Information Technology Journal 3 (3): 327-331, 2004
ISSN 1682-6027

Password Interception in a SSL/TLS Channel

[1]Khawaja Amer Hayat, [1]Umar Waqar Anis and [2]Tauseef ur Rahman
[1]Department of Computer Science, International Islamic University Islamabad, Pakistan
[2]Department of Telecom and Computer Engineering, International Islamic University Islamabad, Pakistan

Abstract: This study represents the attack and demonstrates the optimized form of it. It works against the latest and most popular implementations of SSL/TLS for password interception. This study showed that a password for an IMAP account can be intercepted when the eavesdropper is near the server in an hour of processing. It can be concluded that the versions of SSL/TLS are insecure when used with block ciphers in CBC mode. Before conclusion the study shows the conditions for the attack. In the end this study propose ways to make these versions strong and much more secure.

Key words: IMAP account, RFC 2246, MAC

INTRODUCTION

Cipher-block chaining mode: CBC mode processes the data based on some initialization vector IV which is a l-bit string. The input to the encryption algorithm is the XOR of the current plaintext block and the preceding cipher text block.

Secure Socket Layer (SSL) and Transport Layer Security (TLS): SSL was originated by Netscape. TLS working group was formed within IETF. First version of TLS can be viewed as an SSLv3.1. Figure 1 shows the record format of SSL.

Figure 2 shows the operation of the SSL Record Protocol.

TLS has the same record format as the SSL. It is defined in RFC 2246 and is similar to SSLv3. the differences between SSL and TLS are:

1. Version number
2. Message authentication code
3. Pseudorandom function
4. Alert codes
5. Cipher suites
6. Client certificate types
7. Certificate verify and finished message
8. Cryptographic computations
9. Padding

CBC-PAD in secured channels: Peer-to-peer secure and safe channels can be established by the Transport Layer Security Protocol. It requires:

- Negotiating a cipher suite and security parameters between the two engaged parties.
- Exchanging secret keys between the parties engaged in communication.
- Then messages M are first authenticated with a Message Authentication Code (MAC).

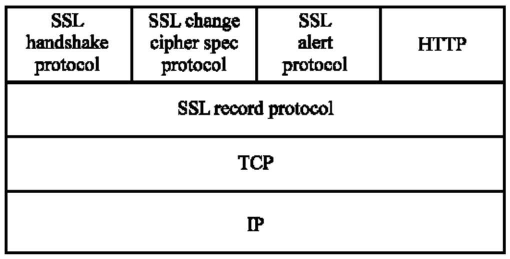

Fig. 1: SSl protocol stack

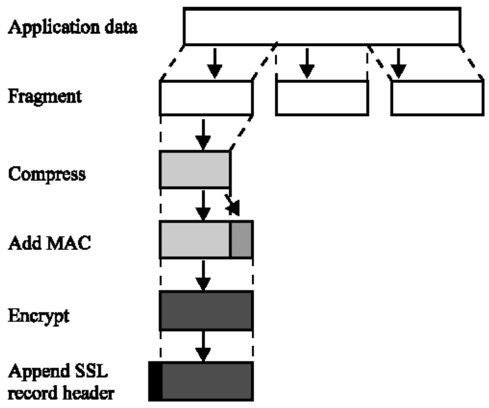

Fig. 2: SSL record protocol

Corresponding Author: Khawaja Amer Hayat, Department of Computer Science, International Islamic University Islamabad, Pakistan E-mail: amerhayat2@hotmail.com

- Then encrypted with a symmetric cipher. Block ciphers, e.g. the Triple DES (3DES) are often used in Cipher Block Chaining (CBC) mode with padding.

Let b be the block length in characters let b = 8 for DES as it requires 64-bit M. Let M be the message to be sent. First we append the MAC of M to M. We obtain MjMAC. Then we pad MjMAC with a padding PAD such that MjMACjPADjLEN is of length a multiple of b where LEN is a single byte whose value ` is the length of PAD in bytes. PAD is required by the TLS specifications to consist of l bytes equal to l. Then MjMACjPADjLEN is cut into a block sequence x 1, x 2..., x n (each x i has a length of b), then encrypted in CBC mode, i.e. changed into y 1, y 2,..., y n with

$$M_i = ENC(y_{i-1} \text{ xor } x_i)$$

where, ENC denotes the block cipher. The initial vector IV y_0 can be either a part of the secret key, or a random value sent with the cipher text, or a xored value. When y 1, y 2,..., y n is received, it is first decrypted back into x 1, x 2,..., xn. Then we look at the last byte LEN, call l its value and separate the padding PAD of length l and LEN from the plaintext. It is required that PAD should be checked to consist of bytes all equal to l. If this is not the case, a padding error is generated. Otherwise the MAC is extracted then checked. If the MAC is not valid, a MAC error is generated. Otherwise the clear text M is extracted and processed. In TLS, fatal errors such as incorrect padding or bad MAC errors simply abort the session. The error messages are sent through the same channel, i.e. they are MACed, padded, then encrypted before being sent. A typical application of TLS is when an email application connects to a remote IMAP server. For this, the client simply sends the user name and password through the secured channel, i.e. the message M includes the password in clear.

Side channel attack against CBC-PAD: It assumes that we can send a cipher text to the server and get the answer which is either an error or an acknowledgment. We model it as an prediction P. When the answer is a padding error message (decryption failed), we say that P answers 0. Otherwise the prediction P returns 1. Let y be the cipher text block to decrypt. The purpose of the attack is to find the block x such that y = ENC(x). We first transform the prediction P into an prediction Check1(y; u) which checks whether the ENC 1 (y) block ends with the byte sequence u or not. We then use this prediction in DecryptByte1(y; s) in order to decrypt a new character in ENC 1 (y) from the known tail s of x. We then use this process in DecryptBlock1(y) in order to decrypt a full block y. The attack of Vaudenay[1] works against WTLS[2]. It does not work against TLS for two reasons. First of all, as soon as a padding or MAC error occurs the session is broken and needs to restart with a freshly exchanged key. The attack could have still worked in order to decrypt only the rightmost byte with a probability of success of 2^{-8}. It can also be adapted in order to test if x ends with a given pattern[1]. This does not work either against TLS for another reason i.e. error messages are not available to the antagonist (they are indeed encrypted and indistinguishable). In order to make them even less distinguishable, standard implementations of the TLS protocol now use the same error message for both types or errors (as specified for SSL) in order to protect against this type of attack[3].

```
DecryptBlock1(y)
1: for i = 1 to b do
2:     c_i = DecryptByte1(y; c_i-1 |...| c_1)
3: end for
4: return c_b |...| c_1
DecryptByte1(y, s)
1: for all possible values of byte c do
2:     if Check1(y, c | s) = 1 then
3:         return c
4:     end if
5: end for
Check1(y, u)
1: let i be the length of u
2: let L be a random string of length b i
3: let R = (i -1)| (i -1)|...|(i -1) of length i
4: r = L|(R⊕u)
5: build the fake cipher text r | y to be sent to the prediction
6: return P(r | y)
```

Fig. 3: Side channel attack against CBC-PAD

Timing attack

Attack principles: In order to get access to the error type which is not directly available, we try to deduce it from a side channel by performing a timing attack[4]. Instead of getting 0 or 1 depending on the error type, we now have prediction which outputs the timing answer T of the server. The principle of the attack is as follows: in order to check if the padding is correct, the server only needs to perform simple operations on the very end of the cipher text. When the padding is correct, the server further needs to perform cryptographic operations throughout the whole cipher text in order to check the MAC and this may take more time. We use the variation between the time it takes to perform the two types of operations in order to get the answer from the prediction P.

We increase the discrepancy of the two types of errors by enlarging the cipher text: the longer the cipher text, the longer the MAC verification. (The MAC verification time increases linearly with the length of

```
DecryptByte2(y| s)
1: repeat
2: for all possible values of byte c do
3: if Check2(y, c | s) = 1 then
4: return c
5: end if
6: end for
7: until byte is found
Check2(y | u)
1: make r in order to test u as in Check1
2: build the fake cipher text f |r| y to be sent to the prediction
   (f is the longest possible random block sequence)
3: query the prediction n times and get T1,..., T n
   (answers which are larger than B are ignored)
4: return ACCEPT(T1,..., T n)
```

Fig. 4: Regular timing attack

MES.) Hence we replace the r | y fake cipher text in DecryptByte1 by f |r| y where f is a random block sequence of the longest acceptable length (i.e. 214+2048 bytes in TLS).

On Fig. 4 is the updated algorithm. It uses a DecryptBlock2 algorithm which is similar to Fig. 3. Note that Check2 may miss the right byte, so DecryptByte2 needs to repeat the loop until the byte is found.

Experiment: We made a statistical analysis of the answer time for the two types of errors. The expected values μ_R and μ_w and the standard deviations μ_R and μ_w for the two distributions are as follows:

$$\mu_R \sim 23{:}63 \quad \mu_w \sim 21{:}57$$
$$\sigma_R \sim 1{:}48 \quad ó_w \sim 1{:}86$$

Note that these values were obtained on a LAN where, a fire wall was present between the attacker and the server, so the attacker was not directly connected to the server.

Analysis of the best ACCEPT predicate: The ACCEPT predicate is used in order to decide whether the distribution of the answers is DR (the predicate should be true) or DW (the predicate should be false). The predicate introduces two types of wrong information. We let ∊+ (resp. ∊-) be the probability of bad decision when the distribution is D_W (resp. D_R). The ∊ + and ∊- probabilities can be interpreted as the probabilities of false positives and false negatives of a character correctness test. The optimal tradeoff between ∊ + and ∊- is achieved by the ACCEPT predicate which is given by the Neyman-Pearson lemma:

$$\text{ACCEPT}: \frac{f_R(T_1)}{f_W(T_1)} \times \ldots \times \frac{f_R(T_n)}{f_W(T_n)} > T$$

with f_R and f_W the density functions of DR and DW, respectively and a given threshold ã. Depending on ã we trade ∊ + against ∊. With the approximation by a normal distribution the ACCEPT test can be written

$$\frac{T_1 + \ldots + T_n}{n} > \frac{\mu R + \mu W}{2} + \frac{\sigma^2 \log \tau}{n(\mu R + \mu W)}$$

so we equivalently can change the definition of τ and consider

$$\text{ACCEPT}: \frac{T_1 + \ldots + T_n}{n} > \tau'$$

as the ACCEPT test choice. With this we obtain

$$\epsilon_- = \varphi\left(\frac{\tau' - \mu R}{\sigma}\sqrt{n}\right)$$

$$\epsilon_+ = \varphi\left(-\frac{\tau' - \mu W}{\sigma}\sqrt{n}\right)$$

where:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \epsilon = \frac{t^2}{2} dt.$$

Using sequential decision rules: The following algorithm shows a more general skeleton. Basically, we collect timing samples T j until some STOP predicate decides that there are enough of these for the ACCEPT predicate to decide. We use DecryptByte3 and DecryptBlock3 algorithms which are similar to DecryptByte2 and ecryptBlock2 (Fig. 5).

Multi-session attack

Attack strategy: Since sessions are broken as soon as there is an error, the attacks from previous sections do not work. We now assume that each TLS session includes a critical plaintext block x which is always the same (e.g. a password) and that we intercept the corresponding cipher text block y = ENC(x y ‘). (Here y 0 is the previous cipher text block following the CBC mode.) The target x is constant in every session, but y and y 0 depend on the session. The full attack is depicted in the algorithm discussed in analysis. Here the Check4 prediction no longer relies on some y since this block is changed in every session. The Check4(u) is called in order to check whether \the x plaintext block ends with the byte sequence u" or not. The plaintext block x is equal to ENC^{-1} (y) y ’ for some current key and some current

```
Check3(y, u)
1:  make r in order to test u as in Check1
2:  build the fake cipher text f |r| y to be sent to the prediction as in
    Ckeck2
3:  j = 0
4:  repeat
5:  j ++
6:  query the prediction and get T j
    (a T j larger than B is ignored and the query is repeated)
7:  until STOP(T1,..., T j)
8:  return ACCEPT(T1,..., T j)
```

Fig. 5: Timing attack with a sequential distinguisher

```
DecryptBlock4
1:  for i = 1 to b do
2:  c i DecryptByte4(c i -1 |...| c1)
3:  end for
4:  return c b |…| c1
DecryptByte4(s)
1:  sort all possible c characters in order of decreasing likelihood.
2:  repeat
3:  for all possible values of character c do
4:  if Check4(c| s) = 1 then
5:  return c
6:  end if
7:  end for
8:  until byte is found
Check4(u)
1:  j= 0
2   repeat
3:  j ++
4:  wait for a new session and get the current y and y 0 blocks
5:  let i be the length of u
6:  let L be a random string of length b i
7:  let R = (i -1)j(i -1)|…| (i- 1) of length i
8:  r = (L| (R⊕u))⊕y'
9:  build the fake cipher text f |r| y to be sent to the prediction
    (f is the longest possible random block sequence)
    10: query the prediction and get T j
    (if it is larger than B then go back to Step 4)
11: until STOP(T1,…,T j)
12: return ACCEPT(T1,..., T j)
```

Fig. 6: Password interception inside SSL/TLS

cipher text blocks y and y'. We assume that the prediction can get y and y '.

Analysis: Let C be the average complexity of DecryptBlock4. Let Z denote the set of all possible byte values. Let p be the probability of success of DecryptBlock4.

Let p i be the success probability of DecryptByte4(s) assuming that s is the right tail of length I-1. We have p = p 1…p b.

In order to simplify our analysis, we assume that the target block is uniformly distributed in Z b so that step 1 of DecryptByte4 can be ignored (Fig. 6). We further consider a weaker algorithm in which the outer repeat/until loop of DecryptByte4 is removed (i.e. we consider that the attack fails as soon as a STOP predicate is satisfied but the ACCEPT predicate takes a bad decision).

Table 1: Calculated complexity C and threshold values log τ- and log τ+ for algorithm DecryptBlock4 given success probability p with μR = 23.63, μW = 21.57, σ = 1.86 and heuristic value B = 32.93 for dictionary and distributions in Z^b

Dictionary, C_0 = 31						
P	0.50	0.60	0.70	0.80	0.90	0.99
C	166.00	181.00	199.00	223.00	261.00	380.00
log τ-	-2.74	-3.05	-3.41	-3.88	-4.62	-6.98
log τ+	4.86	5.16	5.99	5.99	6.74	9.09
Uniform distibution, |Z| = 256, C_0 = 1028						
P	0.50	0.60	0.70	0.80	0.90	0.99
C	4239.00	4750.00	5353.00	6139.00	7397.00	11335.00
log τ-	-2.45	-2.76	-3.12	-3.59	-4.43	-6.69
log τ+	12.15	12.46	12.81	13.28	14.03	16.38
Uniform distibution, |Z| = 128, C_0 = 516						
P	0.50	0.60	0.70	0.80	0.90	0.99
C	2179.00	2346.00	2738.00	3132.00	3764.00	5741.00
log τ-	-2.46	-2.77	-3.12	-3.60	-4.35	-6.70
log τ+	10.76	11.07	11.43	11.90	12.65	15.00
Uniform distibution, |Z| = 64, C_0 = 260						
P	0.50	0.60	0.70	0.80	0.90	0.99
C	1140.00	1269.00	1421.00	1620.00	1938.00	2934.00
log τ-	-2.48	-2.78	-3.14	-3.61	-4.36	-6.71
log τ+	9.38	9.68	10.04	10.51	11.26	13.61

Password interception with dictionary attack

Attack description: We now use the a priori distribution of x in the previous attack in order to decrease the complexity. For instance, if x is a password corresponding to an IMAP authentication, we perform a kind of dictionary attack on x. We assume that we have pre computed a dictionary of all possible x blocks with the corresponding probability of occurrence. We use it in the first step of DecryptByte4 in order to sort the c candidates.

Analysis: We consider a list of possible blocks c $_b$… c $_1$. We let Pr[c $_b$… c $_1$] be the occurrence probability of a plaintext block. We also let Pr[c $_i$… c $_1$] be the sum Pr[c $_b$… c_1] for all possible c $_b$… c $_{i+1}$. We arrange the dictionary of all blocks into a search tree. The root is connected to many sub trees, each corresponding to a c_1 character. Each sub tree corresponding to a c_1 character is connected to many sub-sub trees, each corresponding to a c_2 character... We label each node of the tree by a c i… c 1 string. We assume that the list of sub trees of any node c i… c 1 is sorted in decreasing order of values of Pr[c I+1 c_i… c $_1$]. We let N(c $_{i+1}$… c $_1$) be the rank of the c $_{i+1}$… c $_1$ sub tree of the node c i… c $_1$ in the list.

Numerical example: We have used dictionary[5] from which we have selected only words of size b = 8 characters (i.e. 8 bytes), giving a total word count of 712-786 words and ordered it as described the previous section. For this dictionary, we have calculated that C_0 = 31 and then implemented algorithm DecryptBlock4 and confirmed this result. Note that C_0 = 31 is a quite remarkable result since the best search rule for finding a password out of a dictionary of D = 712 0 786 words consists of $|\log_2 D| = 20$ binary questions, so the overhead is only of 11 questions.

Implementation of the attack : Here we describe how the DecryptBlock4 was implemented in practice against an IMAP email server.

Setup: The multi-session attack has been implemented using the Outlook Express 6.x client from Microsoft under Windows XP and an IMAP Rev 4 server 8. Outlook sends the login and password to the IMAP server using the following format:

XXXX LOGIN "username"
"password"<0x0d><0x0a>

Here XXXX are four random digits which are incremented each time Outlook connects to the server. An interesting feature of Outlook is that (by default) it checks for messages automatically every 5 min and also that it requires an authentication for each folder created on the IMAP user account, i.e. we have a bunch of free sessions every 5 min. For instance, with five folders (in, out, trash, read and draft), we obtain 60 sessions every hour. If Outlook is now configured to check emails every min, the fastest attack of Table 1 with 166 sessions requires half an hour. Outlook notices that some protocol errors occur but this does not seem to bother it at all. The TLS tunneling between the IMAP server and Outlook Express was implemented using stunnel v3.22 9.

This is a man-in-the-middle type attack where connection requests to the IMAP server from the Outlook client are redirected to the attacker's machine using DNS spoofing where the attacker intercepts the authentication messages and attempts to decrypt it using DecryptBlock4. Note that the attack is performed on a Local Area Network.

Conditions for the attack: Obviously, the attack works if the following conditions are met.

- A critical piece of information is repeatedly encrypted at a predictable place.
- A block cipher in CBC mode is chosen.
- The attacker can sit in the middle and perform active attacks.
- The attacker can distinguish time differences between two types of errors.

CONCLUSION

We have derived a multi-session variant of the attack[1] in order to show that it is possible to attack SSL/TLS in the case when the message that is being encrypted remains the same during each session. This is the case, for example, when an email client such as Outlook Express connects to an IMAP server. We have detailed the attack and described the setup we have used in order to perform it. One problem we have encountered is that the error messages sent in SSL/TLS are encrypted and it is not possible to easily differentiate which is being sent by the client or the server. A solution to this problem is to look at timings between errors messages.

REFERENCES

1. Vaudenay, 2002. Security flaws induced by CBC padding applications to SSL, IPSEC, WTLS. In Advances in Cryptology EUROCRYPT'02, Amsterdam, Netherland, Lectures Notes in Computer Science 2332: 534-545.
2. Wireless Transport Layer Security. Wireless application protocol 2001, WAP-261-WTLS-20010406-a. Wireless application protocol forum, http://www.wapforum.org/
3. FIPS 46-3, 1999. Data Encryption Standard (DES). U.S. Department of Commerce, National Institute of Standards and Technology. Federal Information Processing Standard Publication, pp: 46-3.
4. Kocher, P., 1996. Timing attacks on implementations of DiÆe-Hellman, RSA, DSS and other systems. In Advances in Cryptology CRYPTO'96, Santa Barbara, California, USA, Lectures Notes in Computer Science 1109, pp: 104-113.
5. English Word List Elcomsoft Co. Ltd. http://www.elcomsoft.com