***Himesh Buch***

***[CS 419] Computer Security***

***HW2***

***3/25/2020***

1. CBC improves the credibility of the ciphertext by getting rid of the repeated patterns in the plaintext. Likewise, recreating or erasing ciphertext will degenerate in any event one block of the message. We will look at this XOR table to understand bit flipping using CBS mode

|  |  |  |
| --- | --- | --- |
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

We are going to use the switch analogy when considering CBS mode for bit flipping. In the first two rows of the table, the first bit A is 0, so the second bit B is the same as the output. In the last two rows, bit A is 1 so the opposite of bit B is the same as the output. We notice here that changing bit A no matter the value of bit B always causes the output to alternate or flip A with respect to B. We can view the first bit here as the switch to go along with our analogy. The attacker in the attack decides which bits to change and these consistent “flips” cause the next bit to change. In CBS, the mixing of plaintext and cipher text happens before the encryption. The first bit of prior ciphertext is XORed with the first bit of plaintext. If bit 3 of the previous ciphertext (or IV) is switched, bit 3 of the plaintext will be flipped before the encryption.

The attack is particularly dangerous when the attacker knows the structure or format of the message. In such a circumstance, the attacker can transform it into a comparative message in which some significant data is modified. For instance, an adjustment in the destination address may change the message course such that it will drive re-encryption with a more vulnerable figure, hence perhaps making it simpler for an attacker to translate the message

In order to defend against it, we can use a keyed message authentication code, digital signature, or other authentication mechanism as it allows us to detect if any bits were flipped in transit.

* 1. Here N = 91 and e = 5

Computational formula for ‘d’ in RSA is,

e-1mod (phi)(n) -----(1)

Phi(91) = phi(7) x phi(13)

= 6 x 12 = 72

Now, using eq (1).,

d = 5-1 mod (phi)(72) = 5(phi)(72) – 1 mod 72 ------(2)

Now,

Phi(72) = phi(23) x phi(32)

= (23-22) x (32-3)

= 4 x 6 = 24

Looking at eq (2),

We can say d = 524 – 1 mod 72 = 523 mod 72

Doing that calculation, we get **d = 29**

* 1. M1 = 8, N = 91, and e = 5

Computational formula for cipher text is c = M1e mod N

c = 85 mod 91 **= 8**

* 1. M2 = 17, N = 91, and e = 5

c = M1e mod N

= 175 mod 91 = 75

To verify this result, we calculate M = ed mod (phi)(N), where d = 29 (found from part a), we get,

M = 7529 mod 91 **= 17**

1. As explained in the question, passive attackers have access to ga and gb which are the public keys of Alice and Bob respectively, and the goal is for them to output the gab. On a mathematical level, the Diffie-Hellman key trade depends on single direction for its security. These are computations which are easy to do one way, however substantially harder to figure backwards.

Diffie-Hellman problem accepts that under the correct parameters, it is infeasible to compute gab from the different estimations of g, ga and gb. There is presently no publicly known approach to easily discover gab from different values, which is the reason the Diffie-Hellman key trade is viewed as secure, regardless of the way that aggressors can capture the values of g, a, and b.

1. As long as it is executed close by a fitting authentication method and the numbers for g, a, and b have been chosen appropriately, it isn't viewed vulnerable against assault. The security of the Diffie-Hellman key trade is reliant on how it is implemented. As we explained above, it has no methods for verifying the other party by itself, but usually systems are utilized to guarantee that the other party in the association isn't an impostor. Keeping this flaw of not knowing the other party of the algorithm, we can design an attack like this.
2. Assuming the first person is Alice and the other person with whom Alice is communicating is an imposter, say Chuck
3. Now, Alice has the public key ga and private key a, while Chuck has public key gc and private key c.
4. Alice doesn’t know who Chuck is (in our case it’s an imposter) and start the conversation by sending its key ga to Chuck, and Chuck sends its key gc to Alice
5. Now, as the algorithm forwards, Chuck will eventually get hold of the shared key gac. If the same process is repeated with Bob instead of Alice, it will result in the exact same output and Chuck will have the shared key gbc. In the end, it won’t be a hard task for the active adversary to figure out the shared key gab.
6. The game is between the adversary and the challenger:
7. First the challenger generates an encryption keypair and sends the public key to the adversary. (It keeps the secret key.)
8. Next, the adversary selects a pair of messages (of equal length) and sends them to the challenger.
9. The challenger picks a random bit and encrypts one of the two messages with the primary key and sends it back to the adversary.
10. Finally, the adversary guesses the answer. Adversary “wins” if it guesses correctly. It has a probability of ½ to win

Since, the entire process of winning and losing is based on guessing the result, we should not consider the win or lose of the adversary at all. Since it’s the game of guess, we should be more interested in how well the adversary guesses

Here is another approach at winning and at the game. From step 1 of our game, we notice that the attacker gets the public key, keeping that in mind:

1. The adversary picks two messages and encrypts both of them using the public key.
2. When the adversary receives the ciphertext in step (3), it can compare that ciphertext to the two messages he generated.

And by this way, the adversary can always figure out which message was encrypted.

Now comes the question of randomization. If the encryption scheme is not randomized — meaning that every time you encrypt a message using a given public key, you get the same exact ciphertext — this attack works perfectly that an attacker who uses above explained strategy will always win the IND-CPA game and hence, in order to satisfy the IND-CPA definition, any public-key encryption scheme must be randomized. The randomness here will also help to acquire semantic security

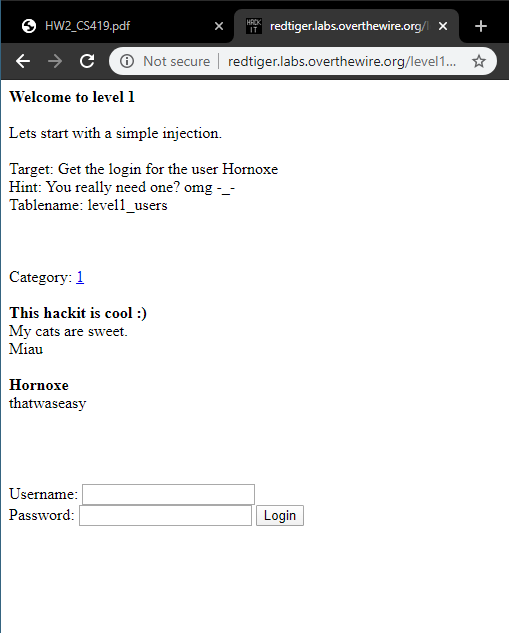
In short: ciphertexts must be expanded here and there so two unique encryptions of the equivalent plaintext don't create the equivalent ciphertext. On the off chance that this was not the situation, at that point the attacker could, without much of a stretch, dominate the game.

1. Level 1:

Clicking on “Category: 1”, we see the website url change to “<https://redtiger.labs.overthewire.org/level1.php?cat=1>”



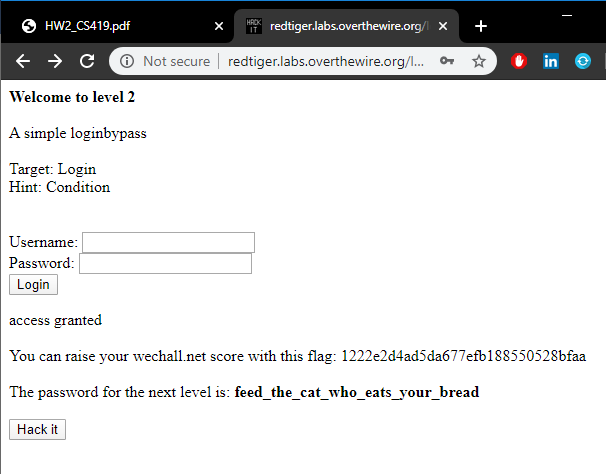
Now, we using this url “[http://redtiger.labs.overthewire.org/level1.php?cat=1 Union select 1,2,username,password from level1\_users](http://redtiger.labs.overthewire.org/level1.php?cat=1%20Union%20select%201,2,username,password%20from%20level1_users)”, which is a basic sql injection will get us the password. We are also given the name of the table on the page. Going to the above url, we get:



As seen in the screenshot, we got the password “thatwaseasy” for the username “Hornoxe” which was given already.

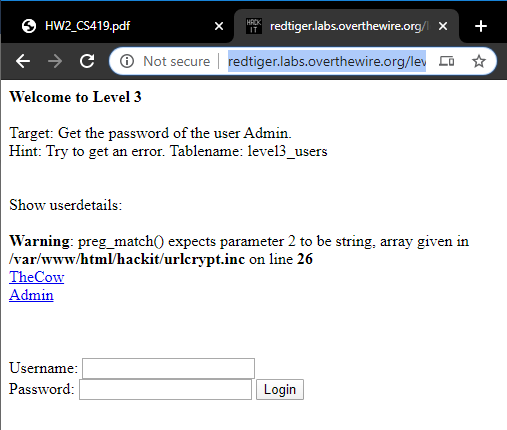
Level 2:

The question itself says that it’s a simple login hack. That gives us the information on using some generic sql statements that can be used for login. Looking at the hint, we know for sure that it is a condition, meaning it cannot be just a string. Looking at the most common username and passwords we get username and password both to be the same as “a ‘or’ ‘=’”



Level 3:

The first thing we notice here is that the hint is given as “Try to get an error”. This gives us the idea of making an error and expecting something out of it. We know that the username is given but in order to make error we use this url, [http://redtiger.labs.overthewire.org/level3.php?usr[]=error](http://redtiger.labs.overthewire.org/level3.php?usr%5b%5d=error), which gives us this output:



This is as far as I could get for this problem. I assume I will get more information if I visit the url that it pointed out and carefully analyze the .inc file.