

Intent-aware Fuzzing for Android Hardened Application

Seongyun Jeong^{1,2}, Minseong Choi^{1,2}, Haehyun Cho³,
Seokwoo Choi⁴, Hyungsub Kim⁵, Yuseok Jeon^{1,2}

¹ **KOREA**
UNIVERSITY

² **UNIST**

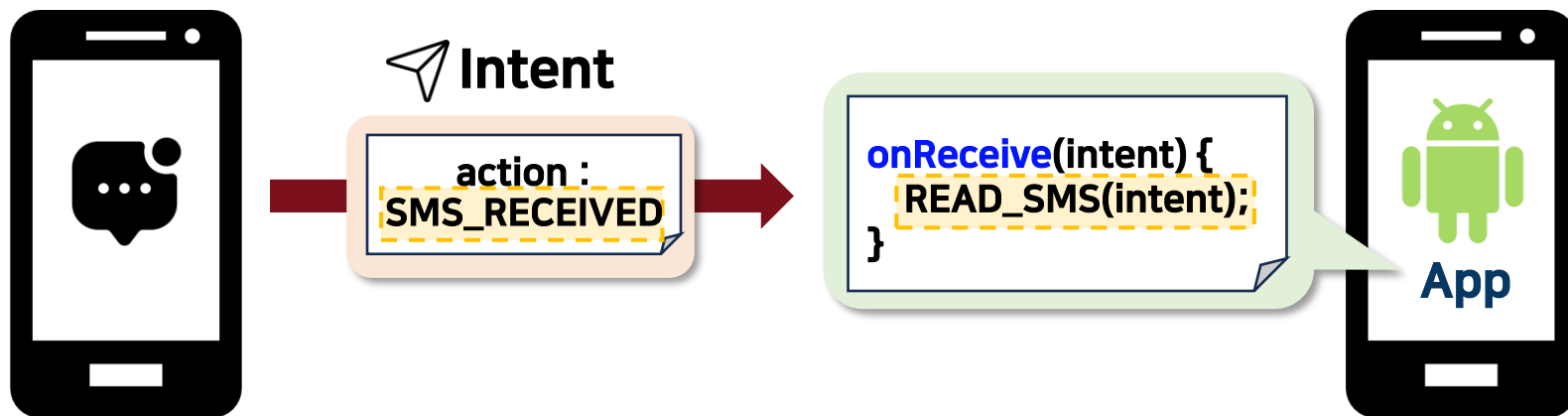
³ 
Soongsil University

⁴ **ETRI**
Electronics and Telecommunications
Research Institute

⁵ 
INDIANA UNIVERSITY

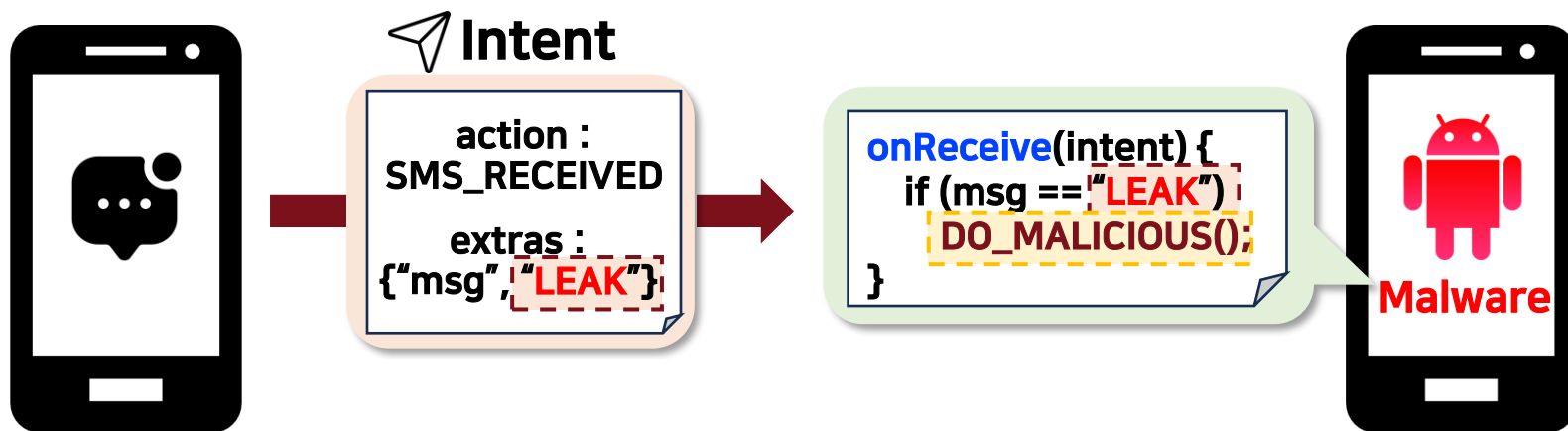
Intent

- ✔ Intent is a message object used for inter-component communication



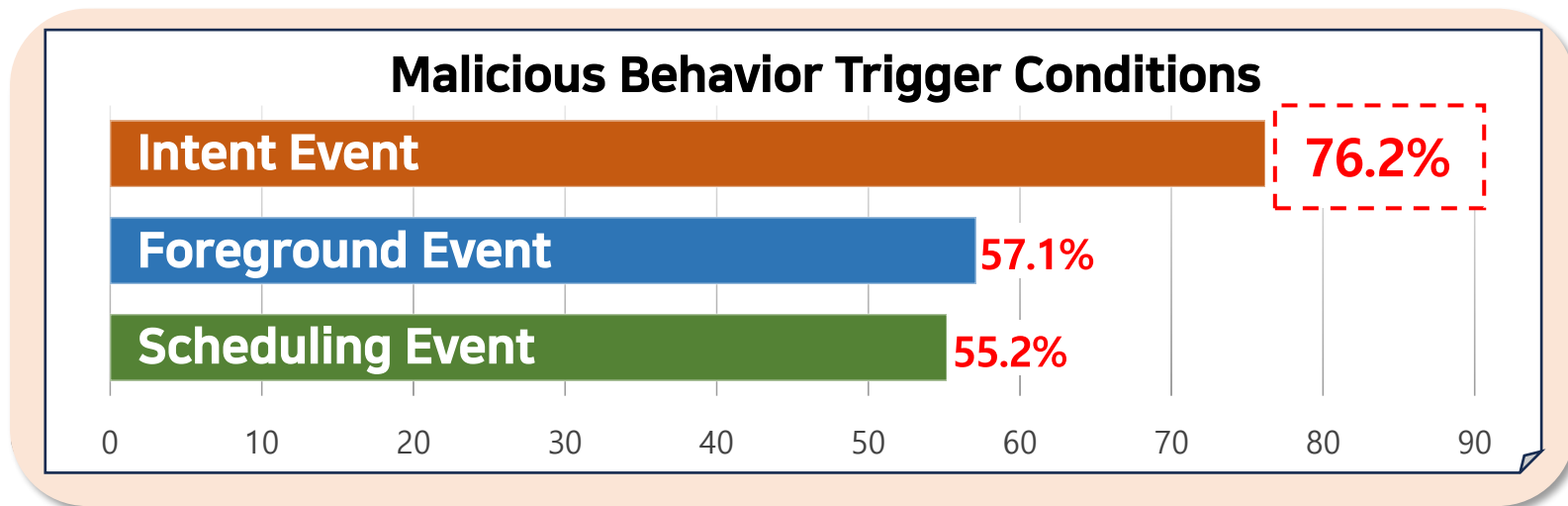
Importance of Intent

- ✓ Intent can contain extra data and **influence an app's execution flow**



Importance of Intent

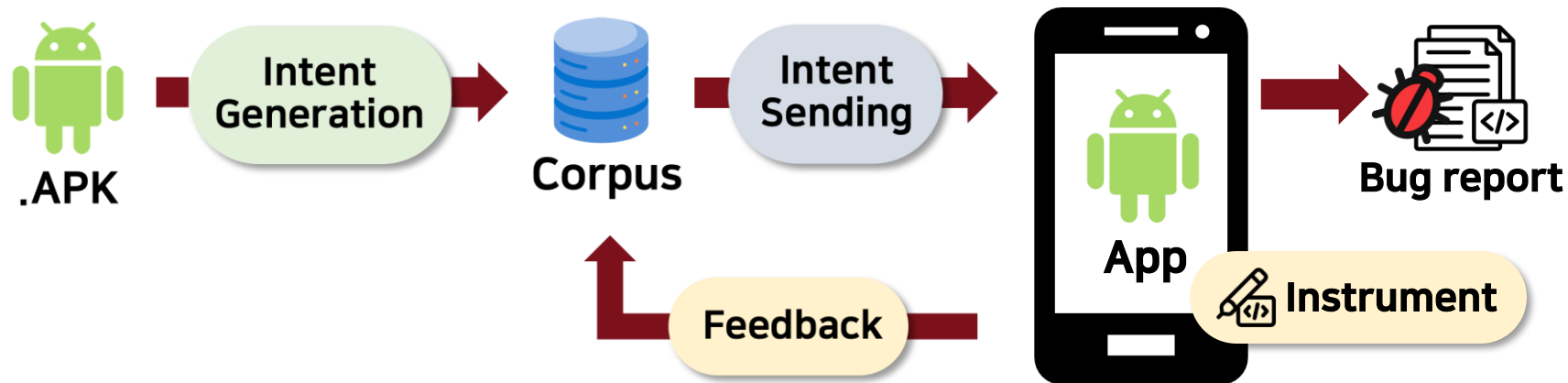
- ✓ Intent can contain extra data and **influence an app's execution flow**



76.2% of malware leverage Intent

Intent Fuzzing

- ✓ For Intent fuzzing, generating **valid, interesting** Intent is important



Intent Fuzzing

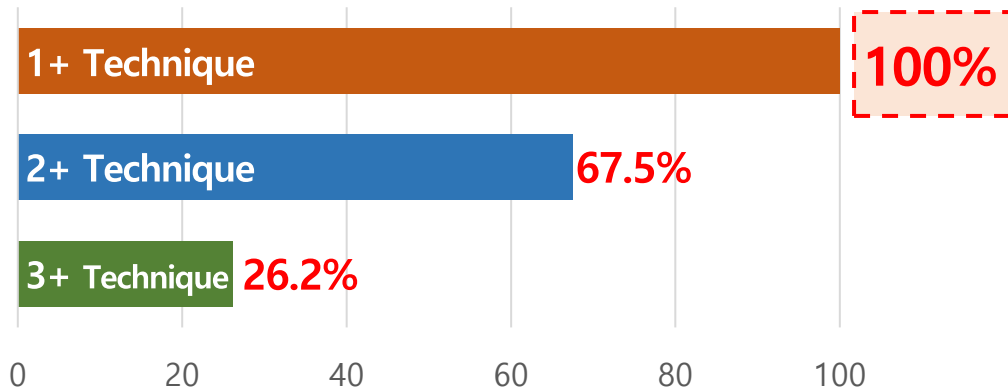
- ✓ For Intent fuzzing, generating **valid, interesting** Intent is important

What happens if the target application uses hardening techniques?

Hardening Techniques

- ✓ **At least one hardening technique is adopted** in most downloaded 200 apps
- ✓ Hardening techniques (e.g., obfuscation, packer) **hinder Intent fuzzing**

Prevalence of App Hardening Techniques



	Benign (%)	Malicious (%)
Obfuscation	39	22
Packer	22	94
Anti-Debugging	74	9
Anti-Emulator	98	33

Hardening Techniques

✓ At least one hardening technique is adopted in most downloaded 200 apps

We should effectively analyze
hardened application through Intent fuzzing

0 20 40 60 80 100

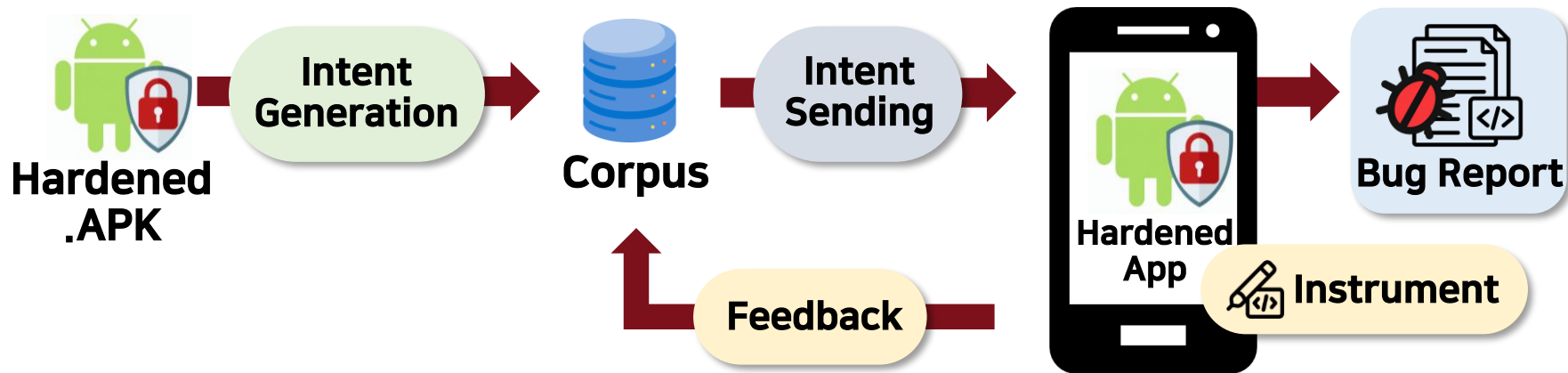
Anti-
Emulator

98

33

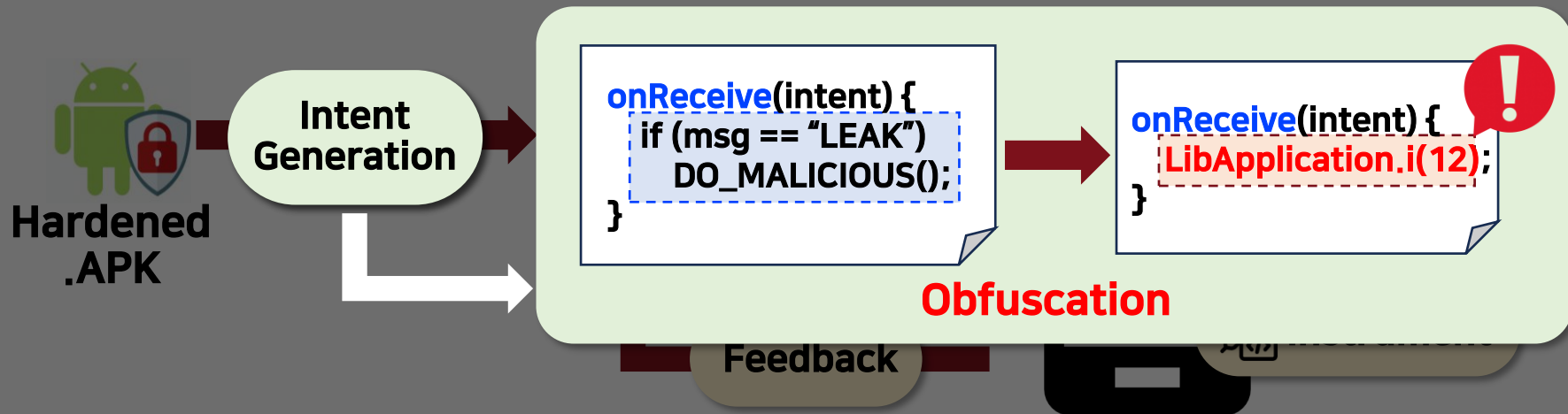
Challenge #1: Valid Intent Generation

- ✓ To generate valid Intent, extracting Intent-related information is needed



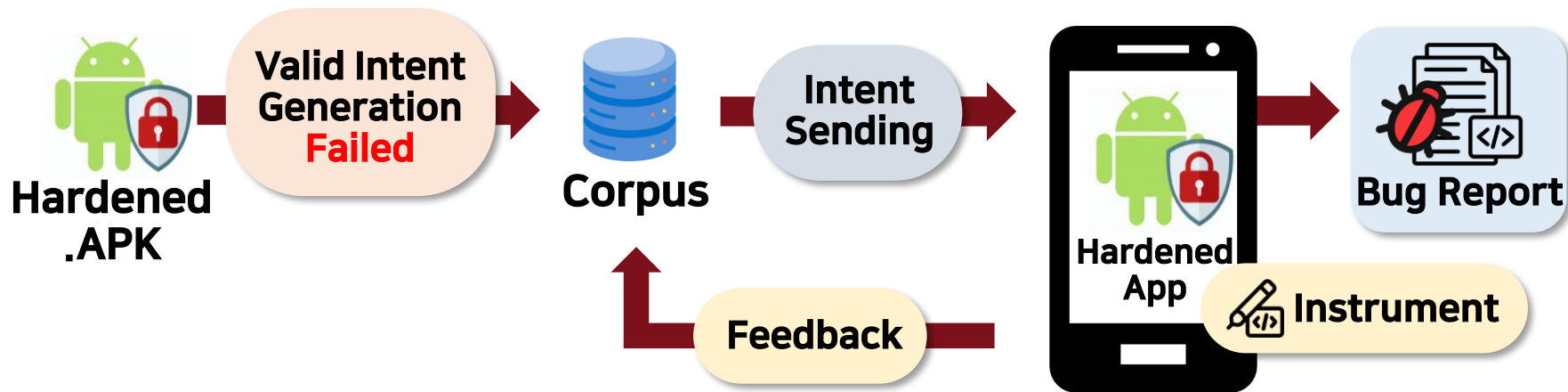
Challenge #1: Valid Intent Generation

- ✓ To generate valid Intent, extracting Intent-related information is needed



Challenge #1: Valid Intent Generation

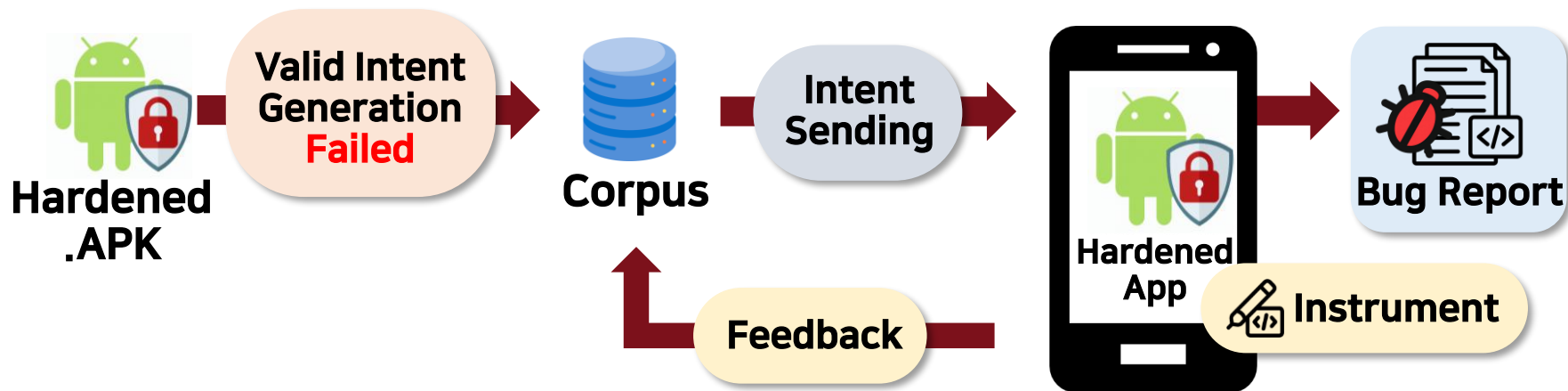
- ✓ To generate valid Intent, extracting Intent-related information is needed



Hardening techniques hinder the extraction of Intent-related information

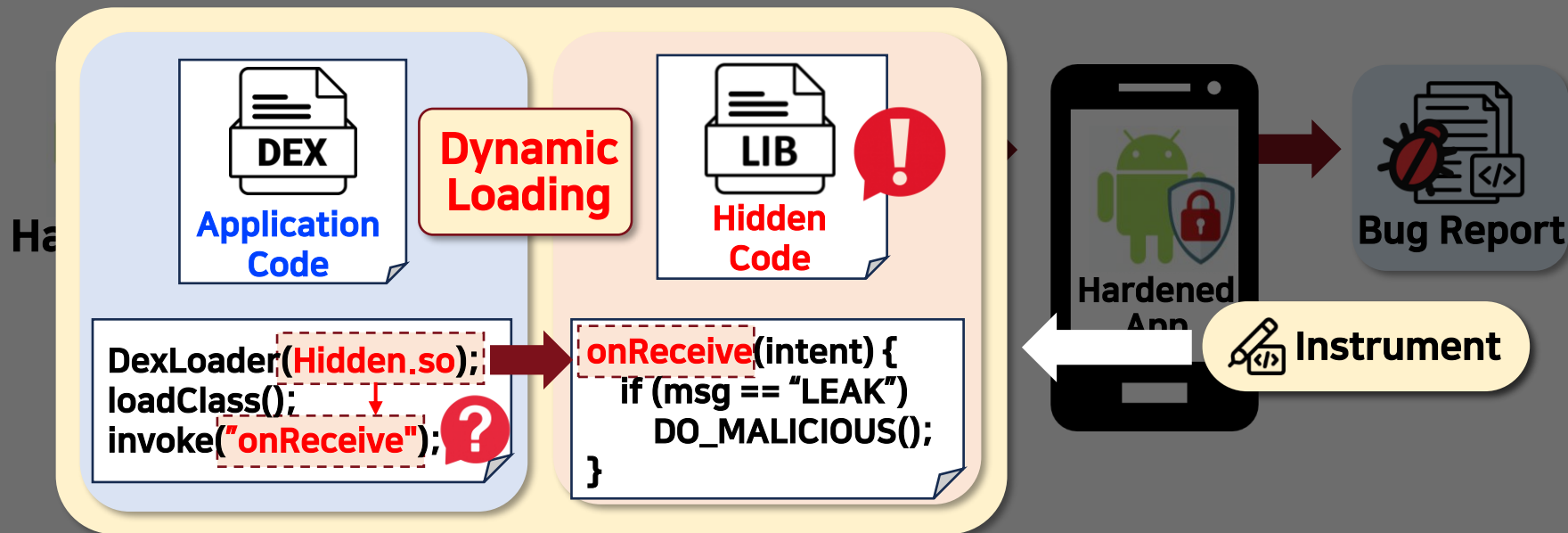
Challenge #2: Coverage Feedback Generation

- ✓ To measure the impact of Intent inputs, coverage feedback is needed



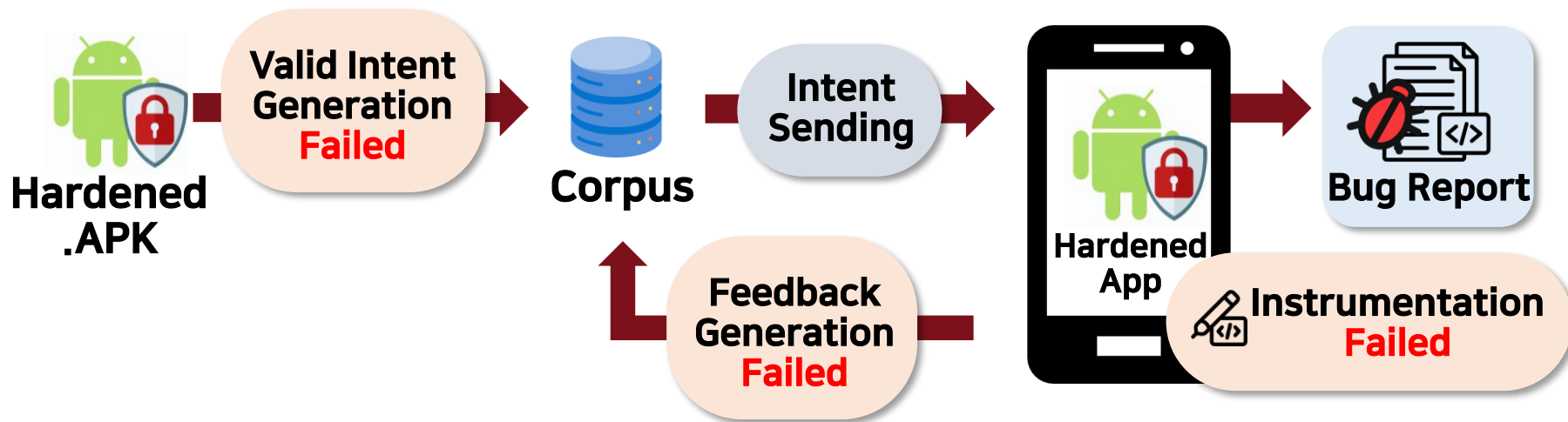
Challenge #2: Coverage Feedback Generation

- ✓ To measure the impact of Intent inputs, coverage feedback is needed



Challenge #2: Coverage Feedback Generation

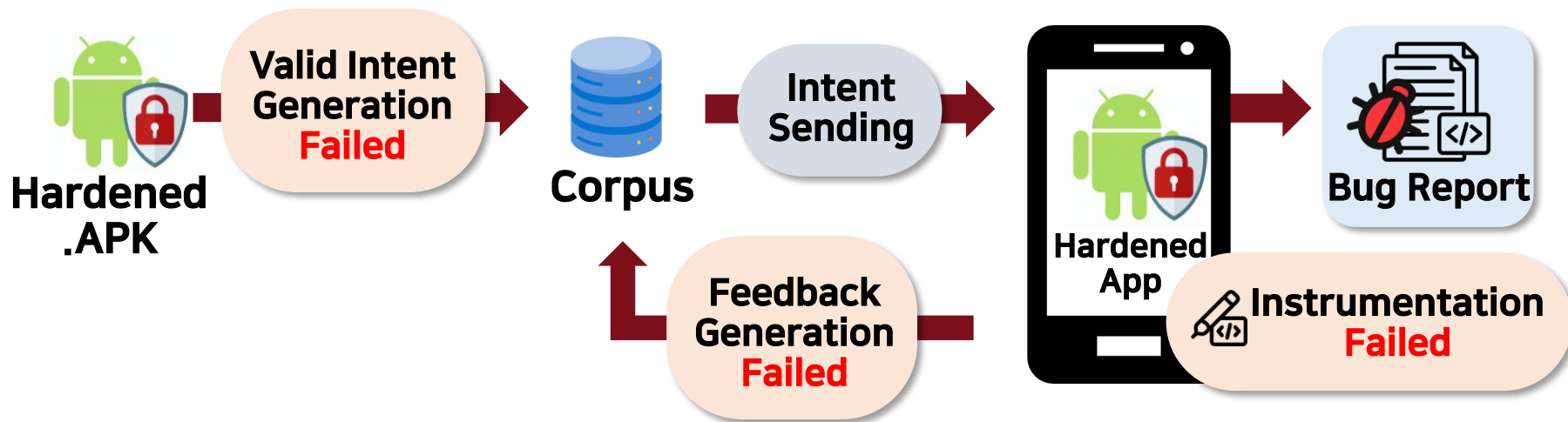
- ✓ To measure the impact of Intent inputs, coverage feedback is needed



Hardening techniques complicate identifying instrumentation target

Challenge #3: Hidden Bugs Triggering & Detecting

- ✓ Even if fuzzer reaches suspicious code, triggering & detecting bugs is needed



Challenge #3: Hidden Bugs Triggering & Detecting

- ✓ Even if fuzzer reaches suspicious code, triggering & detecting bugs is needed

```
onReceive(intent) {  
    String DeviceID;  
    AlarmManager.set(  
        24Hours, !  
        DO_MALICIOUS(DeviceID)  
    );  
}
```

Scheduling

**Trigger Malicious
Behavior
24 Hours Later**



**Instrumentation
Failed**

Challenge #3: Hidden Bugs Triggering & Detecting

- ✓ Even if fuzzer reaches suspicious code, triggering & detecting bugs is needed

```
onReceive(intent) {  
    String DeviceID;  
    AlarmManager.set(  
        24Hours,  
        DO_MALICIOUS(DeviceID)  
    );  
}
```

Scheduling

```
DO_MALICIOUS(ID) {  
    'Log.d(ID)' // Sink point  
}
```

Information Leak



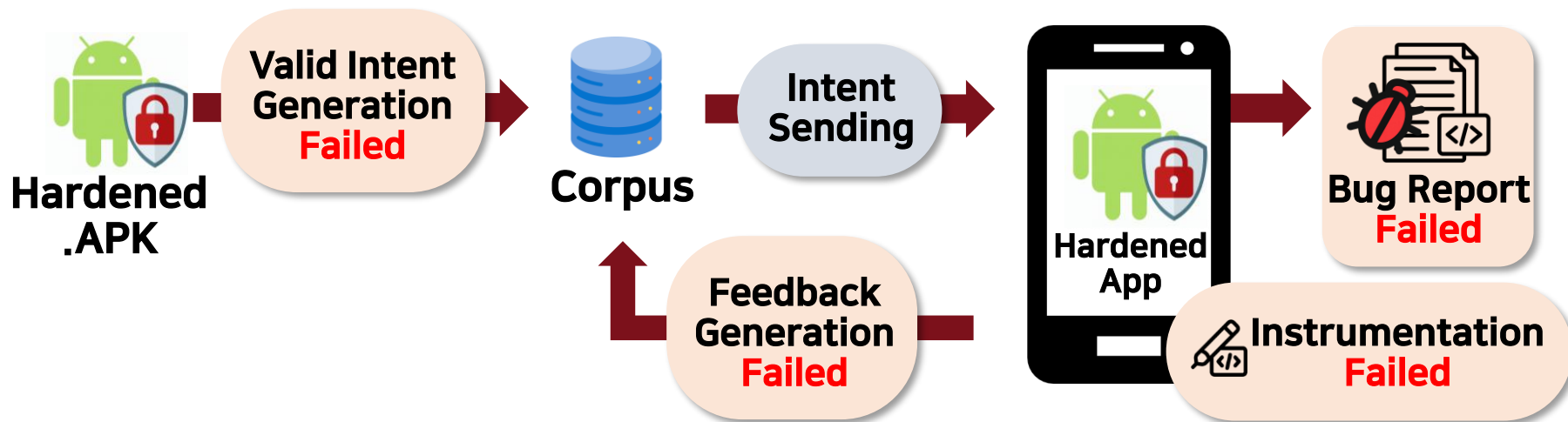
Bug Report



**Instrumentation
Failed**

Challenge #3: Hidden Bugs Triggering & Detecting

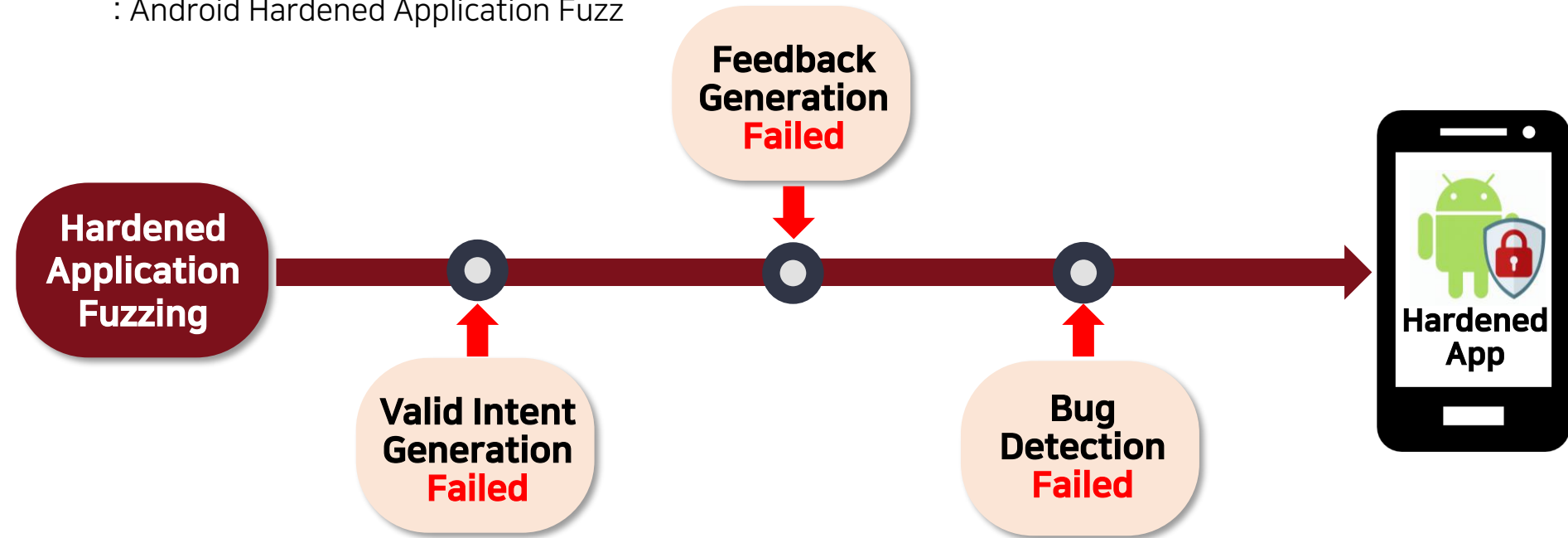
- ✓ Even if fuzzer reaches suspicious code, triggering & detecting bugs is needed



Miss bugs triggered under scheduling, or fail to detect triggered bug

AHA-Fuzz Overview

: Android Hardened Application Fuzz



AHA-Fuzz Overview

: Android Hardened Application Fuzz

Hardened
Application
Fuzzing

Valid
Intent
Generator

Coverage
Feedback
Generator

Bug
Detection
Enhancer

Hardened
App

eBPF



Raw bytes



Object-aware

Action
Key
Value

Mutated
Intent



Scheduling

Time : 6 Hour

Time : 1 Min



Bug detection

- Device ID
- Network
- Debug info



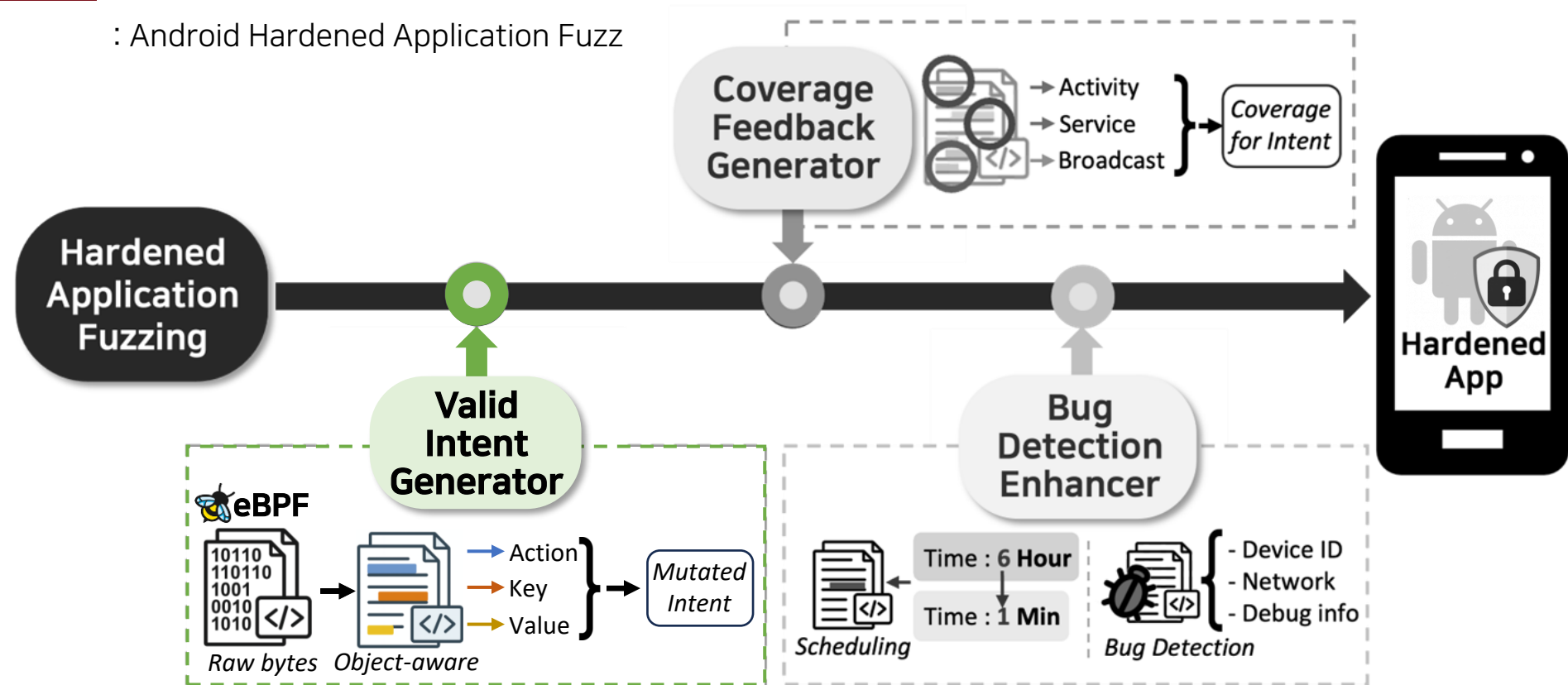
Activity
Service
Broadcast

Coverage
for Intent



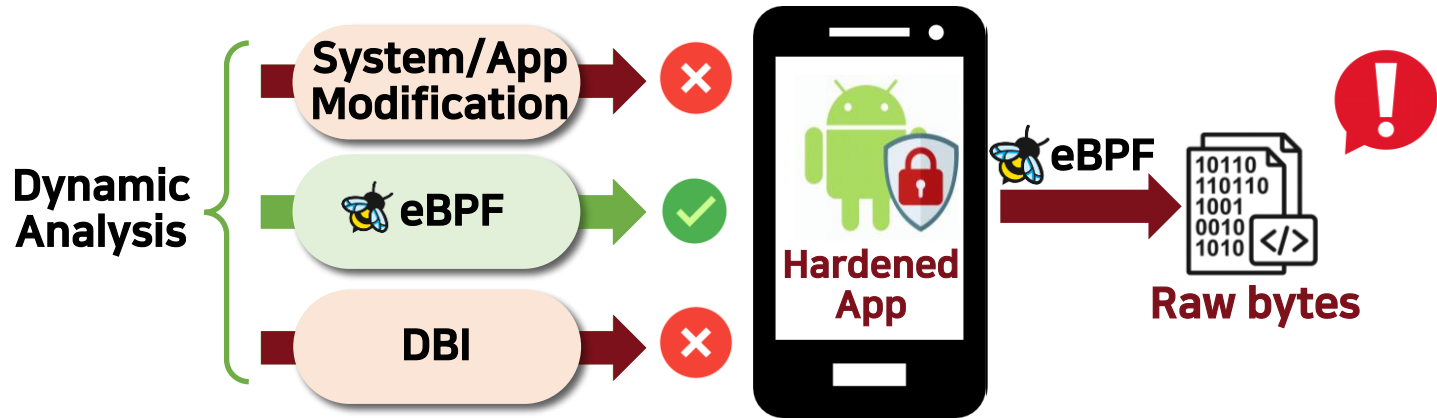
AHA-Fuzz Overview

: Android Hardened Application Fuzz



Extracting Information in Hardened App

- ✓ Step 1. **extract information** at runtime in **hardened environment**
- eBPF enables instrumentation **without interference** from hardening techniques
 - **Limitation** : can **only** access **raw bytes** in memory, making Java objects **hard to observe**



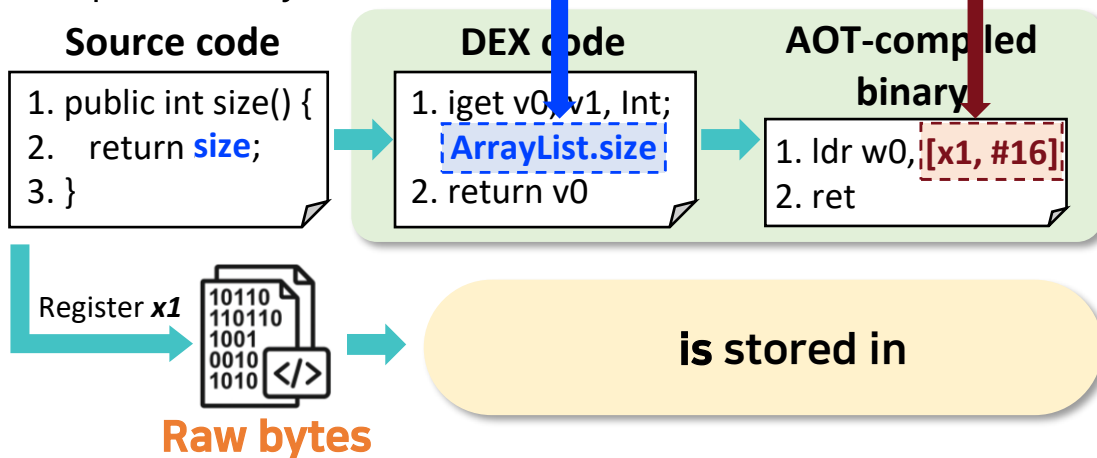
Need to recover Java object layout information

Recovering Java Object Layout

✓ **Solution** : recover layout by combining DEX code and AOT-compiled binary

- **DEX code**: contains **field name**, but **not offset information**
- **AOT-compiled binary** : contains **offset information**, but **not field name**

✓ **Example in ArrayList.size**



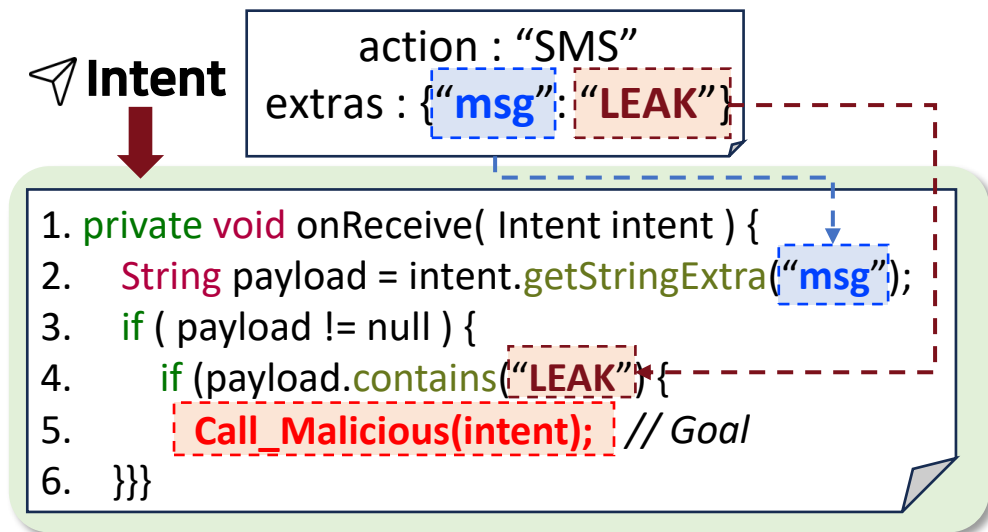
**Recover 82.8% of
Java object layout**



**Information
extraction success**

Valid Intent Generator

- ✓ Step 2. generate **Key-Value** feedback that can trigger deeper code
 - **Key-Value feedback** : generate **Intent metadata** (key-value pair) and guide **input mutation**
 - **Coverage feedback** : measure **input impact** and guide **effective input selection**
- ✓ Example of how key-value pairs are used in Intent



Intent validation
code are stored in
application



Key-Value feedback
is needed

Key-Value Feedback

✓ Hook Intent-related operation & generate feedback

- **Observation** : Intent metadata (extras) are used in **comparison operation**
- **Key feedback** : hooking predefined **extras operation** (e.g., getStringExtra)
- **Value feedback** : hooking **comparison operation** (e.g., contains, equals)

```
1. private void onReceive( Intent intent ) {  
2.   String payload = intent.getStringExtra("msg")  
3.   if ( payload != null ) {  
4.     if (payload.contains("LEAK"))  
5.       Call_Malicious();  
6.   }  
}
```

Key
Feedback

"msg"

Value
Feedback

"LEAK"



Action
Key
Value

Intent

```
action : "SMS"  
extras : {  
    :  
}
```

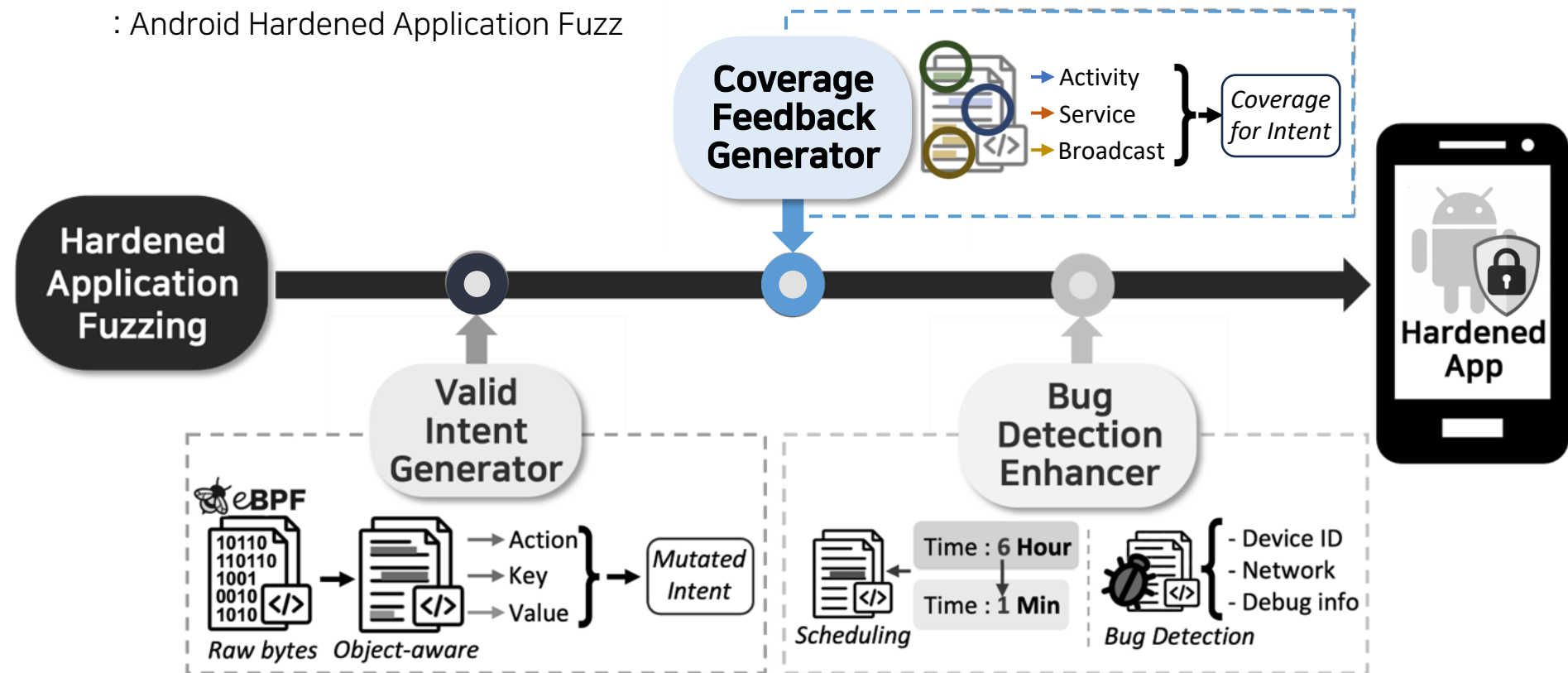
Cover 93.5%
extras values



Generate
valid Intent

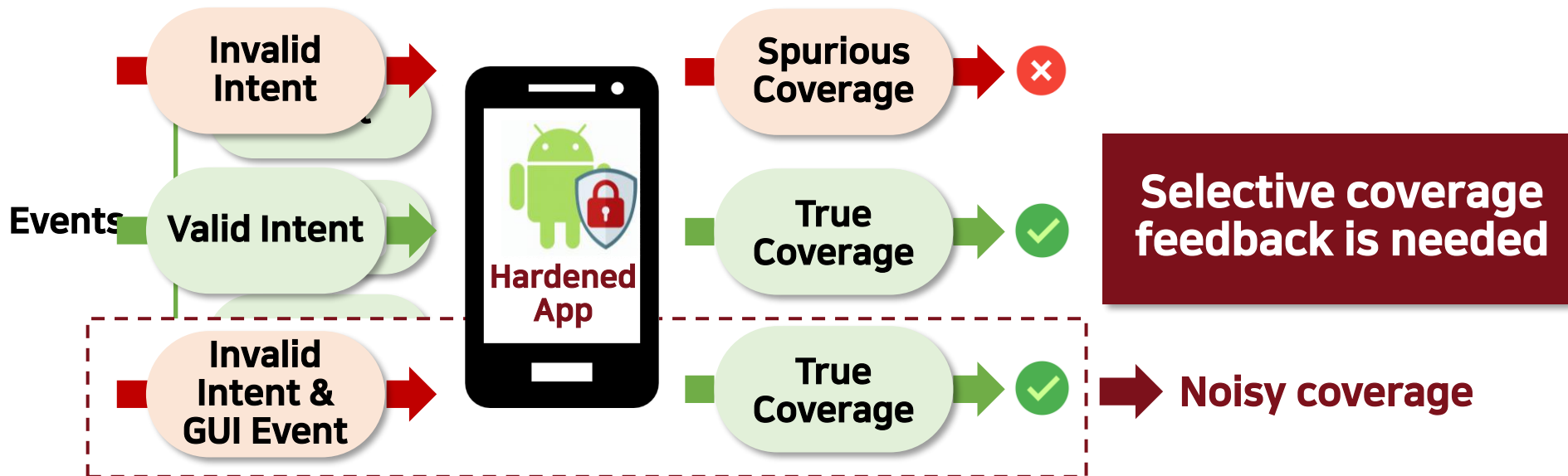
AHA-Fuzz Overview

: Android Hardened Application Fuzz



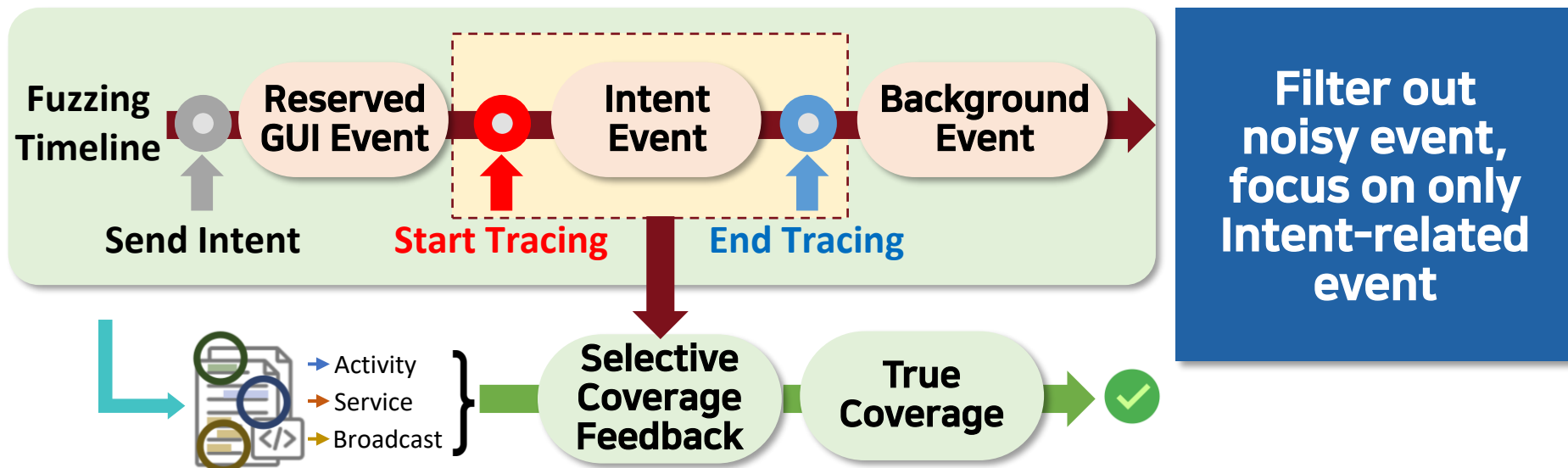
Coverage Feedback Generator

- ✓ For Intent coverage feedback, Intent-specific feedback is needed
 - Noisy coverage (e.g., GUI event) can **negatively impact** Intent fuzzing



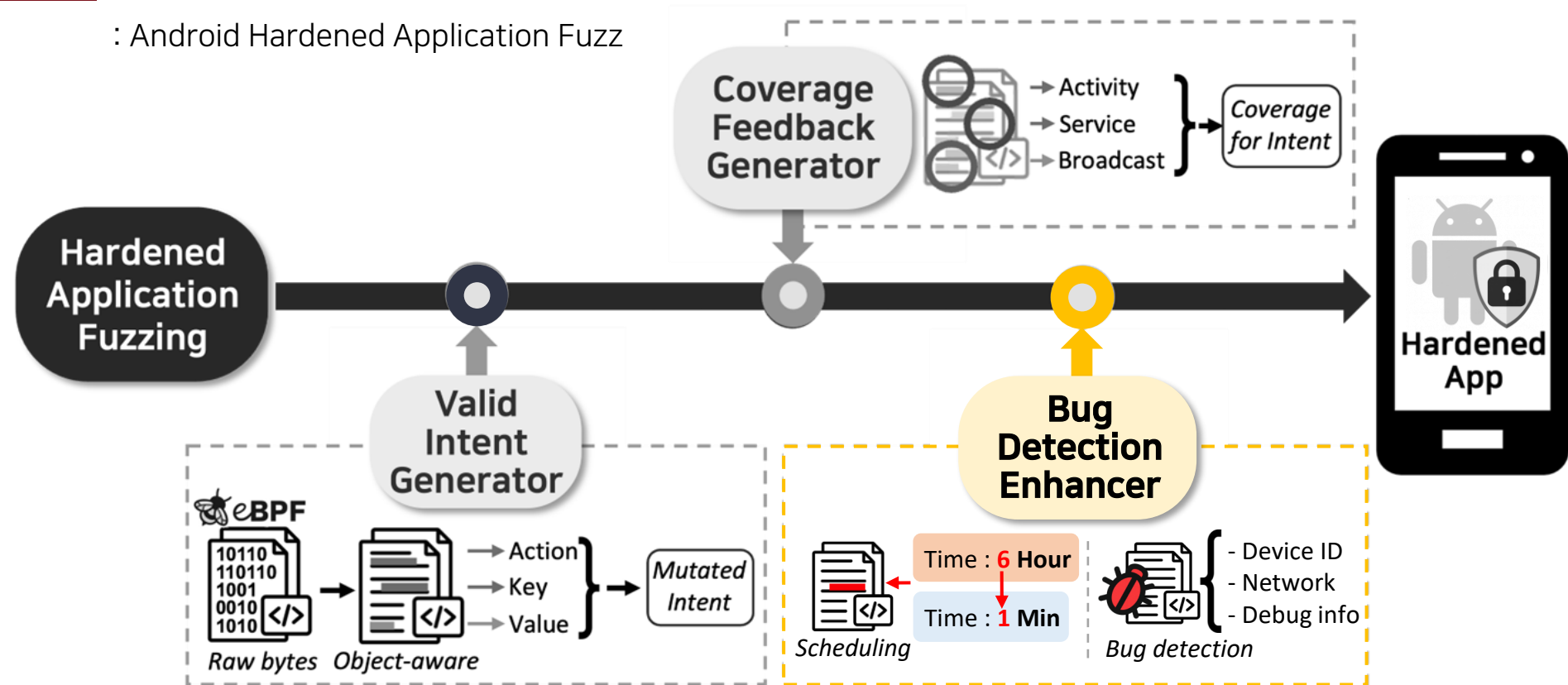
Selective Coverage Feedback

- ✓ Tracing **individual** Intent event execution flow, **generate selective feedback**
 - **Observation** : can **define** Intent execution pattern (e.g., start & end point)



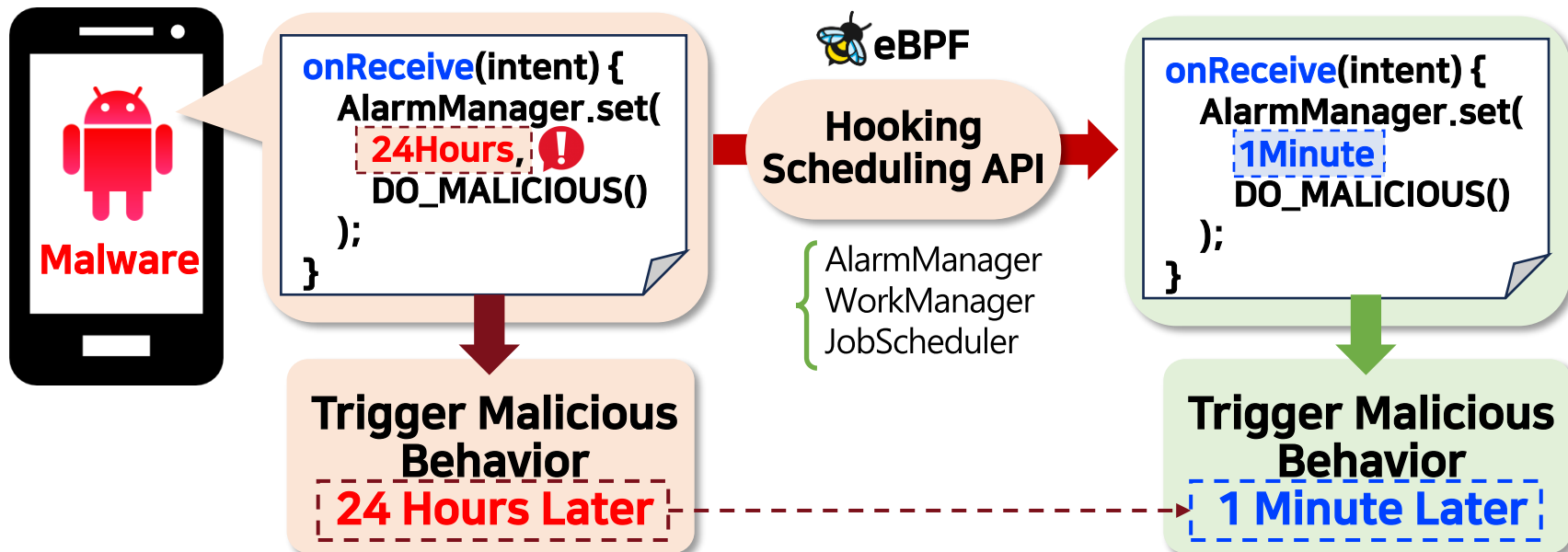
AHA-Fuzz Overview

: Android Hardened Application Fuzz



Triggering Scheduling Event

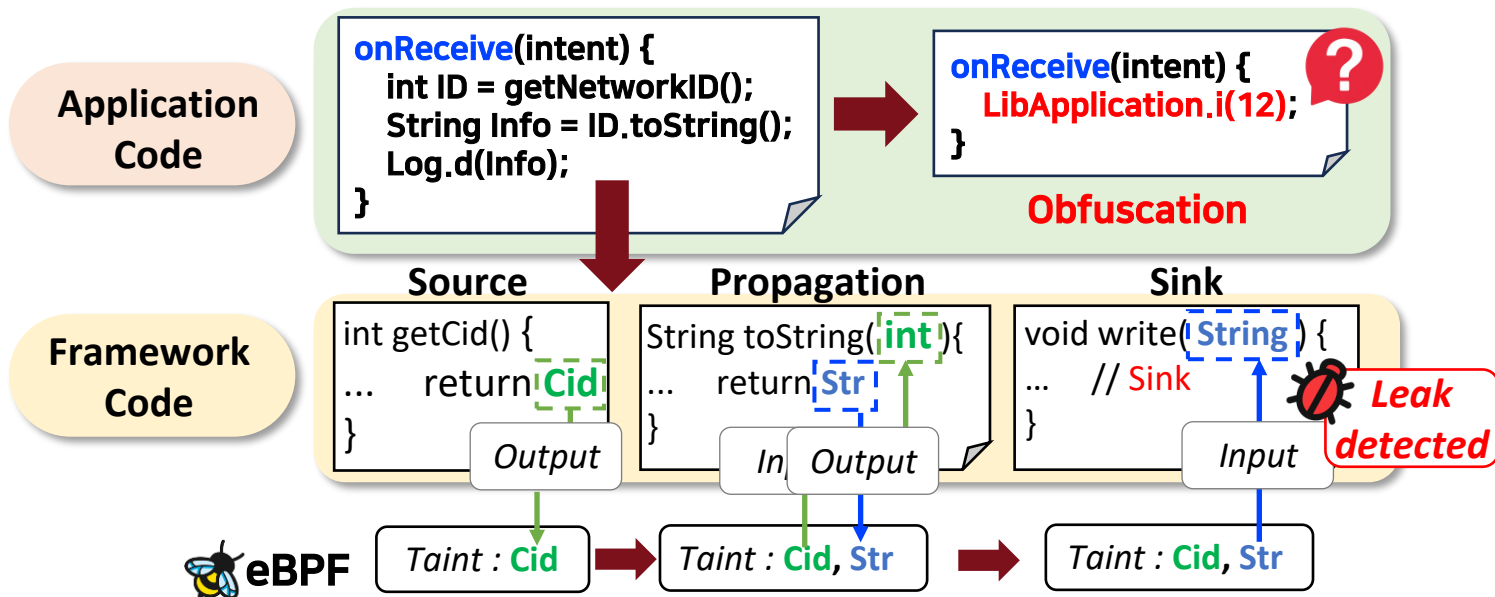
- ✓ Handle scheduling event : **overwrite scheduling time** for fast-forward invoking
 - Hooking all scheduler-related APIs, adjust the associated scheduling times



Detecting Information Leak

✓ **Detect information leak** : lightweight method-level taint analysis

- **Tracking** primitive type values pass through **input** & **output**
- **Observation** : **sensitive information** typically stored and manage as **primitive type**



Evaluation

AHA-Fuzz Evaluation {

- ✓ RQ #1 : Effectiveness of fuzzing
- ✓ RQ #2 : Code Coverage
- ✓ RQ #3 : Bug Detection

}

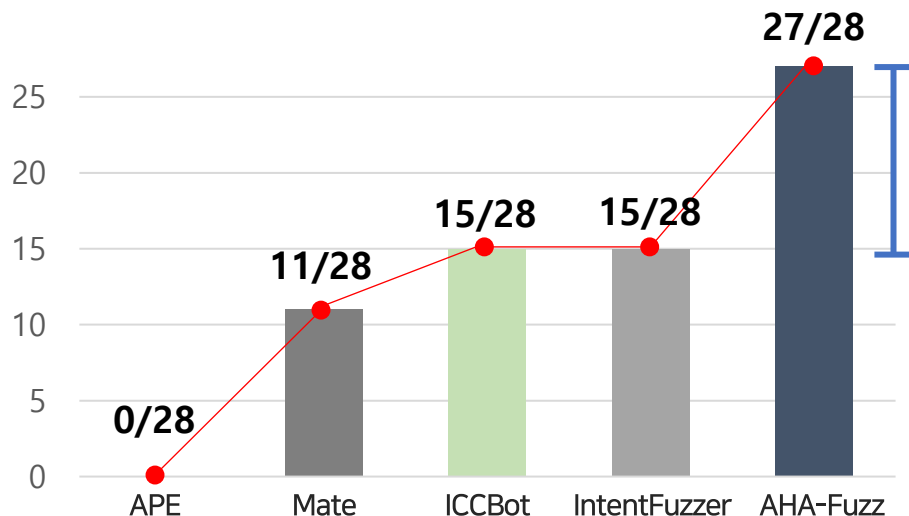
✓ Baseline Fuzzers

- **APE** (ICSE'19) : **GUI** Fuzzer
- **Mate** (ICST'23) : GUI + Intent fuzzer with **intra-procedural** analysis
- **ICCBot** (ICSE'22) : Intent fuzzer with **inter-procedural** analysis
- **IntentFuzzer** (AsiaCCS'14) : Intent fuzzer with **hybrid** analysis
- **AHA-Fuzz** : **GUI** + Intent fuzzer with **key-value & coverage feedback**

Effectiveness of Intent Fuzzing

✓ Evaluate how Intent fuzzer can create valid intents used in malware

- Target : Among 105 Google Play malware, find 13 reproducible apps
- Select **28** malicious behavior triggered by Intent event
- **AHA-Fuzz trigger all malicious behavior except for one**



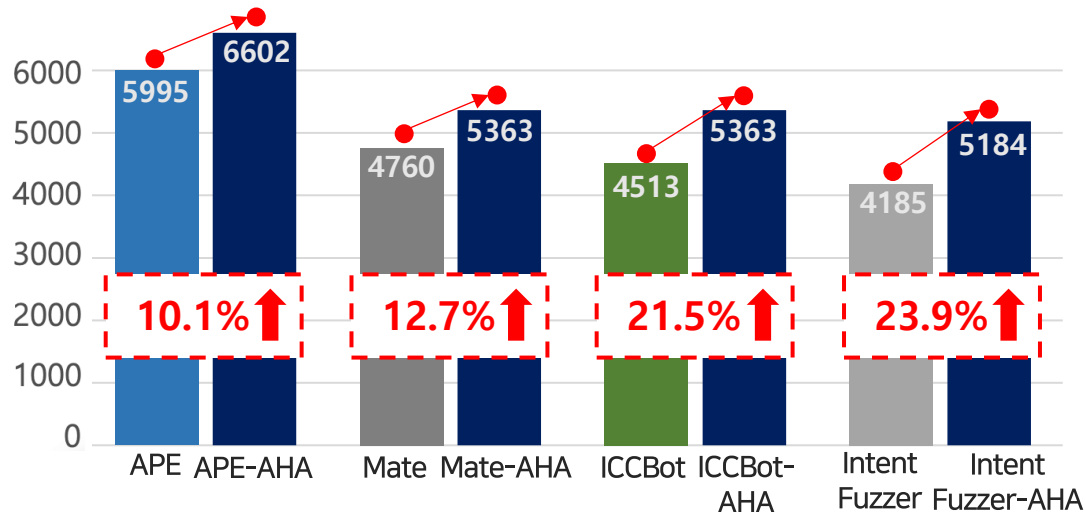
✓ Invoked behavior

**Trigger 92.3%
more Intent events**

Code Coverage

✓ Evaluate method coverage increase in benign/malicious apps

- Target : Top downloaded 20 benign / 20 malicious apps
- **AHA-Fuzz trigger 23.9% more method than previous approach**



✓ Average of invoked method both benign/malicious apps

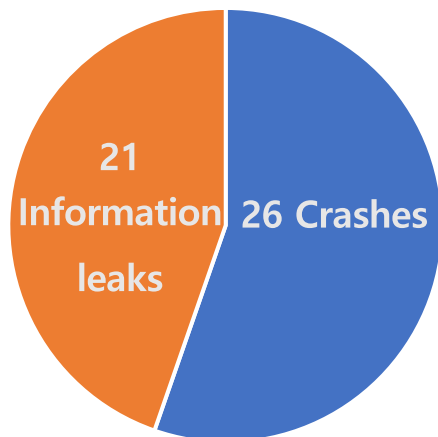
**Trigger 23.9%
more method**

- -AHA : Variant version of each fuzzer, replace each Intent fuzzer to AHA-Fuzz algorithm

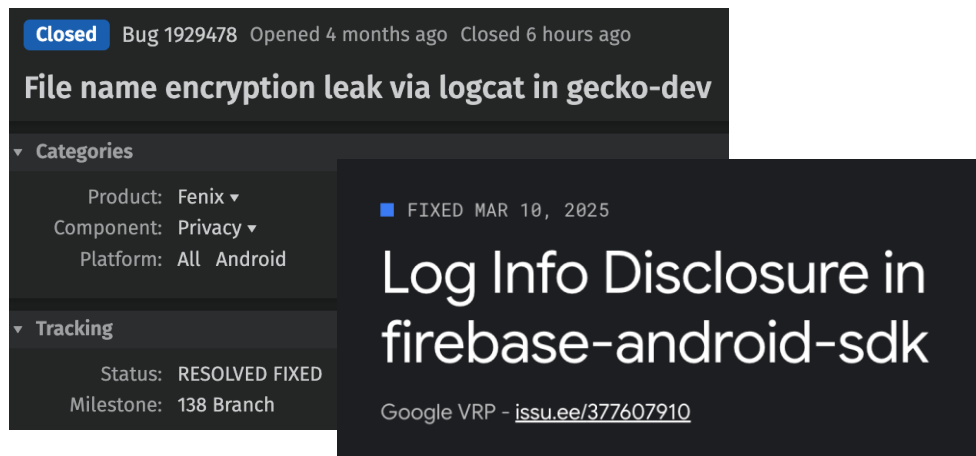
Discovering Unknown Bugs

✓ Evaluate AHA-Fuzz can discover previously-unknown bugs

- Target : Top downloaded 300 google play apps
- AHA-Fuzz discover **47 unknown bugs** that previous works cannot detect
- **Google, Firefox, and Facebook have acknowledged 6 bugs, 3 of them fixed**



✓ Total 47 bugs



✓ Bug fixed in Firefox, Google SDK

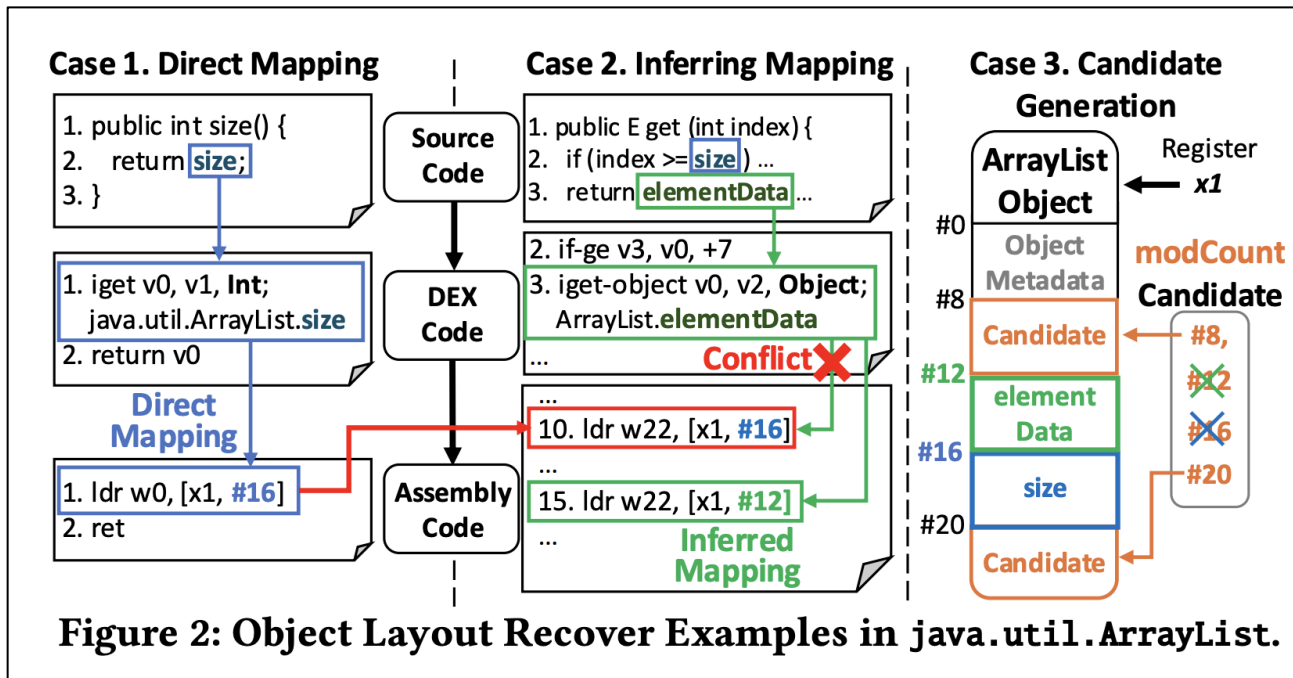
Conclusion

- ✓ **Introduce AHA-Fuzz, the first Intent-aware greybox fuzzer for hardened android app**
 - Valid Intent generator by recovering object layouts and leveraging Key-Value feedback
 - Selective coverage feedback for effective fuzzing
 - Triggering hard-to-trigger events, increase bug detection capability
- ✓ **AHA-Fuzz shows better performance compared to previous works**
 - Trigger 92.3% more Intent patterns in malware (3.45X faster)
 - Invoke 23.9% more methods
 - Find 47 previously unknown bugs that previous works cannot find
 - 6 bugs have been acknowledged by Google, Firefox and Facebook, and 3 have been fixed



<https://github.com/S2-Lab/AHA-fuzz>

Appendix : Recovering Java Object



Appendix : Recovering Java Object Result

Table 3: Java object layout recovery result. Cases 1, 2, and 3 correspond to direct mapping, inferring mapping, and candidate generation, respectively.

Module name	Classes	Fields	Case 1 (%)	Case 2 (%)	Case 3 (%)
content	982	1,729	669 (38.7%)	681 (39.4%)	379 (21.9%)
app	1,559	3,465	1,523 (44.0%)	1,428 (41.2%)	514 (14.8%)
Total	2,541	5,194	2,192 (42.2%)	2,109 (40.6%)	893 (17.2%)

Appendix : Extras Mutation

Table 4: Type ratio analysis of extras values used in 300 most downloaded apps.

Type category (#)	Mutation Strategy	# of unique extras values (%)
Primitive (8)	AFL-based	9,317 (44.2%)
String (1)	Key-Value feedback	10,395 (49.3%)
Special (5)	Invoke constructor	1,369 (6.5%)
Total (14)	-	21,081 (100%)

Appendix : Method-Level Taint Propagation

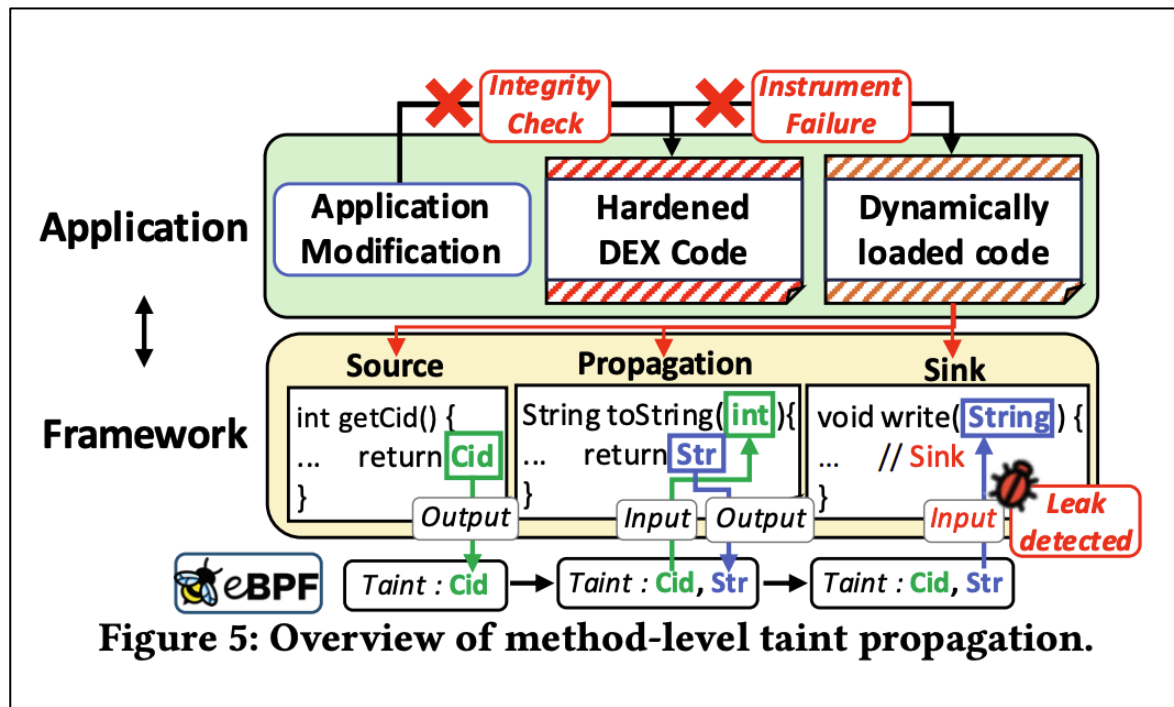


Figure 5: Overview of method-level taint propagation.

Appendix : Evaluation #1

Table 3: Intent usage patterns on 14 malicious apps. 'Improved' column denotes improvements achieved by AHA-Fuzz in the terms of patterns and times to find them, compared with the others. ✓denotes patterns invoked by AHA-Fuzz. Xexpresses that AHA-Fuzz fails to invoke the patterns. 'Time' represents improved time to invoke a pattern compared with the most fast one that invokes the same pattern.

Family	Patterns	Ape (sec)	Mate (sec)	ICCBot (sec)	IntentFuzzer (sec)	AHA-Fuzz (sec)	Improved Patterns	Time (X)	Pattern Details
AceCard	M	-	235	108	115	88	-	1.23×	E : SMS, Payload
	E	-	-	-	-	89	✓	-	
AgentBKY	E	-	-	-	-	30	✓	-	E : SMS
	E	-	-	121	136	89	-	1.36×	E : Google Firebase
Bahamut	M	-	1251	647	829	38	-	17.03×	M : Multiple Entrypoints
	E	-	-	-	-	44	✓	-	E : SMS
ClickerGenG	M	-	283	171	140	74	-	1.89×	S : Run After 6 hours
	S	-	-	-	-	741	✓	-	
HiddenAdOS	M	-	643	322	464	86	-	3.74×	E : Google Firebase
	E	-	-	279	501	172	-	1.61×	
IndexY	E	-	-	-	-	55	✓	-	E : SMS, Multiple Entrypoints
ProjectSpyHRX	M	-	192	44	50	35	-	1.26×	E : SMS
	E	-	-	-	-	30	✓	-	E : Malware developer defined
	E	-	-	-	-	-	X	-	
Reputation1_2019	M	-	281	141	101	32	-	3.15×	E : Dynamically registered, developer defined
	M	-	306	87	128	42	-	3.05×	
	E	-	-	201	133	43	-	3.09×	
SMSAndroidOSWesp	E	-	-	-	-	52	✓	-	E : SMS, Payload
Solid	M	-	174	61	55	50	-	1.1×	E : Payload, Google Firebase
	E	-	-	-	-	393	✓	-	
Sonyvpay	M	-	420	112	128	33	-	3.40×	E : SMS, Payload
	E	-	-	-	-	40	✓	-	
SpyBankerHU	M	-	301	104	114	31	-	3.35×	E : SMS
	E	-	-	-	-	43	✓	-	
TrojanDropperAgentCIQ	S	-	-	-	-	1394	✓	-	S : Run after 24 hours
Vilny	M	-	362	98	109	32	-	3.06×	S : Run after 26 hours
	M	-	265	104	99	33	-	-	
	S	-	-	-	-	156	✓	-	
Total (%)		0/28 (0.0%)	11/28 (39.3%)	15/28 (53.4%)	15/28 (53.4%)	27/28 (96.4%)	92.3%	3.45×	

Appendix : Evaluation #1 Failure Case

```
1 // ProjectSpyHRX
2 public void onReceive(Context context, Intent intent) {
3     String target = intent.getStringExtra("package");
4     int hashed_target = target.hashCode();
5     if (hashed_target == 7696199939) {
6         target = "com.whatsapp";
7         Do_malicious(target);
8     }
9 }
```

Figure 5: AHA-Fuzz's Failure Pattern on ProjectSpyHRX.

Appendix : Ablation Study

Table 4: Ablation Study Results. Improve(%) indicates the performance improvement rate compared to the previous mode (upper low)

Fuzzer	# of invoked pattern	Avg. TTE (Improve(%))
IntentFuzzer	15	206
AHA-KV	27	146 (41.1%)
AHA-CV	27	124 (17.7%)
AHA-Fuzz	27	66 (87.9%)

Appendix : Evaluation #3

Table 5: Measured method-level coverage on 40 apps. Mate can only successfully fuzz 11 benign apps out of 20, and 0 malware apps out of 20. The detailed results for each app are presented in [Table 8](#) (in Appendix).

Dataset	Triggered Method (Incr.(%))							
	Ape	AHA-Fuzz	Mate	Mate-AHA	ICCBot	ICCBot-AHA	IntentFuzzer	IntentFuzzer-AHA
Benign (20)	6,880	7734 (12.4%)	4,761	5,363 (12.7%)	6,021	6,979 (15.9%)	6,010	6,804 (13.2%)
Malware (20)	5,110	5,470 (7.0%)	Failed	Failed	3,006	3,988 (32.6%)	2,360	3,565 (51.0%)
Total (Avg)	5,995.1	6,602.1 (10.1%)	4,760.7	5363.0 (12.7%)	4,513.7	5,483.3 (21.5%)	4,185.0	5,184.5 (23.9%)

Appendix : Evaluation #4

Table 6: 47 Previously Unknown Bugs Discovered by AHA-Fuzz.

Oracle	Bug pattern	Package	Number
Crash	IllegalArgumentException	Gmail + 13 more	14
Crash	ClassCastException	Google Home + 7 more	8
Crash	NullPointerException	Roblox + 4 more	4
Leak	Network	Android-sdk + 5 more	6
Leak	Authenticaition code	Android-sdk	1
Leak	Device id	Facebook	1
Leak	App installation path	Firefox + 12 more	13

Appendix : Existing Dynamic Analysis Approaches

	Analysis method	Can analyze hardened apps	Can analyze without modification	Acceptable Overhead	Can observe Java events
1	App modification	X (Obfuscation)	X (Integrity Check)	O	O
2	System modification	O	X (Version Dependence)	O	O
3	DBI	X (Anti-Debugging)	O	X (~10x)	O
4	eBPF	O	O	O	X (Access raw bytes)

Appendix : Recovering Java Object Layout

✓ Recover layout by comparing DEX code and AOT-compiled Binary

- **Direct Mapping** : Find 1-to-1 Mapping in DEX and Assembly
- **Inferring Mapping** : Find Conflict with Direct Mapping

✓ **Inferring Mapping** Example in java.util.ArrayList

