

# DexLego: Reassembleable Bytecode Extraction for Aiding Static Analysis

Zhenyu Ning and Fengwei Zhang

COMPASS Lab  
Wayne State University

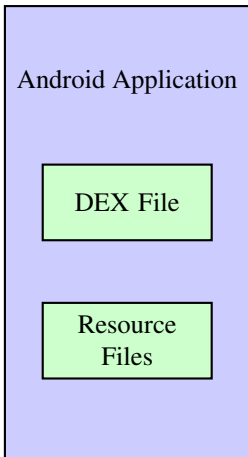
# Outline

- ▶ Introduction
- ▶ System Overview
- ▶ Implementation
- ▶ Evaluation
- ▶ Conclusions

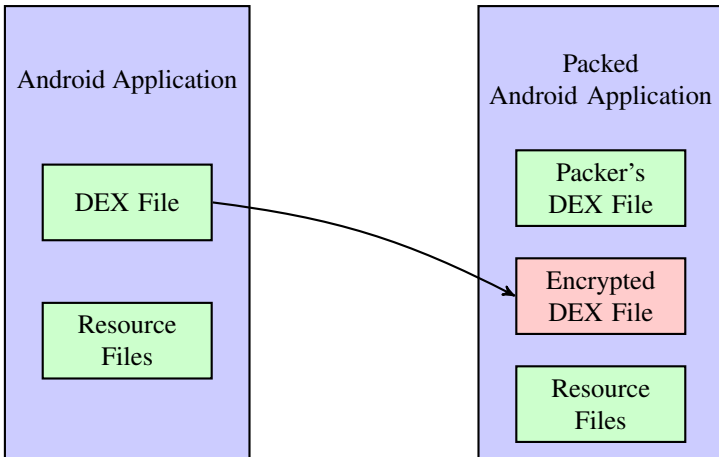
- ▶ [Introduction](#)
- ▶ System Overview
- ▶ Implementation
- ▶ Evaluation
- ▶ Conclusions

## Why still doing static analysis?

# Packing Technique



# Packing Technique



- ▶ Previous Android Unpacking Systems: DexHunter [1], AppSpear [2]
  - ▶ Assuming a clear transition between the packer's code and the original code.
  - ▶ Using method-level collection to collect code.

# Self-modifying Code

---

```
1 public void example() {  
2     helper();  
3     Log.d("Hello World!");  
4 }
```

---



# Self-modifying Code

---

```
1 public void example() {  
2     helper();  
3     Log.d("Hello World!");  
4 }
```

---

---

```
1 public void example() {  
2     helper();  
3     Log.d("Password is 1234!")  
4 }
```

---

- ▶ Android allows the applications modify its bytecode at runtime.
- ▶ These changes cannot be detected by current static analysis tools.

---

```
1 public class Main extends Activity {
2     protected void onCreate(Bundle savedInstanceState) {
3         //...
4         // 55 is the index of method "reflectiveLeak" in the method
           array of class "Main"
5         Method reflectiveLeakMethod = getClass().getMethods()[55];
6         reflectiveLeakMethod.invoke(this, "sensitive data");
7     }
8
9     public void reflectiveLeak(String data) {
10        // leak data
11        SmsManager.getDefault().sendTextMessage("800-123-456", null,
           data, null, null);
12    }
13 }
```

---

- ▶ Additional DEX files can be downloaded from cloud at runtime.
- ▶ Malicious activities in these DEX files are ignored by current static analysis tools.

## Why not using dynamic analysis?

## Challenges of Dynamic Analysis

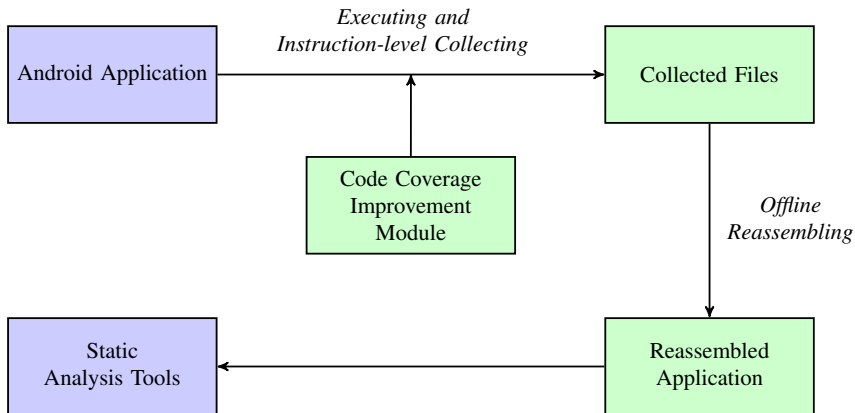
- ▶ Implicit Flows
- ▶ Performance Overhead vs Accuracy

## **DexLego: Instruction-level collecting and offline reassembling**

- ▶ Use dynamic approach to collect executed instruction.
- ▶ Improve the current static analysis via offline reassembling.

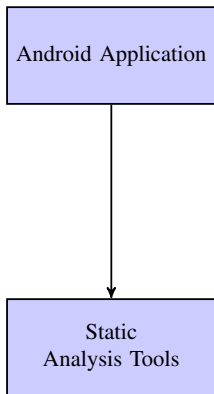
- ▶ Introduction
- ▶ System Overview
- ▶ Implementation
- ▶ Evaluation
- ▶ Conclusions

# System Overview

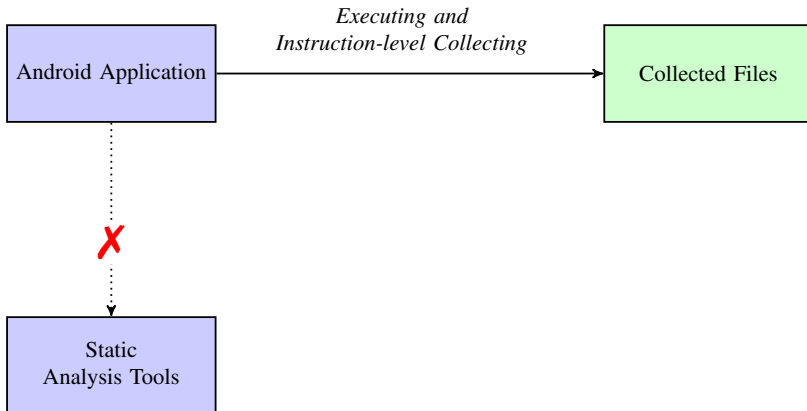




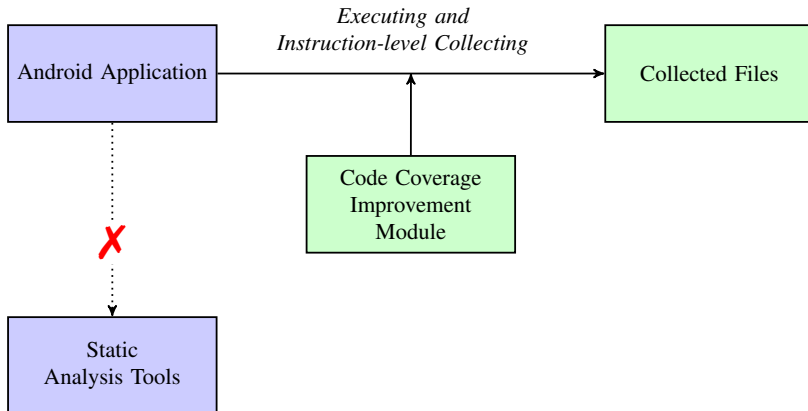
# System Overview



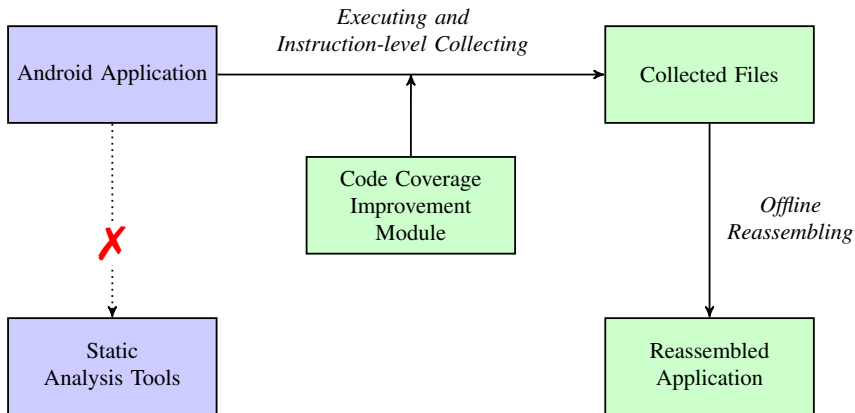
# System Overview



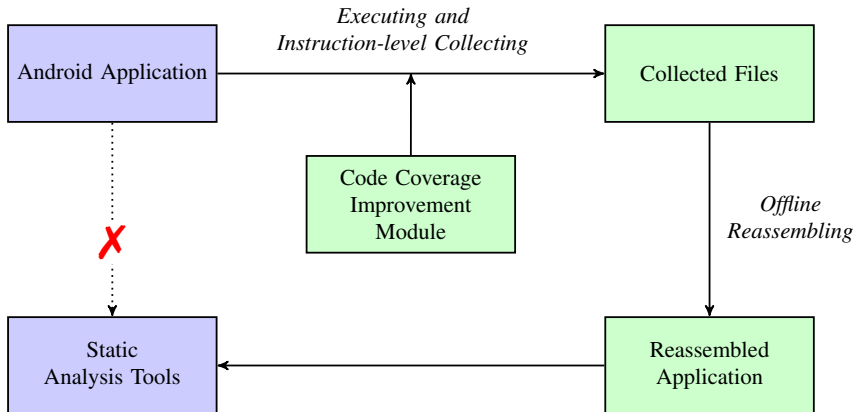
# System Overview



# System Overview



# System Overview



- ▶ Introduction
- ▶ System Overview
- ▶ Implementation
- ▶ Evaluation
- ▶ Conclusions

**Simply list the executing instructions  
one-by-one?**

---

```
1 public void example() {  
2     int i;  
3     for (i = 0; i < 10000; ++i) {  
4         Log.d("Hello World!");  
5     }  
6 }
```

---



---

```
1 public void example() {  
2     int i;  
3     for (i = 0; i < 10000; ++i) {  
4         Log.d("Hello World!");  
5     }  
6 }
```

---

---

```
1 public void example() {  
2     Log.d("Hello World!");  
3     Log.d("Hello World!");  
4     Log.d("Hello World!");  
5     Log.d("Hello World!");  
6     Log.d("Hello World!");  
7     Log.d("Hello World!");  
8     Log.d("Hello World!");  
9     Log.d("Hello World!");  
10    ...  
11 }
```

---

- ▶ Record both the index and the content of each instruction.
- ▶ Instruction with same index and content will not be repeatedly collected.

## How about self-modifying code?

## ► **Collection Tree**

- A independent tree for each execution of each method.
- Each node indicates a piece of changed code.

---

```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         benign();
13         helper();
14     }
15 }
```

---

---

```
1 // No malicious activities
2 public void benign() {}
3
4 // Leak data
5 public void malicious() {}
6
7 // Modify line 12 to
   "malicious()" at runtime
8 public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         benign();
13         helper();
14     }
15 }
```

---

---

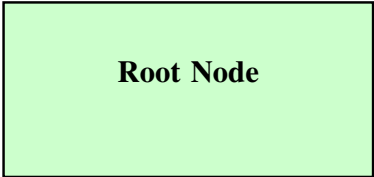
```
1 // No malicious activities
2 public void benign() {}
3
4 // Leak data
5 public void malicious() {}
6
7 // Modify line 12 to
   "malicious()" at runtime
8 public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         malicious();
13         helper();
14     }
15 }
```

---

---

```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         benign();
13         helper();
14     }
15 }
```

---



**Root Node**

**i = 0**

---

```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         benign();
13         helper();
14     }
15 }
```

---

## Root Node

```
for (int i = 0; i < 2; ++i) {
}
```



**i = 0**

---

```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         benign();
13         helper();
14     }
15 }
```

---

## Root Node

```
for (int i = 0; i < 2; ++i) {
    benign();
}
```

**i = 0**

---

```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         benign();
13         helper();
14     }
15 }
```

---

## Root Node

```
for (int i = 0; i < 2; ++i) {
    benign();
    helper();
}
```

**i = 1**

---

```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         malicious();
13         helper();
14     }
15 }
```

---

## Root Node

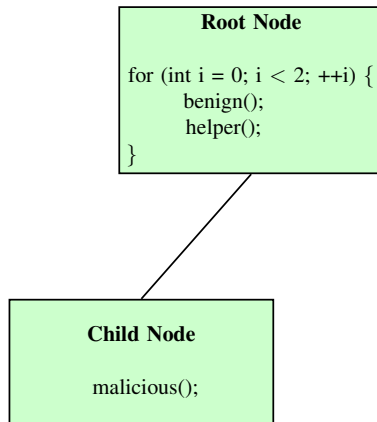
```
for (int i = 0; i < 2; ++i) {
    benign();
    helper();
}
```

**i = 1**

---

```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         malicious();
13         helper();
14     }
15 }
```

---

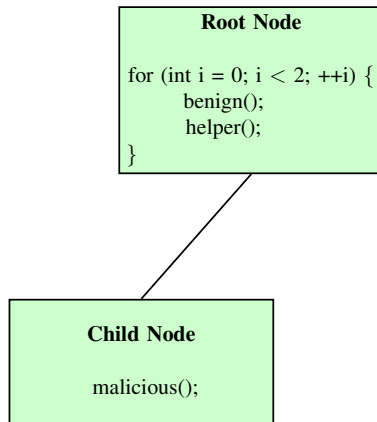


**i = 1**

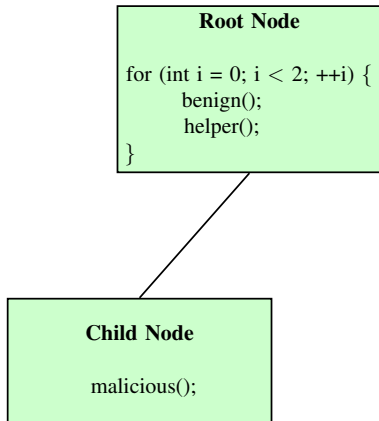
---

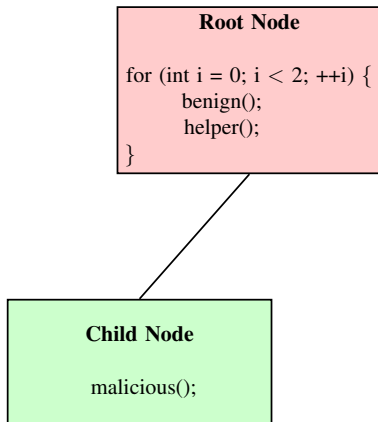
```
1  // No malicious activities
2  public void benign() {}
3
4  // Leak data
5  public void malicious() {}
6
7  // Modify line 12 to
   "malicious()" at runtime
8  public void helper() {}
9
10 public void execute() {
11     for (int i = 0; i < 2; ++i) {
12         malicious();
13         helper();
14     }
15 }
```

---



# Bytecode Reassembling



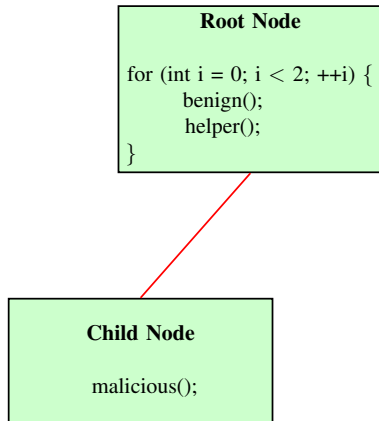


---

```
1 public void execute() {  
2     for (int i = 0; i < 2; ++i) {  
3         benign();  
4         helper();  
5     }  
6 }
```

---

# Bytecode Reassembling

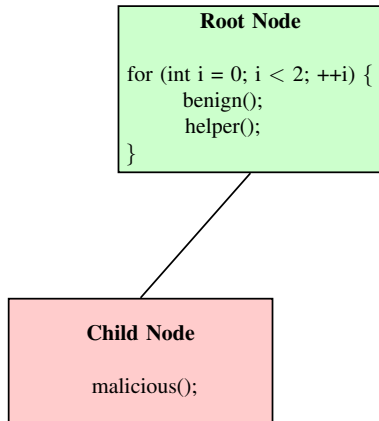


---

```
1  public void execute() {  
2      for (int i = 0; i < 2; ++i) {  
3          if (RANDOM_VALUE) {  
4              benign();  
5          } else {  
6                
7          }  
8          helper();  
9      }  
10 }
```

---





---

```
1  public void execute() {  
2      for (int i = 0; i < 2; ++i) {  
3          if (RANDOM_VALUE) {  
4              benign();  
5          } else {  
6              malicious();  
7          }  
8          helper();  
9      }  
10 }
```

---

- ▶ Replace the reflective calls with the direct calls during bytecode collection.
- ▶ Use force execution to improve the code coverage.

- ▶ Introduction
- ▶ System Overview
- ▶ Implementation
- ▶ [Evaluation](#)
- ▶ Conclusions

- ▶ Testbed Specification
  - ▶ LG Nexus 5X
  - ▶ A dual-core 1.8 GHZ Cortex-A57 cluster and a quad-core 1.4 GHZ Cortex-A53 cluster
  - ▶ Android 6.0 and TWRP Recovery

**Can we correctly reconstruct the  
behavior of apps?**

Table: Test Result of Different Packers.

Applications	HTMLViewer	Calculator	Calendar	Contacts
# of Instructions	217	2,507	78,598	103,602
360 [3]	✓	✓	✓	✓
Alibaba [4]	✓	✓	✓	✓
Tencent [5]	✓	✓	✓	✓
Baidu [6]	✓	✓	✓	✓
Bangcle [7]	✓	✓	✓	✓

**How is DexLego comparing with other tools?**

# Evaluation

- ▶ 134 samples from DroidBench [8].
- ▶ 3 static analysis tools: FlowDroid [9], DroidSafe [10], HornDroid [11].
- ▶ 2 unpacking tools: DexHunter [1], AppSpear [2].

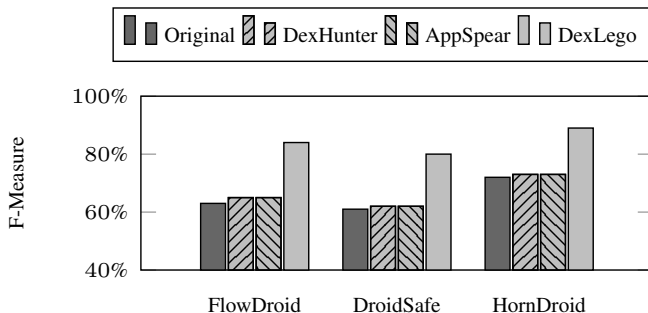


Figure: F-Measure of DroidBench samples.



**Can DexLego work with real-world  
packed apps?**

**Table:** Analysis Result of Packed Real-world Applications.

Package Name	Version	# of Installs	Original	Reassembled
com.lenovo.anyshare	3.6.68	100 million	0	4
com.moji.mjweather	6.0102.02	1 million	0	5
com.rongcai.show	3.4.9	100 thousand	0	3
com.wawoo.snipershootwar	2.6	10 million	0	4
com.wawoo.gunshootwar	2.6	10 million	0	5
com.alex.lookwifipassword	2.9.6	100 thousand	0	2
com.gome.eshopnew	4.3.5	15.63 million	0	3
com.szzc.ucar.pilot	3.4.0	3.59 million	0	5
com.pingan.pabank.activity	2.6.9	7.9 million	0	14

## How about code coverage?

**Table:** Samples from F-Droid [12].

Package Name	Version	# of Instructions
be.ppareit.swiftip	2.14.2	8,812
fr.gaulupeau.apps.InThePoche	2.0.0b1	29,231
org.gnucash.android	2.1.7	56,565
org.liberty.android.fantastischmemopro	10.9.993	57,575
com.fastaccess.github	2.1.0	93,913

**Table:** Code Coverage with F-Droid Applications.

	Class	Method	Line	Branch	Instruction
Sapienz [13]	44%	37%	32%	20%	32%
Sapienz + DexLego	87%	88%	82%	78%	82%

## Performance overhead?

# Evaluation

- ▶ 7.5x, 1.4x, and 2.3x overhead on Java score, native score, and overall score, respectively.

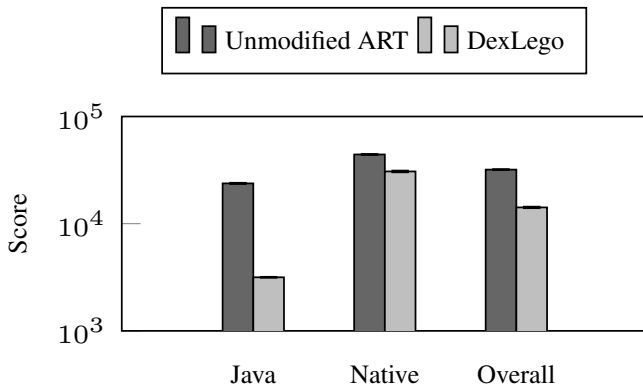


Figure: Performance Measured by CF-Bench [14].

- ▶ Introduction
- ▶ System Overview
- ▶ Implementation
- ▶ Evaluation
- ▶ Conclusions

- ▶ DexLego leverages instruction-level bytecode collecting and offline reassembling to aid existing static analysis tools.
- ▶ It helps to overcome the weakness of static analysis and increases the analysis accuracy with reasonable performance overhead.



- [1] Y. Zhang, X. Luo, and H. Yin, "DexHunter: Toward extracting hidden code from packed Android applications," in *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15)*, 2015.
- [2] W. Yang, Y. Zhang, J. Li, J. Shu, B. Li, W. Hu, and D. Gu, "AppSpear: Bytecode decrypting and DEX reassembling for packed Android malware," in *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions and Defenses (RAID'15)*, 2015.
- [3] Qihoo 360 Inc., "360Protector," <http://jiagu.360.cn/protection>, 2014.
- [4] Alibaba Inc., "AliProtector," <http://jaq.alibaba.com/>, 2014.
- [5] Tencent Inc., "TencentProtector," <http://legu.qqcloud.com/>, 2014.
- [6] Baidu Inc., "BaiduProtector," <http://app.baidu.com/jiagu/>, 2014.
- [7] Bangcle Ltd., "BangcleProtector," <https://www.bangcle.com/>, 2013.
- [8] EC SPRIDE Secure Software Engineering Group, "DroidBench," <https://github.com/secure-software-engineering/DroidBench>, 2013.
- [9] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Oteau, and P. McDaniel, "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14)*, 2014.
- [10] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, "Information flow analysis of Android applications in DroidSafe," in *Proceedings of the 22nd Network and Distributed System Security Symposium (NDSS'15)*, 2015.
- [11] S. Calzavara, I. Grishchenko, and M. Maffei, "HornDroid: Practical and sound static analysis of Android applications by SMT solving," in *Proceedings of the 1st IEEE European Symposium on Security and Privacy (EuroS&P'16)*, 2016.

# References II

- [12] F-Droid, “F-Droid,” <https://f-droid.org/>, 2011.
- [13] K. Mao, M. Harman, and Y. Jia, “Sapienz: Multi-objective automated testing for Android applications,” in *Proceedings of the 25th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'16)*, 2016.
- [14] Chainfire, “CF-Bench,” <https://play.google.com/store/apps/details?id=eu.chainfire.cfbench>, 2013.

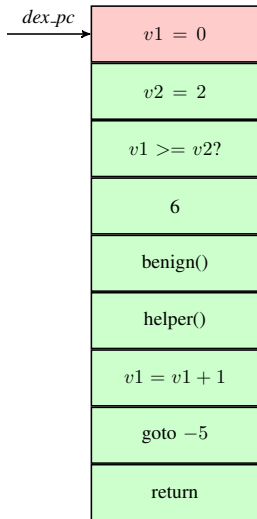
Thank you!

# Questions?

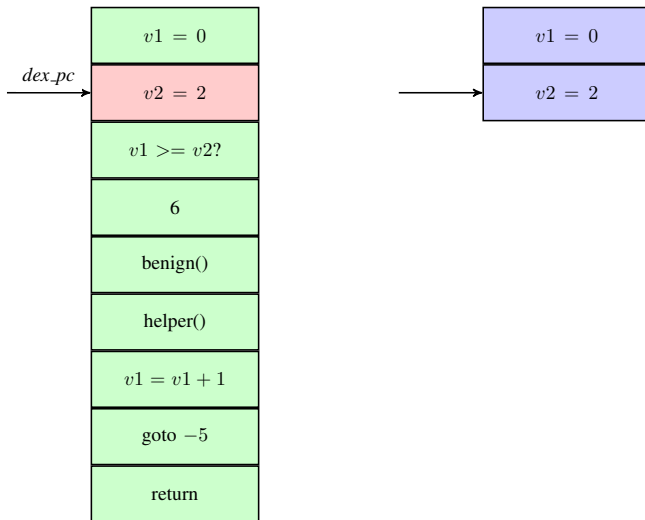
zhenyu.ning@wayne.edu

<http://compass.cs.wayne.edu>

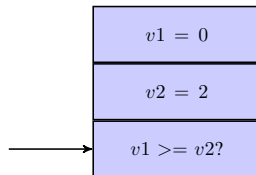
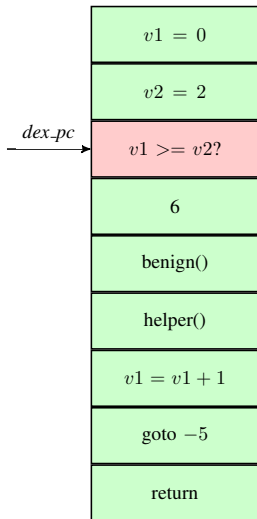
# Branch Instruction



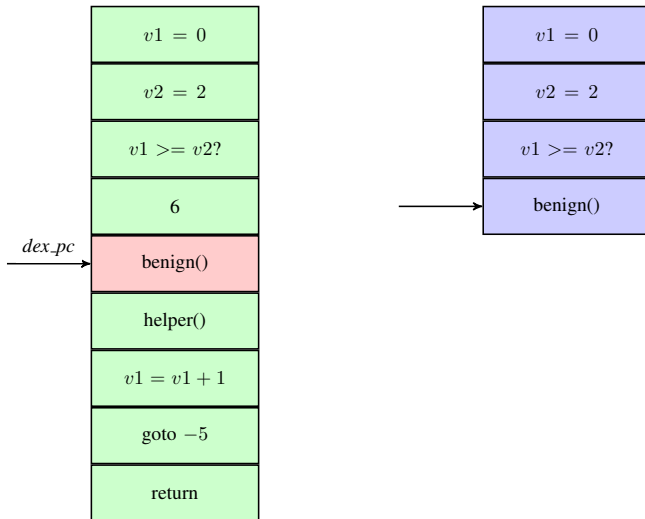
# Branch Instruction



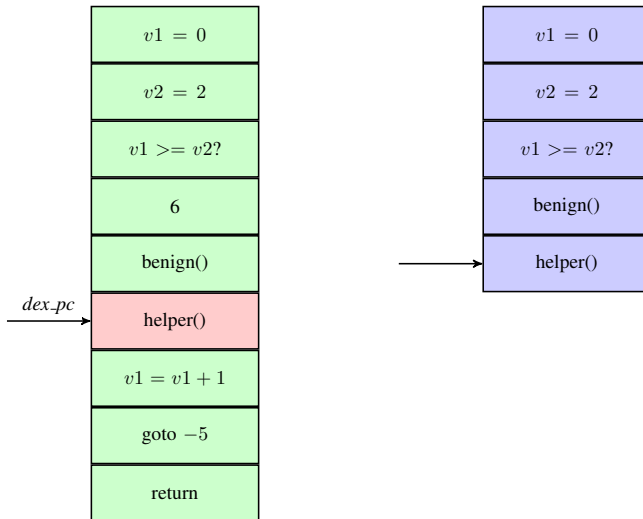
# Branch Instruction



# Branch Instruction

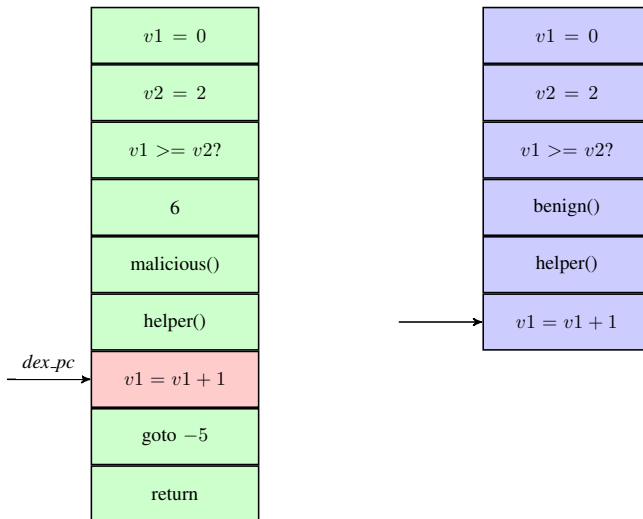


# Branch Instruction

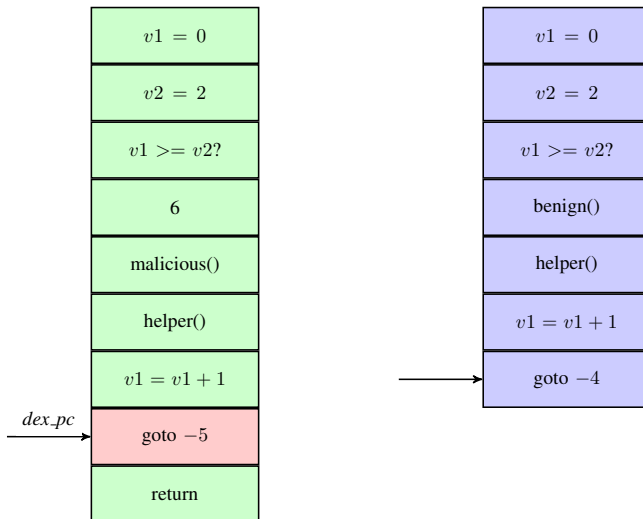




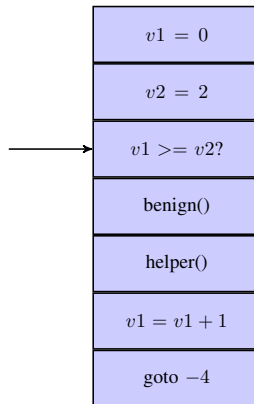
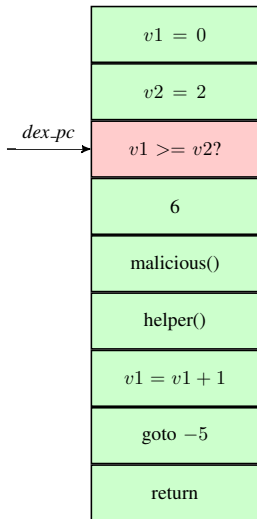
# Branch Instruction



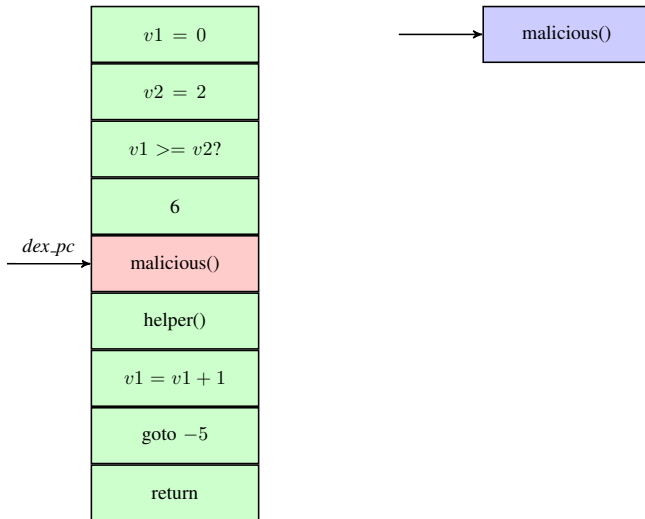
# Branch Instruction



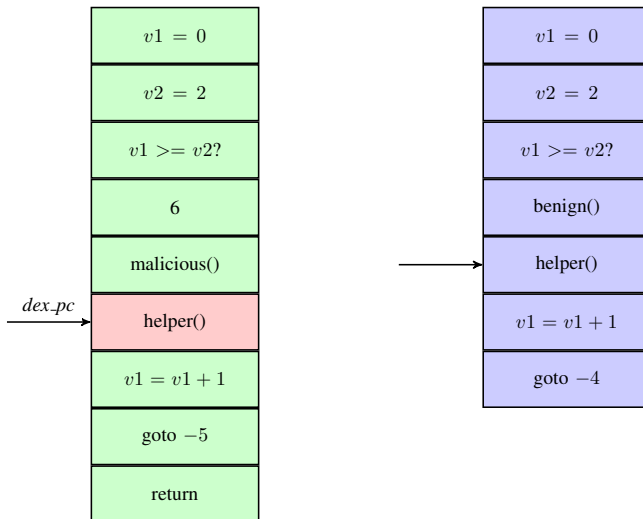
# Branch Instruction



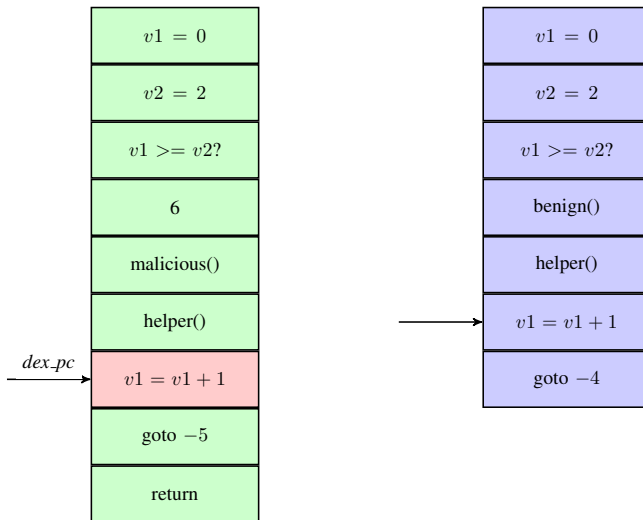
# Branch Instruction



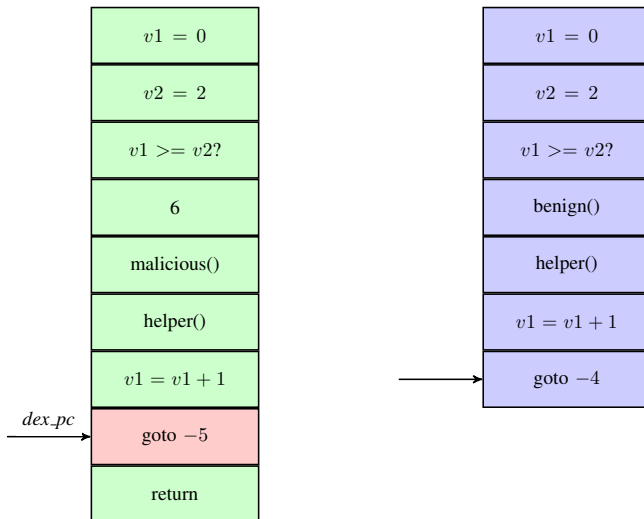
# Branch Instruction



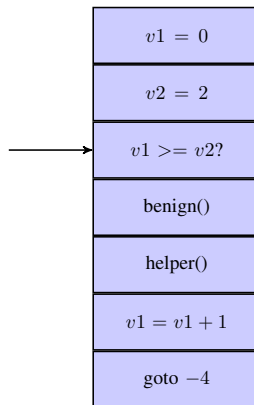
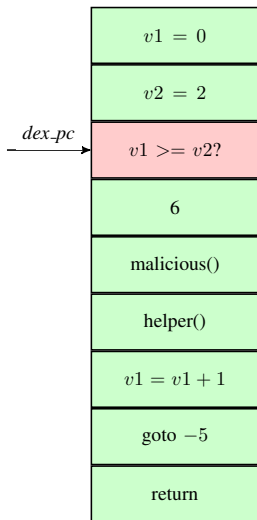
# Branch Instruction



# Branch Instruction

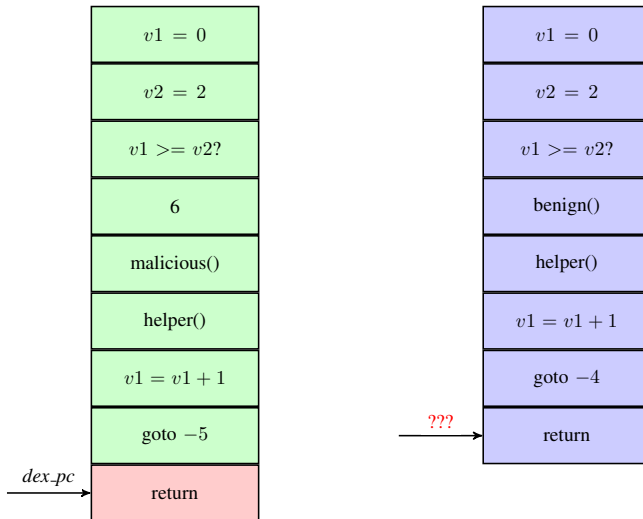


# Branch Instruction

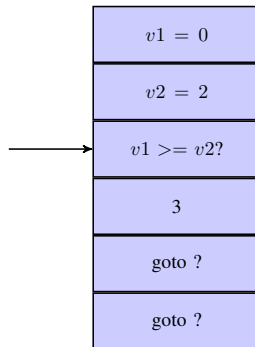
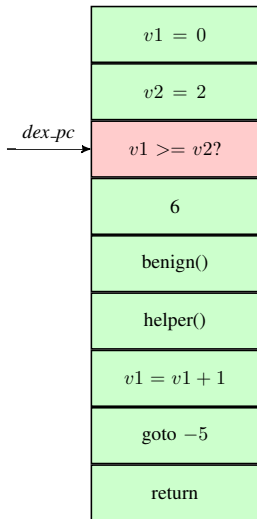




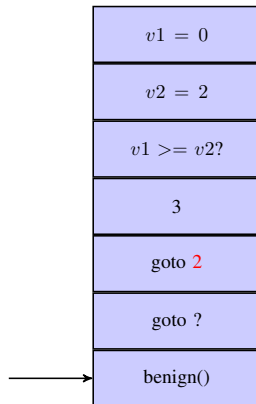
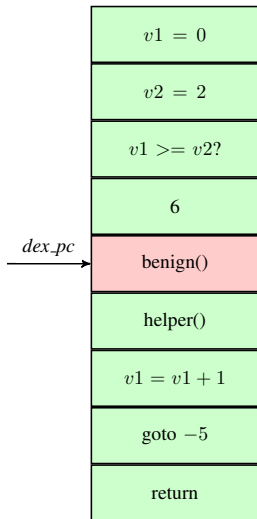
# Branch Instruction



# Branch Instruction



# Branch Instruction



# Branch Instruction

