Finding Architectural Flaws using Constraints

By Radu Vanciu and Marwan Abi-Antoun
Department of Computer Science
Wayne State University
Detroit, Michigan, USA

This work was supported in part by the Army Research Office under Award No. W911NF-09-1-0273

High security risks in mobile apps are due to architectural flaws

- Existing code has high business value
  - > **1M** Android apps** in Goolge Play store
  - >1B Android devices (activated by Sept. 2013***)
- Coding defects: local, found by analyzing one class at a time

  Example: hard-coded password
- Architectural flaws: non-local, found by reasoning about usage context
- Example: Sensitive information disclosure

OWASP 2013 [mobile applications*]
M1: Insecure Data Storage
M2: Weak Server Side Controls
M3: Insufficient Transport Layer Protection
M4: Client Side Injection
M5: Poor Authorization and Authentication
M6: Improper Session Handling
M7: Security Decisions Via Untrusted Inputs
M8: Side Channel Data Leakage
M9: Broken Cryptography
M10: Sensitive Information Disclosure

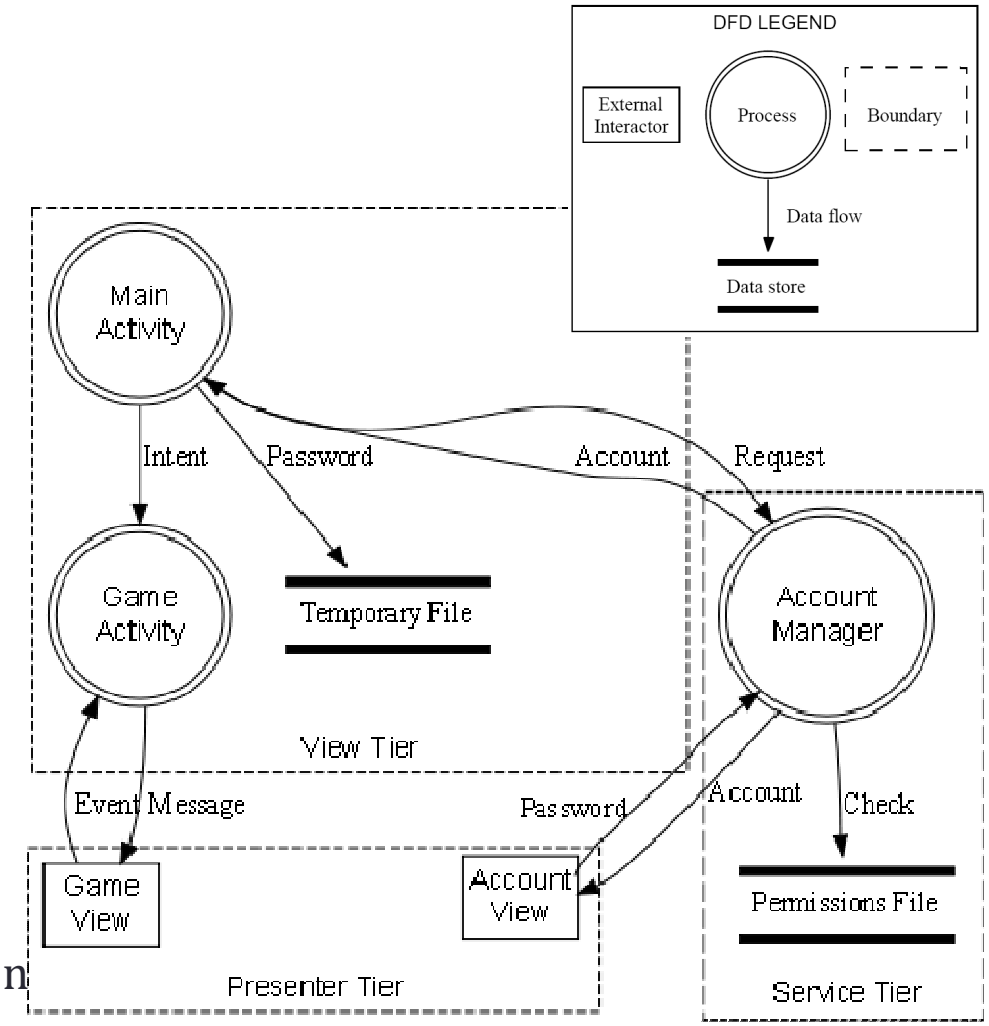
*https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

**http://www.appbrain.com/stats/number-of-android-apps

***http://www.androidcentral.com/android-passes-1-billion-activations

Architectural Risk Analysis (ARA) helps to find architectural flaws [Howard and Lipner, Microsoft Press'06]

- Architects use forest-level view of system (not reading code)
  - Runtime architecture – not code architecture
  - Assign security properties to components and write constraints
- Limitations of ARA approaches
  - Limited support for reverse engineering
  - Runtime architecture is missing or inconsistent
  - Lack of traceability from runtime architecture to code
  - Analyses focus only on presence/absence of communication



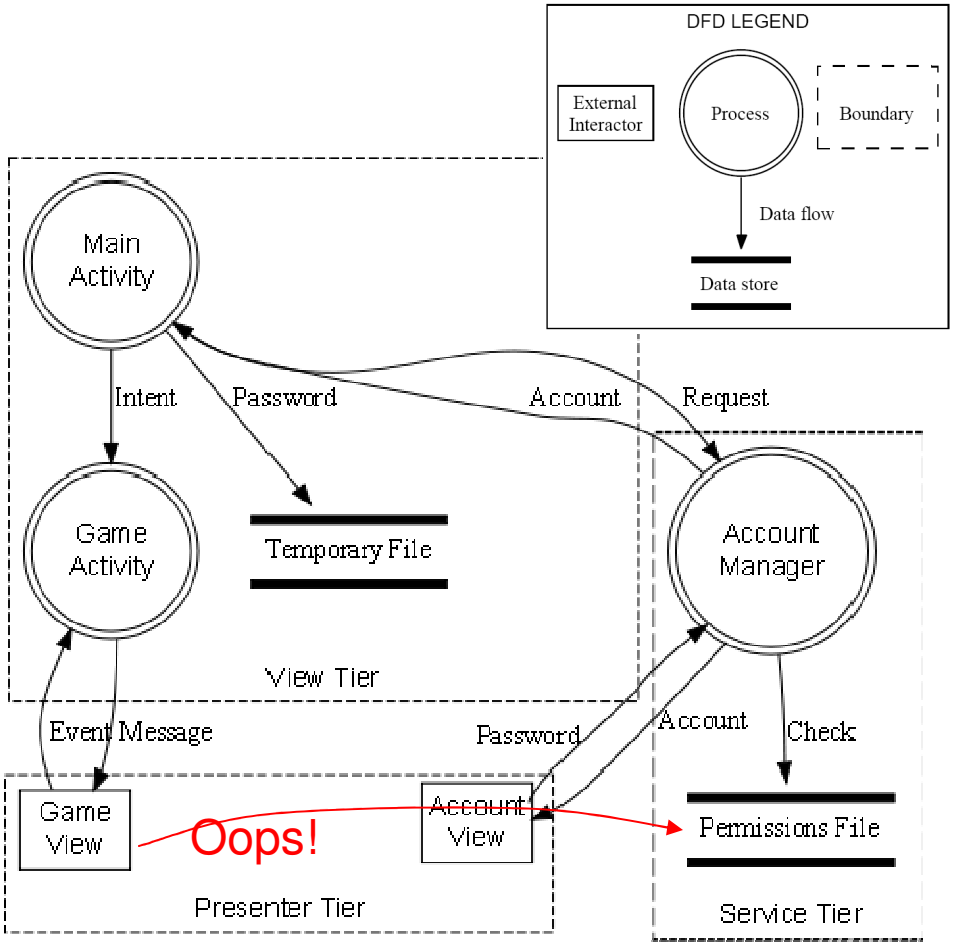
Static analysis to extract a sound approximation of runtime architecture

- Extract from code runtime architecture useful for security
  [Abi-Antoun and Aldrich, OOPLSA'09, Abi-Antoun and Barnes, ASE'10]
  - Annotations convey design intent
  - Sound approximation of the runtime architecture
  - Supports hierarchical decomposition
    - Architectural relevant objects near the top of hierarchy
    - Implementation details further down

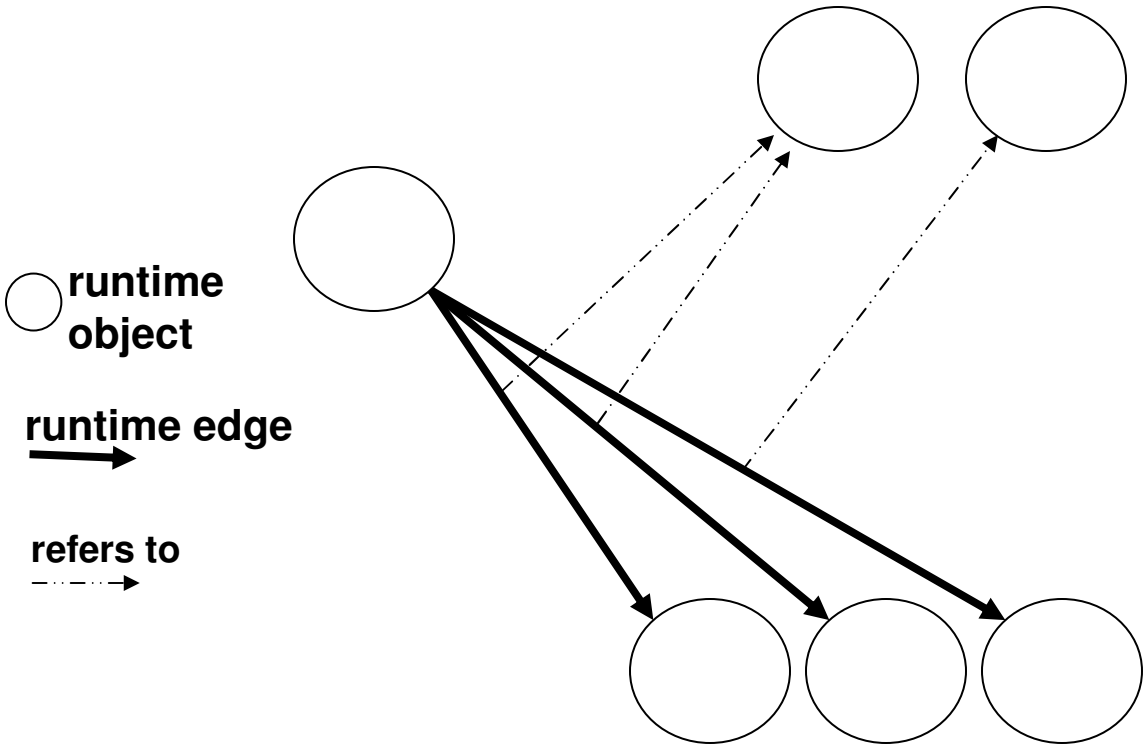
- Reason about dataflow communication
  - Extract dataflow edges instead of approximating them from points-to edges [Vanciu and Abi-Antoun, WCRE'12]
  - Dataflow edge refers to objects
  - Value flow analysis resolves objects that are borrowed or passed linearly [Vanciu and Abi-Antoun, FOOL'13]

Worst-case security analysis requires soundness

- **Represent all** objects and relations that may exist at runtime
- Absence of a connector means absence of communication at runtime
- Find unexpected edges that may occur in exceptional or error handling cases
- Ensure that same runtime entity is not mapped to distinct components

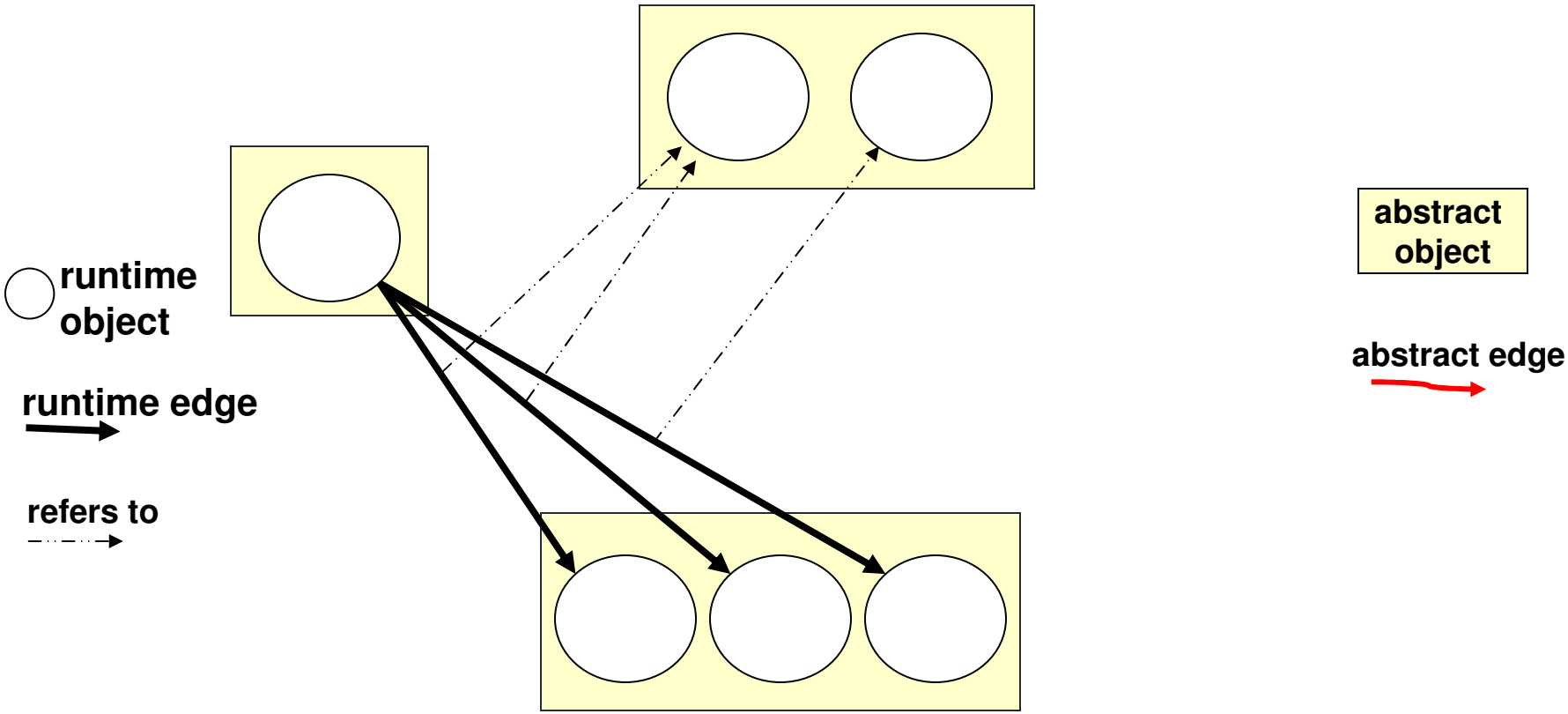


At runtime, object oriented program appears as Runtime Object Graph



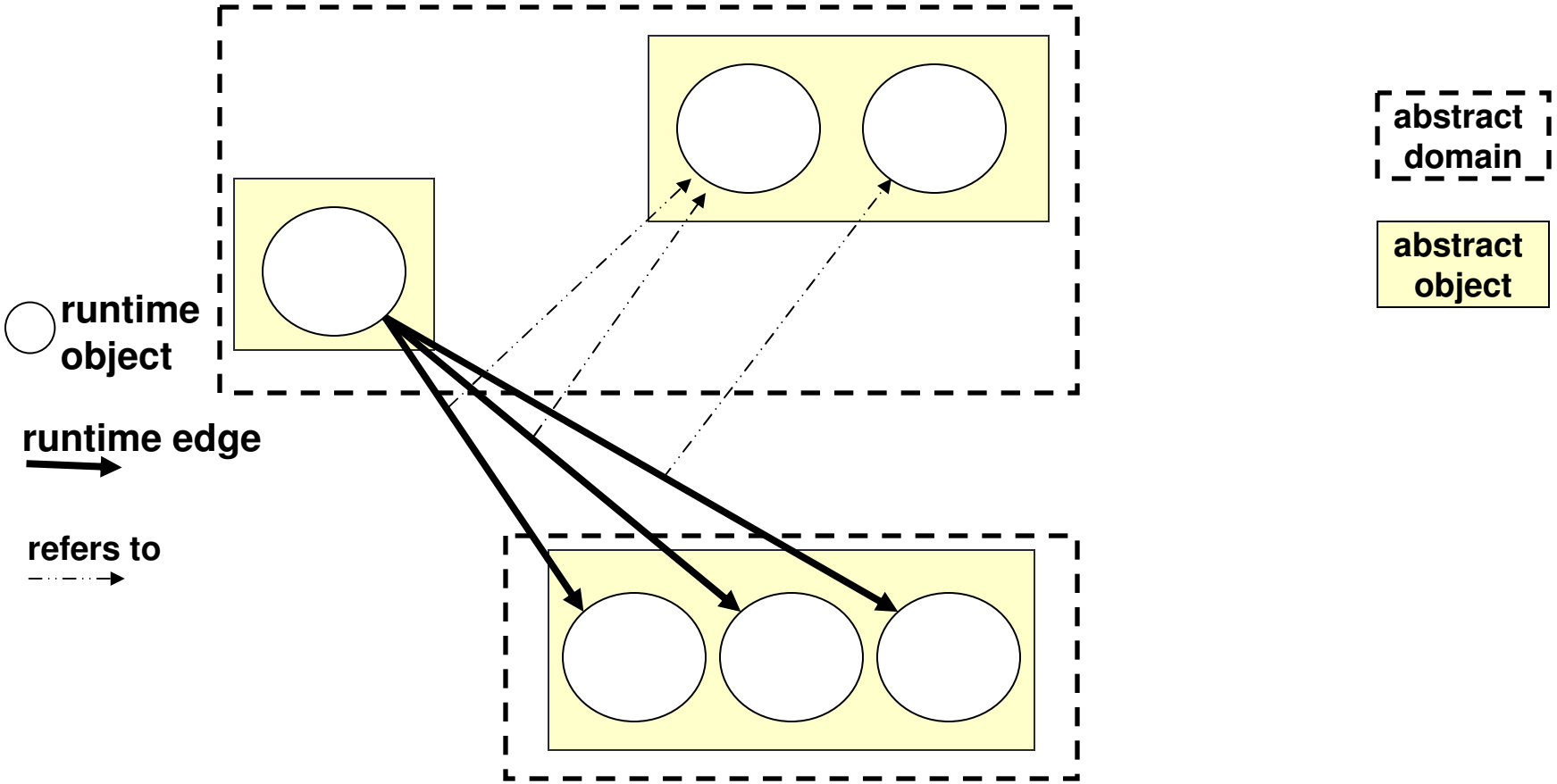
Abstract multiple runtime objects into an abstract object

- Each runtime object has **exactly one** representative in extracted object graph



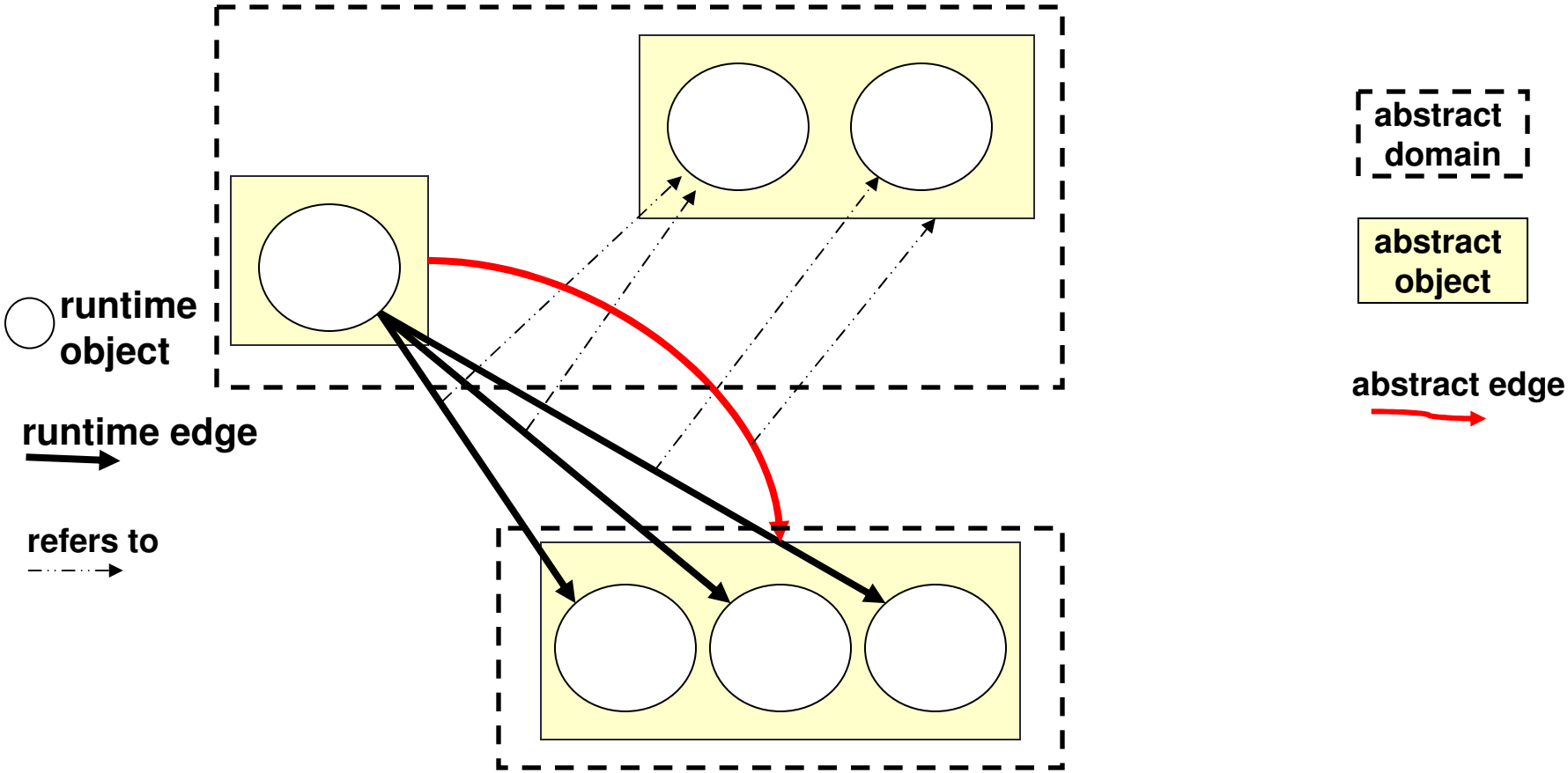
Abstract domain is a group of abstract objects

- Place each abstract object in exactly one conceptual group (abstract domain)



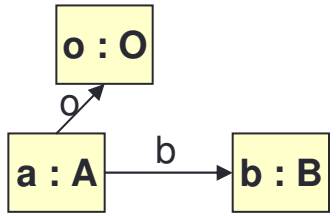
Abstract edge is between abstract objects

- Runtime edge between two objects maps to the abstract edge between the representatives of two objects



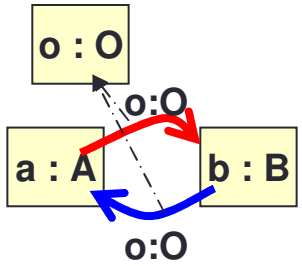
Edges between abstract objects

- Points-to [label is field name]



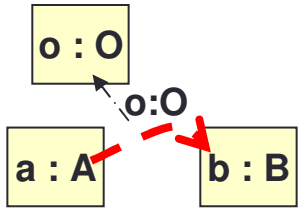

```
class A{
  B b; O o;
```

- Dataflow [label refers to object]




```
O m1(){
 return b.add(o);
}
```

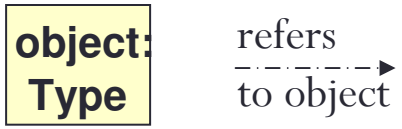
- Creation [label refers to object]




```
void m2(){
 new B(o);
}
```

- Control flow [label is method name]


```
void m3(){
 b.start()
}
}
```

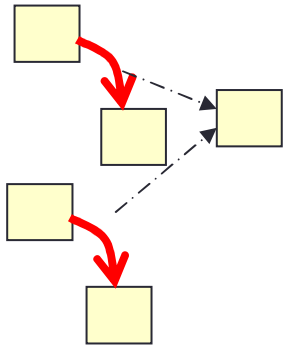


Objects are organized in hierarchically

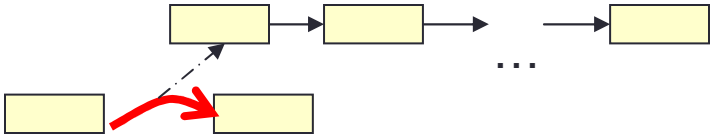
- Abstract object has domains
- Each domain has objects
- Hierarchy of objects extracted by analyzing code with annotations
- Domains provides precision
  - Distinguish between objects of same type in different domains
  - At runtime object does not change domain

```
+-m:Main
  +- LOGIC
  | +- a:UPMApplication
  |   +- MSG
  |     +- i:Intent
  |        +- OWNED
  |           +- map:HashMap
  |     +- cert:File
  +- DATA
  | +- pd:PasswordDatabase
  |   +- OWNED
  |     +- backup:File
  | +- db:File
```

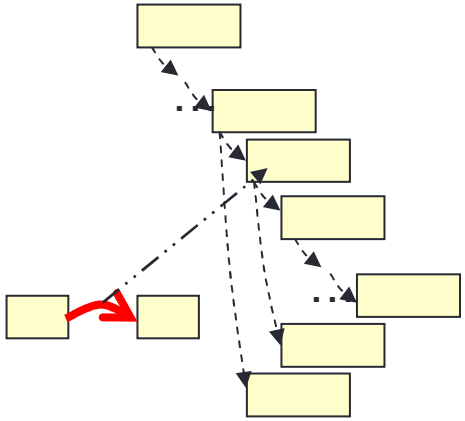
Scoria: reasoning about security using constraints on the hierarchical object graph



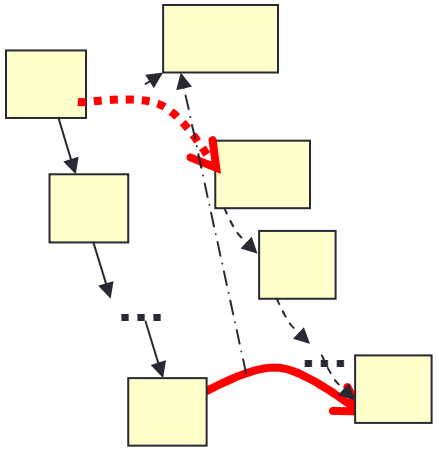
Object provenance



Object reachability



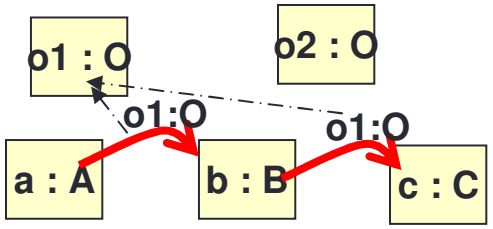
Object hierarchy



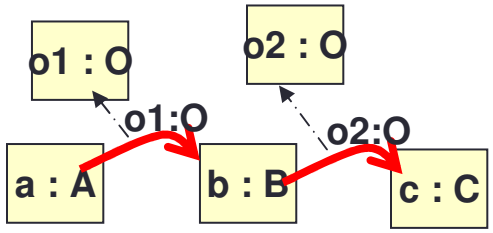
Indirect communication

Architects distinguish between copying and sharing of object → Object identity

- Every abstract object is uniquely identified
- Enable comparison of references



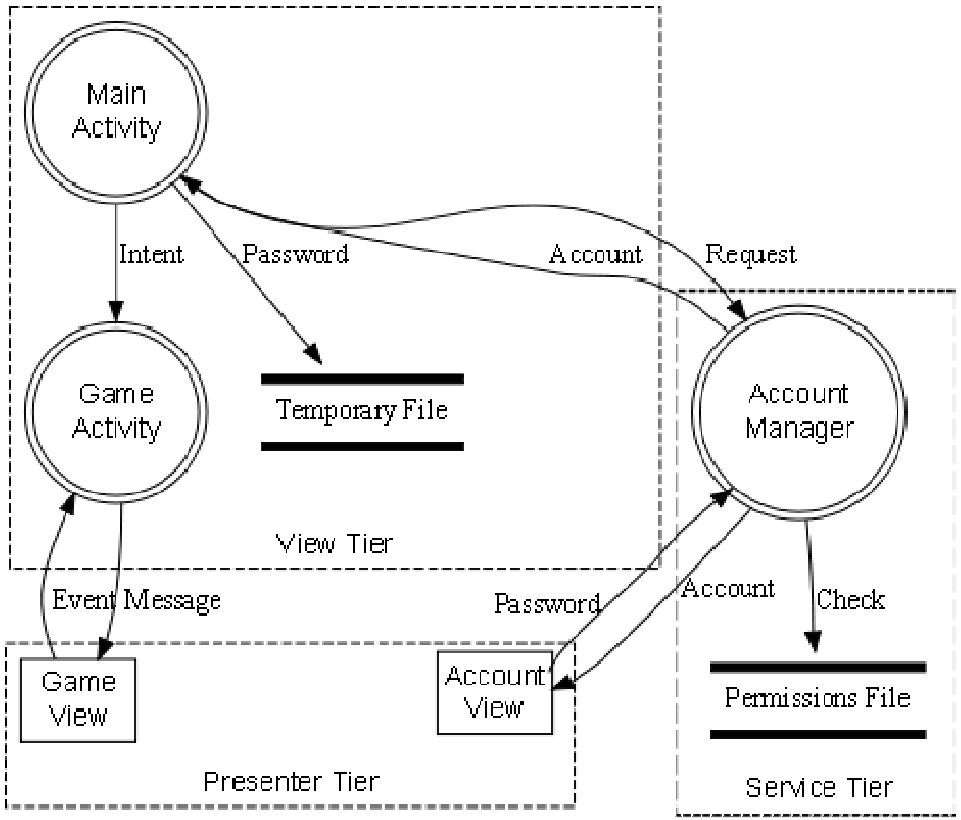
Edges refer to same abstract object



Edges refer to distinct abstract objects of same type

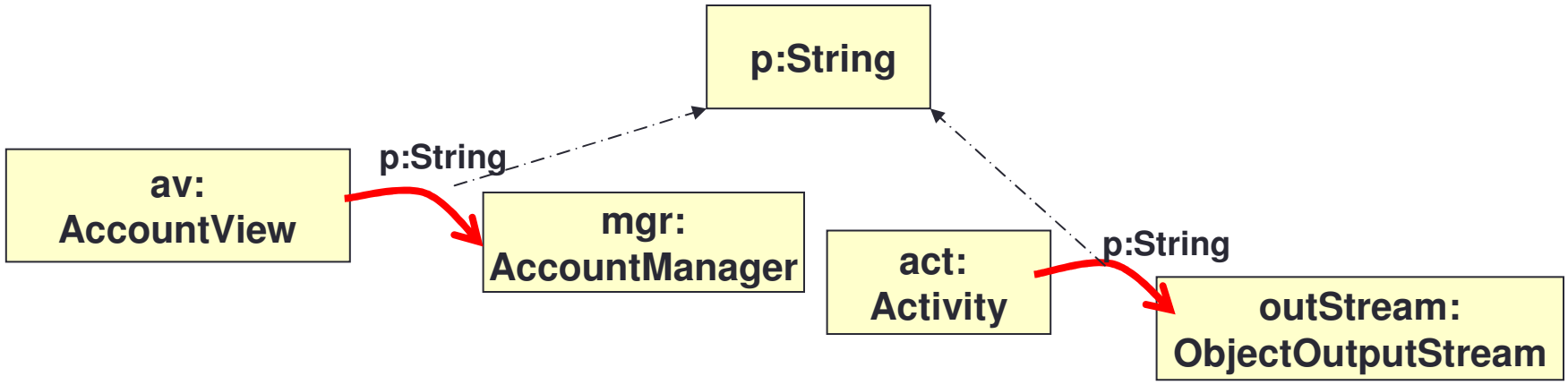
Object provenance → multiple dataflow edges refer to same object

- Password flows from Account View to Account Manager
- Same password flows to Temporary File



Example of object provenance

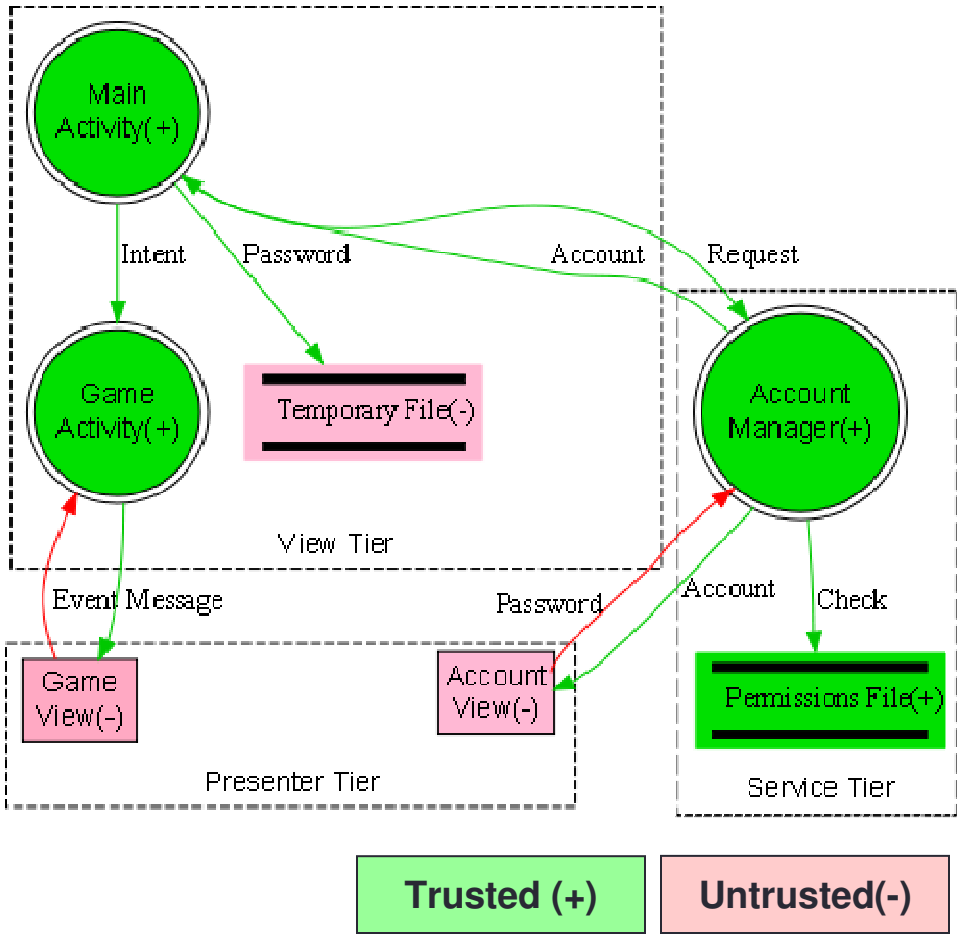
- Same object that flows from av:AccountView to mgr:AccountManager is saved by act:Activity into outputStream:ObjectOutputStream



Security information that is not directly extracted from the code → security properties

- Security property values for each component and connector
  - **TrustLevel**
    - **Trusted(+)**
    - **Untrusted(-)**
    - **Unknown**
  - **IsConfidential**
    - **True**
    - **False**
    - **Unknown**
  - **IsEncrypted**

- Using security properties
  - **Tampering**: **Untrusted(-) → Trusted(+)**
  - **Information Disclosure**: **Trusted(+) → Untrusted(-)**



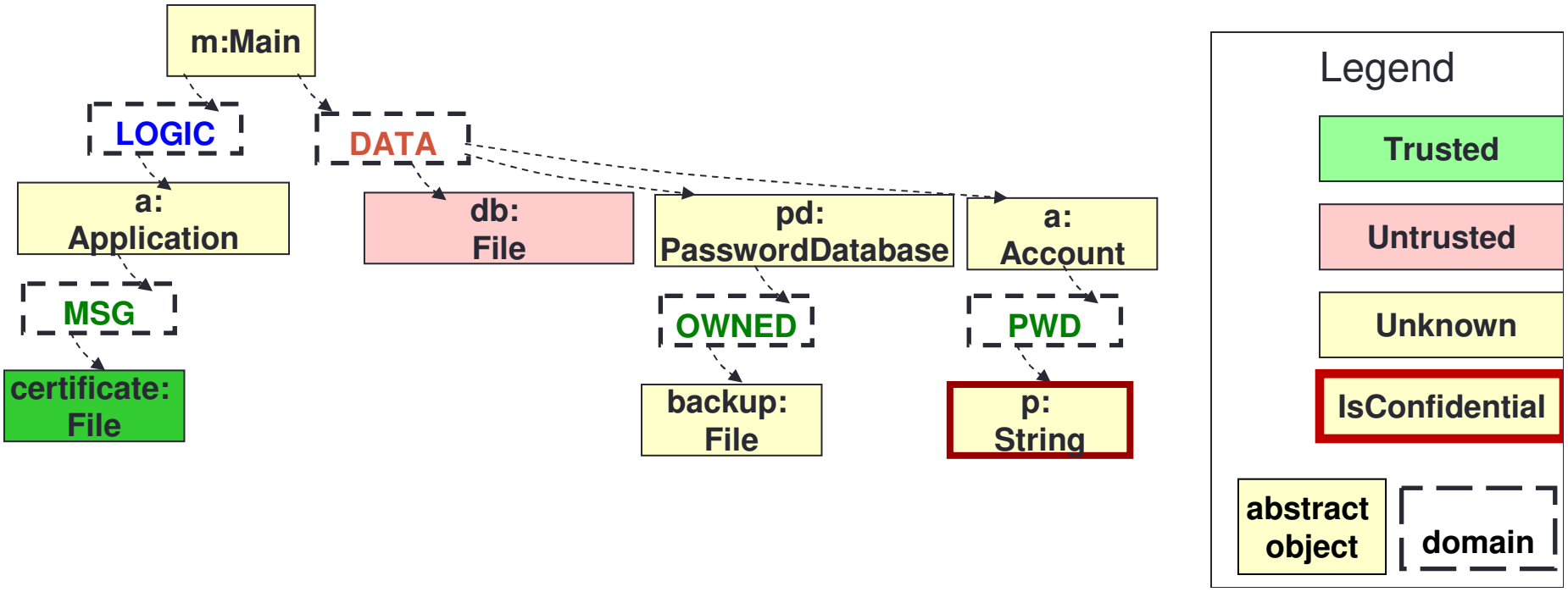
Assign security properties →queries

- Property queries: *setObjectsProperty(props, condition), setEdgesProperty(props, condition)*
  - Assign property values to objects or edges that satisfy condition

- Condition based on:
  - Type + object hierarchy: *IsInDomain, IsChildOf*
  - Type + object reachability: *IsReachableFrom*
  - Type + traceability: *IsCreatedByMethod*
  - Type: *InstanceOf*

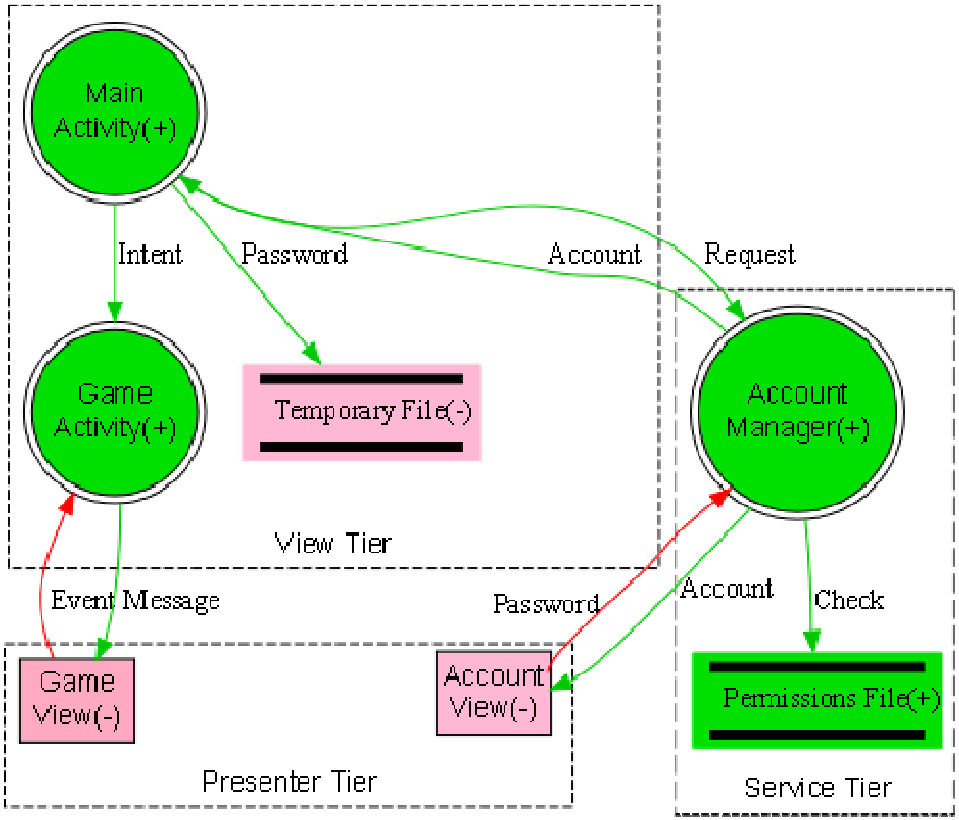
Assign security properties →queries

- *setObjectsProperty*(*TrustLevel.**Untrusted*** , *IsInDomain*(DATA, File))
- *setObjectsProperty*(*TrustLevel.**Trusted***, *IsInDomain*(MSG, File))
- *setObjectsProperty*(*IsConfidential.**true,*** *IsChildOf*(Account, String))



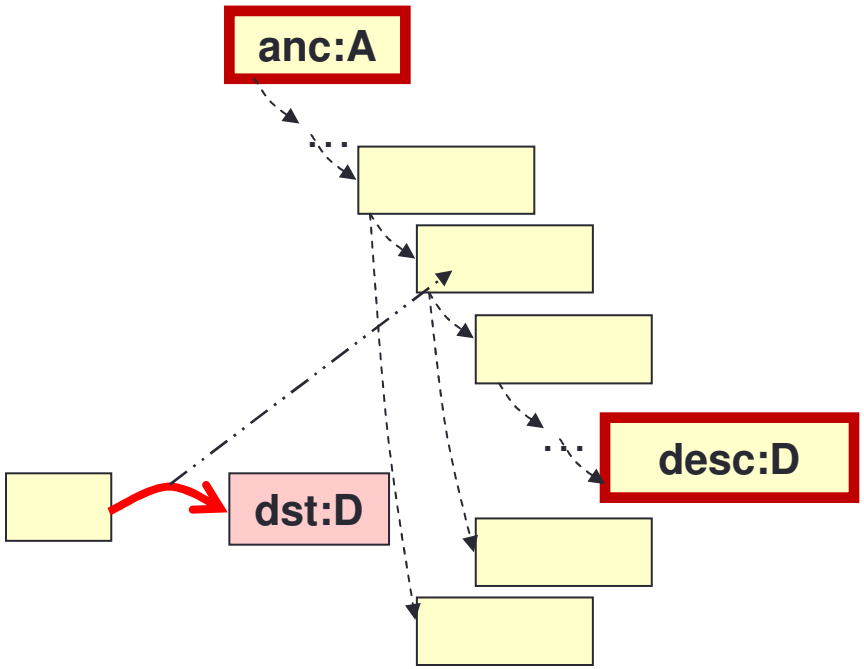
Some objects that carry confidential data may be part of some other object

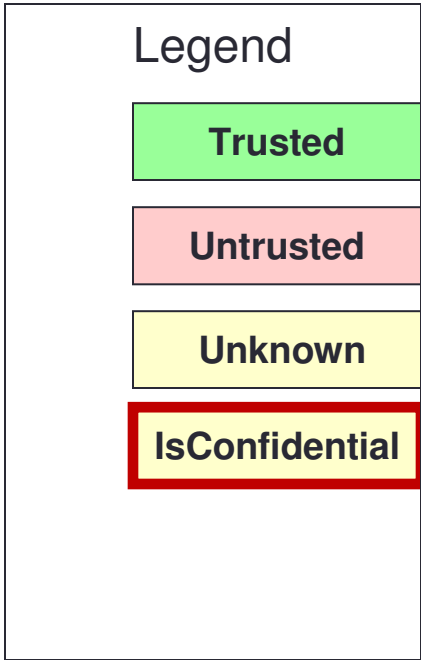
- Account has a password
- Only password is confidential



Identify ancestors and descendants of objects that edges refer to → object hierarchy

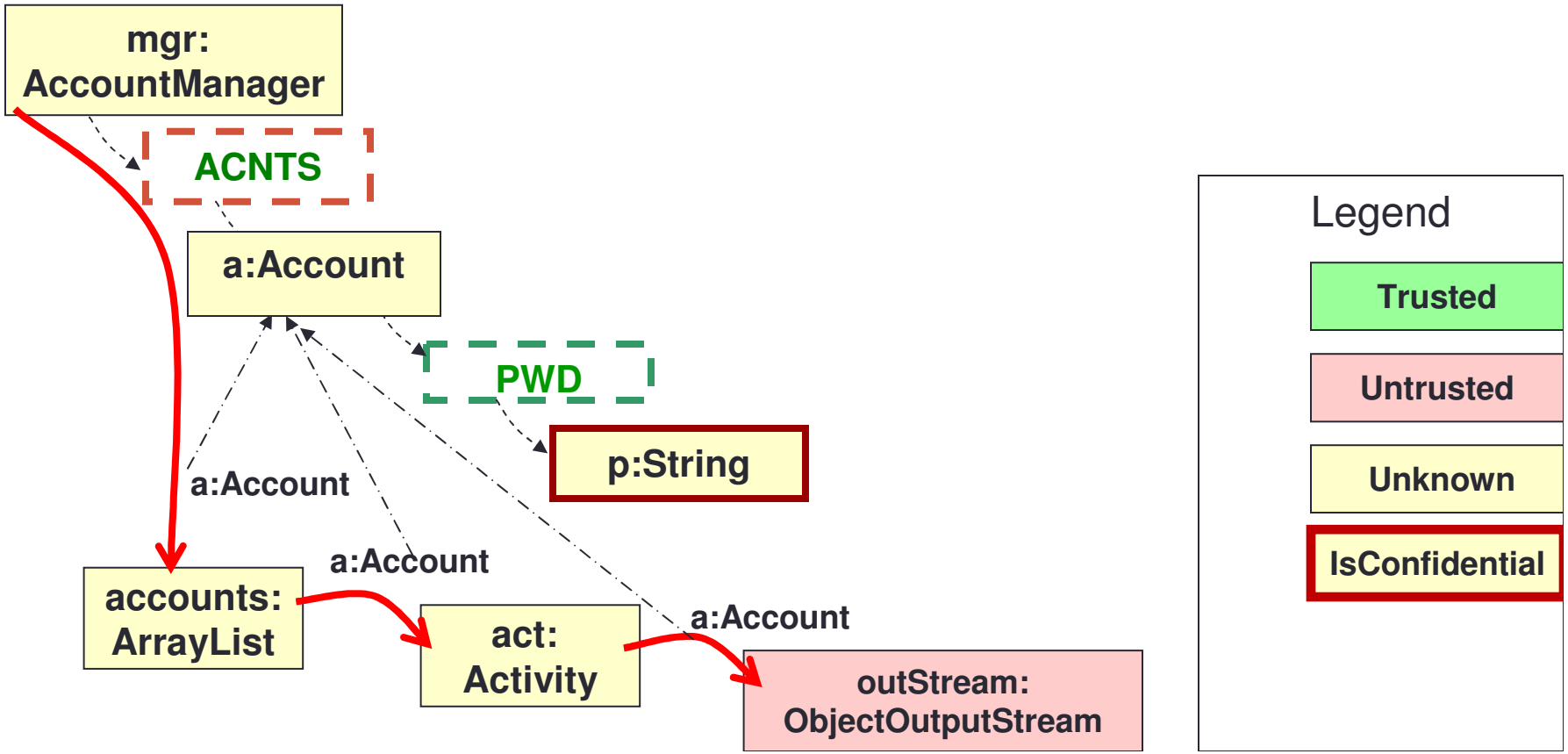
- Only some objects are confidential, but architects also consider:
  - Descendant of object referred from dataflow edge is confidential
  - Ancestor of object referred from dataflow edge is confidential





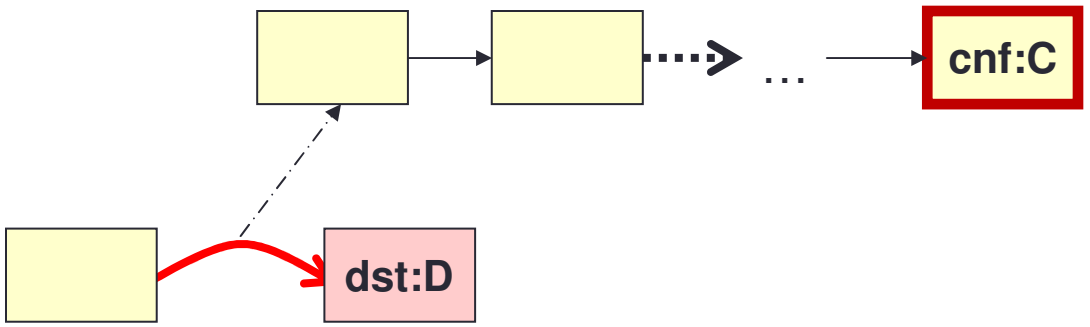
Identify ancestors and descendants of objects that edges refer to → object hierarchy

- Object p:String is confidential
- Dataflow edge refer to its parent

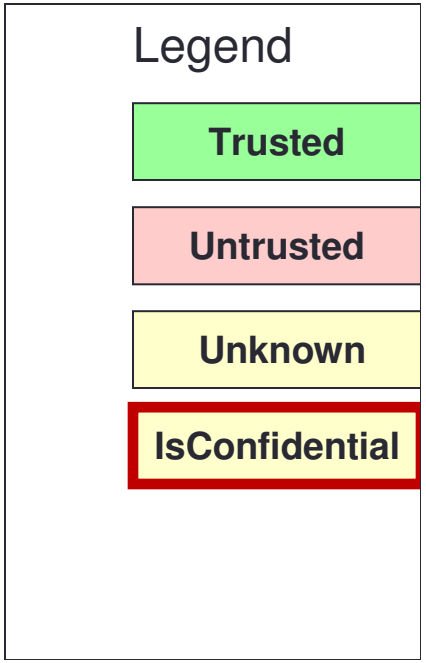


Confidential objects reached indirectly from dataflow edge → object reachability

- Reachability uses various types of edges
  - Points-to
  - Dataflow
  - Creation
  - Control-flow

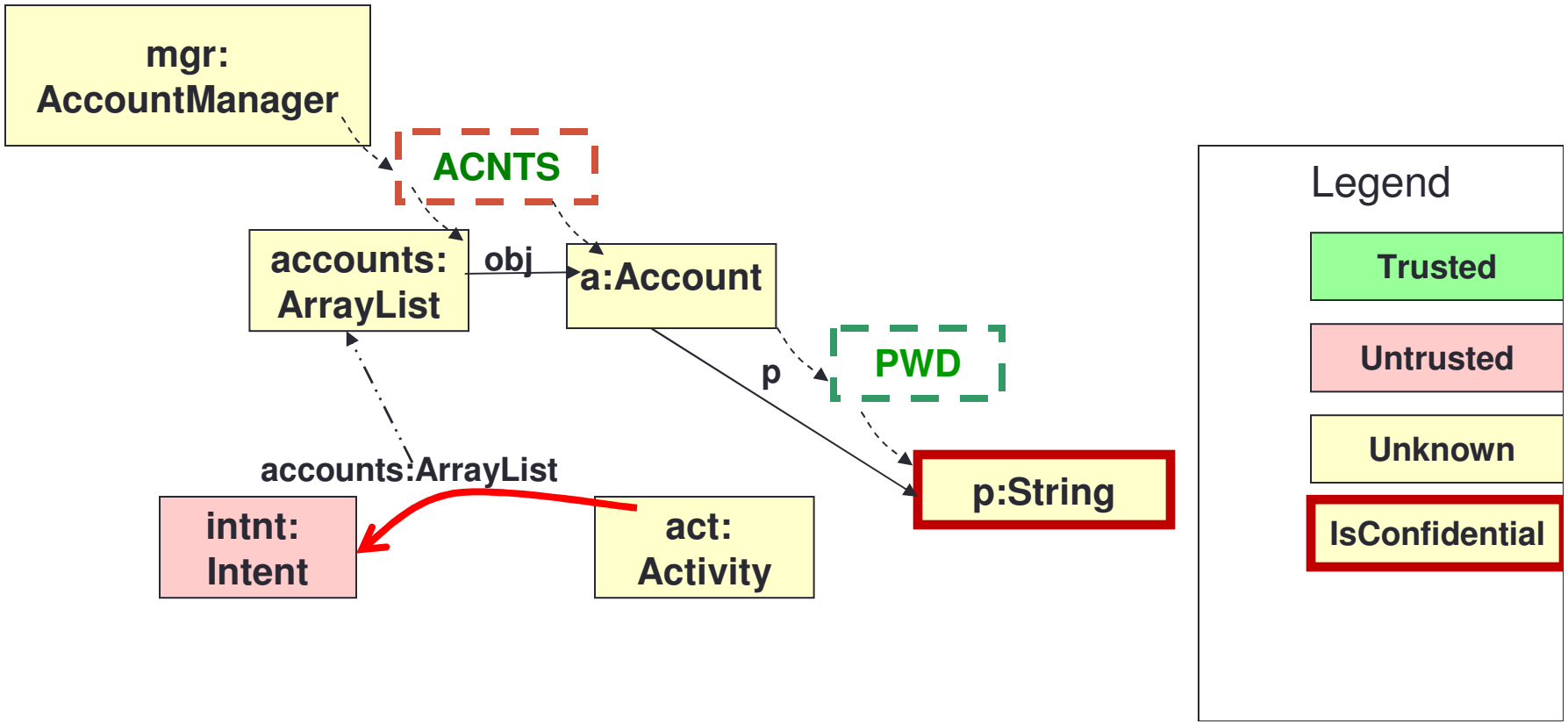


- Untrusted destination may use reachability to access confidential data



Confidential objects reached indirectly from dataflow edge → object reachability

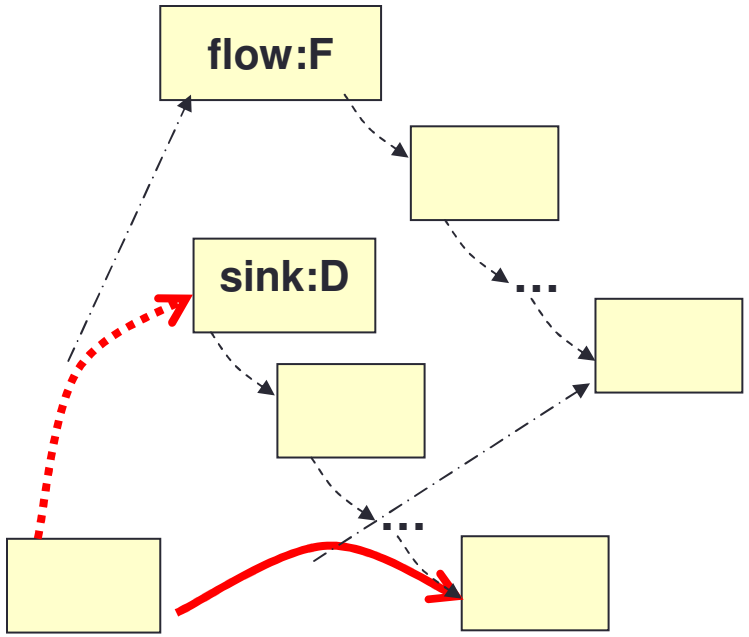
- Password is reachable from accounts:ArrayList,
- accounts:ArrayList flows into untrusted object



Identify indirect communication through object hierarchy and object reachability

getFlowIntoSink(flow, sink)

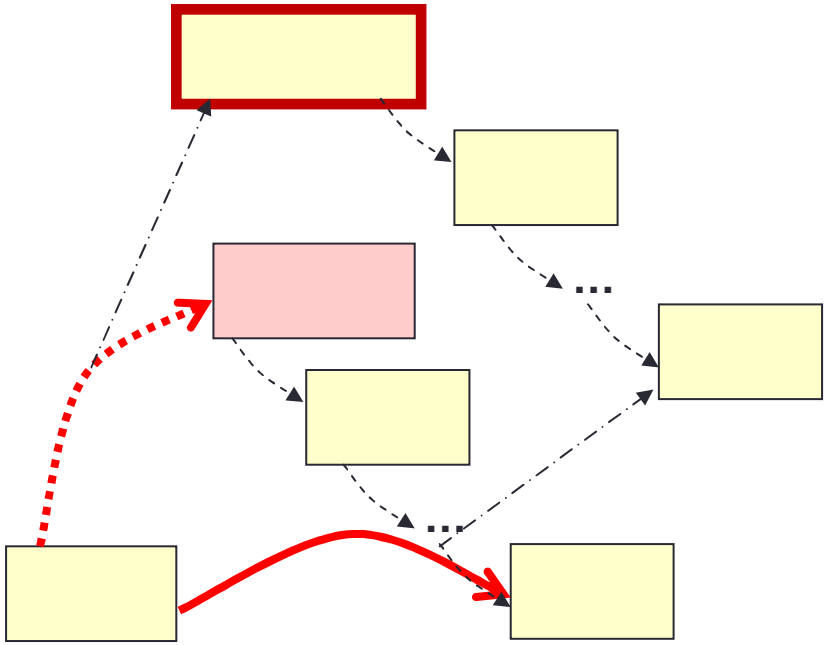
- Query returns dataflow or creation edges
- Destination is descendant of *sink*, or object reachable from *sink*
- Edges refers to descendant of *flow* or object reachable from *flow*

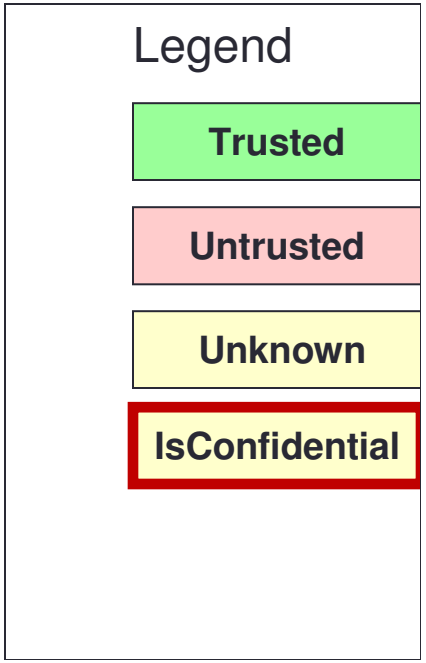


Automating security reasoning → machine checkable constraints on query results

*getFlowIntoSink(IsConfidential.**true,** TrustLevel.**Untrusted**)*

- Query in terms of security properties only — not system specific
- Edges that refer to confidential object with an untrusted destination
- Constraint: returned set is empty





Evaluation

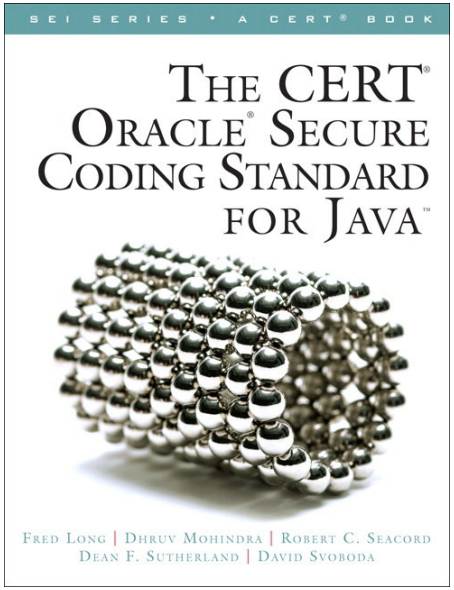
Hypothesis: *Machine-checkable constraints in terms of object provenance and indirect communication that analyze the hierarchy and the reachability of abstract objects on abstract graph can find architectural flaws such as information disclosure and tampering*

- Find injected architectural flaws
- Find architectural flaws in one open source

Evaluation on CERT Oracle Secure Coding Standard for Java

[Long et al. Addison-Wesley'11]

- Goal: write CERT rules as machine-checkable constraints (reusable across systems)
  - Find injected architectural flaws and avoid false positives
  - Select CERT rules for which automated detection was previously unavailable



CERT-ID	Short description
MSC00-J	Use SSLSocket rather than Socket for secure data exchange
FIO05-J	Do not expose buffers created using the wrap() or duplicate() methods to untrusted code
SER03-J	Do not serialize unencrypted, sensitive data
FIO13-J	Do not log sensitive information outside a trust boundary
IDS07-J	Do not pass untrusted, unsanitized data to the Runtime.exec() method

Results

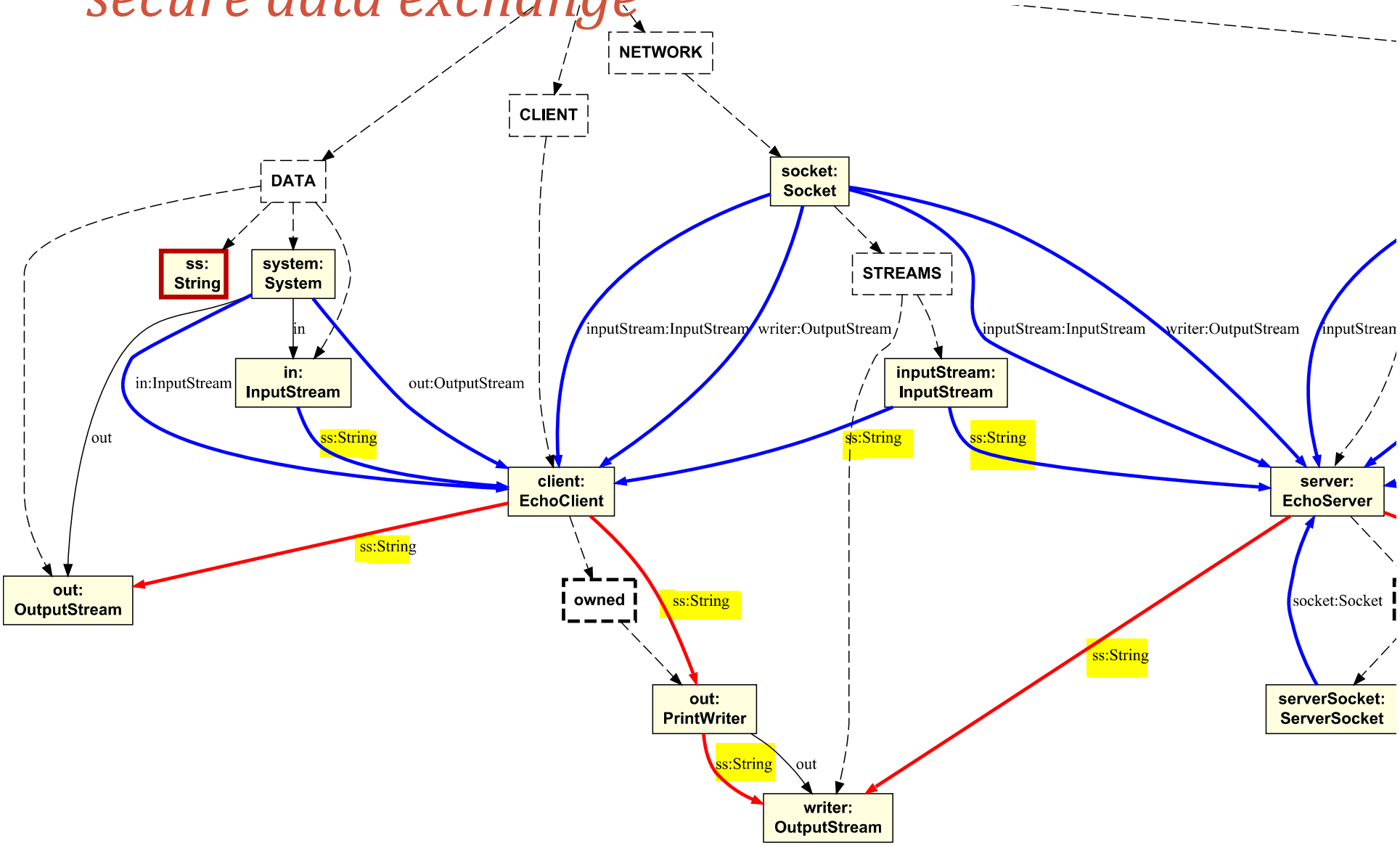
CERT-ID	Vulnerability	Object provenance	Object transitivity	Object hierarchy	Object reachability	Object traceability	Suspicious edges	Avoided false positives	Object security properties	Edge security properties
MSC00-J	Information disclosure	✓	✓	✓	✓		2	2	0	0
FIO05-J	Information disclosure			✓	✓	✓	1	1	2	0
SER03-J	Information Disclosure			✓			1	1	2	2
FIO13-J	Information Disclosure				✓		1	1	2	0
IDS07-J	Tampering	✓	✓			✓	1	1	0	0

MSC00-J. Use SSLSocket rather than Socket for secure data exchange

- Uses the Socket class to implement communication over a network
  - Communication channel does not provide encryption
  - Communication is vulnerable to eavesdropping

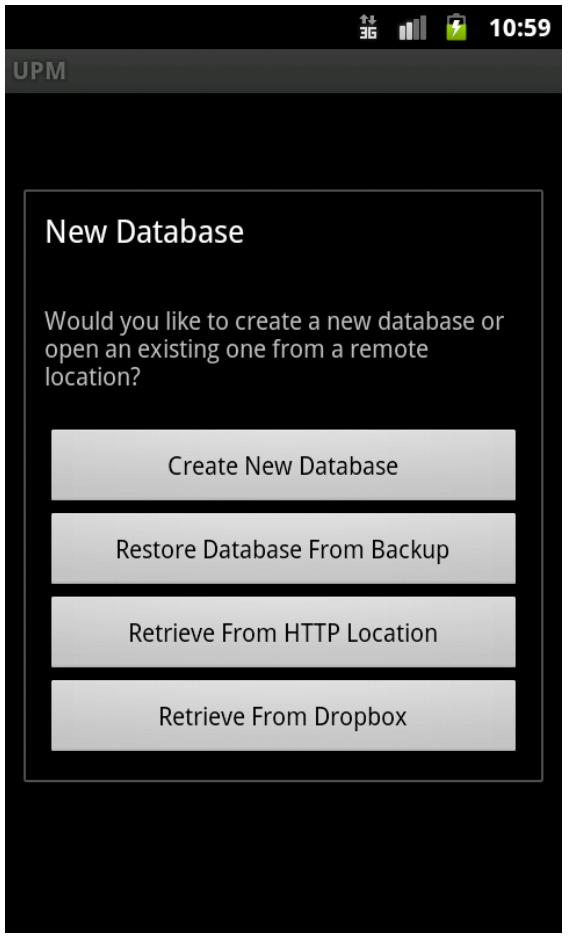
- Object provenance:
  - Same object that flows from user input (confidential message) to client should not flow from client to an object of type Socket

MSC00-J. Use SSLSocket rather than Socket for secure data exchange



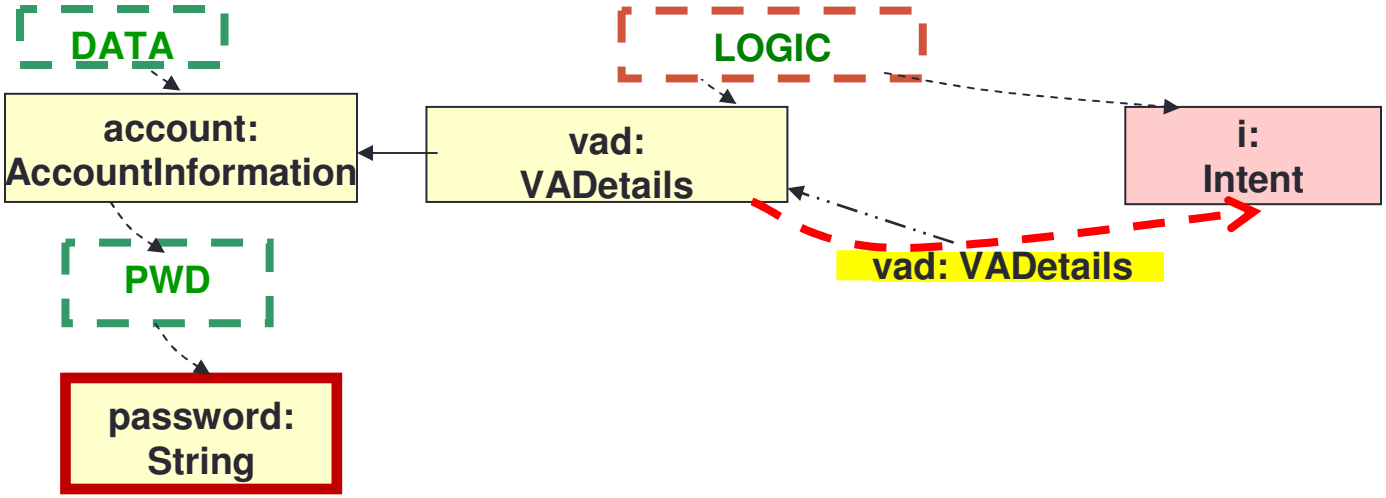
Finding architectural flaws in Android app

- Goal: ensure constraints are expressive
- Intents are like command line arguments used to start an activity [Burns, Black Hat'09]
- Security policy: *Don't put sensitive data into Intents used to start Activities. Callers can't easily require Manifest permissions of the Activities they start, and so your data might be exposed.*
  - *For example processes with the GET_TASKS permission are able to see ActivityManager.RecentTaskInformation which includes the –base Intent used to start Activities.*



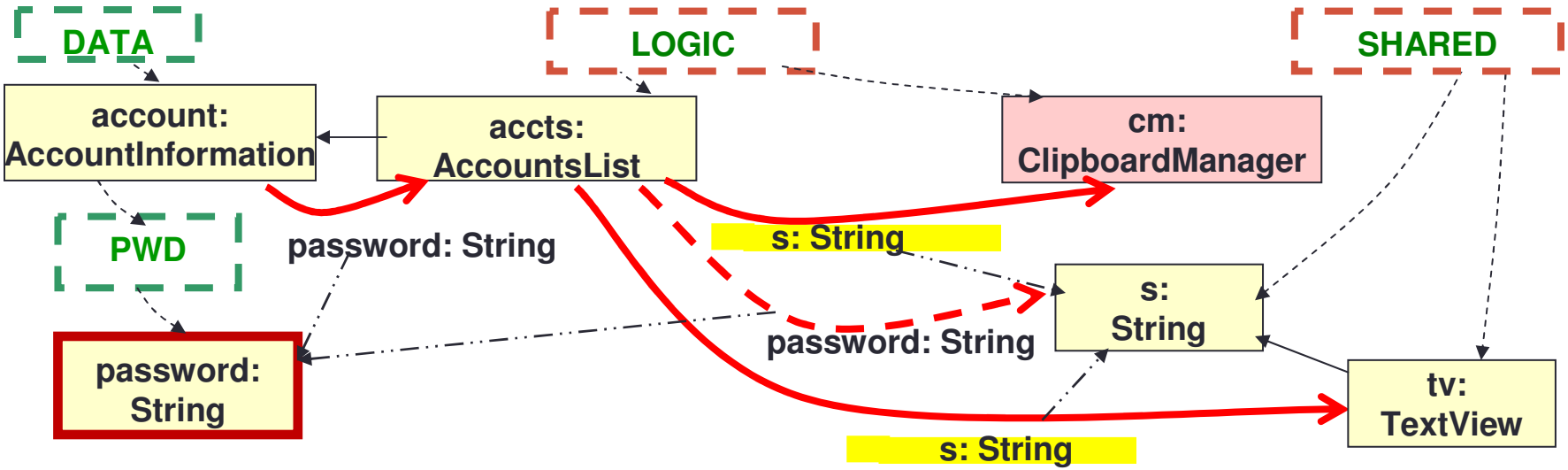
Finding architectural flaws in Android app

- Creation edge refers to vad:VADetails
  - But password:String is reachable from vad:VADetails
  - Destination is untrusted



Finding architectural flaws in Android app

- Copy of password flows to cm:ClipboardManager
  - No permissions needed to access clipboard



Traceability to code: no coding defect

- Copy of password in plaintext is sent to clipboard

```
String getPassword(AccountInformation account){
  return new String(account.getPassword());
}
setClipboardText(getPassword(getAccount(tView)));
```

- Password is reachable from vad:VADetails
  - Framework uses only the name of class and package
  - UPMA sends the whole object. Fix in framework? Fix in app?

```
Intent i = new Intent(VADetails.this, AEditAccount.class);
Intent(Context ctx, Class cls) {
  mComponent = new ComponentName(ctx, cls);
}
ComponentName(Context pkg, String cls) {
  mPackage = pkg.getPackageName();
  mClass = cls;
}
```

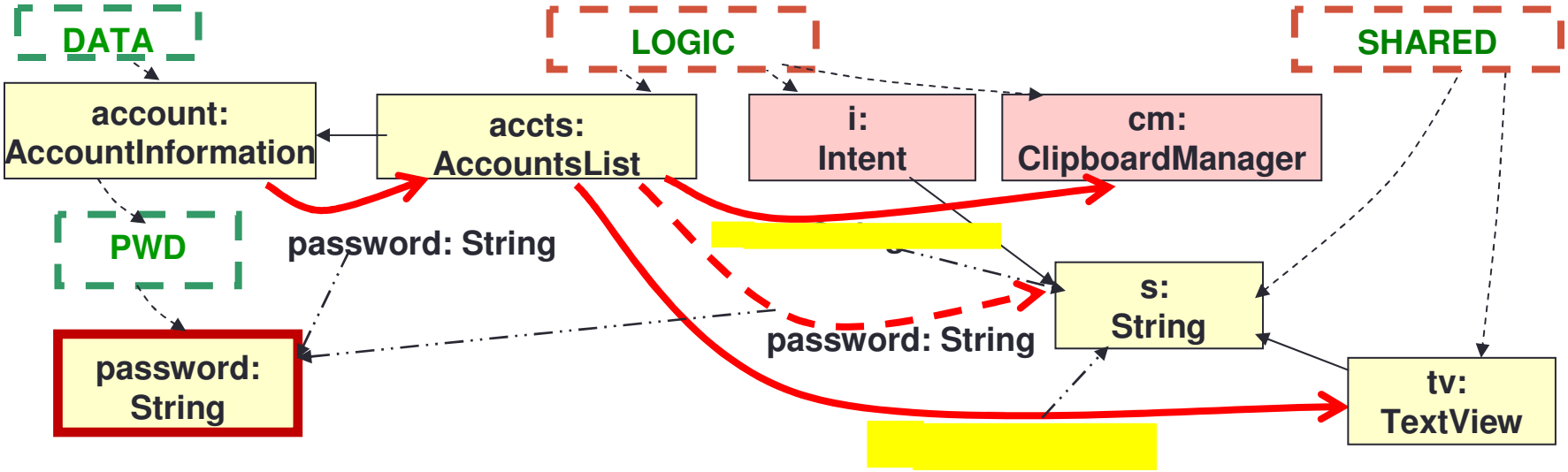
Discussion

- Avoided false positives: 2 false positives for *MSC00-J* if we were to use type information only
  - Multiple objects of type OutputStream
- Not enough to reason about presence and absence of communication
- Constraints can use information from other architectural views
  - Deployment architecture
  - Code architecture
  - Assign IsSerializable.**true** to all objects of type implement Serializable

Limitation: false positives

- UPMA - password is sent to a text view for a user to see
- This is the intended feature in UPMA, not an architectural flaw

```
Class ViewAccountDetails{
 tv.setText(new String(account.getPassword()));
}
```



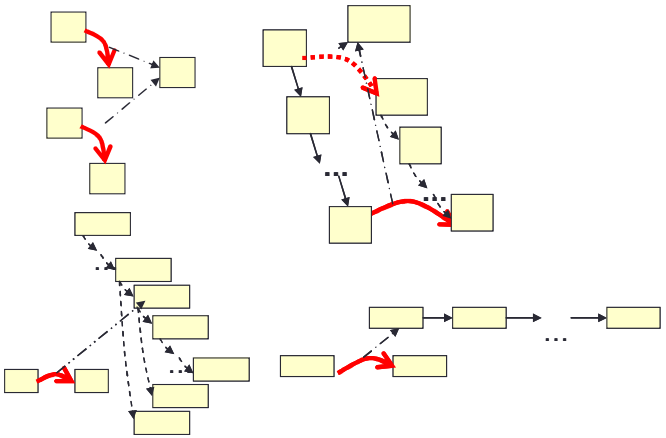
Other limitations

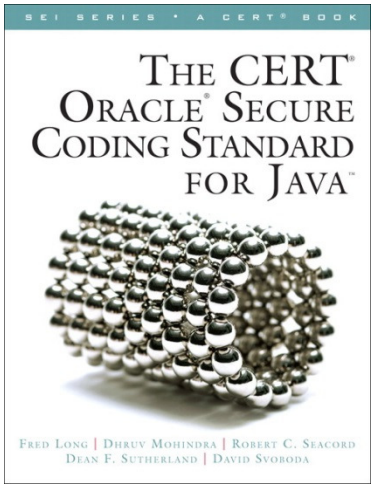
- Scoria supports architectural flaws that deal with structure, rather than behavior (no protocols, no states of objects)
  - **S**poofing
  - ✓**T**ampering
  - **R**epudiation
  - ✓**I**nformation disclosure
  - **D**enial of service
  - **E**levation of privilege

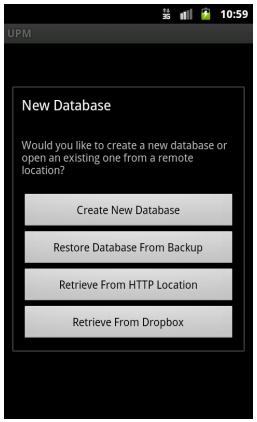
Related work

- Platform level approaches [Mai et al. ASPLOS'13]
  - Vulnerabilities still exist at application level
- Analyses find local coding defects [Hovemeyer and Pugh, SIGPLAN Notices '04]
  - Architectural flaws exist even in code without defects
- Dynamic analysis [Marron et al. TSE'13]
  - Infer only strict hierarchy
- Monitor the application at runtime [Enck et al. OSDI'10]
  - Effective only after release
- Architectural-level approaches [Sohr and Berger ESSOS'10, Almorsy et al. ASE'12]
  - Use code architecture and AST-based representations
- Taint analysis [Tripp et al. PLDI'09, Fritz et al. Tech.Rep'13, Manuel et al. ASE'13]
  - Mixes analysis with constraints and properties
- Query flat object graph [Martin et al. OOPSLA'05]
  - More precision from object hierarchy

Conclusion







- Recent work supports Architectural Risk Analysis by extracting a sound approximation of a runtime architecture
- Constraints implementing CERT rules for which automatic support was previously unavailable
- Scoria can find information disclosure in Android app
- Future work
  - Compare Scoria to related approaches based on benchmarks
  - Study how security architects use Scoria