CS 580 Client-Server Programming
Spring Semester, 2009
Doc 20 Security
April 30, 2009

References

2009 CWE/SANS Top 25 Most Dangerous Programming Errors, March 10, 2009, http://cwe.mitre.org/top25/

SQL Injection - http://en.wikipedia.org/wiki/SQL_injection

Buffer Overflow - http://en.wikipedia.org/wiki/Buffer_overflow

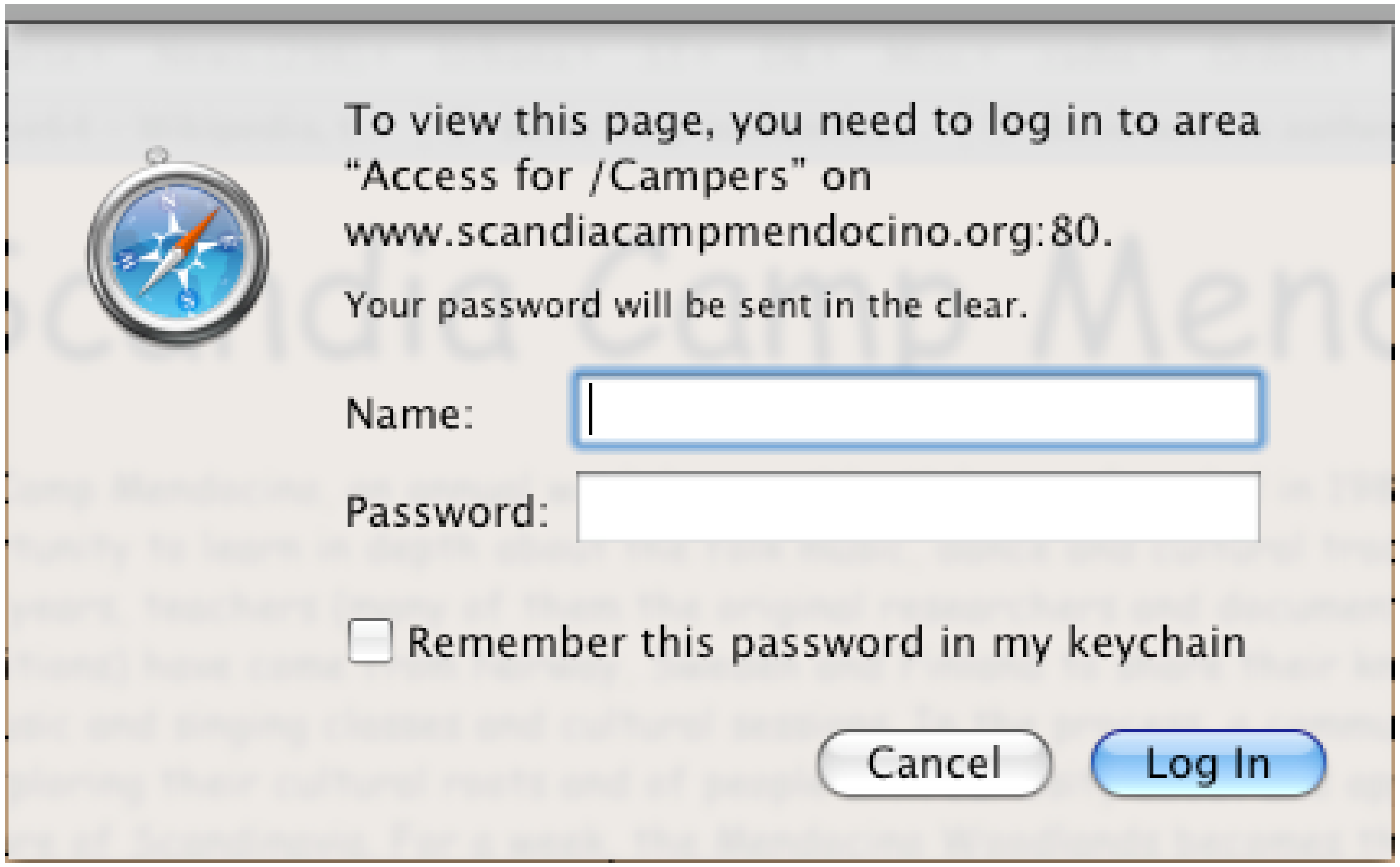
NIH Security Web Site http://www.alw.nih.gov/Security/security.html

Applied Cryptography Second Edition, Bruce Schneier, John Wiley & Sons, 1996

Secrets and Lies: Digital Security in a Networked World, Bruce Schneier, John Wiley & Sons, 2000

Before Security some Information
for the assignment

Basic Http Authentication



Sample Interaction

Client Request

```
GET /private/index.html HTTP/1.0
Host: localhost
```

Server Response

```
HTTP/1.0 401 Authorization Required
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:18:15 GMT
WWW-Authenticate: Basic realm="Secure Area"
Content-Type: text/html
Content-Length: 311

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
  <HEAD>
    <TITLE>Error</TITLE>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
  </HEAD>
  <BODY><H1>401 Unauthorised.</H1></BODY>
</HTML>
```


Example from http://en.wikipedia.org/wiki/Basic_access_authentication

Sample Interaction

Client request (user name "Aladdin", password "open sesame")

```
GET /private/index.html HTTP/1.0
Host: localhost
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Server response:

```
HTTP/1.0 200 OK
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:19:07 GMT
Content-Type: text/html
Content-Length: 10476
```

Base64 Encoding

Encodes any byte sequence into sequence of printable characters

Encoded sequence can be decoded

Used to encode MIME contents for transport
- Email Attachments

Base 64 Algorithm

Divide input into parts each part 24 bits long (3 bytes)

Convert each 24 bit sequence as follows:

- Divide the 24 bits into four groups of 6 bits
- Use the table to convert each 6 bits

Value	Encoding
0	A
1	B
...	...
25	Z

Value	Encoding
26	a
27	b
...	...
51	z

Value	Encoding
52	0
53	1
...	...
61	9

Value	Encoding
62	+
63	/

pad with =

Example

		cats					text
001111111	00111101	01001010	01001001				binary
001111	111001	111010	100101	001001	001		6 bit groups
001111	111001	111010	100101	001001	001000		6 bit groups padded
15	57	58	37	9	8		As decimal
P	5	6	l	J	I	= =	Converted

Base64 Encoding & HTTP Authentication

Use Base64 encoding for user name and password

user name "Aladdin"
password "open sesame"

Aladdin:open sesame

↓

QWxhZGRpbjpvcGVuIHNlc2FtZQ==

↓

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Security

Top 25 Programming Security Errors

Selected based on:

How common the error is

Consequences of error

2009 CWE/SANS Top 25 Most Dangerous Programming Errors
March 10, 2009
http://cwe.mitre.org/top25/

Top 25: Insecure Interaction Between Components

Improper Input Validation
Improper Encoding or Escaping of Output
Failure to Preserve SQL Query Structure (aka 'SQL Injection')
Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
Failure to Preserve OS Command Structure (aka 'OS Command Injection')
Cleartext Transmission of Sensitive Information
Cross-Site Request Forgery (CSRF)
Race Condition
Error Message Information Leak

13

SQL Injection

"SELECT * FROM users WHERE name = '" + userName + "';"

let username be
a’ or ‘t’ = ‘t

SELECT * FROM users WHERE name = 'a' or 't'='t';

which is the same as

SELECT * FROM users;

Example from http://en.wikipedia.org/wiki/SQL_injection

SQL Injection

let username be

a’; DROP TABLE users; Select * FROM data where name = ‘a

"SELECT * FROM users WHERE name = '" + userName + "';"

becomes:

SELECT * FROM users WHERE name = 'a' ’;
DROP TABLE users;
Select * FROM data where name = ‘a’;

Preventing SQL Injection In Java

Replace

```
Connection con = (acquire Connection);
Statement stmt = con.createStatement();
ResultSet rset =
    stmt.executeQuery("SELECT * FROM users WHERE name = '" +
    userName + "';");
```

with

```
Connection con = (acquire Connection)
PreparedStatement pstmt =
    con.prepareStatement("SELECT * FROM users WHERE name = ?");
pstmt.setString(1, userName);
ResultSet rset = pstmt.executeQuery();
```


Wikipedia claims this is safe. I have not verified the claim.

OS Command Injection

Java Example
Running any Program

```
String script = System.getProperty("SCRIPTNAME");
if (script != null) System.exec(script);
```


17

Example from http://cwe.mitre.org/data/definitions/78.html

OS Command Injection

Perl

Running a specific command

Code

```
use CGI qw(:standard);
$name = param('name');
$nslookup = "/path/to/nslookup";
print header;
if (open($fh, "$nslookup $name|")) {
    while (<$fh>) {
        print escapeHTML($_);
    print "<br>\n";
    }
    close($fh);
}
```

Input

cwe.mitre.org%20%3B%20/bin/ls%20-l

Decodes to

/path/to/nslookup cwe.mitre.org ; /bin/ls -l

So return list of current directory

Example From http://cwe.mitre.org/data/definitions/78.html

Top 25: Risky Resource Management

Failure to Constrain Operations within the Bounds of a Memory Buffer
External Control of Critical State Data
External Control of File Name or Path
Untrusted Search Path
Failure to Control Generation of Code (aka 'Code Injection')
Download of Code Without Integrity Check
Improper Resource Shutdown or Release
Improper Initialization
Incorrect Calculation

19

Buffer Overflow

A = 3 byte string

B = integer

A			B	
0	0	0	0	3

Write 'cate' in A

A			B	
c	a	t	e	3

What to use Buffer Overflow for

Overwrite local variable to change program's behavior

Overwrite the return address in a stack frame

Overwrite a function pointer or exception handler

Buffer Overflow Solution 1

Check the Buffer Size

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
  char buffer[10];
  if (argc < 2)
  {
    fprintf(stderr, "USAGE: %s string\n", argv[0]);
    return 1;
  }
  strncpy(buffer, argv[1], sizeof(buffer));
  buffer[sizeof(buffer) - 1] = '\0';  /* explicitly write a string terminator */
  return 0;
}
```

Buffer Overflow Solution 2

Use a language that checks for array out-of-bounds errors

- Java
- Smalltalk
- Ruby
- Python
- etc.

Top 25: Porous Defenses

Improper Access Control (Authorization)
Use of a Broken or Risky Cryptographic Algorithm
Hard-Coded Password
Insecure Permission Assignment for Critical Resource
Use of Insufficiently Random Values
Execution with Unnecessary Privileges
Client-Side Enforcement of Server-Side Security

24

Hard-Coded Password

```
DriverManager.getConnection(url, "admin", "secret");
```

Decompile byte code to see strings

```
javap -c ConnMngr.class
22: ldc #36; //String jdbc:mysql://ixne.com/rxsql
24: ldc #38; //String admin
26: ldc #17; //String secret
```


Example from http://cwe.mitre.org/data/definitions/259.html

Hard-Coded Password - Some Solutions

Store passwords outside of the code in a strongly-protected, encrypted configuration file or database

For inbound authentication apply strong one-way hashes to your passwords

Client-Side Enforcement

Server-side

```
$sock = acceptSocket(1234);
($cmd, $args) = ParseClientRequest($sock);
if ($cmd eq "AUTH") {
    ($username, $pass) = split(/\s+/, $args, 2);
    $result = AuthenticateUser($username, $pass);
    writeSocket($sock, "$result\n");
    # does not close the socket on failure; assumes the
    # user will try again
}
elsif ($cmd eq "CHANGE-ADDRESS") {
    if (validateAddress($args)) {
        $res = UpdateDatabaseRecord($username, "address", $args);
        writeSocket($sock, "SUCCESS\n");
    }
    else {
    writeSocket($sock, "FAILURE -- address is malformed\n");
    }
}
```


Example from http://cwe.mitre.org/data/definitions/602.html

Client Side

```
$server = "server.example.com";
$username = AskForUserName();
$password = AskForPassword();
$address = AskForAddress();
$sock = OpenSocket($server, 1234);
writeSocket($sock, "AUTH $username $password\n");
$resp = readSocket($sock);
if ($resp eq "success") {
    # username/pass is valid, go ahead and update the info!
    writeSocket($sock, "CHANGE-ADDRESS $username $address\n";
}
else {
    print "ERROR: Invalid Authentication!\n";
}
```


28

Here the client checks, but the server does not. As long as people only use the above client we are safe. While honest people will restrict them selves to the above client, hackers trying to break into your system will not.

Client-Side Enforcement - Solutions

Server repeats any security check done by client

Client side security checks help

- Detect an attack
- Provide helpful feedback to the user about the expectations for valid input

29

Some Problems Require Global Solution

Denial of Service Attacks

Cryptography

Security ≠ Cryptography

Kevin Mitnick often got people's passwords by asking

Cryptographic Examples

Alice and Bob communicate with each other in secret

Eve wants to see Alice's and Bob's communication

One-Way Hash Functions

Let M be a message (sequence of bytes)

A one-way hash function f() such that:

f maps arrays of bytes to arrays of bytes
f(M) is always the same length
Given an M it is easy to compute f(M)
Given f(M) it is hard to compute M
Given M it is hard to find N such that f(M) = f(N)

MD5 - Message Digest 5
SHA - Secure Hash Algorithm

MD5 & SHA in Java

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class OneWay
	{
	public static void main(String args[])
		throws NoSuchAlgorithmException
		{
		MessageDigest sha = MessageDigest.getInstance("SHA");
		sha.update("Hi mom".getBytes());
		byte[] shaHash = sha.digest();
		System.out.println(new String(shaHash));

		MessageDigest md5 = MessageDigest.getInstance("MD5");
		md5.update("Hi mom".getBytes());
		byte[] md5Hash = md5.digest();
		System.out.println(new String(md5Hash));
		}
	}
```


35

Authentication with One-Way Functions

Given

list of passwords P_1, P_1, ... , P_k,

One-way function f()

In password file only store $f(P_k)$

Password File

Name1	
Name2	
Name3	

36

Authentication with One-Way Functions

Host stores only one-way functions of passwords

Alice sends host her password

Host performs one-way function on password

Host compares result with stored value for Alice

Access to password file does not help Eve in getting Alice's password

Dictionary Attacks

Eve compiles list of N common passwords

Eve applies one-way function to all N common passwords f(P)

Eve stores all results as map f(P) -> P

For the N common passwords Eve has the inverse function

If

- N = 1,000,000
- f() = MD5
- Passwords are about 8 bytes

Eve's map is ~ 24MB

Salt

Salt - random string

Given

list of passwords P1, P1, ... , Pk,

One-way function f()

Salt S

In password file only store f(Pk + S)

Password File

Name1	$f(P_1 + S)$
Name2	$f(P_2 + S)$
Name3	$f(P_3 + S)$

Dictionary Attacks and Salt

Eve compiles list of N common passwords

Eve has to compile f(P + S) for each password & possible salt

Greatly increases size of Eve's reverse map file

Daniel Klein password-guessing often cracks 40% of salted passwords

SKEY Authentication

Given a one-way function f()

Alice enters a random number R

Computer computes

$x_1 = f(R)$

$x_2 = f(f(R)) = f(x_1)$

$x_3 = f(f(f(R))) = f(x_2)$

...

$x_{100} = f(x_{99})$

Alice remembers $x_1, ..., x_{100}$

Computer stores x_{101}

To log on to computer Alice enters x_{100}

Computer computes $f(x_{100})$ and compares to x_{101}

Computer stores x_{100}

Encryption

Traffic on a network can be sniffed

A solution is encryption of all traffic

This can be done at any layer of the protocol stack

Two basic types of encryption

Symmetric encryption

One key both encrypts and decrypts

Public/Private key encryption

One key encrypts, another decrypts

Symmetric encryption

Work fine except that both parties have to share the same key

Distributing the shared key is as hard as distributing secrets

Public/Private Key Encryption

Algorithm uses two keys

Can use any key to encrypt message

The other key will decrypt the message

The encrypt key cannot be used to decrypt message

One key is made public

Other key is kept private

Public/Private Key Encryption

Require a way to distribute public keys in open

Algorithms tend to be slow

Often used to distribute shared key for Symmetric Encryption algorithm

Common Algorithms

RSA (Rivist, Shamir, Adleman)
DSA (Digital Signature Algorithm)

45

Public/Private key Uses - Digital Signature

Alice has a message she wants to sign

Alice uses her private key to encrypt the message

Anyone can decrypt the message using Alice's public key

If Alice's public key decrypts the message the Alice had to encrypt it

So

- We know the message came from Alice
- The message was not altered
- Alice cannot deny sending the message

Public/Private key Uses - Secret Messages

Alice wishes to send a message to Bob

Alice encrypts the message using Bob's public key

Bob can use his private key to decrypt the message

Eve cannot decrypt the message

RSA

Public Key

Key contains n & e

n = p*q, p & q are primes
e relatively prime to (p-1)*(q-1)

p & q must be kept secret

Private Key

Key contains d & N

$d = e^{-1} \bmod ((p-1)*(q-1))$

that is (d*e) mod ((p-1)*(q-1)) = 1

Encrypting

Let m be a message such that m < n

Let the encrypted message, c be

$c = m^e \bmod n$

Decrypting

$m = c^d \bmod n$

RSA Example

Alice's Keys

Let

p = 47

q = 71

e = 79

Then

n = p*q = 3337

d = 79-1 mod 3220 = 1019

Public key

n = 3337

e = 79

Private key

d = 1019

n = 3337

Sending Message to Alice with RSA

Let the message, m, be 42

Compute the encrypted message

$c = m^e \bmod n$
$= 42^{79} \bmod 3337$
$= 2973$

Decrypting the message

c^d mode n $= 2973^{1019} \bmod 3337$
$= 42$

Public key

$n = 3337$
$e = 79$

Private key

$d = 1019$
$n = 3337$

Why Public/Private Key Systems are slow

2973^{1019} contains 3,540 digits

153324825763087744574373595168451133863015624053349220949383251123354689728853499086796706122461358155001511
627250620626278306977317133156831821375697720222675345237911639440809081188807238912082607767653374694254259
840290502980744098350554315241220637225339547830545096818351337831772204856463314806097152376452676011365414
670176853092829642594505518711049506767196250589624703362307799777581184185156576253928185821270301085489029
251471011188478049071649904951523526941938381678325882075593830187882719425345330541389695451512079977887573
228414291230778194055970170567657509041433759287177541035514341040006129060560767435337541469354368153650425
684863198509972411784588705614218505323935902674612366722498745688220565914013736449006641379652971941221308
300963709942487170353386734189752024346226474610542630666828230545751447635450043801539390831470668403669358
248737428779700737184874716099800172396439452421884696865969471630902845667717616954581331695377628416904063
034129999094435973544456705966852304151936484227959526696982991364168016239049451846941170369989612637339773
706351181034084084883319697941249201034542394154021126566769429760691722761151437765436351125271053153337717
578014584166921049674223746163767891225837938472781637610190680223257687812224759022708196098264798098641054
366373924129011007246003145954926047454413717831751151093566736388227161099058034209365017518905437596664114
799562533924113487752285621564056858117291212037971143868871503847360181171785601631839300245304366927348463
622205769670858717842607743807223402009145330116660895711591540809664506193139195855840187051580386553397969
824035446679401322205570027829903001318599701691036811994708397339741451750677888386436611407844302459107982
609800297619216734487910126055132447277185724332376329368408848506560054857321355637710875799892563687802049
518937425261285913609510013710229415003313852869054209416183502921816053904830678158296784936699144967867158
909020920652986705777604814640903038406370662718316197491038922711582626766472121863766027542311475349495949
186541923687432423977115172799235817008570091266657063604813903405159696630144184721149875502545048077648011
449806299682073291834820998551186348569245704972835670313999020339824090833212789125698880434603485591805229
2112711112566152618324171004278481098729759256776250224702179557589916809381159168703127571137406535404779137
987377800683025109406036661153109776925889349681189712143572224918307393672835992186516188578086860569924574
356410388215679793198388909205725281115973873108388941232400387913337997436784493912376304674676289823112565
103385433817957994879048198740363016168048448638592997671825999947964712683365905653869636249709113565716144
914604807510212385313203242836593575651569497348359662234437053838084260087672669945285744433051990796937678
960342208502409237375098709740947461642990850601592825087682203252981071368327983066823161963009933150774501
645709611962625423343093086289829952749675819394411618696373683215918598796998585825724518770655726709462697
283095983419726196714991491331289049840615529974067762763712023312757670684919924328365114251478888802948610
406120173843572604101848939022910773860327556842455857223175466220133756395933773034693618009831559013180505
039851757312231727983774742502198990010328219242915699183051144364853476070538343728986106438751370131132135
093670487141374029283980218652436765690738168613429069152165412398482547889573676073185555875323343395598824
04632059796981175690884640765411172923700487474033839341512086787168944649246518437

Alice Signing a Document with RSA

Let the message, m, be 42

Encrypt the message
using the Private Key

$c = m^e \bmod n$
$= 42^{1019} \bmod 3337$
$= 2151$

Decrypting the message using the
Public key

c^e mode n $= 2151^{79} \bmod 3337$
$= 42$

Public key

n = 3337
e = 79

Private key

d = 1019
n = 3337