Security of Encrypted rlogin Connections Created With KerberosIV

Kirsten Hildrum
University of California,Berkeley
hildrum@cs.berkeley.edu

Abstract

KerberosIV is an authentication system originally developed by MIT's Project Athena. Using Kerberos authentication, the client and the server can each verify the identity of the other party during connection initialization. As a side effect, the client and the server share a key once the connection is created, allowing subsequent messages to be encrypted.

One common Kerberos application is `rlogin`*. Kerberized* `rlogin` *allows the server to authenticate the client and includes an option that encrypts sent data. Because the Kerberos documentation describes a message format that would protect such encrypted messages from replay and other attacks, most users assume Kerberized* `rlogin` *provides that protection. This paper will show that Kerberized* `rlogin` *does not use that message format once the connection has been established, leaving the user open to attacks reminiscent of the active TCP attacks describe by Joncheray in [3]. Extension of these attacks can possibly lead to discovery of the session key.*

1 Introduction

KerberosIV is an authentication system originally developed by MIT's Project Athena. Using Kerberos authentication, the client and the server can each verify the identity of the other party during connection initialization. As a side effect, the client and the server share a key once the connection is created, allowing subsequent messages to be encrypted.

One common Kerberos application is `rlogin`. Kerberized `rlogin` allows the server to authenticate the client and includes an option that encrypts sent data. Because the Kerberos documentation describes a message format that would protect such encrypted messages from replay and other attacks, most users assume Kerberized `rlogin` provides that protection. This paper will show that Kerberized `rlogin` does not use that message format once the connection has been established, leaving the user open to attacks reminiscent of the active TCP attacks describe by Joncheray in [3]. Extension of these attacks can possibly lead to discovery of the session key.

1.1 Overview of Kerberos

Here I present a brief overview of Kerberos. A detailed description can be found in [4]. KerberosIV relies on a central authority which shares passwords with both clients and servers. To open a connection, the client asks this central authority for a "ticket" for the server. The central authority sends this ticket and a session key (for use between the client and the server) to the client in an encrypted message. The ticket itself is encrypted for the server (and so, unreadable to the client), and includes the same key that was sent to the client. To establish a connection to the server, the client sends the ticket to the server. The server verifies the client's identity using the shared session key.

1.2 Encryption and authentication

This section briefly describes two functions that can be used when creating an `rlogin` connection. This paper does not discuss weaknesses in these functions, but I mention them for comparison with `des_write`, the function used to encrypt messages once the connection has been established.

The Kerberos distribution (as described in [4]) provides a functions to send authenticated messages, and a similar function to send encrypted messages. Both of these are done using the shared session key. The format of an authenticated (or "safe") message consists of: protocol information, user data, timestamp, and sender's network address. All information except the protocol information is included in the checksum, which is calculated using the key. A private (or encrypted) message, created with a function called `krb_priv`, contains protocol information and a length field for the length of the encrypted data followed by an encrypted section consisting of: length of user data, user data, time-stamp, and sender's IP address.

1.3 Specifics of encryption in KerberosIV

KerberosIV uses DES (Data Encryption Standard) for encryption. DES is a symmetric key block cipher which encrypts in blocks of eight bytes. I will use DES(P) to mean the DES algorithm applied to P (in this paper, the key will be implied), where P is a block of eight bytes.

DES encryption has several modes. Kerberos uses PCBC mode (Propagating Cipher Block Chaining) to chain blocks together. In this mode, before encryption, the plaintext is exclusive-ored with the ciphertext and plaintext of the previous block. As a result, if there is an error, all the blocks after the error are garbled.

In mathematical notation, if the plaintext blocks are $P_1, P_2, \ldots P_n$, then the ith ciphertext block C_i is $DES(P_i \oplus P_{i-1} \oplus C_{i-1})$. Normally, an initial vector IV is used in encrypting the first block, so $C_0 = DES(P_0 \oplus IV)$. Decryption is the reverse: $P_i = DES^{-1}(C_i) \oplus C_{i-1} \oplus P_{i-1}$, and $P_0 = DES^{-1}(C_0) \oplus IV$.

In KerberosIV, a 64 bit version of the key is used as the IV. The 56 bit DES key is divided into eight seven-bit pieces. To each of these seven-bit pieces, a parity bit is added, so that the now eight-bit block has even parity. Then these eight eight-bit pieces are concatenated to form the IV.

1.4 This paper

In this paper, I will look at the weaknesses in `krsh` and `krlogin`, which are versions `rsh` and `rlogin` which use Kerberos. The specific version used is described as "rlogin.c 8.3 (Berkeley) 8/31/94." Since I do not discuss plain `rlogin`, I use `rlogin` to refer to `rlogin` with KerberosIV. I made my observations using TCP dump on encrypted `rlogin` cconnections from Linux to HPUX and Solaris. The client, or the user, is the one who opens the connection. The server is the computer to which the user is logging in.

2 Replay attacks

When `rlogin` or `rsh` sends an encrypted message, it does not use the encryption function described in section 1.2. Instead, it uses a function called `des_write`, shown in figure 1.

To summarize, all encryption is done with DES in PCBC mode using the key as the initial vector. The function starts by sending the length of the unencrypted data. Then, if the data is more than eight bytes, it pads the data with zeros, encrypts it, and sends. If the data is less than eight bytes, it is left-padded with random data before encryption.

This means that identical messages longer than eight bytes always encrypt to the same ciphertext. Furthermore, identical messages shorter than eight bytes (individual keystrokes) have different, but interchangeable, encryptions. This is because they are padded with random data that is ignored upon decryption.

The first and most obvious effect of this is that it is possible to replay any keystroke sent by the user. To do this, copy the packet in which the keystroke was sent, adjust the TCP sequence number and checksum, and resend. The faked packet will be accepted as if it was sent by the user. Similarly, any packet sent *to* the user can also be resent such that it appears to have come from the user; simply adjust the IP addresses along with the sequence number and checksum. If the attacker can get a big enough dictionary, the attacker can send an arbitrary command: say `echo ++ > .rhosts`. This could be a serious security hole.

If the Kerberos private message format (described in section 1.2) had been used, this sort of attack would be much more difficult since `krb_priv` messages have a timestamp. Even the Kerberos safe message format, while not providing privacy, would protect against this attack, assuming the security of the checksum (which may not be wise [1]).

Below, I list several peculiarities of the `rlogin` and `des_write` which increase the keystrokes available to replay.

2.1 Small packets

Many of the packets exchanged in a `rlogin` session are likely to be single keystrokes. For example, when the user logs in to read mail, after the initial set up packets, the next packets to be sent are "p", "i", "n", "e", and "\n". Of course, the attacker will not know which letters were typed—he or she will only see that there were five single-letter packets followed by a large response from the server. This would probably be enough for the attacker to guess what the user has just typed. Having an encryption of "p","i","n", and "e", the attacker could send "pine", but could also send any of the letters individually just as easily.

Note that the attacker cannot simply do a frequency analysis on the packets encrypting a single letter. Before encryption, data less than eight bytes long is left-padded with random data; so while the encryptions are interchangeable, since the random data is thrown away upon decryption, the encryptions are different.

```
int
des_write( int fd, const char* buf, int len)
{
        static  int     seeded = 0;
        static  char    garbage_buf[8];
        S_BIT32 net_len, garbage;

        if(len < 8) {
                if(!seeded) {
                        seeded = 1;
                        srandom((int) time((time_t *)0));
                }
                garbage = random();
                /* insert random garbage */
                (void) bcopy(&garbage, garbage_buf, MIN(sizeof(S_BIT32),8));
                /* this "right-justifies" the data in the buffer */
                (void) bcopy(buf, garbage_buf + 8 - len, len);
        }
        /* pcbc_encrypt outputs in 8-byte (64 bit) increments */
        (void) des_pcbc_encrypt((len < 8) ? garbage_buf : buf,
                            des_outbuf,
                            (len < 8) ? 8 : len,
                            key_schedule,       /* DES key */
                            key,                /* IV */
                            ENCRYPT);
       /* tell the other end the real amount, but send an 8-byte padded
           packet */
        net_len = htonl(len);
        (void) write(fd, &net_len, sizeof(net_len));
        (void) write(fd, des_outbuf, roundup(len,8));
        return(len);
}
```

Figure 1: `des_write`: the function actually used by `rlogin` to encrypt data. The format has been slightly modified.

2.2 Length in the clear

The length is sent in the clear before the encrypted data. That means that an encrypted message of “You have mail” could be replayed as a number of things, such as “You have m” or “You have mail” just by adjusting the length field. By only sending the first eight bytes of the encrypted messages and adjusting the length field, the attacker could also send any of the following: “e”, “ve”,“have”, or “You have”. In general, since the length is in the clear, the attacker can chose to replay the first n characters of the message, if $n \geq 8$, and the last n characters in the first packet if $n \leq 8$. This gives the attacker additional flexibility in replaying packets.

2.3 Same encryption both directions

Since the same key is used in both directions, messages from A to B can be used to impersonate messages from B to A. This gives the attacker additional replay material. Notice that `krb_priv` the message format described in section 1.2 prevents this from happening, as long as the sender and the attacker are on different machines, because it includes the sender’s IP address in the encrypted message.

Combined with some traffic analysis, this is useful to the attacker. For example, in many cases, the prompt is the same every time it is sent, and oftentimes, prompts are predictable. The attacker could look for the encryption of the prompt (by looking for something repeated frequently) and use it for additional material to replay. Starting an email program could also send predictable screen drawing commands, and if the attacker can guess which packet is the encryption of those screen drawing commands, the attacker can use that as additional replay material.

2.4 Active code book building

A clever attacker need not wait to get by chance the encryption of a certain keystroke. In certain situations, the attacker can *cause* an encryption of a given keystroke to be sent.

For example, assume that the client is reading email over the connection using Pine. The user will probably hit "n" to view the next message. When the server machine was an HP, the packet sent after the I hit "n" was always 240 bytes in length. Suppose the attacker sends the following email:

```
Date: Sat, 12 Dec 1998 19:38:28 -0800 (PST)
From: "Kris W. Hildrum" <hildrum@cs.berkeley.edu>
To: hildrum@cs.berkeley.edu
Subject: aaecho ++ > .rhosts
```

The first 240 byte encrypted chunk is not very useful, but the message as been carefully prepared so that the second block of encrypted data will begin with the encryption of `echo ++ >.rhosts`. That means that the attacker can now replay `echo ++ >.rhosts`, or any other chosen text. A clever attacker might even being able to arrange the rest of the header so the subject line is something fairly plausible.

In general, it may not be possible to predict the points that will be the start of an encrypted packet. (For example, on the Sun OS, it is not possible to do exactly the attack described above.) However, the attacker may still be able to send an email message and discover the encryption of a given plain text. For example, if the attacker needs an encryption of a "+", the attacker can send an email message consisting entirely of "+"s, and then is guaranteed to get an encryption of the "+".

3 Recovering the key in PCBC mode

If the attacker can force either party to decrypt ciphertext chosen by the attacker, the attacker can recover the initial vector when PCBC mode is used. PCBC mode is exactly the mode chosen by KerberosIV. The idea is taken from [5, 2].

Since the initial vector is also the key, this is very bad. Further, while this key is called a "session key", it is used over several sessions. It is valid for as long as the ticket is valid, which is essentially all day. This key is also vital to the authentication process; if the attacker knows the key, the attacker can masquerade as the client and establish new connections.

3.1 Chosen ciphertext attack

Suppose that you know some plaintext P_0 that is encrypted with the initial vector to get C_0. Then, the attacker waits to see some ciphertext $C_1C_2\ldots C_n\ldots$. If the attacker can put C_0 somewhere into that stream of ciphertext, and get the decryption, then the attacker can recover the key. Suppose the attacker managed to get the decryption of $C_1C_2\ldots C_nC_0C_{n+1}\ldots$. Let this decrypt to $P_1P_2P_nPP_{n+1}\ldots$.

Here is how the attacker can recover the key.

$$\begin{aligned} P &= DES^{-1}(C_0) \oplus C_n \oplus P_n \\ &= (P_0 \oplus K) \oplus C_n \oplus P_n \end{aligned}$$

So, $K = P \oplus P_0 \oplus C_n \oplus P_n$. Since the attacker knows P_0, P, C_n and P_n, the attacker has the key.

One problem is that all blocks after the P_n will not decrypt correctly. However, [2] shows that inserting two copies of C_0 will allow the attacker to recover the initial vector but will correctly decrypt P, but with 16 bytes of random bits between P_n and P_{n+1}.

3.2 A more practical view

While not implemented, there is a way to make the above attack work in practice. The idea is from David Wagner [5]. Notice that finding a P_0 is easy—one example is the prompt. The difficult part would be getting a decryption of the chosen ciphertext. However, using a list of replayable keystrokes built using the methods above, this could be done by replaying commands to the victim's computer to write mail to the attacker.

The ciphertext to be decrypted could be sent as the body of the email. The attacker would then get the decryption of the chosen ciphertext. Alternatively, the attacker could also just send mail anywhere and watch traffic to the sendmail port.

This could be carried out without the user noticing. Imagine that the user was logged working on one computer and had `rsh` into another computer on which he or she was reading mail. After the mail is sent, the sequence numbers of the real connection would be wrong, but the user would probably simply close the connection, not realizing that there was a security problem. By then, however, the attacker has the key (which is still valid when the user reinitiates the connection).

4 Conclusions

There are a number of small lessons to be learned from this exercise and there is also one larger lesson. The small lessons concern the cryptographic flaws, many of which already appear in [1].

- Using any constant initial vector defeats the purpose of using a chaining mode, especially if the data length is frequently less than the block size.
- The key should not be used as an initial vector.
- Session keys valid for only a single session make it more difficult to carry out attacks.
- Messages of encrypted data should contain their length to prevent truncation; a timestamp or sequence number to prevent replay; and the sender to prevent echoing.

These lessons are well understood by most cryptographic protocol designers, are corrected in KerberosV, and the most serious are addressed in the KerberosIV authentication protocol.

But the larger lesson is that applications using cryptographic systems need to be examined in detail even when the cryptographic underpinnings are known to be sound. The flaws in the KerberosIV `rlogin` protocol are obvious. The degree to which those flaws are exploitable is surprising, but not astonishing. Even though KerberosIV `rlogin` has been deployed for years and the source code has been available, no one noticed the problem because no one looked.People have found much more subtle flaws in the Kerberos protocol itself because the details were published. Had the `rlogin` implementation been proprietary, its flaws might never have been revealed.

5 Acknowledgments

I would like to thank David Wagner for both the initial idea for this paper and his helpful suggestions on how to organize it. I would also like to thank Charlie Kaufman for his help in preparing the final version.

References

[1] Steven M. Bellovin and Michael Merritt. Limitations of the Kerberos authentication system. In *USENIX*, 1991.

[2] Adrian M Iley. Kerberos ivec attack. `http://www.andrew.cmu.edu/~iley/kerbatck/kerbatck.html`.

[3] Laurent Joncheray. A simple active attack against tcp. In *USENIX UNIX Security Symposium*, 1995.

[4] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX*, 1988.

[5] David Wagner. sci.crypt: Re: Security of des key encrypted with its self???? `http://www.cs.berkeley.edu/~daw/my-posts/key-as-iv-broken-again`, December 1996.