CS 598 - Computer Security in the Physical World:
Project Submission #2

Professor Adam Bates
Fall 2016

Last Deliverable (BG+RW)

- I will have feedback back to you by this evening.
- Generally positive impressions. Did people have trouble writing these?
- Poor scores were mostly due to people failing to follow instructions.
- If you fix the issues that I identified prior to the final submission, the first deliverable will not have a major impact on your final course project.

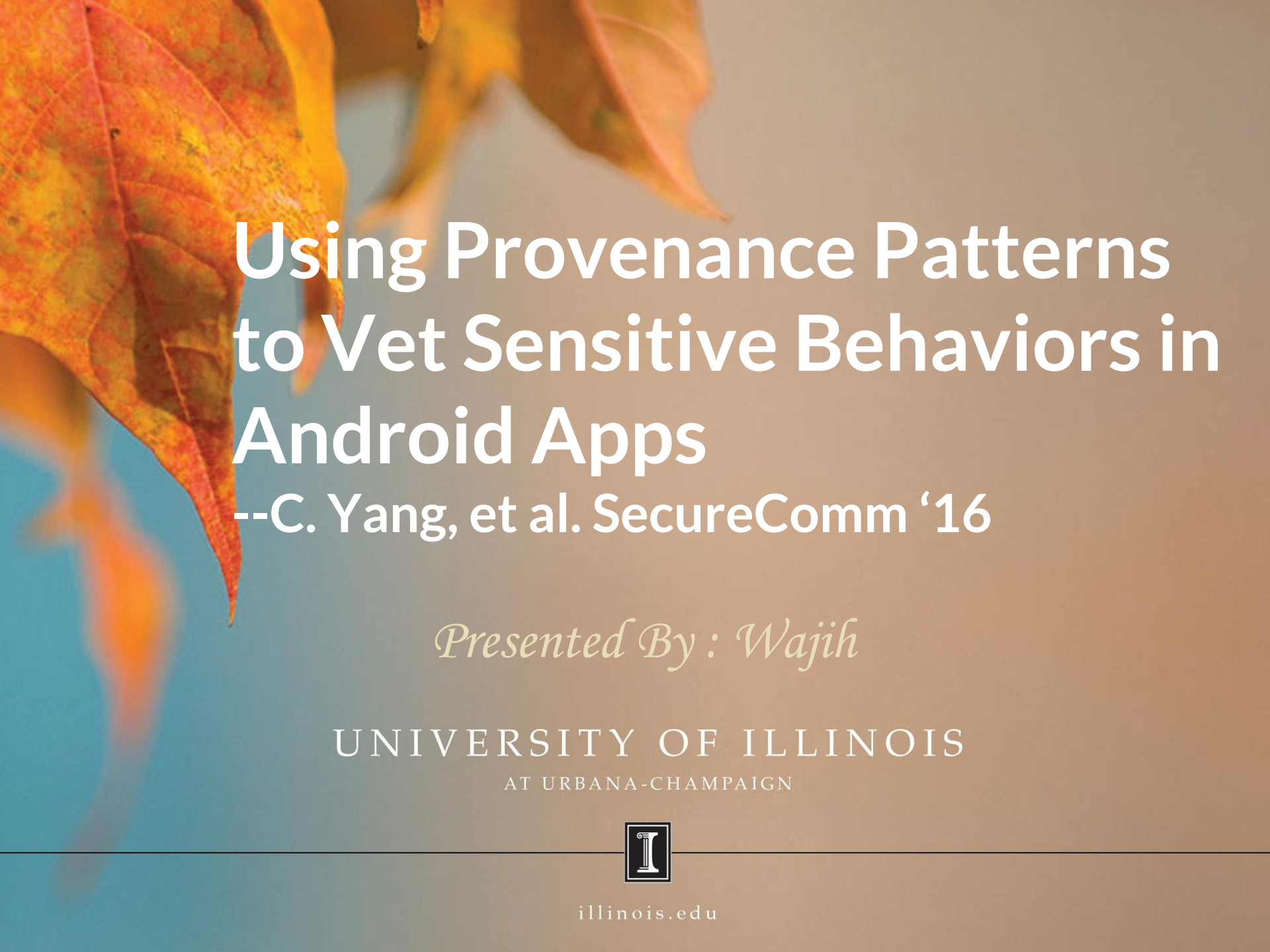
Oct 18th Deliverable

- Last submission we "locked in" the background and related work sections of our paper. This submission will be a living document as you begin to do the real work.
- Purpose: Tell me specifically what are you doing to DO in your project
- Format: Add a new section called "Experimental Proposal" to the LaTeX Two Column ACM document.
- Submission: Email me the PDF (include [cs598] in subject)

Oct 18th Deliverable

Be sure to include:

- Hypothesis: Based on what you've learned so far, "commit" to a prediction that is the basis of your paper.
  - What does a hypothesis in a defensive paper look like?
- Methodology and/or Design: What techniques are you going to use? How are you going to use them? Will you leverage existing tools? Convince me that you will succeed in executing your methodology.
- Evaluation and/or Analysis: How will you determine the extent to which you have succeeded in your goal?



Using Provenance Patterns to Vet Sensitive Behaviors in Android Apps

--C. Yang, et al. SecureComm '16

Presented By : Wajih

UNIVERSITY OF ILLINOIS

AT URBANA-CHAMPAIGN

illinois.edu

Problem

How can you tell whether software you
– Develop
– Buy
-Install
is **safe** to run?

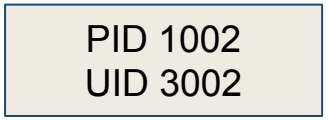
What is Provenance?

- Provenance, a.k.a. lineage of data
  - Data's life cycle
    - Origins
    - Accesses
    - Deletion
  - Provides confidence in authenticity

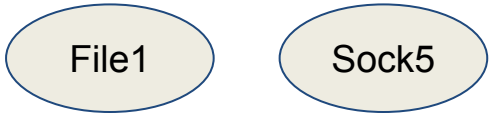
- Was the latest data used in the computation?
- Was the data deleted after its use?
- Was the sensitive data sent on network?

Open Provenance Model (OPM)

Process Vertices: Represent dynamic entities
e.g. operating system processes



Artifact Vertices: Represent static elements that are consumed or produced by processes.
e.g. files, sockets etc.

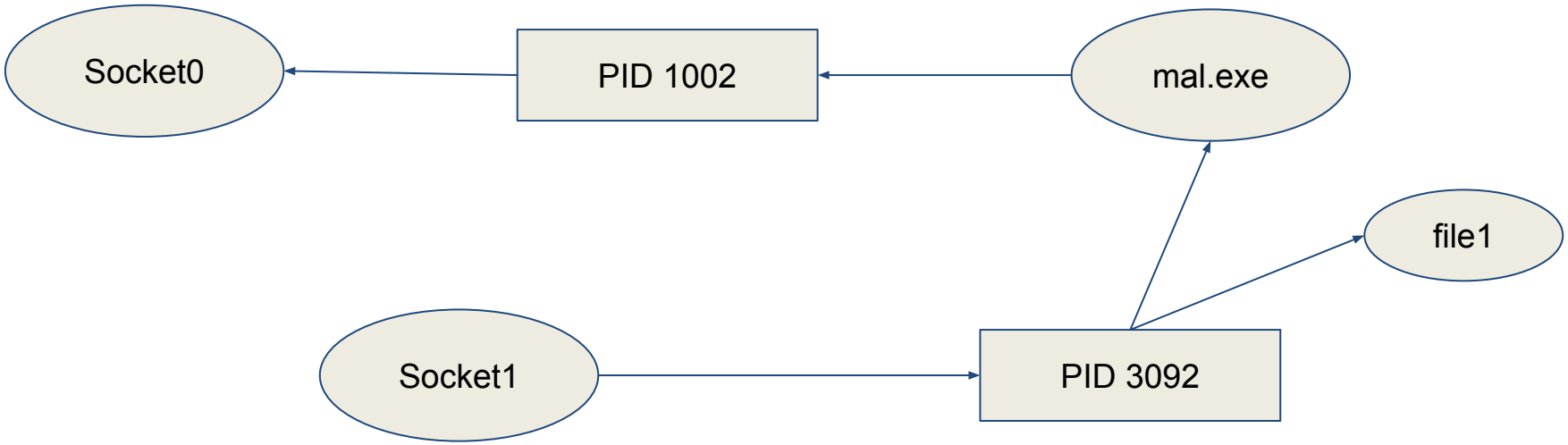


Edges: Represent dependency between pair of vertices; Directed

- *WasTriggeredBy:* from a process to another process
- *WasGeneratedBy:* from an artifact to a process
- *Used:* from a process to an artifact
- *WasDerivedFrom:* from an artifact to another artifact.

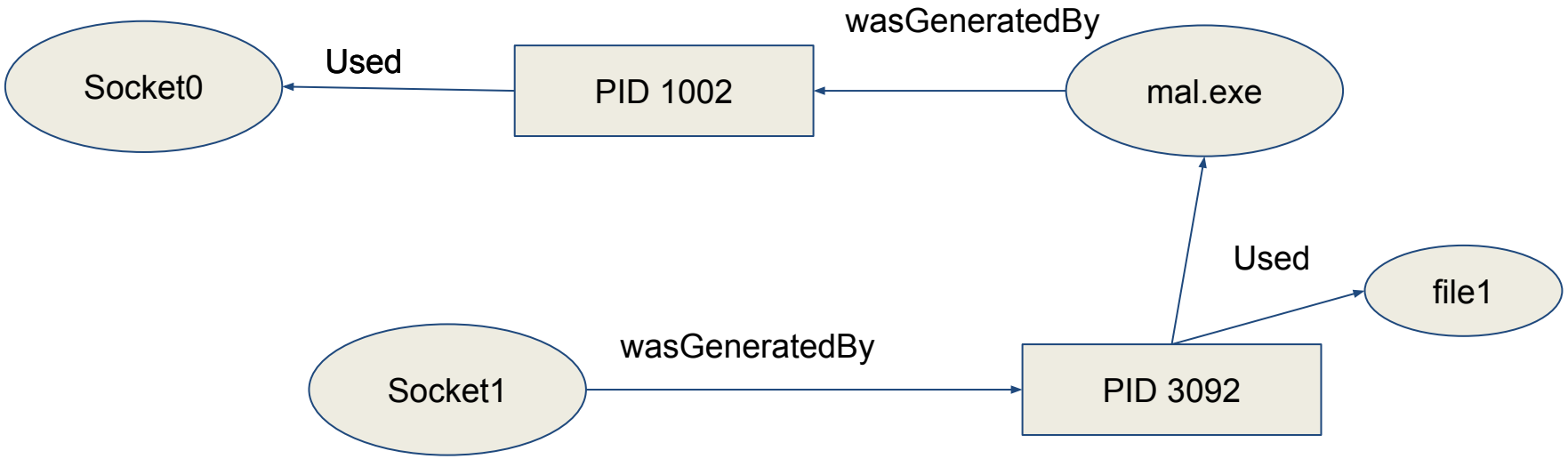
Provenance Graph

- Log all the system events and then make OPM relationships to generate provenance graphs
- **Example:** Download file from network and executes it:



Provenance Graph

- Log all the system events and then make OPM relationships to generate provenance graphs
- **Example:** Download file from network and executes it:



Program Behavior Analysis

Traditional Analysis methods

Static Analysis: Investigates the properties that can be investigated by inspecting the downloaded app and its source code only. Eg: Signature based inspection used by anti-virus technologies.

Dynamic Analysis: In this method, app is run in a secure environment such as sand-box and logs every relevant operation of the app.

Static vs Dynamic Analysis

Static –

- Consider all possible inputs
- Can prove absence of bugs/vulnerability

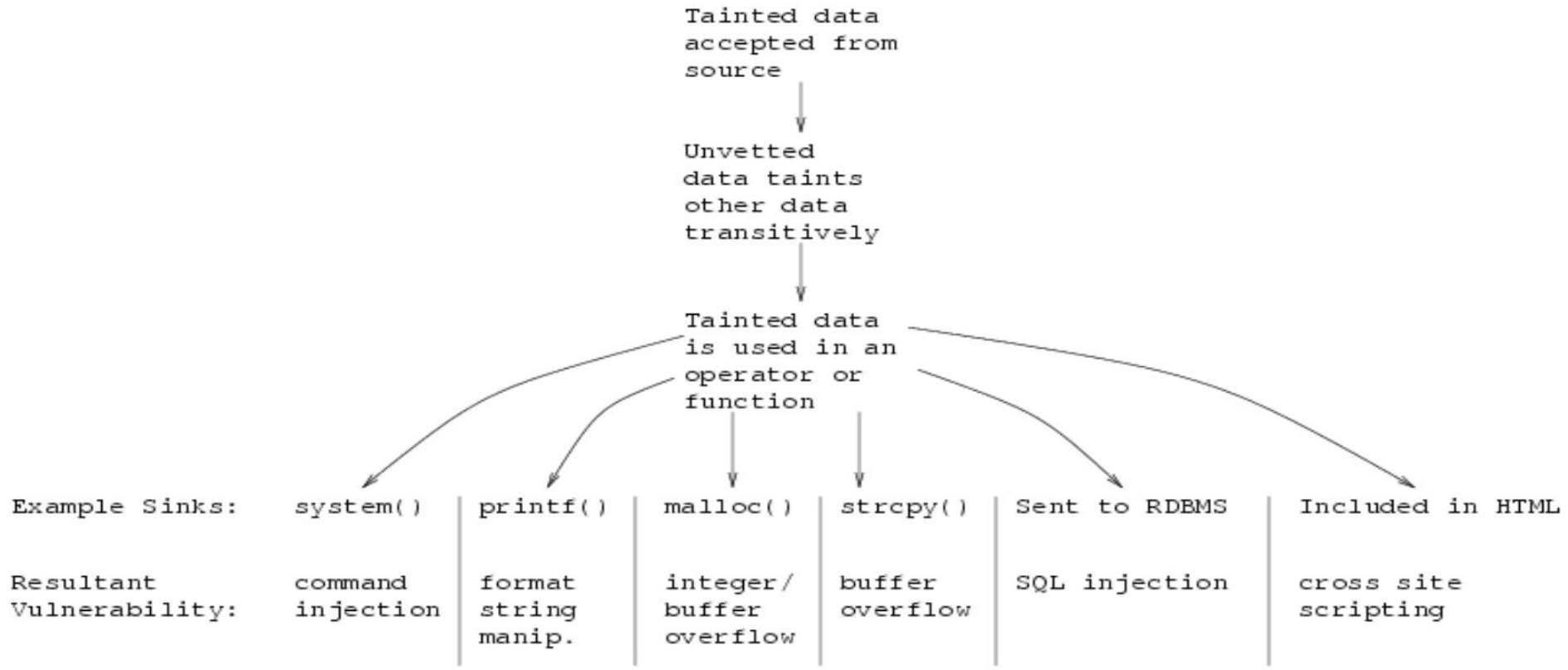
- Cannot handle code obfuscation (Java reflections, source encryption etc)
- Cannot find vulnerabilities in RT
- HUGE overhead, reason about every input

Dynamic –

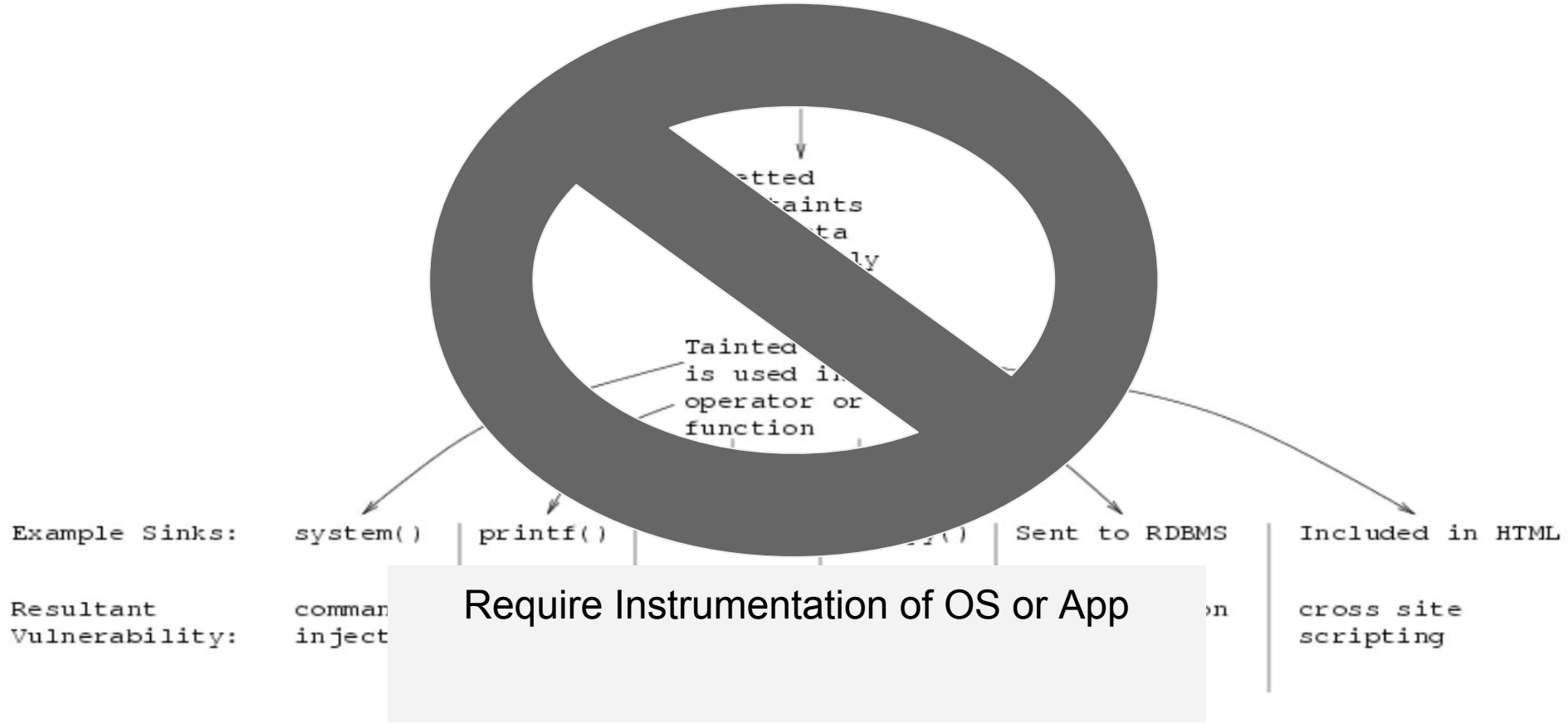
- Consider some inputs
- Can prove presence of bugs/vulnerability

- Cannot guarantee the full coverage of the source code
- May require virtual Machine instrumentation

Example: Taint Checking Solution for Behavior Analysis



Example: Taint Checking Solution for Behavior Analysis



Require Instrumentation of OS or App

Paper's Approach

- Collect provenance of the applications behaviors **without** modifications and generate provenance graphs
- Pattern match generated provenance graphs with sensitive behavior patterns

Dagger

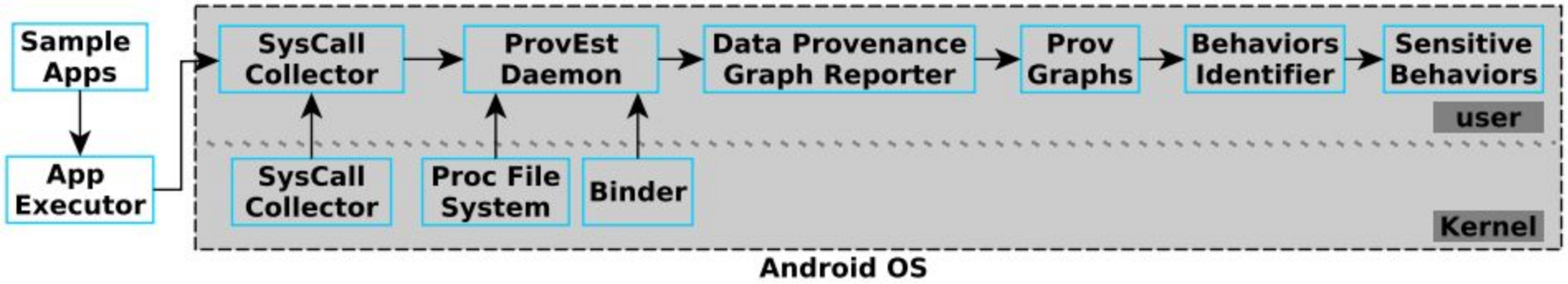
- A lightweight system to dynamically vet sensitive behaviors in Android apps by making provenance graphs.
  - No VMI instrumentation
  - Less Overhead
  - Just tracks apps interactions with underlying platform

- Dagger uses the open source **SPADE** provenance middleware to collect three types of low-level execution information:
  - Linux System Calls -- via Strace
  - Android Binder transactions -- via sysfs
  - App process details -- via /prof fs

System Design

- **App Executor:**
  - Executes the app in a sandbox
- **Syscall Collector:**
  - Collects the system call invocation
- **ProvEst Daemon:**
  - Collects information from binder transactions; Builds relationships between artifacts and process
- **Graph Reporter:**
  - Outputs Data provenance graph from the relationships made by ProvEst daemon
- **Behavior Identifier:**
  - Detects sensitive behaviors from the provenance graph

Dagger runs each app, collect provenance records and perform pattern matching



Collection of Sensitive Behaviours

- They ran Android apps with selected input that is known *a priori* to trigger sensitive behavior
- Dependent on internal working of Android

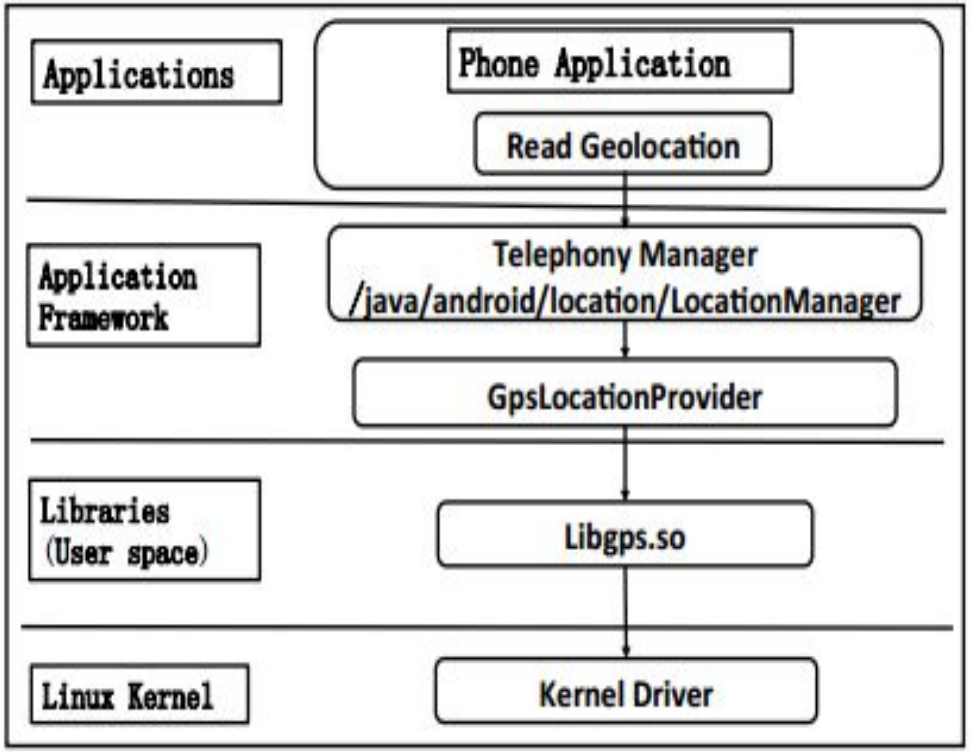
Table 2. Fined-grained sensitive behaviors associated with malicious behaviors

Index	Malicious Behaviors	Sensitive Behaviors	Index	Malicious Behaviors	Sensitive Behaviors
1	Phone Call	Phone Call	5	Steal Contact	Read Contact and Net
2	Send SMS	Send SMS	6	Track Location	Read Location and Net
3	Block SMS	Receive SMS, not Write SMSDB	7	Execute Shell	Execute Shell
4	Steal SMS	Read SMSDB and Net	8	Net	Net

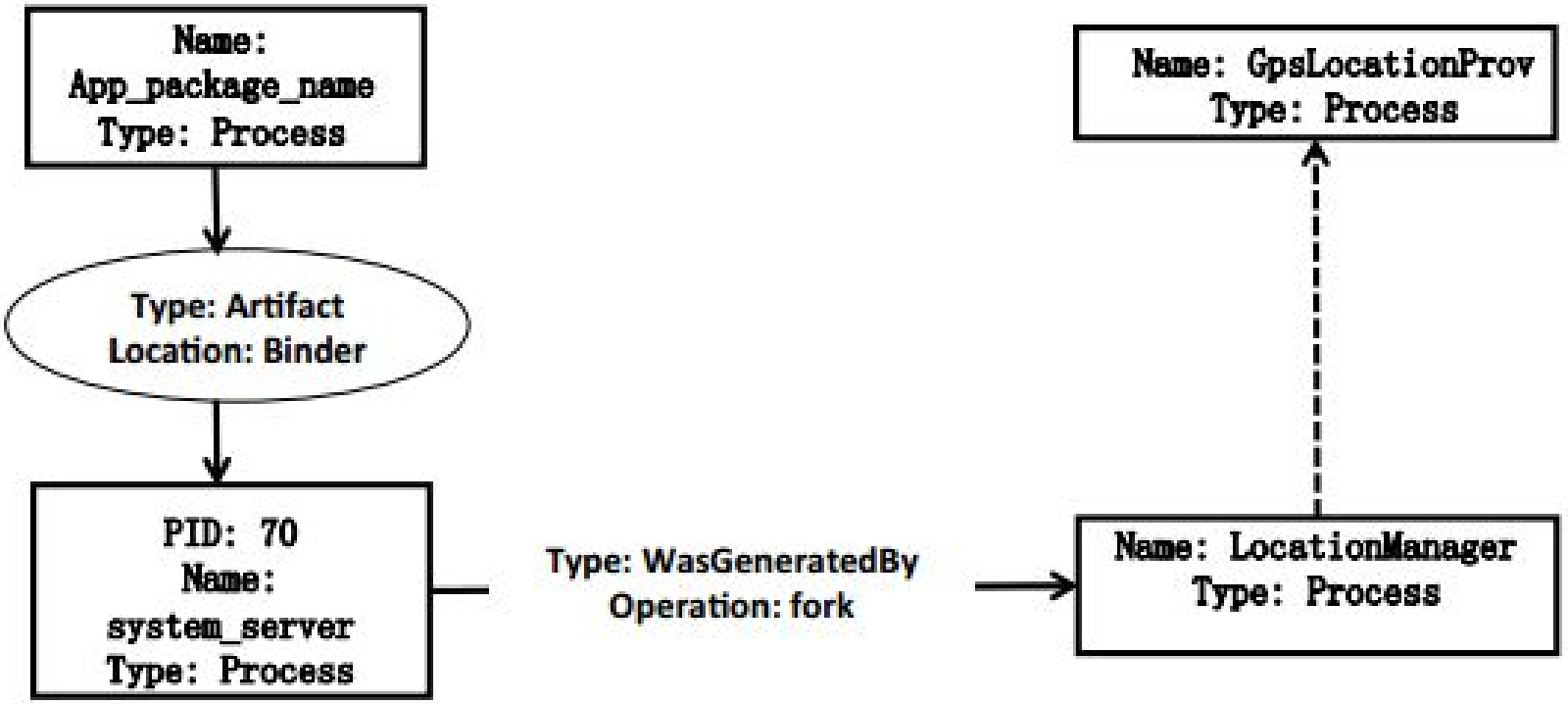
Example Sensitive Behaviour

Pattern : Read Geolocation:

- An app attempts to read the geographic location
- Interacts with the Location Manager Service
- Requests the location from the GpsLocationProvider.



Provenance Generated from the Behaviour



Evaluations

Evaluated **EFFECTIVENESS** from three perspectives:

1. Vetting real-world malware case studies
   1.1. Gamex: Code Encryption
   1.2. Gone60: Privacy Leakage
   1.3. Zsone: SMS Service Usage

2. Vetting Android Genome Project malware [S&P 2012]
   2.1. 1,260 real-world malware samples collected from the Genome Project

3. Vetting official market (Google Play) apps
   3.1. 1000 apps sampled from 18,527 official market (Google Play)

Evaluations

- Evaluated **EFFICIENCY** from three perspectives:
  (i) CPU overhead
  (ii) Memory overhead
  (iii) Runtime overhead

- Used AnTuTu (android benchmark app) to test performance

- Both CPU and Memory overhead was less than 10 %
- The runtime overhead is less than 63%
  - Strace overhead

Discussion

What are the key contributions of this paper?

What are the limitations of this paper?

Can we use Provenance Patterns for something else?

BACKUP SLIDES

Taint Analysis
Explicit – Passed in assignments
Implicit – Passed in control flow structures (ICC, Broadcasts, Media)

Data-Flow Analysis
Most use some form of Data Flow Analysis with Taint Analysis
Sources: Location Data, Unique IDs, Call State, Authentication Data and Contact/Calendar Data
Sinks: SMS Communications, File Output, Network Communication, Intents, Content Resolver
Examples: TaintDroid (2010), AndroidLeaks (2012)

Android Architecture