SMV-HUNTER: *Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps*

David Sounthiraraj **Justin Sahs** Garret Greenwood
Zhiqiang Lin Latifur Khan

University of Texas at Dallas

February 26, 2014

Problem Statement

- Many Android apps use SSL/TLS to transmit sensitive data
- Android allows developers to override the built-in validation
  - Used to connect to servers whose certificates come from non-standard Certificate Authorities (CAs)
  - Used to avoid purchasing certificates for testing or user acceptance environment
  - Can lead to SSL Man-in-the-Middle Vulnerabilities (SMVs)

SSL/TLS

In SSL/TLS, a server's identity is verified by a certificate chain. A chain is valid if:

- Each certificate has not expired
- The root certificate of the chain is from a CA present in the keystore
- Each certificate has a valid cryptographic signature from the CA immediately after it in the chain

Additionally, the certificate chain's hostname must match the domain name being connected to (possibly with wildcards).

Example Vulnerability

A famous example is the Chase Banking App (CVE-2012-5810):

```
1  public final void checkServerTrusted(X509Certificate[]
2        paramArrayOfX509Certficate, String paramString)
3  {
4    if ((paramArrayOfX509Certficate != null) && (
5         paramArrayOfX509Certficate.length == 1))
6      paramArrayOfX509Certficate[0].checkValidity();
7    while (true)
8    {
9      return;
10     this.a.checkServerTrusted(
11         paramArrayOfX509Certficate,paramString);
12   }
13 }
```

(from (Georgiev *et al.*, 2012))

Example Vulnerability

A famous example is the Chase Banking App (CVE-2012-5810):

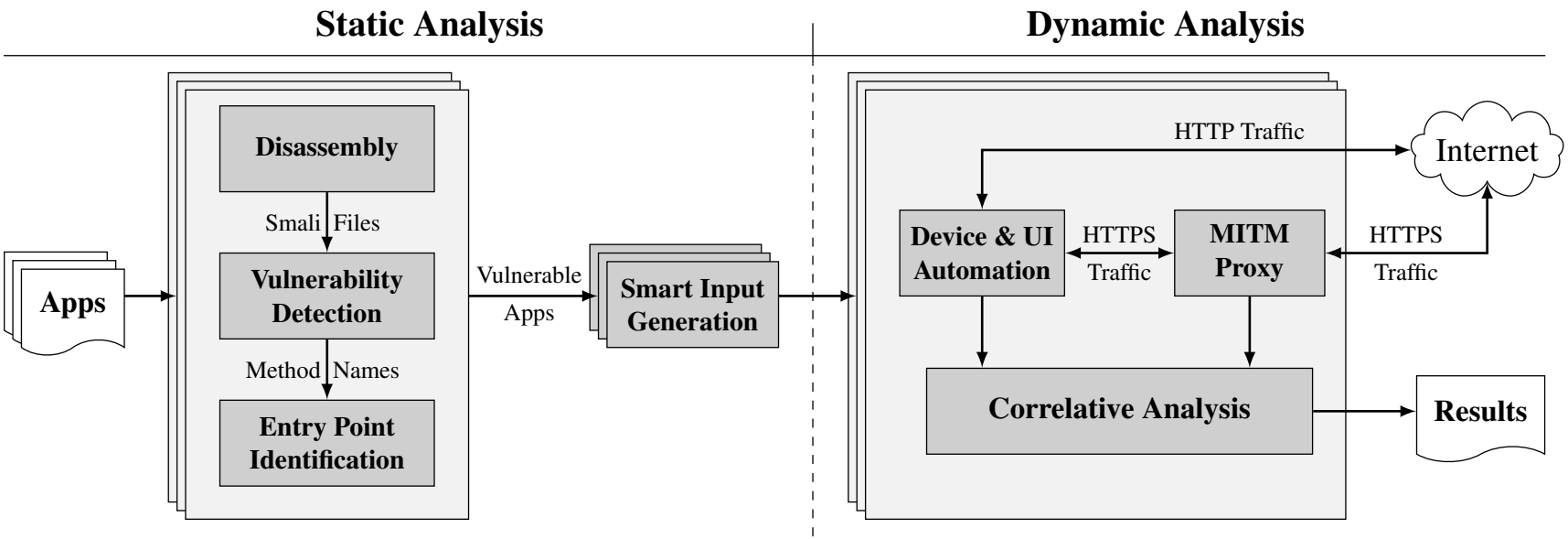
```
1  public final void checkServerTrusted(X509Certificate[]
2        paramArrayOfX509Certficate, String paramString)
3  {
4    if ((paramArrayOfX509Certficate != null) && (
5         paramArrayOfX509Certficate.length == 1))
6      paramArrayOfX509Certficate[0].checkValidity();
7    while (true)
8    {
9      return;
10     this.a.checkServerTrusted(
11         paramArrayOfX509Certficate,paramString);
12   }
13 }
```

(from (Georgiev *et al.*, 2012))

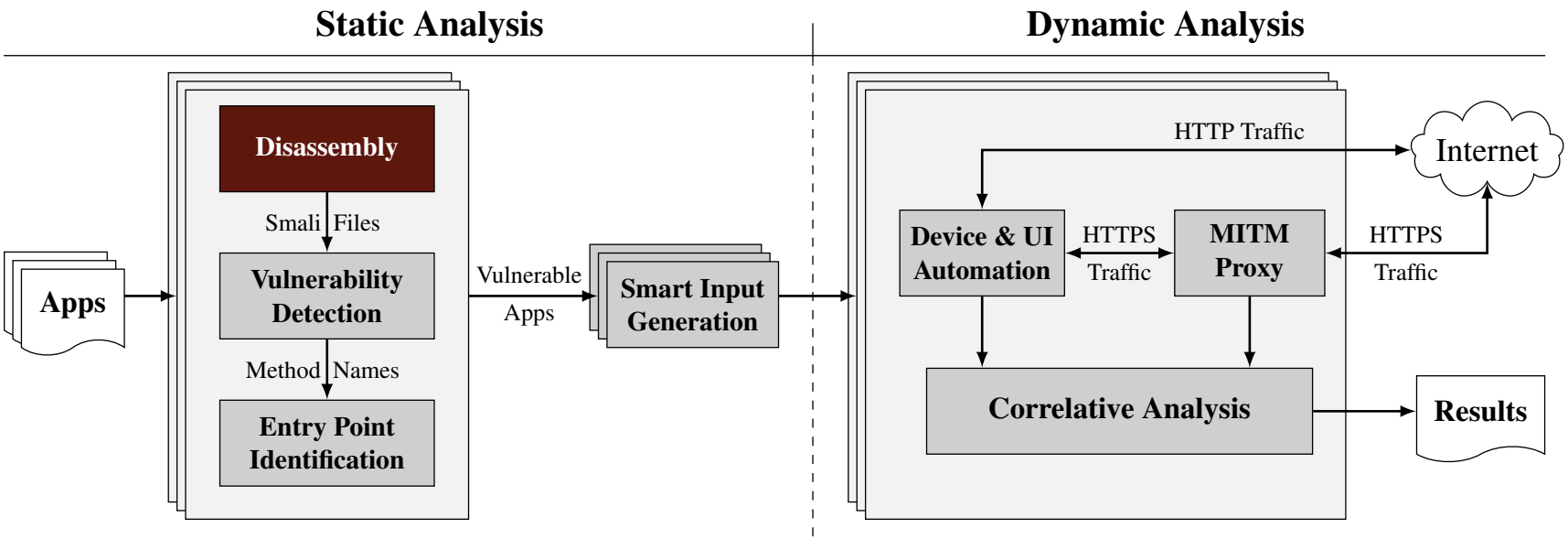
Approach

- Purely static analysis unreliable
- Purely dynamic analysis infeasible
  - enumerate all possible UI interaction paths
  - text input
- We propose a hybrid approach
  - use static analysis to prune the search space for and provide valid text to dynamic analysis

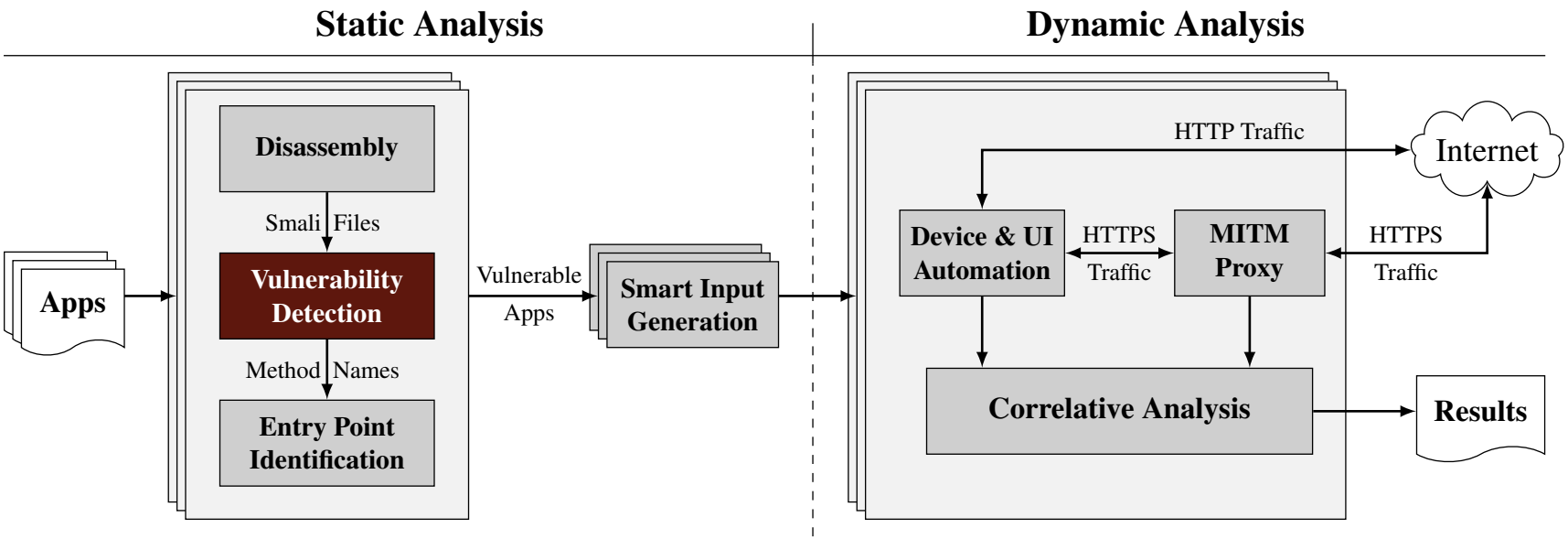
System Overview



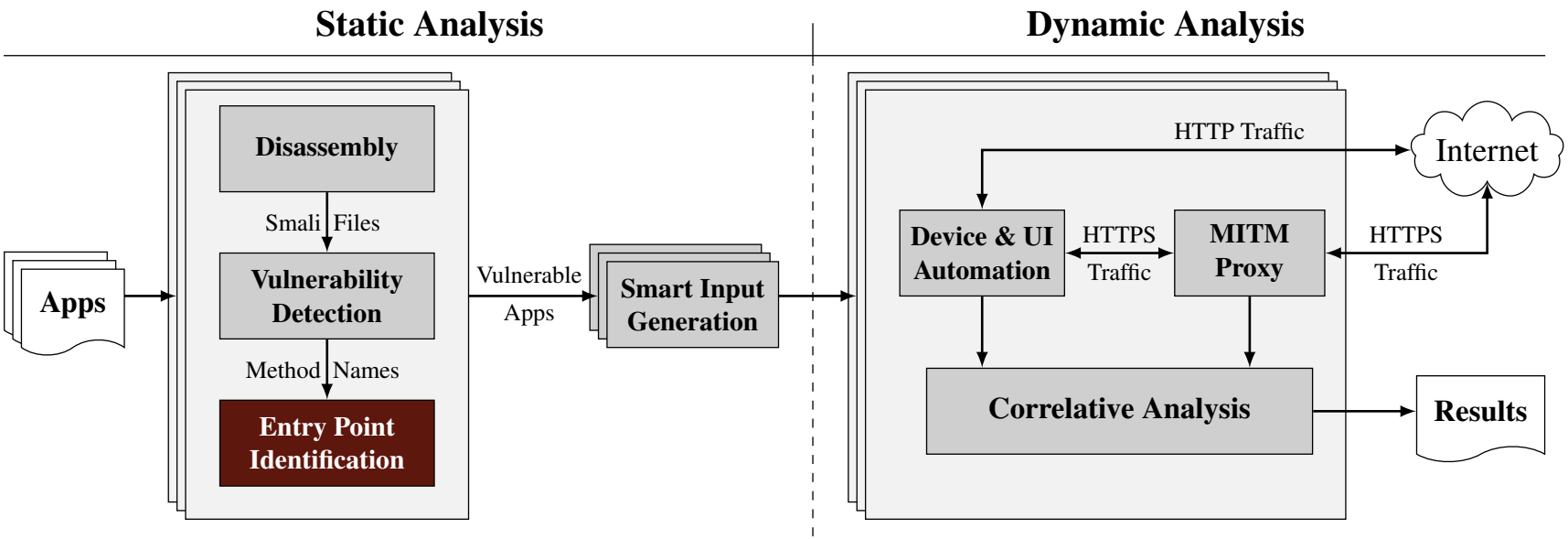
System Overview



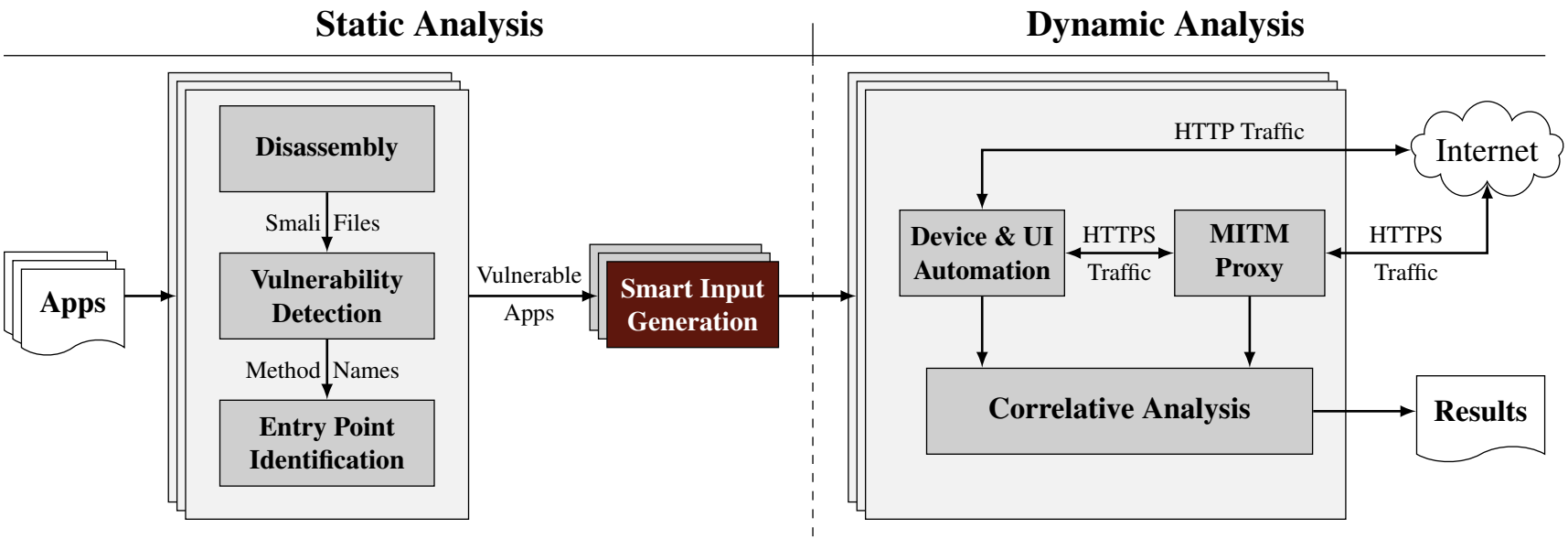
System Overview



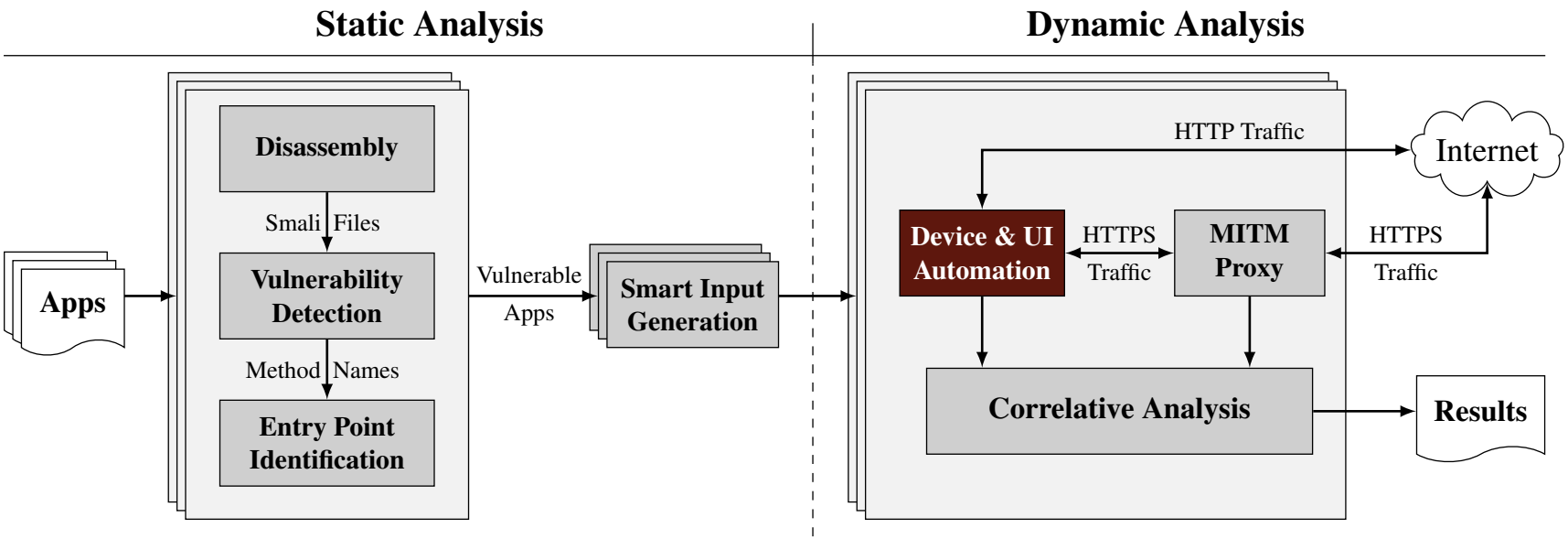
System Overview



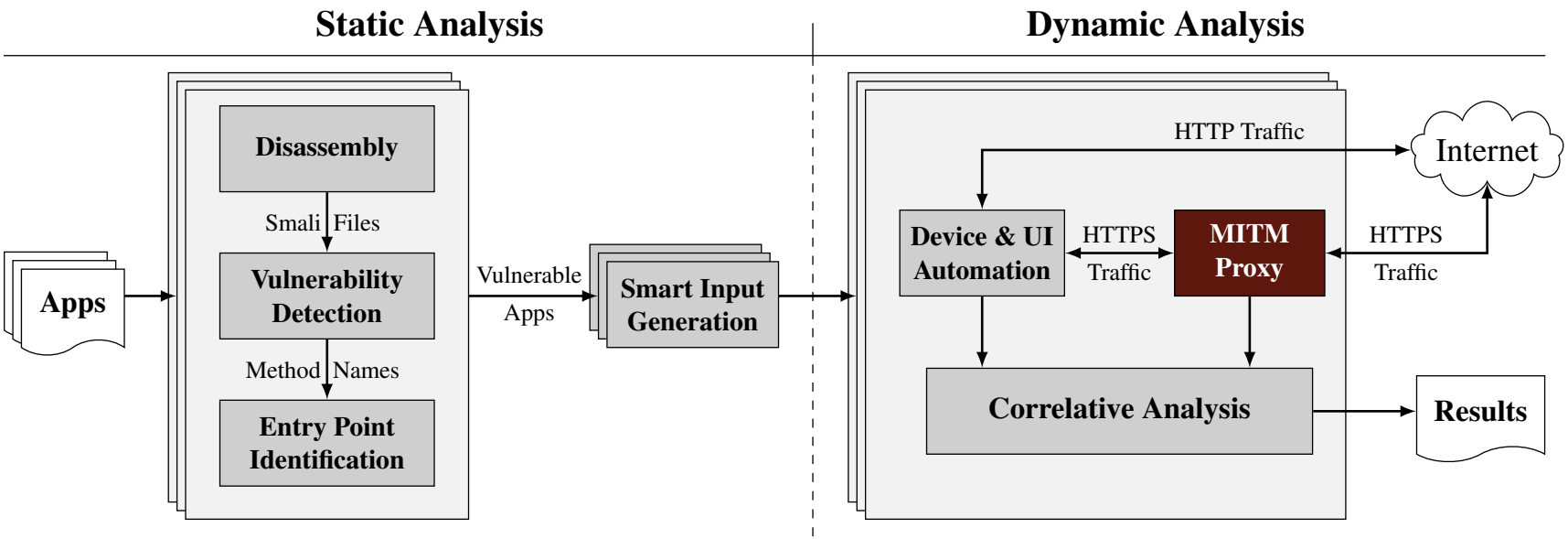
System Overview



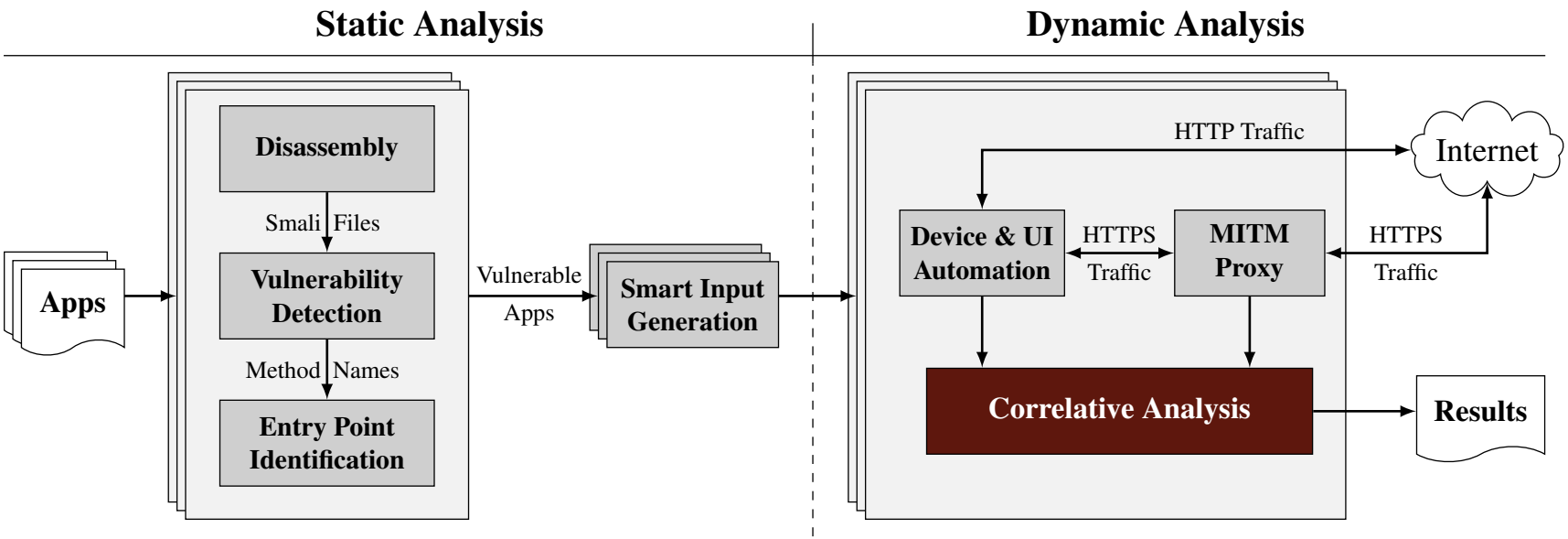
System Overview



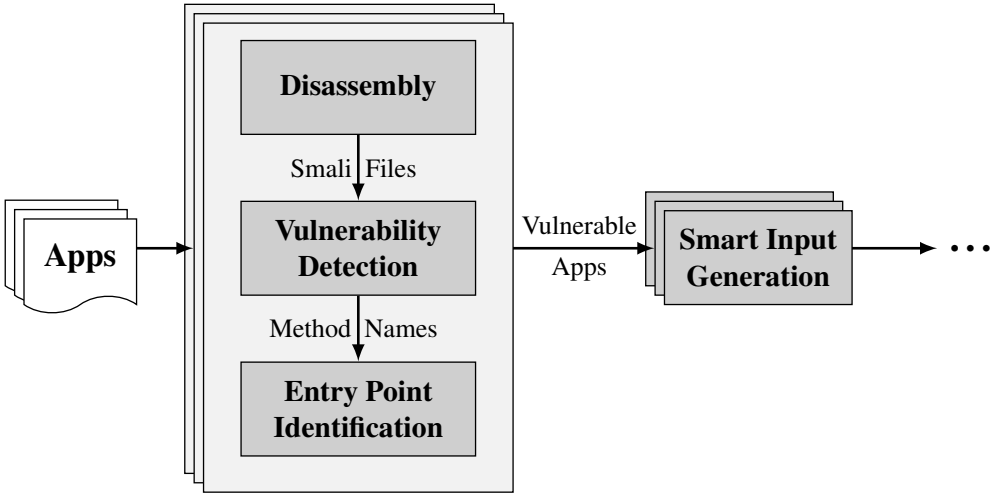
System Overview



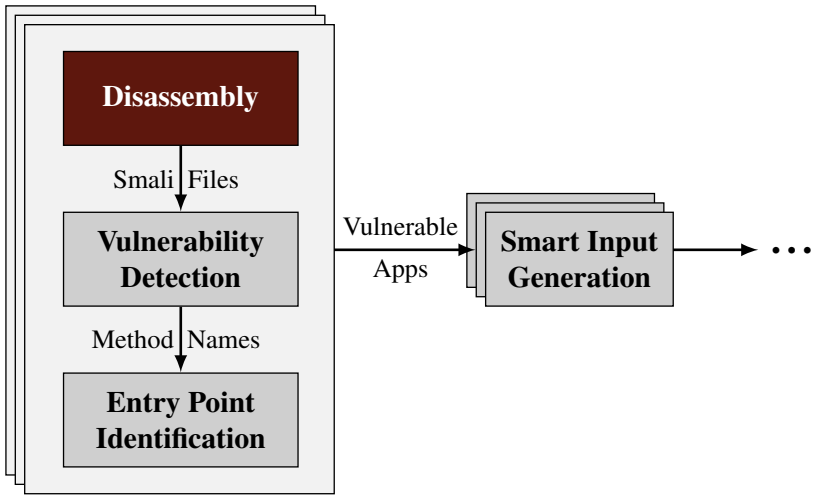
System Overview



Static Analysis

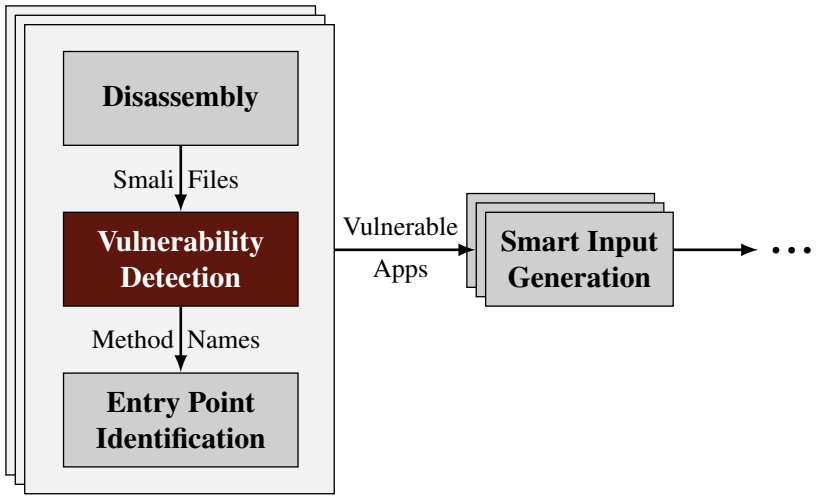


Disassembly



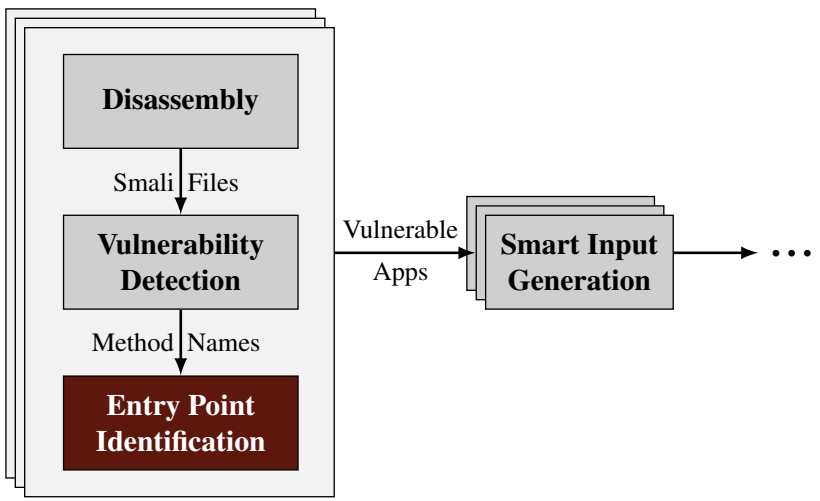
- `apktool` to disassemble the packaged compiled code into a human-readable format called `Smali`.
- Significantly faster and more reliable than decompilation, especially when the code has been obfuscated

Static SMV Detection



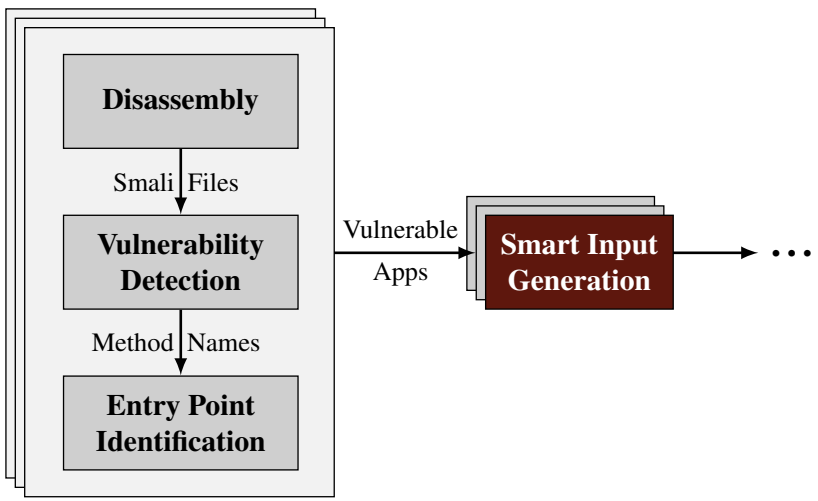
- Simply check whether the X509TrustManager or HostNameVerifier interfaces have been overridden
- Apps that do not override these either do not use SSL or use the built-in SSL support without modification

Vulnerable Entry Point Identification



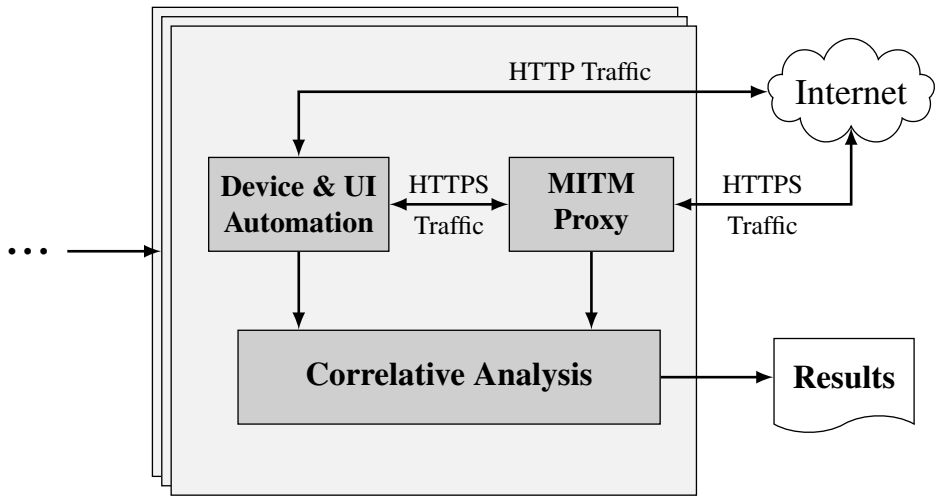
- Each app can be started at a number of entry points (called *activities*)
- Many entry points will not trigger secure connections
- Trace backwards through method calls to identify entry points that might trigger potential vulnerabilities

Smart Input Generation



- Apps often perform validation on text input or convert text to other datatypes (e.g. integers)
- Intelligently provide input based on:
  - Input type annotations
  - Type cast operations in the code

Dynamic Analysis



Device Management

For completeness and scalability, our system must:

- Manage multiple emulators in parallel,
- Handle emulator crashes and other errors,
- Schedule and distribute app testing across running emulators, and
- Collect and manage log data including installation and uninstallation details and network traffic.

Device Management

The device management component has two threads:

- Emulator Management
- App Scheduling

Device Management

The device management component has two threads:

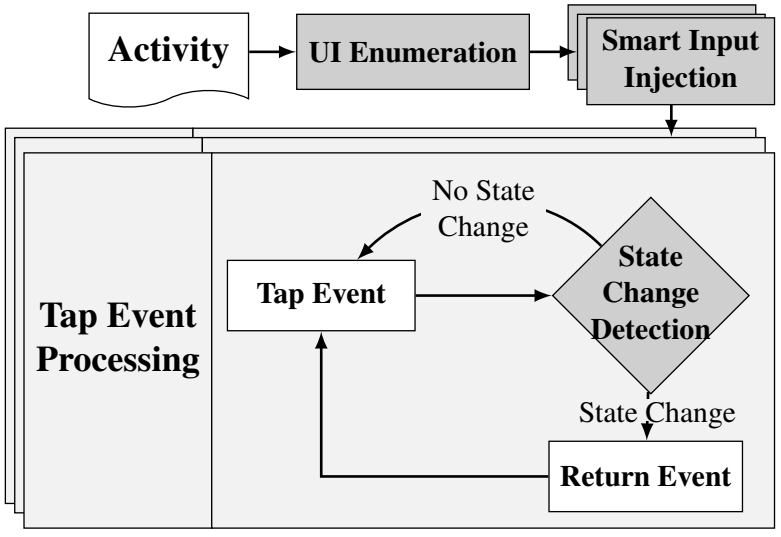
- Emulator Management
  - Maintains a pool of active and free emulators
  - Monitors the state of each emulator, restarting ones that go "offline" or crash
- App Scheduling

Device Management

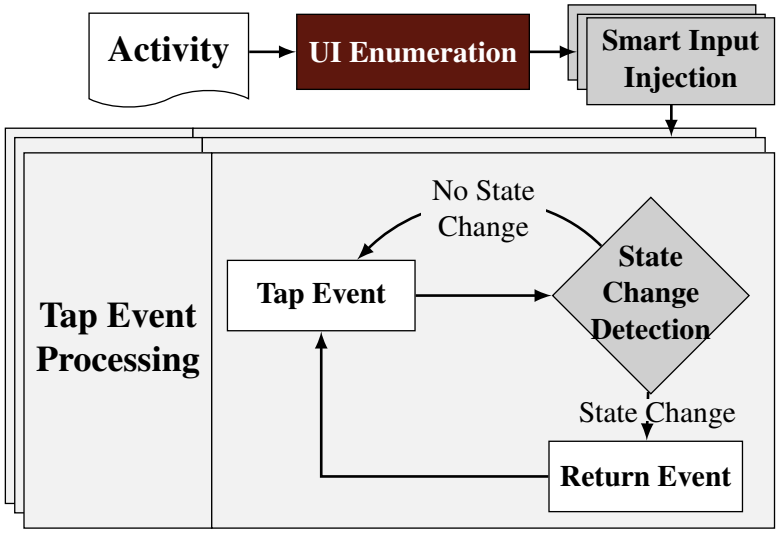
The device management component has two threads:

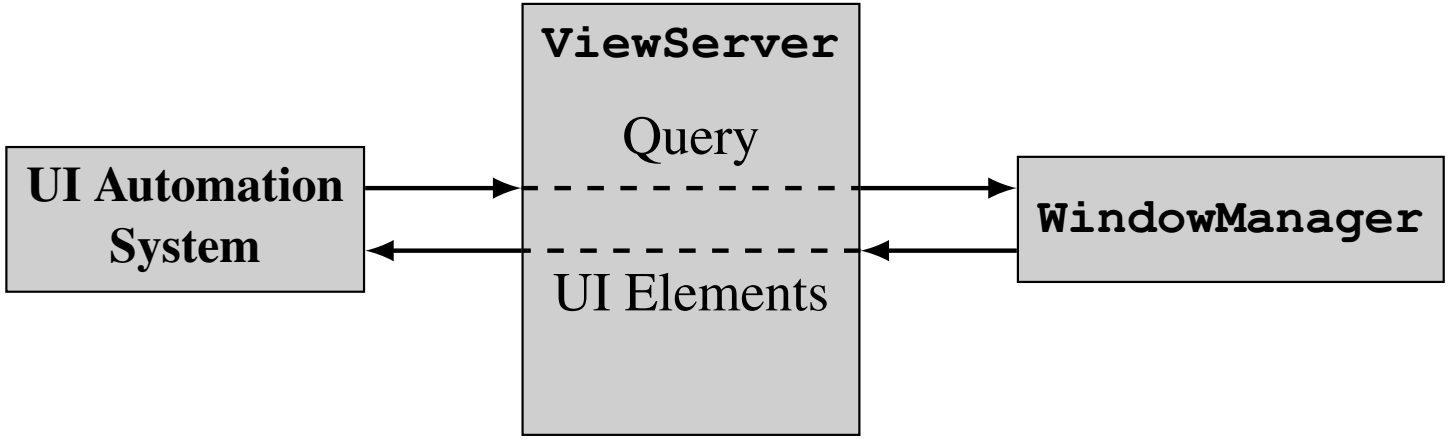
- Emulator Management
- App Scheduling
  - Executes UI Automation on each activity identified by static analysis
  - Handles errors that do not crash the emulator (e.g. app crashes)
  - Logs installation/uninstallation timestamps and DNS queries

UI Automation

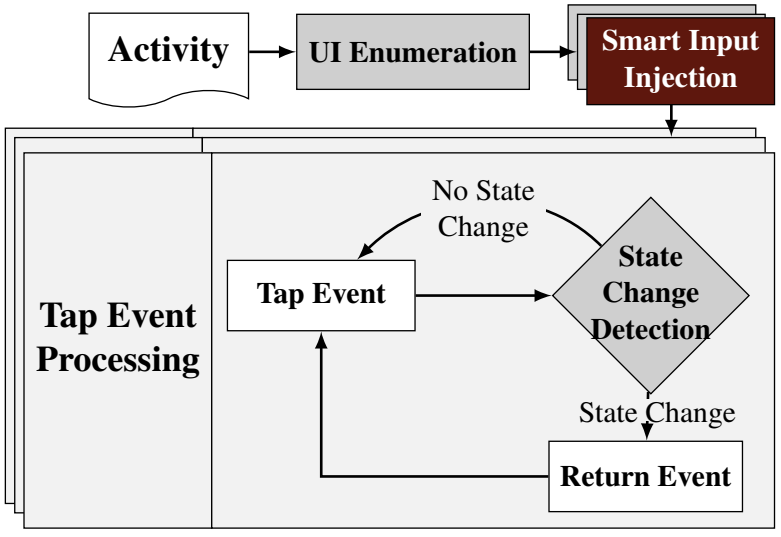


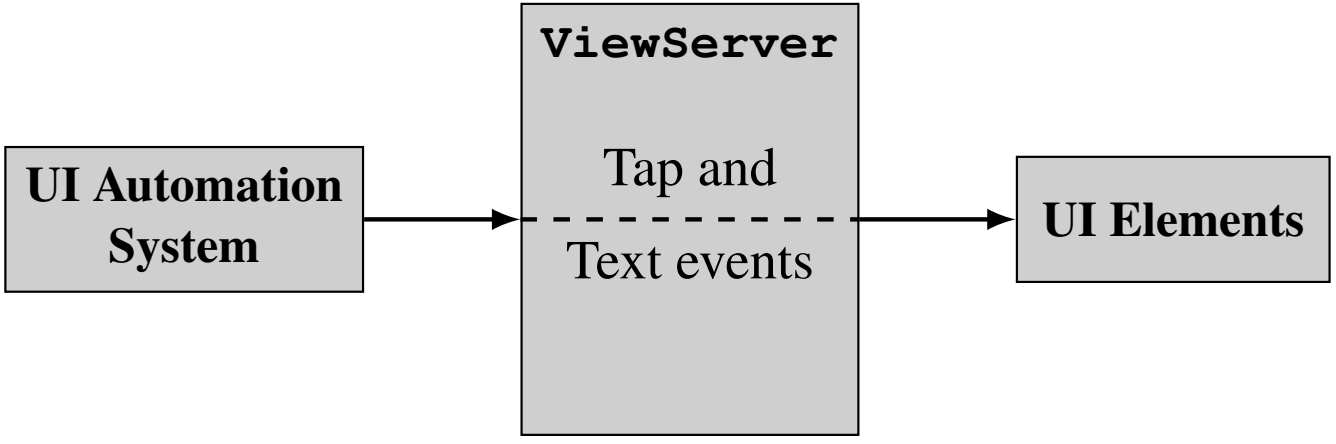
UI Automation



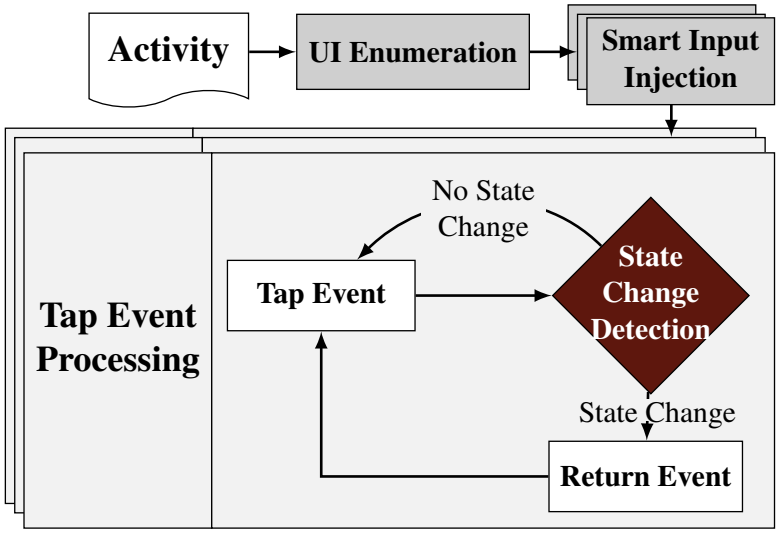


UI Automation





UI Automation



- The system uses `WindowChange` and `FocusChange` events that are triggered when the interface changes
- Back button events are used to return to the target activity
  - When a "non-cancellable" dialog appears that disables the back button, events are generated to tap on "`OK`" or "`Cancel`" buttons

MITM Proxy

- During UI automation, all HTTPS traffic is directed through a proxy that provides illegitimate certificates for each connection
- Successful connections are logged

Correlative Analysis

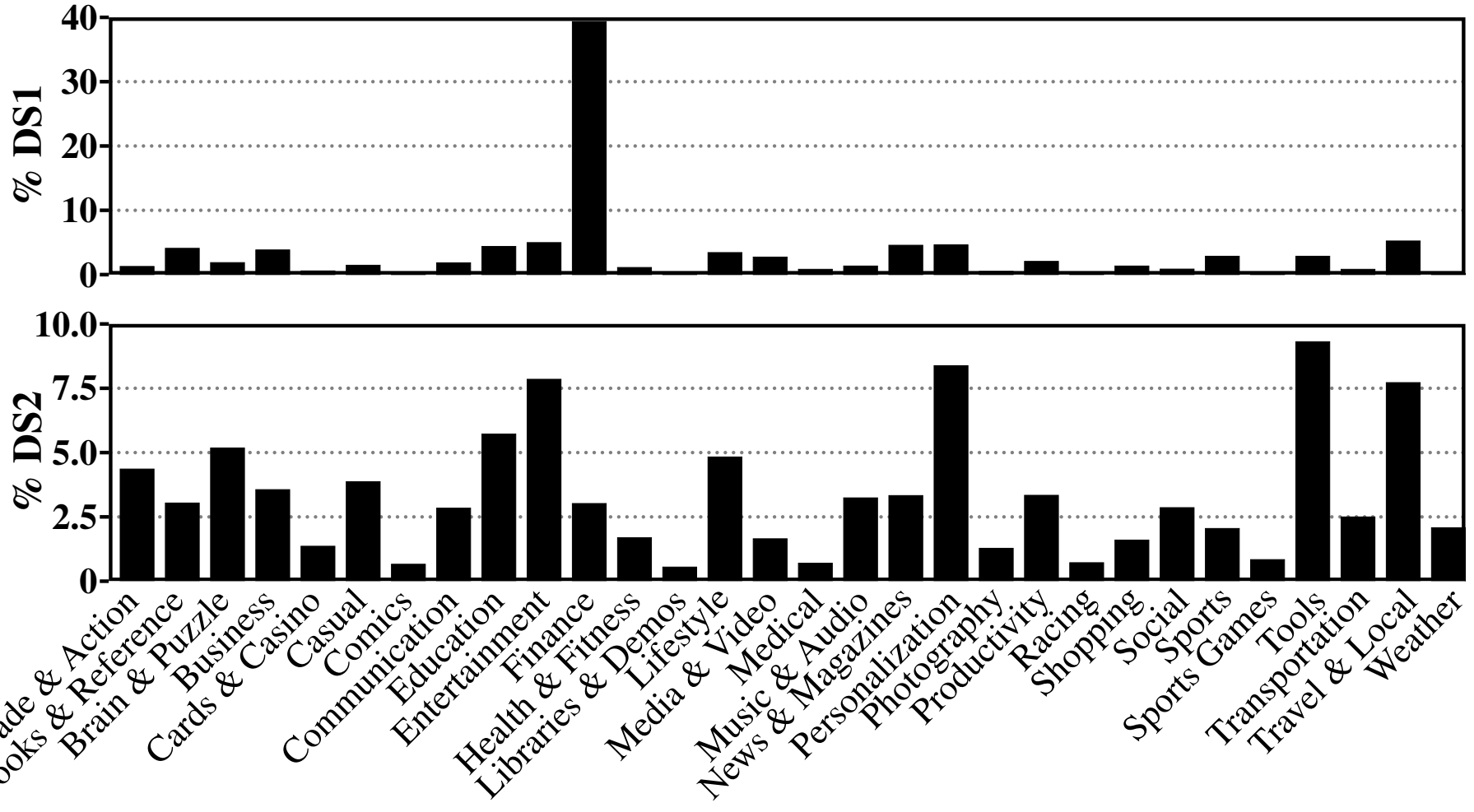
- The MITM proxy only sees network traffic, cannot map successful attacks to vulnerable apps
- The correlative analysis component matches attack timestamps with application installation timestamps
  - Identifies what apps were running during the attack
- DNS query logs are used to identify which app(s) were actually attacked

Data Sets

Two datasets crawled from the Google Play market:

- DS1: 3,165 finance-related apps (using finance-specific query terms)
  - Banking apps more likely to use SSL/TLS
- DS2: 20,316 apps
  - Contains apps with more complex UIs (e.g. games)

Data Set Distributions



Static Analysis

- Time Requirements:
  - Disassembly took 0.42 seconds per app, on average (compared to 276 seconds per app to decompile)
  - Vulnerable Entry Point Identification took 3.63 seconds per app, on average
  - Smart Input Generation took 1.2 seconds per app, on average
- Of 260,395 activities, 8,713 were identified as potentially vulnerable

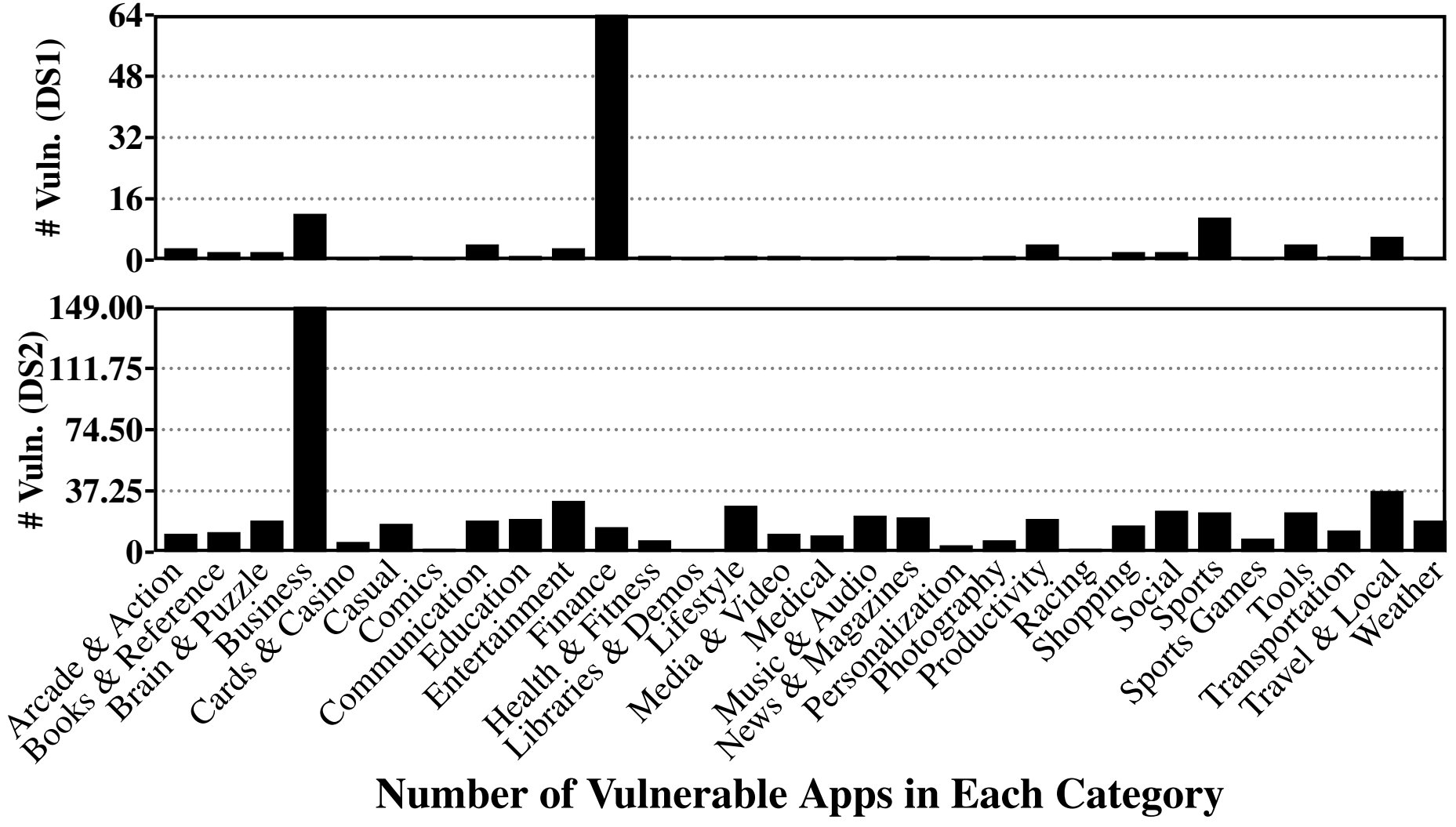
Static Analysis

	DS1	DS2
Vulnerable Apps	221	1322
Vulnerable *Activities*	1670	7043
Disassembly	23.5 minutes	2.4 hours
Entry Point Identification	3.2 hours	20.5 hours
Apps with Detectable Text Fields	87	417
Detected Text Fields	600	5599
Annotated Text Fields	289	3532
Type Casts	92	263
Space Requirements	26G	176G
Smali Files	1.3 million	8.7 million

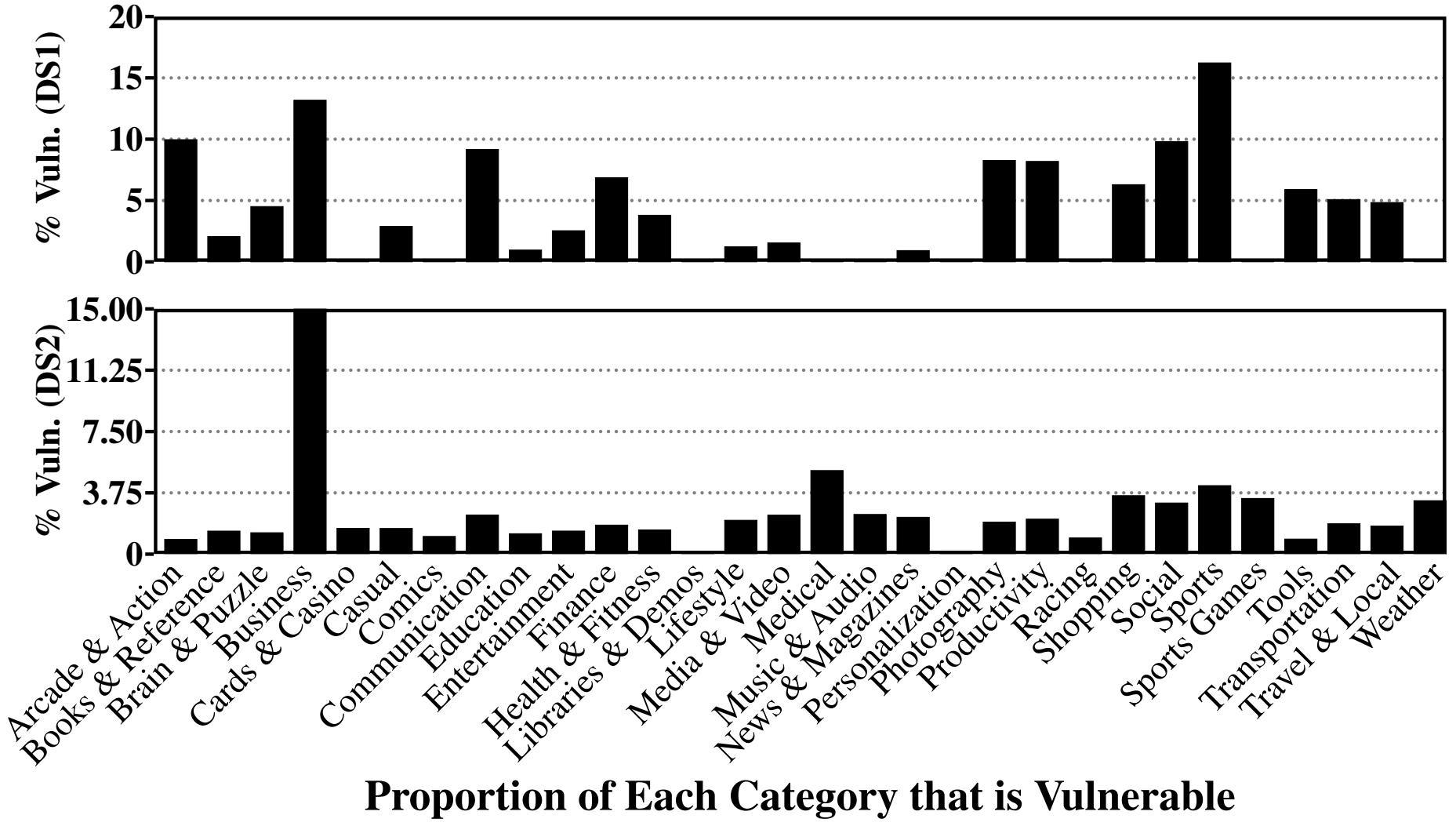
Dynamic Analysis

- Eight emulators running Android OS 4.1 to test the apps in parallel
- The process took 18.81 hours (2.91 for DS1, 15.90 for DS2)
- We recorded 12 emulator crashes, and each emulator crashed or went “offline” at least once
- Of the 8,713 tested entry points, 1,705 crashed on launch
  - more likely in finance category apps, likely because of missing login credentials

Vulnerable Apps



Vulnerable Apps



Vulnerable Apps

- This project was conducted over a one-year window, allowing us to revisit vulnerable apps
- We attempted to re-download all 726 confirmed-vulnerable apps
- 14.6% were unavailable, and 76.17% were still vulnerable

Limitations/Future Work

- The dynamic analysis component can introduce false negatives due to some limitations:
  - Multi-Page input
  - Advanced UI Operations (e.g. swipe, long touch)
  - WebViews: embedded browser components that cannot be analyzed by the `ViewServer`

Conclusion

- Our system combines static and dynamic analysis techniques to perform large-scale, automated SMV detection on Android
- We identified 726 confirmed-vulnerable apps (out of 23,481 apps, approx. 3%)
- Months later, more than $\frac{3}{4}$ were still vulnerable

This material is based upon work supported by The Air Force Office of Scientific Research under Award No. FA-9550-12-1-0077.