CSCI 345 – Homework

TCP/IP stack, XSS, SQL Injection

1. Cryptography (20 pts)

a. DES

(*weak DES keys*) There are four so called *weak* DES keys. One of those keys is

K = 00011111 00011111 00011111 00011111 00001110 00001110 00001110 00001110.

a) What happens if you use this key?

b) Can you find the other three weak keys?

b. RSA

In this exercise we consider an RSA modulus n = p x q where p and q are large prime numbers (here, by large we mean at least equal to 5). We consider a valid RSA exponent e for RSA.

a. Show that neither (p mod 3) nor (q mod 3) can be equal to 0.
b. Under which condition e is a valid exponent for a modulus n?
c. From now on, we will assume that e = 3. Show that neither p − 1 nor q − 1 can be multiples of 3.
d. Deduce that p mod 3 = q mod 3 = 2.
e. What is the value of n mod 3?

2. TCP/IP stack (20 pts)

Complete the following GENI experiment: https://witestlab.poly.edu/blog/tcp-ip-protocol-stack/ . There is an exercise at the end of this webpage. Submit this exercise in place of your experiment report.

3. IP Routing:

Complete the following GENI experiment: https://groups.geni.net/geni/wiki/GENIEducation/SampleAssignments/IPRouting/Procedure. Submit a report according to the instructions at the last part "What to hand in".

4. XSS and SQL Injection (60 pts)

a. For this assignment you will use the *OWASP Juice Shop Project*, an intentionally vulnerable e-commerce website.
b. Use Docker for your installation – the goal is to learn about containers! https://github.com/bkimminich/juice-shop#setup
c. You will need to perform the tasks listed below. There will be hints for each task:
    i. *Perform a reflected XSS attack with <script>alert("XSS1")</script>*
    Background: Reflected Cross-site Scripting (XSS) occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users

who open a maliciously crafted link or third-party web page. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.

Hints

1. Look for an input field where its content appears in the response HTML when its form is submitted.
2. Try probing for XSS vulnerabilities by submitting text wrapped in an HTML tag which is easy to spot on screen, e.g. <h1> or <strike>.

ii. Access the administration section of the store.

Hints

1. Knowing it exists, you can simply guess what URL the admin section might have.
2. Alternatively, you can try to find a reference or clue within the parts of the application that are not usually visible in the browser

iii. *Perform a persistent XSS attack with <script>alert("XSS2")</script> bypassing a client-side security mechanism.*

Background: This challenge is founded on a very common security flaw of web applications, where the developers ignored the following golden rule of input validation: **Be aware that any JavaScript input validation performed on the client can be bypassed by an attacker that disables JavaScript or uses a Web Proxy. Ensure that any input validation performed on the client is also performed on the server.**

Hints

1. There are only some input fields in the Juice Shop forms that validate their input.
2. Even less of these fields are persisted in a way where their content is shown on another screen.
3. Bypassing client-side security can typically be done by either disabling it on the client (i.e. in the browser by manipulating the DOM tree) or by ignoring it completely and interacting with the backend instead.

iv. *Order the Christmas special offer of 2014.*

To solve this challenge you need to order a product that is not supposed to be available any more.

Hints

1. Find out how the application hides deleted products from its customers.
2. Try to craft an attack string that makes deleted products visible again.
3. You need to get the deleted product into your shopping cart and trigger the Checkout.

v. BONUS POINTS: Retrieve a list of all user credentials via SQL Injection (3 pts)

This challenge explains how a considerable number of companies were affected by data breaches without anyone breaking into the server room or sneaking out with a USB stick full of sensitive information. Given your application is vulnerable to a certain type of SQL Injection attacks, hackers can have the same effect while comfortably sitting in a café with free WiFi.

Hints

1. Try to find a page where you can influence a list of data being displayed.
2. Craft a UNION SELECT attack string to join data from another table into the original result.
3. You might have to tackle some query syntax issues step-by-step, basically hopping from one error to the next

What to submit:

A single document report named <LastName1>_<LastName2>_HW2 that includes:

1. Answers to questions in experiments 1, 2

2. A short description of your thinking process and a screenshots of the successful result for each challenge in part 3. *If you do not complete a challenge, describe your work and thought process for partial credit.*