

[UNIX] Vulnerabilities in the Kerberos Version 4 Protocol

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2003-03/0053.html>

From: support@securiteam.com

Date: 03/19/03

From: support@securiteam.com

To: list@securiteam.com

Date: 19 Mar 2003 13:40:34 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

In the US?

Contact Beyond Security at our new California office housewarming rates on automated network vulnerability scanning. We also welcome ISPs and other resellers!

Please contact us at: 323-882-8286 or ussales@beyondsecurity.com

Vulnerabilities in the Kerberos Version 4 Protocol

SUMMARY

Several cryptographic vulnerabilities exist in the basic Kerberos Version 4 protocol that could allow an attacker to impersonate any user in a Kerberos realm and gain any privilege authorized through that Kerberos realm. Knowledge of the key shared between two realms for Kerberos 4 cross-realm authentication or the ability to create arbitrary principals in a realm is sufficient to print any ticket in the realm. As an example, knowing [krbtgt.ZONE.MIT.EDU@ATHENA.MIT.EDU](#) is sufficient to print an Athena TGT for any Athena realm client. Additional vulnerabilities in a MIT extension to use triple DES keys for Kerberos 4 tickets may allow attackers who can passively observe the network to construct tickets for some users if certain alignment constraints are met.

The Kerberos 5 protocol is not vulnerable to this issue. However, implementations that implement both Kerberos 4 and Kerberos 5 tend to use the same keys for both protocols. As a result, the Kerberos 4 vulnerabilities can be used to compromise Kerberos 5 services with these implementations.

DETAILS

Kerberos version 4 tickets include neither a cryptographic hash of the encrypted data, random padding, nor a random initial vector. As such, if an attacker can cause the right text to be encrypted in a Kerberos service key, then the attacker can fabricate a ticket. Normally an attacker does not control much of the text in the ticket so this cryptographic weakness is hard to exploit.

The initial portion of a Kerberos 4 ticket is a one-byte flags field (either 0 or 1) followed by the client name. Since all of this initial text is constant, the beginning of a ticket for a given client/service will be the same. An attacker thus knows the encryption of the initial plaintext in the service key. If an attacker can control client principals whose names he chooses, then he can get the encryption of these plaintext values in the service key. An attached paper details how to choose principal names in order to encrypt arbitrary plaintext and how to use this ability to construct tickets for both Kerberos 4 and Kerberos 5.

Because of concerns about single DES weaknesses, MIT implemented support for Kerberos 4 tickets encrypted in triple DES service keys. This support shares all the cryptographic weaknesses of single DES Kerberos 4. In addition, since it uses CBC mode rather than PCBC mode, it introduces new weaknesses not found in other Kerberos 4 implementations. When certain alignment constraints are met, it is possible to splice two tickets together, allowing an attacker to get a ticket with a known session key for a client without knowing that client's long term key. This attack does require sniffing a ticket for that client.

We do not believe the password changing service is vulnerable to the single DES attacks as the KDC will never issue password-changing tickets in an application request. It is probably vulnerable to the triple DES splicing attacks.

Specific Vulnerabilities

1) ECB Oracle for Single DES

By controlling principals of an attacker's choice, an attacker can encrypt arbitrary plaintext in a single DES service key.

2) ECB Oracle for Triple DES

By controlling principals of an attacker's choice, an attacker can encrypt arbitrary plaintext in a triple DES service key.

3) PCBC First Block

It turns out that being able to encrypt arbitrary plaintext is not quite enough to construct a ticket for a single DES service key. You also need to be able to construct the first block of the ticket; you do not know what plaintext to use because the IV for the first block is the long-term

service key. However since the only thing in the first block of the ticket is the first seven bytes of the client, controlling a principal with the same first seven bytes as the principal being attacked is sufficient to get the first block. As a practical matter, principals whose principal and instance components fit within six bytes (including trailing nulls) may be harder to attack. The attached paper discusses mechanisms for mounting such an attack.

4) Cross Realm

If realms A and B share a cross-realm key and the attacker knows that key or can get arbitrary plaintext encrypted in that key, then the attacker may get A to issue tickets for any principal claiming to be in realm B and vice versa. This is sufficient to meet conditions of vulnerabilities 1 and 2 above and to encrypt arbitrary plaintext in the service keys of realm A and B.

5) Kerberos 4 Ticket Printing

The conditions of 2 above are sufficient to print arbitrary tickets in a triple DES service key. The conditions of 1 and 3 are sufficient to print any ticket in a single DES service key.

6) Kerberos 5 Ticket Printing

The conditions of 1 above are sufficient to construct a des-cbc-md4 or des-cbc-md5 Kerberos 5 ticket if the KDC uses the same DES key for v4 and v5. While the Kerberos 5 ticket does have a confounder and checksum, the checksum is not keyed and thus the confounder and checksum can be fabricated by an attacker. We believe that des-cbc-crc is safe unless you can contain a ciphertext block and a corresponding plaintext block; the paper discusses situations where this is possible. However most Kerberos implementations will allow des-cbc-md5 to be used even if des-cbc-crc is normally used. We are not aware of any vulnerabilities in des3-hmac-sha1-kd or rc4-hmac-md5 implementation.

7) Ticket Splicing Attack

A Kerberos 4 ticket contains an eight-byte session key. If client principal names are chosen carefully then this session key will line up with a DES block boundary. For triple DES service keys, this creates an opportunity for an attack. Consider the case where an attacker has obtained a ticket t1 with a known session key K and has sniffed a ticket t2 with unknown session key for the same service. The attacker can create a new valid ticket t2' by replacing the part of t2 starting with the session key block with the session key from t1. This new ticket will have a session key K XOR-ed with the ciphertext blocks preceding the session key in t1 and t2. In other words, if triple DES service keys are used, client principals with the wrong name lengths are inherently vulnerable to sniffing.

8) Realm Hopping

Kerberos 4 does not normally support multi-hop cross-realm authentication. However cross-realm tickets are just normal service keys; points 1,2 and 3 are sufficient to satisfy the conditions of point 4 for a service key. That is, an attacker can hop through realms, exploiting these vulnerabilities against any realm that is in the transitive closure of the initial realm. Anyone who shares keys with ATHENA.MIT.EDU now trusts ZONE.MIT.EDU.

9) Krb 524 Does Not Help

Traditionally realms desiring higher security but still wishing to have some Kerberos 4 services have disabled KDC support for V4 and used krb524d to issue only the services that are needed. These vulnerabilities work as well against any service key that the krb524d knows as they do against service keys in a v4 KDC. Of course, a fabricated krb5 ticket can be converted to Kerberos 4 using krb524d.

Potential Solutions:

1) V4 Cross Realm Considered Harmful

Kerberos implementations should gain an option to disable Kerberos 4 cross-realm authentication both in the KDC and in any implementations of the krb524 protocol. This configuration should be the default.

2) Application Migration

Application vendors and sites should migrate from Kerberos version 4 to Kerberos version 5. The OpenAFS community has introduced features that allow Kerberos 5 to be used for AFS in OpenAFS 1.2.8. Patches are available to add Kerberos 5 support to OpenSSH. Several other implementations of the SSH protocol also support Kerberos 5. Applications such as IMAP, POP, and LDAP already support Kerberos 5.

3) TGT Key Separation

One motivation for the V4 triple DES support is that if a single DES key exists for the TGT principal then an attacker can attack that key both for v4 and v5 tickets. Kerberos implementations should gain support for a DES TGT key that is used for v4 requests but not v5 requests.

4) Remove Triple DES Kerberos 4 Support

The cut and paste attack is a critical failure in MIT's attempt at Kerberos 4 Triple DES. Even without cross-realm authentication, this can be exploited in real-world situations. As such, the support for 3DES service keys should be disabled.

Obtaining Patches:

The following Kerberos implementations have provided statements on how

Securiteam: [UNIX] Vulnerabilities in the Kerberos Version 4 Protocol

vendors should obtain patches:

MIT: Patches are available for the 1.2.x and 1.3 versions of MIT Kerberos 5. Contact <mailto:hartmans@mit.edu> Sam Hartman or <mailto:tlyu@mit.edu> Tom Yu for patches.

Acknowledgements:

This description was written by Sam hartman. The attached paper detailing the cryptographic details of the attack was written by Tom Yu. The exploit was written by Ken Raeburn and Sam Hartman. All parties were involved in discovering the vulnerabilities.

*) Paper by Tom on weaknesses

ADAPTIVE CHOSEN-PLAINTEXT ATTACKS AGAINST KERBEROS 4

Introduction

The Kerberos 4 protocol is vulnerable to a number of adaptive chosen-plaintext attacks. Some of these attacks may be used, indirectly, against Kerberos 5 as well, and an attack exists that can generate arbitrary tickets in any realm that either directly or transitively shares a key with a realm controlled by an adversary.

These attacks have their basis in the ability of an adversary to create an Electronic Code Book (ECB) oracle for a block cipher with a fixed unknown key, given:

- * The ability to choose a single block of a larger plaintext for a victim to encrypt with that key using the block cipher in a chaining mode
- * The output ciphertext block of the chaining-mode encryption corresponding to the chosen plaintext
- * Prior knowledge of the feedback block that will be XORed with the chosen plaintext block prior to ECB-encryption

The existence of this oracle permits the attacker, without knowledge of the key, to construct a ciphertext that will decrypt with a chained block cipher to any desired plaintext (with the possible exception of the first block). This constructed ciphertext can be the entirety of an arbitrary Kerberos 4 ticket. Kerberos 4 is vulnerable to this attack due to the lack of random confounders, the lack of random initialization vectors, and the lack of cryptographically strong integrity checking in its use of block ciphers. Kerberos 5 is somewhat less vulnerable, but a related weakness permits a cross-protocol attack from Kerberos 4 to Kerberos 5 when single-DES is used to encrypt tickets. In addition, these vulnerabilities render any realm that either directly or transitively shares a key with an adversary-controlled realm vulnerable to compromise.

Definitions and notation

[heavily adapted from B. Schneier, *Applied Cryptography*, 2nd ed., John Wiley & Sons, Inc., 1996, chapter 9]

$$A \wedge B = A \text{ XOR } B$$

$C[n]$ = nth ciphertext block

$P[n]$ = nth plaintext block

$E(k, x)$ = x ECB–encrypted with key k

$D(k, x)$ = x ECB–decrypted with key k

Cipher Block Chaining (CBC) mode is used in the implementation of triple–DES in krb4 that is used in the MIT krb5 release:

$$C[n] = E(k, P[n] \wedge C[n-1])$$

$$P[n] = D(k, C[n]) \wedge C[n-1]$$

Propagating Cipher Block Chaining (PCBC) mode is used for single–DES encryption in krb4:

$$C[n] = E(k, P[n] \wedge C[n-1] \wedge P[n-1])$$

$$P[n] = D(k, C[n]) \wedge C[n-1] \wedge P[n-1]$$

It is useful to generalize these block cipher–chaining modes by considering the block that is to be ECB–encrypted as being a plaintext block XORed with a feedback block:

$F[n]$ = nth feedback block (not transmitted)

$$C[n] = E(k, P[n] \wedge F[n])$$

$$P[n] = D(k, C[n]) \wedge F[n]$$

$F[0]$ = initialization vector (IV)

For CBC:

$$F[n] = C[n-1]$$

For PCBC:

$$F[n] = C[n-1] \wedge P[n-1]$$

There are other, more obscure modes, e.g. Block Chaining (BC) mode:

$$F[n] = F[n-1] \wedge C[n-1]$$

However, this document will not discuss them further, since the krb4 protocol does not use them.

General ECB oracle attack on chained block ciphers

An ECB oracle for a block cipher can be constructed from a block–chaining mode of that cipher if an attacker has access to the ciphertext block corresponding to a chosen plaintext block of a larger plaintext, under certain conditions.

To learn $E(k, x)$ for arbitrary x , note that

$$C[n] = E(k, P[n] \wedge F[n])$$

In the generalized form for chained block ciphers. Let

$$x = P[n] \wedge F[n] ,$$

And assume that the attacker has prior knowledge of the block $F[n]$ that will be used when the victim encrypts the complete plaintext containing the chosen plaintext block $P[n]$. The attacker may then manipulate the chosen plaintext block $P[n]$ to produce

$$C[n] = E(k, x)$$

By choosing

$$P[n] = x \wedge F[n] ,$$

Because the XOR operation is its own inverse. The attacker now has an oracle that can ECB-encrypt arbitrary plaintext using the key k , without knowing the key. The complete set of requirements for the attacker to construct this ECB oracle is:

- * The ability to choose a plaintext block $P[n]$ as part of a (possibly) larger plaintext to be chaining-mode encrypted with the fixed key k
- * Prior knowledge of the feedback block $F[n]$ that will be XORed with $P[n]$ when this chaining-mode encryption occurs
- * Knowledge of the ciphertext block $C[n]$ that results from the chaining-mode encryption

In a well-designed cryptosystem, an attacker should have significant difficulty assembling such an oracle, since it should be difficult to determine $F[n]$ ahead of time. If $F[n]$ is known to be constant between two instances of the chaining-mode encryption, an attacker may rather easily construct the ECB oracle.

As an example of how to mount this attack, assume that the attacker has access to the output of a black box that produces ciphertext that is encrypted in a chaining mode with a fixed key k . Additionally, assume that the attacker can vary a constrained subset of inputs to the black box such that the only resulting changes in the ciphertext, up to and including $C[n]$, are in $C[n]$ itself, and that the feedback block $F[n]$ associated with $C[n]$ does not change. The ciphertext following $C[n]$ is not important. If the attacker can manipulate the constrained subset of the black box's inputs in a way that completely controls the contents of the plaintext block $P[n]$ that will encrypt to $C[n]$, then the attacker has an ECB oracle for the fixed key k .

Note that the attacker does not need to control the entire plaintext that the black box encrypts; it is sufficient to manipulate one aligned

block of plaintext. Also, note that the condition of fixed output ciphertext up through $C[n]$ is slightly less restrictive in the general case; the attacker only needs to ensure that $F[n]$ does not vary with manipulation of the constrained subset of inputs to the black box, though this usually implies that the output ciphertext up to $C[n]$ is also fixed.

Using an ECB oracle to construct arbitrary ciphertext

To construct a complete ciphertext that a chained block cipher will decrypt to a desired plaintext, it is necessary to proceed one block at a time, taking into account the feedback block each time. Since

$$C[n] = E(k, P[n] \oplus F[n]),$$

It is only necessary to take the desired plaintext block $P[n]$, XOR it with the feedback block $F[n]$, and encrypt it using the ECB oracle. The construction of $C[0]$ for a desired $P[0]$ using the ECB oracle is only easily possible if the initialization vector $F[0]$ that will be used for decryption is known. If $F[0]$ is not known, it will be necessary to manipulate $P[0]$ to be encrypted in the chaining mode by k using the unknown value of $F[0]$, which is not in general possible. In general, the attacker will know $F[n]$ for non-zero values of n because $F[n]$ will depend solely on the previously constructed blocks of ciphertext (or, additionally, the previous plaintext block in the case of PCBC).

In the specific case of single-DES used in PCBC mode in krb4, $F[0]$ is the key k , and is not known to an attacker, even if the ECB oracle can be obtained. To produce a $C[0]$ that decrypts to a desired value, the attacker must be able to manipulate the $P[0]$ that will be encrypted by k .

In the case of triple-DES used in CBC mode used in recent krb4 implementations in the krb5 sources, $F[0]$ is an all-zeros IV, so the ECB oracle can be used to produce a $C[0]$ that decrypts to an arbitrary $P[0]$ even if $P[0]$ cannot be directly manipulated to be encrypted with k in CBC mode by the attacker.

Building an ECB oracle using krb4 tickets

A krb4 ticket is encrypted in the long-term key of the service. The plaintext contents of the ticket are:

- unsigned char flags namely, HOST_BYTE_ORDER
- string pname client's name
- string pinstance client's instance
- string prealm client's realm
- 4 bytes paddress client's address
- 8 bytes session session key
- 1 byte life ticket lifetime
- 4 bytes time_sec KDC timestamp
- string sname service's name
- string sinstance service's instance
- <=7 bytes null null pad to 8 byte multiple

Securiteam: [UNIX] Vulnerabilities in the Kerberos Version 4 Protocol

The fields labeled "string" are NUL-terminated ASCII strings, each limited to 40 characters (including the terminal NUL).

The following example of ECB oracle construction assumes that the attacker varies $P[1]$ to obtain the ECB-encryption of a desired plaintext in $C[1]$. It may be generalized to use manipulation of an arbitrary $P[n]$, though.

Assume that the attacker controls a realm "HAX0R.EXAMPLE", the attacked realm is "VICTIM.EXAMPLE", and that they share a key. Since the realms "HAX0R.EXAMPLE" and "VICTIM.EXAMPLE" share a key, the attacker knows the key for the principal

```
"krbtgt.HAX0R.EXAMPLE@VICTIM.EXAMPLE"
```

(Which has the same key as "krbtgt.VICTIM.EXAMPLE@HAX0R.EXAMPLE" in krb4). The attacker can fabricate a cross-realm ticket for the client principal

```
"a234567XXXXXXXXX@HAX0R.EXAMPLE" ,
```

Where "a234567" is a constant string and "XXXXXXXXX" is chosen to produce the ECB encryption of an arbitrary plaintext by the key to be attacked. The service principal of this magic ticket will of course be for the service principal

```
"krbtgt.VICTIM.EXAMPLE@HAX0R.EXAMPLE" ,
```

Since it is a cross-realm TGT.

The attacker then uses the cross-realm ticket to obtain a service ticket for the key that will be attacked, for example,

```
"krbtgt.VICTIM.EXAMPLE@VICTIM.EXAMPLE"
```

(for maximum damage). The attacked KDC (which acts as the black box assumed in the section "General ECB oracle attack") encrypts the client principal's name, instance, and realm from the cross-realm TGT with the attacked service's key, thus providing ciphertext corresponding to a partially known plaintext. Note that the session key, timestamp, etc. may not be known to the attacker before encryption, but they do not affect the ciphertext blocks of interest.

The fixed string "a234567", appended to the one-byte flags field, becomes the first plaintext block $P[0]$. This means that chosen plaintext string "XXXXXXXXX" is the entirety of the second plaintext block $P[1]$. $C[0]$ will be a fixed value, given the fixed string, since the flags don't change, the first 7 characters of the principal name don't change, the IV and key don't change, and there is no prepended confounder (unlike in krb5 uses of block-chaining modes). Likewise, $F[1]$ won't change, since it is based on fixed values ($C[0]$, which is additionally XORed with $P[0]$ in PCBC mode). The attacker merely has to change the variable string "XXXXXXXXX", which is $P[1]$, to be the result of $F[1]$ XORed with the desired plaintext to be

ECB-encrypted. The C[1] in the resulting ticket will then be the ECB encryption of the desired plaintext. The attacker can obtain the initial fixed value of F[1] by performing one pass with the chosen P[0] and discarding C[1].

If any P[1] obtained by XORing F[1] with the desired plaintext contains one NUL character, the value of P[1] will be split between the "pname" and "pinstance" fields, rather than being completely contained within the "pname" field. This is still acceptable. If there are two or more NUL characters in the resulting P[1], the KDC will probably not create a useful ticket, so the previously fixed string P[0] must be permuted to obtain a F[1] value that does not result in an unacceptable number of NUL characters in P[1]. It may also be possible to permute P[0] such that P[1] will not need to contain any "suspicious" characters, e.g. non-alphanumeric, in order to obtain a desired C[1].

Note that almost none of this attack is specific to the particular block-chaining mode (CBC or PCBC), if the mode is of the general form described under "Definitions".

Note also that it is not necessary to control a cross-realm key with the realm "VICTIM.EXAMPLE"; it is merely sufficient to be able to create or control a large number of principals in "VICTIM.EXAMPLE" and to know their keys. Having control of a cross-realm key merely makes the attack much easier.

Even that level of control may not be required, if the attacker has control of a key for a principal whose name is of the correct length to cause the "paddress" field to begin on a block boundary. Then, assuming that the attacker can receive arbitrary UDP packets originating from the attacked KDC and that are addressed to arbitrary IP addresses, the attacker can effectively vary the "paddress" field, by sending from arbitrary source IP addresses, in order to obtain control over 4 bytes of plaintext. Note that this increases the work factor to $O(2^{32})$ per ciphertext block, since the immediately following 4 bytes are the first half of the (presumed) random session key, which the attacker can obtain only after the KDC issues the ticket. Obviously, controlling smaller amounts of address space will result in a proportionally increased work factor, but it can still be significantly below $O(2^{56})$. The downside to this attack is that the large volume of ticket requests required may be quite noticeable to a KDC administrator.

Most of the "life" field and some of the "time_sec" field are also under the control of an attacker, and can be used as chosen plaintext with which to mount the ECB oracle attack, though this method may be somewhat more difficult, and suffers from the same drawbacks as the address space attack.

Constructing an arbitrary krb4 ticket using the ECB oracle
Given the ECB oracle, it is possible to construct an arbitrary krb4 ticket, subject to certain conditions. The attacker must usually build up

the ciphertext of the fabricated ticket one block at a time, since the feedback blocks are required; there may be optimizations that permit multiple ciphertext blocks to be produced and spliced in at once, but the length constraints inherent in krb4 principal names, in addition to the NUL-terminated string condition, limit these optimizations somewhat.

An arbitrary block of ciphertext $C[n]$ corresponding to a chosen plaintext $P[n]$ may be produced as described above in "General ECB oracle attack". Production of the first ciphertext block, $C[0]$, may be difficult if $F[0]$ is not known. This is the case in krb4 single-DES in PCBC mode, since the key is used as the initialization vector $F[0]$. In the krb4 usage of triple-DES CBC mode, the IV is all zeros, so it is not necessary to force encryption of a chosen $P[0]$ in order to obtain a desired $C[0]$.

Since, in the krb4 usage of single-DES PCBC, $C[0]$ will be constant for a constant $P[0]$, it is sufficient to somehow obtain a $C[0]$ corresponding to the desired $P[0]$. This may be trivially done by controlling a realm with a shared key, since the $C[0]$ may be obtained directly in that case. Consider an attacker who wants to fabricate a ticket for

"somaluser@VICTIM.EXAMPLE" .

The attacker may simply synthesize a cross-realm ticket for the client "somelu@HAX0R.EXAMPLE" ,

Since the resulting $P[0]$,

"\000somelu"

Ends immediately prior to the terminating NUL of the "somelu" principal name string. Upon using the cross-realm ticket to obtain a service ticket in the "VICTIM.EXAMPLE" realm, the first block of the ticket will be exactly the $C[0]$ required to synthesize the remainder of the ticket, even though $F[0]$ is unknown. Note that fabricating a ticket in this manner for a client whose total length of principal name and instance (including terminating NUL characters) is less than 7 bytes is probably impossible unless the attacker's realm name and the victim's realm name share an initial substring. This constraint is a consequence of the requirement that the client realm name in the ticket must match the issuing realm of the ticket.

Without access to a realm with a shared key, it suffices to sniff a $C[0]$ corresponding to a desired $P[0]$ off the network. This may be difficult if the client principal(s) being attacked are careful not to obtain or use krb4 tickets for authentication. There are other ways to obtain the desired initial ciphertext block $C[0]$, as described in later sections.

Cross-protocol attack from krb4 to krb5

If a service is authenticated with krb5, accepts tickets encrypted with single-DES, and does not necessarily accept krb4 authentication, it is still possible to mount an attack against it using the krb4 ECB oracle if

the KDC for the service's realm can issue krb4 tickets for the service. The format of a krb5 single-DES plaintext immediately prior to encryption is

concat(confounder, checksum, data, pad) .

The confounder is one block of random bytes. The checksum may be CRC-32, MD4, or MD5. None of these checksums is keyed, which is the crucial feature that enables this attack. For des-cbc-md4 and des-cbc-md5, the IV is all zeros. For des-cbc-crc, the IV is the key. The checksum is computed over the entire concatenated plaintext, with the checksum field zeroed out. This frustrates ciphertext generation via chosen-plaintext manipulation if the attacker has no knowledge of the confounder; an adversary will find it very difficult, if not impossible, to create an ECB oracle based on manipulating krb5 tickets.

For des-cbc-md4 and des-cbc-md5, an attacker can simply fabricate any desired confounder using the ECB oracle, since the IV is known and constant. For des-cbc-crc, the attack is somewhat more difficult, since the IV is the key and not known to the attacker. For this particular attack, it is necessary to obtain the ciphertext block C[0] corresponding to the desired confounder, possibly by the same method that can be used for the C[0] attack on PCBC krb4, i.e. by forcing the KDC to encrypt an initial P[0] with the attacked key (which also requires control of a realm with a shared key).

An easier method for obtaining C[0] for attacking krb5 des-cbc-crc is to sniff, or otherwise acquire, any krb4 ticket encrypted in the attacked key, provided that the first block of plaintext is known. The first ciphertext block of both the krb5 des-cbc-crc encryption and the krb4 PCBC encryption of the same plaintext will be identical, as they both use the same key and an IV equal to the key.

Incidentally, if krb524d is running on the attacked KDC, the cross-protocol attack from krb4 to krb5 may be extended back into an attack on single-DES krb4, circumventing the problems of constructing C[0] for the krb4 ticket, especially the difficulties inherent in fabricating a ticket for a client with a short principal name. The attacker acquires C[0] of a krb4 ticket encrypted for the target service, provided that the corresponding P[0] is known. Note that this is trivial if the attacker knows *_any_* client principal key in the attacked realm. The attacker can then use this C[0] as the encrypted confounder for a krb5 des-cbc-crc ticket, and proceed to construct a krb5 ticket for that service, complete with correct checksum. Then, the attacker merely submits the fabricated krb5 ticket to krb524d, which converts it into a valid krb4 ticket.

This weakness of the single-DES krb5 cryptosystems results largely from the practice of encrypting plaintext checksums. Without knowing the key, an attacker can still fabricate ciphertext that decrypts and appears to have valid integrity. [S. M. Bellare, personal communication, 1999] This would not be possible if a keyed checksum or a HMAC were used. Note,

Securiteam: [UNIX] Vulnerabilities in the Kerberos Version 4 Protocol

however, that checksumming of the random confounder along with the data prevents several types of chosen–plaintext cut–and–paste attacks.

Attack on transitive closure of trust

Normally, cross–realm trust in krb4 is not transitive, because the code in the KDC implementation forbids realm hopping by use of cross–realm tickets. This can be circumvented by an adversary, though. The nature of cross–realm trust in krb4 means that any remote realm, if it can synthesize tickets for services in the local realm, can synthesize tickets for services in any other realm sharing keys with the local realm. Consider a local target realm "VICTIM.EXAMPLE", which shares keys with an attacker's realm "HAX0R.EXAMPLE" and with another target realm "OWNZ0RED.EXAMPLE". The attacker can not only synthesize tickets for clients in "VICTIM.EXAMPLE" for services in "VICTIM.EXAMPLE", but can also synthesize a ticket for an arbitrary client in the "VICTIM.EXAMPLE" realm for the service

"krbtgt.OWNZ0RED.EXAMPLE@VICTIM.EXAMPLE" .

This ability to synthesize arbitrary cross–realm tickets between "VICTIM.EXAMPLE" and "OWNZ0RED.EXAMPLE" means that the attacker can synthesize a ticket for a client in "OWNZ0RED.EXAMPLE" for any service in "OWNZ0RED.EXAMPLE" by using the ECB oracle attack against the service key. The attacker may recourse this attack indefinitely, compromising the entire web of trust. Combined with the krb4–to–krb5 cross–protocol attack, the results may be devastating.

Conclusions

The Kerberos 4 protocol has weaknesses in its uses of cryptography that permit an attacker to mount an adaptive chosen–plaintext attack to obtain an ECB oracle. This oracle can then be used to fabricate arbitrary tickets in any realm that shares a key with one under the attacker's control. This attack has some limitations, but the combination of cross–protocol attack to the Kerberos 5 protocol with the related cross–protocol attack back to Kerberos 4 permits the attacker to circumvent most of these limitations. Additionally, these fabricated tickets permit the attacker to extend the attack to all transitively trusted realms, in spite of the normal lack of cross–realm trust transitivity in the Kerberos 4 protocol. Combined, these attacks can have a profound detrimental effect on a set of realms that trust each other.

ADDITIONAL INFORMATION

The information has been provided by <<mailto:hack4life@hushmail.com>> hack4life, <<mailto:hartmans@mit.edu>> Sam Hartman and <<mailto:tlyu@mit.edu>> Tom Yu.

=====

This bulletin is sent to members of the SecuriTeam mailing list.
To unsubscribe from the list, send mail with an empty subject line and body to:

Securiteam: [UNIX] Vulnerabilities in the Kerberos Version 4 Protocol

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====
=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.