Some Fixes To SSH

Xu ZiJie
xuzijiewz@gmail.com

Abstract

To against some known attacks to Secure Shell (SSH), I propose some fixes to SSH. The fixes include add a key producer function and revise the MAC.

Keywords: Authenticated Encryption, SSH, CBC

1. Introduction

Secure Shell (SSH) is designed to protect the data under an authenticated encryption scheme. It is an authenticated encryption. The authenticated encryption provides both privacy and integrity. The basic idea that achieves the goal is use an encryption function provides secrecy and a hash function provides authenticity. At the same time, sent data should let receiver know how many bits will be sent, the sent data will be encode with some conventional format. To know the received data is intact or not, the receiver will do some checks. The check main include data length and MAC. It is obvious the basic idea is rational.

People had found some attacks against SSH. I learn some attacks from papers[1,2,3]. These attacks ably uses the SSH packet format, CBC mode format and check response. I found these attacks have same connotative condition. Then I suggest some fixes to break these connotative conditions.

Wei Dai's attack [3] and plaintext-recover attack [1] has the connotative condition all blocks are encrypted with same key. So I design a key produce function to produce different keys for blocks. Then it cannot apply Wei Dai's attack and plaintext-recover attack.

Replay attack, out-of-order attack and information leakage attack has the connotative condition it need wait for same counter value. Then I improve the MAC that make MAC depend on counter, packet and previous MAC. This fix make attacker need wait for same counter value and same previous MAC value. This will reduce the chance to apply these attacks. If attacker tries to control the previous MAC value to apply these attacks, he need find the collision of hash function.

There are some excellent fixes [2] to SSH. To enrich the fixes, I propose some fixes in this paper. I will glancing SSH and propose some fixes to SSH in section 2. In section 3, I will explain how these fixes against some attacks [1,2,3].

2. SSH Binary Packet Protocol and Some Fixes

In this section, I will simple expound SSH Binary Packet Protocol and some fixes.

2.1 SSH Binary Packet Protocol

I learn SSH Binary Packet Protocol from [1,2]. In a SSH session, at first, the client and the server should agree upon something like shared symmetric keys uk, packet format, encryption function ENC, decoding function DEC, hash function H.

Then sender will encode the message into an encoded packet as follow, suppose packet length is *payloadlen*, padding length is *padlen*, payload message is *m*, padding is *padding*.

$$p = payloadlen \,||\, padlen \,||\, m \,||\, padding$$

If the encryption use CBC mode, the above encoded packet will be divided into blocks that the size depend the encryption function ENC as follow:

$$p = b_0 || b_1 ||, \ldots || b_n$$

Then these blocks will be encrypted with user key uk. In CBC mode, these blocks will be encrypted as follow:

$$cpb_i = ENC_{uk}(cpb_{i-1} \oplus b_i)$$

Where cpb_{-1} is initial value. And the ciphertext is as follow:

$$cp = cpb_0 || cpb_1 ||, \ldots, || cpb_n$$

The MAC as follow:

$$MAC = H(ctr||p)$$

Where ctr is a counter hold by sender and receiver. The ctr is the message number.

The resulting ciphertext is the concatenation of cp and MAC as follow:

$$c = cp||MAC$$

When receiver receive c, receiver will get cp and MAC, he will decrypts the first packet to get the *payloadlen* to know how many data will receive. Then receiver will divided ciphertext into blocks and decipher the ciphertext as follow:

$$cp = cpb_0 || cpb_1 ||, \ldots, || cpb_n$$

$$b_i = cpb_{i-1} \oplus DEC_{uk}(cpb_i)$$

$$p = b_0 || b_1 ||, \ldots || b_n$$

And compute MAC' as follow:

$$MAC' = H(ctr||p)$$

Then check MAC equal MAC' or not to know p is right or not. Because ctr is hold by sender and receiver, so ctr is not send and receiver will know the correct ctr value.

Receiver will do some checks[1]: Length check, Block Length check, MAC check. These checks and the packet format let receiver know how many data the message has and the MAC authenticated the message. And SSH use the mode called encrypt-and-authenticate mode.

2.2 Fixes

The fixes include fixes to CBC mode and encrypt-and-authenticate mode.

2.2.1 Fix to CBC Mode

CBC mode is design to against active attack. [4] shows CBC mode encryption is insecure. I deem the main reason is all block is encrypted with same key. So I propose the fix that

produce different key for every block.

If there is a key uk, the simple way produce key as follow:

$$k_i = uk \oplus i \tag{2.1}$$

k_i is key for i-th block. In a SSH session, there are more than one message. Then to blocks that have same order will has same key. My goal is make attacker hard to find two blocks has same key or not. To achieve this goal, I improve (2.1) as follow:

$$step = uk \vee 0x1$$
$$k_{j,i} = KPF(uk, j, i) = (uk + (step \times j)) \oplus i \tag{2.2}$$

The function KPF(uk,j,i) produce key for i-th block of j-th message. The addition is modular 2^n addition, where n is bit-length of user key.

It can look on (2.2) as produce the key for i-th block of j-th message in two steps:

1.Compute $uk+j\times step$ for j-th message. To convenient, call $uk+j\times step$ message key, noted as $mk_j = uk+j\times step$.

2. Compute the key for i-th block as: $k_{j,i} = mk_j \oplus i$

There exists follow lemmas.

Lemma 2.1: if there exists: $step = uk \vee 0x1$

There exists: $uk+j\times step \in \{0, \cdots, 2^n-1\}$.

n is bit-length of user key uk, $j \in \{0, \cdots, 2^n-1\}$

Proof:

Because: $step = uk \vee 0x1$

So there exists: $step \equiv 1 \bmod 2$

So *step* and 2^n are relatively-prime. Therefore there exists: $uk+j\times step \in \{0, \cdots, 2^n-1\}$. □

So to j-th message, where $j \in \{0, \cdots, 2^n-1\}$, message key is unique. But if the user key has been used to encrypt more than 2^n messages, there will exists same message key. The main size of block cipher are 56, 126,192,256.

By above discuss, this fix is use a key producer function as (2.2) produce key for blocks. Then the i-th block of j-th packet will be encrypted as follow:

$$k_{j,i} = (uk+j\times step) \oplus i$$
$$cpb_{j,i} = ENC_{k_{j,i}}(cpb_{j,i-1} \oplus b_{j,i}) \tag{2.3}$$

To two key $k_{j,i}$ and $k_{j1,i1}$, if there exists $k_{j,i}=k_{j1,i1}$, then there exists:

$$k_{j,u} = k_{j1,i1}$$
$$(uk+j\times step) \oplus i = (uk+j1\times step) \oplus i1$$
$$i \oplus i1 = (uk+j1\times step) \oplus (uk+j\times step)$$

If attacker do not know the user key uk, it is hard to know which two keys $k_{j,i}$ and $k_{j1,i1}$ are same.

2.2.2 Fixes to Encrypt-and-Authenticate Mode

[5,6] summarize that encrypt-and-authenticate mode is insecure. I deem one reason is the

MAC depends on current message and counter value. When attacker applies replay attack, he just need wait the same counter value. We can make the MAC not only depends on current message and counter value but also previous data. In this paper, I use previous MAC. This fix will make the MACs like a chain. If the receiver checks the MAC, the result will depend on the message and the previous MAC. Then attacker applies the attacks that use MAC check need wait same previous MAC and same message number.

Let MAC_i is i-th MAC, then let the MAC computed as follow

$$MAC_i=H(ctr||p||MAC_{i-1}) \quad (2.4)$$

And let the initial value MAC_{-1} is 0.

Let pro(x) is probability of x. Then there exists:

Lemma 2.2: To two pairs (ctr1,p1) and (ctr2,p2), if the corresponding previous MAC is MAC1 and MAC2. If MAC1 and MAC2 are independent, there exists:

$$Pro(H(ctr1||p1||MAC1)= H(ctr2||p2||MAC2))= 2^{-hl}$$

The hash value has *hl* bits.

Proof:

To a given hash value *hv*, there exists:

$$Pro(H(ctr1||p1||MAC1)=hv)=2^{-hl}$$
$$Pro(H(ctr2||p2||MAC2)=hv)= 2^{-hl}$$

There exists 2^{hl} hash values. So there exists:

$$\begin{aligned}&Pro(H(ctr1||p1||MAC1)=H(ctr2||p2||MAC2))\\&=\sum_{i=0}^{2^{hl-1}} Pro(H(ctr1||p1||MAC1)=H(ctr2||p2||MAC2)=i)\\&=2^{hl}\times 2^{-hl}\times 2^{-hl}\\&=2^{-hl}\end{aligned}$$

So there exists:

$$Pro(H(ctr1||p1||MAC1)= H(ctr2||p2||MAC2))= 2^{-hl} \quad \square$$

3. Some Attacks

In this section, I will explain some attacks I learn from [1,2,3]. These attacks have some common characteristic skillfully use packet format and check response. When apply these attacks, it need some conditions. In this section, I will explain how the fixes expounded in section 2 against these attacks.

3.1Plaintext-recover Attack

In [1], plaintext-recover attack had been detailed described. The attacker will collects i-th ciphertext block and (i-1)-th ciphertext block of j-th message $c_{i,i}$' and $c_{j,i-1}$'. Suppose the plaintext is p' and key is k. Then there exists:

$$p'=DEC_k(c_{i,i}')\oplus c_{j,i-1}' \quad (3.1)$$

Then attacker injecting $c_{i,i}$' as the first block of a new packet[1]. Let the initial value of CBC mode is IV. Then there exists:

$$P^{*}=DEC_k(c_{i,i}')\oplus IV \quad (3.2)$$

If p* pass then check, by the first packet format, some bits of p* are fixed value. Combining (3.1) and (3.2), there exists:

$$p'\oplus P^{*}= c_{j,i-1}'\oplus IV \quad (3.3)$$

Then it can recover some bits of p'. In (3.1) and (3.2), the block $c_{i,i}$' is decrypted with same key k. This make let attacker can use (3.3) to bypass the key. If use (2.2) produce key for blocks, suppose attacker will injecting $c_{i,i}$' as the first block of j1-th packet. This exists:

$$step=k\vee 0x1$$
$$k1=(k+(step\times j))\oplus I$$
$$k2=(k+(step\times j1))\oplus 0$$
$$p'=DEC_{k1}(c_{i,i}')\oplus c_{j,i-1}' \quad (3.4)$$
$$P^{*}=DEC_{k2}(c_{i,i}')\oplus IV \quad (3.5)$$

If combining (3.4) and (3.5), there exists:

$$p'\oplus P^{*}= c_{j,i-1}'\oplus IV \oplus DEC_{k1}(c_{i,i}')\oplus DEC_{k2}(c_{i,i}') \quad (3.6)$$

Then the attacker cannot apply this attack. To apply this attack, it need k1=k2, then there exists two pairs (i,j) and (0,j1). There exists:

$$k1=k2$$
$$(uk+j\times step)\oplus i= (uk+j1\times step)\oplus 0$$
$$i=(uk+j1\times step)\oplus(uk+j\times step) \quad (3.6a)$$

If $i\neq 0$, without user key uk, attacker cannot know how to choice i, j, j1 to make (3.6a) tenable. Then it cannot find two blocks decrypted with same key and apply this attack.

3.2 Wei Dai's Attack

W. Dai had reported an attack in [3]. And the attack is represented as follow: If the attacker guesses that plaintext block Pi might be x, and wants to test whether that's the case, he would choose the next plaintext block Pj to be $x\oplus C_{i-1}\oplus C_{j-1}$. If his guess is correct, then $C_j = ENC_k(P_j\oplus C_{j-1}) = ENC_k(P_i\oplus C_{i-1}) = C_i$, and so he can confirm his guess by looking at whether $C_j = C_i$.

W. Dai mentioned a fix [3] that switch to RC4 for encryption to against this attack. And RC4 is a stream cipher.

If the encryption is block cipher and CBC mode, it can use the fix in section 2.2.1 to against this attack. If the attacker guesses that i-th plaintext block $p_{i,j}$ of j-th message might be x, and then choose i1-th plaintext block $p_{i1,j1}$ of j1-th message to be $x\oplus C_{i-1,j}\oplus C_{i1-1,j1}$. Then there exists:

$$step=k\vee 0x1$$
$$k1=(k+(step\times j))\oplus i$$
$$k2=(k+(step\times j1))\oplus i1$$
$$C_{i-1,j}= ENC_{k1}(p_{i,j}\oplus C_{i-1,j})= ENC_{k1}(x\oplus C_{i-1,j}) \quad (3.7)$$
$$C_{i1-1,j1}= ENC_{k2}(p_{i1,j1}\oplus C_{i1-1,j1})= ENC_{k2}(x\oplus C_{i-1,j}) \quad (3.8)$$

To apply this attack, it need k1=k2. Then there exists:

$$k1=k2$$
$$(uk+j\times step)\oplus i= (uk+j1\times step)\oplus i1$$
$$i\oplus i1=(uk+j1\times step)\oplus(uk+j\times step)$$

Without user key uk, attacker cannot know how to choice i, j, i1, j1 to make k1=k2. Then it cannot find two blocks decrypted with same key and apply this attack.

3.3 Replay and Out-Of-Order Delivery Attack

I learn replay and out-of-order delivery attack from [2]. In SSH, the MAC is compute as follow:

$$MAC=H(ctr||p) \tag{3.9}$$

Where ctr is a 32-bits counter of message number, and p is encoded packet. The MAC depends on counter and present message.

For replay attacks, attacker get a ciphertext c, then the receiver will decipher the ciphertext and get the packet p and MAC. Suppose the message number counter value is ctr. There exists:

$$MAC=H(ctr||p)$$

Then attacker will wait for more 2^{32}-1 messages. Then the message number counter value will be ctr again. Then attacker can send the ciphertext c to receiver again. And receiver deciphers ciphertext c and get same packet p and MAC again. It is undoubtedly that p will pass check again.

For out-of-order delivery attacks, if sender had the sender has encrypted more than 2^{32} messages. If there exists two pair (ctr1,c1) and (ctr2,c2), suppose the corresponding packets of c1 and c2 are p1 and p2. And there exists:

$$\begin{cases} ctr1 \equiv ctr2 \mod 2^{32} \\ H(ctr1 \bmod 2^{32} \,||\, p1) = H(ctr2 \bmod 2^{32} \,||\, p2) \end{cases}$$

Attack can modify the order that swap c1 and c2.

Mihir Bellare [2] mentioned that if the counter is allowed to repeat, it is easy apply replay and out-of-order delivery attack. For fix in section 2.2.1, by lemma 2.1, the key produce function will produce same key after 2^n messages. And the main key size of block cipher are 56, 126,192,256. The circle of message key is bigger than 2^{32}. If receiver receives the ciphertext after 2^{32} messages, the ciphertext will be deciphered with different key.

By lemma 2.1, the circle of message key is 2^n, 2^n is integer multiple of 2^{32}. So to apply replay and out-of-order delivery attack, attacker need wait for more than 2^n messages to wait key produce function produce same key.

By (3.9), we will know that attacker can wait for more 2^{32} messages for the same counter and apply reply and out-of-order delivery attack. The fix in section 2.2.2 will make the MAC depend on current message, counter and previous MAC. Fix in section 2.2.2 will make attacker not only wait for same counter value but also same previous MAC. Let MAC is (i-1)-th MAC, MAC' is (i-1+2^{32})-th MAC. The probability of MAC=MAC':

$$
\begin{aligned}
&\Pr o(\mathrm{MAC} = \mathrm{MAC}') \\
&= \sum_{i=0}^{2^{hl-1}} \Pr o(\mathrm{MAC} = \mathrm{MAC}' = \mathrm{i}) \\
&= 2^{hl} \times 2^{-hl} \times 2^{-hl} \\
&= 2^{-hl}
\end{aligned}
$$

If the previous MACs are not same, by lemma 2.2, if ciphertexts are replayed or reordered, the probability that pass MAC check is 2^{-hl}.

3.4 Information Leakage

Information leakage attack is mentioned in [2]. Suppose that an attacker get a ciphertext with a MAC tag t. The attacker can guess the underlying payload is M. He will wait for the sender to encrypt $2^{32} - 1$ more packets and then requests the sender to encrypt the payload M. Then the attacker gets the new MAC tagt t'. If the guess is right, there exists t'=t.

In encrypt-and-authenticate mode the MAC is not encrypted, so attacker just checks the MAC. The fix in section 2.2.1 is not help for against this attack.

The fix in section 2.2.2 will make the MAC depend on current message, counter and previous MAC. After wait for the sender to encrypt $2^{32} - 1$ more packets, if the MAC is not the same MAC, even attacker's guess is right, the probability of t'=t is is 2^{-hl} by lemma 2.2.

4. Improvement

The fixes in section 2 are base ideas. There are some improvements for these fixes.

4.1 Improvement to Fix in Section 2.2.1

In section 3.3, we can find that the circle of message is 2^n. 2^n is an integer multiple of 2^{32}. So attacker can wait for more 2^n messages. If circle of key produce function is 2^n-1, 2^n-1 and 2^{32} are relatively-prime, then attacker will wait for more 2^{n+32}-2^{32} messages to wait for same counter value and same key. There is a simple way to do this. The key of i-th block of j-th message as follow:

$$
\begin{aligned}
&step = uk \vee 0x1 \\
&mk_j = \begin{cases} uk & j = 0 \\ uk + step & mk_{j-1} + step = uk \\ mk_{j-1} + step & else \end{cases} \\
&k_{j,i} = mk_j \oplus i
\end{aligned}
$$

When j>0, there exists $mk_j \neq uk$. So the circle of mk is 2^n-1.

4.2 Improvement to Fix in Section 2.2.2

The fix in section 2.2.2 makes the i-th MAC computed as follow

$$MAC_i=H(ctr||p||MAC_{i-1})$$

In same situation, the packet is not arriving in order. Then it cannot get the previous MAC immediately. There are some ways to solve this problem. I provide a simple resolvent here. It is change (2.4) as follow:

$$ind(i)=\begin{cases}-1 & i<2^{32}\\ i-2^{32}-(i \bmod 2^{32}) & 2^{32}\le i\end{cases}$$

$$MAC_i=H(ctr||p||MAC_{ind(i)}) \qquad (4.1)$$

(4.1) is use a steady value in a period of time. Then it just need hold some MACs, it will solve this problem. In this paper, I do not discuss it in details.

5. Countermeasures

In this paper, I study some attacks [1,2,3] on SSH. I find the condition that these attacks need. And I propose some fixes to SSH in section 2. And these fixes are easy and simple.

The fix in section 2.2.1 produce difference key for blocks with a key produce function. The fix can make it hard to use CBC mode format to apply attack. And I notice that the fix maybe can be applied to ECB mode.

The fix in section 2.2.2 improve encrypt-and-authenticate mode. It makes MACs like a chain. The fix makes it hard to use MAC check apply attacks.

From the attacks that mentioned in [1,2,3], we can find that against attack is a complicated task. It need consider some different attacks and the limit by some environment. In this paper, I propose fixes that can against the attacks that I know. These fixes are base ideas. In some severe environment, it need revise these fixes to fit the environment.

References

1. Martin R. Albrecht, Kenneth G. Paterson, Gaven J. Watson: Plaintext Recovery Attacks against SSH. IEEE Symposium on Security and Privacy 2009: 16-26

2. M. Bellare, T. Kohno, and C. Namprempre. Breaking and Provably Repairing the SSH Authenticated Encryption Scheme: A Case Study of the Encode-then-Encrypt-and-MAC Paradigm. ACM Transactions on Information and Systems Security, 7(2):206–241, 2004.

3. W. Dai. An Attack Against SSH2 Protocol. Email to the SECSH Working Group available from ftp://ftp.ietf.org/ietf-mail-archive/secsh/2002-02.mail, 6th Feb. 2002.

4. P. Rogaway. Problems with proposed IP cryptography, 1995. Available at http://www.cs. ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt.

5. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, Advances in Cryptology – ASIACRYPT 2000, volume 1976 of Lecture Notes in Computer Science, pages 531–545.

Springer-Verlag, Berlin Germany, Dec. 2000.

6. H. Krawczyk. The order of encryption and authentication for protecting communications (or:How secure is SSL?). In J. Kilian, editor, Advances in Cryptology – CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science, pages 310–331. Springer-Verlag, Berlin Germany, Aug. 2001.