Attack Lab 3: Breaking Encryption

Date Due: Tuesday, October 5, 2021, by 11:59 PM.

Introduction

This lab will demonstrate some of the common encryption and password related weaknesses in web applications and some of the attacks that make those weaknesses dangerous. This assumes you have completed Lab 0 which involves performing the Lab environment setup. For this lab, you will need to run WebGoat.NET application which was performed as part of Lab 0: Phase 2.

WARNING: Do not attempt to use these techniques elsewhere. Only run them on the WebGoat.NET application either inside the virtual machine or the application running on your own machine. There are legal consequences to you if you use these elsewhere!

For each of the following on the WebGoat.NET application, please submit a single writeup containing all the phases and the answers to each of the questions provided in each phase. For each question, please look at the Lesson Instructions button on the given page unless specified otherwise. The answers should be provided in the following format as part of a writeup in Word or PDF:

Question Number.

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

Phase 1: Improper Password Storage (10 points)

The WebGoat.NET application, if you have noticed, uses a Login page (link at the top right of the page). The login page belongs to the WebGoat Coins Customer Portal. The WebGoat Coins Customer Portal application stores passwords insecurely. Your task is to find out how it is stored. You may use the MySQL Workbench or DBeaver application to explore the WebGoat database or look at the source code to figure out how passwords are stored.

After you have discovered how the passwords are stored, go to the 'Customer Login' page under the WebGoat Coins Customer Portal section, and login as one of the customers. Now answer the questions at the top of this lab handout based on what you performed and found.

Phase 2: Hash Cracking (10 points)

As you saw in the earlier phase, storing passwords incorrectly can lead to a breach in sensitive user data. You might think that the use of stronger cryptographic hash algorithms can be used and still store the password as is. ASP.NET Identity services Version 3 for example uses PBKDF2 with HMAC-SHA256 with a 128-bit salt, 256-bit subkey, and 10,000 iterations to store the password (See https://salslab.com/a/what-algorithm-does-aspnet-core-identity-use-to-hash-passwords/). However, as you will see in this phase, these are still subject to dictionary-based attacks.

A popular hash cracking application used is called hashcat (https://hashcat.net/hashcat/). **Please download and install a portable version on your computer (fast) or the VM (slow).** It takes advantage of graphics cards to increase throughput of cracking attempts combined with rules to trying different password combinations in order to crack password hashes stored in different formats. One of the formats supported is PBKDF2 with HMAC-SHA256. **Note: This phase may take much longer than the others.**

I have dumped an ASP .NET MVC application database containing identity information and already run the password through a utility identity-to-hashcat (https://github.com/edernucci/identity-to-hashcat) which converts the password into a format hashcat understands. Put the following lines into a text file called hashes.txt and attempt to run a brute-force password guess using the dictionary file from SecLists (https://github.com/danielmiessler/SecLists/tree/master/PasZZZswords) as done by the author of this blog (https://laconicwolf.com/2018/09/29/hashcat-tutorial-the-basics-of-cracking-passwords-with-hashcat/):

```
sha256:10000:wBCFkC1NTOXlQxwh5LNFXw==:R/SRWYWdgQcwdHFwG4extfdQjsmIhc4jDFDRns+SqTM=
```

```
sha256:10000:8ZIYTt76CeTqdBsXV1qtOA==:uVEYFou9iMkembgILy6A5QWMr5hGqDTfTObLAZhUaBY=
```

For this phase, only answer items b), c), d) and f). For item a) put N/A and for item c) where you describe the output result, give the plaintext password and also include how long it took for you to guess the two passwords using hashcat. A part of this exercise is for you to figure out how to use hashcat.

Phase 3: Insecure RNGs (10 points)

Sorin Ostafiev implemented an experimental cryptographic protocol called Secure Remote Password for .NET designed by Prof. Thomas Wu at Stanford University (http://srp.stanford.edu/ndss.html) that allows a user to login to a remote service using a chosen password directly (that is without the use of secure tunneling protocols like TLS etc.). It was also submitted to the IETF under an RFC (https://www.ietf.org/rfc/rfc2945.txt). Take a look at Sorin's experimental code available at https://github.com/osorin/srp4net. For this phase answer the following question: Does it look like he used a secure random number generator as part of the protocol implementation? **Please only answer items e) and f), with N/A for all other items.**

Submission Criteria

Please submit a writeup with:

1. Your name, UMD email ID and Lab Number.
2. The answers provided in the format given at the top of this lab handout as specified in each of the phases.

Please only submit a **Word document** or a **PDF**. There is no code submission for this lab.