Program Analysis for Security

John Mitchell

Acknowledgments: Lecture slides are from the Computer Security course taught by Dan Boneh at Stanford University. When slides are obtained from other sources, a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.
MOTIVATION FOR PROGRAM ANALYZERS
Software bugs are serious problems

Thanks: Isil and Thomas Dillig
Facebook missed a single security check…

Man Finds Easy Hack to Delete Any Facebook Photo Album

Facebook awards him a $12,500 "bug bounty" for his discovery
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.

LinkedIn Business

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

View business apps in the App Store

Google Education

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

View education apps in the App Store

HBO GO 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

Apple 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 financial apps in the App Store

Food & Drink


View food and drink apps in the App Store
How can you tell whether software you
– Develop
– Buy is safe to install and run?
Manual testing only examines a small subset of behaviors.
Program Analyzers

<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>
Cost of Fixing a Defect

Credit: Andy Chou, Coverity
Cost of security or data privacy vulnerability?
Two options

• Static analysis
  – Inspect code or run automated method to find errors or gain confidence about their absence

• Dynamic analysis
  – Run code, possibly under instrumented conditions, to see if there are likely problems
Static vs Dynamic Analysis

- **Static**
  - Consider all possible inputs (in summary form)
  - Find bugs and vulnerabilities
  - Can prove absence of bugs, in some cases

- **Dynamic**
  - Need to choose sample test input
  - Can find bugs/vulnerabilities
  - Cannot prove their absence
Static Analysis

- Long research history
- Decade of commercial products
  - FindBugs, Fortify, Coverity, MS tools, ...
- Main topic for this lecture
Dynamic analysis

• 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

• Will come back to later in course
Summary

• Program analyzers
  – Find problems in code before it is shipped to customers or before you install and run it

• Static analysis
  – Analyze code to determine behavior on all inputs

• Dynamic analysis
  – Choose some sample inputs and run code to see what happens
STATIC ANALYSIS
Static Analysis: 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 checkers results
• Static analysis for of Android apps

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

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

- Correctness
  - Verify the absence of certain classes of errors
Sound Program Analyzer

- Code
- Spec

Program Analyzer

Sound: may report many warnings

May emit false false alarms

Analyze large code bases

<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>

May emit false alarms
Over-approximation of Behaviors

Reported Error

approximation is too coarse...
...yields too many false alarms

False Alarm

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 ?

exit

crash

```
Does this program ever crash?

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

non-termination!  
... therefore, need to approximate
\[
X = 0 \quad \rightarrow \quad X \leftarrow X + 1 \\
X = 1
\]

Dataflow equation:
\[
d_{out} = f(d_{in})
\]
\( X = 0 \)

\[ X \leftarrow X + 1 \]

\( X = 1 \)

\( X = 1 \)

\( X = 1 \)

\( \text{Is } Y = 0 ? \)

\( d_{\text{in1}} \)

\[ d_{\text{out1}} = f_1(d_{\text{in1}}) \]

\( d_{\text{out2}} = f_2(d_{\text{in2}}) \)

\( d_{\text{out1}} = d_{\text{out2}} \)

\( d_{\text{in2}} = d_{\text{out1}} \)
Try analyzing with “signs” approximation...

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

entry

$X \leftarrow 0$

Is $Y = 0$?

$X \leftarrow X + 1$

$X \leftarrow X - 1$

Is $Y = 0$?

Is $X < 0$?

exit

crash

true

$X = 0$

$Y = 0$

$X = 0$

$X = \text{pos}$

$X = \text{neg}$

$X = \text{pos}$

$Y = 0$

$Y \neq 0$

$X = \text{pos}$

$Y = 0$

$Y \neq 0$

$X = \text{neg}$

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,
Unsound Program Analyzer

- Mem leak: 324
- Buffer overlow: 4,353,245
- Sql injection: 23,212
- Stack overlow: 86,923
- Danger ptr: 8,491

Program Analyzer Spec

- May emit false alarms
- May emit false alarms
- Anayze large code bases

Not sound: may miss some bugs
Demo

• Coverity 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
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: 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

```
chroot()
chdir("/")
open("..\file",...)
```
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
Finding Local Bugs

#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;

END
Path Traversal

char * buf[8];

if (a)

b = new char [5];

if (a && b)

buf[8] = a;

delete [] b;

*b = 'x';

*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
  !a && b
    delete [] b;
    *b = 'x';
    *a = *b;
END

“buf is 8 bytes”

“a is null”

Null pointers  Use after free  Array overrun
char * buf[8];

if (a)
    if (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”
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!”
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
char * buf[8];

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

END

“a in [0,0]”

“a == 0 is true”
False Path Pruning

char * buf[8];

if (a)

!a

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

Impossible

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

Branch

Integer Range

Disequality
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
  – Reducing false positive using circumstantial evidence
  – Sample security-related results
• Static analysis for Android malware
  – ...

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

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

```c
int *p = 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?!
Statistical Analysis

- Assume the code is usually right

```c
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));
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));
if(p) *p = 42;

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

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

```c
/* 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);
```
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</td>
<td>3</td>
</tr>
<tr>
<td>Corrupt memory</td>
<td>43</td>
<td>2</td>
</tr>
<tr>
<td>Read arbitrary memory</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>Denial of service</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>Minor</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>125</td>
<td>12</td>
</tr>
</tbody>
</table>

60
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

Static Analysis
More behaviors, fewer details

Dynamic Analysis
Fewer behaviors, more details

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

- **getLoc()**
  - Source: Location
  - sendSMS()
  - Sink: SMS
- **sendInet()**
  - Sink: Internet

- **Location** → **SMS**
- **Location** → **Internet**

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

• 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
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

To expensive!

App

Android

OS

HW
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)
# Contact Sync Permissions

<table>
<thead>
<tr>
<th>Category</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your Accounts</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>Network Communication</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>Your Personal Information</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>System Tools</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>Your Accounts</td>
<td>GET_ACCOUNTS</td>
<td>Discover known accounts</td>
</tr>
<tr>
<td>Extra/Custom</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, loadDataWithURL, postUrl
  • addJavascriptInterface(obj, name) allows JavaScript code in the web content to call Java object method name.foo()
Sample results

Analyze 998,286 free web apps from June 2014

<table>
<thead>
<tr>
<th>Mobile Web App Feature</th>
<th>% Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript Enabled</td>
<td>97</td>
</tr>
<tr>
<td>JavaScript Bridge</td>
<td>36</td>
</tr>
<tr>
<td>shouldOverrideUrlLoading</td>
<td>94</td>
</tr>
<tr>
<td>shouldInterceptRequest</td>
<td>47</td>
</tr>
<tr>
<td>onReceivedSslError</td>
<td>27</td>
</tr>
<tr>
<td>postUrl</td>
<td>2</td>
</tr>
<tr>
<td>Custom URL Patterns</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vuln</th>
<th>% Relevant</th>
<th>% Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsafe Navigation</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>Unsafe Retrieval</td>
<td>40</td>
<td>56</td>
</tr>
<tr>
<td>Unsafe SSL</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>Exposed POST</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Leaky URL</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>
Summary

• Static vs dynamic analyzers
• General properties of static analyzers
  – 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
  – STAMP method, sample studies

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