Secure Coding Practices for Middleware

Barton P. Miller
James A. Kupsch
Computer Sciences Department
University of Wisconsin

bart@cs.wisc.edu

Elisa Heymann
Computer Architecture and
Operating Systems Department
Universitat Autònoma de Barcelona

Elisa.Heymann@uab.es

Condor Week 2012
Madison May 1, 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.

Who we are

Bart Miller
Jim Kupsch
Karl Mazurak
Daniel Crowell
Wenbin Fang
Henry Abbey

Elisa Heymann
Eduardo Cesar
Jairo Serrano
Guifré Ruiz
Manuel Brugnoli

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

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

Our experience

Condor, University of Wisconsin
Batch queuing workload management system
15 vulnerabilities — 600 KLOC of C and C++

SRB, SDSC
Storage Resource Broker - data grid
5 vulnerabilities — 280 KLOC of C

MyProxy, NCSA
Credential Management System
5 vulnerabilities — 25 KLOC of C

glExec, Nikhef
Identity mapping service
5 vulnerabilities — 48 KLOC of C

Gratia Condor Probe, FNAL and Open Science Grid
Feeds Condor Usage into Gratia Accounting System
3 vulnerabilities — 1.7 KLOC of Perl and Bash

Condor Quill, University of Wisconsin
DBMS Storage of Condor Operational and Historical Data
6 vulnerabilities — 7.9 KLOC of C and C++

Our experience

Wireshark, wireshark.org
Network Protocol Analyzer
2 vulnerabilities — 2400 KLOC of C

Condor Privilege Separation, Univ. of Wisconsin
Restricted Identity Switching Module
2 vulnerabilities — 21 KLOC of C and C++

VOMS Admin, INFN
Web management interface to VOMS data
4 vulnerabilities — 35 KLOC of Java and PHP

CrossBroker, Universitat Autònoma de Barcelona
Resource Mgr for Parallel & Interactive Applications
4 vulnerabilities — 97 KLOC of C++

ARGUS 1.2, HIP, INFN, NIKHEF, SWITCH
gLite Authorization Service
0 vulnerabilities — 42 KLOC of Java and C

Our experience

VOMS Core INFN
Virtual Organization Management System
1 vulnerability 161 KLOC of Bourne Shell, C++ and C

iRODS, DICE
Data-management System
9 vulnerabilities (and counting) 285 KLOC of C and C++

Google Chrome, Google
Web browser
in progress 2396 KLOC of C and C++

WMS, INFN
Workload Management System
in progress 728 KLOC of Bourne Shell, C++, C, Python, Java, and Perl

Who funds us

- United States
  - DHS
  - NSF
- European Commission
  - EGI
  - EMI
- Spanish Government
- NATO

Roadmap

- Introduction
- Handling errors
- Pointers and Strings
- Numeric Errors
- Race Conditions
- Exceptions
- Privilege, Sandboxing and Environment
- Injection Attacks
- Web Attacks
- Bad things

Discussion of the Practices

- Description of vulnerability
- Signs of presence in the code
- Mitigations
- Safer alternatives

Handling Errors

- If a call can fail, always check for errors
  optimistic error handling (i.e. none) is bad
- Error handling strategies:
  - Handle locally and continue
  - Cleanup and propagate the error
  - Exit the application
- All APIs you use or develop, that can fail, ***must*** be able to report errors to the caller
- Using exceptions forces error handling

Pointers and Strings

Buffer Overflows

http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.html#Listing

1. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
2. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
3. Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
4. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
5. Missing Authentication for Critical Function
6. Missing Authorization
7. Use of Hard-coded Credentials
8. Missing Encryption of Sensitive Data
9. Unrestricted Upload of File with Dangerous Type
10. Reliance on Untrusted Inputs in a Security Decision

Buffer Overflows

- Description
  - Accessing locations of a buffer outside the boundaries of the buffer
- Common causes
  - C-style strings
  - Array access and pointer arithmetic in languages without bounds checking
  - Off by one errors
  - Fixed large buffer sizes (make it big and hope)
  - Decoupled buffer pointer and its size
    - If size unknown overflows are impossible to detect
    - Require synchronization between the two
    - Ok if size is implicitly known and every use knows it (hard)

Why Buffer Overflows are Dangerous

- An overflow overwrites memory adjacent to a buffer
- This memory could be
  - Unused
  - Code
  - Program data that can affect operations
  - Internal data used by the runtime system
- Common result is a crash
- Specially crafted values can be used for an attack

Buffer Overflow of User Data Affecting Flow of Control

C/C++

```
char id[8];
int  validId = 0;    /* not valid */
```

id								validId			
								\0	\0	\0	\0

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

id								validId			
e	v	i	l	l	o	g	i	110 ‘n’	\0	\0	\0

```
/* validId is now 110 decimal */
if (IsValid(id)) validId = 1; /* not true */
if (validId)                  /* is true  */
    {DoPrivilegedOp();}    /* gets executed */
```

Buffer Overflow Danger Signs:
Missing Buffer Size

C/C++

- `gets`, `getpass`, `getwd`, and `scanf` family (with `%s` or `%[…]` specifiers without width)
  - Impossible to use correctly: size comes solely from user input
  - Source of the first (1987) stack smash attack.
  - Alternatives:

Unsafe	Safer
`gets(s)`	`fgets(s, sLen, stdin)`
`getcwd(s)`	`getwd(s, sLen)`
`scanf("%s", s)`	`scanf("%100s", s)`

strcat, strcpy, sprintf, vsprintf

C/C++

- Impossible for function to detect overflow
  - Destination buffer size not passed
- Difficult to use safely w/o pre-checks
  - Checks require destination buffer size
  - Length of data formatted by printf
  - Difficult & error prone
  - Best incorporated in a safe replacement function

Proper usage: concat s1, s2 into dst

```
If (dstSize < strlen(s1) + strlen(s2) + 1)
      {ERROR("buffer overflow");}
strcpy(dst, s1);
strcat(dst, s2);
```

Buffer Overflow Danger Signs:
Difficult to Use and Truncation

C/C++

- `strncat(dst, src, n)`
  - *n* is the maximum number of chars of *src* to append (trailing null also appended)
  - ***can overflow if*** `n>=(dstSize-strlen(dst))`
- `strncpy(dst, src, n)`
  - Writes *n* chars into *dst*, if `strlen(src)<n`, it fills the other `n-strlen(src)` chars with 0's
  - If `strlen(src)>=n`, *dst* is not null terminated
- **Truncation detection not provided**
- **Deceptively insecure**
  - Feels safer but requires same careful use as `strcat`

Safer String Handling:
C-library functions

C/C++

- **snprintf**(buf, bufSize, fmt, …) **and** **vsnprintf**
  - Returns number of bytes, **not including \0 that would've been written.**
  - Truncation detection possible (**result >= bufSize** implies truncation)
  - Use as safer version of **strcpy** and **strcat**

Proper usage: concat s1, s2 into dst

```
r = snprintf(dst, dstSize, "%s%s",s1, s2);
If (r >= dstSize)
      {ERROR("truncation");}
```

THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

NATO OTAN

NSF

C11 and ISO/IEC TR 24731

Extensions for the C library: Part 1, Bounds Checking Interface

- Functions to make the C library safer
- Meant to easily replace existing library calls with little or no other changes
- Aborts on error or optionally reports error
- Very few unspecified behaviors
- All updated buffers require a size param
- http://www.open-std.org/jtc1/sc22/wg14

Stack Smashing

- This is a buffer overflow of a variable local to a function that corrupts the internal state of the run-time system
- Target of the attack is the value on the stack to jump to when the function completes
- Can result in arbitrary code being executed
- Not trivial, but not impossible either

Pointer Attacks

- First, overwrite a pointer
  - In the code
  - In the run-time environment
    - Heap attacks use the pointers usually at the beginning and end of blocks of memory
- Second, the pointer is used
  - Read user controlled data that causes a security violation
  - Write user controlled data that later causes a security violation

Attacks on Code Pointers

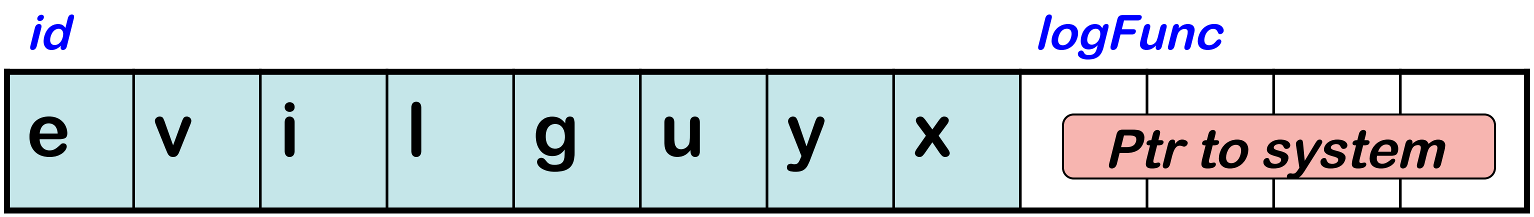
- Stack Smashing is an example
- There are many more pointers to functions or addresses in code
  - Dispatch tables for libraries
  - Return addresses
  - Function pointers in code
  - C++ vtables
  - `jmp_buf`
  - `atexit`
  - Exception handling run-time
  - Internal heap run-time data structures

Buffer Overflow of a User Pointer

C/C++

```
{
  char id[8];
  int  (*logFunc)(char*) = MyLogger;
```


```
  gets(id);      /* reads "evilguyx Ptr to system */
```




```
  /* equivalent to system(userMsg) */
  logFunc(userMsg);
```

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

Integer Danger Signs

- Mixing signed and unsigned integers
- Converting to a smaller integer
- Using a built-in type instead of the API's typedef type
- However built-ins can be problematic too: `size_t` is unsigned, `ptrdiff_t` is signed
- Assigning values to a variable of the correct type before data validation (range/size check)

Numeric Parsing Unreported Errors

C/C++

- `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
  - Use `strtol`, `strtoul`, `strtoll`, `strtoull`, `strtof`, `strtod`, `strtold` which allow error detection

Race Conditions

Race Conditions

- Description
  - A race condition occurs when multiple threads of control try to perform a non-atomic operation on a shared object, such as
    - Multithreaded applications accessing shared data
    - Accessing external shared resources such as the file system
- General causes
  - Threads or signal handlers without proper synchronization
  - Non-reentrant functions (may have shared variables)
  - Performing non-atomic sequences of operations on shared resources (file system, shared memory) and assuming they are atomic

File System Race Conditions

- A file system maps a path name of a file or other object in the file system, to the internal identifier (device and inode)
- If an attacker can control any component of the path, multiple uses of a path can result in different file system objects
- Safe use of path
  - eliminate race condition
    - use only once
    - use file descriptor for all other uses
  - verify multiple uses are consistent

File System Race Examples

C/C++

- 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)`
  - Never use `O_CREAT` without `O_EXCL`
  - Better still use safefile library
    - 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 Temporary Files

- Temporary directory (`/tmp`) is a dangerous area of the file system
  - Any process can create a directory entry there
  - Usually has the sticky bit set, so only the owner can delete their files
- Ok to create *true temporary files* in `/tmp`
  - Create using `mkstemp`, `unlink`, access through returned file descriptor
  - Storage vanishes when file descriptor is closed
- Safe use of `/tmp` directory
  - create a secure directory in `/tmp`
  - use it to store files

Race Condition Examples

C/C++

- **Your Actions** — time — **Attackers Action**

```
s=strdup("/tmp/zXXXXXX")
tempnam(s)
// s now "/tmp/zRANDOM"
                                link = "/etc/passwd"
                                file = "/tmp/zRANDOM"
                                symlink(link, file)
f = fopen(s, "w+")
// writes now update
// /etc/passwd
```

Safe Version

```
fd = mkstemp(s)
f = fdopen(fd, "w+")
```

The University of Wisconsin Madison — Universitat Autònoma de Barcelona — U.S. Department of Homeland Security — NATO OTAN — NSF

Successful Race Condition Attack

```
void TransFunds(srcAcct, dstAcct, xfrAmt)  {
  if (xfrAmt < 0)
    FatalError();
  int srcAmt = srcAcct.GetBal();
  if (srcAmt - xfrAmt < 0)
    FatalError();
  srcAcct.SetBal(srcAmt - xfrAmt);
  dstAcct.SetBal(dstAcct.getBal() + xfrAmt);
}
```

Thread 1	Thread 2	Balances Bob	Ian
XfrFunds(Bob, Ian, 100)	XfrFunds(Bob, Ian, 100)	100	0
srcAmt = 100			
	srcAmt = 100		
srcAmt – 100 < 0 ?			
	srcAmt – 100 < 0 ?		
srcAcct.SetBal(100 – 100)		0	
	srcAcct.SetBal(100 – 100)	0	
dst.SetBal(0 + 100)			100
	dst.SetBal(0 + 100)		200

Mitigated Race Condition Attack

```
void synchronized TransFunds(srcAcct, dstAcct, xfrAmt)  {
  if (xfrAmt < 0)
    FatalError();
  int srcAmt = srcAcct.GetBal();
  if (srcAmt - xfrAmt < 0)
    FatalError();
  srcAcct.SetBal(srcAmt - xfrAmt);
  dstAcct.SetBal(dstAcct.getBal() + xfrAmt);
}
```

JAVA

	time		Balances	
Thread 1		Thread 2	Bob	Ian
XfrFunds(Bob, Ian, 100)		XfrFunds(Bob, Ian, 100)	100	0
In use? No, proceed				
		In use? Yes, wait.		
srcAmt = 100				
srcAmt – 100 < 0 ?				
srcAcct.SetBal(100 – 100)			0	
dst.SetBal(0 + 100)				100
		srcAmt = 0		
		srcAmt – 100 < 0? Yes, fail		

THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

NATO OTAN

NSF

Exceptions

Exception Vulnerabilities

- Exception are a nonlocal control flow mechanism, usually used to propagate error conditions in languages such as Java and C++.

```
try {
    // code that generates exception
} catch (Exception e) {
   // perform cleanup and error recovery
}
```

- Common Vulnerabilities include:
  - Ignoring (program terminates)
  - Suppression (catch, but do not handled)
  - Information leaks (sensitive information in error messages)

Proper Use of Exceptions

- Add proper exception handling
  - Handle expected exceptions (i.e. check for errors)
  - Don’t suppress:
    - Do not catch just to make them go away
    - Recover from the error or rethrow exception
  - Include top level exception handler to avoid exiting: catch, log, and restart
- Do not disclose sensitive information in messages
  - Only report non-sensitive data
  - Log sensitive data to secure store, return id of data
  - Don't report unnecessary sensitive internal state
    - Stack traces
    - Variable values
    - Configuration data

Exception Suppression

JAVA

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`

Unusual or Exceptional Conditions Mitigation

JAVA

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) {
        loggedIn = false;
    }
     return loggedIn;
}
```

2. System does not grant access `Login() returns false`

THE UNIVERSITY of WISCONSIN MADISON Universitat Autònoma de Barcelona

NATO OTAN NSF

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);
…
```

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)

The Right Amount of Information

JAVA

```
Login {
   try {
      ValidatePwd(user, pwd);
   } catch (Exception e) {
      logId = LogError(e);  // write exception and return log ID.
      print("Login failed, username or password is invalid.\n");
      print("Contact support referencing problem id " + logId
              + " if the problem persists");
      return;
  }
}
```

Log sensitive information

Generic error message
(id links sensitive information)

```
void ValidatePwd(… user, … pwd) throws BadUser, BadPwd  {
 realPwdHash = GetPwdHashFromDb(user)
 if (realPwdHash == null)
   throw BadUser("user=" + HashUser(user));
 if (!HashPwd(user, pwd).equals(realPwdHash))
   throw BadPwdExcept("user=" + HashUser(user));
 …
}
```

User and password are hashed
(minimizes damage if breached)

Privilege, Sandboxing, and Environment

Not Dropping Privilege

- Description
  - When a program running with a privileged status (running as root for instance), creates a process or tries to access resources as another user
- General causes
  - Running with elevated privilege
  - Not dropping all inheritable process attributes such as uid, gid, euid, egid, supplementary groups, open file descriptors, root directory, working directory
  - not setting close-on-exec on sensitive file descriptors

Not Dropping Privilege: `chroot`

- `chroot` changes the root directory for the process, files outside cannot be accessed
- Only root can use `chroot`
- `chdir` needs to follow `chroot`, otherwise relative pathnames are not restricted
- Need to recreate all support files used by program in new root: `/etc`, libraries, ...

Makes `chroot difficult to use.`

Insecure Permissions

- Set `umask` when using `mkstemp` or `fopen`
  - File permissions need to be secure from creation to destruction
- Don't write sensitive information into insecure locations (directories need to have restricted permission to prevent replacing files)
- Executables, libraries, configuration, data and log files need to be write protected

Insecure Permissions

- If a file controls what can be run as a privileged, users that can update the file are equivalent to the privileged user

  File should be:
  - Owned by privileged user, or
  - Owned by administrative account
    - No login
    - Never executes anything, just owns files
- DBMS accounts should be granted minimal privileges for their task

Trusted Directory

- A trusted directory is one where only trusted users can update the contents of anything in the directory or any of its ancestors all the way to the root
- A trusted path needs to check all components of the path including symbolic links referents for trust
- A trusted path is immune to TOCTOU attacks from untrusted users
- This is extremely tricky to get right!
- safefile library
  - http://www.cs.wisc.edu/mist/safefile
  - Determines trust based on trusted users & groups

Directory Traversal

- Description
  - When user data is used to create a pathname to a file system object that is supposed to be restricted to a particular set of paths or path prefixes, but which the user can circumvent
- General causes
  - Not checking for path components that are empty, "." or ".."
  - Not creating the canonical form of the pathname (there is an infinite number of distinct strings for the same object)
  - Not accounting for symbolic links

Directory Traversal Mitigation

- Use `realpath` or something similar to create canonical pathnames
- Use the canonical pathname when comparing filenames or prefixes
- If using prefix matching to check if a path is within directory tree, also check that the next character in the path is the directory separator or `'\0'`

Directory Traversal
(Path Injection)

- User supplied data is used to create a path, and program security requires but does not verify that the path is in a particular subtree of the directory structure, allowing unintended access to files and directories that can compromise the security of the system.
  - Usually <*program-defined-path-prefix*> + "/" + <*user-data*>

<*user-data*>	Directory Movement
../	up
./ or empty string	none
<dir>/	down

- Mitigations
  - Validate final path is in required directory using canonical paths (realpath)
  - Do not allow above patterns to appear in user supplied part (if symbolic links exists in the safe directory tree, they can be used to escape)
  - Use chroot or other OS mechanisms

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"

Moral: Don't try to *fix* user input, verify and reject instead

Mitigated Directory Traversal

JAVA

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

```
String file = request.getParameter("file");
if (file.length() == 0) {
    throw new PathTraversalException(file + " is null");
}
File prefix = new File(new File("/safedir").getCanonicalPath());
File path = new File(prefix, file);
if(!path.getAbsolutePath().equals(path.getCanonicalPath())){
    throw new PathTraversalException(path + " is invalid");
}
path.getAbsolutePath().delete();
```

2. Throws error `/safedir/../etc/passwd is invalid`

Command Line

- Description
  - Convention is that `argv[0]` is the path to the executable
  - Shells enforce this behavior, but it can be set to anything if you control the parent process
- General causes
  - Using `argv[0]` as a path to find other files such as configuration data
  - Process needs to be setuid or setgid to be a useful attack

Environment

- List of (name, value) string pairs
- Available to program to read
- Used by programs, libraries and runtime environment to affect program behavior
- Mitigations:
  - Clean environment to just safe names & values
  - Don’t assume the length of strings
  - Avoid PATH, LD_LIBRARY_PATH, and other variables that are directory lists used to look for execs and libs

Injection Attacks

Injection Attacks

- Description
  - A string constructed with user input, that is then interpreted by another function, where the string is not parsed as expected
    - Command injection (in a shell)
    - Format string attacks (in printf/scanf)
    - SQL injection
    - Cross-site scripting or XSS (in HTML)
- General causes
  - Allowing metacharacters
  - Not properly neutralizing user data if metacharacters are allowed

SQL Injections

- User supplied values used in SQL command must be validated, quoted, or prepared statements must be used
- Signs of vulnerability
  - Uses a database mgmt system (DBMS)
  - Creates SQL statements at run-time
  - Inserts user supplied data directly into statement without validation

SQL Injections:
attacks and mitigations

PERL

- Dynamically generated SQL without validation or quoting is vulnerable

```
$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 --'
```

- Use *prepared statements* to mitigate

```
$sth = $dbh->do("select * from t where u = ?", $u);
```

  - SQL statement template and value sent to database
  - No mismatch between intention and use

THE UNIVERSITY of WISCONSIN MADISON · Universitat Autònoma de Barcelona · NATO OTAN · NSF

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

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
```

Mitigated SQL Injection Attack

`SELECT * FROM members WHERE u =` $?_1$ `AND p =` $?_2$

$?_1$ `= "admin"` $?_2$ `= "' OR 'x'='x"`

2. DB Queried

3. Returns null set

JAVA

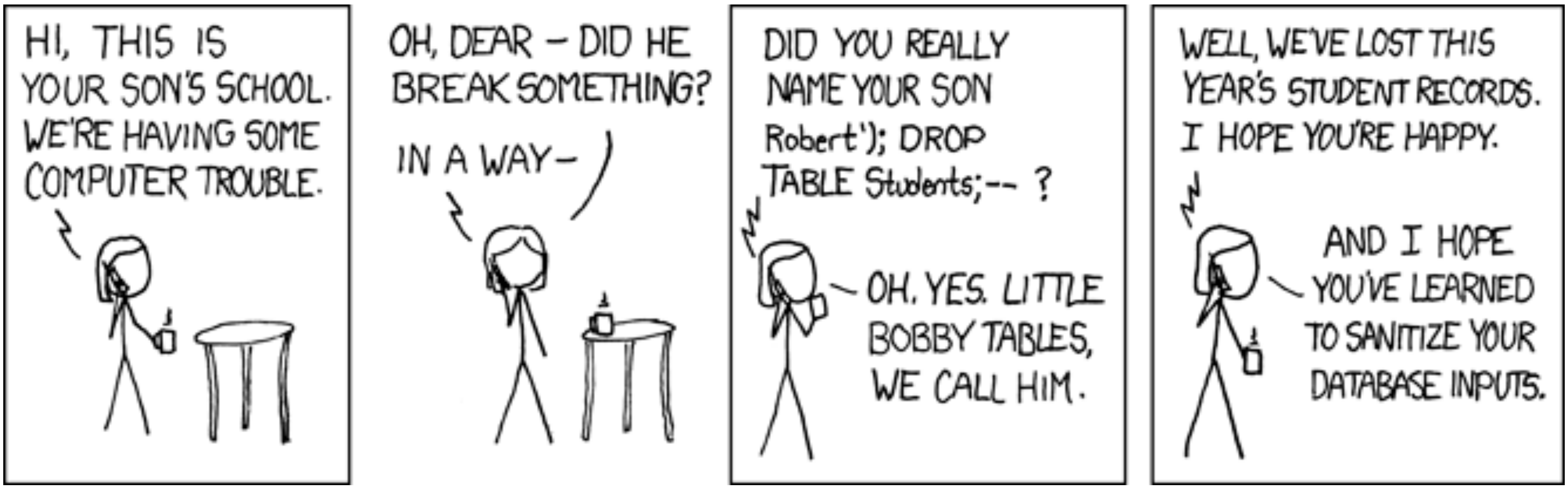
1. User sends malicious data

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

```
boolean Login(String user, String pwd)  {
    boolean loggedIn = false;
    conn = pool.getConnection( );
    PreparedStatement pstmt = conn.prepareStatement(
        "SELECT * FROM members WHERE u = ? AND p = ?");
    pstmt.setString( 1, user);
    pstmt.setString( 2, pwd);
    ResultSet results = pstmt.executeQuery( );
    if (rs.next())
        loggedIn = true;
}
```

4. System does not grant access

`Login() returns false`



http://xkcd.com/327

Command Injections

- User supplied data used to create a string that is the interpreted by command shell such as `/bin/sh`
- Signs of vulnerability
  - Use of `popen`, or `system`
  - `exec` of a shell such as `sh`, or `csh`
  - Argument injections, allowing arguments to begin with `"-"` can be dangerous
- Usually done to start another program
  - That has no C API
  - Out of laziness

Command Injection Mitigations

- Check user input for metacharacters
- Neutralize those that can't be eliminated or rejected
  - replace single quotes with the four characters, `'\''`, and enclose each argument in single quotes
- Use `fork`, drop privileges and `exec` for more control
- Avoid if at all possible
- Use C API if possible

Command Argument Injections

- A string formed from user supplied input that is used as a command line argument to another executable
- Does not attack shell, attacks command line of program started by shell
- Need to fully understand command line interface
- If value should not be an option
  - Make sure it doesn't start with a -
  - Place after an argument of -- if supported

Command Argument Injection Example

C/C++

- **Example**

```
snprintf(s, sSize, "/bin/mail -s hi %s", email);
M = popen(s, "w");
fputs(userMsg, M);
pclose(M);
```

- If email is `-I` , turns on interactive mode …
- … so can run arbitrary code by if userMsg includes: `~!cmd`

Perl Command Injection Danger Signs

PERL

- `open(F, $filename)`
  - Filename is a tiny language besides opening
    - Open files in various modes
    - Can start programs
    - `dup` file descriptors
  - If `$filename` is `"rm -rf /|"`, you probably won't like the result
  - Use separate mode version of open to eliminate vulnerability

Perl Command Injection Danger Signs

PERL

- Vulnerable to shell interpretation

```
open(C, "$cmd|")          open(C, "-|", $cmd)
open(C, "|$cmd")          open(C, "|-", $cmd)
`$cmd`                    qx/$cmd/
system($cmd)
```

- Safe from shell interpretation

```
open(C, "-|", @argList)
open(C, "|-", @cmdList)
system(@argList)
```

Perl Command Injection Examples

PERL

- `open(CMD, "|/bin/mail -s $sub $to");`
  - Bad if `$to` is `"badguy@evil.com; rm -rf /"`
- `open(CMD, “|/bin/mail -s '$sub' '$to'");`
  - Bad if `$to` is `"badguy@evil.com'; rm -rf /'"`
- ```
  ($qSub = $sub) =~ s/'/'\\''/g;
  ($qTo = $to)    =~ s/'/'\\''/g;
  open(CMD, "|/bin/mail -s '$qSub' '$qTo'");
  ```
  - Safe from command injection
- `open(cmd, "|-", "/bin/mail", "-s", $sub, $to);`
  - Safe and simpler: use this whenever possible.
  ```

Eval Injections

PERL

- A string formed from user supplied input that is used as an argument that is interpreted by the language running the code
- Usually allowed in scripting languages such as Perl, sh and SQL
- In Perl `eval($s)` and `s/$pat/$replace/ee`
  - `$s` and `$replace` are evaluated as perl code

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

Mitigated OS Injection Attack

JAVA

1. User sends malicious data

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

2. Application uses nslookup only if input validates

```
String rDomainName(String hostname)  {
  …
  if (hostname.matches("[A-Za-z][A-Za-z0-9.-]*")) {
    String cmd = "/usr/bin/nslookup " + hostname);
    Process p = Runtime.getRuntime().exec(cmd);
  } else {
    System.out.println(“Invalid host name”);
    …
```

3. System returns error

```
"Invalid host name"
```

Format String Injections

C/C++

- User supplied data used to create format strings in `scanf` or `printf`
- `printf(userData)` is insecure
  - `%n` can be used to write memory
  - large field width values can be used to create a denial of service attack
  - Safe to use `printf("%s", userData)` or `fputs(userData, stdout)`
- `scanf(userData, ...)` allows arbitrary writes to memory pointed to by stack values
- ISO/IEC 24731 does not allow `%n`

THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

NATO OTAN

NSF

Code Injection

- Cause
  - Program generates source code from template
  - User supplied data is injected in template
  - Failure to neutralized user supplied data
    - Proper quoting or escaping
    - Only allowing expected data
  - Source code compiled and executed
- Very dangerous – high consequences for getting it wrong: arbitrary code execution

Code Injection Vulnerability

PYTHON PERL

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');#')
```

THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

NATO OTAN

NSF

Code Injection Mitigated

PYTHON

PERL

1. logfile – name's value is user controlled

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

2. Perl log processing code – use QuotePyString to safely create string literal

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

```
sub QuotePyString  {
  my $s = shift;
  $s =~ s/\\/\\\\/g;     # \  → \\
  $s =~ s/'/\\'/g;       # '  → \'
  $s =~ s/\n/\\n/g;      # NL → \n
  return "'$s'";         # add quotes
}
```

3. Python source executed – 2nd LogIt is now safe

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

Web Attacks

Cross Site Scripting (XSS)

- Injection into an HTML page
  - HTML tags
  - JavaScript code
- Reflected (from URL) or persistent (stored from prior attacker visit)
- Web application fails to neutralize special characters in user supplied data
- Mitigate by preventing or encoding/escaping special characters
- Special characters and encoding depends on context
  - HTML text
  - HTML tag attribute
  - HTML URL

Reflected Cross Site Scripting (XSS)

JAVA

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);
}
…
```

THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

NATO OTAN

Reflected Cross Site Scripting (XSS)

JAVA

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);
}
…
```

XSS Mitigation

JAVA

3. Generated HTML displayed by browser

```
<html>
…
Invalid query
…
</html>
```

1. Browser sends request to web server

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

2. Web server code correctly handles request

```
…
String query = request.getParameter("q");
if (query != null) {
    if (query.matches("^\\w*$"))  {
        out.writeln("You searched for:\n" + query);
    } else  {
        out.writeln("Invalid query");
    }
}
…
```

Cross Site Request Forgery (CSRF)

- CSRF is when loading a web pages causes a malicious request to another server
- Requests made using URLs or forms (also transmits any cookies for the site, such as session or auth cookies)
  - http://bank.com/xfer?amt=1000&toAcct=joe HTTP GET method
  - HTTP POST method

```
<form action=/xfer method=POST>
  <input type=text name=amt>
  <input type=text name=toAcct>
</form>
```

- Web application fails to distinguish between a user initiated request and an attack
- Mitigate by using a large random nonce

Cross Site Request Forgery (CSRF)

1. User loads bad page from web server
   - XSS
   - Bad guy’s server
   - Fake server
   - Compromised server
2. Web browser makes a request to the victim web server directed by bad page
   - Tags such as
     <img src=‘http://bank.com/xfer?amt=1000&toAcct=evil37’>
   - JavaScript
3. Victim web server processes request and assumes request from browser is valid
   - Session IDs in cookies are automatically sent along

SSL does not help – channel security is not an issue here

Successful CSRF Attack

JAVA

1. User visits evil.com

```
http://evil.com
```

2. evil.com returns HTML

```
<html>
…
<img src='http://bank.com/xfer?amt=1000&toAcct=evil37'>
…
</html>
```

3. Browser sends attack

```
http://bank.com/xfer?amt=1000&toAcct=evil37
```

4. bank.com server code handles request

```
…
String id = response.getCookie("user");
userAcct = GetAcct(id);
If (userAcct != null)  {
    deposits.xfer(userAcct, toAcct, amount);
}
```

CSRF Mitigation

JAVA

1. User visits evil.com

2. evil.com returns HTML

3. Browser sends attack

Very unlikely attacker will provide correct nonce

4. bank.com server code correctly handles request

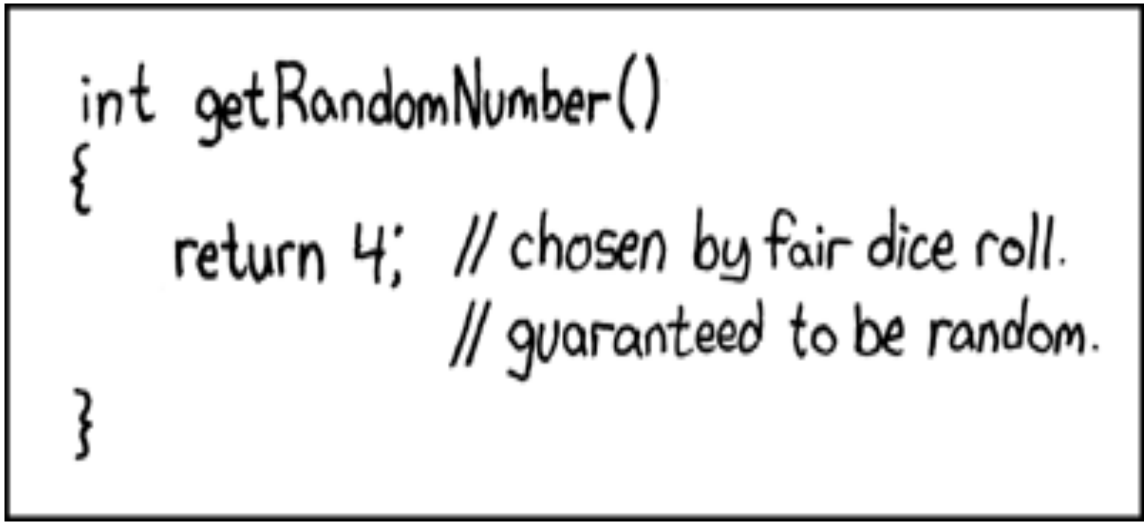
```
...
String nonce = (String)session.getAttribute("nonce");
String id = response.getCookie("user");
if (Utils.isEmpty(nonce)
      || !nonce.equals(getParameter("nonce")  {
    Login();  // no nonce or bad nonce, force login
    return;   //    do NOT perform request
}             // nonce added to all URLs and forms
userAcct = GetAcct(id);
if (userAcct != null)  {
    deposits.xfer(userAcct, toAcct, amount);
}
```

THE UNIVERSITY of WISCONSIN MADISON

Session Hijacking

- Session IDs identify a user's session in web applications.
- Obtaining the session ID allows impersonation
- Attack vectors:
  - Intercept the traffic that contains the ID value
  - Guess a valid ID value (weak randomness)
  - Discover other logic flaws in the sessions handling process

Good Session ID Properties



http://xkcd.com/221

- **Hard to guess**
  - **Large entropy (big random number)**
  - **No patterns in IDs issued**
- **No reuse**

Session Hijacking Mitigation

- Create new session id after
  - Authentication
  - switching encryption on
  - other attributes indicate a host change (IP address change)
- Encrypt to prevent obtaining session ID through eavesdropping
- Expire IDs after short inactivity to limit exposure of guessing or reuse of illicitly obtained IDs
- Entropy should be large to prevent guessing
- Invalidate session IDs on logout and provide logout functionality

Session Hijacking Example

1. An insecure web application accepts and reuses a session ID supplied to a login page.
2. Attacker tricked user visits the web site using attacker chosen session ID
3. User logs in to the application
4. Application creates a session using attacker supplied session ID to identify the user
5. The attacker uses session ID to impersonate the user

Successful Hijacking Attack

JAVA

1. Tricks user to visit

```
http://bank.com/login;JSESSIONID=123
```

2. User Logs In

```
http://bank.com/login;JSESSIONID=123
```

3. Creates the session

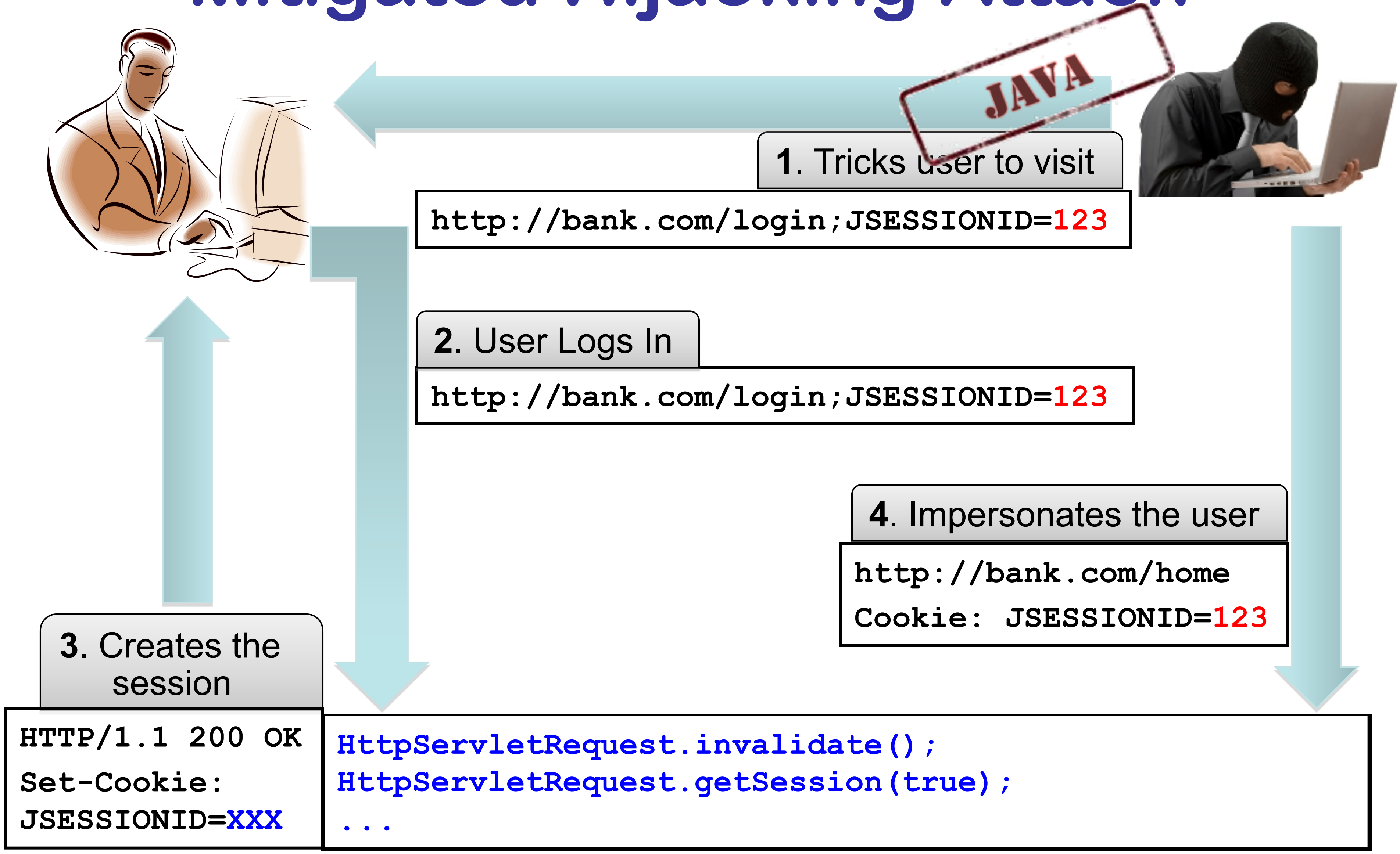
```
HTTP/1.1 200 OK
Set-Cookie:
JSESSIONID=123
```

```
if(HttpServletRequest.getRequestedSessionId() == null) {
     HttpServletRequest.getSession(true);
}
...
```

4. Impersonates the user

```
http://bank.com/home
Cookie: JSESSIONID=123
```

Mitigated Hijacking Attack



Open Redirect

(AKA: URL Redirection to Untrusted Site, and Unsafe URL Redirection)

- Description
  - Web app redirects user to malicious site chosen by attacker
    - URL parameter (reflected)

      http://bank.com/redir?url=http://evil.com
    - Previously stored in a database (persistent)
  - User may think they are still at safe site
  - Web app uses user supplied data in redirect URL
- Mitigations
  - Use white list of tokens that map to acceptable redirect URLs
  - Present URL and require explicit click to navigate to user supplied URLs

Open Redirect Example

1. User receives phishing e-mail with URL

   `http://www.bank.com/redir?url=http://evil.com`
2. User inspects URL, finds hostname valid for their bank
3. User clicks on URL
4. Bank's web server returns a HTTP redirect response to malicious site
5. User's web browser loads the malicious site that looks identical to the legitimate one
6. Attacker harvests user's credentials or other information

Successful Open Redirect Attack

JAVA

1. User receives phishing e-mail

```
Dear bank.com costumer,
Because of unusual number of invalid login attempts...
<a href="http://bank.com/redir?url=http://evil.com">
Sign in to verify</a>
```

2. Opens

```
http://bank.com/redir?url=http://evil.com
```

```
String url = request.getParameter("url");
if (url != null)  {
    response.sendRedirect( url );
}
```

3. Web server redirects

```
Location: http://evil.com
```

4. Browser requests http://evil.com

```
<h1>Welcome to bank.com<h1>
Please enter your PIN ID:
<from action="login">
•••
```

5. Browser displays forgery

Open Redirect Mitigation

JAVA

1. User receives phishing e-mail

```
Dear bank.com costumer,
•••
```

2. Opens

```
http://bank.com/redir?url=http://evil.com
```

```
boolean isValidRedirect(String url) {
   List<String> validUrls = new ArrayList<String>();
   validUrls.add("index");
   validUrls.add("login");
   return (url != null && validUrls.contains(url));
}
•••
if (!isValidRedirect(url)){
   response.sendError(response.SC_NOT_FOUND, "Invalid URL");
   •••
```

3. bank.com server code correctly handles request

```
404 Invalid URL
```

THE UNIVERSITY of WISCONSIN MADISON
Universitat Autònoma de Barcelona

NATO OTAN
NSF

Generally Bad Things

General Software Engineering

- Don't trust *user data*
  - You don't know where that data has been
- Don't trust your own *client* software either
  - It may have been modified, so always revalidate data at the server.
- Don't trust your operational configuration either
  - If your program can test for unsafe conditions, do so and quit
- Don't trust your own code either
  - Program *defensively* with checks in high and low level functions
- KISS - Keep it simple, stupid
  - Complexity kills security, its hard enough assessing simple code

Denial of Service

- Description
  - Programs becoming unresponsive due to over consumption of a limited resource or unexpected termination.
- General causes
  - Not releasing resources
  - Crash causing bugs
  - Infinite loops or data causing algorithmic complexity to consume excessive resources
  - Failure to limit data sizes
  - Failure to limit wait times
  - Leaks of scarce resources (memory, file descriptors)

Information Leaks

- Description
  - Inadvertent divulgence of sensitive information
- General causes
  - Reusing buffers without completely erasing
  - Providing extraneous information that an adversary may not be able to otherwise obtain
    - Generally occurs in error messages
    - Give as few details as possible
    - Log full details to a database and return id to user, so admin can look up details if needed

Information Leaks

- General causes (cont.)
  - Timing attacks where the duration of the operation depends on secret information
  - Lack of encryption when using observable channels
  - Allowing secrets on devices where they can't be erased such as swap space (mlock prevents this) or backups

General Software Engineering

- Don’t trust user data
  - You don’t know where that data has been
- Don’t trust your own client software either
  - It may have been modified, so always revalidate data at the server.
- Don’t trust your own code either
  - Program defensively with checks in high and low level functions
- KISS - Keep it simple, stupid
  - Complexity kills security, its hard enough assessing simple code

Let the Compiler Help

- Turn on compiler warnings and fix problems
- Easy to do on new code
- Time consuming, but useful on old code
- Use lint, multiple compilers
- **-Wall** is not enough!

  gcc: **-Wall, -W, -O2, -Werror, -Wshadow, -Wpointer-arith, -Wconversion, -Wcast-qual, -Wwrite-strings, -Wunreachable-code** and many more

  – Many useful warning including security related warnings such as format strings and integers

Let the Perl Compiler Help

- `-w` - produce warning about suspect code and runtime events
- `use strict` - fail if compile time
- `use Fatal` - cause built-in function to raise an exception on error instead of returning an error code
- `use diagnostics` - better diagnostic messages

Perl Taint Mode

- Taint mode (-T) prevents data from untrusted sources from being used in dangerous ways
- Untrusted sources
  - Data read from a file descriptor
  - Command line arguments
  - Environment
  - User controlled fields in password file
  - Directory entries
  - Link referents
  - Shared memory
  - Network messages
- Environment sanitizing required for `exec`
  - `IFS PATH CDPATH ENV BASH_ENV`

Books

- Viega, J. & McGraw, G. (2002). *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley.
- Seacord, R. C. (2005). *Secure Coding in C and C++.* Addison-Wesley.
- Seacord, R. C. (2009). *The CERT C Secure Coding Standard*, Addison-Wesley.
- McGraw, G. (2006). *Software security: Building Security In*. Addison-Wesley.
- Dowd, M., McDonald, J., & Schuh, J. (2006). *The Art of Software Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley.

Would you like this tutorial (and related ones) 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

THE UNIVERSITY of WISCONSIN MADISON
Universitat Autònoma de Barcelona

Secure Coding Practices for Middleware

Elisa Heymann

Elisa.Heymann@uab.es

Barton P. Miller
James A. Kupsch

bart@cs.wisc.edu

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

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

Questions?

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