Attack Lab 3: Breaking Encryption

Date Due: Friday, October 6, 2023, 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 the WebGoatCore 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 WebGoatCore 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!

IMPORTANT

Reset your local database

Once you have pulled the new changes, reset your database before attempting this attack lab. Make sure that you have exited Visual Studio first. Next, please delete the **bin** folder located in the **webgoatcore/WebGoatCore/bin**. Then open the Visual Studio and perform a clean operation by going into **Build** > **Clean**. Now you can run the IIS Server which will clone the NORTHWIND.sqlite database located in the root of the project automatically.

Submission Instructions

For each of the Phases provided in this lab, you will be exercising attacks on the WebGoatCore 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, the answers should be provided in the following format in a Word or PDF document:

Question number.

a) Describe in words 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. If BurpSuite was used, **copy the cURL command from BurpSuite** and provide it as part of this answer.
b) Describe the result (the output as well as any effects that were not seen).
c) Provide screenshot(s) of any output or effects observed.
d) Provide any applicable CWE-ID, Filename, Line Number of all the weaknesses that relate to the vulnerability. Please refer to the **ENPM809W Project handout Phase 3: Code Review Findings** section for a template of how to report this. You may have multiple findings related to the vulnerability you attacked.

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

Phase 1: Improper Password Transport (10 points)

The WebGoatCore application, if you have noticed, uses a Login page (link at the top right of the page). You may use the DBeaver application to explore the Northwind database or look at the source code to figure out how passwords are stored and how they are transported between the server and the client browser.

Go to the Login page, and login as the user 'MyUser' with password 'MyPassword'. Now press the Logout button at the top right. Now try to register a new user with username as 'newuser', email as 'newuser' and with password 'aa'.

Now answer the questions at the top of this lab handout based on what you just performed and found.

Phase 2: Hash Cracking (10 points)

As you saw in the earlier phase, insecure passwords 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. You may use the DBeaver application to explore the Northwind database or look at the source code to figure out how passwords are stored.

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 try different password combinations in order to crack password hashes stored in different formats. One of the formats supported is PBKDF2 with HMAC-SHA256. **Please note: This phase may take much longer than the others.**

Assuming we have a dump of the WebGoatCore ASP .NET Core application database containing identity information and access to 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/).

Now using identity-to-hashcat, obtain the hash and salt for users 'chester' and 'HarveySpecter' and attempt to derive the original password. The Combined Passwords file from the Daniel Miessler article has been provided on ELMS in the Week 5 module. **Make sure you echo your name and UID in the screenshots for hash cracking.**

For this phase, only answer items b), c), d) and e). For item a) put N/A and for item b) 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 d) and e), with N/A for all other items.**

Phase 4: Inadequate Validation of Cryptographic Signatures (10 points)

In the "About" page of the application, a security weakness has been identified concerning file uploads. Investigate the file upload functionality and describe the observed behavior when uploading files with reference to their naming convention. Explain why this behavior may be a security concern in the context of data integrity and authenticity. Now answer the questions at the top of this lab handout based on what you just performed and found.

Hint:

Consider the importance of maintaining data integrity and ensuring that data from different sources or versions is not mixed. And remember this is in reference to cryptographic signatures.

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.