Security analysis of VTun

Jerome Etienne jme@off.net

Abstract

This text is a security analysis of VTun. It includes a description of the security (see section 2) based on the source and lists the possible attacks (see section 3). An attacker can modify packets, replay them, learn pattern of the plain text or easily guess low-entropy passwords.

1 Introduction

From the man page, "VTun provides the method for creating Virtual Tunnels over TCP/IP networks and allows to shape, compress, encrypt traffic in that tunnels." The analysis has been done on VTun version 2.5b1 which has been found at http://vtun.sourceforge.net.

From the FAQ, "VTun doesn't try to be the MOST secure tunneling software in the world, it tries to be ... secure enough instead." In my opinion, it is a rather dangerous statement as the definition of the 'enough' entirely depends on the user and not on the designer or implementor. This text aims to clarify the security provided by VTun.

2 Security description

The security has been analyzed from the source as the distribution doesn't contain any detailed description.

2.1 packet forwarding

The forwarded packets are encrypted with blowfish in ECB using MD5(user password) as encryption key (see lfd_encrypt.c). As ECB requires the cipher text to be block aligned and blowfish has 64bit blocks, the packet is 64bit aligned. The pad is zeros prepended to the packet and the first byte of the packet is its length.

2.2 Connection establishment

During the connection establishment, the client authenticates itself to the server with a challenge/response scheme (i.e. a simple way to authenticate without sending passwords in clear) based on a user password. The challenge is 16bytes of random (see VTUN_CHAL_SIZE) chosen by the server. They are encrypted with a key equal to MD5(user password). The server sends the encrypted challenge to the client, the client decrypts it and replies to it.

The above explanation assumes the HAVE_SSL is defined. If it isn't, the authentication is very insecure because the challenges is just XOR-ed with the password, and the challenge is based on rand() output which is known as easily predictable.

3 Vulnerabilities

This section explain how an attacker can modify packets, replay them, learn pattern of the plain text or easily guess a low-entropy password.

3.1 forwarded packet aren't authenticated

The forwarded packets aren't authenticated, so an attacker can modify them without being detected. The aim of encryption is to make the data unreadable for anybody who doesn't know the key. It doesn't prevent an attacker from modifying the data. People assume that an attacker won't do it because the attacker wouldn't be able to choose the resulting clear text. But this section shows that the attacker can choose the resulting clear text to some extent and that modifying the cypher text data may be interesting even if the attacker ignores the result.

3.1.1 To insert random data

If the attacker modifies the cipher text without choosing the resulting clear text, it will likely produce random data. The legitimate user won't detect the modification and will use them as if they were valid. As they likely appears random, it will result in a Denial of Service (aka DoS).

3.1.2 To insert chosen data

The encryption mode used by encrypted loop device is ECB[oST81]. ECB allows cut/paste attacks i.e. the attacker can cut encrypted data from one part of

a packet and paste them anywhere in another packet. As both data sections have been encrypted by the same key, the clear text won't be completely random data.

This lack of authentication isn't a ECB flaw. Authentication isn't considered an aim of the encryption mode, so most modes (e.g. CBC, CFB, OFB) don't authenticate the data. To use another mode would be flawed in the same way except if they explicitly protect against forgery. Recently some modes including authentication popped up to speed up the encryption / authentication couple but as far as i know they are all patented.

3.2 Easy dictionary attacks

The authentication is based on a secret key chosen by the user. The key is trivially derived from the user password by computing MD5(user password).

Unfortunately, users often choose low-entropy passwords because those are easier to remember, even if it is a bad behavior from a security point of view. This allows attackers to try dictionary attacks i.e. to try likely password (e.g. jack the ripper). This weakness isn't inherently a VTun weakness as the password choice depends on the users. He may choose a random password (e.g. /dev/random output) and so won't be vulnerable.

When the security ultimately relies on a low-entropy password chosen by a user, dictionary attacks can't be stopped but they can be made sufficiently harder to be impractical (e.g. salt, key derivation sufficiently slow). VTun doesn't use those tricks.

3.3 No anti-replay protection

VTun doesn't include any protection against packet's replay, so an attacker who eavesdrops the encrypted packets can successfully replay them later and the destination will consider them as legitimate. They can be replayed inside the same tunnel or in another instance the tunnel. The attacker can even replay them to the source: a packet from A to B can be send to A which will accept it.

3.4 Usage of ECB

VTun uses blowfish with ECB but ECB doesn't hide the plain-text patterns inside a given packet or between distincts packets. A given plain text block will produce the same cipher text block independently of the packet it is in and of its location inside it. The attacker can recognize the repetition of identical cipher text blocks and obtain information on the plain text.

4 Conclusion

This text describes vulnerabilities of VTun security. An attacker can modify packets, replay them, learn patterns of the plain text or easily guess low-entropy passwords. All those attacks are independent and can be combined to perform even stronger attacks.

References

[oST81] National Institute of Standards and Technology. implementing and using the nbs data encryption standard. *Federal information processing standards fips74*, April 1981.