Program Analysis for Security

John Mitchell
Software bugs are serious problems

Thanks: Isil and Thomas Dillig
App stores

Apps for whatever you’re up for.

Stay on top of the news. Stay on top of your finances. Or plan your dream vacation. No matter what you want to do with your iPhone, there’s probably an app to help you do it.

Business

iPhone is ready for work. Manage projects, track stocks, monitor finances, and more with these 9-to-5 apps.

View business apps in the App Store

Education

Keep up with your studies using intelligent education apps like King of Math and NatureTap.

View education apps in the App Store

Entertainment

Kick back and enjoy the show. Or find countless other ways to entertain yourself. These apps offer hours of viewing pleasure.

View entertainment apps in the App Store

Family & Kids

Turn every night into family night with interactive apps that are fun for the whole house.

View family and kids apps in the App Store

Finance

Create budgets, pay bills, and more with financial apps that take everything into account.

View finance apps in the App Store

Food & Drink


View food and drink apps in the App Store
App stores

Save Big!
Get 25% off select accessories for a limited time

Our Favorites
Cool Games

Cut the Rope 2
Zeptolab
FREE

Meow Meow Star A
GaloPL, Inc.
FREE

Fairway Solitaire Bi
Big Fish Games
FREE

CSR Classics
NaturalMotionGames L
FREE
How can you tell whether software you
– Buy
– Develop
– Install
is safe to run?
Cost of Fixing a Defect

Credit: Andy Chou, Coverity
Several approaches

• Instrument code for testing
  – Heap memory: Purify
  – Perl tainting (information flow)
  – Java race condition checking

• Black-box testing
  – Fuzzing and penetration testing
  – Black-box web application security analysis

🌟Static code analysis
  – FindBugs, Fortify, Coverity, MS tools, …
Outline

• General discussion of static analysis tools
  – Goals and limitations
  – Approach based on abstract states
• More about one specific approach
  – Property checkers from Engler et al., Coverity
  – Sample security-related results
• Static analysis for of Android apps

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, ...
Manual testing only examines small subset of behaviors.
Program Analyzers

- Code
- Spec

- Program Analyzer

<table>
<thead>
<tr>
<th>Report</th>
<th>Type</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mem leak</td>
<td>324</td>
</tr>
<tr>
<td>2</td>
<td>buffer oflow</td>
<td>4,353,245</td>
</tr>
<tr>
<td>3</td>
<td>sql injection</td>
<td>23,212</td>
</tr>
<tr>
<td>4</td>
<td>stack oflow</td>
<td>86,923</td>
</tr>
<tr>
<td>5</td>
<td>dang ptr</td>
<td>8,491</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10,502</td>
<td>info leak</td>
<td>10,921</td>
</tr>
</tbody>
</table>

- Analyze large code bases
- Potentially reports many warnings
- May emit false alarms
- False alarm
Example

• Coverity demo video: http://youtu.be/_Vt4niZfNeA

• Observations
  – Code analysis integrated into development workflow
  – Program context important: analysis involves sequence of function calls, surrounding statements
  – This is a sales video: no discussion of false alarms
Static analysis goals

• Bug finding
  – Identify code that the programmer wishes to modify or improve

• Correctness
  – Verify the absence of certain classes of errors
## Soundness, Completeness

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundness</td>
<td>If the program contains an error, the analysis will report a warning. “Sound for reporting correctness”</td>
</tr>
<tr>
<td>Completeness</td>
<td>If the analysis reports an error, the program will contain an error. “Complete for reporting correctness”</td>
</tr>
<tr>
<td>Complete</td>
<td>Incomplete</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>Sound</strong></td>
<td><strong>Incomplete</strong></td>
</tr>
<tr>
<td>Reports all errors</td>
<td>Reports all errors</td>
</tr>
<tr>
<td>Reports no false alarms</td>
<td>May report false alarms</td>
</tr>
<tr>
<td><strong>Undecidable</strong></td>
<td><strong>Decidable</strong></td>
</tr>
<tr>
<td><strong>Undecidable</strong></td>
<td><strong>Decidable</strong></td>
</tr>
<tr>
<td>May not report all errors</td>
<td>May not report all errors</td>
</tr>
<tr>
<td>Reports no false alarms</td>
<td>May report false alarms</td>
</tr>
</tbody>
</table>
So#ware

Behaviors

Sound
Over-approximation of Behaviors

Error

False Alarm

approximation is too coarse...

...yields too many false alarms

Software
Outline

• General discussion of tools
  – Goals and limitations
  ➔ Approach based on abstract states

• More about one specific approach
  – Property checkers from Engler et al., Coverity
  – Sample security-related results

• Static analysis for Android malware
  – ...

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, ...
Does this program ever crash?

```
entry

X ← 0

Is Y = 0?

X ← X + 1
X ← X - 1

Is Y = 0?

Is X < 0?

crash

exit
```

Yes or no at each step:

- Is Y = 0? (yes or no)
- Is X < 0? (yes or no)
- Is Y = 0? (yes or no)
- Is Y = 0? (yes or no)
- Is Y = 0? (yes or no)
- Is Y = 0? (yes or no)
- Is Y = 0? (yes or no)
Does this program ever crash?

infeasible path!
... program will never crash
Try analyzing without approximating...

non-termination!
... therefore, need to approximate
\[ d_{\text{out}} = f(d_{\text{in}}) \]
$X = 0$

$X \leftarrow X + 1$

$X = 0$

Is $Y = 0$?

$X = 1$

$X = 1$

$X = 1$

$X = 1$

$d_{in1}$

$f_1$

$d_{out1}$

$d_{in2}$

$d_{in2} = d_{out1}$

$I_{out1} = f_1(d_{in1})$

$d_{out1} = d_{in2}$

$d_{out2} = f_2(d_{in2})$
What is the space of dataflow elements, \( \Delta \)?
What is the least upper bound operator, \( \sqcup \)?

\[
\begin{align*}
d_{out1} &= f_1(d_{in1}) \\
d_{out2} &= f_2(d_{in2}) \\
d_{\text{join}} &= d_{out1} \sqcup d_{out2} \\
d_{\text{join}} &= d_{in3} \\
d_{out3} &= f_3(d_{in3})
\end{align*}
\]
Try analyzing with “signs” approximation...

... but reports false alarm
... therefore, need more precision
Try analyzing with “path-sensitive signs” approximation...

```
X ← 0

Is Y = 0 ?
  yes
  X ← X + 1
  Y = 0
  X = pos
  X ← X ← X - 1
  X = neg
  Y ≠ 0
  no
  X = neg
  Y ≠ 0

Is X < 0 ?
  yes
  crash
  X = pos
  Y = 0
  no
  exit
```

- no precision loss
- refinement

terminates...
... no false alarm
... soundly proved never crashes
Outline

• General discussion of tools
  – Goals and limitations
  – Approach based on abstract states

★ More about one specific approach
  – Property checkers from Engler et al., Coverity
  – Sample security-related results

• Static analysis for Android malware
  – ...

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, ...
Bugs to Detect

Some examples

- Crash Causing Defects
- Null pointer dereference
- Use after free
- Double free
- Array indexing errors
- Mismatched array new/delete
- Potential stack overrun
- Potential heap overrun
- Return pointers to local variables
- Logically inconsistent code

- Uninitialized variables
- Invalid use of negative values
- Passing large parameters by value
- Underallocations of dynamic data
- Memory leaks
- File handle leaks
- Network resource leaks
- Unused values
- Unhandled return codes
- Use of invalid iterators

Slide credit: Andy Chou
Example: Check for missing optional args

- Prototype for open() syscall:
  ```c
  int open(const char *path, int oflag, /* mode_t mode */...);
  ```

- Typical mistake:
  ```c
  fd = open("file", O_CREAT);
  ```

- Result: file has random permissions

- Check: Look for oflags == O_CREAT without mode argument
Example: Chroot protocol checker

• **Goal:** confine process to a “jail” on the filesystem
  - chroot() changes filesystem root for a process

• **Problem**
  - chroot() itself does not change current working directory

```plaintext
chroot()  ->  chdir("/")

Error if open before chdir
```
TOCTOU

• Race condition between time of check and use

• Not applicable to all programs
Tainting checkers

Tainted data accepted from source

Unvetted data taints other data transitively

Tainted data is used in an operator or function

Example Sinks: system() printf() malloc() strcpy() Sent to RDBMS Included in HTML

Resultant Vulnerability: command injection format string manip. integer/ buffer overflow SQL injection cross site scripting
#include <stdlib.h>
#include <stdio.h>

void say_hello(char * name, int size) {
    printf("Enter your name: ");
    fgets(name, size, stdin);
    printf("Hello %s.\n", name);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Error, must provide an input buffer size.\n");
        exit(-1);
    }
    int size = atoi(argv[1]);
    char * name = (char*)malloc(size);
    if (name) {
        say_hello(name, size);
        free(name);
    } else {
        printf("Failed to allocate %d bytes.\n", size);
    }
}
Reverse Topological Sort

Idea: analyze function before you analyze caller
Tool has built-in summaries of library function behavior.
Bottom Up Analysis

Analyze function using known properties of functions it calls
Bottom Up Analysis

Analyze function using known properties of functions it calls
Bottom Up Analysis

Finish analysis by analyzing all functions in the program
#define SIZE 8
void set_a_b(char * a, char * b) {
    char * buf[SIZE];
    if (a) {
        b = new char[5];
    } else {
        if (a && b) {
            buf[SIZE] = a;
            return;
        } else {
            delete [] b;
        }
    }
    *b = 'x';
}
*a = *b;
char * buf[8];

if (a)
   b = new char [5];
if (a && b)
   buf[8] = a;
delete [] b;
*b = 'x';
*a = *b;

Represent logical structure of code in graph form
Path Traversal

char * buf[8];

if (a)

b = new char[5];

if (a && b)

buf[8] = a;

delete [] b;

*a = *b;

END

Conceptually Analyze each path through control graph separately

Actually Perform some checking computation once per node; combine paths at merge nodes
Apply Checking

char * buf[8];

if (a)
!
a
if (a && b)
!(a && b)
delete [] b;

*b = 'x';

*a = *b;

END

Null pointers Use after free Array overrun

See how three checkers are run for this path

Checker
• Defined by a state diagram, with state transitions and error states

Run Checker
• Assign initial state to each program var
• State at program point depends on state at previous point, program actions
• Emit error if error state reached
Apply Checking

Null pointers Use after free Array overrun

char * buf[8];

if (a)

!a

if (a && b)

!(a && b)

delete [] b;

*b = 'x';

*a = *b;

END

“buf is 8 bytes”
char * buf[8];
if (a)
!a
if (a && b)
!(a && b)
delete[] b;
*b = 'x';
*a = *b;
END

Apply Checking

Null pointers Use after free Array overrun

“buf is 8 bytes”

“a is null”
char * buf[8];

if (a)

!a

if (a && b)

!(a && b)

delete [] b;

*b = 'x';

*a = *b;

END
Apply Checking

char * buf[8];

if (a)
!a
if (a && b)
!(a && b)
delete [] b;
*b = 'x';
*a = *b;

END

Null pointers Use after free Array overrun

“buf is 8 bytes”

“a is null”

“b is deleted”
Apply Checking

Null pointers Use after free Array overrun

```
char * buf[8];
if (a)
if (a && b)
delete [] b;
*b = 'x';
*a = *b;
```
char * buf[8];

if (a)

!a

if (a && b)

!(a && b)

delete [] b;

*b = 'x';

*a = *b;

END

Null pointers
Use after free
Array overrun

“buf is 8 bytes”

“a is null”

“b is deleted”

“b dereferenced!”

No more errors reported for b
False Positives

• What is a bug? Something the user will fix.

• Many sources of false positives
  − False paths
  − Idioms
  − Execution environment assumptions
  − Killpaths
  − Conditional compilation
  − “third party code”
  − Analysis imprecision
  − ...

char * buf[8];

if (a) {
    b = new char [5];
    if (a && b) {
        buf[8] = a;
        delete [] b;
        *b = 'x';
        *a = *b;
    }
}

END
char * buf[8];

if (a)

!a

if (a && b)

buf[8] = a;

END
False Path Pruning

```
char * buf[8];
if (a)
  if (a && b)
    buf[8] = a;
```

```
"a in [0,0]"
"a == 0 is true"
```

Integer Range  Disequality  Branch
False Path Pruning

char * buf[8];

if (a)

!a

if (a && b)

a && b

buf[8] = a;

END

“a in [0,0]”

“a == 0 is true”

“a != 0”

“a != 0”
False Path Pruning

```
char * buf[8];
if (a)
  if (a && b)
    buf[8] = a;
END
```

- **Impossible**
  - “a in [0,0]”
  - “a == 0 is true”
  - “a != 0”

**.Integer Range**

**Disequality**

**Branch**
Environment Assumptions

• Should the return value of malloc() be checked?

\[
\text{int } *p = \text{malloc(sizeof(int))}; \\
*p = 42;
\]

OS Kernel: Crash machine.  
File server: Pause filesystem.  
Web application: 200ms downtime

Spreadsheet: Lose unsaved changes.  
Game: Annoy user.  
IP Phone: Annoy user.

Library: ?  
Medical device: malloc?!
• Assume the code is usually right

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
if(p) *p = 42;
```

```
int *p = malloc(sizeof(int));
*p = 42;
if(p) *p = 42;
```

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
if(p) *p = 42;
```

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
```
Application to Security Bugs

• Stanford research project
  − Used modified compiler to find over 100 security holes in Linux and BSD
  − http://www.stanford.edu/~engler/

• Benefit
  − Capture recommended practices, known to experts, in tool available to all
Sanitize integers before use

Warn when unchecked integers from untrusted sources reach trusting sinks

Linux: 125 errors, 24 false; BSD: 12 errors, 4 false
Example security holes

- Remote exploit, no checks

/* 2.4.9/drivers/isdn/act2000/capi.c:actcapi_dispatch */
isdn_ctrl cmd;
...
while ((skb = skb_dequeue(&card->rcvq))) {
    msg = skb->data;
    ...
    memcpy(cmd.parm.setup.phone,
           msg->msg.connect_ind.addr.num,
           msg->msg.connect_ind.addr.len - 1);
Example security holes

• Missed lower-bound check:

```c
/* 2.4.5/drivers/char/drm/i810_dma.c */

if(copy_from_user(&d, arg, sizeof(arg)))
    return -EFAULT;
if(d.idx > dma->buf_count)
    return -EINVAL;
buf = dma->buflist[d.idx];
Copy_from_user(buf_priv->virtual, d.address, d.used);
```
User-pointer inference

- **Problem: which are the user pointers?**
  - Hard to determine by dataflow analysis
  - Easy to tell if kernel *believes* pointer is from user!

- **Belief inference**
  - “*p*” implies safe kernel pointer
  - “copyin(p)/copyout(p)” implies dangerous user ptr
  - Error: pointer p has both beliefs.

- **Implementation: 2 pass checker**
  - inter-procedural: compute all tainted pointers
  - local pass to check that they are not dereferenced
Results for BSD and Linux

- All bugs released to implementers; most serious fixed

<table>
<thead>
<tr>
<th>Violation</th>
<th>Linux Bug Fixed</th>
<th>BSD Bug Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain control of system</td>
<td>18 15</td>
<td>3 3</td>
</tr>
<tr>
<td>Corrupt memory</td>
<td>43 17</td>
<td>2 2</td>
</tr>
<tr>
<td>Read arbitrary memory</td>
<td>19 14</td>
<td>7 7</td>
</tr>
<tr>
<td>Denial of service</td>
<td>17 5</td>
<td>0 0</td>
</tr>
<tr>
<td>Minor</td>
<td>28 1</td>
<td>0 0</td>
</tr>
<tr>
<td>Total</td>
<td>125 52</td>
<td>12 12</td>
</tr>
</tbody>
</table>
Outline

• General discussion of tools
  – Goals and limitations
  – Approach based on abstract states
• More about one specific approach
  – Property checkers from Engler et al., Coverity
  – Sample security-related results

Static analysis for Android malware
  – ...

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, ...
STAMP Admission System

Dynamic Analysis
Fewer behaviors, more details

Static Analysis
More behaviors, fewer details

Alex Aiken,
John Mitchell,
Saswat Anand,
Jason Franklin
Osbert Bastani,
Lazaro Clapp,
Patrick Mutchler,
Manolis Papadakis
Analysis

**Step 1**
Convert bytecode to intermediate format (called Quads)

**Step 2**
Compute call graph using Class Hierarchy Analysis

**Step 3**
Build an edge-labeled graph $G$ by processing Quads of each class

**Step 4**
Add new edges to $G$ as per a set of rules until no rules apply
Data Flow Analysis

Source-to-sink flows
- Sources: Location, Calendar, Contacts, Device ID etc.
- Sinks: Internet, SMS, Disk, etc.
Data Flow Analysis in Action

• Malware/Greyware Analysis
  ○ Data flow summaries enable enterprise-specific policies

• API Misuse and Data Theft Detection

• Automatic Generation of App Privacy Policies
  ○ Avoid liability, protect consumer privacy

• Vulnerability Discovery

Privacy Policy
This app collects your:
- Contacts
- Phone Number
- Address
Challenges

- Android is 3.4M+ lines of complex code
  - Uses reflection, callbacks, native code

- **Scalability**: Whole system analysis impractical

- **Soundness**: Avoid missing flows

- **Precision**: Minimize false positives
STAMP Approach

- Model Android/Java
  - Sources and sinks
  - Data structures
  - Callbacks
  - 500+ models

- Whole-program analysis
  - Context sensitive

Too expensive!
Building Models

- **30k+ methods in Java/Android API**
  - 5 mins x 30k = 2500 hours

- **Follow the permissions**
  - 20 permissions for sensitive sources
    - ACCESS_FINE_LOCATION (8 methods with source annotations)
    - READ_PHONE_STATE - (9 methods)
  - 4 permissions for sensitive sinks
    - INTERNET, SEND_SMS, etc.
Identifying Sensitive Data

- Returns device IMEI in String
- Requires permission GET_PHONE_STATE

```java
android.Telephony.TelephonyManager: String getDeviceId()
```

```java
@STAMP(
    SRC ="$GET_PHONE_STATE.deviceid",
    SINK ="@return"
)
```
Data We Track (Sources)

- Account data
- Audio
- Calendar
- Call log
- Camera
- Contacts
- Device Id
- Location
- Photos (Geotags)
- SD card data
- SMS

30+ types of sensitive data
Data Destinations (Sinks)

- Internet (socket)
- SMS
- Email
- System Logs
- Webview/Browser
- File System
- Broadcast Message

10+ types of exit points
Currently Detectable Flow Types

396 Flow Types

Unique Flow Types = Sources x Sink
Example Analysis

Contact Sync for Facebook (unofficial)

Description:

This application allows you to synchronize your Facebook contacts on Android.

IMPORTANT:

* "Facebook does not allow [sic] to export phone numbers or emails. Only names, pictures and statuses are synced."

* "Facebook users have the option to block one or all apps. If they opt for that, they will be EXCLUDED from your friends list."

Privacy Policy: (page not found)
## Contact Sync Permissions

<table>
<thead>
<tr>
<th>Category</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Your Accounts</strong></td>
<td>AUTHENTICATE_ACCOUNTS</td>
<td>Act as an account authenticator</td>
</tr>
<tr>
<td></td>
<td>MANAGE_ACCOUNTS</td>
<td>Manage accounts list</td>
</tr>
<tr>
<td></td>
<td>USE_CREDENTIALS</td>
<td>Use authentication credentials</td>
</tr>
<tr>
<td><strong>Network Communication</strong></td>
<td>INTERNET</td>
<td>Full Internet access</td>
</tr>
<tr>
<td></td>
<td>ACCESS_NETWORK_STATE</td>
<td>View network state</td>
</tr>
<tr>
<td><strong>Your Personal Information</strong></td>
<td>READ_CONTACTS</td>
<td>Read contact data</td>
</tr>
<tr>
<td></td>
<td>WRITE_CONTACTS</td>
<td>Write contact data</td>
</tr>
<tr>
<td><strong>System Tools</strong></td>
<td>WRITE_SETTINGS</td>
<td>Modify global system settings</td>
</tr>
<tr>
<td></td>
<td>WRITE_SYNC_SETTINGS</td>
<td>Write sync settings (e.g. Contact sync)</td>
</tr>
<tr>
<td></td>
<td>READ_SYNC_SETTINGS</td>
<td>Read whether sync is enabled</td>
</tr>
<tr>
<td></td>
<td>READ_SYNC_STATS</td>
<td>Read history of syncs</td>
</tr>
<tr>
<td><strong>Your Accounts</strong></td>
<td>GET_ACCOUNTS</td>
<td>Discover known accounts</td>
</tr>
<tr>
<td><strong>Extra/Custom</strong></td>
<td>WRITE_SECURE_SETTINGS</td>
<td>Modify secure system settings</td>
</tr>
</tbody>
</table>
Possible Flows from Permissions

Sources
- READ_CONTACTS
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- GET_ACCOUNTS
- INTERNET

Sinks
- INTERNET
- WRITE_SETTINGS
- WRITE_CONTACTS
- WRITE_SECURE_SETTINGS
- WRITE_SETTINGS
Expected Flows

Sources
- READ_CONTACTS
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- GET_ACCOUNTS
- INTERNET

Sinks
- INTERNET
- WRITE_SETTINGS
- WRITE_CONTACTS
- WRITE_SECURE_SETTINGS
- WRITE_SETTINGS
Observed Flows

FB API → Source: FB_Data → Write Contacts → Sink: Contact_Book

Read Contacts → Source: Contacts → Send Internet → Sink: Internet
Example Study: Mobile Web Apps

• Goal
  Identify security concerns and vulnerabilities specific to mobile apps that access the web using an embedded browser

• Technical summary
  • WebView object renders web content
  • methods loadUrl, loadData, loadDataWithBaseUrl, postUrl
  • addJavascriptInterface(obj, name) allows JavaScript code in the web content to call Java object method name.foo()
Summary

• Analyze a dataset of 737,828 Android apps
• Found large number of apps contain severe vulnerabilities
• 37,418 apps are vulnerable to a remote code execution exploit when run on any Android device, because of security oversight in older versions and slow adoption of safe versions
• 45,689 apps are vulnerable to a remote code execution exploit when run on 73% of the in-use Android devices.
• Offer recommendations for developers who wish to avoid these vulnerabilities.
• Severity of exploits in apps that include Bridge Objects and can navigate to content belonging to untrusted origins.
Severity of Bridge Object exploits in apps that load content over HTTP.
• Prevalence of apps that can or must ignore SSL errors based on implementations of `onReceivedSslError`
• Severity of Bridge Object exploits in apps that must ignore SSL certificate errors.
Lecture Summary

• General discussion of tools
  – Fundamental limitations
  – Basic method based on abstract states

• More details on one specific method
  – Property checkers from Engler et al., Coverity
  – Sample security-related results

• Static analysis for Android malware
  – ...

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken,