10 common programming mistakes that make you vulnerable to attack

Elisa Heymann
Computer Architecture and Operating Systems Department
Universitat Autònoma de Barcelona
Elisa.Heymann@uab.es

Barton P. Miller
Jim Kupsch
Computer Sciences Department
University of Wisconsin
bart@cs.wisc.edu

Condor Week
Madison May 3, 2012

This research funded in part by Department of Homeland Security grant FA8750-10-2-0030 (funded through AFRL). Past funding has been provided by NATO grant CLG 983049, National Science Foundation grant OCI-0844219, the National Science Foundation under contract with San Diego Supercomputing Center, and National Science Foundation grants CNS-0627501 and CNS-0716460.
Things That We All Know

• All software has vulnerabilities
• Critical infrastructure software is complex and large
• Vulnerabilities can be exploited by both authorized users and by outsiders
• Programmers must be security-aware
  – Designing for security and the use of secure practices and standards does not guarantee security… but helps
What do we do

• **Assess Middleware:** Make cloud/grid software more secure
• **Train:** We teach tutorials for users, developers, sys admins, and managers
• **Research:** Make in-depth assessments more automated and improve quality of automated code analysis

1. Buffer overflow
char id[8];
int validId = 0; /* not valid */

gets(id); /* reads "evillogin"*/

/* validId is now 110 decimal */
if (IsValid(id)) validId = 1; /* not true */
if (validId) /* is true */
{ DoPrivilegedOp(); } /* gets executed */
2. Numeric Errors
Integer Vulnerabilities

• Description
  – Many programming languages allow silent loss of integer data without warning due to
    • Overflow
    • Truncation
    • Signed vs. unsigned representations
  – Code may be secure on one platform, but silently vulnerable on another, due to different underlying integer types.

• General causes
  – Not checking for overflow
  – Mixing integer types of different ranges
  – Mixing unsigned and signed integers
Numeric Parsing
Unreported Errors

• `atoi`, `atol`, `atof`, `scanf` family (with `%u`, `%i`, `%d`, `%x` and `%o` specifiers)
  – Out of range values results in unspecified behavior
  – Non-numeric input returns 0
3. Race Conditions
File System Race Examples

• Check properties of a file then open
  Bad: access or stat \(\rightarrow\) open
  Safe: open \(\rightarrow\) fstat

• Create file if it doesn’t exist
  Bad: if stat fails \(\rightarrow\) creat(fn, mode)
  Safe: open(fn, O_CREAT|O_EXCL, mode)

• http://www.cs.wisc.edu/mist/safefile
Race Condition Examples

• Your Actions

```c
s = strdup("/tmp/zXXXXXX")
tempnam(s)
// s now "/tmp/zRANDOM"

f = fopen(s, "w+")
// writes now update
// /etc/passwd
```

• Attackers Action

```c
link = "/etc/passwd"
file = "/tmp/zRANDOM"
symlink(link, file)
```
4. Exceptions
Exception Suppression

1. User sends malicious data

```
boolean Login(String user, String pwd){
    boolean loggedIn = true;
    String realPwd = GetPwdFromDb(user);
    try {
        if (!GetMd5(pwd).equals(realPwd)) {
            loggedIn = false;
        }
    } catch (Exception e) {
        //this can not happen, ignore
    }
    return loggedIn;
}
```

user="admin",pwd=null

2. System grants access

Login() returns true
5. Too much information
WTMI (Way Too Much Info)

```java
Login(... user, ... pwd) {
    try {
        ValidatePwd(user, pwd);
    } catch (Exception e) {
        print("Login failed.\n");
        print(e + "\n");
        e.printStackTrace();
        return;
    }
}

void ValidatePwd(... user, ... pwd)
    throws BadUser, BadPwd {
    realPwd = GetPwdFromDb(user);
    if (realPwd == null)
        throw BadUser("user=" + user);
    if (!pwd.equals(realPwd))
        throw BadPwd("user=" + user + " pwd=" + pwd + " expected=" + realPwd);
    ...
}
```
6. Directory Traversal
Successful Directory Traversal Attack

1. Users requests
   File="....///etc/passwd"

```java
String path = request.getParameter("file");
path = "/safedir/" + path;
// remove ../'s to prevent escaping out of /safedir
Replace(path, "/", "/");
File f = new File(path);
f.delete();
```

2. Server deletes
   /etc/passwd

Before Replace  path = "/safedir/....///etc/passwd"
After Replace   path = "/safedir/..//etc/passwd"
7. SQL Injection Attacks
Successful SQL Injection Attack

• Dynamically generated SQL without validation or quoting is vulnerable

```perl
$u = " ' ; drop table t --";
$sth = $dbh->do("select * from t where u = '$u'" );
```

Database sees two statements:

```
select * from t where u = ' ' ; drop table t --'
```
Successful SQL Injection Attack

1. User sends malicious data
   user="admin"; pwd="'OR 'x'='x"

2. DB Queried
   SELECT * FROM members
   WHERE u='admin' AND p=''' OR 'x'='x'

3. Returns all row of table members

4. System grants access
   Login() returns true
8. Command Injections
Successful OS Injection Attack

1. User sends malicious data
   
   ```java
   hostname="x.com;rm -rf /*"
   ```

2. Application uses `nslookup` to get DNS records

   ```java
   String rDomainName(String hostname) {
       ...
       String cmd = "/usr/bin/nslookup " + hostname;
       Process p = Runtime.getRuntime().exec(cmd);
       ...
   }
   ```

3. System executes `nslookup x.com;rm -rf /*`

4. All files possible are deleted
9. Code Injection
**Code Injection Vulnerability**

1. logfile – name's value is user controlled

   ```perl
   name = John Smith
   name = '\');import os;os.system('evilprog');#
   ```

2. Perl log processing code – uses Python to do real work

   ```perl
   %data = ReadLogFile('logfile');
   PH = open("|/usr/bin/python");
   print PH "import LogIt\n";
   while ( ($k, $v) = (each %data) ) {
     if ($k eq 'name') {
       print PH "LogIt.Name('$v')";
     }
   }
   ```

3. Python source executed – 2nd LogIt executes arbitrary code

   ```python
   import LogIt;
   LogIt.Name('John Smith')
   LogIt.Name('');import os;os.system('evilprog');#
   ```
10. Web Attacks
Reflected Cross Site Scripting (XSS)

1. Browser sends request to web server

http://example.com?q=widget

2. Web server code handles request

```java
...  
String query = request.getParameter("q");
if (query != null) {
    out.writeln("You searched for:\n" + query);
}
...  
```

3. Generated HTML displayed by browser

```html
<html>
...
You searched for:
widget
...
</html>
```
Reflected Cross Site Scripting (XSS)

1. Browser sends request to web server

http://example.com?q=<script>alert('Boo!')</script>

2. Web server code handles request

```java
...
String query = request.getParameter("q");
if (query != null) {
    out.writeln("You searched for:
" + query);
}
...
```

3. Generated HTML displayed by browser

```html
<html>
...
You searched for:
<script>alert('Boo!')</script>
...
</html>
```
Would you like an expanded tutorial taught at your site?

Tutorials for users, developers, administrators and managers:

– Security Risks
– Secure Programming
– Vulnerability Assessment

Contact us!

Barton P. Miller       Elisa Heymann
bart@cs.wisc.edu     Elisa.Heymann@uab.es
Questions?

http://www.cs.wisc.edu/mist