }
    $this->ps = $this->ps ?? PaperStatus::make_prow($this->user, $this->prow);
    $old_overrides = $this->user->add_overrides(Contact::OVERRIDE_CONFLICT);
    foreach ($this->prow->form_fields() as $o) {
        if ($this->user->can_edit_option($this->prow, $o)) {
            $ov = $this->prow->force_option($o);
            $ov->set_message_set($this->ps);
            $o->value_check($ov, $this->user);
        }
    }
    $this->user->set_overrides($old_overrides);
    $this->prow->set_allow_absent(false);
}
```

```php
if ($Me->can_view_review($prow, $rrow)) {
    $rname = Ht::link($rname, $prow->reviewurl(["r" => $rrow->reviewId]));
}
echo $rname, ': ', $namex,
    '</div><div class="f-h"><ul class="x mb-0">';
echo '<li>requested';
if ($rrow->timeRequested) {
    echo ' ', $Conf->unparse_time_relative((int) $rrow->timeRequested);
}
if ($rrow->requestedBy == $Me->contactId) {
    echo " by you";
} else if ($Me->can_view_review_requester($prow, $rrow)) {
    echo " by ", $Me->reviewer_html_for($rrow->requestedBy);
}
echo '</li>';
if ($rrow->reviewStatus === ReviewInfo::RS_ACCEPTED) {
    echo '<li>accepted';
    if ($req[1]) {
        echo ' ', $Conf->unparse_time_relative($req[1]);
    }
    echo '</li>';
}
```

```php
function my_rrow($prefer_approvable) {
    $myrrow = $apprrow1 = $apprrow2 = null;
    $admin = $this->user->can_administer($this->prow);
    foreach ($this->prow->reviews_as_display() as $rrow) {
        if ($this->user->can_view_review($this->prow, $rrow)) {
            if ($rrow->contactId === $this->user->contactId
                || (!$myrrow && $this->user->is_my_review($rrow))) {
                $myrrow = $rrow;
            } else if ($rrow->reviewStatus === ReviewInfo::RS_DELIVERED
                       && !$apprrow1
                       && $rrow->requestedBy === $this->user->contactXid) {
                $apprrow1 = $rrow;
            } else if ($rrow->reviewStatus === ReviewInfo::RS_DELIVERED
                       && !$apprrow2
                       && $admin) {
                $apprrow2 = $rrow;
            }
        }
    }
    if (($apprrow1 || $apprrow2)
        && ($prefer_approvable || !$myrrow)) {
        return $apprrow1 ?? $apprrow2;
    } else {
        return $myrrow;
    }
}
```

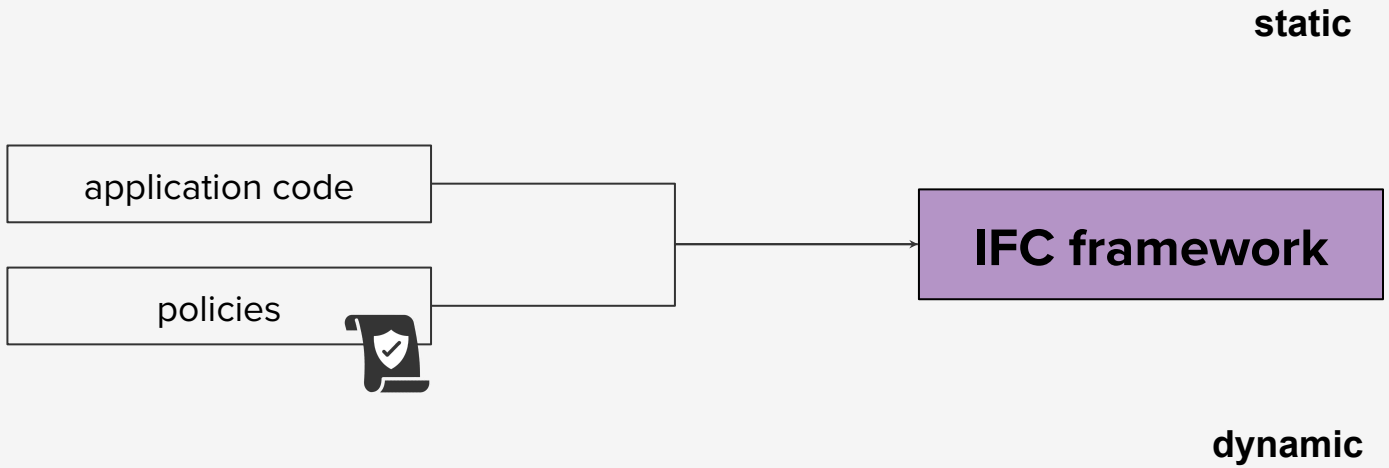
how do we guarantee they satisfy the application's security policies?

IFC Frameworks



policies are much smaller than application code

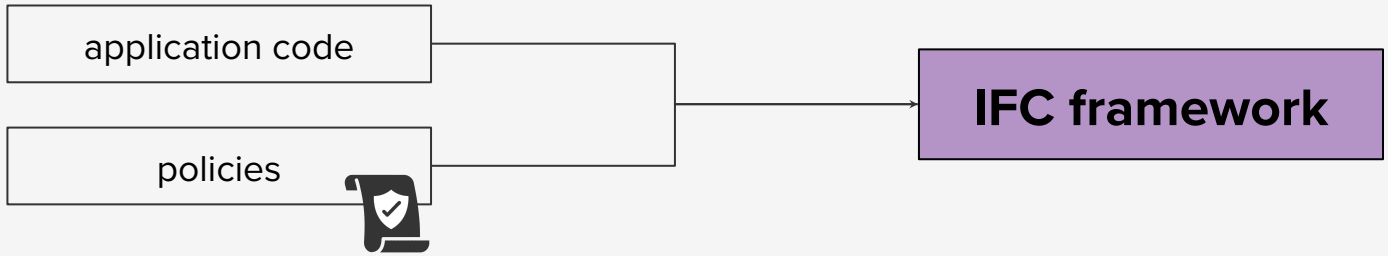
Existing IFC Frameworks



Existing IFC Frameworks

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

static



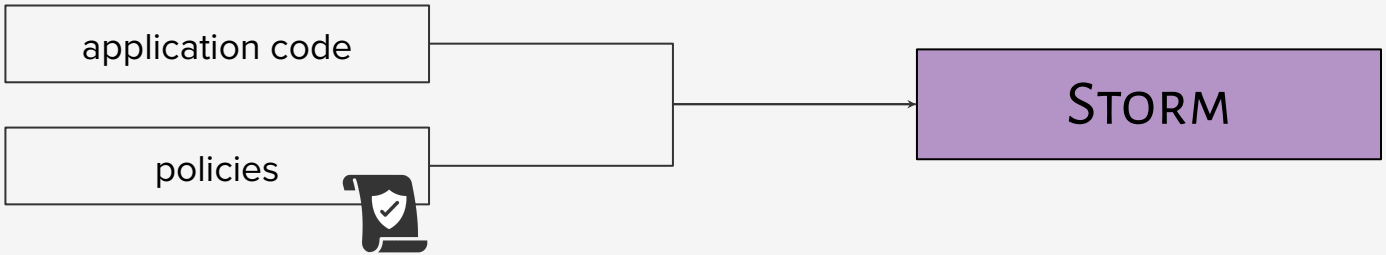
- crashes/unpredictable behavior
- run-time overhead
+ support for rich policies

dynamic

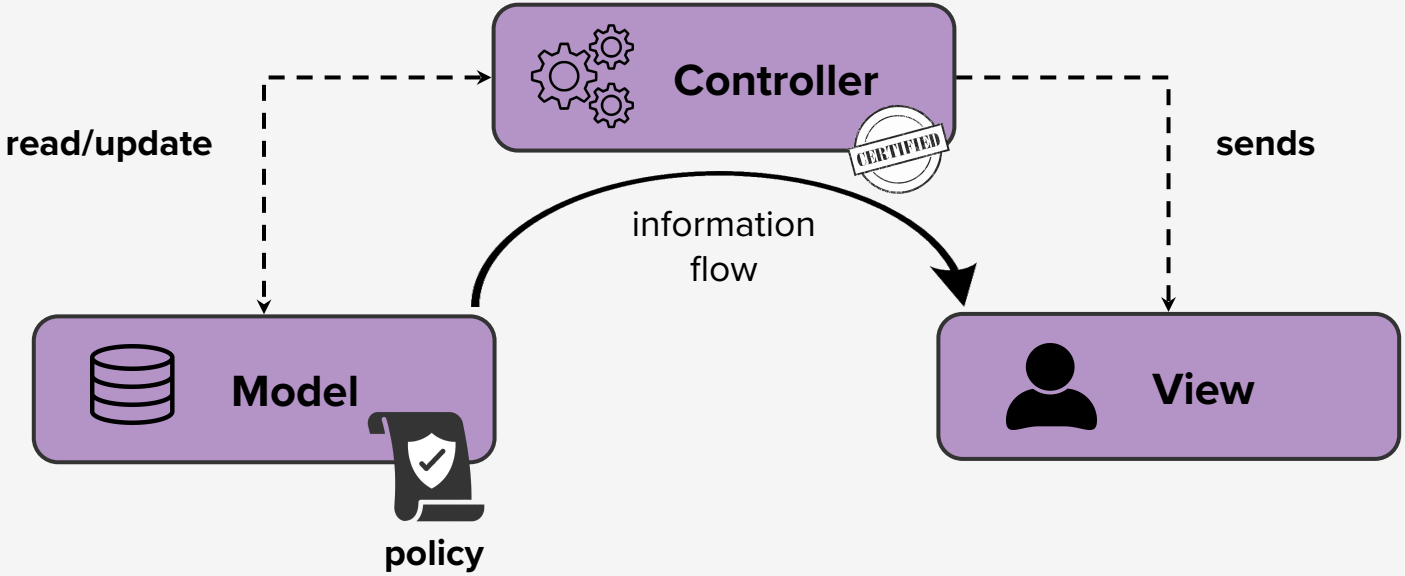
Our solution: STORM

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

static



Our solution: STORM



In this Talk

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

Ranjit's Wishes

Share

Ranjit Jhala

ADD TO THIS WISHLIST

Godzilla vs. Kong: Special Edition (DVD)

$27.99

VIEW GIFTER

DETAILS

Rank: 1 SAVE

DELETE

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

$19.56

VIEW GIFTER

DETAILS

Rank: 2 SAVE

DELETE

private

Coding For Dummies (For Dummies (Computers))

$15.19

I like it - not a gift

DETAILS

Rank: 3 SAVE

DELETE

KaraoKing 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

DETAILS

Rank: 4 SAVE

DELETE

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

  read [description] @OnlyPublic
```

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

Wishlist App

```
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;
}
```

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

Wishlist App

```
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;
}
```

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

Wishlist App

```
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;
}
```

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

Wishlist App

```
User
  name  String
  email String

Wish
  owner       UserId
  description String
  accessLevel String

  read [description] @OnlyPublic
```

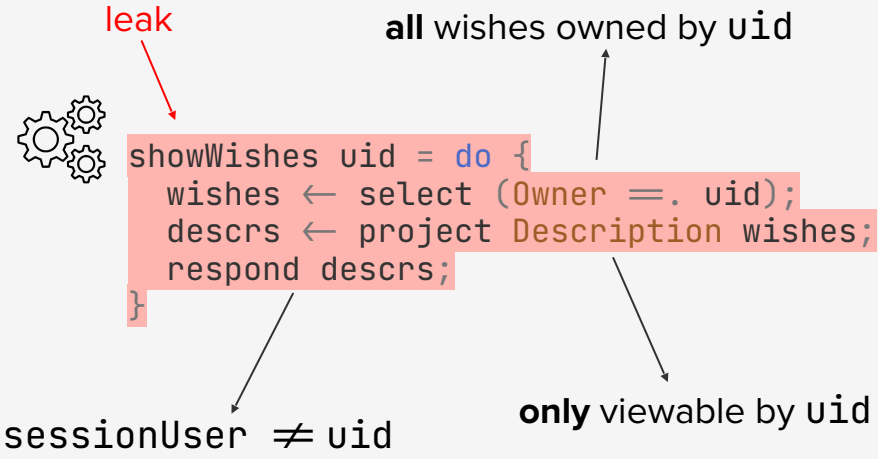
```
showWishes uid = do {
  wishes ← select (Owner ==. uid);
  descrs ← project Description wishes;
  respond descrs;
}
```

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

Wishlist App

```
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
```



Wishlist App

```
User
  name  String
  email String

Wish
  owner       UserId
  description String
  accessLevel String

  read [description] @OnlyPublic
```

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

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

Wishlist App

```
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 ←
    if viewer == uid
    then select (Owner ==. uid)
    else select (Owner ==. uid &&.
                 AccessLevel ==. "public");
  descrs ← project Description wishes;
  respond descrs;
}
```

Ranjit's Wishes

Share

Ranjit Jhala

Followers


Nico


Rose


Jean


Jordan


Deian


Nadia

ADD TO THIS WISHLIST

Godzilla vs. Kong: Special Edition (DVD)

$27.99

VIEW GIFTER

DETAILS

Rank: 1 SAVE

DELETE

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

$19.56

VIEW GIFTER

DETAILS

Rank: 2 SAVE

DELETE

Coding For Dummies (For Dummies (Computers))

$15.19

I like it - not a gift

DETAILS

Rank: 3 SAVE

DELETE

KaraoKing Karaoke Machine for Kids & Adults Wireless Microphone Speaker with Disco Ball, 2 Wireless Bluetooth Microphones & Phone/Tablet Holder - Karaoke Bluetooth Toys

DETAILS

Rank: 4 SAVE

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

Wishlist App

```
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

```
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

```
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

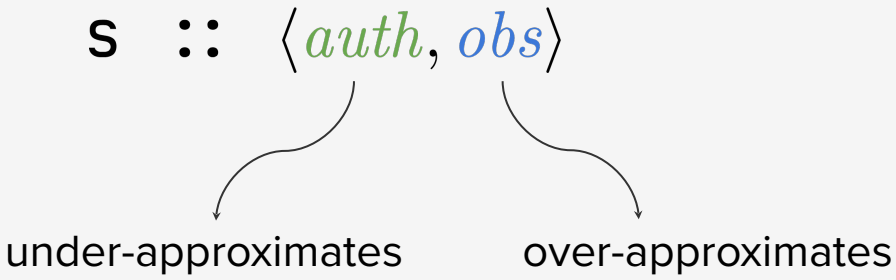
1. Overview: Wishlist App
2. **How does Storm enforce IFC automatically**
3. Evaluation

Authorizees and Observers

```
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
```



Compositional IFC

$$obs_2 \subseteq auth_1$$

s1 ; s2

$\langle auth_1, obs_1 \rangle$ $\langle auth_2, obs_2 \rangle$

$$\langle auth_1 \cap auth_2, obs_1 \cup obs_2 \rangle$$

Sets as Predicates

$\lambda u \rightarrow true$

"The set of all users"

$\lambda u \rightarrow false$

"The empty set"

$\lambda u \rightarrow follows(u, uid)$

"The set of followers of uid"

$\lambda u \rightarrow u = sessionUser$

"The singleton set {sessionUser}"

Sets as Predicates

$$u = sessionUser \implies follows(u, uid)$$

s1; $\langle \lambda u \to follows(u, uid), \lambda u \to false \rangle$

s2 $\langle \lambda u \to true, \lambda u \to u = sessionUser \rangle$

LiquidHaskell

SMT Solver

```
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;
}
```

$\langle \lambda u \to true, \lambda u \to false \rangle$

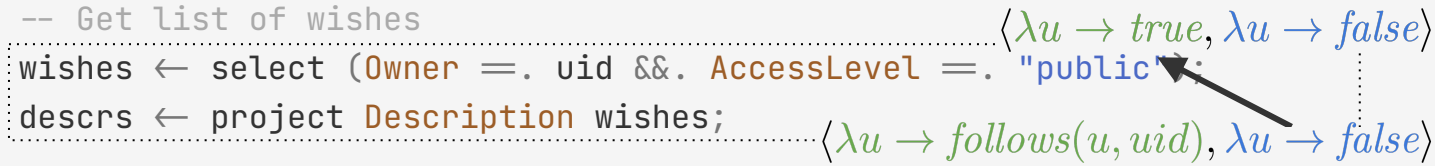
```
notifyFollowers uid = do {
    -- Get list of wishes
    wishes ← select (Owner ==. uid &&. AccessLevel ==. "public");
    descrs ← project Description wishes;

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

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

only **public** wishes owned by **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;
}
```

$\langle \lambda u \to follows(u, uid), \lambda u \to false \rangle$

```
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;
}
```

$\langle \lambda u \to follows(u, uid), \lambda u \to false \rangle$

```
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);
                                    ⟨λu → follows(u, uid), λu → false⟩
    -- 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;
}
```

$\langle \lambda u \to true, \lambda u \to follows(u, uid) \rangle$

```
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;
}
```

$\langle \lambda u \to follows(u, uid), \lambda u \to false \rangle$

$\langle \lambda u \to true, \lambda u \to follows(u, uid) \rangle$

In this Talk

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

Evaluation

1. Expressiveness
2. Effort
3. Auditability

Expressiveness

System	Benchmark	Model	Policy
URFLOW	secret	8	9
	poll	14	16
	calendar	15	29
	gradebook	18	24
	forum	19	34
JACQUELINE	conference	42	46
	course	32	11
	health	79	23
HAILS	gitsar	16	21
LWEB	bibifi	312	101
Total		555	314

*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: _ -> TaggedT<{\u -> PcOrAuthorOrAccepted p u}, {\_ -> False}> _ _ _ @-}
getAuthors :: Entity Paper -> Controller [Text]
getAuthors paper = do {
  (paperId, authorId) <- project2 (PaperId, PaperAuthor) paper;

  author              <- select (UserId ==. authorId);
  authors             <- mapT (project UserName) author;

  coauthors           <- select (PaperCoauthorPaper ==. paperId);
  coauthorNames       <- mapT (project PaperCoauthorAuthor) coauthors;

  return $ authors ++ coauthorNames;

}
```

Effort

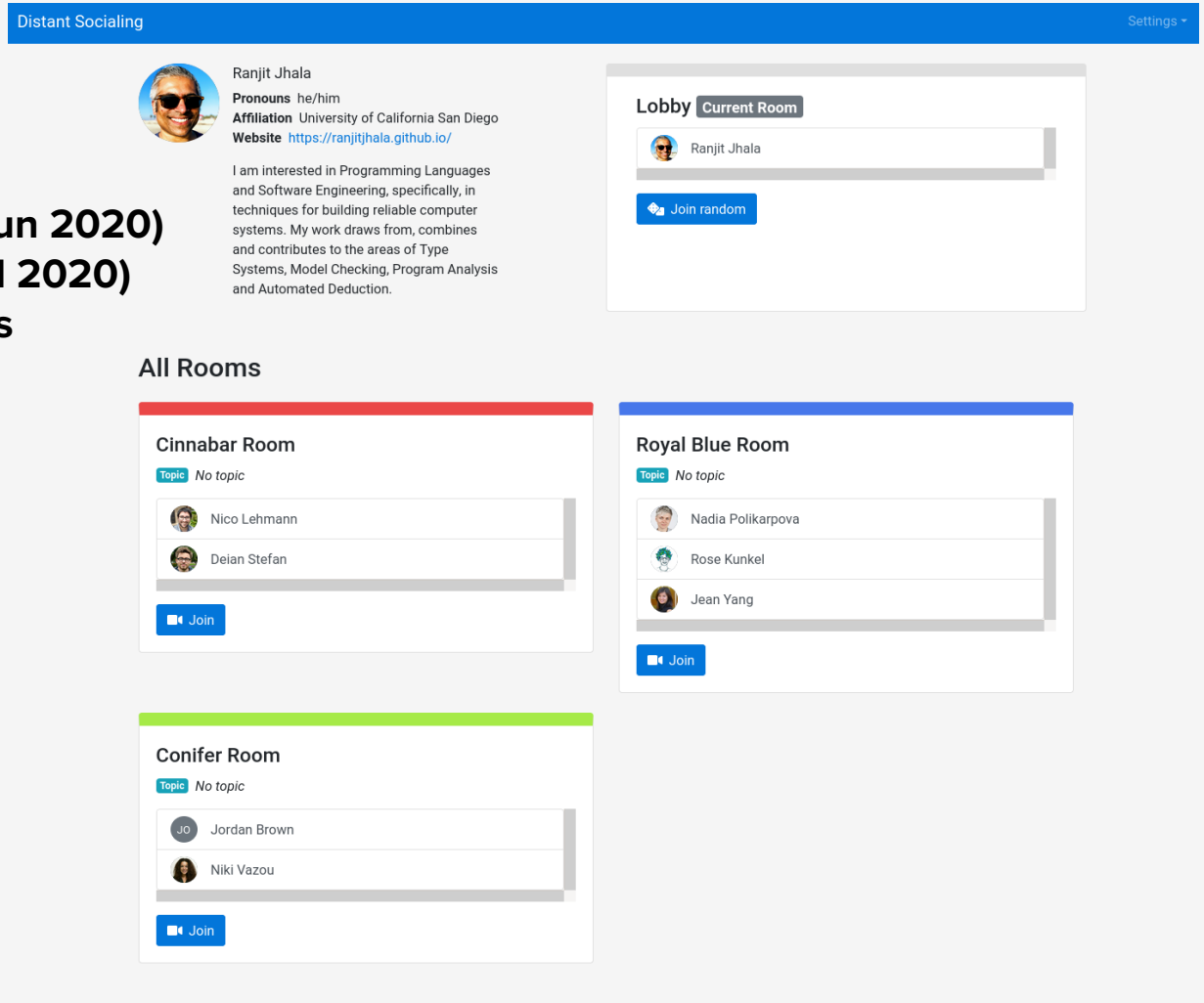
Application	LOC		
	Code	*Annot.*	*Annot./Code*
`conference`	644	43	0.07
`course`	198	5	0.03
`wishlist`	334	20	0.6

*__1__ line of annotation per __19__ lines of code

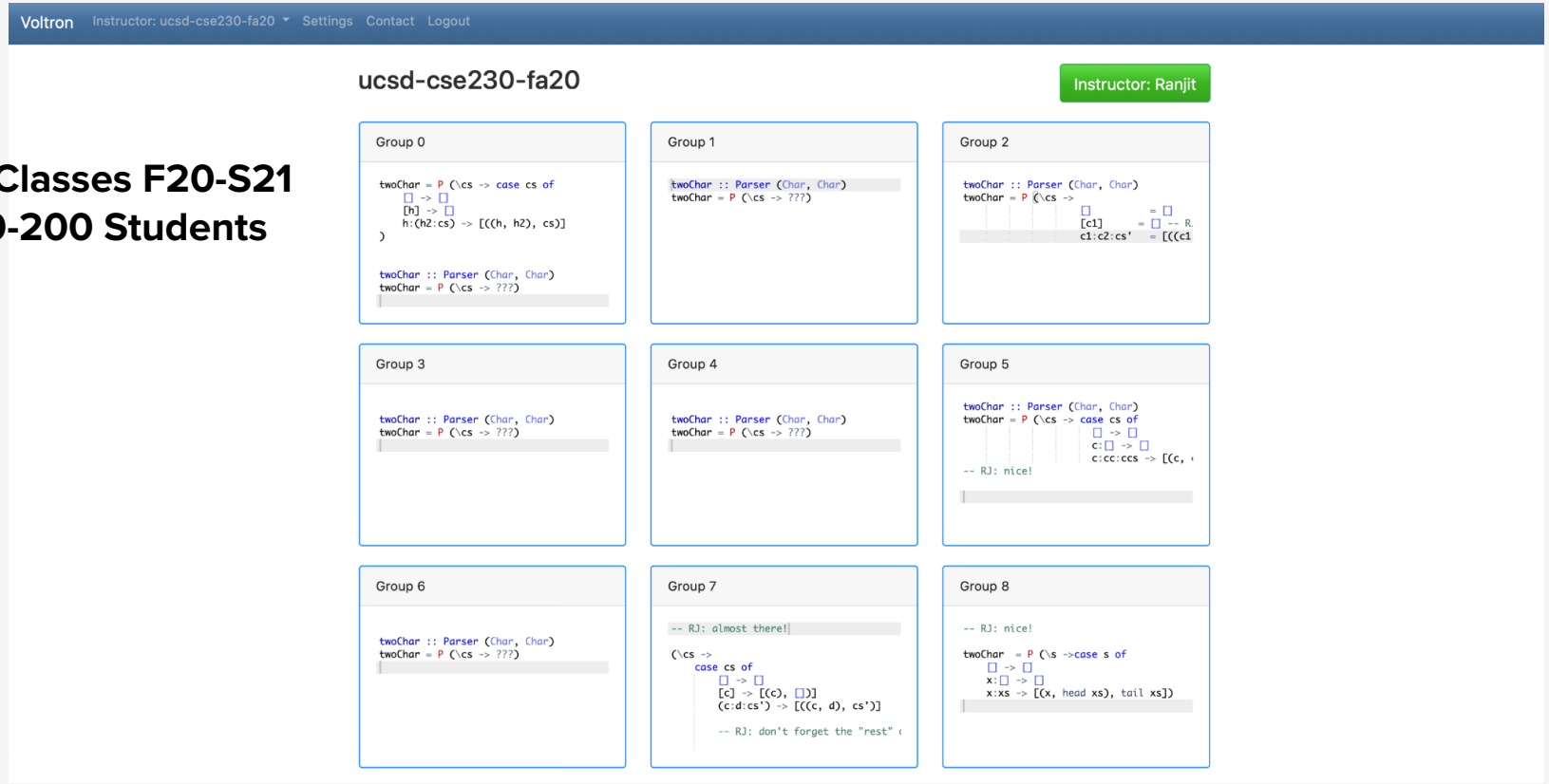
Evaluation

1. Expressiveness
2. Effort
3. **Auditability**

- PLMW (Jun 2020)
- VMW (Jul 2020)
- 100 Users



- **4 Classes F20-S21**
- **50-200 Students**



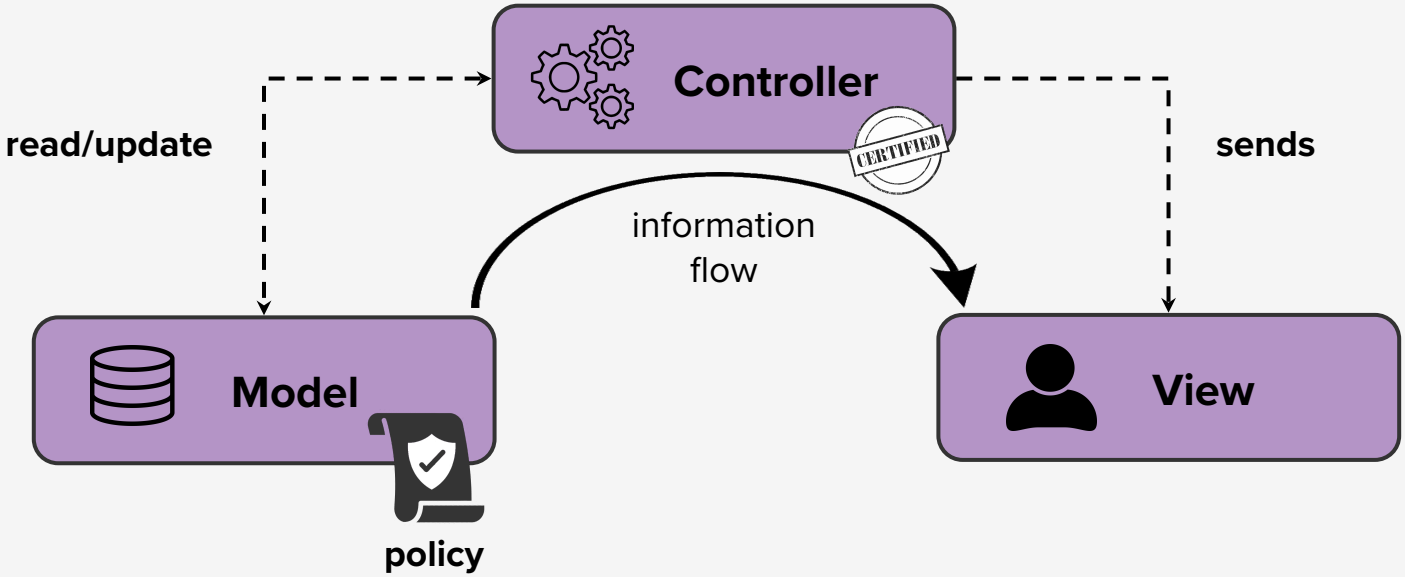
Disco and Voltron Architecture

Auditability

Application	LOC			
	Server	*Models*	*Policy*	*Client*
`voltron`	756	32	37	1012
`disco`	859	42	32	4630
Total	1615	74	69	5642

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

Storm


Controller
CERTIFIED
read/update
sends
information
flow
Model
View
policy