The Debian OpenSSL fiasco

Pedro Miguel Nicolau Escaleira
escaleira@ua.pt

01/12/2020

Contents

1 Introduction

The aim of the study stated on this report was to deepen our knowledge on the usage of asymmetric cryptography, more precisely, the handling of the RSA algorithm. To accomplish that, our work was to understand the practical implications of the Debian OpenSSL Fiasco and how an attacker could take advantage of this vulnerability [1].

In order to reach this goal, the Applied Cryptography teacher provided us a list with users and the respective public data needed to encrypt a message using the RSA algorithm. He also supplied a set of intercepted messages between the referred users, being our task to decode as many messages as possible.

All the code created for the sake of this project was done using the programming language Go [2] and can be acceced on the git repository [3].

2 Method

On this section we will describe the method we used to obtain the users private keys from the respective public keys made available by the subject's teacher. We will also present how we intercepted the secret contents of the users messages, as much as the implemented strategy to cypher information on behalf of some previous exposed user, using its private key information. At last, we will also expose the challenge we sent to the teacher while impersonating someone we were able to compromise.

2.1 Finding the private data

Since the foundation for this problem was the reuse of the same factors to create the RSA keys, if two distinct keys where obtained using at least one of the same factors between them, we can obtain those factors by calculating the greatest common divisor between the two public modulus of the keys. That is, for example, if we have the Alice's public modulus represented by m_{alice} and the Bob's public modulus represented by m_{bob}:

$$m_{alice} = p_1 \cdot q_1 \tag{1}$$

$$m_{bob} = p_2 \cdot q_2 \tag{2}$$

Then:

$$p_1 = p_2 \implies \begin{cases} m_{alice} = p_1 \cdot q_1 \\ m_{bob} = p_1 \cdot q_2 \end{cases} \implies gcd(m_{alice}) = gcd(m_{bob}) = p_1 \tag{3}$$

Then, if we have one of the factors of the public modulus, it is straightforward how we can obtain the other one:

$$q_1 = \frac{m_{alice}}{p_1} \tag{4}$$

$$q_2 = \frac{m_{bob}}{p_1} \tag{5}$$

Therefore, for us to find the private data needed to encrypt a user's message, we just needed to find the public modulus that had a common factor. In order to achieve that, we just had to combine all the modulus with each other, to verify which combinations had a greatest common divisor different than one, that implying that those modulus had a common factor, and that we could obtain the respective private information.

With this method, we successfully obtained $12,553$ private factors from different users i.e., 12.553 q and p pairs that constitute the respective user private data necessary to encrypt a message in his behalf. With the code provided on our repository, the time needed to achieve this results using our computer (using 6 threads[1]) was approximately 5900 seconds.

[1]The CPU model is *AMD Ryzen™ 7 PRO 3700U*.

2.2 Encode a message impersonating someone

Considering that we have in our possession the public modulus and encryption exponent, that were provided by the teacher, and now, from the previous subsection, the correspondent private factors of $12,553$ users, we can obtain the corresponding decryption exponent. Therefore, if we want the Alice's decryption exponent, D_{alice}, considering p_{alice} and q_{alice} her private factors and E_{alice} her encryption exponent, we have:

$$D_{alice} = E_{alice}^{-1} \mod lcm(p_{alice} - 1, q_{alice} - 1) \tag{6}$$

With all this information, we can now encrypt a message on the name of Alice:

$$M' = M^{D_{alice}} \mod m_{alice} \tag{7}$$

However, if we want to restrict the user that can access the message content i.e., authenticate him, this is not sufficient. Therefore, if we want only Bob to have access to the message, we also need to encrypt the resultant encrypted message M' with the Bob's public data:

$$C = M'^{E_{bob}} \mod m_{bob} \tag{8}$$

With this in mind, we now have the possibility to send a message to everyone of the list on behalf of $12,553$ users of that list.

2.3 Decode the intercepted messages

On the previous subsection we presented how to encrypt a message. The decryption is the reverse process. Therefore, if we continue with the notion that the final encrypted message was sent to Bob, he can obtain the Alice's cyphered message M' with his private information:

$$D_{bob} = E_{bob}^{-1} \mod lcm(p_{bob} - 1, q_{bob} - 1) \tag{9}$$

$$M' = C^{D_{bob}} \mod m_{bob} \tag{10}$$

Now, he can decipher this message with the Alice's public information:

$$M = M'^{E_{alice}} \mod m_{alice} \tag{11}$$

It was with these notions that we manage to decipher 12 of the 20 cyphered messages that were made available by the teacher.

2.3.1 Results

You can appreciate the successfully deciphered messages on the table 1 .

Sender	**Receiver**	**Deciphered Message**
Shirley Torres	Samantha Jenkins	-
Wendy Bass	Jeremy Montes	-
Herbert Burns	Daniel Howell	The real test is not whether you avoid this failure, because you won't. It's whether you let it harden or shame you into inaction, or whether you learn from it; whether you choose to persevere.
Kenneth Bell	Tommy Armstrong	It is better to keep your mouth closed and let people think you are a fool than to open it and remove all doubt.
Nancy Lyons	Mark Lucas	-
Emma Christensen	Brian Krueger	-
Lloyd Lopez	Justin Smith	If you set your goals ridiculously high and it's a failure, you will fail above everyone else's success.
Antonio Cole	Charlie Anderson	-
Minnie Cameron	Larry Lambert	Age is an issue of mind over matter. If you don't mind, it doesn't matter.
Fabian Portillo	Elmer Henderson	The only impossible journey is the one you never begin.
Marion Zhang	Willie Dillon	Many of life's failures are people who did not realize how close they were to success when they gave up.
Daniel Tyler	Cindy Green	-
Barry Quinn	Travis Hernandez	Don't judge each day by the harvest you reap but by the seeds that you plant.
Michael Whitehead	Aaron Lee	I find that the harder I work, the more luck I seem to have.
Sheila Rodriguez	Marion Kaur	-
Betty Mitchell	Carol Villarreal	-
Ruben Turner	Gerda Wright	Never put off till tomorrow what you can do the day after tomorrow.
Mark Schultz	Michael Bowman	Go to Heaven for the climate, Hell for the company.
Barbara Ramirez	Tammy Hunter	Tell me and I forget. Teach me and I remember. Involve me and I learn.
Erick Green	Joni Eaton	The greatest glory in living lies not in never falling, but in rising every time we fall.

Table 1: Deciphered messages.

2.4 Message sent to the teacher

As requested by the teacher (with the pseudonym "Charlie Brown"), we sent him an encrypted message, in our case, impersonating the user "Brian York". The encrypted message can be checked out below:

```
102299805558262027092109640132740146026462657681433952535985363784070808607201451270451670
483364245962595224397812325468949444143951325219931778503689419813613053415839300576064074
179939680394628626993607163349439505396734408604212711171176144594600723287466163500851127
985964206527497729969966288309525863017620332449404316897742362329460263147688813216405552
799727625173362254792405980297422139602714158523855795492262492737714179360836680199403734
425145993596391297716158276179378960746810270036428987965790169090537225705146538657131230
090316796716776558829081870658018795017099326278924895545968024286906579151080274267422703
503396909080281786951860809098337153430454317477800851727915064651765415847167466130336417
835332438570850002300504357768555833554606427489290169298365044640851878812029111332649515
142365509704656300990441583741980445522308856488949579699799834988521878056479522402302995
734213174666027950720057592061771293227087564205732629584474512354248383121543295508554436
770389317149746480403744742532936397224233361052812664472753818473319212570984569378117861
073732609913636353342940540724631199342559751112155535036663471008710357504669266207964029
03083495235016228398888695966279002377381555711137027295766631 8
```

Although the teacher only requested us to send a simple message, probably with some text message, like the ones he had provided, we decided to create something that could be more enjoyable to him. Therefore, the original message contains a simple challenge.

How to crack the challenge (with the purpose of not immediately giving the answer, the text below is in white and you can read if you select it with your mouse):

[2] For example, the `https://cryptii.com/pipes/caesar-cipher`

3 Conclusion

Having finished the work, we think that now we have a better understanding of the underlying mechanisms of the RSA algorithm, explained on the subject's theoretical component. It was also an important work to complement our education on the Cybersecurity field, being this a vulnerability with some major consequences and that teaches us that a robust cryptographic algorithm implementation is as much important as its theoretical strength.

4 References

[1] Russ Cox. Lessons from the debian/openssl fiasco. Available: `https://research.swtch.com/openssl` [Accessed 01-12-2020].

[2] The go programming language. Available: `https://golang.org/` [Accessed 01-12-2020].

[3] Pedro Escaleira. Debian openssl fiasco. Available: `https://github.com/oEscal/debian_openssl_fiasco` [Accessed 01-12-2020].