AppSealer:

Automatic Generation of Vulnerability-Specific Patches for Preventing Component Hijacking Attacks in Android Applications

Mu Zhang

Heng Yin

Department of Electrical Engineering and Computer Science,
Syracuse University

L.C.Smith College of Engineering and Computer Science

Motivation: Component Hijacking Attacks in Android

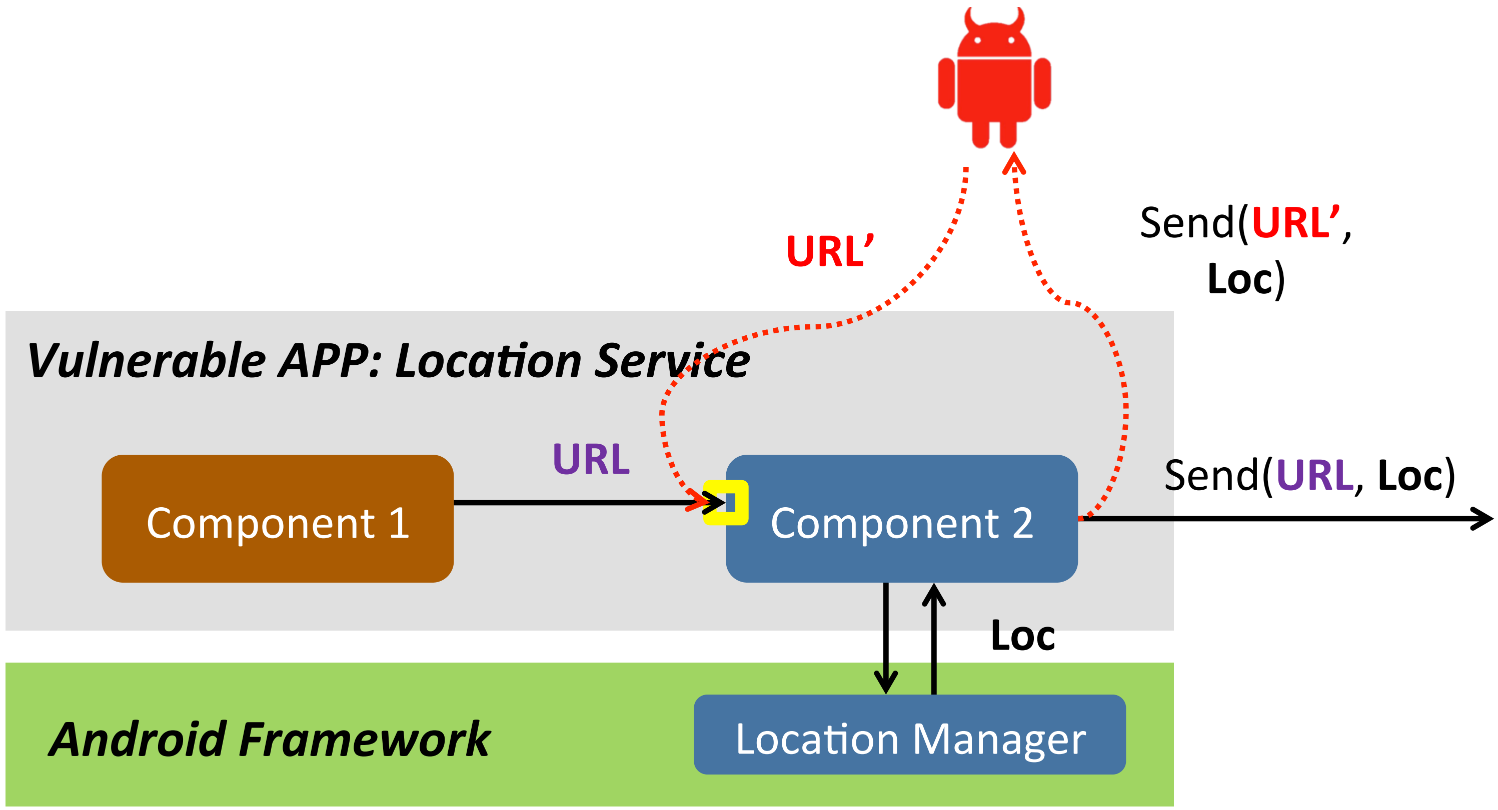
"A class of attacks that seek to

gain **unauthorized access** to protected or private resources

through **exported components** in vulnerable Android apps."

(L. Lu et al. CHEX, *CCS'12*)

Motivation: Component Hijacking Attacks in Android



Motivation: Current Countermeasures

- **Detection: Static Dataflow Analysis**
  - Conservative
- **Fix: Manual Effort**
  - Inexperienced
  - Not easy to confirm vulnerability

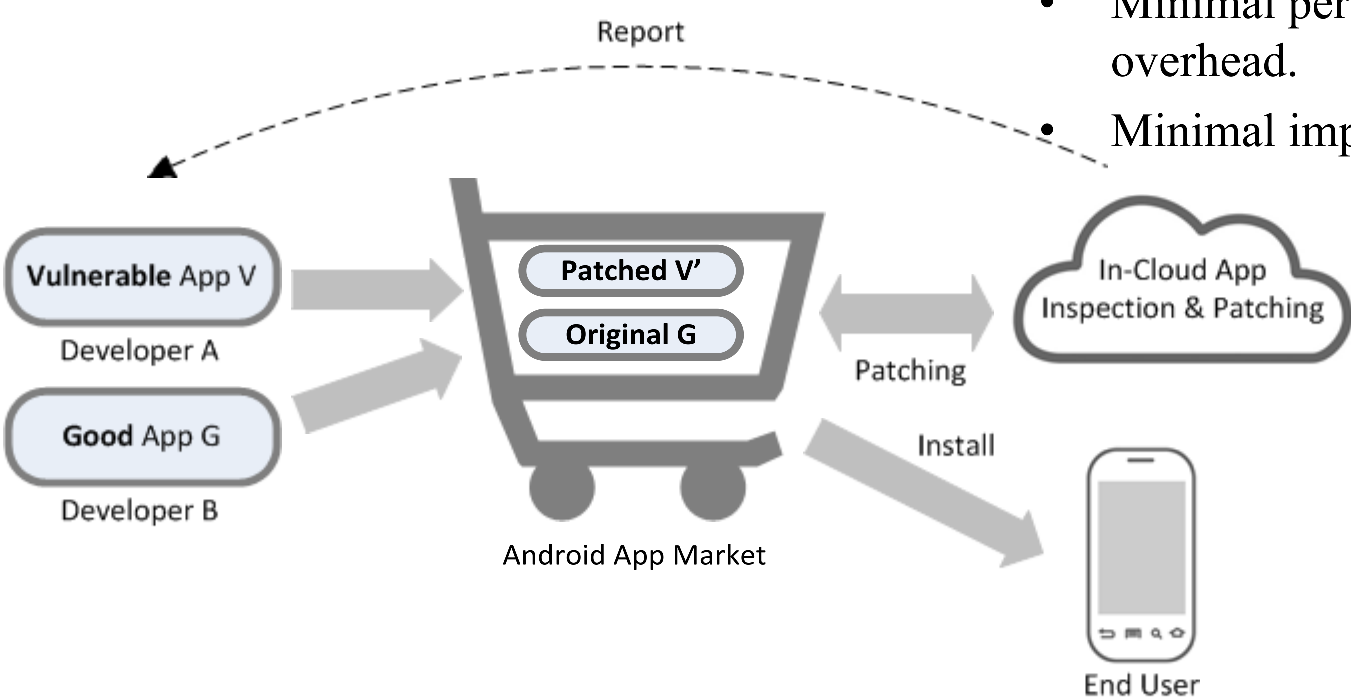
16 Reported in **Oct. 2012**
13 Not fixed until **Aug. 2013**

AppSealer: Automatic Patch Generation

- **Goal:** to automatically generate a patch that is specific to the discovered component hijacking vulnerability.

Design Requirements:

- No source code access.
- Vulnerability-specific patching.
- Minimal performance overhead.
- Minimal impact on usability.

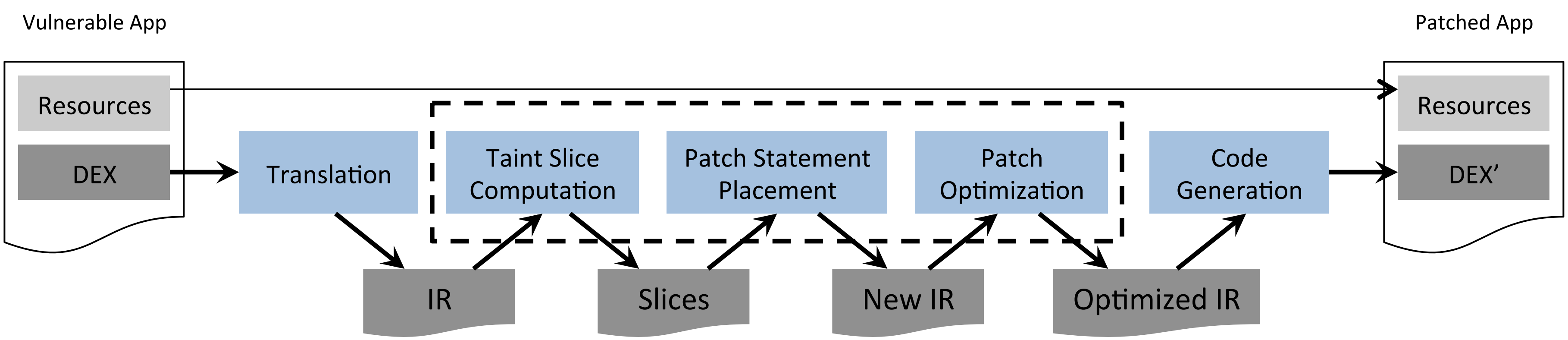


Related Work: Automatic Patch Generation

- **Data Patch**
  - W. Cui et al. ShieldGen, *Oakland'07*
  - D. Brumley et al. *Oakland'06*
  - M. Costa et al. Vigilante, *SOSP'05*
  - M. Costa et al. Bouncer, *SOSP'07*
  - J. Caballero et al. *RAID'09*
- **Code Patch**
  - Z. Lin et al. AutoPaG, *ASIACCS'07*
  - C. Zhang et al. IntPatch, *ESORICS'10*
  - Sidiroglou and Keromytis, *IEEE Security and Privacy*
  - J. Newsome et al. VSEF, *NDSS'06*

Technical Approach

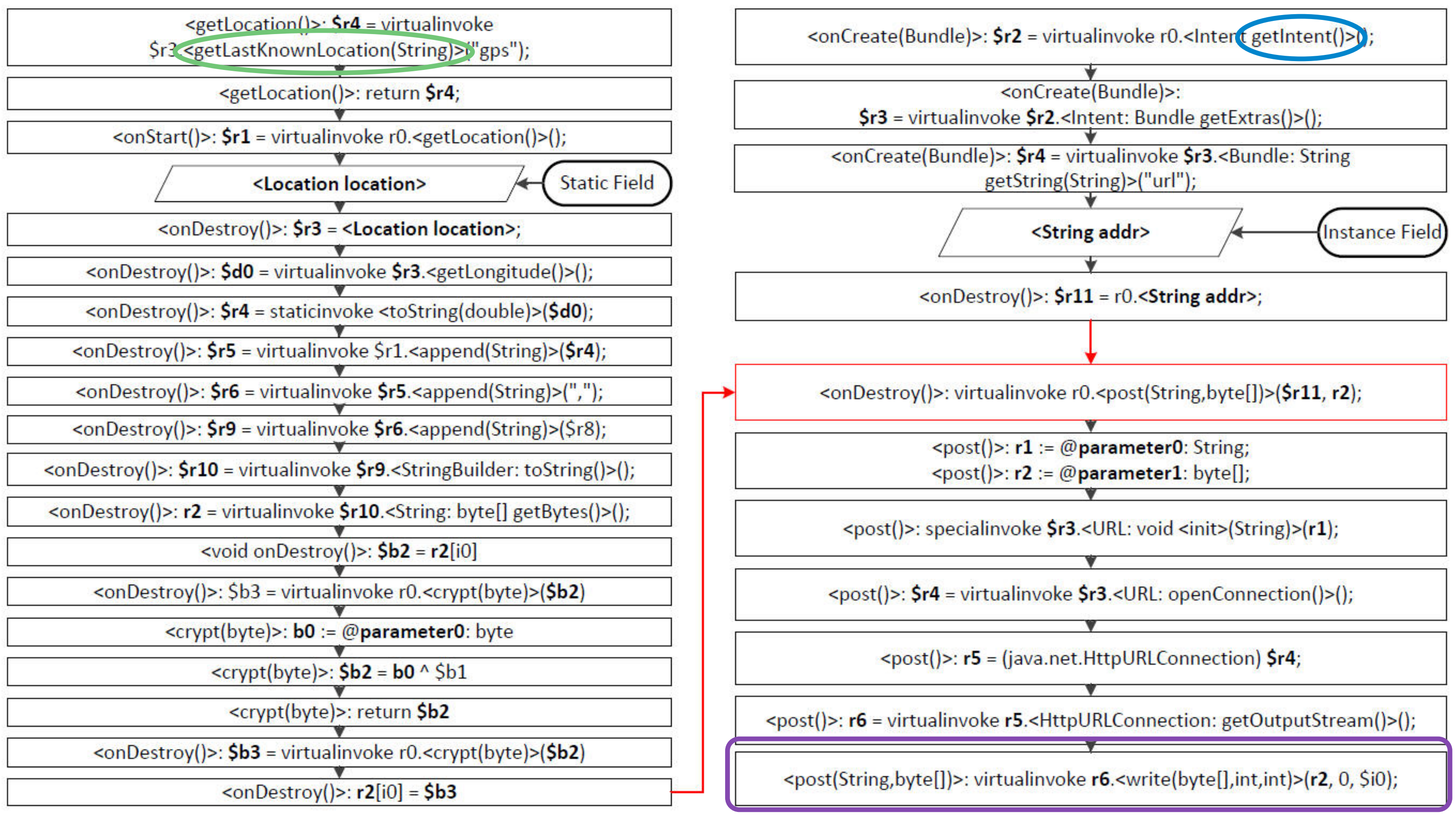
- Key: to place ***minimally*** required code into the vulnerable program to ***accurately*** keep track of dangerous information.



A Running Example

```java
public class VulActivity extends Activity{
  private String DEFAULT_ADDR = "http://default.url";
  private byte DEFAULT_KEY = 127;

  private String addr;
  private static Location location;
  private byte key;

  /* Entry point of this Activity */
  public void onCreate(Bundle savedInstanceState){
    this.key = DEFAULT_KEY;

    this.addr = getIntent().getExtras().getString("url");
    if(this.addr == null){
      this.addr = DEFAULT_ADDR;
    }
  }

  public void onStart(){
    VulActivity.location = getLocation();
  }

  public void onDestroy(){
    String location =
      Double.toString(location.getLongitude())
      + "," + Double.toString(location.getLatitude());
    byte[] bytes = location.getBytes();
    for(int i=0; i<bytes.length; i++)
      bytes[i] = crypt(bytes[i]);
    String url = this.addr;
    post(url, bytes);
  }

  public byte crypt(byte plain){
    return (byte)(plain ^ key);
  }

  public Location getLocation(){
    Location location = null;
    LocationManager locationManager = (LocationManager)
      getSystemService(Context.LOCATION_SERVICE);
    location = locationManager.getLastKnownLocation
      (LocationManager.GPS_PROVIDER);
    return location;
  }

  public void post(String addr, byte[] bytes){
    URL url = new URL(addr);
    HttpURLConnection conn =
      (HttpURLConnection)url.openConnection();
    ...
    OutputStream output = conn.getOutputStream();
    output.write(bytes, 0, bytes.length);
    ...
  }
}
```

Taint Slice Computation



Patch Statement Placement

```
1    public class VulActivity extends Activity{
       ...
5      private String addr;
P      public boolean addr_s0_t;
6      private static Location location;
P      public static boolean location_s1_t;
       ...
10     public void onCreate(Bundle savedInstanceState){
         ...
13       this.addr=getIntent().getExtras().getString("url");
P        if(isExternalIntent()){
P          this.addr_s0_t = true;
P        }else{
P          this.addr_s0_t = false;
P        }
14       if(this.addr == null){
15         this.addr = DEFAULT_ADDR;
P          this.addr_s0_t = false;
16       }
17     }
18
19     public void onStart(){
20       VulActivity.location = getLocation();
P        VulActivity.location_s1_t = true;
21     }
22
```

```
23     public void onDestroy(){
         ...
29       String url = this.addr;
P        BoolWrapper bytes_s1_w = new BoolWrapper();
P        bytes_s1_w.b = VulActivity.location_s1_t;
P        BoolWrapper url_s0_w = new BoolWrapper();
P        url_s0_w.b = this.addr_s0_t;
30       post(url, bytes, url_s0_w, bytes_s1_w);
31     }
       ...
44     public void post(String addr, byte[] bytes,
         BoolWrapper addr_s0_w, BoolWrapper bytes_s1_w){
P        boolean output_s0_t = addr_s0_w.b;
P        boolean bytes_s1_t = bytes_s1_w.b;
         ...
48       OutputStream output = conn.getOutputStream();
P        if(output_s0_t == true && bytes_s1_t == true)
P          promptForUserDecision();
49       output.write(bytes, 0, bytes.length);
50       ...
51     }
52   }
```

Patch Optimization

```
public byte crypt(byte, BoolWrapper, BoolWrapper, BoolWrapper) {
    r0 := @this: VulActivity;
    b0 := @parameter0: byte;

    w_p0 := @parameter1: BoolWrapper;
    w_t := @parameter2: BoolWrapper;
    w_r := @parameter3: BoolWrapper;

    r0_t = w_t.<BoolWrapper: boolean b>;
    b0_t = w_p0.<BoolWrapper: boolean b>;
    $b2_t = w_r.<BoolWrapper: boolean b>;

    $b1 = r0.<VulActivity: byte key>;
    $b2 = b0 ^ $b1;
    $b2_t = b0_t | 0;

    w_t.<BoolWrapper: boolean b> = r0_t;
    w_p0.<BoolWrapper: boolean b> = b0_t;
    w_r.<BoolWrapper: boolean b> = $b2_t;

    return $b2;
}
```

O1: Removing Redundant BoolWrapper Statements

O2: Removing Redundant Function Parameters

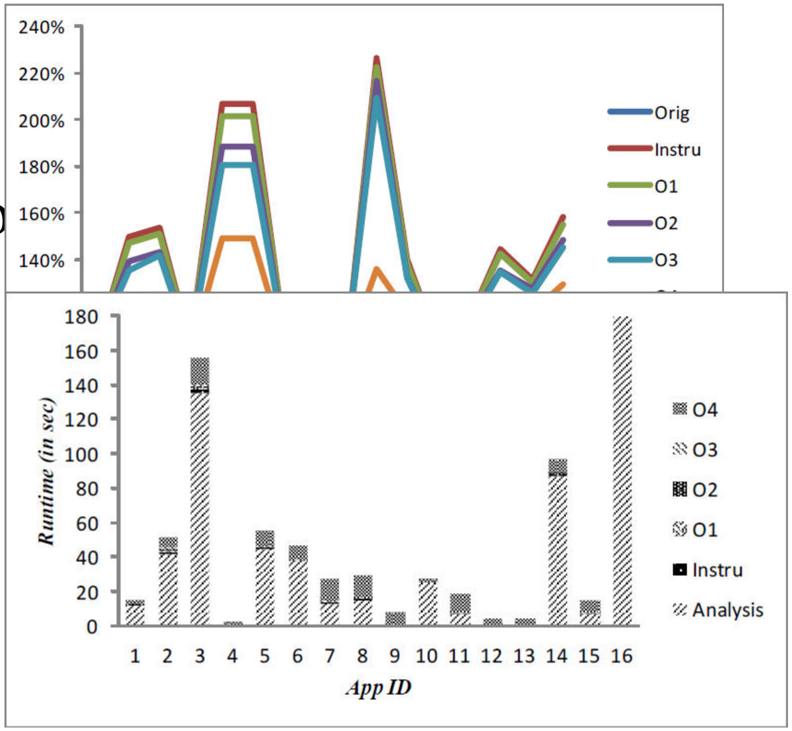
Patch Optimization

```
public byte crypt(byte, BoolWrapper, BoolWrapper) {
    r0 := @this: VulActivity;
    b0 := @parameter0: byte;
    w_p0 := @parameter1: BoolWrapper;
    w_r3 := @parameter2: BoolWrapper;
    $b3 = virtualinvoke r0.<VulActivity: byte crypt(byte)>($b2);
    b0_t = w_p0.<BoolWrapper: boolean b>;
    tmp21 = $b0_t;
    $b1 = r0.<VulActivity: byte key>;
    $b2 = b0 ^ $b1;
    $b2_t = tmp21 | 0;
    w_r3.<BoolWrapper: boolean b> = $b2_t;
    $b3_t = tmp23;
    return $b2;
}
```

O3: Inlining Instrumentation Code
O4: Soot Built-in Optimizations

Evaluation: Overview

- **16 real-world apps with component hijacking vulnerabilities**
  - Increase of Program Size
  - Performance of Patch Generatio
  - Runtime Overhead
    - Average: 2%
    - Worst Case: 9.6%
  - Effectiveness
    - Benign Context: No Interruption
    - Under Attack: Warning



Evaluation: Case Study

- **6** apps with **Pop-up Dialogs**
- **3** apps with **Selection Views**
- **3** apps with Multiple **Threads**

Related Work

[1] W. Cui, M. Peinado, and H. J. Wang, “ShieldGen: Automatic Data Patch Generation for Unknown Vulnerabilities with Informed Probing,” in Proceedings of Oakland’07.

[2] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha, “Towards Automatic Generation of Vulnerability-Based Signatures,” in Proceedings of Oakland’06.

[3] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, “Vigilante: End-to-End Containment of Internet Worms,” in Proceedings of SOSP’05.

[4] M. Costa, M. Castro, L. Zhou, L. Zhang, and M. Peinado, “Bouncer: Securing Software by Blocking Bad Input,” in Proceedings of SOSP’07.

[5] J. Caballero, Z. Liang, Poosankam, and D. Song, “Towards Generating High Coverage Vulnerability-Based Signatures with Protocol-Level Constraint-Guided Exploration,” in Proceedings of RAID’09.

[6] Z. Lin, X. Jiang, D. Xu, B. Mao, and L. Xie, “AutoPaG: Towards automated Software Patch Generation with Source Code Root Cause Identification and Repair,” in Proceedings of ASIACCS’07.

[7] C. Zhang, T. Wang, T. Wei, Y. Chen, and W. Zou, “IntPatch: Automatically Fix Integer-Overflow-to-Buffer-Overflow Vulnerability at Compile-Time,” in Proceedings of ESORICS’10.

[8] S. Sidiroglou and A. D. Keromytis, “Countering Network Worms Through Automatic Patch Generation,” IEEE Security and Privacy, vol. 3, no. 6, pp. 41–49, Nov. 2005.

[9] J. Newsome, D. Brumley, and D. Song, “Vulnerability-specific execution filtering for exploit prevention on commodity software,” in Proceedings of NDSS’06.

Conclusion

- We developed a technique to ***automatically*** generate patch for Android applications with **component hijacking vulnerability**.
- The key is to place ***minimally*** required code into the vulnerable program to ***accurately*** keep track of dangerous information and effectively block the attack at the security sensitive APIs.

Questions?

Discussion

- **Soundness of Patch Generation**
  - Static analysis: standard, FP
  - Taint tracking: taint policy of TaintDroid, FP
  - Optimization: compiler algorithms
  - In theory, FP; In practice, no FP observed.