ENPM809W - Introduction to Secure Coding

Lab - 3 – Attack Lab – Breaking Encryption

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

Phase 1: Improper 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

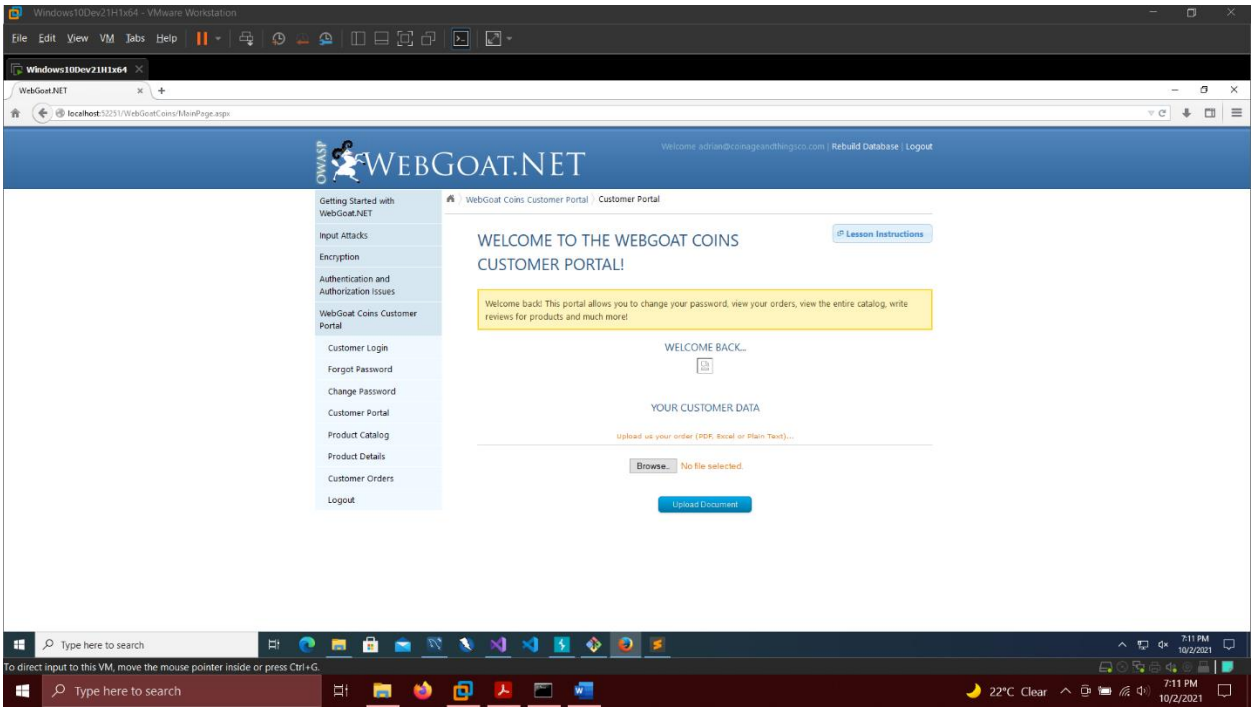
b) Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each. If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.

The input to the customer login form was one of the customers with the following credentials:
adrian@coinageandthingsco.com
rainbow

c) Describe the output result.

I was able to login with the above user credentials successfully in the Customer Login portal.

d) Provide a screenshot of the output.



e) Provide the CWE-ID, Filename, Line Number of the weakness that is at the heart of the vulnerability.

CWE-ID: CWE-916 - Use of Password Hash with Insufficient Computational Effort
Filename: MySqlDbProvider.cs
Line Number: 115

f) Describe the vulnerability and what role the weakness plays in allowing the input (attack vector) to compromise the application.

The vulnerability lies in the way the passwords are expected to be secured. In the current context, the passwords are being stored as Base 64 encoded strings in the database. Usage of Base 64 encoding does provide a little bit of ciphertext, but it is insignificant for a machine to recalculate the hash or if the attacker tries to decode the string using a Base 64 decoder.

Phase 2: Hash Cracking

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.

NA

b) Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each. If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.

The were two inputs to Hashcat.

1. The base hashes present in hashes.txt
   sha256:10000:wBCFkC1NTOXlQxwh5LNFXw==:R/SRWYWdgQcwdHFwG4extfdQjsmIhc4jDFDRns+SqTM=
   sha256:10000:8ZIYTt76CeTqdBsXV1qtOA==:uVEYFou9iMkembgILy6A5QWMr5hGqDTfTObLAZhUaBY=

2. The dictionary that was used to perform the brute-force attack (combined_seclists_password_list.txt)

Command executed
.\hashcat.exe -a 0 -m 10900 -w 3 .\hashes.txt .\SecLists-master\Passwords\combined_seclists_password_list.txt -o .\cracked.txt -O

c) Describe the output result.

The output was written to a cracked.txt file.

1. sha256:10000:wBCFkC1NTOXlQxwh5LNFXw==:R/SRWYWdgQcwdHFwG4extfdQjsmIhc4jDFDRns+SqTM=:Passw0rd!
2. sha256:10000:8ZIYTt76CeTqdBsXV1qtOA==:uVEYFou9iMkembgILy6A5QWMr5hGqDTfTObLAZhUaBY=:kmitnick

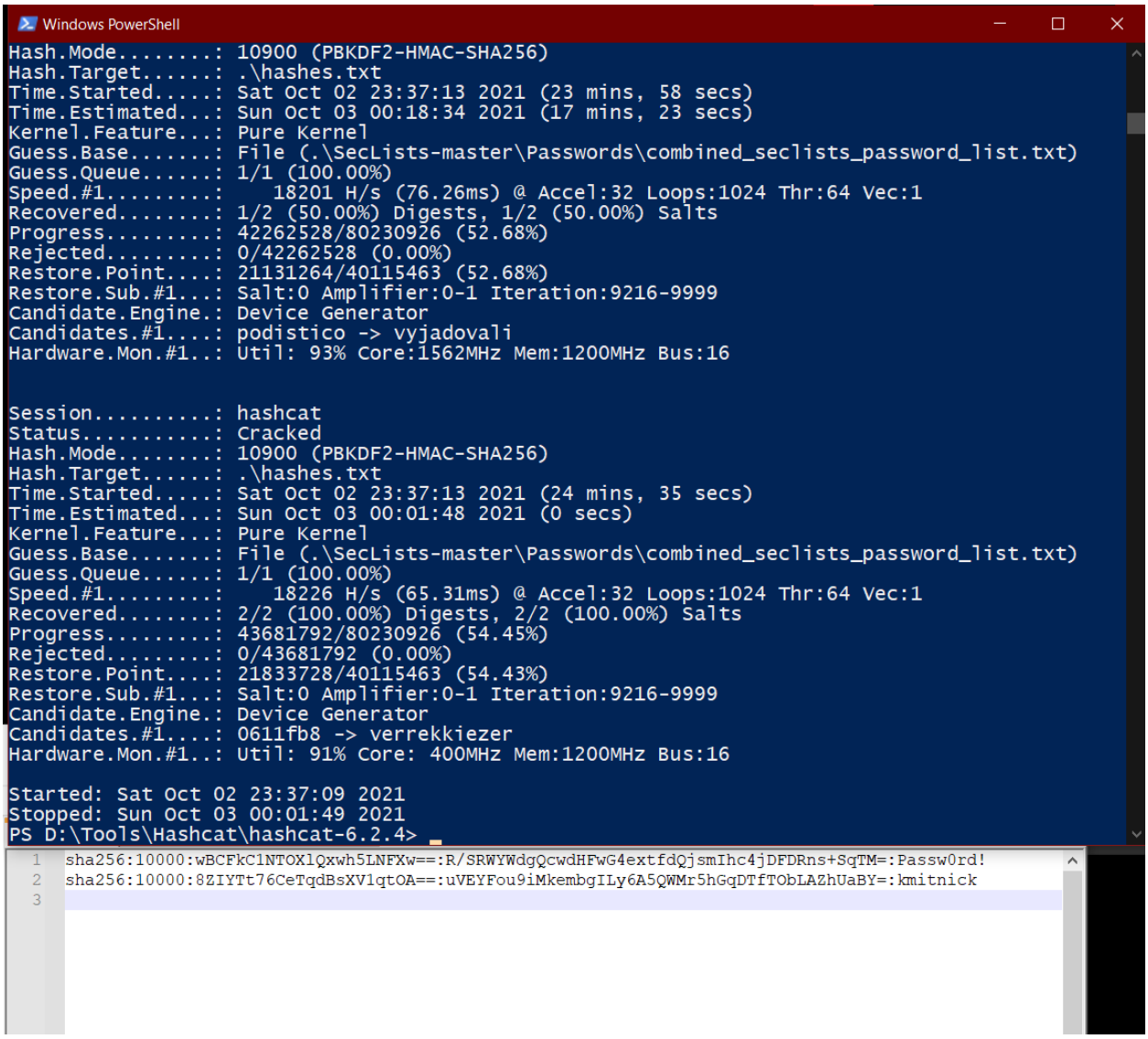
Logs
Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 10900 (PBKDF2-HMAC-SHA256)
Hash.Target.....: .\hashes.txt
Time.Started.....: Sat Oct 02 23:37:13 2021 (24 mins, 35 secs)
Time.Estimated...: Sun Oct 03 00:01:48 2021 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (.\SecLists-master\Passwords\combined_seclists_password_list.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 18226 H/s (65.31ms) @ Accel:32 Loops:1024 Thr:64 Vec:1
Recovered.....: 2/2 (100.00%) Digests, 2/2 (100.00%) Salts
Progress.....: 43681792/80230926 (54.45%)
Rejected.....: 0/43681792 (0.00%)
Restore.Point....: 21833728/40115463 (54.43%)

Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:9216-9999
Candidate.Engine.: Device Generator
Candidates.#1....: 0611fb8 -> verrekkiezer
Hardware.Mon.#1..: Util: 91% Core: 400MHz Mem:1200MHz Bus:16

Started: Sat Oct 02 23:37:09 2021
Stopped: Sun Oct 03 00:01:49 2021

Total time taken: 24 mins, 35 seconds

d) Provide a screenshot of the output.



e) Provide the CWE-ID, Filename, Line Number of the weakness that is at the heart of the vulnerability.

NA

f) Describe the vulnerability and what role the weakness plays in allowing the input (attack vector) to compromise the application.

The issue is with the weak hashing algorithm being used that seems to use PBKDF2 of 10,000 rotations. A strong hardware will be able to crack the password, given the time and knowledge such as Hashcat. If the rotations are increased to 100,000, it will make the hashes a little more secure with a bit of performance hit. Also, choosing a strong password which comprises a mix of uppercase, lowercase, digits, and numbers and has a length of at least 8 characters.

Phase 3: Insecure 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.

N/A

b) Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each. If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.

N/A

c) Describe the output result.

N/A

d) Provide a screenshot of the output.

N/A

e) Provide the CWE-ID, Filename, Line Number of the weakness that is at the heart of the vulnerability.

CWE-ID: CWE-330: Use of Insufficiently Random Values, CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
Filename: SRP.cs
Line Number: 91

f) Describe the vulnerability and what role the weakness plays in allowing the input (attack vector) to compromise the application.

The vulnerability lies in the random number generator that is being used as part of the development process. The “System.Random” method uses the system’s clock to generate a random number for salt, which in all its sense is random but not cryptographically strong. Because of this, an attacker can predict the algorithm’s salt and potentially can run it across a brute-force or dictionary-based attacks. Instead, using “System.Security.Cryptography.RNGCryptoServiceProvider” to generate a secure random number which is cryptographically strong.