CS 5435 – Security and Privacy Concepts in the Wild

Homework 3

Name: Samhitha Tarra st786, Martin Eckardt me424 *Fall 2022*

Question 1.5

1. Why does using GCM prevent mauling and padding oracle attacks?

   GCM prevents attacks like mauling and padding oracle because it doesn't use padding. Instead of using block ciphers it uses stream ciphers so each block is encrypted with a pseudo random number that has been generated. In CBC mode each block is XORed using the iv as the previous block which makes it susceptible to the above attacks. Due to these factors GCM mode cannot be exploited in the same way because we cannot utilize the padding to brute force recover byte values nor can we strategically XOR values to manipulate the bits.

2. For each attack above, explain whether enabling HTTPS for the entire payment site (as opposed to just the login page) prevents the attack if no other counter- measure is applied.

   Enabling HTTPS will not prevent against a CBC oracle padding attack or a cookie malleability attack because enabling HTTPS will just encrypt all network traffic but the attacks themselves will still be possible and need other countermeasures to protect against them. Due to the fundamental way CBC mode encryption operates the attacks will still be possible through byte manipulation and brute force. An attacker will still be able to query in order to test to see if valid padding is given and brute force recover each byte and with cookie mauling once they have the hex they can still flip the bits through XOR operations.

Question 2.3

SipHash is designed in such a way that even if an attacker gets to see a series of hashes of some values, he does not learn anything about the key. This means that the attacker can only bruteforce possible keys and must compute bucket collisions for each possible key. The number of calculated collisions does not have to be so high that the server crashes, but only so high that the attacker can tell that he has found the right key from an extended response time. Then he can compute a large number of bucket collisions and send the request. If the attacker has access to sufficient resources, he can exploit this vulnerability for a denial of service attack.

This can be fixed by sampling a fresh random SipHash key per request. However, it would be even better to implement a collision-resolution mechanism in the Hash Table.

Question 2.4

If the bucket count is unknown to the attacker he has three options: Guessing the bucket count, computing collisions that work for several different bucket counts, and testing attacks with different bucket counts.

If the space of possible bucket counts is too large, the first two option render unfeasible. This leaves the attacker to the latter approach. He sends several requests where each is including bucket collisions for one particular bucket count. The attacker then monitors if the server takes longer to respond to requests with certain bucket counts. This way he can determine the bucket count.

Question 2.5

Proof of Work is an effective countermeasure when the attacker uses a large number of requests that are cheap for him to produce and difficult for the server to process. It can make these requests more expensive for the attacker. However, the amount of work for a single request must be reasonable so that normal users can use the service without too much cost.

In the type of denial-of-service attack described above, the attacker does not rely on a large number of requests, but on a single request that exhausts the servers resources. Therefore, proof of work is unsuitable as a defense against the attack described above. The computational effort is negligible compared to the work required to find the collisions.