Team LulzSec

Secure Development Lifecycle

ICS 491

Spencer Bishop

Quynh-Tram Nguyen

Andy Omori

August 9, 2015

CONTENTS

EXECUTIVE SUMMARY …………………………………………..………………………3

REQUIREMENTS

1. Security Requirements ………………………………………….………………4
2. Quality Gates/Bug Bars ………………………………………………………….5
3. Security and Privacy Risk Assessments ……………………………………….7

DESIGN

1. Design Requirements …………………………………….…………………...8
2. Attack Surface Analysis/Reduction …………….………………….………..10
3. Threat Modeling ……………………..……………………….………………..11

IMPLEMENTATION

1. Approved Tools …………………………...…………………….………………12
2. Deprecate/Unsafe Functions …………………………...……………….….……12
3. Static Analysis ……………………………...……………………....…………..13

VERIFICATION

1. Dynamic Analysis Tool: JUnit Review ...……………..…..………………….….…….15
2. Fuzzy Testing ……………………..……………………...……………………...16
3. Attack Surface Review ……….………………...………………….....…………...22

RELEASE

1. Incident Response Plan ……….………………...…………………....……………..24
2. Final Security Review ……….………………...…………………...………………..29
3. Certify Release and Archive ……….……………...…………………....……...…29

EXECUTIVE SUMMARY

Team LulzSec have decided that we would use Java for our program, while using a Java applet program. Team LulzSec will have our program store PlayStation 4 video games and present them in a scroll box that you can look through and have a login in order to protect the databases. The program codes will be hosted on GitHub.

REQUIREMENTS

1) Security Requirements

To ensure that our program is secure, our program needs to meet the following security and privacy requirements:

1. Secure storage of username and password so that user's data will be only viewable to that specific viewer.
2. The user must create strong login information – username and password.
3. The user must enter the correct the username and password in order to access the program.
4. Only the admin account has the security clearance to make changes to the database.

Our team will be using GitHub to host our program codes. In order to keep track of security flaws throughout the development, our team will be using GitHub Issue Tracker. This feature will make it be easy to keep track of all the security flaws and everyone in the group will be constantly updated with the current security flaws and patches.

2) Quality Gates/Bug Bars

PRIVACY

CRITICAL

-Data protection: must make sure all personally identifiable information is encrypted and stored in an encrypted database that only management can access.

-Cookie encryption: must make sure that all data saved as a cookie is encrypted.

-User controls: must allow the user to access and change their own information.

-User knowledge: user must know when we are collecting personal information and can opt out of it if possible.

IMPORTANT

-Non-sensitive information: must inform the user when using non-sensitive information from user computer and explicit opt in consent.

-Random data collection: must allow user to stop collection of random data collected by application.

-Minimizing data: must minimize amount of sensitive necessary data transferred through 3rd party to achieve application use.

-Data protection: user information needs a login mechanism to access information.

-Data management: must have a retention plan for data.

MODERATE

-Data protection: must secure all temporary data.

-Minimizing data: must minimize amount of non-sensitive necessary data transferred through 3rd party to achieve application use.

-Cookie usage: must use a current cookie or create a new cookie. a cookie must not be held longer than necessary and one must not be made if unnecessary.

<u>SECURITY</u>

<u>CRITICAL</u>

-Privilege elevation(remote): Unauthorized system access, running of anything without extensive user actions.

<u>IMPORTANT</u>

-Privilege elevation(local): low privilege level users can elevate the status of their account to administrator or another user.

-Spoofing: ability for an attacker to make a similar application to steal users information.

-Information disclosure: must make sure an attacker cannot read any local system items regarding information stored on the server.

-Denial of service: System corruption requiring re-installation of system or components.

<u>MODERATE</u>

-Denial of service: System corruption requiring a reboot or showing a “blue screen” or bug.

3) Security and Privacy Risk Assessments

To assess which parts of our program need threat modeling security reviews, our group has created the following requirements:

1. Who would have the security level clearance to access this information?
2. Does this part of the program contain sensitive information such as username or password?

Our goals are any part of the program that contains sensitive information would need to threat modeling and security reviews and only the administrator account has the security clearance to make changes to the database.

DESIGN

1) Design Requirements

We agree that this database will store different types of games made exclusively for the PlayStation 4 system that came out not too long ago. The program will stores sensitive information like a login information, in order to have what the user purchases or has whatever he wants loaded in to a file where it will act like a shopping cart for the user.

First we must fulfill question in order to understand what we want. We decided that this application would be fitting for the gamer audience. Since we assume that they know a lot about technology and how it works, we understand that they may know a lot about how sensitive data can be over a network.

Also we can ask ourselves where the application will run. We have decided that it can be run on separate files for now like a catalog that you can download so it won't always be directed towards the Internet.

To protect the user information being stored and preventing data from the catalog from being manipulated or tampered. We will ask the user for login information -- username and password. Since we are new to encryption techniques, we will have to design an encryption technique and that will encrypt when the user will first create the username and password. The user must create a strong password. The password length can be anywhere from the minimum (8 characters) to the maximum allowed password length (13 characters); it cannot contain the user name, and it must include uppercase, lowercase, and special characters. The username must not contain the user's full name and be at least 6 characters long. The passwords will be stored on an output file to ensure it can't be hijacked through the program. Also we won't contain highly critical data like Social Security Numbers or addresses because it is not a type of program thats needs to confirm identities.

If the application was compromised, then we would fear that attacker have access to user information and passwords. They would also be able to manipulate data in the application and possibly inject malware, adware, etc. In order to reduce traffic on who gets to use this application, we will restrict privilege on who gets to update the application to only admins in order to protect data.

2) Attack Surface Analysis/Reduction

Open sockets
We should check for open sockets using third party tools lessening vulnerabilities

Enabled Accounts
Accounts can probably be manipulated so there may be a vulnerability which can be preventable by giving privilege to admin

Default, blank, and weak username/password
We can counter this by asking for better password input through different combinations.

Extensive user and group privileges
It would be best to restrict privilege to admins.

Buffer overflows
Probably the most common and preventable, we can assign the right memory for each variable.

Privilege escalation
We will only give privilege to admins

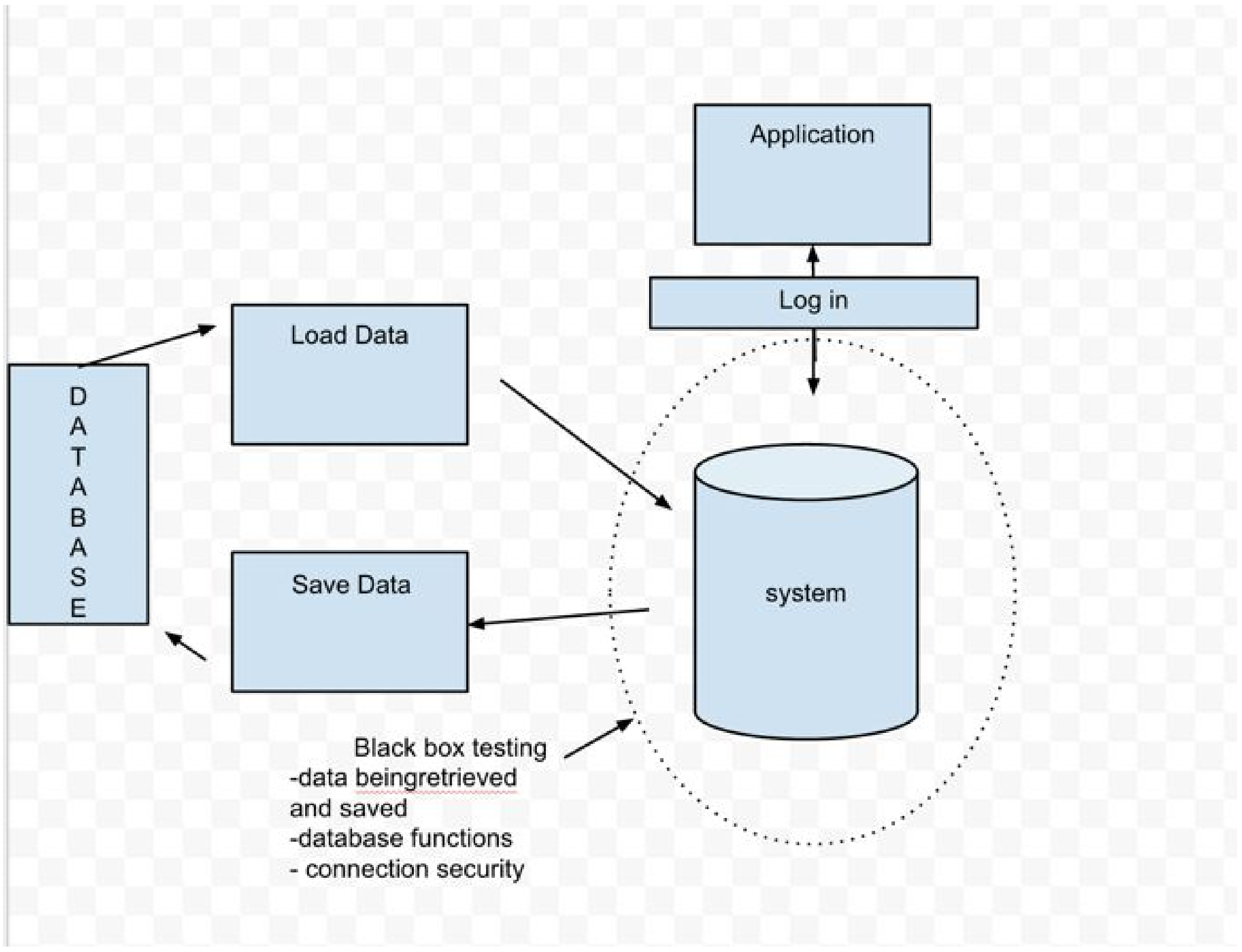
Denial-of-service attack
We predict this is really common and can’t be 100 percent prevented therefore this is a vulnerability.

Unpatched databases
The database should always be patched and up to date leading to less of a chance of vulnerabilities.

Unencrypted sensitive data at rest and in motion
We should always have encryption in order to prevent plain text passwords from being breached.

SQL Injections
These can be widely used in database style programs so we should constrain input and use parameters with stored procedures to prevent a sql vulnerability.

3) Threat Modeling



IMPLEMENTATION

1) Approved Tools

The following is a list of approved tool that LulzSec is using for our program:

a. Eclipse Luna 4.4.2
b. Microsoft Excel 2010
c. OS: Windows 7, 8.1
d. Java SE 8
e. MySQL

2) Undeprecate Unsafe Functions

Deprecate/Unsafe functions are functions that programmers have discouraged from using, typically because they are 1) dangerous, buggy, highly inefficient, 2) they are obsolete, 3) there are better alternative exist. The following is a working list of deprecate/unsafe Java methods and classes that our team may use in our code:

Deprecate/Unsafe Java Class/Method	**Alternative Safer Java Class/Method**
java.io.StringBufferInputStream	java.io.StringReader
action	ActionListener
preferredSize	setPreferredSize
show()	setVisible(true)
appendText	append
createScrollPaneForTable	JScrollPane

3) Static Analysis

Static Analysis Tool Used: Checkstyle for Eclipse Plug-in

We decided that we wanted to use Checkstyle because it is compatible with Eclipse and it is one of the approved tools. Also instead of installing another IDE such as IntelliJIDE, we decided it would save time by just installing Checkstyle since all of our members are using Eclipse and it is a plug-in for Eclipse IDE.

Initially, installing the plugin was just basic since it was available in the Eclipse marketplace, so installation was handy. When running the program, Checkstyle provided real-time feedbacks and was extremely quick to find coding problems. Checkstyle seems to highlight even the smallest mistakes. This is great because Checkstyle will ensures your code is organized, which decreases the amount of small coding mistakes.

Pros

- Quick and easy installation for Eclipse.
- Works quickly using yellow as an identification on what problems you have on your code.
- Can find even the smallest errors like unnecessary indentation, separating variables.
- Clearly does its job locating any sort of vulnerabilities, can be very effective.

Cons

- A lot of the suggestions were just indentation, and separating a lot of the variables and instances (minor issues which were easily fixable).

In the end we didn't really find much major problems with our code. After looking through 100 problems, 75 percent of them was indentation issues, which is easily fixable. The rest was just separating statements into individual variables. This is also easily fixable.

VERIFICATION

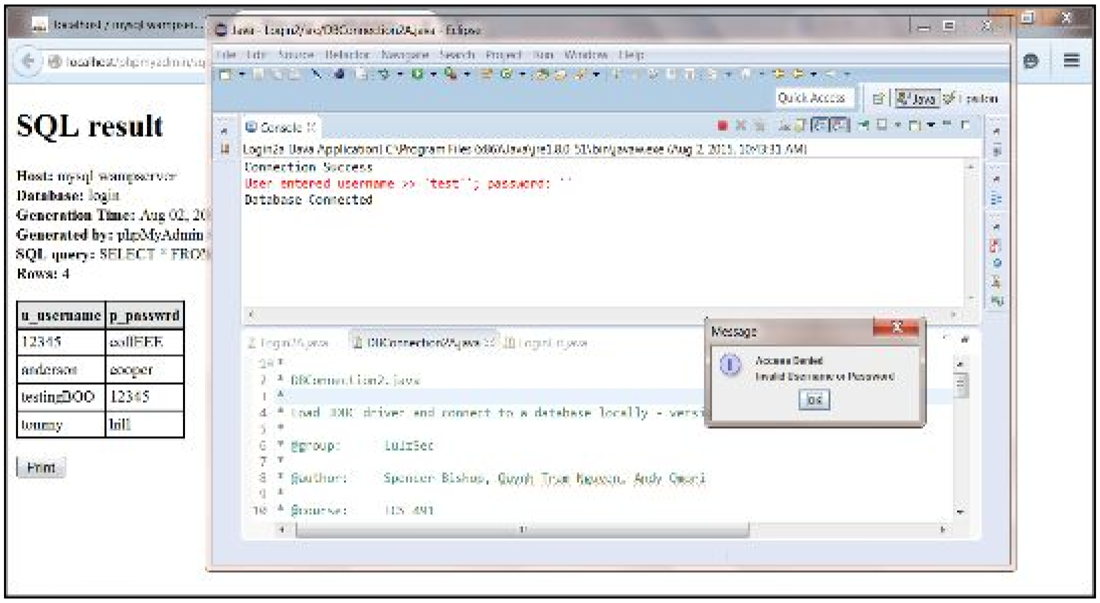
1) Dynamic Analysis Tools Review

We used JUnit to do our dynamic analysis. We had no real issues when writing and running our program. Our input and output really had no issues and we didn't use JUnit testing to test our database connections. Our biggest problem was connecting to a database for the login information. Other than that our code worked fine and everything was in working order. While JUnit is easy to use and to install, JUnit is very tedious to use and we spent a fair amount of time using it for testing. We might use a different tool for testing from here on out, but we have yet to decide. JUnit testing works well, but the time it takes to set it up and run it can be a bit much. Well, since much of our program just displays information, we don't really need to have a lot of test cases. Since are code is not heavy in LOC (Lines of Code), we feel as if we can manually test everything without the need for test cases from JUnit. Also most of our program will be grabbing information out of a separate database such as excel or a SQL server which could probably be out of scope for some JUnit test cases.

2) Fuzzy Testing

a. <u>Failed SQL injections</u>

For the following two testings (Fig 2.1 and 2.2) on DBConnection2A.java and Login2A.java, our members performed two well known SQL injections --- <u>test’</u> and <u>‘ OR 1=1</u> attack. The results show we are not able to gain access because the program does not recognize these input as valid input. In the first test, the single quote is a string delimiter in SQL. If user input is not sanitized, hacker could easily use the single quote character to escape the developer’s SQL query in a program and tamper with the program data/database. In the second test, <u>‘ OR 1=1</u> would have force the selection of a valid username/password since 1 is always 1, which would allow a hack to gain access our database and tamper the data. We think by using a prepared statement to extra the login information in our database and avoiding using deprecated functions (i.e. getText() for password text field) may most likely have prevented these attacks. Sanitizing user input would prevent similar SQL injection.



SQL result

Host: mysql wampserver
Database: login
Generation Time: Aug 02, 20
Generated by: phpMyAdmin
SQL query: SELECT * FRO
Rows: 4

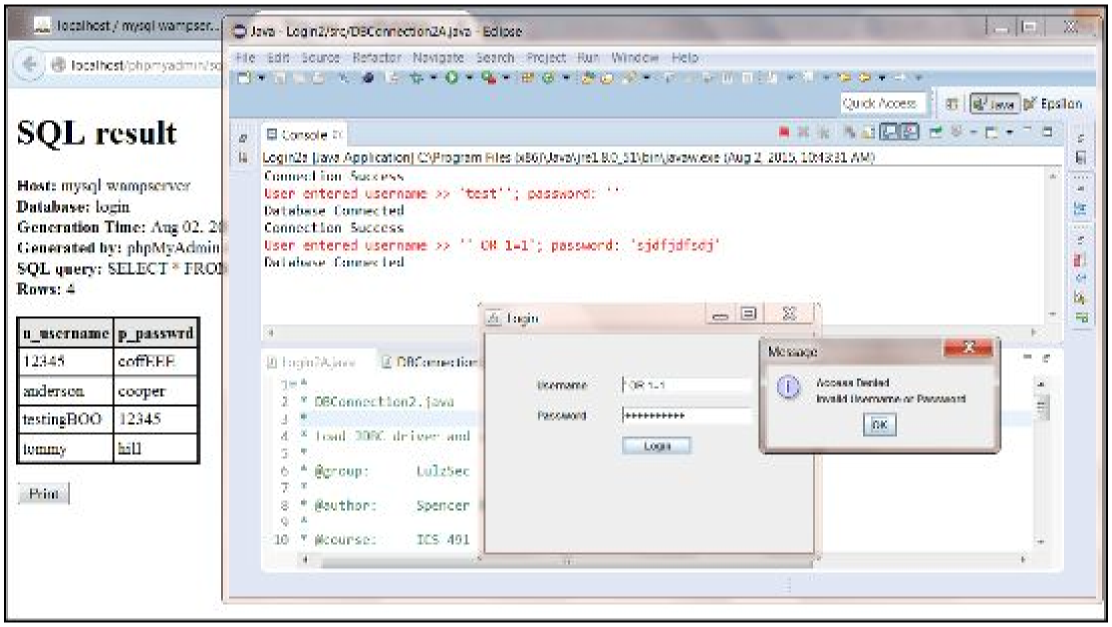
u_username	p_passwrd
12345	coffEEE
anderson	cooper
testingBOO	12345
tommy	hill

```
Login2a [Java Application] C:\Program Files (x86)\Java\jre1.8.0_51\bin\javaw.exe (Aug 2, 2015, 10:43:31 AM)
Connection Success
User entered username >> 'test''; password: ''
Database Connected
```

Message: Access Denied. Invalid Username or Password

```
 * DBConnection2.java
 *
 * Load JDBC driver and connect to a database locally - versi
 *
 * @group:      LulzSec
 *
 * @author:     Spencer Bishop, Quynh Tran Nguyen, Andy Omori
 *
 * @course:     ICS 491
```

Figure 2.1: SQL injection testing using syntax <test’>. Failed testing.



SQL result

Host: mysql wampserver
Database: login
Generation Time: Aug 02, 20
Generated by: phpMyAdmin
SQL query: SELECT * FRO
Rows: 4

u_username	p_passwrd
12345	coffEEE
anderson	cooper
testingBOO	12345
tommy	hill

```
Login2a [Java Application] C:\Program Files (x86)\Java\jre1.8.0_51\bin\javaw.exe (Aug 2, 2015, 10:43:31 AM)
Connection Success
User entered username >> 'test''; password: ''
Database Connected
Connection Success
User entered username >> '' OR 1=1'; password: 'sjdfjdfsdj'
Database Connected
```

Login — Username: 'OR 1=1 — Password: *********** — Login

Message: Access Denied. Invalid Username or Password

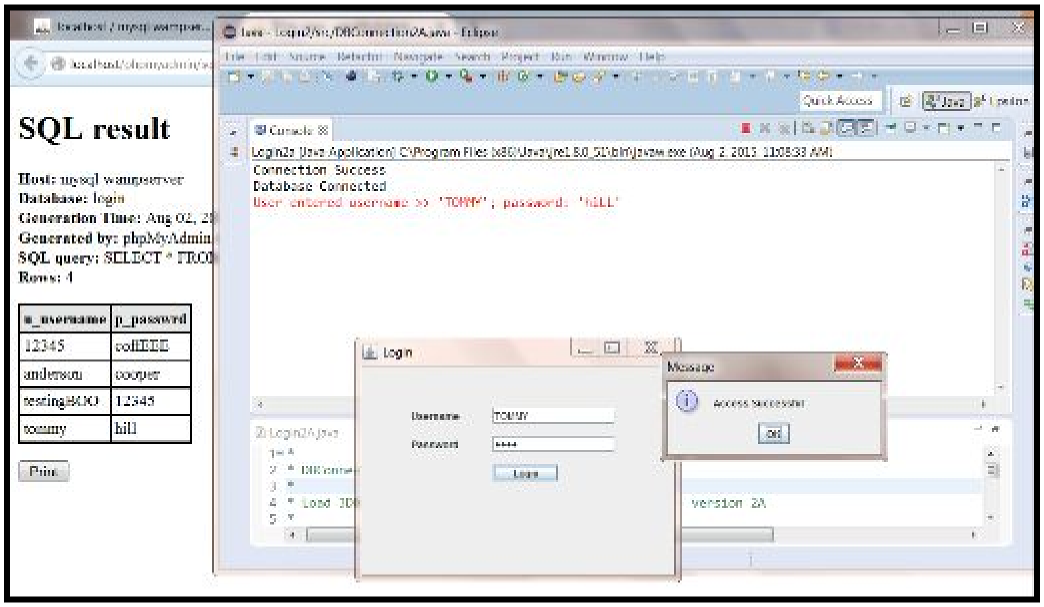
```
 * DBConnection2.java
 *
 * Load JDBC driver and
 *
 * @group:      LulzSec
 *
 * @author:     Spencer
 *
 * @course:     ICS 491
```

Figure 2.2: SQL injection testing using syntax <’OR 1=1>. Failed testing.

b. <u>Successful attacks</u>

Username and Password fields are not case sensitive

For this test, we tested files Login2A.java and DBConnection.java to see whether or not our program for login information window is case sensitive. We found that the information is not case sensitive. For instance, we conduct the test the using the login name “TOMMY” and the password “hiLL.” The result shows we are able to gain access even though the correct login name is “tommy” and the correct password is “hill”. This is big secure issue because the program is not validating user login/input correctly. The solutions are to sanitize user input and change the SQL command that retrieve data from the database. Change "<u>select usrname from login where usrname=? and passwrd=?</u>" to "<u>select usrname, passwrd from login where usrname=? and passwrd=?</u>".



Host: mysql wampserver
Database: login
Generation Time: Aug 02, 2
Generated by: phpMyAdmin
SQL query: SELECT * FRO
Rows: 4

u_username	p_password
12345	collEEE
anderson	cooper
testing800	12345
tommy	hill

```
Login2a [Java Application] C:\Program Files (x86)\Java\jre1.8.0_51\bin\javaw.exe (Aug 2, 2015 11:05:33 AM)
Connection Success
Database Connected
User entered username >> 'TOMMY'; password: 'hiLL'
```

Username: TOMMY
Password: ****

Access Successful

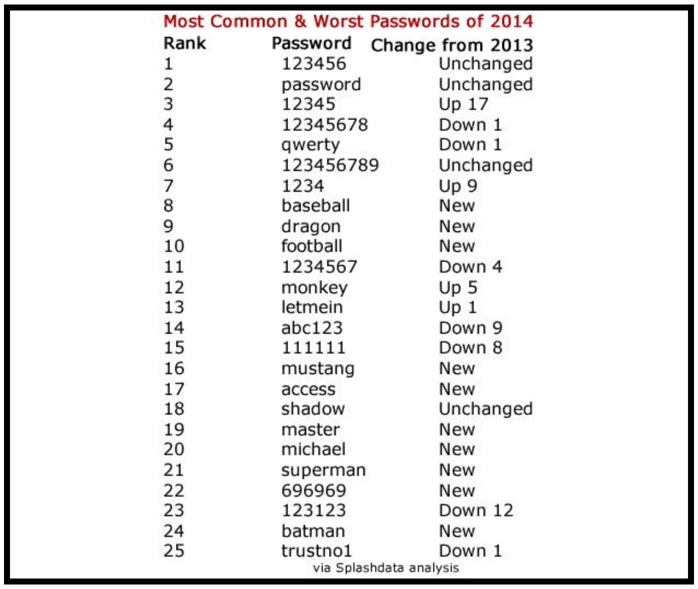
Figure 2.3: Successful attack on login window system. Testing shows the login information is not case sensitive.

c. <u>Brute Force & Social Engineering Attacks</u>

Brute force attacks are common and the most simplest test you could do. By simply trying every single common password combination, you could be able to attempt to break into a weak system. We performed two types of brute force -- one for password and another for user name. For our project, we just have a default password for now. We will change the password in the near future when we have fixed all major bug or design issue in our program. The password is currently set at default to eliminate confusing which of member has the right or wrong password. Leaving the password as the default is dangerous and a major security risk. Default password is simple and typically never change; therefore, it can be easily hacked and cause data tampering. In our second brute force attack, we tried to hack our program with some commonly used user names or default user names, such as “test”.

We also tested to see if we can bypass each other login information using social engineering. We were able to hack one of our members’ database by using what we know about that member; the member’s name was part of the login information.

These types of brute attacks and social engineering attack can be easily preventable by warning users that their passwords are too simple or too common when they register their login information and informing users to not use their name as part of their login information. Below is a list of commonly used password in 2004.



Most Common & Worst Passwords of 2014

Rank	Password	Change from 2013
1	123456	Unchanged
2	password	Unchanged
3	12345	Up 17
4	12345678	Down 1
5	qwerty	Down 1
6	123456789	Unchanged
7	1234	Up 9
8	baseball	New
9	dragon	New
10	football	New
11	1234567	Down 4
12	monkey	Up 5
13	letmein	Up 1
14	abc123	Down 9
15	111111	Down 8
16	mustang	New
17	access	New
18	shadow	Unchanged
19	master	New
20	michael	New
21	superman	New
22	696969	New
23	123123	Down 12
24	batman	New
25	trustno1	Down 1

via Splashdata analysis

Figure 2.4: Common passwords used in 2014

d. <u>Cross-site Scripting(XSS)</u>

Cross-site scripting(XSS) attacks are a type of injection in which a user maliciously injects scripts into a trusted website. Usually this is reserved for web applications but since we plan to put this up as a web app in the future we decided to test it using XSS. At first we thought it wouldn't have any effect as we were putting it in the login window, but still we had a pop up show up showing the username and password of what we entered. After looking at it in more detail we realized that the XSS attack didn't work and that all it did was pull up the console feature in Eclipse and show us what we would see at anytime we ran the application normally. The group found out later that the member who wrote this part of the code had done this on purpose for testing. Since our application is not published to the web we cannot predict how differently it would act after it has been released. We also believe that it might show a pop up for a non-admin account after being published allowing them to see something they shouldn’t, even if it’s their own login credentials. We believe this because when you run the program normally you would have to manually bring up the console feature to see the information.

3) Attack Surface Review

Originally, our database is hosted online through www.freemysqlhosting.net, so all our members would be using the same database -- that way it's convenience and keep our work organize. However, due to ethical and legal concerns, our group have decided to host our database locally. For our database, we decided on two database development software -- the latest version of SQL Workbench 6.3 and phpMyAdmin 4.4.12. There were some required software issues with install SQL Workbench for some of our members, so we finalized on phpMyAdmin 4.4.12.

In our program version 2.0, our group have fixed the database connectivity issue so this required installing and running new software. The following are new tools/software that we are using to work on version 2.0 of our program:

- phpMyAdmin 4.4.12 -- to host the database
- WampServer Version 2.5 -- to run Apache/2.4.9 (Win32) and PHP/5.5.12 on phpMyAdmin
- MySQL Connector/J 5.1.36 -- to connect JDBC driver to MySQL servers for Java.
- Firefox 39.0/Google Chrome 44.0.2403.125 m -- to run the database software locally.
- Java SE 8 u51 -- to run the program

There was a minor update to Java since our last update but it was not a critical update and didn’t need to be installed. Eclipse has not had any updates and neither has MySQL. We have added Windows 10 to our used programs as well and all seems to be working correctly.

We are currently working on encrypting the data in the database. Without fixing this issue, our program and online data are vulnerable to unauthorized access and data tampering. phpMyAdmin 4.4.12 provide a functionality to hash (i.e. using MD5 algorithm) or encrypt the data.

RELEASE

1) Incident Response Plan

a. <u>Escalation Response Team</u>

- Escalation Manager: Quynh-Tram Nguyen
- Legal Representative: Spencer Bishop
- Public Relations Representative: Andy Omori

Escalation Response Team contact information: uhmanoaasq@gmail.com

Escalation Manager Responsibility

Escalation manager is responsible for bringing order and structure during the escalation process. This include bringing the appropriate representation from across the organization, driving the completion of the process, and focusing management attention to the client and customer's problems in order to gain complete consumer satisfaction in LulzSec and its product.

Legal Representative Responsibility

Legal representative is responsible for obtaining and maintaining all necessary licenses for the software, and ensuring LulzSec meet all state, federal, and company guidelines, policies, and regulations during the program developing process and after the software is published.

Public Relations Representative Responsibility

Public relations representative is responsible for planning, directing, or coordinating activities designed to create or maintain a favorable public image or raise issue awareness for LulzSec and its customer. In addition, public relation representative must establish and maintain effective working relationships with clients, government officials, and media representatives and use these relationships to develop new business opportunities.

b. <u>Incident Response Plan</u>

Should a threat to the software occurs and the Incident Response has been notified of the threat, the Incident Response team must execute the following guidelines immediately to limit the effects of the threat, increase consumer and customer confidence, and reduce recovery time and costs.

1. Analyse and assess the threat to determine the proper response.
    a. Determine the size of the threat.
    b. Determine what data or property is threatened and how critical is it.
    c. Determine whether the threat is perceived or real.
    d. Determine whether or not the threat is still in progress.
    e. Determine what impact the threat has on business if the attack succeed -- minimal, serious, or critical.

    f. Determine what system or systems are targeted.

2. Create a ticket of the threat incident according to the following categories:
    a. Category one: a threat to public safety or like
    b. Category two: a threat to data
    c. Category three: a threat to computer system
    d. Category four: a disruption of services
3. Determine a proper response strategy based on threat analysis and assessment.
4. Backup data and system.
5. Contain the threat/incident: take action to prevent the cause of the problem and further intrusion or damage.
6. Use forensic techniques, including reviewing systems log, reviewing software codes, looking for gaps in the logs, reviewing intrusion logs, interviewing witnesses and any incident victims to determine the cause of the threat. Only authorized personnel may perform interviews and examine evidences.
7. Take action to prevent re-infection, which may include:
    a. Patching affected system.
    b. Shutdown any infected system until it can be re-installed.
    c. Re-install infected system from restore data from backup.
    d. Disable all unused services on affected system.

8. Contact the local authorities and/or the FBI to find the intruder or attacker to stop the threat, and take any legal action needed.
9. Restore infected system/network.
10. Document all findings on the threat/incident: how the threat was discovered, how the threat happened, where the threat came from, the response, and whether or not the response is effective
11. Preserve evidences: make copy of all communications (emails, logs, and other documentable communication).
12. Recommend changes to prevent occurrence from happening in the future or infecting other system.
13. Submit changes to management for approval. Implement changes once management has given approval.
14. Document all findings on the threat/incident: how the threat was discovered, how the threat happened, where the threat came from, the response, and whether or not the response is effective
15. Preserve evidences: make copy of all communications (emails, logs, and other documentable communication.
16. Informing the public, customer, and shareholder of the threat. Provide public relation outreach to anyone who’s affected by the threat.
17. Provide patches for software.
18. Update documentations as needed.

19. Assess opportunity and recovery cost -- estimate the cost for the organization to contain the threat, to fix it, and prevent in the future.
20. Review response and update policies -- plan and take preventative steps to prevent the threat from happening again.
    a. Determine if additional policy is necessary to prevent the threat.
    b. Determine whether the procedure or policy was followed that caused the threat, and consider what could be changed so the procedure and policy is followed in the future.
    c. Determine what we learned from the threat/incident.
    d. Determine whether or not the response was appropriate for the threat and if the response(s) works.

2) Final Security Review

We feel our program passed final security review. We have made sure our program runs securely and all avenues of attack have either been dealt with, or we have set up processes to respond to such instances. We give it a grade of “Passed FSR” since we have followed our threat model and implemented it in such a way that it follows protocol. There should not be a chance of SQL injections, since random inputs have been weeded out like our examples in the fuzzy testing. Also, we have secured our login database using a MD5 hash through a function in phpMyAdmin so that our database is protected -- if that fails then that is a vulnerability failure. Our Excel file shouldn’t have any problems because it is a completely separate file that we are grabbing from preventing database type injections from happening.

3) Certify Release and Archive

Team LulzSec’s PlayStation 4 program can be found at:

https://github.com/qtnguyen/ICS491-LulzSec-SDLProj