Michael Prelich
20424478
mprelich

1) a)

i) We wait for Alice to initiate the protocol. She sends us r1. We pass along r1 to Bob who sends us r2, and x1 that he calculated with MyF. We are now done with Bob so we simply stop communications with him, and he thinks that the authentication failed. We pass along x1 and r2 to Alice who computes y1 using r1 and r2. x1 and y1 equal so Alice calculates a y2 using r2 and sends it back to us. We simply continue communications and the protocol is complete.

ii) With this modification, we see that the bitwise & operation can lead to the first parameter in the addition to equal 0 if we pick a correct r1, meaning that the third parameter used in MyF would just be r2. We will use this knowledge in our attack. We first monitor the protocol between Alice and Bob, noting down the value of r1. If during the protocol, one of them pick an r1 such that 18302628885633695807&r1 = 0, we continue monitoring their protocol and noting down the values of r2, y1, x1, y2, and x2. The next time Alice initiates the protocol with a value of r1 such 18302628885633695807&r1 = 0, we can impersonate Bob. We simply pick the same r2 and x2 value that we noted before and send it back to Alice. When Alice calculates her y1 she will be calculating y1=MyF(K, “Bob”, r2) since her r1 value is zeroed out in the addition because of the bitwise & operation. This y1 will equal the x1 that we sent and she will reply with some y2, to which we reply and the protocol is complete.

iii) With this modification, we see that the remainder of the addition is really what is taken as the third parameter. We will use this knowledge in our attack. We monitor the protocol between Alice and Bob once, noting the values of r1 and r2 that they use, and what y1, x1, y2 and x2 they pass. We also note what r1+r2(mode 2^64) equals, let’s call this value Z. We now have enough information to impersonate Bob. The next time Alice initiates the protocol we intercept her r1 and pick an r2 such that r1+r2(mod 2^64) = Z from before. We then pass Alice this r2 and the x1 value we intercepted earlier. When Alice calculates her y1, she will be calculating y1 = MyF(K, “Bob”, Z) and since we already know this value from before since Z is the same, it will equal and she will calculate some y2 value that she passes to us. We simply continue the chat, pretending to be Bob and the protocol is complete.

iv) We wait for Alice to initiate with Bob and intercept the r1 that she sends to us. We notes this r1 down as it will be the same every time Alice initiates. At some other time, we initiate the protocol with Bob and send him this r1. He then generates a random r2 and computes x1 = MyF(K, “Bob”, r1||r2), and sends us x1, and r2. We can then abort the chat session or send Bob some fake y2 that he will see does not match his calculated x2 and he will end the chat, as we are done with Bob. We then wait for Alice to initiate another chat, where she sends us the same r1 value. We then send her back the x1 and r2 that we got from Bob. Since r1 will be the same, when Alice computes y1 = MyF(K, “Bob”, r1||r2), it will match the x1 we sent her, so she will reply with y2 = MyF(K, “Alice”, r2). We simply, continue the chat session with Alice, she will think we calculated x2 and checked that it matched her y2, and the protocol has been complete.

1) b)

The bank authenticates Alice using two factors, knowledge (something she knows), and possession (something she has). The knowledge factor is the pin that she has memorized, and the possession factor is her credit card that she must insert into the ATM. Without both, the bank cannot authenticate Alice. Alice can identify the bank by checking the logo on the ATM and seeing if it is that of the banks (something the bank is), or by using an ATM that she has used before that she knows is owned by the bank (the bank's context). The bank identifies Alice by her credit card (something she has) and ensuring that the pin that she enters is the same as the pin they have on file that is matched to her credit card (something she knows).

2) a)

For Alice to be falsely rejected, she would need to be rejected 4 times in one window. Denote X as a swipe where Alice is rejected and O as an accepted swipe. Then any sequence of swipes with 4 X's would count as a false rejection (ie. XXXOX, XXXX, OOXXOXOX, etc.). Note that since once the phone hits 4 rejections it locks, we stop swiping at 4 rejections (not all windows will have 8 swipes since the phone locks before they can be completed). This models a negative binomial distribution where we denote r as the number of failures (Alice rejected), P as the probability of a failure, and x as the total number of swipes in that 8 swipe window. Since the phone can lock before 8 swipes but needs a minimum of 4 swipes to lock, we sum from 4 to 8 as follows:

$$FRR = \sum_{x=4}^{8} \binom{x-1}{r-1} P^r (1-P)^{x-r} = \sum_{x=4}^{8} \binom{x-1}{3} (0.12)^3 (0.88)^{x-3} = 0.0097 = 0.97\%$$

For a stranger to be falsely accepted, they would need to have less than 4 rejected swipes in the 8 swipe window. We can see this as all permutations of XXXOOOOO with less than 4 X's. We can represent this as a binomial distribution where x = the number of rejected swipes, P is the probability of a rejected swipe. We sum from 0 to 3 since all permutations with less than 4 rejections will count as a false acceptance:

$$FAR = \sum_{x=0}^{3} \binom{8}{x} P^x (1-P)^{x-1} = \sum_{x=0}^{3} \binom{8}{x} (0.93)^x (0.07)^{8-x} = 0.000079 = 0.0079\%$$

b)

Denote: L = Locked in 8 swipes, S = Stranger using phone, A = Alice using phone

Then the probability we are looking for is P(S|L). Using Bayes' Rule we get:

$$P(S|L) = \frac{P(L|S)P(S)}{P(L)} = \frac{P(L|S)P(S)}{P(L|A)P(A) + P(L|S)P(S)}$$

P(L|A) is just the FRR from part a. P(L|S) is just 1-FAR = 0.999921. Subbing into the equation we get:

$$P(S|L) = \frac{(0.999921)(0.08)}{(0.0097)(0.92) + (0.999921)(0.08)} = 0.8996 = 90.0\%$$

3)

a) Secret key encryption could be preferred simply because it is computationally more efficient. It is computationally expensive to generate public and private key pairs for encryption and decryption. The difficulty of decryption relies on it being difficult to generate a private key from a public one. The algorithm for coming up with a secret key is much faster.

b) Keeping methods of defeating the attack were kept secret so as to not tip off the attacker. If the attacker does not know that their malware has been defeated, they will continually use it to try and attack, not knowing that the victim can just recover from each attack since it has been defeated. If the defeats to the attacks were publicized, the attacker may have found out, updated their malware, and used a more complex attack.

c) When the author says it cycles the same block 14 times, he is saying that the AES algorithm is using the key to encrypt the same block over and over. So the block is encrypted using the steps outlined in the AES algorithm, then that encrypted block is encrypted again using the AES algorithm. This is repeated 14 times.

d) The attackers may have released the public key for several reasons. They may have switched their time and effort to newer ransomware, with the fear that TeslaCrypt was about to be defeated anyway. They could also have already made so much money that they want to retire from building ransomware before getting caught. Maybe even, they were hacked by a rival ransomware team who released the master key in order to sabotage their rivals business.

Works Cited

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

https://en.wikipedia.org/wiki/Public-key_cryptography

https://nakedsecurity.sophos.com/2016/05/19/teslacrypt-ransomware-gang-shuts-up-shop-reveals-master-key/

Programming

Sploit1)

By inspecting the html for the page, I was able to find that vote.php takes parameters directly from the URL. One of these parameters is the userid. By changing the userid from alice's to another one, in this case userid 5 for c5yao, I am able to up vote or down vote any post as c5yao. In the exploit I down vote the post with id 3, "For the good of the country!".

This violates the principle of failsafe defaults. We are given direct access to an object, the userid that is passed as a parameter. This is a direct object reference. We should be denied access to this object as omitting it from the URL results in less permissions.

Sploit2)

By poking around the files on the server, I was able to find the db_functions.php~ file. This file contains the sql query for logging in. I noticed it does not sanitize the input, making it possible to perform sql injection. By inserting bob' AND '1'='1' – as my username, I am able to return the information for bob without requiring his password, allowing me to log in as him. Bob has permissions to post articles, so I post one acting as him.

This violates the principle of complete mediation, as the input from the user is trusted as though the user is not harmful. Every access to every object, in this case the username provided, must be checked.

Sploit3)

By poking around the files on the server, I was able to find the directory /files123 which contained a file containing old passwords. By just trying these username and password combinations, I was able to login as nasghar with password weERKDekj4, as he did not change it. He has permissions to post links, so I posted a link as him linking to Wikipedia's page on hackers.

This violates the principle of open design. By storing the old passwords on the server, in the files123 folder, the webmaster believes that no one will find this file. If an attacker, like us, pokes around a bit however they will stumble upon this file and can use these old passwords to try to crack the new ones, or in our case just use one of the old ones that is still current.

Sploit4)

Again, by looking in the db_functions.php~ file, I was able to find the function used by confirm.php which takes a parameter 'hash'. I noticed that the sql query in this function looks for the 'confirm' variable in the database for each user, and sees if it matches the hash. If a user is confirmed, their confirm variable is set to empty or ''. If you pass in an empty hash, the query matches with the first confirmed user and logs in as them, in this case dstinson. dstinson has permissions to do all actions, so I upload an image as him.

This violates the principle of failsafe defaults. Again we are given direct access to an object, this time the hash that is passed as a parameter. This is a direct object reference. We should be denied access to this object as omitting it from the URL results in less permissions.

Sploit5)

I noticed by inspecting the forms that the post command does not filter out html or javascript. This makes it possible to perform cross site scripting and html injection. For this sploit, I performed html injection to make it look as though lagarwal had posted a comment. To do this I first post a real comment that will be posted by alice, and then inserted the necessary <hr> tags, links, and formatting to make it look like a comment was posted after alice's. The comment, along with the html, is inserted directly into the page, in this case the "For the good of the country!" article.

This violates the principle of complete mediation, again as the input from the user is trusted and not sanitized. The user can pass along data into the program that can act maliciously. Every access to every object, in this case the comment being posted, must be checked.