FANS: Fuzzing Android Native System Services via Automated Interface Analysis

Baozheng Liu$^{1,2}$, Chao Zhang$^{1,2}$, Guang Gong$^3$, Yishun Zeng$^{1,2}$, Haifeng Ruan$^4$, Jianwei Zhuge$^{1,2}$

$^1$Institute of Network Science and Cyberspace, Tsinghua University
$^2$Beijing National Research Center for Information Science and Technology
$^3$Alpha Lab, 360 Internet Security Center
$^4$Department of Computer Science and Technology, Tsinghua University
Background

- Android native system services provide many fundamental functionalities
- Meanwhile, they are attractive to attackers
- However, to the best of our knowledge, existing researches paid little attention to them
Related work

- Gong\textsuperscript{[1]} mainly finds system services vulnerabilities \textbf{manually}.
- BinderCracker\textsuperscript{[2]} captures the input model through \textbf{app traffic}.
  - Fuzz system services by mutating the traffic.
- Chizpurfle\textsuperscript{[3]} focuses on the vendor-implemented \textbf{Java services}.

\textsuperscript{[1]} Guang Gong. Fuzzing android system services by binder call to escalate privilege. BlackHat USA, 2015.
\textsuperscript{[2]} Huan Feng and Kang G. Shin. Understanding and defending the Binder attack surface in Android. ACSAC, 2016.
Application-Service Communication Model

Application

Service

Service Manager

Register service, e.g., MediaExtractor Service

Request service, e.g., MediaExtractor Service

Return the service interface, i.e., a top-level interface

Request a multi-level interface, e.g., IMediaExtractor, or call other transactions

Return the interface, or the transaction results
Challenges

- **C1. Multi-Level Interface Recognition**
  - Collect all Interfaces
  - Identify multi-level interfaces

- **C2. Interface Model Extraction**
  - Collect all of the possible transactions
  - Extract the input and output variables in the transactions

- **C3. Semantically-correct Input Generation**
  - Variable name and variable type
  - Variable dependency
  - Interface dependency
Overview

Interface Collector
- Source
- Compile
- status_t XXX::onTransact(…)
- IMediaExtractorService
- IMediaExtractor
- IDataSource
- …

Interface Model Extractor
- Abstract Syntax Tree
- onTransact
- tx1
- data
- pid
- …
- if stmt
- …
- num
- 4996
- writeInt32
- return
- reply
- BAD_VALUE
- ERROR

Dependency Inferer
- Transaction Dependency Graph
- IMediaExtractor-Service
- IMediaExtractor
- IDataSource
- Interface Dependency Graph

Fuzzer Engine
- Fuzzer Manager
  - push corpus
  - push fuzzer
  - pull logs
  - store logs
  - fetch corpus
- Corpus Database
- Crash Log
Interface Collector

Interface Collector

Source

Compile

status_t XXX::onTransact(...) 

IMediaExtractorService
IExtractor
IDataSource
...

Interface Model Extractor

onTransact

tx1
readInt32
... num 4996

if stmt
... return

reply BAD_VALUE ERROR

Abstract Syntax Tree

Dependency Inferer

Transaction Dependency Graph

Dependency Inferer

IMediaExtractorService
IMediaExtractor
IDataSource

Interface Dependency Graph

Fuzzer Engine

Fuzzer Manager

push corpus push fuzzer

pull logs

fetch corpus store logs

Corpus Database Crash Log

7
Interface Collector

- Interface feature
  - Services use `onTransact` method to dispatch transactions

- Collection approach
  - Compile AOSP and record compilation commands
    - During compilation, interface-related files will be used
  - Scan every C++ source file in compilation commands
    - Seek for those files which contain the `onTransact` pattern
Interface Model Extractor

Interface Collector

Source

Compile

status_t XXX::onTransact(...)

IMediaExtractorService
IMediaExtractor
IDataSource
...

Interface Model Extractor

Abstract Syntax Tree

Interface Dependency Graph

Transaction Dependency Graph

Dependency Inferer

Transaction

IMediaExtractor-Service
IMediaExtractor
IDataSource
Interface
Dependency
Graph

Fuzzer Engine

Fuzzer Manager

push corpus
push fuzzer

Fuzzing

pull logs

fetch corpus

store logs

Corpus Database

Crash Log
Transaction Code Identification

- Services use onTransact method to dispatch transactions
  - This process is usually implemented as a switch statement

- Identification Solution
  - Identify all transactions of a target interface by analyzing case nodes in the abstract syntax tree
Input and Output Variable Extraction

- System services utilize *special methods* (e.g., `readInt32`, `writeInt32`) to deal with input and output variables
- Extract I/O variables through recognizing such methods
  - **Variable pattern**
    - Variables might locate in sequential / conditional / loop statements
    - Sequential pattern, conditional pattern, loop pattern
  - **Variable name**
  - **Variable type**

For more details, please refer to the paper.
Auxiliary Information Extraction

- **Transaction paths**
  - Separated by the return statement

- **Extract type definition**
  - Structure and union definition
  - Enumeration definition
  - Type alias
Dependency Inferer

Interface Collector

Source

Interface Model Extractor

status_t XXX::onTransact(...)

IMediaExtractorService
IMediaExtractor
IDataSource
...

Interface Infrer

Transaction Dependency Graph

Fuzzer Engine

Fuzzer Manager

push corpus
push fuzzer

pull logs

fetch corpus
store logs

Corpus Database
Crash Log
Interface Dependency

- **Generation dependency**
  - writeStrongBinder method

- **Use dependency**
  - readStrongBinder method

```cpp
/* The following code is in IMediaExtractorService.cpp. */
// generation dependency
sp<IDataSource> source = makeIDataSource(fd, offset, length);
reply->writeStrongBinder(IInterface::asBinder(source));
// use dependency
status_t ret = data.readStrongBinder(&b);
... sp<IDataSource> source = interface_cast<IDataSource>(b);
```
Variable Dependency

- **Intra-transaction dependency, e.g., conditional dependency**
  - It can be inferred when extracting the interface model

- **Inter-transaction dependency, inference principles:**
  - One variable is input, and the other is output
  - These two variables are located in different transactions
  - Input variable’s type is equal to the output variable’s type
  - Either the input variable type is complex, or the input variable name and the output variable name are similar
Fuzzer Engine
Fuzzer Engine

- **Fuzzer**
  - Randomly generate a transaction
  - Generate the corresponding interface
  - Invoke the target transaction

- **Fuzzer manager**
  - Run fuzzer
  - Monitor fuzzer’s status and restart fuzzer when finding it exited
  - Synchronize logs from mobile to host
Implementation

- Language: C++, Python
- LoC: more than 10,000 lines

<table>
<thead>
<tr>
<th>Component</th>
<th>Language</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Collector</td>
<td>Python</td>
<td>145</td>
</tr>
<tr>
<td>Interface Model Collector</td>
<td>C++, Python</td>
<td>5238</td>
</tr>
<tr>
<td>Dependency Inferer</td>
<td>Python</td>
<td>291</td>
</tr>
<tr>
<td>Fuzzer Engine</td>
<td>C++, Python</td>
<td>5070</td>
</tr>
<tr>
<td>Total</td>
<td>C++, Python</td>
<td>10744</td>
</tr>
</tbody>
</table>

18
Evaluation

- Q1. How many interfaces have been found? What is the relationship between them?
- Q2. What does the extracted interface model look like? Is the model complete and precise?
- Q3. How effective is FANS in discovering vulnerabilities of Android native system services?
Environment

- **Host**
  - Ubuntu 18.04, i9-9900K CPU, 32GB memory, 2.5T SSD

- **Mobile Phone**
  - 1 Pixel, 4 Pixel 2 XLs, 1 Pixel 3 XL

- **Android version: android-9.0.0_r46**
  - The source code can be different for different Pixel models
  - We answer the Q1 and Q2 through the experiment results carried out on Pixel 2 XL
Q1 - Interface Statistics

- 43 top-level interfaces
- 25 multi-level interfaces
- Most interfaces are written manually
Q1 - Interface Dependency

- **Interface generation**
  - e.g., IMemory

- **Deepest interface**
  - IMemoryHeap (five-level)

- **Customized interface**
  - e.g., IEffectClient
Q2 - Extracted Interface Model Statistics

- **Transaction**
  - 530 transactions in top-level interfaces
  - 281 transactions in multi-level interfaces

- **Variable**
  - Most variables are under constraint(s)
Q2 - Completeness and Precision

- **Background**
  - There is no ground truth about the interface model

- **Methodology**
  - Randomly select 10 interfaces
  - Manually check the interface models

- **Result**
  - **Completeness**: all of the transaction codes are recovered
  - **Precision**: almost all variable patterns, variable names, and variable types are recovered
  - The imprecision is mainly due to the complexity of the source code
We intermittently ran FANS for around 30 days
FANS triggered thousands of crashes
- 30 vulnerabilities in native programs
  - Google has confirmed 20 vulnerabilities
- 138 Java exceptions

Comparison with BinderCracker
- BinderCracker found 89 vulnerabilities on Android 5.1 and Android 6.0
- FANS discovered 168 vulnerabilities on android-9.0.0_r46
Discussion

- Improve the accuracy of the interface model
- Integrate coverage into FANS
- Improve the efficiency of FANS
- Extend FANS to other interface-based programs in Android
  - e.g., native system services implemented by vendor, java system services
Conclusion

- A systematical investigation of interface dependency
- An approach to automatically extract interface model
- An approach to infer inter-transaction variable dependency
- A prototype of FANS
  - 30 vulnerabilities in native programs and 138 Java exceptions
  - Source: [https://github.com/iromise/fans](https://github.com/iromise/fans)
Thanks for listening!

Q & A

Contact: Baozheng Liu, uromise@gmail.com