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

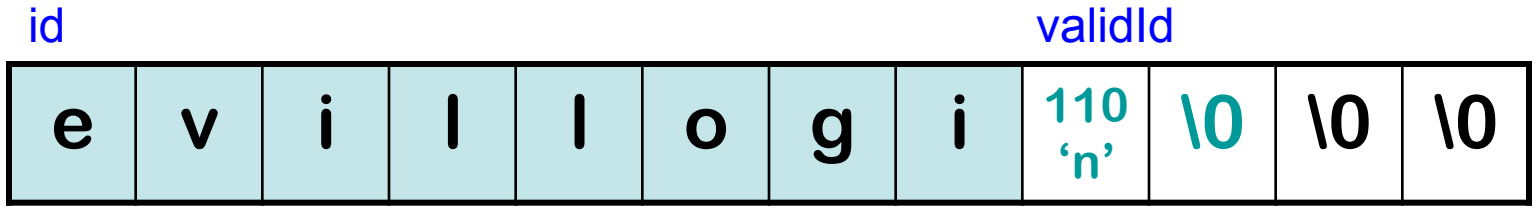
http://www.cs.wisc.edu/mist/papers/VAshort.pdf

1. Buffer overflow

Buffer Overflow of User Data Affecting Flow of Control


```
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

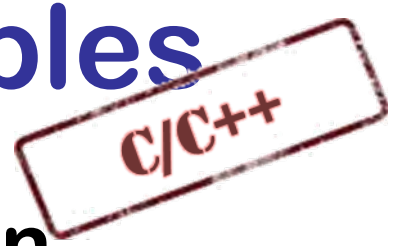
- Description
  - Many programming languages allow silent loss of integer data without warning due to
    - Overflow
    - Truncation
    - Signed vs. unsigned representations
  - Code may be secure on one platform, but silently vulnerable on another, due to different underlying integer types.
- General 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` → `open`
  - Safe: `open` → `fstat`
- Create file if it doesn't exist
  - Bad: if `stat` fails → `creat(fn, mode)`
  - Safe: `open(fn, O_CREAT|O_EXCL, mode)`

- http://www.cs.wisc.edu/mist/safefile
  James A. Kupsch and Barton P. Miller, "How to Open a File and Not Get Hacked," *2008 Third International Conference on Availability, Reliability and Security* (ARES), Barcelona, Spain, March 2008.

Race Condition Examples

C/C++

- **Your Actions**

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


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

time

Attackers Action

```
link = "/etc/passwd"
file = "/tmp/zRANDOM"
symlink(link, file)
```

4. Exceptions

Exception Suppression

1. User sends malicious data | `user="admin",pwd=null`

```
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;
}
```

2. System grants access | `Login() returns true`

5. Too much information

WTMI (Way Too Much Info)


```
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);
…
```

User exists

Entered pwd

```
Login failed.
BadPwd: user=bob pwd=x expected=password
BadPwd:
 at Auth.ValidatePwd (Auth.java:92)
 at Auth.Login (Auth.java:197)
 …
 com.foo.BadFramework(BadFramework.java:71)
 …
```

User's actual password ?!?
(passwords aren't hashed)

Reveals internal structure
(libraries used, call structure, version information)

6. Directory Traversal

Successful Directory Traversal Attack

JAVA

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

```
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

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

Database sees <u>two</u> statements:

```
select * from t where u = ' '; drop table t --'
```

Successful SQL Injection Attack

2. DB Queried

```
SELECT * FROM members
WHERE u='admin' AND p='' OR 'x'='x'
```

3. Returns all row of table members

JAVA

1. User sends malicious data

```
user="admin"; pwd="'OR 'x'='x"
```

```
boolean Login(String user, String pwd)  {
    boolean loggedIn = false;
    conn = pool.getConnection( );
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT * FROM members"
                           + "WHERE u='" + user
                           + "' AND p='" + pwd + "'");
    if (rs.next())
        loggedIn = true;
}
```

4. System grants access

```
Login() returns true
```

8. Command Injections

Successful OS Injection Attack

JAVA

1. User sends malicious data

```
hostname="x.com;rm -rf /*"
```

2. Application uses nslookup to get DNS records

```
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

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

Read logfile

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

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

Start Python, program sent on stdin

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

```
import LogIt;
LogIt.Name('John Smith')
LogIt.Name('');import os;os.system('evilprog');#')
```

10. Web Attacks

Reflected Cross Site Scripting (XSS)

3. Generated HTML displayed by browser

```
<html>
...
You searched for:
widget
...
</html>
```

1. Browser sends request to web server

```
http://example.com?q=widget
```

2. Web server code handles request

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

Reflected Cross Site Scripting (XSS)

3. Generated HTML displayed by browser

```
<html>
…
You searched for:
<script>alert('Boo!')</script>
…
</html>
```

1. Browser sends request to web server

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

2. Web server code handles request

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

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
bart@cs.wisc.edu

Elisa Heymann
Elisa.Heymann@uab.es

Questions?

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