**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **17-June-2020** | **Name:** | **Raziya Banu** |
| **Course:** | **Introduction to Cyber Security** | **USN:** | **4AL16EC058** |
| **Topic:** | **Vulnerabilities & Password Security** | **Semester & Section:** | **8th sem & ‘B’ section** |
| **Github Repository:** |  |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  In my first session today I have studied about - Vulnerabilities & Password Security  Weak passwords and password reuse are still some of the most serious concerns for cybersecurity. There are several ways to increase password security but they are often not adopted by users and administrators. Here’s how you can make sure that sensitive data in your web application is not compromised due to insecure user passwords. Length or Complexity? The most common password policy enforced by administrators, both in the case of web applications as well as other systems, is a length and complexity policy. For example, a complex password may be required to contain at least 8 characters, uppercase and lowercase letters, numbers, and special characters. However, this policy is actually quite weak and should not be recommended.  Several websites exist, where you can check how long it would take to break your password using a brute-force attack. If you enter an 8-character password with numbers, uppercase/lowercase, and special characters in [How Secure Is My Password](https://howsecureismypassword.net/), it says that a computer could break your password in 9 hours. On the other hand, if you enter a 16-character password that uses only lowercase letters, the result is 224 million years. Is Length Enough? If password cracking were only based on the brute-force method (trying every single possible combination), password length would be the best way to practically make attacks impossible. However, there is also a cyberattack technique called dictionary attacks, which basically means password guessing based on commonly used words. For example, a password the quick brown fox jumps over the lazy dog would be cracked by a dictionary attack almost instantly. On the other hand, a password with exactly the same letters: vromjon tobki huhet qecar dzowyf xup selg, would be nearly impossible to crack.  Luckily, dictionary attacks are also very easy to avoid if you use fake words that are easy to memorize because of the way they sound. For example, you can use a long password such as bargle zaws gubble meh brudda dulgly. Those who know obscure languages have it even easier because most dictionary attacks are based on English vocabulary and several other popular world languages. For example, the password nista’ niekol il-ħġieġ, ma jweġġaniex could be considered quite secure.  Another interesting technique for creating secure passwords is using just the first couple of letters from each word to form a long password that is based on a real sentence. For example, you can make a safer password from the quick brown fox… sentence: thequibrofoxjumovethelazdog (although it still contains several dictionary words, which is not optimal). You can also, for example, take the first letter of every word from a longer phrase that you know well: skwklbfhkskppp (the Soft kitty song lyrics). Of course, in all these cases changing some letters into uppercase and adding numbers between the actual words even increases the security (for example, The7Qui0Bro1Fox2Jum3Ove4The5Laz6Dog).  Note that password breaking tools also replicate user tricks such as replacing some characters with numbers that look similar (e.g. 1 = i, 2 = z), reversing words (e.g. drowssap), adding a number at the end, etc. Therefore, passwords such as dr0w554P123! are easily cracked. The False Sense of Security Another very common mechanism used by web applications and other systems to increase password security is forcing the user to regularly change their password. Such mechanisms usually store the hashes for old passwords and therefore do not let the user reuse any of their previous passwords.  Unfortunately, this policy introduces a very false sense of security because users find easy ways to go around it. Let’s be honest, what do we do when the system asks us to update our password? We usually add the next consecutive number at the end and just keep replacing it every three months when asked (password1, password2, password3…).  This technique does not increase password entropy and does not in any way prevent dictionary attacks. Therefore, more and more big players including Microsoft are moving away from recommending regular password changes. Even large institutions such as FTC are now [recommending against this](https://www.ftc.gov/news-events/blogs/techftc/2016/03/time-rethink-mandatory-password-changes), so don’t implement this mechanism in your web application. The Danger of Password Reuse Even if you come up with the most secure password, it becomes insecure if you use it on every website and in every application. With the number of global data breaches, there is a big chance that your password for some site has been compromised. If you think this is unlikely, you may be surprised: just enter your email address on [Have I Been Pwned](https://haveibeenpwned.com/).  Luckily, most sites do not actually store passwords in plain text (although [even the biggest players on the market have been guilty of it](https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/)). This means, that in the case of a data breach, it’s only the password hash that is compromised. However, in many cases, the hashes are not secure. Many [web applications use old and easy to compromise hash algorithms](https://www.acunetix.com/blog/web-security-zone/insecure-default-password-hashing-cms/) such as MD5. In such a case, the attacker needs much less time to find the password on the basis of a hash. Of course, the simpler your password, the faster it will be compromised.  Unfortunately, there is no way to check for password reuse in your web application. Therefore, you must simply educate your users and trust that they follow your suggestions. Security of Password Managers To combat password reuse, we resort to password managers. They became more popular in recent years and there is quite a few to choose from for every platform. Many of them are very easy to use and cross-platform. This means that you can, for example, store your password database (secured with a single long and very complex password) in the cloud and access it from your PC, from your mobile, and from anywhere else via a web interface. Password managers combat password reuse and they can also generate complex unique passwords for you, although such passwords will not be easy to remember.  Some password managers make it even easier for web application users and introduce their own web browser plugins. Such plugins take the password out of the password database and automatically use it on the website. However, such plugins are the Achilles heel of password managers and should be avoided, as proven for example by the [latest LastPass vulnerability](https://www.forbes.com/sites/daveywinder/2019/09/16/google-warns-lastpass-users-were-exposed-to-last-password-credential-leak/). You will be much more secure if you manually open the password manager, copy the password, and paste it in the web application. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **17-June-2020** | **Name:** | **Raziya Banu** | |
| **Course:** | **Udemy** | **USN:** | **4AL16EC058** | |
| **Topic:** | **Java collection Frameworks** | **Semester & Section:** | **8th sem & ‘B’ section** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **ArrayList:** The first part of a series on the Java Collections Framework, an absolutely vital set of classes for organising data in your code. In this part we'll look at ArrayList; an expandable array. ArrayList is probably the most used and easiest to use member of the collections framework  App.java:  import java.util.ArrayList**;**  import java.util.List**;**  **public** **class** **App** **{**  **public** **static** **void** **main(**String**[]** args**)** **{**  ArrayList**<**Integer**>** numbers **=** **new** ArrayList**<**Integer**>();**  *// Adding*  numbers**.**add**(**10**);**  numbers**.**add**(**100**);**  numbers**.**add**(**40**);**  *// Retrieving*  System**.**out**.**println**(**numbers**.**get**(**0**));**  System**.**out**.**println**(**"nIteration #1: "**);**  *// Indexed for loop iteration*  **for** **(int** i **=** 0**;** i **<** numbers**.**size**();** i**++)** **{**  System**.**out**.**println**(**numbers**.**get**(**i**));**  **}**  *// Removing items (careful!)*  numbers**.**remove**(**numbers**.**size**()** **-** 1**);**  *// This is VERY slow*  numbers**.**remove**(**0**);**  System**.**out**.**println**(**"nIteration #2: "**);**  **for** **(**Integer value **:** numbers**)** **{**  System**.**out**.**println**(**value**);**  **}**  *// List interface ...*  List**<**String**>** values **=** **new** ArrayList**<**String**>();**  **}**  **}**  **HashMap:**  A tutorial on HashMap. Maps are data collections that function like lookup tables; basically you can store objects via "keys" (names, IDs, or even complex objects) and quickly retrieve them without having to look through an entire list.  App.java:    import java.util.HashMap**;**  import java.util.Map**;**  **public** **class** **App** **{**  **public** **static** **void** **main(**String**[]** args**)** **{**  HashMap**<**Integer**,** String**>** map **=** **new** HashMap**<**Integer**,** String**>();**    map**.**put**(**5**,** "Five"**);**  map**.**put**(**8**,** "Eight"**);**  map**.**put**(**6**,** "Six"**);**  map**.**put**(**4**,** "Four"**);**  map**.**put**(**2**,** "Two"**);**    String text **=** map**.**get**(**6**);**    System**.**out**.**println**(**text**);**    **for(**Map**.**Entry**<**Integer**,** String**>** entry: map**.**entrySet**())** **{**  **int** key **=** entry**.**getKey**();**  String value **=** entry**.**getValue**();**    System**.**out**.**println**(**key **+** ": " **+** value**);**  **}**    **}**  **}**    Six  2: Two  4: Four  5: Five  6: Six  8: Eight | | | |