Password Interception in a SSL/TLS Channel

Brice Canvel[1], Alain Hiltgen[2], Serge Vaudenay[1], and Martin Vuagnoux[3]

[1] Swiss Federal Institute of Technology (EPFL) - LASEC
http://lasecwww.epfl.ch
[2] UBS AG
email:alain.hiltgen@ubs.com
[3] EPFL - SSC, and Ilion
http://www.ilionsecurity.ch

Abstract. Simple password authentication is often used e.g. from an email software application to a remote IMAP server. This is frequently done in a protected peer-to-peer tunnel, e.g. by SSL/TLS.
At Eurocrypt'02, Vaudenay presented vulnerabilities in padding schemes used for block ciphers in CBC mode. He used a side channel, namely error information in the padding verification. This attack was not possible against SSL/TLS due to both unavailability of the side channel (errors are encrypted) and premature abortion of the session in case of errors. In this paper we extend the attack and optimize it. We show it is actually applicable against latest and most popular implementations of SSL/TLS (at the time this paper was written) for password interception.
We demonstrate that a password for an IMAP account can be intercepted when the attacker is not too far from the server in less than an hour in a typical setting.
We conclude that these versions of the SSL/TLS implementations are not secure when used with block ciphers in CBC mode and propose ways to strengthen them. We also propose to update the standard protocol.

1 Introduction

1.1 CBC-PAD in Secured Channels

Peer-to-peer secure channels can be established by the TLS protocol [9]. It consists in, first, negotiating a cipher suite and security parameters, second, exchanging secret keys. Then messages are first authenticated with a Message Authentication Code (MAC), then encrypted with a symmetric cipher. Block ciphers, e.g. the Triple DES (3DES) [3] are frequently used in Cipher Block Chaining (CBC) mode [4] with padding. Let b be the block length in characters (e.g. $b = 8$ for DES).

Let MES be the message to be sent. First we append the MAC of MES to MES. We obtain MES|MAC. Then we pad MES|MAC with a padding PAD such that MES|MAC|PAD|LEN is of length a multiple of b where LEN is a single byte whose value ℓ is the length of PAD in bytes. PAD is required by the TLS specifications to consist of ℓ bytes equal to ℓ. Then MES|MAC|PAD|LEN is cut into a block sequence $x_1, x_2, \ldots, x_n$ (each x_i has a length of b), then encrypted in CBC mode, i.e. transformed into $y_1, y_2, \ldots, y_n$ with

$$y_i = \mathrm{ENC}(y_{i-1} \oplus x_i)$$

where ENC denotes the block cipher. (We do not discuss about the initial vector y_0 which can be either a part of the secret key, or a random value sent with the ciphertext, or a fixed value.)

When $y_1, y_2, \ldots, y_n$ is received, it is first decrypted back into $x_1, x_2, \ldots, x_n$. 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 cleartext MES is extracted and processed.

In TLS, fatal errors such as incorrect padding or bad MAC errors simply abort the session. It should also be outlined that 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 [8]. For this, the application (client) simply sends the user name and password through the secured channel, i.e. the message MES includes the password in clear.

1.2 Side Channel Attack against CBC-PAD

In 2002, Vaudenay [17] presented an attack which enables the decryption of blocks provided that error messages are available (as a side channel attack) and sessions do not abort. We thus assume that we can send a ciphertext to the server and get the answer which is either an error or an acknowledgment. We modelize this as an oracle $\mathcal{O}$. When the answer is a padding error message (`decryption_failed`), we say that $\mathcal{O}$ answers 0. Otherwise (`bad_record_mac`), the oracle $\mathcal{O}$ answers 1.

Let y be the ciphertext block to decrypt. The purpose of the attack is to find the block x such that $y = \mathrm{ENC}(x)$. Following [17], and as depicted on Fig. 1, we first transform the oracle $\mathcal{O}$ into an oracle **Check1**(y, u) which checks whether "the $\mathrm{ENC}^{-1}(y)$ block ends with the byte sequence u" or not. We then use this oracle in **DecryptByte1**(y, s) in order to decrypt a new character in $\mathrm{ENC}^{-1}(y)$ 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 [17] 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. As pointed out in [17], 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. (In [17] the oracle in the corresponding

DecryptBlock1(y)
for $i = 1$ to b **do**
  $c_i \leftarrow$**DecryptByte1**$(y, c_{i-1}|\ldots|c_1)$
end for
return $c_b|\ldots|c_1$

DecryptByte1(y, s)
for all possible values of byte c **do**
  if Check1$(y, c|s) = 1$ **then**
    return c
  end if
end for

Check1(y, u)
let i be the length of u
let L be a random string of length $b - i$
let $R = (i-1)|(i-1)|\ldots|(i-1)$ of length i
$r \leftarrow L|(R \oplus u)$
build the fake ciphertext $r|y$ to be sent to the oracle
return $\mathcal{O}(r|y)$

Fig. 1. Side Channel Attack against CBC-PAD.

attacks is called a "bomb oracle".) This does not work either against TLS for another reason: because error messages are not available to the adversary (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 [13].

Other studies investigated attacks against paddings, like Black-Urtubia [6] which considered other modes than the CBC mode, and Paterson-Yau[4] which considered the CBC mode of ISO/IEC 10116 [1].

1.3 Structure of this Paper

In this paper we first explain in Section 2 how to distinguish between the two types of errors by using another side channel attack based on timing discrepancies. We perform experimental analysis and optimize the attack. Section 3 then explains how to push the attack forward in several broken TLS sessions. We analyze the attack. Section 4 optimizes the attack by performing dictionary attacks against password authentication. We finally describe an experimental attack (Section 5), discuss about practical consequences (Section 6), and conclude.

[4] Personal communication.

2 Timing Attack

2.1 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 [12]. Instead of getting 0 or 1 depending on the error type, we now have an oracle 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 ciphertext. When the padding is correct, the server further needs to perform cryptographic operations throughout the whole ciphertext in order to check the MAC, and this may take more time. We use the discrepancy between the time it takes to perform the two types of operations in order to get the answer from the oracle.

We increase the discrepancy of the two types of errors by enlarging the ciphertext: the longer the ciphertext, the longer the MAC verification. (The MAC verification time increases linearly with the length of MES.) Hence we replace the $r|y$ fake ciphertext in **DecryptByte1** by $f|r|y$ where f is a random block sequence of the longest acceptable length (i.e. $2^{14} + 2048$ bytes in TLS).[5]

Formally we let D_W (resp. D_R) be the distribution of the timing answer from the server when there is a padding error (resp. a MAC error). We let μ_W (resp. μ_R) be its expected value. We will use the following approximation.

Conjecture 1. We approximate D_R and D_W by normal distributions with the same standard deviation σ and expected values μ_R and μ_W respectively. We assume w.l.o.g. that $\mu_W < \mu_R$.

We further make the query to the oracle several times in order to make a statistical analysis. We use a predicate ACCEPT in order to decide whether or not the error type is a padding or a MAC.

Provided that the adversary is close to the server, the time measurement may be influenced by a little noise. An unexpectedly long answer may occur due to other protocol issues. These answers are ignored in the experiment. In practice we ignore times which are greater that a given threshold B.

On Fig. 2 is the updated algorithm. It uses a **DecryptBlock2** algorithm which is similar to **DecryptBlock1**. Note that **Check2** may miss the right byte, so **DecryptByte2** needs to repeat the loop until the byte is found.

2.2 Experiment

We made a statistical analysis of the answer time for the two types of errors. The distributions D_W and D_R can be seen on the graph of Fig. 3. The expected values

[5] This trick applies assuming that CBC decryption and MAC verification are not done at the same time.

```
DecryptByte2(y, s)
repeat
    for all possible values of byte c do
        if Check2(y, c|s) = 1 then
            return c
        end if
    end for
until byte is found
Check2(y, u)
make r in order to test u as in Check1
build the fake ciphertext f|r|y to be sent to the oracle
(f is the longest possible random block sequence)
query the oracle n times and get T_1, ..., T_n
(answers which are larger than B are ignored)
return ACCEPT(T_1, ..., T_n)
```

Fig. 2. Regular Timing Attack.

μ_R and μ_W and the standard deviations σ_R and σ_W for the two distributions are as follows:

$$\mu_R \approx 23.63 \qquad \mu_W \approx 21.57$$
$$\sigma_R \approx 1.48 \qquad \sigma_W \approx 1.86.$$

We take $\sigma = \sigma_W \approx 1.86$. From the graph, we can clearly see that the two distributions are distinguishable. The following section formalizes this distinguishability. Note that these values were obtained on a LAN where a firewall was present between the attacker and the server, so the attacker was not directly connected to the server.[6]

2.3 Analysis of the Best ACCEPT Predicate

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

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

[6] More precisely, the route between the attacker and the server included two switches and a firewall (a PC running Linux).

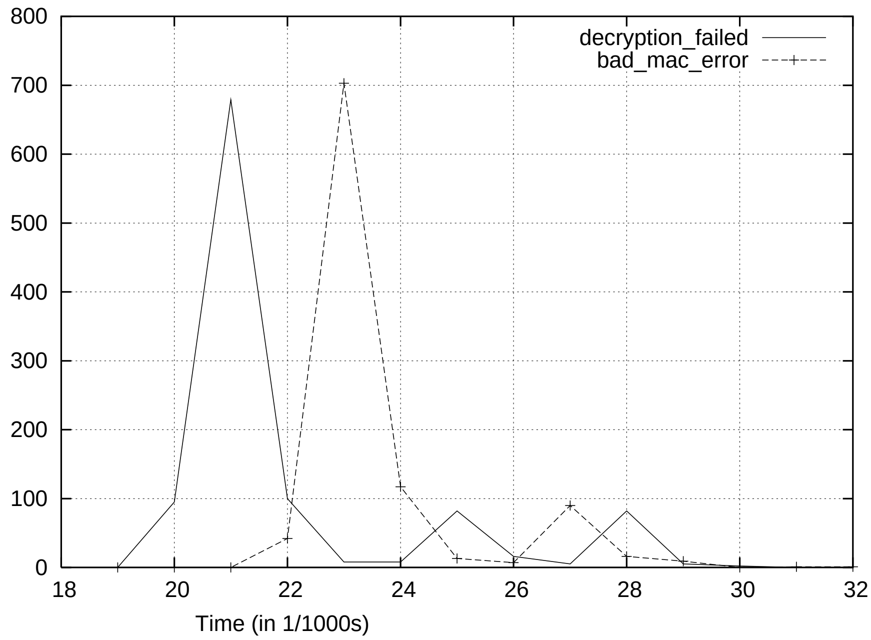

Fig. 3. Distribution of the number of `decryption_failed` and `bad_mac_error` error messages with respect to time.

with f_R and f_W the density functions of D_R and D_W respectively and a given threshold τ. Depending on τ we trade ε_+ against ε_-.

With the approximation by a normal distribution the ACCEPT test can be written

$$\prod_{i=1}^{n} \frac{e^{-\frac{(T_i-\mu_R)^2}{2\sigma^2}}}{e^{-\frac{(T_i-\mu_W)^2}{2\sigma^2}}} > \tau$$

which is equivalent to

$$\frac{T_1 + \ldots + T_n}{n} > \frac{\mu_R + \mu_W}{2} + \frac{\sigma^2 \log \tau}{n(\mu_R - \mu_W)}$$

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

$$\varepsilon_- = \varphi\left(\frac{\tau' - \mu_R}{\sigma}\sqrt{n}\right)$$

$$\varepsilon_+ = \varphi\left(-\frac{\tau' - \mu_W}{\sigma}\sqrt{n}\right)$$

Check3(y, u)
make r in order to test u as in **Check1**
build the fake ciphertext $f|r|y$ to be sent to the oracle as in **Ckeck2**
$j \leftarrow 0$
repeat
  $j \leftarrow j + 1$
  query the oracle and get T_j
  (a T_j larger than B is ignored and the query is repeated)
until $\mathrm{STOP}(T_1, \ldots, T_j)$
return $\mathrm{ACCEPT}(T_1, \ldots, T_j)$

Fig. 4. Timing Attack with a Sequential Distinguisher.

where

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

2.4 Using Sequential Decision Rules

On Fig. 4 is a more general algorithm 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 **DecryptBlock2**.

By using the theory of hypothesis testing with sequential distinguishers (see Junod [11]), we obtain that the most efficient algorithms are obtained with the following STOP and ACCEPT tests.

$$\begin{aligned} \mathrm{STOP}:\ & \frac{f_R(T_1)}{f_W(T_1)} \times \ldots \times \frac{f_R(T_j)}{f_W(T_j)} \not\in [\tau_-, \tau_+] \\ \mathrm{ACCEPT}:\ & \frac{f_R(T_1)}{f_W(T_1)} \times \ldots \times \frac{f_R(T_j)}{f_W(T_j)} > \tau_+ \end{aligned}$$

where τ_- and τ_+ are two given thresholds. Assuming Conjecture 1, this is equivalent to

$$\mathrm{STOP}:\ T_1 + \ldots + T_j - j\frac{\mu_R + \mu_W}{2} \not\in [\tau'_-, \tau'_+] \tag{1}$$

$$\mathrm{ACCEPT}:\ T_1 + \ldots + T_j - j\frac{\mu_R + \mu_W}{2} > \tau'_+ \tag{2}$$

where

$$\tau'_+ = \frac{\sigma^2}{\mu_R - \mu_W} \log \tau_+ \tag{3}$$

$$\tau'_- = \frac{\sigma^2}{\mu_R - \mu_W} \log \tau_-. \tag{4}$$

Thanks to the Wald Approximation, we can freely select ε_+ and ε_-, compute the corresponding τ_+ and τ_- by

$$\tau_+ \approx \frac{1-\varepsilon_-}{\varepsilon_+}$$
$$\tau_- \approx \frac{\varepsilon_-}{1-\varepsilon_+}$$

and deduce the expected number J of samples (i.e. the j iterations) until STOP holds and ACCEPT takes the right decision by

$$J_W \approx -\frac{2\sigma^2}{(\mu_R-\mu_W)^2}\log\tau_-$$
$$J_R \approx \frac{2\sigma^2}{(\mu_R-\mu_W)^2}\log\tau_+$$

when the character to be tested is wrong or right respectively.[7]

3 Multi-session Attack

3.1 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 ciphertext block $y = \mathrm{ENC}(x \oplus y')$. (Here y' is the previous ciphertext block following the CBC mode.) The target x is constant in every session, but y and y' depend on the session. The full attack is depicted on Fig. 5. Here the **Check4** oracle 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 $\mathrm{ENC}^{-1}(y) \oplus y'$ for some current key, and some current ciphertext blocks y and y'. We assume that the oracle can get y and y'.

3.2 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 $p = p_1 \cdots p_b$.

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. 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).

[7] For a mathematical treatment on these results we refer to Siegmund [15].

DecryptBlock4
1: **for** $i = 1$ to b **do**
2: $c_i \leftarrow$**DecryptByte4**$(c_{i-1}| \dots |c_1)$
3: **end for**
4: **return** $c_b| \dots |c_1$

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: $j \leftarrow 0$
2: **repeat**
3: $j \leftarrow j + 1$
4: wait for a new session and get the current y and y' blocks
5: let i be the length of u
6: let L be a random string of length $b - i$
7: let $R = (i-1)|(i-1)| \dots |(i-1)$ of length i
8: $r \leftarrow (L|(R \oplus u)) \oplus y'$
9: build the fake ciphertext $f|r|y$ to be sent to the oracle
(f is the longest possible random block sequence)
11: query the oracle and get T_j
(if it is larger than B then go back to Step 4)
13: **until** $\mathrm{STOP}(T_1, \dots, T_j)$
14: **return** $\mathrm{ACCEPT}(T_1, \dots, T_j)$

Fig. 5. Password Interception inside SSL/TLS.

Here we assume that all characters of the unknown block x are independent and uniformly distributed in the alphabet Z. Thus all p_is are equal. We have

$$\begin{aligned} p_1 &= \sum_{i=1}^{|Z|} \frac{1}{|Z|}(1-\varepsilon_+)^{i-1}(1-\varepsilon_-) \\ &= \frac{1-(1-\varepsilon_+)^{|Z|}}{|Z|\varepsilon_+}(1-\varepsilon_-) \\ &\approx (1-\varepsilon_+)^{\frac{|Z|-1}{2}}(1-\varepsilon_-) \end{aligned}$$

when $\varepsilon_+ \ll \frac{1}{|Z|}$ thus

$$p \approx (1-\varepsilon_+)^{b\frac{|Z|-1}{2}}(1-\varepsilon_-)^b.$$

Note that $p \approx 1$ when $\varepsilon_+ \ll \frac{1}{b|Z|}$ and $\varepsilon_- \ll \frac{1}{b}$. Assuming that the algorithm succeeds, the average number or iterations per byte is

$$\sum_{i=1}^{|Z|} \frac{1}{|Z|}((i-1)J_W + J_R) = \frac{|Z|-1}{2}J_W + J_R$$

so the average complexity per block is

$$C = b\frac{|Z|-1}{2}J_W + bJ_R.$$

Numerical examples will be given in a more general context in Section 4.3.

4 Password Interception with Dictionary Attack

4.1 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 precomputed 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.

4.2 Analysis

We consider a list of possible blocks $c_b \dots c_1$. We let $\Pr[c_b \dots c_1]$ be the occurrence probability of a plaintext block. We also let $\Pr[c_i \dots c_1]$ be the sum $\Pr[c_b \dots c_1]$ for all possible $c_b \dots c_{i+1}$.

We arrange the dictionary of all blocks into a search tree. The root is connected to many subtrees, each corresponding to a c_1 character. Each subtree corresponding to a c_1 character is connected to many sub-subtrees, each corresponding to a c_2 character... We label each node of the tree by a $c_i \dots c_1$ string.

We assume that the list of subtrees of any node $c_i \ldots c_1$ is sorted in decreasing order of values of $\Pr[c_{i+1}c_i \ldots c_1]$. We let $N(c_{i+1} \ldots c_1)$ be the rank of the $c_{i+1} \ldots c_1$ subtree of the node $c_i \ldots c_1$ in the list.

We let C_i be the average number to trials for finding c_i (if the attack succeeds) and $C_0 = C_1 + \ldots + C_b$.

We have

$$C_i = \sum_{c_i,\ldots,c_1} \Pr[c_i, \ldots, c_1] N(c_i \ldots c_1)$$

so

$$C_0 = \sum_{i=1}^{b} \sum_{c_i,\ldots,c_1} \Pr[c_i, \ldots, c_1] N(c_i \ldots c_1).$$

Note that $C_0 = b\frac{|Z|-1}{2} + b$ when $c_b \ldots c_1$ is uniformly distributed in Z^b so this generalizes the analysis from Section 3.2.

The expected complexity in case of success is $(C_0 - b)J_W + bJ_R$ thus approximately

$$C \approx \frac{2\sigma^2}{(\mu_R - \mu_W)^2}\left((C_0 - b)\log\frac{1}{\varepsilon_-} + b\log\frac{1}{\varepsilon_+}\right)$$
$$p \approx (1-\varepsilon_+)^{C_0-b}(1-\varepsilon_-)^b$$

when ε_- and ε_+ are small against b^{-1} and C_0^{-1} respectively. The problem is to select ε_- and ε_+ in order to maximize p and minimize C. Computations shows that this is the case when

$$\left(\frac{\delta C}{\delta\varepsilon_+}\right) / \left(\frac{\delta \log p}{\delta\varepsilon_+}\right) = \left(\frac{\delta C}{\delta\varepsilon_-}\right) / \left(\frac{\delta \log p}{\delta\varepsilon_-}\right)$$

hence,

$$\frac{b/\varepsilon_+}{(C_0-b)/(1-\varepsilon_+)} = \frac{(C_0-b)/\varepsilon_-}{b/(1-\varepsilon_-)}.$$

With the assumption that $\varepsilon_+ \ll 1$ and $\varepsilon_- \ll 1$, we deduce

$$\frac{\varepsilon_+}{\varepsilon_-} \approx \frac{b^2}{(C_0-b)^2}$$

thus

$$\varepsilon_+ = \frac{b^2}{t}$$
$$\varepsilon_- = \frac{(C_0-b)^2}{t}$$

for some parameter t. With the same approximation we obtain

$$p \approx \exp\left(-\frac{C_0(C_0-b)b}{t}\right)$$
$$\varepsilon_+ \approx \frac{b}{C_0(C_0-b)}\log\frac{1}{p}$$
$$\varepsilon_- \approx \frac{C_0-b}{C_0 b}\log\frac{1}{p}$$

and finally, we deduce C in terms of the success probability p as well as τ_- and τ_+:

$$C \approx \frac{2\sigma^2}{(\mu_R-\mu_W)^2}\left(C_0\log C_0 - (C_0-2b)\log\frac{C_0-b}{b} - C_0\log\log\frac{1}{p}\right) \quad (5)$$
$$\log\tau_+ \approx \log C_0 + \log(C_0-b) - \log b - \log\log\frac{1}{p} \quad (6)$$
$$\log\tau_- \approx \log(C_0-b) - \log C_0 - \log b + \log\log\frac{1}{p}. \quad (7)$$

Note that $C = O(-\log\log\frac{1}{p})$. If $p \sim 1$ we have $\log\frac{1}{p} \sim 1-p$ so $p = 1-e^{-\Omega(C)}$. The failure probability decreases exponentially with the complexity.

4.3 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 $D = 712'786$ words consists of $\lceil\log_2 D\rceil = 20$ binary questions, so the overhead is only of 11 questions.

Example 2. Table 1 shows complexity C and values $\log\tau_-$ and $\log\tau_+$ for a given success probability p in the case of the dictionary [5] being used and also for uniform distributions with $|Z| = 256$, 128, and 64.

The complexity for $p = 50\%$ is 166 with the dictionary attack and 4239 for a fully random block. Similar computation with the **DecryptBlock2** strategy shows that sequential distinguishers lead to a improvement factor between 4 and 5. This illustrates the power of this technique.

5 Implementation of the Attack

In this section we describe how the **DecryptBlock4** was implemented in practice against an IMAP email server.

Dictionary, $C_0 = 31$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	166	181	199	223	261	380
$\log \tau_-$	-2.74	-3.05	-3.41	-3.88	-4.62	-6.98
$\log \tau_+$	4.86	5.16	5.52	5.99	6.74	9.09

Uniform distribution, $|Z| = 256$, $C_0 = 1028$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	4239	4750	5353	6139	7397	11335
$\log \tau_-$	-2.45	-2.76	-3.12	-3.59	-4.34	-6.69
$\log \tau_+$	12.15	12.46	12.81	13.28	14.03	16.38

Uniform distribution, $|Z| = 128$, $C_0 = 516$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	2179	2346	2738	3132	3764	5741
$\log \tau_-$	-2.46	-2.77	-3.12	-3.60	-4.35	-6.70
$\log \tau_+$	10.76	11.07	11.43	11.90	12.65	15.00

Uniform distribution, $|Z| = 64$, $C_0 = 260$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	1140	1269	1421	1620	1938	2934
$\log \tau_-$	-2.48	-2.78	-3.14	-3.61	-4.36	-6.71
$\log \tau_+$	9.38	9.68	10.04	10.51	11.26	13.61

Table 1. Calculated complexity C and threshold values $\log \tau_-$ and $\log \tau_+$ for algorithm **DecryptBlock4** given success probability p with $\mu_R = 23.63$, $\mu_W = 21.57$, $\sigma = 1.86$ and heuristic value $B = 32.93$ for dictionary [5] and uniform distributions in Z^b.

5.1 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 minutes 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 minutes. 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 minute, 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].

The attack 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 [16] 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.

5.2 Problems and Notes

Two main problems arose when implementing the multi-session attack using the above setup. Firstly, Outlook uses the RC4_MD5 algorithm by default[10] despite the fact that [9] and [14] suggest that 3DES_EDE_CBC_SHA should be supported by default. Hence, we had to force the IMAP server to only offer block ciphers in CBC mode.

The second problem comes from the format of the authentication message. It can be the case that the last bytes of the password belong to the first block of the MAC of the message. So it sometimes happens that the last few bytes of the password cannot be decrypted using the multi-session attack described in this paper. For example, assume that the user name has four characters so that we have the following for an 8-byte block cipher:

```
|0021␣LOG|IN␣"name|"␣"passw|ord"<0x0d><0x0a><HMAC1><HMAC2>|
```

then it will not be possible to decrypt the last three characters from the password.

[8] `http://www.washington.edu/imap/`
[9] `http://www.stunnel.org`
[10] Some other applications like stunnel use the CBC mode by default.

As explained in [13], a countermeasure against the CBC-PAD problem [17] has been implemented in OpenSSL 0.9.6d and following versions so that only the `bad_mac_error` error message is sent when an incorrect padding or an incorrect MAC are detected. However, during our experiments, we have seen that the timing differences still exist and are even easier to identify with this version than previous ones.

6 Discussion

Obviously, the attack works if the following conditions are met.

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

In this paper we focused on the password access control in the IMAP protocol. We can also consider the basic authentication in HTTP [10] which is also used for access control. This means that we can consider intercepting the password for accessing to an Intranet server (e.g. the web server of the program committee of the Crypto'03 Conference!). The attack would work in the same way provided that the above conditions are met, and in particular when the clients of program committee members send their passwords more than a hundred times and do not care about errors.

We can also consider other critical information than just passwords which are sent by the clients to the server. We can consider decrypting a particular constant block of plaintext of a private URL which is retrieved from a server by many clients or by a single one many times. For instance, we can try to decrypt data about the bank account of a client in electronic banking systems. Systems whose security fully rely only on the SSL/TLS protocol would certainly face to security threats. Fortunately, systems that we are aware of use additional security means. They use challenge-response authentication instead of password authentication. They also block accesses in case of multiple failures.

Fixing the problem is quite simple (it was actually done in OpenSSL versions older than 0.9.6i).[11] One can simply try to make error responses time-invariant by simulating a MAC verification even when there is a padding error. One can additionally add some random noise in the time delay. Obviously, session errors should be audited. Note that the problem should also be fixed on the client side which will take much more time than fixing it on the server side. In the meantime, servers should become more sensitive to the types or errors issued by our attack.

Some people claimed that the problem we pointed out in this paper is related to an implementation mistake for SSL/TLS. We can however argue that the problem was raised in [17] in 2002, that an update was made, and that an error

[11] See `http://www.openssl.org`

was still present in current versions one year after. Since there are so many possible mistakes in implementing the protocol we can reasonably claim that this is a protocol problem rather than an implementation one.

For future versions of the TLS protocol we recommend to invert the MAC and padding processes: the sender first pad the plaintext then MAC it, so that the receiver can first check the MAC then check the padding if the MAC is valid. This would thwart any active attack in which messages which are received are not authentic.

7 Conclusion

In this paper we have derived a multi-session variant of the attack of [17] 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. We have shown that when using sequential distinguishers, we can efficiently intercept and decrypt a password for an IMAP account in less than an hour. In doing so, we have also shown that the post-[17] version of OpenSSL [13] is not secure when used with block ciphers in CBC mode. Hopefully, this will have been easily fixed by the time the present paper is published.

Interestingly, we have run one of the first timing attacks through a network and not directly on the attacked device. Another timing attack (against RSA) was run in parallel by Brumley and Boneh [7].

Our attack also illustrates how the theory of sequential distinguishers [11] can be used in order to optimize practical attacks.

8 Acknowledgments

We owe Pascal Junod useful discussions about the timing attack, the idea to fill the message with f, and some helpful details about sequential tests. We thank Gildas Avoine for useful comments. We would also like to thank Bodo Möller for his immediate feedback and the OpenSSL community for caring about our attack in real time. We thank the media for there very positive interest in our results. We also received threats from several companies involved in security which seems to mean that they cared about our results.

The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

References

1. *ISO/IEC 10116*, Information Processing — Modes of Operation for an n-bit Block Cipher Algorithm. International Organization for Standardization, Geneva, Switzerland, 1991.
2. Wireless Transport Layer Security. Wireless Application Protocol WAP-261-WTLS-20010406-a. Wireless Application Protocol Forum, 2001. `http://www.wapforum.org/`
3. *FIPS 46-3*, Data Encryption Standard (DES). U.S. Department of Commerce — National Institute of Standards and Technology. *Federal Information Processing Standard Publication 46-3*, 1999.
4. *FIPS 81*, DES Modes of Operation. U.S. Department of Commerce — National Bureau of Standards, National Technical Information Service, Springfield, Virginia. Federal Information Processing Standards 81, 1980.
5. English Word List Elcomsoft Co. Ltd. `http://www.elcomsoft.com`
6. J. Black, H. Urtubia. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. In *Proceedings of the 11th Usenix UNIX Security Symposium*, San Francisco, California, USA, USENIX, 2002.
7. D. Brumley, D. Boneh. Remote Timing Attacks are Practical. To appear in *Proceedings of the 12th Usenix UNIX Security Symposium*, USENIX, 2003.
8. M. Crispin. Internet Message Access Protocol - Version 4. RFC 1730, standard tracks, University of Washington, 1994.
9. T. Dierks, C. Allen. The TLS Protocol Version 1.0. RFC 2246, standard tracks, the Internet Society, 1999.
10. J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. Internet standard. RFC 2617, the Internet Society, 1999.
11. P. Junod. On the Optimality of Linear, Differential and Sequential Distinguishers. In *Advances in Cryptology EUROCRYPT'03*, Warsaw, Poland, Lectures Notes in Computer Science 2656, pp. 17–32, Springer-Verlag, 2003.
12. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems. In *Advances in Cryptology CRYPTO'96*, Santa Barbara, California, U.S.A., Lectures Notes in Computer Science 1109, pp. 104–113, Springer-Verlag, 1996.
13. B. Möller. Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures. 2002. `http://www.openssl.org/~bodo/tls-cbc.txt`
14. C. Newman Using TLS with IMAP, POP3 and ACAP. RFC 2595, standard tracks, the Internet Society, 1999.
15. D. Siegmund. *Sequential Analysis — Tests and Confidence Intervals*, Springer-Verlag, 1985.
16. M. Ricca. The Denver Projet - A Combination of ARP and DNS Spoofing. Ecole Polytechnique Fédérale de Lausanne, LASEC, Semester Project, 2002. `http://lasecwww.epfl.ch`
17. S. Vaudenay. 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, pp. 534–545, Springer-Verlag, 2002.
18. M. Vuagnoux. CBC PAD Attack against IMAP over TLS. omen. Ecole Polytechnique Fédérale de Lausanne, LASEC, Semester Project, 2003. `http://omen.vuagnoux.com`