Designed-In Security for Mobile Applications

High Confidence Software and Systems – Designed-In Security

Jonathan Aldrich

(joint work with Michael Maass, Joshua Sunshine, Cyrus Omar, Marwan Abi-Antoun, and Ciera Jaspan)

Carnegie Mellon

isr institute for SOFTWARE RESEARCH

Mobile Apps are Vulnerable

- Examples
  - Siemens SMS Chinese character vulnerability (2003)
  - Commwarrior virus spread via MMS (2006)
  - iPhone jailbreaks based on web browser, PDF (ongoing)
  - Popular apps (Netflix, Google wallet, Wikivest) criticized for insecure password, data storage (2010-2011)

- Factors
  - Mobile apps provide mission-critical information and operations
  - Mobile applications are (typically) distributed
  - Mobile apps inherit web or native app vulnerabilities
  - Models of interaction among mobile apps

Underlying Causes of Vulnerabilities

- Many ways to look at the problem
  - process, coordination, human weakness, etc.

- Hypothesis: many vulnerabilities arise because:
  - desired security properties are **not explicit**;
  - these properties are only **loosely related to code**; and
  - code is written at a **low level of abstraction**

- That is, if it were not for the issues above, we could more readily prevent many vulnerabilities in real software

Tracing Vulnerabilities to Causes

- Consider the OWASP Top 10 web app vulnerabilities
  (shared by many mobile applications)

Vulnerability	**Cause**
1. Command injection	Missing data format; Command created implicitly; Low-level string manipulation
2. Cross-site scripting (XSS)	*Similar to command injection*
3. Broken authentication and sessions	Authentication/sessions model missing or not explicit in code; built out of low-level operations
4. Insecure direct object references	Permissions for accessing object missing or not explicit; enforced at low level
5. Cross-site request forgeries (CSRFs)	Missing models for verifying request origin and intended usage pattern; low-level enforcement

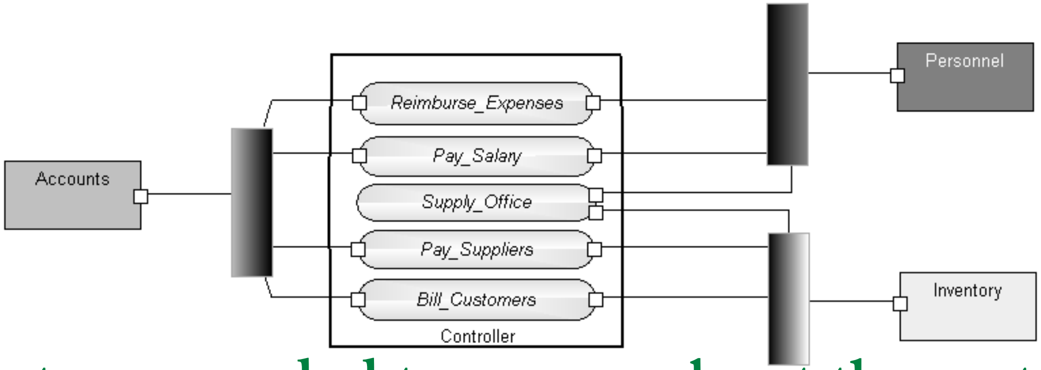
Designing Security In

- Make **design** intent explicit
  - **How** security is enforced
    - Overall application design (e.g. architectural structure)
    - Design choices in code (e.g. protocols, algorithms, data formats)

- Explicitly express **security** constraints
  - **What** properties are required
    - Requirements to call an interface
    - Confidentiality, integrity properties

- Verify design and security **in** code
  - **Unify** design and implementation (via languages, libraries)
    - Opportunity: mobile/web app world is evolving rapidly
  - **Check** implementation against design (via analysis, types, model checking, reviews)

Software Architecture



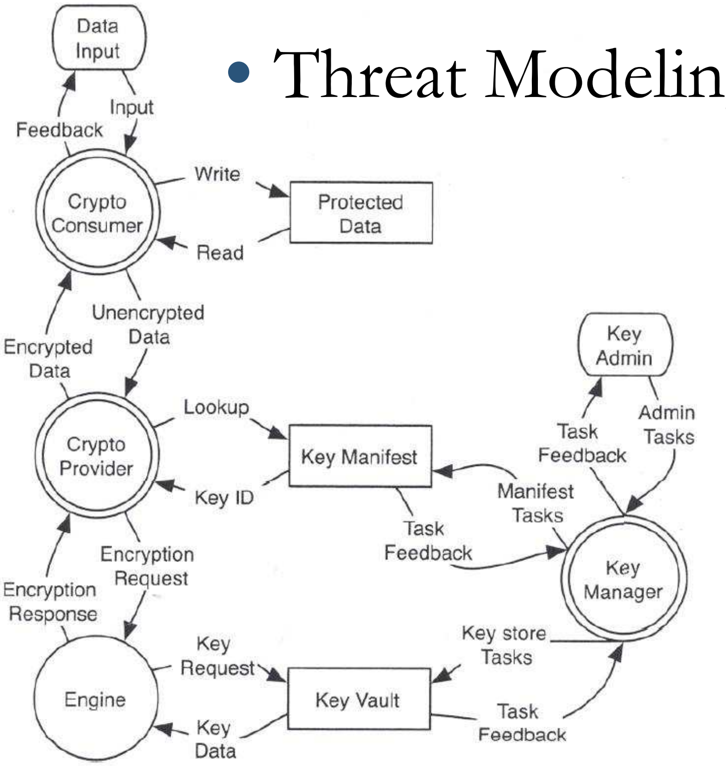
- the set of **structures needed to reason about the system**, which comprise software elements, relations among them, and properties of both – Clements et al.

- the set of **principal design decisions** made about the system – Taylor et al.

- Software architecture enables reasoning about a software system based on its design characteristics.
  - Can we leverage architecture to reason about mobile security?
  - Can we link architecture to application implementation?

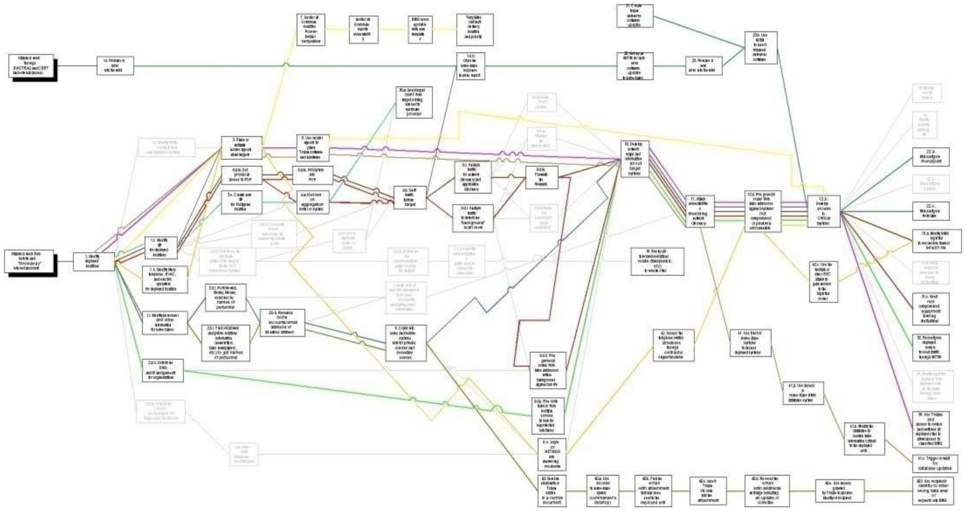
Architectural Reasoning about Security

- Threat Modeling



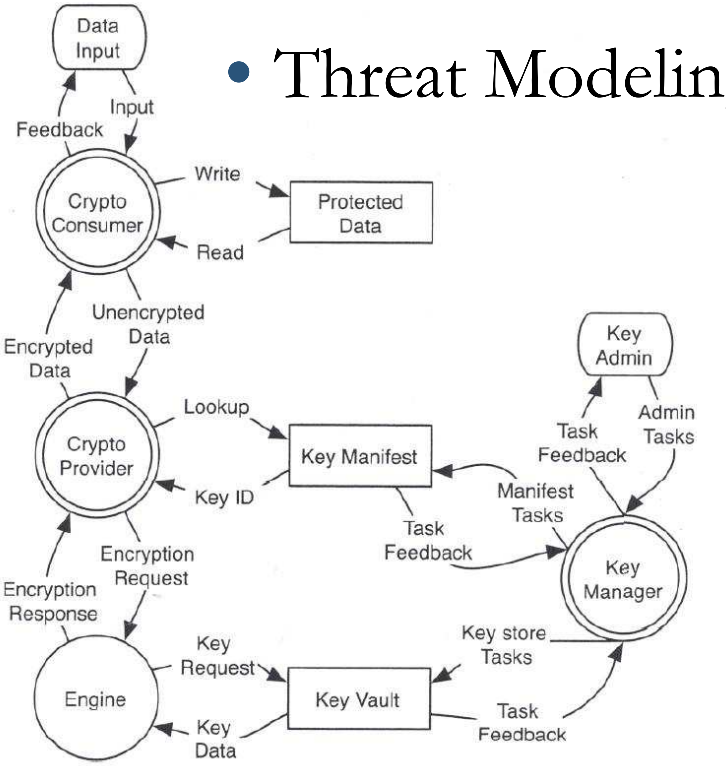
- Data flow diagrams
  - Processes, data, trust
  - Analyzed for attacks
- Used at Microsoft, others

- Attack Graphs



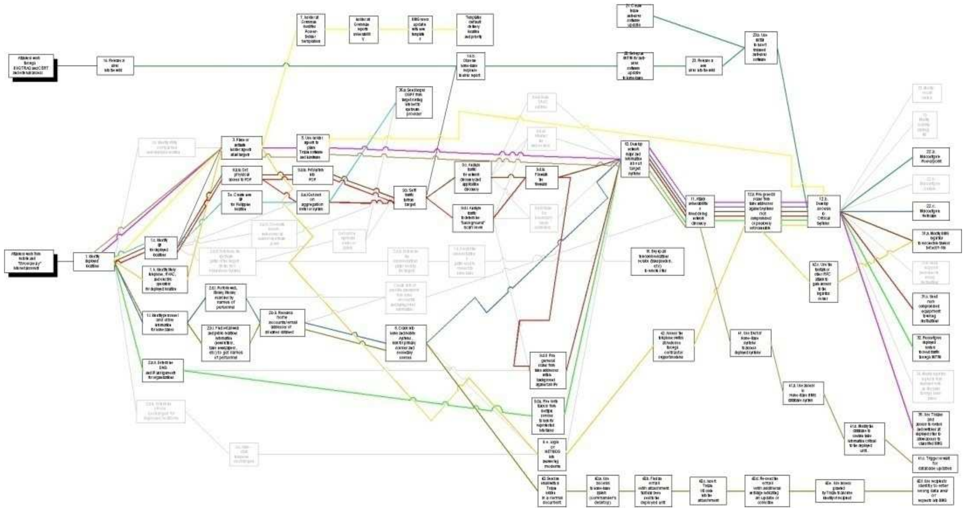
- Possible steps in an attack
- Analyze attack/defense opts.
  - Least cost attack path
  - Coverage of defense strat.

Architectural Reasoning about Security



- Threat Modeling

- Attack Graphs



- I

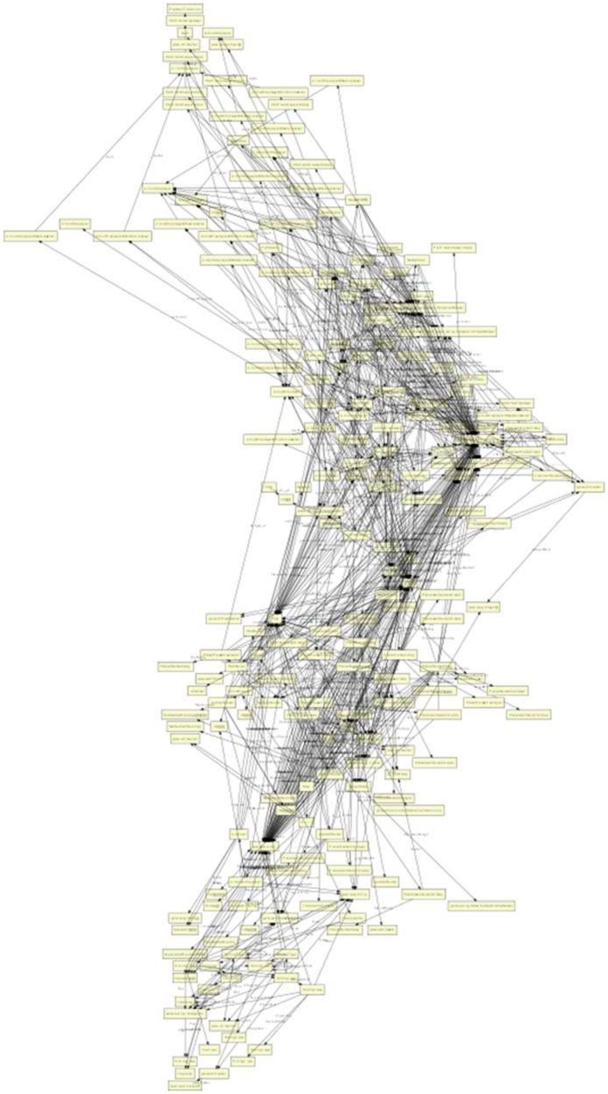
- Possible steps in an attack

opts.

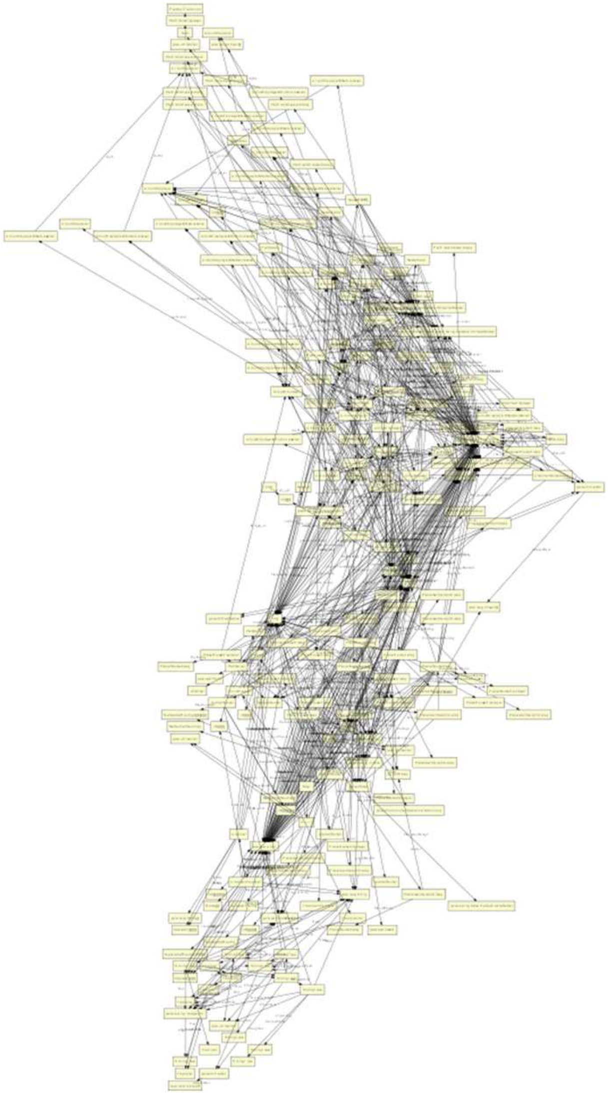
- Used at Microsoft, others

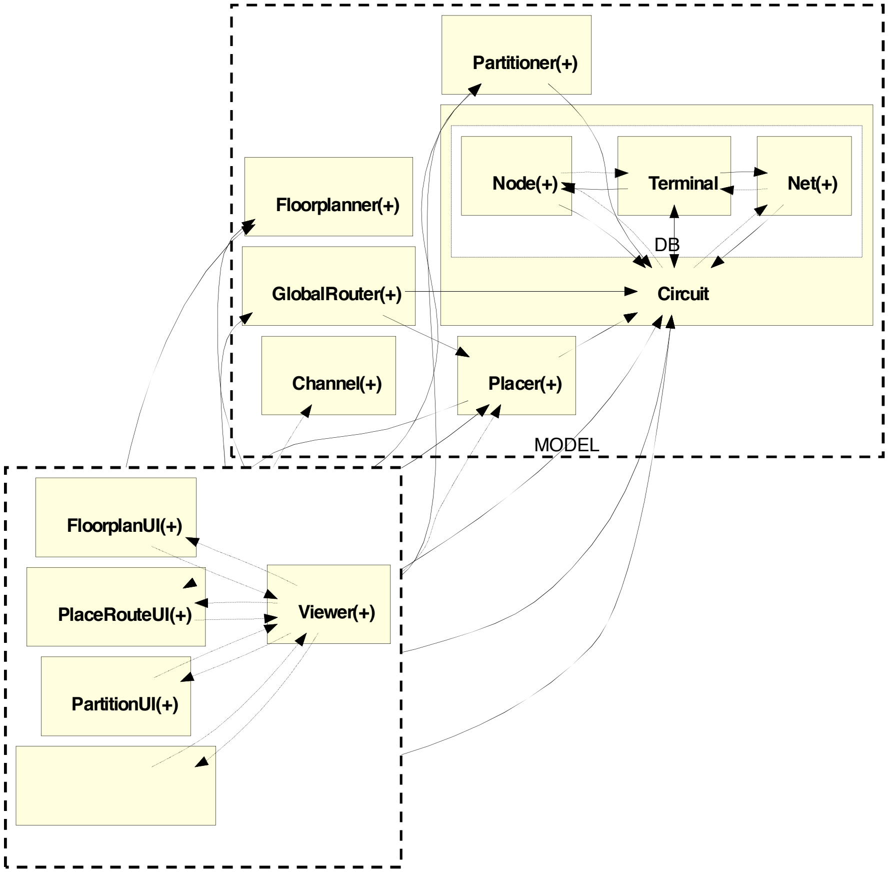
Can we related these architectural reasoning techniques more directly to code?

Architecture: Naïve object graph extraction



Architecture: Design Intent Approach



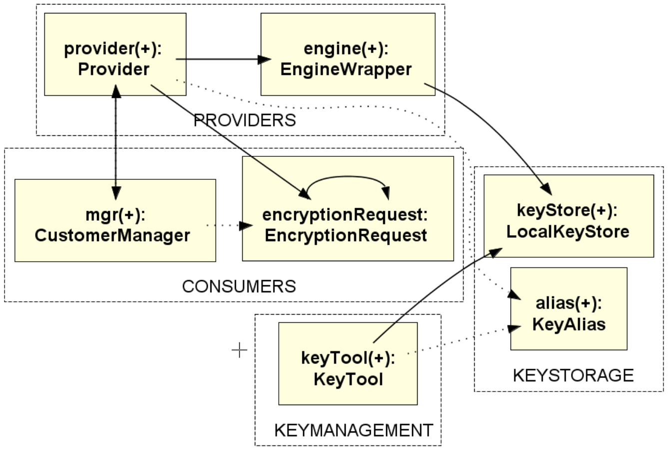


[Abi-Antoun & A, OOPSLA ‘09]

Architectural Design Intent

- Labeled groups
  - @Domain: Put in logical part of architecture

```
class Main {
                    Provider provider;
                     CustomerManager mgr;
                      LocalKeyStore keyStore;
}
```

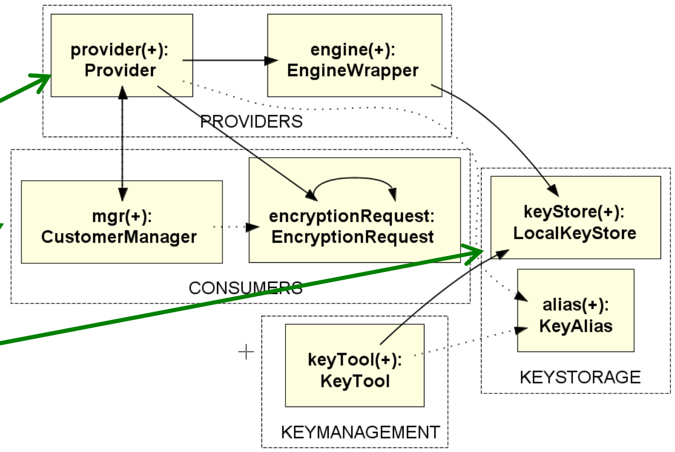


[Abi-Antoun & A, OOPSLA '09]

Architectural Design Intent

- Labeled groups
  - @Domain: Put in logical part of architecture

```
class Main {
   @Domain("PROVIDERS") Provider provider;
   @Domain("CONSUMERS") CustomerManager mgr;
   @Domain("KEYSTORAGE") LocalKeyStore keyStore;
}
```

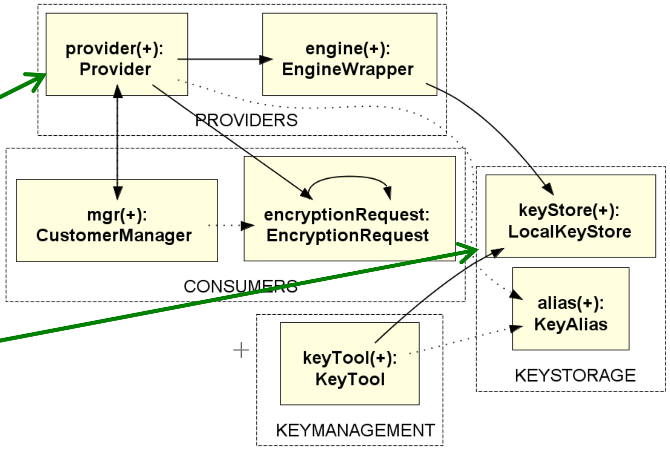


[Abi-Antoun & A, OOPSLA '09]

Architectural Design Intent

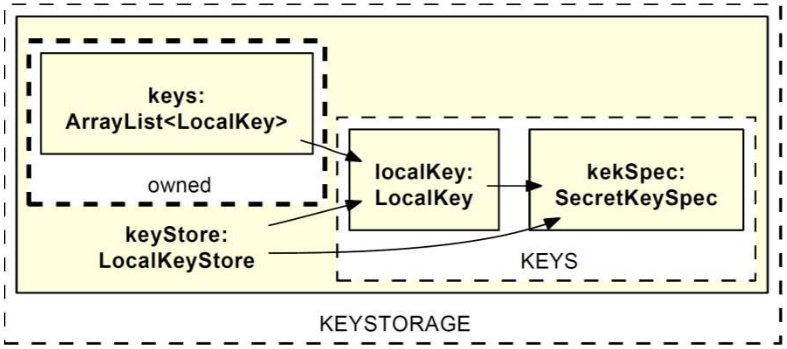
- Labeled groups
  - @Domain: Put in logical part of architecture

```
class Main {
    @Domain("PROVIDERS") Provider provider;
    @Domain("CONSUMERS") CustomerManager mgr;
    @Domain("KEYSTORAGE") LocalKeyStore keyStore;
}
```



- Data structure encapsulation
  - OWNED: Hide data objects within high-level abstractions

```
class LocalKeyStore {
                    List<LocalKey> keys;
}
```

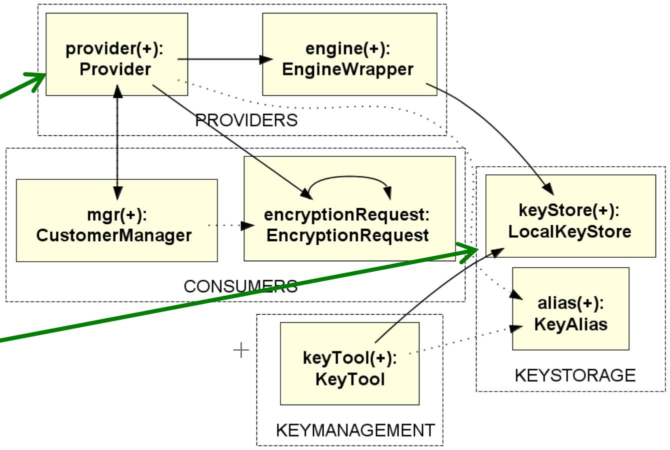


[Abi-Antoun & A, OOPSLA '09]

Architectural Design Intent

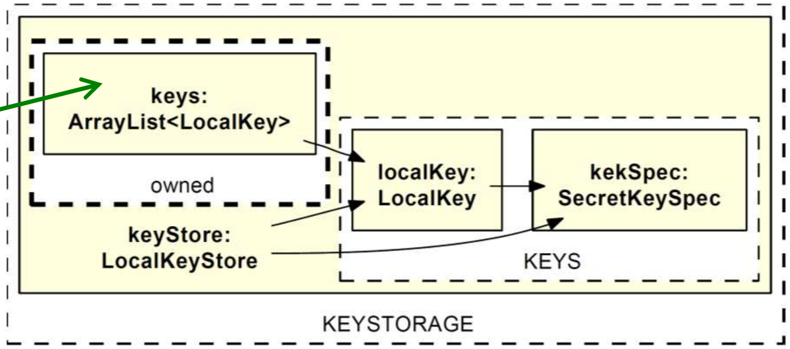
- Labeled groups
  - @Domain: Put in logical part of architecture

```
class Main {
    @Domain("PROVIDERS") Provider provider;
    @Domain("CONSUMERS") CustomerManager mgr;
    @Domain("KEYSTORAGE") LocalKeyStore keyStore;
}
```



- Data structure encapsulation
  - OWNED: Hide data objects within high-level abstractions

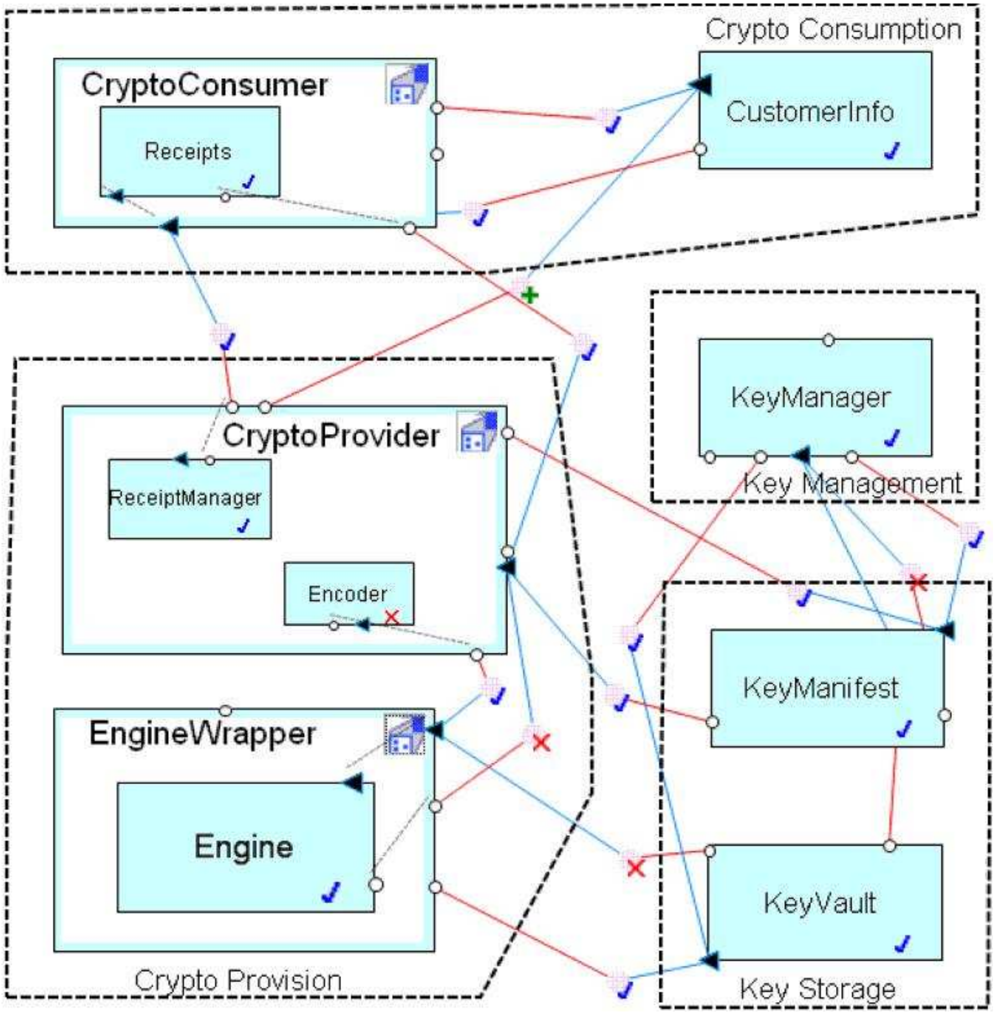
```
class LocalKeyStore {
    @Domain("OWNED<KEYS>") List<LocalKey> keys;
}
```



[Abi-Antoun & A, OOPSLA '09]

CryptoDB Case Study Results

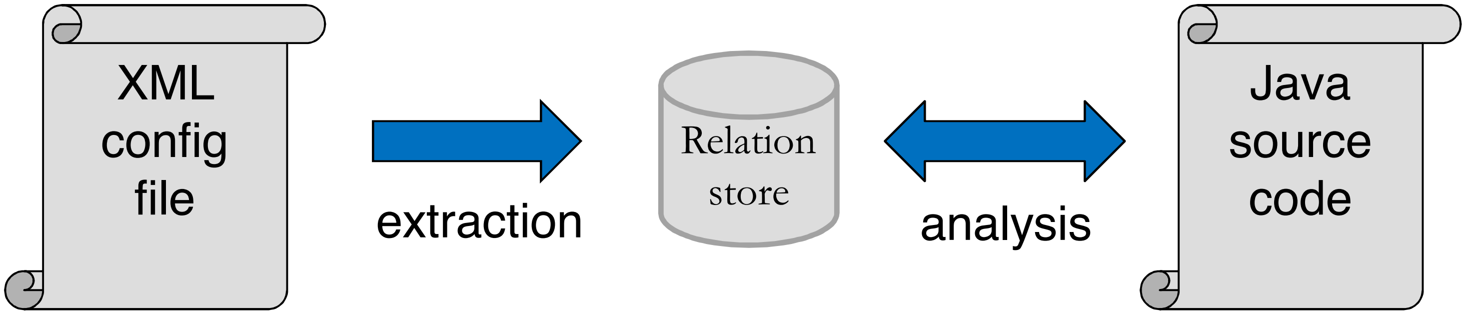
- Comparison non-trivial
  - Names in code differ from diagram
  - Multiple design components merged into one
- Diagrams mostly consistent
  - A few differences marked with X (missing) or + (added)
- Conformance analysis easily found injected defects



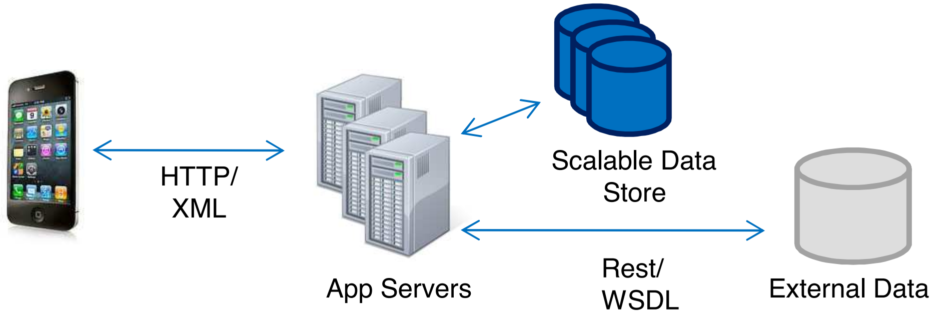
[Abi-Antoun & Barnes, ASE '10]

Configuration Files as Architecture

- Architecture already in industry frameworks
  - Framework configuration files describe structure, properties
  - Spring: web app framework
    - Describes structure, security properties of web site
  - Android framework
    - Describes event-based communication, UI flow, security properties
- Can we check these for consistency?
  - Specific tools for some frameworks—can we do it generally?
- FUSION tool at CMU/Cal Poly Pomona [C. Jaspan thesis, 2011]



Vision: Mobile App Architecture in Impl

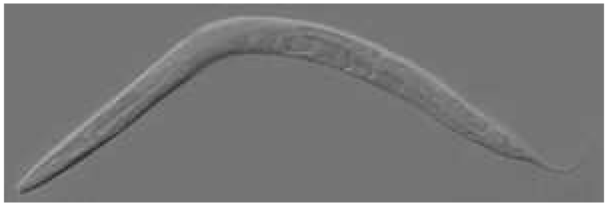


- Concept: ***Executable documentation***
  - E.g. declaring a protocol defines encoding used in components
  - Structure, redundancy, wire protocol, format, interfaces
  - Typechecking/analysis tools ensure consistency with code
- Enables analysis capabilities: attack graphs, threat models
- Challenge: making it open
  - Nothing "built-in" – implement security protocols as libraries
  - Thus libraries must also extend analysis capabilities
- End-to-end guarantee for what you implement "in the system"
  - Bridge to external systems via separate analysis tools

Why Ruby on Rails Works

- Flexible language syntax that supports embedded DSLs
  - But not much checking!

- Challenge: extensible language with extensible checking
- Approach: type-driven compilation and checking
  - Ability to pair a type with
    - Code generation
    - Semantic checks
  - Open source prototype: cl.oquence (OpenCL + C. Elegans)
    - Python syntax, C type system, OpenCL code generation for neuroscience



[Cyrus Omar, ongoing work at CMU]

- Applications
  - Prepared SQL statements – best defense against SQL injection
  - Communication protocols

Lower Level Design: Security by Default

- Integers
  - Default: infinite precision (relatively cheap to implement)
  - Ranged integers (enforced statically or dynamically)
  - Machine words if you really want them (low-level algorithms)

- Strings
  - Describe the format/contents (char classes, regular expressions)
  - Convenient common abstractions (names, numbers, etc.)
  - Arbitrary strings only if you really want them (low-level code)

- How to make it practical?
  - Convenient syntax and defaults
  - Leverage specifications to reduce engineering effort
    - E.g. input validation code can be driven by specifications

Unified data model

- Different data models
  - Client (JavaScript, Objective C)
  - HTTP (XML)
  - Server (Java, C++)
  - Database (SQL)
- Assurance challenges
  - Inconsistent semantics
  - Command injection
- Unified model
  - OO + database integrity constraints
    - Help with expressing security constraints
  - Can generate XML, SQL, encodings
- Challenge: interoperate with components we don't control

```
class Person {
        Name id;
        Collection<Course> coursesTaken
                inverse students;
}
class Course {
        Collection<Person> instructors;
        Collection<Student> students;
        Collection<Assignment> assgns;
}
class Assignment {
        Name name;
        nat possible;
        Course course inverse assgns;
}
```

Policy specifications

```
// in policy file

fun ScoreAccess(Grade g)
        principal in g.assignment.course.instructor

fun ScoreRead(Grade g)
        principal == g.student
```

- Policies leverage data model
    - Assignment, course, instructor are bidirectional relations
- Expressed using language abstractions
    - Built-in concept of principal
    - Permission, checks are extensible, reflective

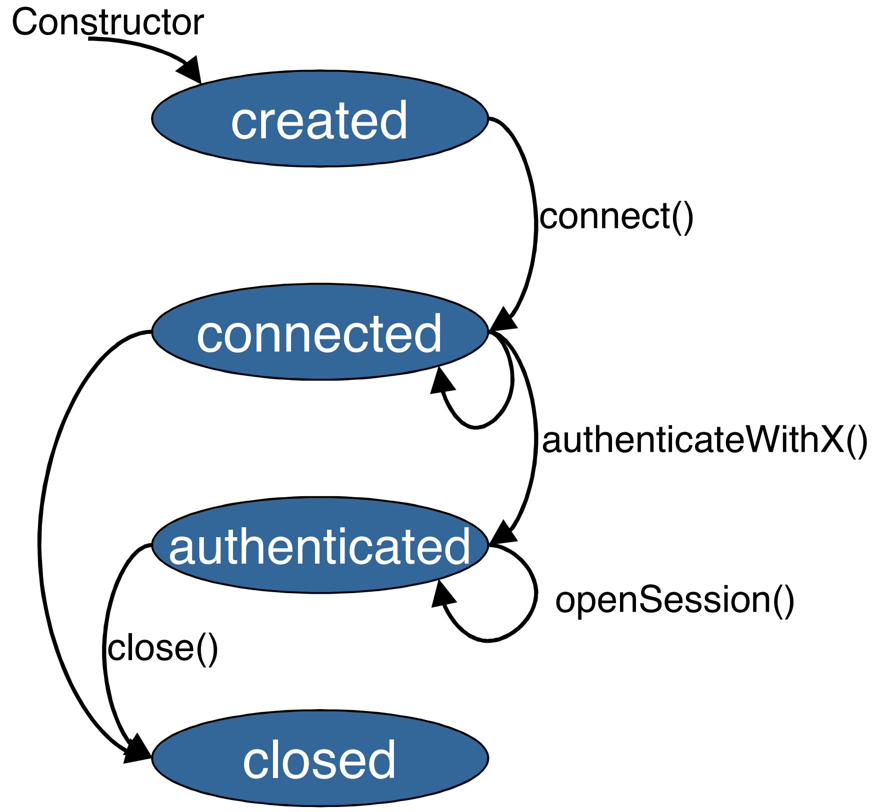
```
class Person { … }
class Course {
        Collection<Person> instructors;
        Collection<Student> students;
        Collection<Assignment> assgns;
}
class Assignment {
        Name name;
        Course course inverse assgns;
}
class Grade {
        Assignment assignment;
        Person student;

        @Read ScoreRead
        @Access ScoreAccess
        nat score;
}
```

Secure Protocols for Components, Communication

- Protocol constraints
  - *More common than type parameters!* *[ECOOP '11]*
  - Order of calls
  - Required argument state
- Frameworks
  - *Now underlie nearly all apps*
  - Verifying relationships among objects
- Concurrency
  - *Increasing in importance*
  - Time of check-time of use (TOCTOU) vulnerabilities

Ganymed SSH-2 Protocol



[With Kevin Bierhoff, Nels Beckman, Ciera Jaspan, Duri Kim]

Protocol Checking Experience [FSE ’05, ECOOP ’09]

Java Specifications

- Ganymed SSH-2 Protocol
- Collections and iterators
- I/O streams, Sockets
- XML, trees
- Timers, Tasks
- JDBC (database connectivity)
- Regular expressions
- Exceptions

Verification Studies

- **Breadth:** JabRef, PMD, JSpider…
  - 100+ kLOC open source code
  - Multiple APIs assured

- **Depth:** Apache Beehive
  - Open Source resource access library
  - Has its own protocol
    - Common scenario: one API builds on another
  - Verified implementation uses JDBC correctly

Among the first field studies of semantically deep resource analysis for objects at this scale

[With Kevin Bierhoff, Nels Beckman]

Protocols and Productivity

Keyword(s)	# of results
Java IllegalStateException	880
Java NullPointerException	3,137
Java UnsupportedOperationException	610
Java	239,525

- Protocols cause problems
  - Many hits on stackoverflow
- But bugs not often released
- Observational study: 8 professional programmers
  - Greenfield programming/debugging tasks with protocols
    - Error messages not helpful:
      "java.sql.SQLException: invalid cursor state: cannot FETCH NEXT, PRIOR, CURRENT, or RELATIVE, cursor position is unknown"
    - 60 pages of documentation
- Results: 88% time spent answering questions about protocols
- Barriers
  - State encoded at low level
  - Unhelpful error messages
  - Documentation & tools not context-specific
  - Documentation does not clearly separate state from functionality

[With Ciera Jaspan]

Protocols and Productivity

Keyword(s)	# of results
Java IllegalStateException	880
Java NullPointerException	3,137
Java UnsupportedOperationException	610
Java	239,525

- Protocols cause problems
  - Many hits on stackoverflow
- But bugs not often released
- Observational study: 8 professional programmers

Next step: can protocol checking tools enhance productivity? By what mechanisms?

- Results: 88% time spent answering questions about protocols
- Barriers
  - State encoded at low level
  - Unhelpful error messages
  - Documentation & tools not context-specific
  - Documentation does not clearly separate state from functionality

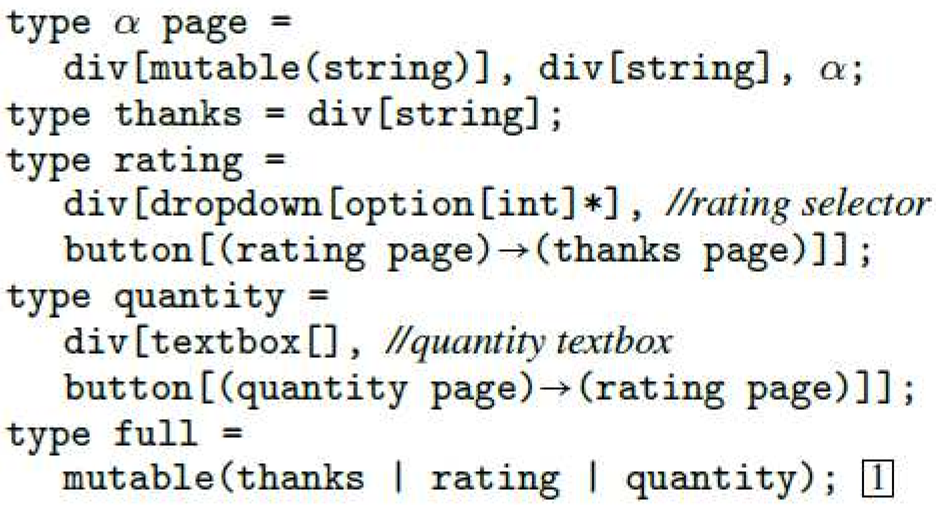
[With Ciera Jaspan]

User Interface Protocols

Title: Types and Programming Languages

Author: Benjamin C. Pierce

Price: $57.60



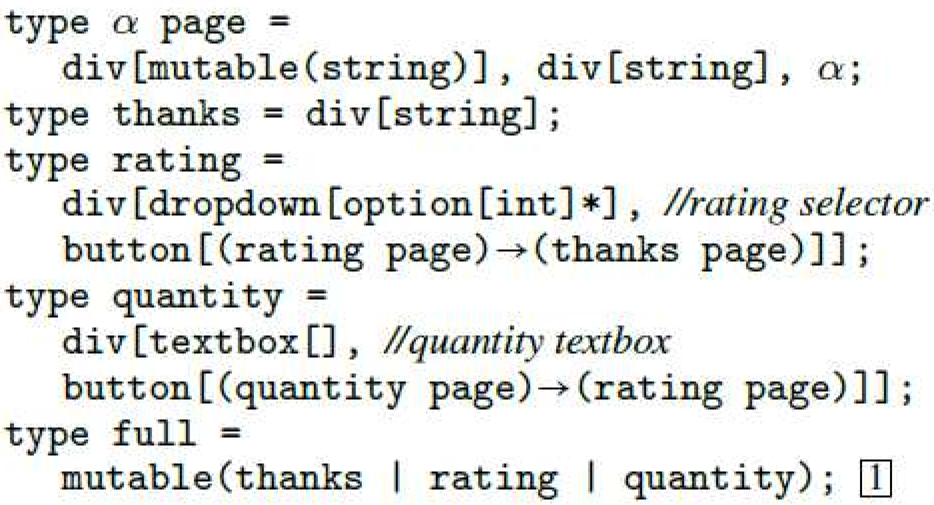
```
type α page =
   div[mutable(string)], div[string], α;
type thanks = div[string];
type rating =
   div[dropdown[option[int]*], //rating selector
   button[(rating page)→(thanks page)]];
type quantity =
   div[textbox[], //quantity textbox
   button[(quantity page)→(rating page)]];
type full =
   mutable(thanks | rating | quantity); [1]
```

- Protocols appear in UIs as well as libraries
- Checking approach [APLWACA `10]
  - Declaratively specify states of web page
  - Check that code is consistent with web page changes
- Software engineering benefits enhance security, too
  - Declarative UI enables link to input data validation

[With Joshua Sunshine]

User Interface Protocols

Title: Types and Programming Languages
Author: Benjamin C. Pierce
Price: $57.60



```
type α page =
   div[mutable(string)], div[string], α;
type thanks = div[string];
type rating =
   div[dropdown[option[int]*], //rating selector
   button[(rating page)→(thanks page)]];
type quantity =
   div[textbox[], //quantity textbox
   button[(quantity page)→(rating page)]];
type full =
   mutable(thanks | rating | quantity); [1]
```

Other applications of protocols:
Mitigating cross-site request forgery (CSRF) attacks

- Software engineering benefits enhance security, too
  - Declarative UI enables link to input data validation

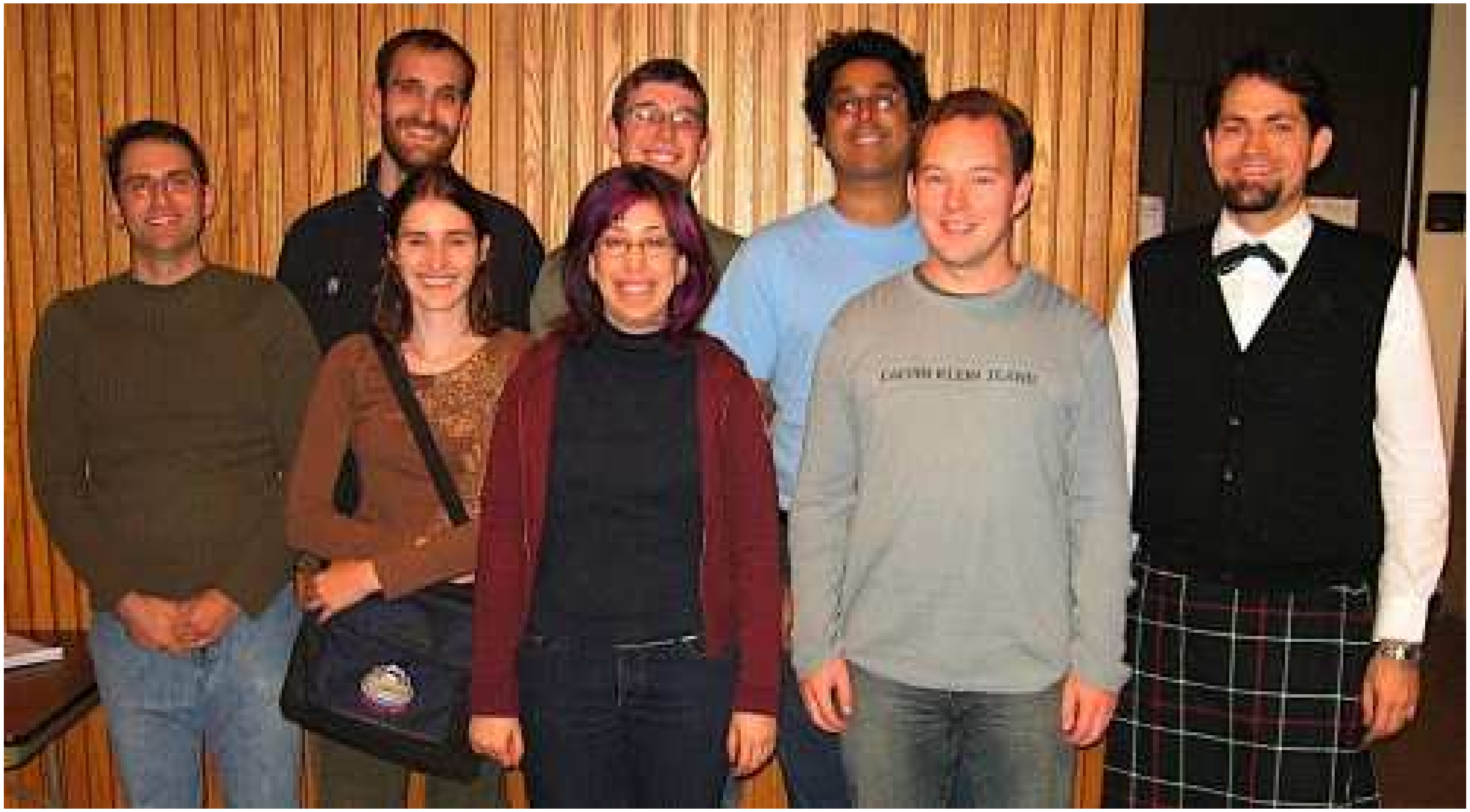
[With Joshua Sunshine]

Designed-In Security for Mobile Apps

- Techniques for designing security into application code
  - Architectural models tie components together
  - Design intent describes security policy, means of assurance
  - Secure-by-default language constructs, libraries

- Benefits for both security and software engineering
  - Connect existing security practices to source code
  - Assurance at systems level and code level
  - Improve productivity by raising level of abstraction

The Plaid Group



(from a couple of years ago)