

Driving Execution of Target Paths in Android Applications with (a) CAR

Michelle Y. Wong and David Lie

University of Toronto

AsiaCCS 2022










The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Outline

- Background on targeted execution
- CAR - Context Approximation and Refinement
 - Combination of static and dynamic techniques
 - 3.1x increase in coverage of sensitive targets in Android applications
- Design and Implementation
- Evaluation

Android security

- Smartphones contain a trove of personal data and sensitive functionality
 - Contacts, emails, messages  
 - Location and health tracking   
 - Access to bank accounts and smart home devices  
- Lucrative target for malware developers
- Application marketplaces want to remove malware from their offerings



Program analysis for malware detection

Static analysis

- Analyze application's code or binary
- No access to run-time data during analysis
- Trade-off between precision, completeness, and scalability

```
MainActivity.java
.method public constructor <init>()V
    .locals 3
    .prologue
    const/4 v2, 0x1
    const/4 v1, 0x0
    .line 131
    invoke-direct {p0}, Lcom/gye/ace/kjv/ActionBarThemeActivity;-><init>()V
    ...
    .method logScribeEvent(Lcom/mopub/network/AdResponse;Lcom/mopub/volley/
    NetworkResponse;Landroid/location/Location;)V
    .locals 6
    ...
    .line 365
    :goto_1
    invoke-virtual {v2, v0}, Lcom/mopub/common/event/BaseEvent$Builder;
    >withAdHeightPx(Ljava/lang/Double;)Lcom/mopub/common/event/
    BaseEvent$Builder;
    move-result-object v2
    if-eqz p3, :cond_3
    invoke-virtual {p3}, Landroid/location/Location;->getLatitude()D
    move-result-wide v4
    invoke-static {v4, v5}, Ljava/lang/Double;->valueOf(D)Ljava/lang/Double;
    ...
```

Dynamic analysis

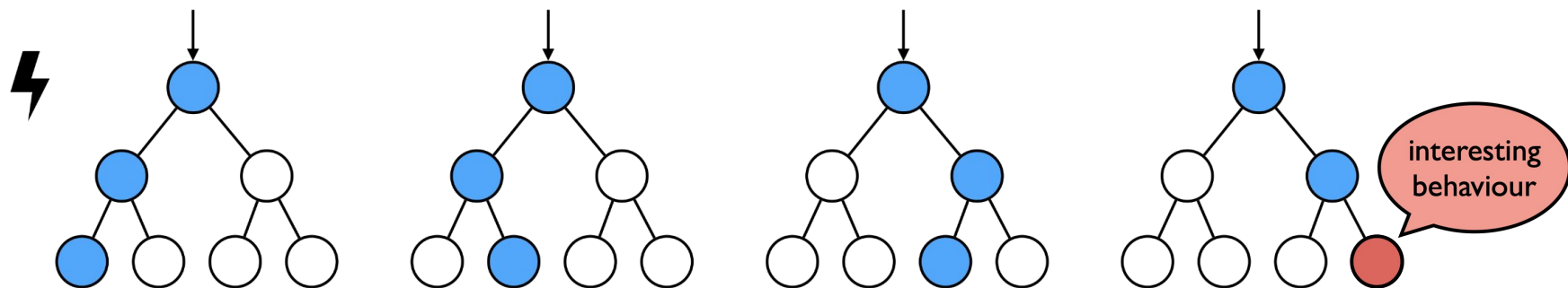
- Analyze application during execution
- Analysis limited to parts of applications that are executed
- Malware activity hidden if only activated under specific triggering conditions

```

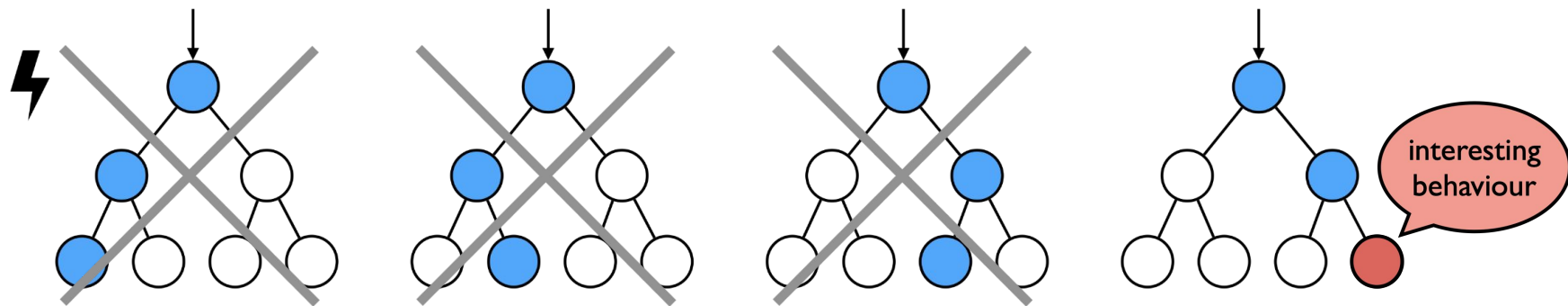
invoke-special r0 <init>
<frame set-up>
r2 <- 0x1
r1 <- 0x0
invoke-super r0 <init>
...
invoke-virtual logScribeEvent
<frame set-up>
...
invoke-virtual r2, r0 withAdHeightPx
r2 <- r0
invoke-virtual r3 getLatitude()
r4 <- r0
invoke-static r4, r5 valueOf
```

Increasing Dynamic Code Coverage

- Code exploration to execute more of the application (e.g. fuzzing, symbolic execution)
 - Goal is full application coverage
- Security analysis aims to detect specific interesting or suspicious behaviours
 - Occur in only a few locations in the application's code base



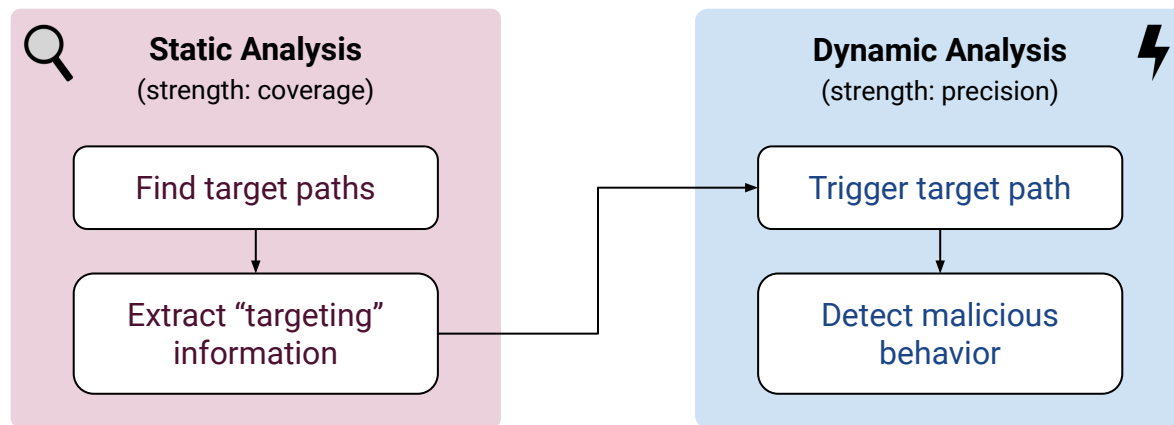
Alternative Approach: Targeted Execution



Challenges:

- How to find and target interesting behaviours and paths?
- How to inject and execute specific paths?

Targeting using Static Information



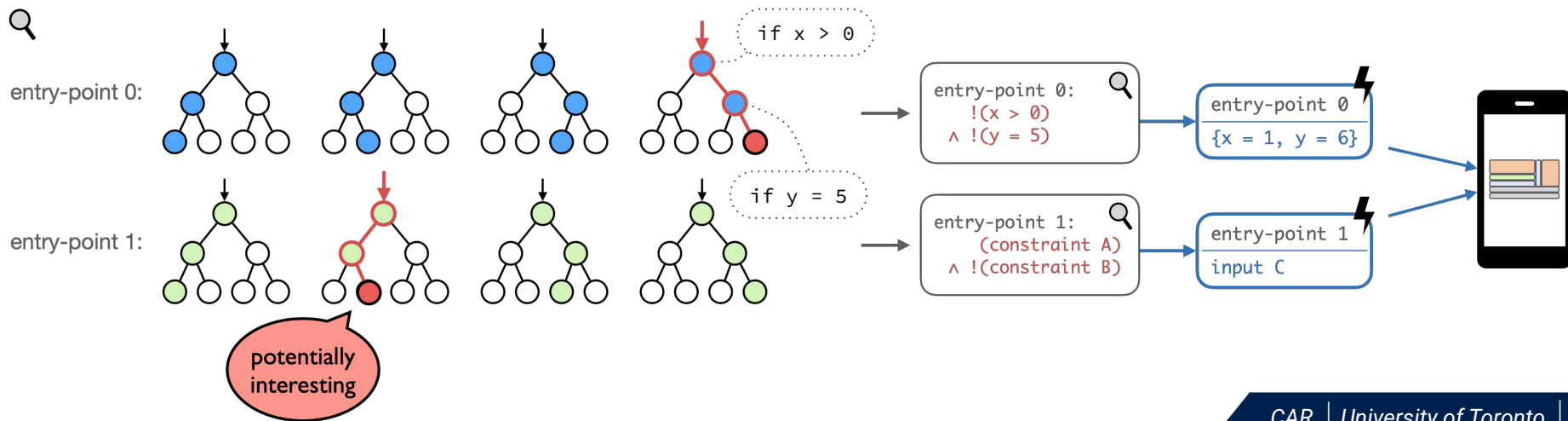
Prior work:

- Guided symbolic/concolic execution [expensive]
- Forced execution (modify conditional branches) [unsound]
- Static symbolic constraint analysis to inject input/system values [incomplete]

Existing Approach to Static Targeting

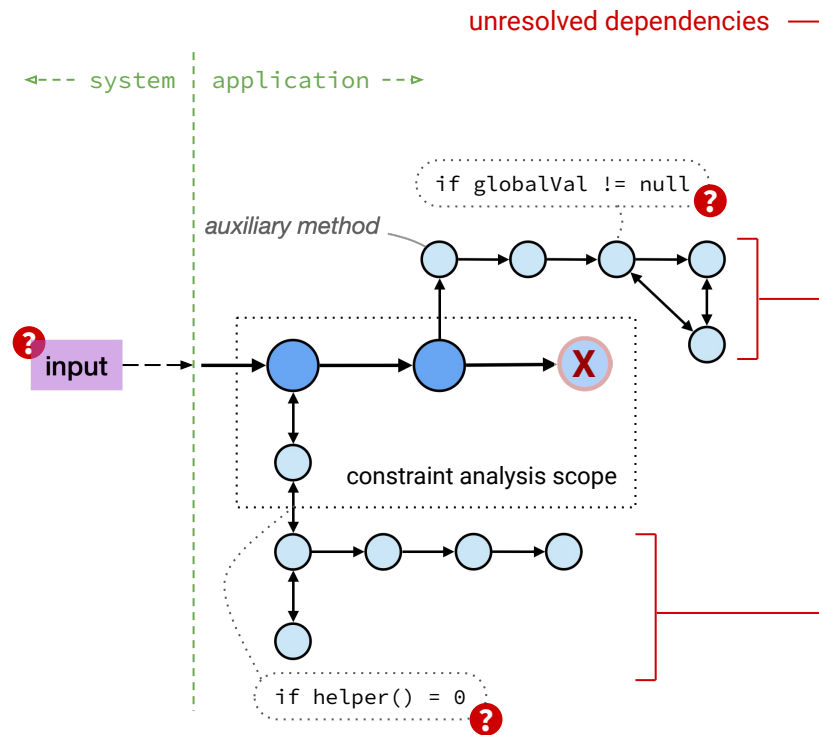
Prior work (IntelliDroid and TIRO):

- Find target paths that reach sensitive targets.
- Use static constraint analysis to extract path conditionals.
- Solve conditional expression to obtain input/system values to inject.
- Injected values resolve path dependencies, path is executed.



Limitations

- Static constraint analysis is expensive
 - Limit scope based on depth of auxiliary methods.
 - Unresolved dependencies for methods outside of analysis scope.
 - Prevent path from being executed in full [incomplete].
- Injection at system level is expensive.
 - Need to model each input or API type.
 - Any missing models leads to inability to trigger path in execution [incomplete].



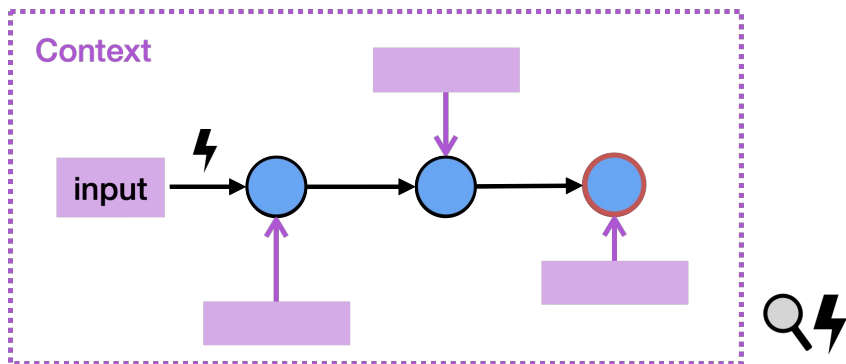
CAR: Context Approximation and Refinement

Context - Set of program state accessed by a target path and constrained by its execution

- Input, global values, method/API return values

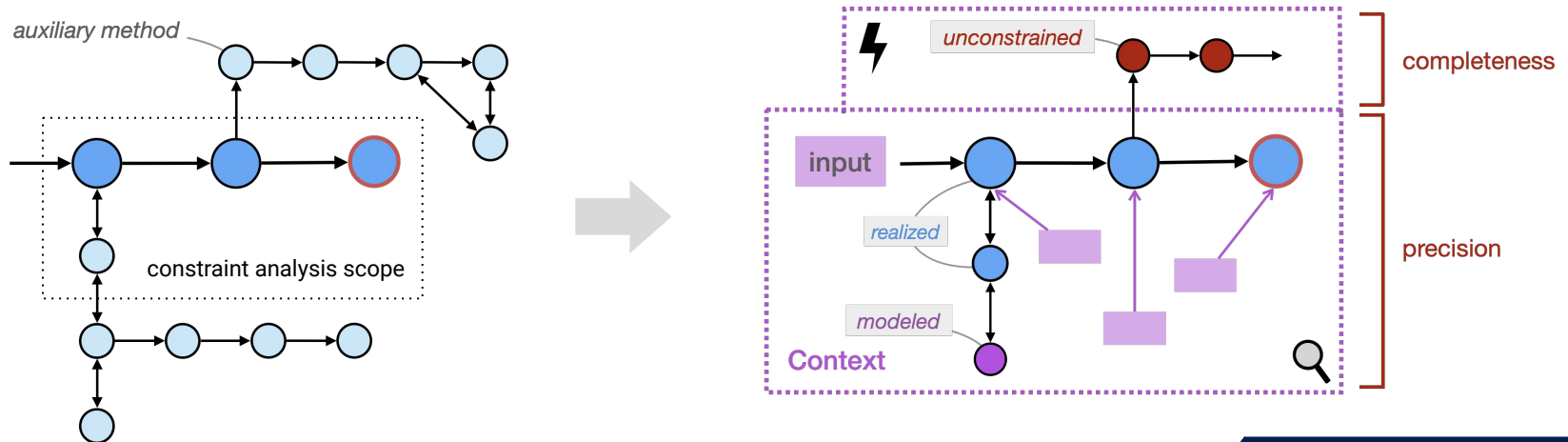
Context **A**pproximation and **R**efinement

- Combination of static and dynamic techniques to infer context for a target path



Handling Static Targeting Limitations

- Scalability requires limiting scope of constraint analysis.
- Create an *approximate context*.
 - Split control flow of target path into realized, modeled, and unconstrained methods



Example: Approximate Context Generation

```
void onClick(View view) {  
    TextView x = (TextView)view;  
    if (x.position > 5) {  
        method1();  
    }  
}  
void method1() {  
    if (helper1()) {  
        <target 0>  
    }  
}  
bool helper1() {  
    methodWithSideEffect();  
    return helper2() && Config.Enabled;  
}
```

realized

unconstrained

modeled

CAR: injection.dex

```
void PathDriver0() {  
    View arg0 = new ConstrainedTextView_Path0();  
    Config.Enabled = true;  
    onClick(arg0);  
}  
class ConstrainedTextView_Path0 extends TextView {  
    void <init>() {  
        super.<init>();  
        this.position = 6;  
    }  
}  
static void helper2() { // Modified  
    if (<Path 0>) {  
        return true;  
    } else {  
        return original_helper2a();  
    }  
}
```

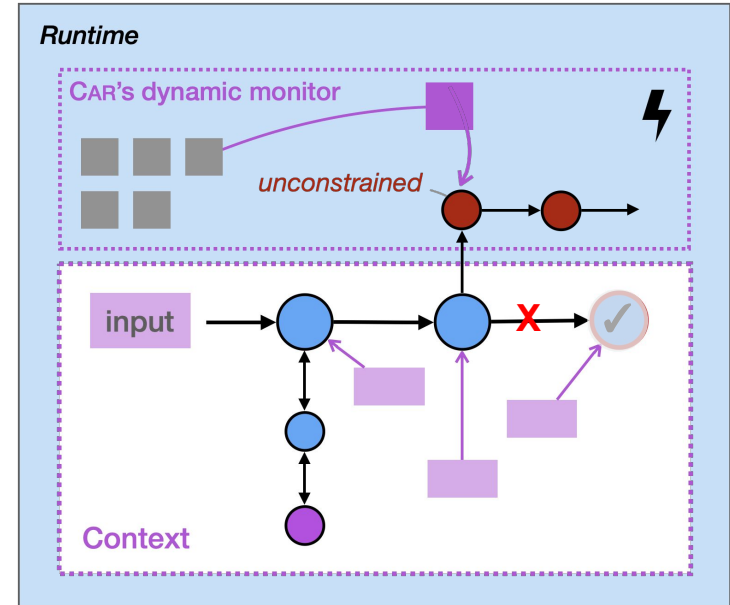
constrained subclass

constrained method



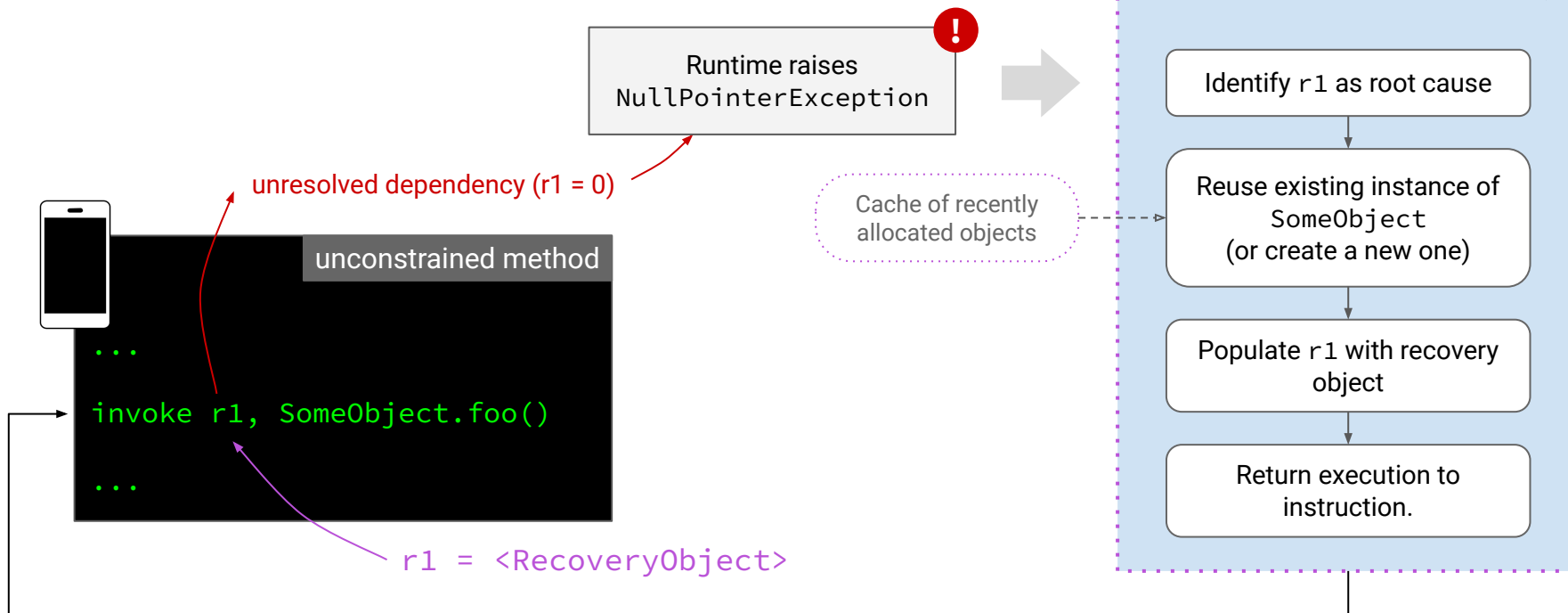
Handling Unresolved Dependencies

- Unconstrained methods can contain unresolved dependencies.
- Refine context using *dynamic error recovery*.
 - Lightweight instrumentation to transparently resolve dependency errors as they occur.
- CAR: Dynamic monitor
 - Tracks errors arising from unresolved dependencies.
 - Resolves dependency through error recovery.
- Unconstrained method proceeds, returns to target path.



Example: Dynamic Context Refinement

Most common: `NullPointerException`



Implementation

Static component

- Built on the Soot static analysis framework.
- Based on IntelliDroid and TIRO.
 - Increased dependency tracking for contexts.
 - Added code generation to produce event driving framework for approximated context.



Dynamic component

- Built on AOSP (Android 10).
- Instrumented class loader to inject generated context classes.
- Instrumented exception handler to track and resolve dependencies for context refinement.



Evaluation

Metrics

- Effectiveness of contexts
 - Target coverage against existing state-of-the-art dynamic tools.
 - False positives and infeasible paths.
- Performance

Datasets

- Benchmark of evasive applications
- Real-world applications (most popular applications on Google Play)
- Known harmful applications

Evaluation: EvaDroid Benchmark

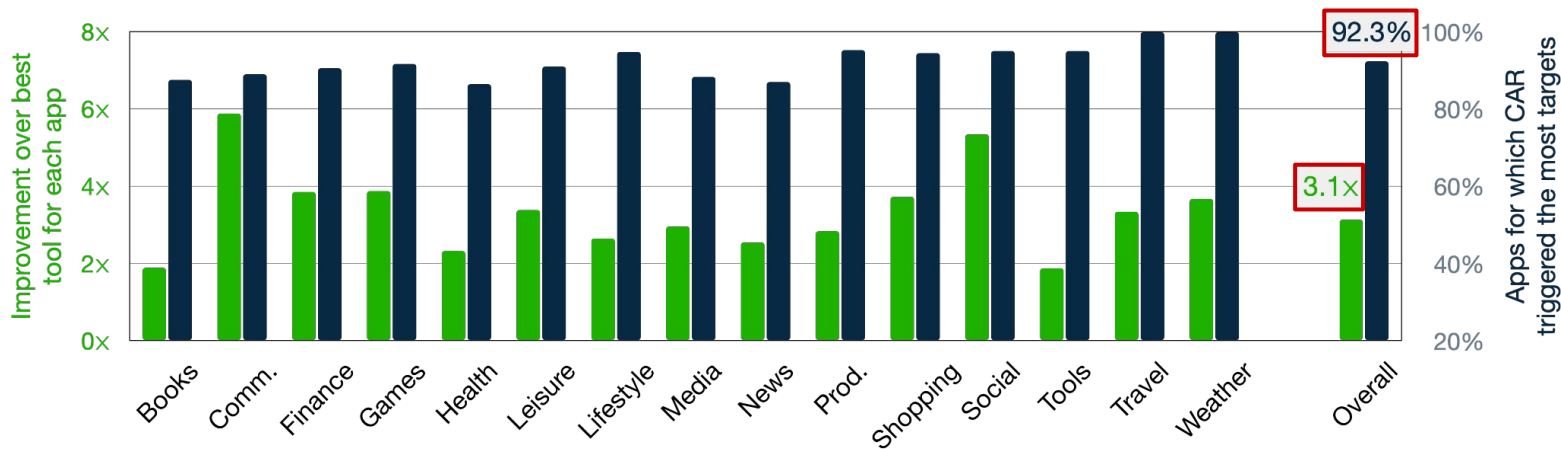
- Benchmark of evasive behavior in Android applications
- Each sample tries to hide a payload from executing during dynamic analysis
- Missed payloads: time/sleep-based constraints not yet supported

| Dynamic Tool | Payloads Triggered |
|--------------|--------------------|
| CAR | 73% |
| DroidBot | 15% [1] |
| GroddDroid | 37% [1] |
| IntelliDroid | 33% [1] |

[1] Aleieldin Salem, Michael Hesse, Jona Neumeier, and Alexander Pretschner. 2019. Towards Empirically Assessing Behavior Stimulation Approaches for Android Malware. In Proceedings of the 13th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2019). IARIA XPS Press, 47–52.

Evaluation: Most Popular Apps on Google Play

- 310 applications with the most installations on Google Play, across 15 categories.
- Measured ability to reach target security sensitive APIs.
- Compared target coverage against: Monkey, DroidBot [1], APE [2].



[1] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: A lightweight UI-guided test input generator for Android. In Proceedings of the 39th International Conference on Software Engineering Companion Volume (ICSE-C 2017). IEEE, 23–26.

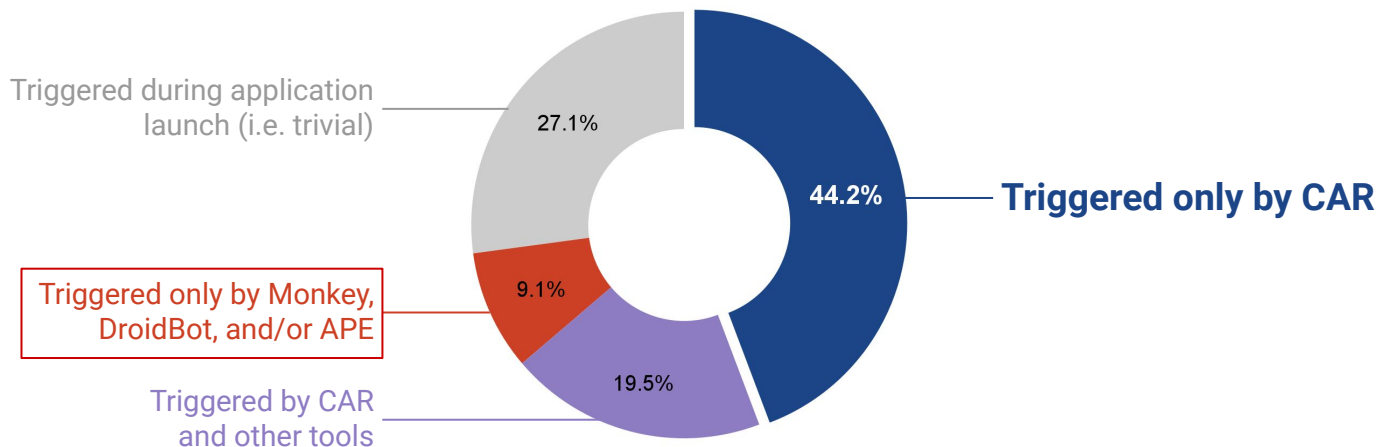
[2] Tianxiao Gu, Chengnian Sun, Xiaoxing Ma, Chun Cao, Chang Xu, Yuan Yao, Qirun Zhang, Jian Lu, and Zhendong Su. 2019. Practical GUI testing of Android applications via model abstraction and refinement. In Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE 2019). IEEE/ACM, 269–280.

Effectiveness of Contexts

44.2% of executed target paths were triggered only by CAR

- Context approximation (generated constrained classes) required in **84.1%**.
- Context refinement (dynamic error recovery) required in **46.3%**.

Estimated false negative rate of **9.1%**



Evaluation: Coverage of Security Sensitive Behaviors

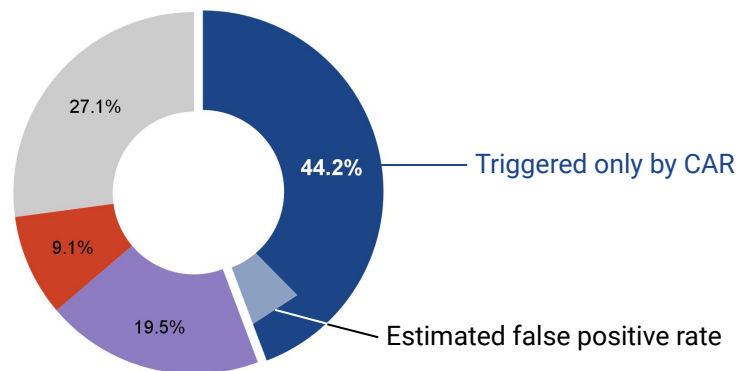
- Datasets:
 - 310 applications from Google Play
 - 91 malware samples from the Creepware dataset [1]
- Triggered sensitive behaviours missed by all other tools
 - 90% required context-based dependency resolution

| Sensitive functionality | Google Play | | | Creepware | | |
|-------------------------|-------------|-------|------|-----------|-------|------|
| | Apps | Calls | Cxt. | Apps | Calls | Cxt. |
| Location | 84 | 237 | 89% | 14 | 42 | 86% |
| Personal data | 6 | 6 | 83% | 4 | 4 | 100% |
| Media | 3 | 3 | 100% | 1 | 1 | 100% |
| Telephony | 7 | 11 | 100% | 4 | 4 | 50% |
| Network | 59 | 99 | 74% | 11 | 20 | 85% |
| Files | 220 | 1199 | 88% | 63 | 361 | 10% |
| Databases | 76 | 187 | 90% | 11 | 47 | 90% |
| Reflection | 169 | 447 | 84% | 25 | 113 | 75% |
| Code loading | 7 | 7 | 71% | 10 | 10 | 60% |
| Native code | 223 | 1090 | 88% | 29 | 120 | 68% |

[1] Kevin A Roundy, Paula Barmaimon Mendelberg, Nicola Dell, Damon McCoy, Daniel Nissani, Thomas Ristenpart, and Acar Tamersoy. 2020. The Many Kinds of Creepware Used for Interpersonal Attacks. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP 2020). IEEE.

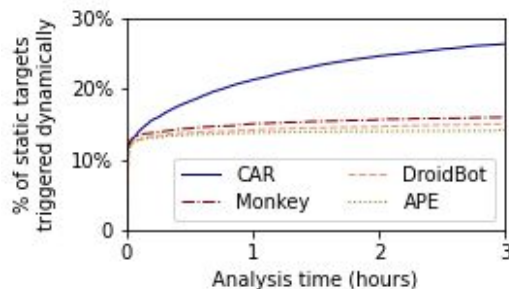
False Positives

- Randomly sampled 75 paths that were triggered only by CAR.
 - Across different app categories.
 - Manually analyzed feasibility of path.
- **9.0%** estimated false positive rate.
 - Primarily due to imprecision in the points-to analysis and call-graph, especially for third-party libraries.
 - Concentrated in the Games category.



Performance

- Static target path extraction, constraint analysis, context generation
 - Timeout: 240 minutes
- Dynamic path injection
 - Timeout: 3 hours, with 10s throttle between each injected path
 - Average 1.4 seconds for each path to reach target



Conclusion

- CAR - Context Approximation and Refinement
- Combine static and dynamic analysis for scalable targeted execution
 - Static analysis to generate approximate context
 - Dynamic error recovery to refine context for unresolved dependencies
- Reached **3.1x** more sensitive targets than state-of-the-art dynamic tools
- False detection rate of **9.0%**
- Reach and triggered more security sensitive behaviors for dynamic analysis

Thank you!