ENPM809W - Introduction to Secure Coding

Lab - 4 – Defense Lab – Secure Cryptography

Author: Syed Mohammad Ibrahim
UID: iamibi
Email: iamibi@umd.edu

Phase 1: Fixing Password Storage

a) Provide the URL of the WebGoat.NET application page where you are exercising the question. Do not include any query parameters or any other special characters in the answer.

http://localhost:52251/WebGoatCoins/CustomerLogin.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

I am going to replace the Encoder.Encode method with a secure hashing algorithm SHA-256 with secure random salt of 64-bytes per password. To implement this, I would require rebuilding the database and change the schema of CustomerLogin since we must update all the passwords with their respective hash digests and store salt in a new column. Apart from this, I will be sanitizing the input of any HTML tags.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

Since the passwords are now converted to hash digests, it will be difficult to analyze as to what the passwords are, in contrast to just using Base64 decoder priorly. This significantly improves the password storing and verification mechanism. The verification now happens by generating a hash digest using the same salt of the user and then comparing the stored hash with the generated hash. This way, the password is secured at multiple levels.
P.S.: To test the functionality please rebuild the database.

d) List all the paths with filenames you are changing to implement the fix for the weakness.

- C:\Users\student\Workspace\webgoat\WebGoat\App_Code\DB\MySqlDbProvider.cs (Updated IsValidCustomerLogin() and UpdateCustomerPassword())
- C:\Users\student\Workspace\webgoat\WebGoat\App_Code\DB\DbMigrationScript.cs (New)
- C:\Users\student\Workspace\webgoat\WebGoat\App_Code\Hashing_Utility\PasswordWithSaltHasher.cs (New)
- C:\Users\student\Workspace\WebGoat\WebGoat\DB_Scripts\create_webgoatcoins.sql (Modified the CustomerLogin schema)
- C:\Users\student\Workspace\WebGoat\WebGoat\DB_Scripts\create_webgoatcoins_sqlite3.sql (Modified the CustomerLogin schema)
- C:\Users\student\Workspace\webgoat\WebGoat\DB_Scripts\datafiles\customerlogin.txt (Updated the user passwords and added salts)

e) Commit ID: c87df7c653263e4c7c4f240edc60bad0edc91e8b

Phase 2: Correct and Strong Hashing

a) Provide the URL of the WebGoat.NET application page where you are exercising the question. Do not include any query parameters or any other special characters in the answer.

   http://localhost:52251/WebGoatCoins/CustomerLogin.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

   As the last hashed passwords provided were cracked using Hashcat, I updated the same algorithm to use better standards, that is, using Rfc2898DeriveBytes with 512-bit salt and 100,000 iterations of PBKDF2. This makes breaking the hash difficult and would take a lot of time and resources to be cracked. Apart from this, sanitized the input text to escape any HTML tags if passed in.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

   The intended fix securely generates a secure random 512-bit salt per password and a digest which is derived from Key Derivative Function (PBKDF2) having 100,000 iterations. As of now, this is assumed to be a secure standard and will take a lot of time and resources to be cracked. I used this algorithm from ASP.NET core Rfc2898DeriveBytes which implements it. Using this algorithm, I recomputed all the hash

digests for the passwords and created a new column in the existing table to store the random salt per password. To finally update the database with the new hashes, I used Rebuild database functionality to fill up the database with the new entries from customerlogin.txt. More details in the code comments.
P.S.: To test the changes, please Rebuild the database after checking out the branch.

d) List all the paths with filenames you are changing to implement the fix for the weakness.

- C:\Users\student\Workspace\webgoat\WebGoat\App_Code\HashingAlgorithm.cs (New)
- C:\users\student\workspace\webgoat\WebGoat\App_Code\DB\DatabaseMigration.cs (New)
- C:\Users\student\Workspace\webgoat\WebGoat\App_Code\DB\MySqlDbProvider.cs: IsValidCustomerLogin() and UpdatedCustomerPassword()
- C:\Users\student\Workspace\WebGoat\WebGoat\DB_Scripts\create_webgoatcoins.sql (Modified the CustomerLogin schema)
- C:\Users\student\Workspace\WebGoat\WebGoat\DB_Scripts\create_webgoatcoins_sqlite3.sql (Modified the CustomerLogin schema)
- C:\Users\student\Workspace\webgoat\WebGoat\DB_Scripts\datafiles\customerlogin.txt (Updated the user passwords and added salts)

e) Commit ID: 30fa88f322ba69b79ddff73e5b53ca84c3f72525

Phase 3: Secure RNGs

a) Provide the URL of the WebGoat.NET application page where you are exercising the question. Do not include any query parameters or any other special characters in the answer.

   https://github.com/osorin/srp4net/blob/master/SRP/SRP.cs#L91

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

   Using a secure random generator provided by the System.Security.Cryptography library, we can use RNGCryptoServiceProvider class to generate secure random bytes of fixed length.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

The fix will make sure that secure random bytes are generated. These bytes can be converted to an integer by the methods available in C# library.

d) List all the paths with filenames you are changing to implement the fix for the weakness.

- https://github.com/osorin/srp4net/blob/master/SRP/SRP.cs

e) Change line of code:
   a. using System.Security.Cryptography;
   b. Change the line at https://github.com/osorin/srp4net/blob/master/SRP/SRP.cs#L91 to use the following piece of code

      ```
      RNGCryptoServiceProvider rngCryptoServiceProvider = new
      RNGCryptoServiceProvider();
      byte[] randomBytes = new byte[_nbits];
      rngCryptoServiceProvider.GetBytes(randomBytes);
      ```

      The randomBytes array can be converted to BigInteger or whichever datatype is required.

Phase 4: Using CPSA to identify cryptographic protocol vulnerabilities

a) Provide the URL of the WebGoat.NET application page where you are exercising the question. Do not include any query parameters or any other special characters in the answer.

   N/A

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

   The MITM weakness in the Item 4 is that the attacker has access to basic data and can act as a sender for a genuine user.

   ```
   (defskeleton dh_mim
     (vars (n text) (y x rndx))
     (defstrand init 3 (n n) (h (exp (gen) y)) (x x))
     (defstrand resp 3 (n n) (h (exp (gen) x)) (y y))
     (precedes ((0 0) (1 0)) ((0 2) (1 2)) ((1 1) (0 1)))
     (absent (y (exp (gen) x)))
     (pen-non-orig y x)
   ```

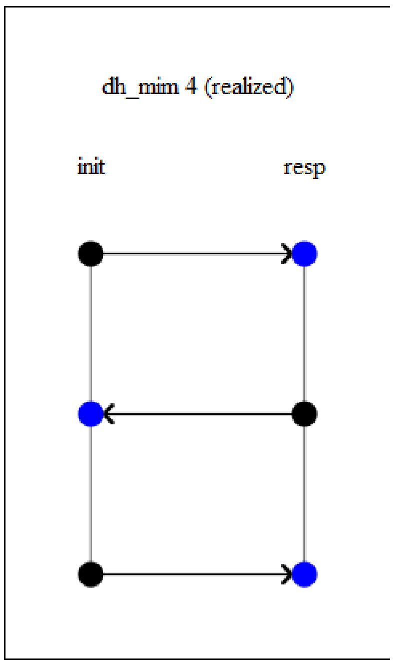
```
(uniq-gen y x)
(uniq-orig n)
(operation nonce-test (displaced 2 0 init 1) (exp (gen) x-0) (1 0))
(label 4)
(parent 1)
(unrealized)
(shape)
(maps
  ((0 1) ((n n) (hx (exp (gen) x)) (hy (exp (gen) y)) (x x) (y y))))
(origs (n (0 2))))
```

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

   An attacker with enough basic data can imitate a legitimate user and act as a sender between two parties. This can cause the receiver to be tricked and send critical information to the attacker.

d) List all the paths with filenames you are changing to implement the fix for the weakness.

Item 4. Parent: 1.



e) Commit ID: N/A