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
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

Vulnerable APP: Location Service

Component 1 → URL → Component 2

Android Framework

Location Manager

Send(URL', Loc)

Send(URL, Loc)
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**
  – C. Zhang et al. IntPatch, *ESORICS’10*
  – Sidiroglou and Keromytis, *IEEE Security and Privacy*
  – J. Newsome et al. VSEF, *NDSS’06*
• 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(){
            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
public class VulActivity extends Activity{
    ... 
    private String addr;
    public boolean addr_s0_t;
    private static Location location;
    public static boolean location_s1_t;

    public void onCreate(Bundle savedInstanceState){
        this.addr=getIntent().getExtras().getString("url");
        if(isExternalIntent()){
            this.addr_s0_t = true;
        }else{
            this.addr_s0_t = false;
        }
        if(this.addr == null){
            this.addr = DEFAULT_ADDR;
            this.addr_s0_t = false;
        }
    }

    public void onStart(){
        VulActivity.location = getLocation();
        VulActivity.location_s1_t = true;
    }

    public void onDestroy(){
        ... 
    }

    public void post(String addr, byte[] bytes, boolean addr_s0_w, boolean bytes_s1_w){
        boolean output_s0_t = addr_s0_w.b;
        boolean bytes_s1_t = bytes_s1_w.b;
        if(output_s0_t == true & bytes_s1_t == true)
            promptForUserDecision();
        output.write(bytes, 0, bytes.length);
    }

    public void post(String url, byte[] bytes, boolean addr_s0_w, boolean bytes_s1_w){
        String url = this.addr;
        BoolWrapper bytes_s1_w = new BoolWrapper();
        bytes_s1_w.b = VulActivity.location_s1_t;
        BoolWrapper url_s0_w = new BoolWrapper();
        url_s0_w.b = this.addr_s0_t;
        post(url, bytes, url_s0_w, bytes_s1_w);
    }
}
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;
}
Patch Optimization

```java
public byte crypt(byte, BoolWrapper, BoolWrapper) {
    r0 := @this: VulActivity;
    b0 := @parameter0: byte;
    w_p0 := @parameter1: BoolWrapper;
    $b3 := virtualinvoke @VulActivity: byte crypt(byte>($b2);
    w_r = w_p0.<BoolWrapper:boolean b>
    $b2 = b0 ^ $b1;
    $b3<BoolWrapper:boolean b> = $b2_t;
    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 Generation
  – 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


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?
• **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.