The Perils of Unauthenticated Encryption: Kerberos Version 4*

Tom Yu Sam Hartman Kenneth Raeburn
Massachusetts Institute of Technology

Abstract

Version 4 of the widely deployed Kerberos authentication protocol encrypts essential information without adequate authentication. We have implemented an efficient chosen-plaintext attack that uses this design flaw to impersonate arbitrary principals. Related flaws exist in version 5 of the protocol. We discuss the mistakes in the design of the protocol that contribute to these vulnerabilities, and how to avoid making them. We identify corrective measures taken in the proposed revisions to version 5, which repair these flaws.

1. Introduction

The dangers of unauthenticated encryption are well known [6, 7, 8, 13, 20, 22, 39]. Although most cryptographic attacks focus on recovering a plaintext or a key, a more powerful attack is to forge a ciphertext that decrypts to a desired plaintext, particularly when attacking an authentication system. This sort of forgery is often far more dangerous than a breach of confidentiality; it is far more useful to *become* someone than to merely know what someone said once.

Consider a transaction in a hypothetical banking protocol in which Alice instructs her bank to send $100 to Bob. An eavesdropper Eve will probably not be that interested in reading such a message. On the other hand, Eve will probably find it much more useful to modify the message so that Alice appears to have instructed the bank to send $100 to Eve. Even more devastating is for Eve to have the capability to impersonate Alice, so that Eve need not modify an existing message that Alice sends. Authentication is usually more important than confidentiality.

Kerberos version 4 [28, 37] has a critical authentication vulnerability which allows an attacker to impersonate arbitrary principals. This vulnerability results from multiple design errors. Additional flaws in MIT's implementation of version 4 enable additional attacks. The current specification of Kerberos version 5, Internet RFC 1510 [23], fixes some flaws in version 4, though it too has some vulnerabilities. Ongoing work on the specification of version 5 repairs even those flaws. Despite the progress made in updating the Kerberos protocol, version 4 remains in widespread use, and that fact illustrates that protocols have a longer life than their designers might anticipate.

Kerberos version 4 uses unauthenticated encryption for essential authentication information. This allows an attacker to forge credentials impersonating arbitrary principals by using an adaptive chosen-plaintext attack as an encryption oracle. We have successfully implemented a startlingly efficient attack based on this oracle: $O(n)$ oracle queries are needed to forge a credential ciphertext n blocks long. The attack is sufficiently inexpensive that its successful execution may go completely undetected.

The use of unauthenticated encryption is one of several design errors in version 4. Other mistakes include a failure to explicitly identify the role of encryption and a failure to sufficiently abstract the encryption scheme. Though RFC 1510 corrects some of the mistakes of version 4, it too has deficiencies. The MIT implementation's use of identical keys between versions 4 and 5 allows a cross-protocol attack, which we describe later. Ongoing revisions to version 5 remedy the existing flaws in RFC 1510, and implement current best practices in authenticated encryption.

Protocol designers should clearly identify the role of encryption in their protocols. The designers of Kerberos version 4 failed to explicitly identify an important assumption: nonmalleability. The unauthenticated encryption used in version 4 constitutes a lack of nonmalleability, leading to the attack we describe. If a protocol requires nonmalleable encryption, which is more likely than not, clearly state so in the specification, and ensure that the encryption scheme actually achieves this requirement.

Designers should sufficiently abstract the use of encryption from the contents of the protocol messages, so that the exact form and layout of the protocol messages do not compromise the efficacy of the encryption. The designers of version 4 assumed that the message layout itself would enhance the security properties of the encryption scheme. The resulting confusion creates cross-dependencies between details of protocol layout and the security of the encryption scheme, further enabling the

*An unauthorized copy of an earlier version of this paper appeared on `full-disclosure@lists.netsys.com` in March 2003.

attack that we describe.

One reason why our attack succeeds is because the encryption scheme in Kerberos version 4 is deterministic. An attacker can use a predictable mapping from plaintext to ciphertext to assemble an efficient encryption oracle, which can then be used to forge new ciphertexts. Kerberos version 5 thwarts the creation of an encryption oracle.

Designers should ensure that attackers cannot use ciphertext surgery to subvert message authentication schemes in a protocol. Though designers should not allow an attacker to implement an encryption oracle at all, they should also ensure that an attacker cannot use an encryption oracle to attack the message authentication schemes. As we show later, encrypted plaintext checksums, specified in RFC 1510, can be subverted. The proposed revisions to the specification of version 5 remedy this flaw by improving the authentication scheme for encrypted messages.

Despite its known shortcomings, Kerberos version 4 remained in widespread use until the discovery of this vulnerability. The inherent difficulties in deploying a security architecture appear to create resistance to change unless a clear and present danger exists. Security protocols could therefore benefit from conservative design, and users would also do well to evaluate whether apparently theoretical vulnerabilities may be indicative of more serious problems.

2. Kerberos history

Kerberos is a trusted-third-party authentication protocol based on the Needham-Schroeder symmetric-key protocol [33], modified to use timestamps [18] to mitigate replay attacks. Kerberos originated in MIT's Project Athena, where version 4 [28, 37] was deployed for quite some time. The current specification of Kerberos version 5, RFC 1510 [23], corrects many known flaws in version 4, and is undergoing standardization in the Internet Engineering Task Force (IETF). Proposed revisions to version 5 [34, 36] remedy vulnerabilities present in RFC 1510.

Kerberos version 4 was widely deployed; the use of Kerberos version 4 as the authentication infrastructure of CMU's Andrew File System (AFS) led to the widespread adoption of Kerberos at large universities and corporations by virtue of their adoption of AFS. Although AFS was commercialized by Transarc and IBM, release of an open-source version, OpenAFS, led to even wider adoption. Difficulties inherent in the design of the AFS protocol discouraged the implementation of Kerberos version 5 support in AFS. Our discovery of the chosen-plaintext attack on version 4 finally prompted rapid implementation of version 5 support in AFS. Similar scenarios have doubtless occurred with other protocols which have been built on top of Kerberos version 4.

Some researchers have performed formal analyses [2, 3, 14] regarding the correctness and secrecy of Kerberos version 4, but they make assumptions (often implicit) about its use of encryption, and largely ignore possible vulnerabilities in the actual Kerberos encryption scheme. Although others have noted shortcomings in the Kerberos encryption scheme [12, 38], none are quite as serious as the vulnerabilities we have discovered. Lowe [27] demonstrates some related encryption-oracle attacks against several other protocols, but unlike Lowe's attacks, ours involves the blockwise forgery of complete ciphertexts.

Kerberos continues to evolve. RFC 1510 fixes some of the flaws in version 4, and ongoing work in the IETF to revise the specification of version 5 and its encryption schemes [34, 36] further repair cryptographic flaws. The strategies used in these revisions are similar to those proposed in recent work [6] on the Secure Shell (SSH) protocol.

3. Design shortcomings of Kerberos

Kerberos version 4 has some serious shortcomings in its abstract design principles, in addition to fatal problems in its concrete design choices. Some of the concrete design choices may have seemed justifiable at the time they were made, however, they are hardly excusable today.

Demillo and Merritt [17] advise explicitly stating cryptographic assumptions during the design of a protocol. Abadi and Needham [1] echo this sentiment, pointing out some apparent confusion in the Kerberos protocol's use of encryption. The designers of version 4 clearly did not make all their cryptographic assumptions explicit; among other things, they unwittingly inherited the assumption of nonmalleable cryptography from the underlying Needham-Schroeder protocol.

Dolev, Dwork, and Naor [19] observe that the original Needham-Schroeder protocol implicitly requires nonmalleable encryption. Roughly, nonmalleable encryption prevents an attacker, given a ciphertext, from producing a different ciphertext whose plaintext is meaningfully related to the plaintext of the original ciphertext. This is precisely the guarantee that Kerberos version 4 fails to provide, by inadequately authenticating its messages. The designers did not consider nonmalleability, probably because the concept was not well-developed then.

The specific design choices in version 4 are not much better. The protocol uses the Data Encryption Standard (DES) [31], now known to have an insufficient key size, in the nonstandard *propagating cipher block chaining* (PCBC) mode. This cipher mode is slightly different from the more usual *cipher block chaining* (CBC) [30]

mode. Although DES was the standard at the time, the use of a non-standard mode of operation seems unwise, particularly in hindsight: PCBC has some odd properties, such as ciphertext block swaps being undetectable without additional integrity checking.

The design document for version 4 [28] implicitly (and incorrectly) assumes that the error propagation properties of PCBC mode will cause ciphertext errors to scramble the plaintext following the manipulated ciphertext, and even suggests that certain fields near the end of a ticket, *e.g.*, principal names and timestamps, be used for integrity checking. While it is *usually* true that modification of a PCBC ciphertext propagates errors through the remainder of the decrypted plaintext, this assumption only holds for random transmission errors. The willful manipulations of an attacker with an encryption oracle hardly constitute random transmission errors. Relying on PCBC mode itself to ensure integrity against active attackers utterly fails.

The dependency between integrity and message layout in Kerberos version 4 betrays a lack of sufficient abstraction between the encryption scheme and the rest of the protocol, leading to the sort of subtle vulnerabilities that we have discovered. Bellovin and Merritt [12] emphasize the importance of applying this separation to future revisions to the Kerberos protocol. The security of the encryption should not depend on the details of the packet layout of the protocol.

The security assumptions of version 4 do not include an attacker with an encryption oracle. Many security analyses applied to version 4 assume an attacker capable of intercepting messages, but fail to include chosen-plaintext attacks. Voydock and Kent [40] identify a chosen-plaintext attack against CBC mode with a fixed initialization vector (IV), of which the designers of version 4 appear to have been unaware. The designers of version 5 [24] came very close to identifying the attack we describe, noting the possible consequences of chosen-plaintext attacks in version 4. These concerns were dismissed with the argument that in version 4, the first ciphertext block contains a random session key, frustrating a chosen-plaintext attack. This is true for the KDC's encrypted reply to the client [15], but not for the ticket itself [16], which is more interesting to attack. Once again, this sort of logic reveals an unhealthy lack of abstraction of the encryption scheme.

4. Kerberos version 4 (simplified)

A simplified description of version 4 of the protocol follows. The trusted third party in Kerberos is called the Key Distribution Center (KDC). Fundamental to Kerberos is the concept of a *ticket*. A ticket issued to a client A for a service B is encrypted in the long-term key k_b shared between the KDC and the service B. The ticket contains information identifying the client A, the service B, as well as a session key k_{ab}. A ticket alone is not sufficient to authenticate to a service; proof of knowledge of the session key is also required. The ticket, along with the associated session key, constitute a *credential*.

4.1. Kerberos protocol exchanges

There are two conceptual services embodied in the KDC. One of these services is the *authentication service* (AS); the other is the *ticket granting service* (TGS). The AS exchange is used to obtain credentials encrypted with the client's long-term key k_a, while the TGS exchange treats the TGS itself as a special service used for obtaining additional credentials. The TGS enables a client to avoid typing a password for each credential obtained. A typical Kerberos login session begins with the client using the AS exchange to obtain credentials for the TGS:[1]

$$\begin{aligned} A \rightarrow S &: A, S \\ S \rightarrow A &: \{k_{as}, S, \{A, S, t_s, k_{as}\}k_s\}k_a, \end{aligned}$$

where t_s is the KDC's timestamp and $\{M\}k_x$ denotes M encrypted with the key k_x. A is the client's name, and S is the name of the TGS. For clarity, we omit some additional information included in the tickets, such as lifetimes and network addresses. The ticket is

$$\{A, S, t_s, k_{as}\}k_s.$$

k_{as} is the session key shared between the client A and the TGS.

The client may then use this *ticket-granting ticket* (TGT) to obtain credentials for other services. The TGS exchange for the client A obtaining credentials for the service B is

$$\begin{aligned} A \rightarrow S &: \{A, S, t_s, k_{as}\}k_s, \{A, t_a\}k_{as}, B \\ S \rightarrow A &: \{k_{ab}, B, \{A, B, t_s, k_{ab}\}k_b\}k_{as}, \end{aligned}$$

where

$$\{A, t_a\}k_{as}$$

is the *authenticator*, and t_a is the client's timestamp. The authenticator assures the TGS that the client has recent knowledge of the session key k_{as}, since values of t_a too far removed from the current time will cause the TGS to reject the authenticator. The entire scheme relies upon close synchronization of clocks; we will not further discuss that issue here.

[1] The superfluous superencryption of the ticket, both here and in the TGS exchange, is only present in version 4; it has been eliminated in version 5, at the expense of introducing a delayed denial-of-service attack. Ongoing work on the specification of version 5 is investigating countermeasures for this denial of service.

To actually use the service ticket

$$\{A, B, t_s, k_{ab}\}k_b,$$

the client sends the message

$$A \rightarrow B : \{A, B, t_s, k_{ab}\}k_b, \{A, t_a\}k_{ab}.$$

As in the TGS exchange, the authenticator

$$\{A, t_a\}k_{ab}$$

proves to the service B that the client has recent knowledge of the session key k_{ab}.

4.2. Kerberos names

The structure of names in Kerberos version 4 has some implications for the execution of the attack, as we show later. A version 4 principal name is a three-tuple

$$\{primaryname, instance, realm\}.$$

The usual way of displaying principal names to users is "*primaryname*`.`*instance*`@`*realm*". The period between the instance and primary name is usually omitted if the instance is empty. The *realm* indicates which authentication domain the principal belongs to; generally, one KDC (or replicated set of KDCs) functions to authenticate principals of only one realm to each other. In order to authenticate principals of one realm to principals of another realm, shared keys must be established between the two realms. These are in the form of specialized TGS principals.

The normal TGS principal name for a realm is "`krbtgt.`*realm*`@`*realm*". A TGS principal name used for cross-realm authentication takes a different form: the TGS principal name, as received from a foreign principal by the local realm's KDC, is "`krbtgt.`*localrealm* `@`*foreignrealm*". The local realm KDC ensures that each cross-realm TGT originating from a foreign realm has its client principal name's *realm* component matching the name of the foreign realm. This normally ensures that the local realm's KDC cannot be tricked by foreign realm's KDC into issuing a ticket for an incorrect client principal realm. As we demonstrate later, design flaws in the Kerberos version 4 encryption scheme allow for this measure to be circumvented.

As originally designed in version 4, sharing a cross-realm key with a foreign realm only implies a trust that the foreign realm will truthfully authenticate principals in its realm. There was no reason that compromise of a foreign realm could compromise principals in the local realm, so realm administrators tended to freely exchange cross-realm keys. Compromise of a foreign realm's KDC would only render principals in the foreign realm untrustworthy; administrators in the local realm can limit the damage from an untrustworthy foreign realm merely by refusing to grant any privileges to principals from that realm. As we demonstrate, cryptographic flaws in version 4 invalidate these trust assumptions.

5. A block-encryption oracle

An essential component of the attack on Kerberos version 4 is a block-encryption oracle based on chosen plaintext. The oracle takes advantage of the structure of the CBC or the PCBC mode. CBC mode is defined as

$$\begin{aligned} C_{i+1} &= k(P_{i+1} \oplus C_i) \\ P_{i+1} &= k^{-1}(C_{i+1}) \oplus C_i, \end{aligned}$$

where $C_0, C_1, \ldots, C_n$ are the ciphertext blocks; $P_0, P_1, \ldots, P_n$ are the plaintext blocks; $k(x)$ denotes the encryption of the block x with the key k; $k^{-1}(x)$ denotes decryption of the block x with the key k; and $x \oplus y$ denotes the bitwise exclusive-OR of x with y.

PCBC mode is defined as

$$\begin{aligned} C_{i+1} &= k(P_{i+1} \oplus C_i \oplus P_i) \\ P_{i+1} &= k^{-1}(C_{i+1}) \oplus C_i \oplus P_i. \end{aligned}$$

It is useful to generalize these cipher modes as block-feedback modes by considering the input block of the block cipher to consist of the exclusive-OR of a plaintext block with a feedback block:

$$\begin{aligned} C_i &= k(P_i \oplus F_i) \\ P_i &= k^{-1}(P_i) \oplus F_i, \end{aligned}$$

where F_i represents the i-th feedback block, which is not necessarily transmitted. For CBC mode, $F_{i+1} = C_i$, while for PCBC mode, $F_{i+1} = C_i \oplus P_i$. For these cipher modes, F_0 is known as the *initialization vector* (IV).

Assume that an attacker can cause the production of some ciphertext $C_0, C_1, \ldots, C_j, \ldots, C_n$ with the contents of the feedback block F_j known in advance and with its corresponding plaintext block P_j chosen by the attacker. To obtain the ciphertext X of an arbitrary plaintext block M under the block cipher, the attacker can choose $P_j = M \oplus F_j$. The resulting ciphertext block is

$$\begin{aligned} C_j &= k(P_j \oplus F_j) \\ &= k(M \oplus F_j \oplus F_j) \\ &= k(M) = X. \end{aligned}$$

In a well-designed protocol, an attacker should have significant difficulty assembling such an oracle, since F_j should not be known in advance. This is not the case in version 4 of the Kerberos protocol.

6. Constructing the desired ciphertext

To construct a complete ciphertext that will decrypt to a desired plaintext, it is necessary to proceed one block at a time, taking into account the feedback block of the constructed ciphertext each time. If the desired ciphertext consists of the blocks $\{X_i\}$, and the desired plaintext consists of the blocks $\{M_i\}$, then the relation

$$X_i = k(M_i \oplus \Phi_i),$$

where Φ_i indicates the feedback block related to X_i, may be used to choose plaintext blocks for encryption by the encryption oracle. The only difficulty is if the initialization vector Φ_0 is unknown. In the case of Kerberos version 4, Φ_0 is the actual key, which will not be known to the attacker. There are chosen-ciphertext attacks that can recover the IV from a chaining mode [21], but they are of little practical importance here. To produce a X_0 that decrypts to a desired value M_0, the attacker must be able to directly submit the desired M_0 as the first block of a plaintext to be encrypted under the key k. This does not usually pose a significant difficulty in Kerberos version 4, though it does place some limitations on which client principal names can be forged.

7. Ticket ciphertext as oracle

The predictable plaintext of a version 4 ticket enables an attacker to create a block-encryption oracle by manipulating a portion of the plaintext of a ticket, preferably part of the client principal name, as it occurs earliest in the ticket plaintext. The easiest way to perform this chosen-plaintext attack is by exploiting the cross-realm authentication exchange, as it allows nearly any principal name to be fabricated. It is also possible to implement a chosen-plaintext attack on the AS exchange, which can be done by creating principals in the target realm, among other ways.

The plaintext of a Kerberos version 4 ticket [16] prior to encryption is:

1 byte	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

The fields labeled "string" are NUL-terminated ASCII strings, each limited to 40 characters (including the terminal NUL). The *flags* byte has only one meaningful bit, `HOST_BYTE_ORDER`; this flag indicates the byte order of the KDC issuing the ticket. An attacker can usually predict the value of this bit without difficulty. The following example of encryption oracle construction assumes that the attacker varies P_1 to encrypt a desired plaintext block into C_1. It may be generalized to assume manipulation of an arbitrary P_j, though.

We initially describe the attack based on the TGS exchange, as it is reasonably easy to implement. Assume that the attacker controls a realm `A`, the target realm is `B`, and that these two realms share a cross-realm TGS key. The attacker knows the key for the TGS principal `krbtgt.B@A`, which has the same key as `krbtgt.A@B` in the realm `B`. Using the known cross-realm TGS key, the attacker can fabricate a cross-realm ticket having the unusual-looking client principal

`a234567XXXXXXXX@A`,

where "`a234567`" is an arbitrary string held constant between iterations of the attack, and "`XXXXXXXX`" is a block P_1 chosen to produce the encryption of a desired plaintext block. (The client principal's realm must be `A`, because the target KDC will verify that the requested client principal's realm corresponds to that of the issuing realm.)

The attacker then uses the fabricated cross-realm ticket to perform a TGS exchange with the KDC of `B` to obtain a service ticket for the service being attacked, for example, `krbtgt.B@B` (for maximum damage).[2]

The target KDC, when issuing the service ticket, copies the client principal name from the fabricated cross-realm ticket into the plaintext of the service ticket. It then encrypts the plaintext using the target service's long-term key. This results in a ciphertext whose plaintext is partially controlled by the attacker. Note that while the session key, timestamp, etc., may not be known to the attacker before encryption, these unknown values 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 "`XXXXXXXX`" is the entirety of the second plaintext block P_1. C_0 will be a fixed value, given the fixed string, since the first 7 characters of the principal name, the IV, and the key all remain fixed. Likewise, F_1 will not change, since it is based on fixed values ($F_1 = C_0 \oplus P_0$, assuming PCBC mode). The attacker merely has to choose the variable string "`XXXXXXXX`", which is P_1, to be $M \oplus F_1$, where M is the block that is to be encrypted using the oracle. The C_1 in the resulting ticket will then be the encryption of the desired block. The attacker can obtain the initial fixed value of F_1 by

[2] The ticket-granting principal in the target realm may only be attacked by virtue of an implementation design flaw, described later.

performing one iteration of the attack with the chosen P_0 to acquire C_0.

If a particular P_1 in an iteration of the attack needs to contain one NUL character, the value of P_1 will be split between the *pname* and *pinstance* fields of the ticket, rather than being completely contained within the *pname* field. This does not pose a problem. However, if two or more NUL characters[3] need to appear in P_1, the target KDC will probably not create a useful ticket, so the attacker must change the previously fixed string appearing in P_0 in order to obtain a value of F_1 which does not require more than one NUL character to appear in P_1. Similarly, the attacker may also permute P_0 such that the complete client principal name will not need to contain any "suspicious" characters, *e.g.*, non-alphanumeric characters, though this practice will increase the number of iterations needed to complete the attack.

Almost none of this attack is specific to the particular cipher mode (CBC or PCBC), provided that the mode is of the general block-feedback form. The scenario of a compromised realm sharing a key with the target realm is not farfetched; it is not uncommon for a realm to set up a shared key with a less-secure realm for testing purposes. As mentioned earlier, this practice would not constitute a security exposure if it were not for this cryptographic attack.

The attacker need not control a cross-realm key shared with the target realm; knowledge of a sufficient number of keys in the target realm enables the use of the AS exchange, instead of the TGS exchange, as an oracle. This is particularly easy if the attacker has privileges to adaptively *create* principals in the target realm's database, in which case the required number is not large at all.

For example, the MIT implementation provides a facility for allowing a less-privileged administrator (typically one responsible for deploying server hosts) to create or change a key associated with a host-based service. Typically, site policy limits the principal names to forms such as "`rcmd.`*hostname*`@`*realm*"; the `rcmd` keys are used for authenticating remote logins to a host. If the *hostname* portion of these principals is not restricted to the set of characters typically valid for use as hostnames, this attack may require the creation of as few as n principals to forge a ticket that is n ciphertext blocks long. Tighter constraints, *e.g.*, character set restrictions, on the form of the principal names that the administrator is permitted to create will cause the required number of principals to increase. Regardless, this approach is not useful in a version 4-only environment, as the interesting client principal names to impersonate rarely begin with "`rcmd`"; the attacker must somehow obtain (possibly by packet-sniffing) an initial ciphertext block of a ticket issued to the target client principal. This limitation can be circumvented by the cross-protocol attack, which we describe later.

Controlling a cross-realm key is merely one of the most efficient methods of attack, and it is the only attack variant that we have implemented so far. Our implementation of the attack only requires about one oracle query per forged ciphertext block, needing slightly more queries when too many NUL characters would have to appear in the relevant plaintext block.[4] This sort of attack might go completely unnoticed by all but the most vigilant administrators, allowing the attacker to surreptitiously compromise the realm. As noted above, but not implemented by us, the attacker can utilize a larger number of queries to avoid using principal names containing "suspicious" characters, making the attack even less detectable.

8. Forging arbitrary tickets

Given the encryption oracle, an attacker may construct an arbitrary Kerberos version 4 ticket, subject to certain conditions. The attacker must usually obtain the ciphertext of the counterfeit ticket one block at a time, since the feedback block corresponding to a particular ciphertext block is generally not known in advance. Optimizations may exist which permit multiple ciphertext blocks to be produced and spliced together at once, but the length constraints inherent in version 4 principal names, as well as the use of NUL-terminated strings in the tickets, limit these optimizations somewhat.

To forge a ticket ciphertext $\{X_i : X_i = k(M_i \oplus \Phi_i)\}$ that decrypts to the plaintext $\{M_i\}$, the attacker iterates through each M_i, determining the corresponding Φ_i, and submitting $M_i \oplus \Phi_i$ to the encryption oracle to receive X_i. The use of the key as the IV in Kerberos version 4 introduces a slight difficulty in forging the initial ciphertext block, X_0; the attacker must cause the TGS to encrypt M_0 as the first block of a ticket. The attacker cannot use the oracle to directly produce X_0 because the feedback block Φ_0 is secret.

This property of X_0 places some constraints on the set of client principal names that an attacker may forge. In particular, if the target client principal's *primaryname* and *instance* components total less than 7 bytes, including terminating NUL characters, the target realm must share an initial substring with the attacking realm. This initial substring requirement arises from the TGS's checking for a match between the client realm and the realm issuing the cross-realm ticket. The length limitation on the target principal name may be circumvented by copying the first ciphertext block of a legitimate ticket issued to the target

[3] Actually, if the second NUL character appears at the end of the first block, it is not a problem.

[4] This complication occurs with probability $\sim 2^{-11}$, assuming that all possible feedback block contents are equally likely.

principal, *e.g.*, by sniffing the network while the target principal is using the ticket to authenticate to a service.

9. Implementation flaws

The MIT implementations of Kerberos version 4 have a few design flaws which enable more widespread damage than would be possible if the implementation were better designed. We emphasize that these additional flaws are not flaws in the protocol, but rather are flaws in the implementation. Even the most secure protocol is useless if the implementation is insecure, and we encourage specifiers of cryptographic protocols to identify particular pitfalls that implementors may encounter.

In the MIT implementation of version 4, a poorly designed lineage check for cross-realm authentication allows an attacker to compromise realms other than the initial target realm. Also, the MIT implementation of version 5 uses identical keys for encrypting version 4 and version 5 tickets for the same service principal, enabling a cross-protocol attack against version 5. These flaws are believed to exist in non-MIT implementations of the Kerberos protocols as well.

9.1. Realm hopping

The MIT implementation of the Kerberos version 4 KDC attempts to prevent client principals from transiting or "hopping" between realms, *i.e.*, a client principal accessing a service in a foreign realm must have its local realm directly share a key with the foreign realm. The KDC implementation does not enforce this policy by preventing the issuance of a cross-realm ticket to a foreign principal; it instead relies on the receiving KDC to deny further tickets based on that cross-realm ticket. For example, if a client principal `clienta@A` obtains a ticket for `krbtgt.B@A`, it may subsequently obtain a ticket for `krbtgt.C@B`. The KDC for the `C` realm is responsible for denying the client principal `clienta@A` further tickets in the `C` realm. For similar reasons, it is also possible for that client to obtain a ticket for `krbtgt.B@B`. While the client from `A` may obtain the ticket `krbtgt.B@B`, it will normally be unable to obtain further tickets in `B` using that ticket.

Unfortunately, these restrictions are not sufficient to prevent an attacker from using the chosen-plaintext attack to attack the TGS principal in `B`, or, for that matter, any cross-realm TGS principals in `B`. This flaw permits an attacker to recursively compromise an entire authentication network comprising numerous realms, even though transiting through multiple realms is normally not permitted in Kerberos version 4. Hopping through realms is expensive: $O(c^n)$ tickets must be forged to compromise a realm n hops away.

9.2. Cross-protocol attack

The reuse of a key for multiple cryptographic purposes can also lead to a vulnerability. The MIT implementation of version 5 has a backwards compatibility mode that allows the KDC to issue version 4 tickets. A single database is used for handling both version 4 and version 5 requests, and the KDC uses the same key to encrypt both version 4 and version 5 tickets for a given service principal. An attacker can use the version 4 encryption oracle to attack the version 5 protocol. This enables a wider attack than would be possible against version 4 alone.

Tickets in Kerberos version 5 also consist of a string of encrypted data, although, unlike version 4, the version 5 encryption scheme performs some encoding on the data prior to encryption. The format of the encoded plaintext in version 5 immediately prior to encryption is

$$\{confounder, checksum, data, pad\}.$$

The confounder is one block of random bytes. The Kerberos version 5 specification [23] chooses the block cipher as DES, which is used in CBC mode, and allows the checksum to be CRC-32, MD4, or MD5. None of these checksums is keyed. This lack of keying enables the cross-protocol attack from version 4 to version 5.

For DES with MD4 (des-cbc-md4) and DES with MD5 (des-cbc-md5), the IV of the encryption is a block of zeros. For DES with CRC-32 (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 difficult, if not impossible, to create an encryption oracle by manipulating version 5 tickets.

For des-cbc-md4 and des-cbc-md5, an attacker can simply fabricate any desired confounder using the version 4 encryption 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 in version 4, *i.e.*, by forcing the KDC to encrypt an initial P_0 with the attacked key (which generally requires control of a realm sharing a key with the target realm).

10. The evolution of Kerberos encryption

The original specification of Kerberos version 5, while it makes significant improvements on version 4, still contains some flaws. In particular, it attempts to provide authenticated encryption, but remains vulnerable to attacks based on encryption oracles. Work continues on

Kerberos encryption schemes; the proposed changes remedy the cryptographic vulnerabilities in both version 4 and RFC 1510, and eliminate the oracle-based attacks we describe.

10.1. Kerberos version 5

Kerberos version 5 adds checksums to the encoded plaintext. This provides some protection against simple attacks, but it is not sufficient to prevent the chosen-plaintext attacks we describe. Given an encryption oracle, an attacker can simply fabricate a valid checksum as part of the desired plaintext. An unforgeable signature, *e.g.*, a keyed hash whose creation requires knowledge of a secret, will thwart an attacker equipped with an encryption oracle, assuming that the attacker does not have a signature oracle.

Version 5 alone would prevent an encryption oracle by its use of random confounders. An attacker cannot predict the ciphertext corresponding to a chosen plaintext block, and is therefore unable to create an oracle. The MIT implementation's sharing of version 4 and version 5 keys enables the cross-protocol attack against version 5, by allowing an attacker to use the encryption oracle of version 4 to forge ciphertexts for version 5.

The confounder in version 5 also prevents a cut-and-paste attack against the TGS exchange. The version 5 TGS exchange permits a client to submit arbitrary plaintext for the KDC to encrypt in a ticket, in the form of the *AuthorizationData* field. If it were not for the confounder, an attacker could submit the plaintext of a desired forged ticket as *AuthorizationData* in a legitimate TGS request and obtain the desired ticket's ciphertext as a substring of the actual ticket returned by the TGS [11]. Others [9, 38] have previously noted the weaknesses of encrypted checksums against cut-and-paste attacks. The presence of a checksummed confounder in the ticket prevents an attacker from predicting the feedback block for the initial ciphertext block of the forged ticket. If the confounder were not checksummed with the plaintext, the attack would still be possible, as the receiving service would discard the confounder without verifying its integrity.

10.2. Upcoming revisions to Kerberos

The upcoming revision [34] to the Kerberos version 5 protocol specification repairs many of the flaws in RFC 1510. This revision moves the encryption specification [36] to a distinct document, which increases the separation between encryption and protocol recommended by Bellovin and Merritt [12]. Ciphers stronger than single-DES, *e.g.*, triple-DES or the Advanced Encryption Standard (AES) [32, 35], may now be used. The revised encryption specification recommends that a ciphertext output format of

$$\{encrypt(k_e, plaintext), HMAC(k_c, plaintext)\},$$

where *encrypt* is the encryption function, *e.g.*, triple-DES in CBC mode, or AES in CTS mode; *HMAC* is a keyed hash [26]; and *plaintext* is

$$\{confounder, data, pad\}.$$

k_e is a key used only for performing encryption, and k_c is a key only used for creating keyed hashes. Both keys are derived from the underlying key exchanged in the protocol.

The use of HMAC prevents an attacker from constructing a valid ciphertext without access to the signature key k_c, thus precluding many chosen-plaintext attacks. This feature of HMAC improves upon the weak scheme used in the original version 5 protocol, where an attacker could easily construct a valid ciphertext using an encryption oracle. The updated protocol derives the keys k_e and k_c from the actual shared key via a one-way *key derivation* function. The new encryption specification also recommends using key derivation to produce separate keys for each possible use of a shared key, thus restricting the actual quantity and type of plaintext encrypted with each key. This practice makes chosen-plaintext attacks even more difficult.

Bellare and Namprempre [7], as well as Krawczyk [25], analyze the composition of encryption and authentication primitives. Krawczyk determines that the "Encrypt-and-Authenticate" scheme (called "Encrypt-and-MAC plaintext" by Bellare and Namprempre) is insecure in the general case. The independent encryption and HMAC of the revised Kerberos version 5 encryption scheme appears to fall into this category, but could be better identified as an "Encode-then-E&M" scheme, into which Bellare *et al.* [6] classify the SSH encryption scheme.

Bellare *et al.* [6] recall that the SSH protocol's predictable chained IVs make it insecure against a chosen-plaintext attack. One of their proposed fixes adds random padding as part of an encoding step prior to encrypting and MACing the plaintext, and uses randomized IVs for each separate ciphertext packet. Kerberos version 5 uses a similar strategy, though Kerberos doesn't specify the use of a keyed hash until the recent revision of the encryption specification. Bellovin and Merritt [12] note that version 5's random confounder is equivalent to a random IV, though they criticize the use of the former.

The work of Bellare *et al.* on the SSH encryption scheme shows that the unmodified "Encode-then-E&M" scheme is not sufficient if it has predictable IVs. While Kerberos version 4 effectively has predictable (and constant!) IVs, version 5's use of random confounders prevents an attacker from predicting the relationships between

plaintext and ciphertext. The revisions to the version 5 encryption scheme [36] add a MAC, which further prevents various attacks.

11. Conclusions

The lack of authenticated encryption in Kerberos version 4 leads to a very serious attack. This deficiency is one of several errors in version 4. Though the existing version 5 specification (RFC 1510) is an improvement over version 4, it too has vulnerabilities. Ongoing revisions to version 5 remedy the existing flaws in RFC 1510.

Protocol designers should clearly identify the role of encryption in their protocols. If the protocol requires nonmalleable encryption, clearly state so in the specification, and ensure that the encryption scheme actually achieves this requirement. The use of unauthenticated encryption in version 4 violates the implicit requirement of nonmalleability, with dramatic results.

Designers should abstract the use of encryption away from the layout of the protocol messages. The designers of version 4 erroneously assumed that the message layout itself would enhance the security properties of the encryption scheme. Such cross-dependencies complicate analysis of the protocol, leading to security vulnerabilities.

CBC and related cipher modes have weaknesses when used with deterministic IVs. In particular, if a fixed plaintext encrypts to a fixed ciphertext, an attacker can build an encryption oracle. Even if a protocol does not allow an encryption oracle, message authentication schemes should be robust against encryption oracles. As we have shown, encrypted plaintext checksums are insufficient protection.

Using a single key for multiple purposes allows weaknesses in one system to compromise another, as illustrated by the cross-protocol attack. The revisions to version 5 specify using key derivation to produce different keys for different cryptographic purposes.

Protocols have a longer life than their designers might anticipate. Despite its age, and weaknesses related to its age, *e.g.*, its use of single-DES, Kerberos version 4 remains in widespread use. Even published criticisms about the security of version 4 have not significantly diminished its use. It has taken the revelation of a cryptographic flaw of this magnitude to encourage users to finally begin moving away from the protocol.

Acknowledgments

We thank Steve Bellovin, Mark Eichin, Radia Perlman, Amy Yu, and the anonymous referees for reading and providing useful commentary on earlier versions of this paper. We also thank Steve Bellovin, Matt Blaze, Love Hörnquist-Åstrand, Jeff Hutzelman, Perry Metzger, Jeff Schiller, and Ted Ts'o for useful discussions and advice. Steve Dorner brought to our attention a ciphertext-corrupting application bug that inspired us to discover the attack. We thank Marshall Vale and the rest of the MIT Kerberos Development Team for their ongoing support.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[2] G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 361–375. LNCS 1485, Springer-Verlag, 1998.

[3] G. Bella and E. Riccobene. Formal analysis of the Kerberos authentication system. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

[4] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 1997.

[5] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. Available from `http://www.cs.ucsd.edu/users/mihir/papers/sym-enc.html`, 2000.

[6] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In *Proceedings of 9th ACM Conference on Computer and Communications Security*. ACM, November 2002.

[7] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology – ASIACRYPT 2000*, pages 531–545. LNCS 1976, Springer-Verlag, 2000.

[8] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Available from `http://www.cs.ucsd.edu/users/mihir/papers/oem.html`, 2000.

[9] S. M. Bellovin. Problem areas for the IP security protocols. In *Proceedings of the Sixth USENIX UNIX Security Symposium*. USENIX, July 1996.

[10] S. M. Bellovin. Cryptography and the internet. In *Advances in Cryptology – CRYPTO '98*, pages 46–55. LNCS 1462, Springer-Verlag, 1998.

[11] S. M. Bellovin and D. Atkins. Private communications, 1999.

[12] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. In *USENIX Conference Proceedings*, pages 253–267, Dallas, TX, Winter 1991. USENIX.

[13] J. Black and H. Urtubia. Side-channel attacks on symmetric encryption schemes: The case for authenticated encryption. In *Proceedings of the 11th USENIX Security Symposium*. USENIX, August 2002.

[14] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 426(1871):233–271, 1989.

[15] `cr_ciph.c`. Source code of Kerberos version 4 library, MIT, 1986.

[16] `cr_tkt.c`. Source code of Kerberos version 4 library, MIT, 1985.

[17] R. DeMillo and M. Merritt. Protocols for data security. *Computer*, 16(2):39–50, February 1983.

[18] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[19] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[20] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley, 2003.

[21] A. M. Iley. Kerberos ivec attack. Web page, now defunct.

[22] A. Joux, G. Martinet, and F. Valette. Blockwise-adaptive attackers. In *Advances in Cryptology – CRYPTO 2002*, pages 17–30. LNCS 2442, Springer-Verlag, 2002.

[23] J. Kohl and C. Neuman. The Kerberos network authentication service (v5). Internet Request for Comments 1510, Internet Engineering Task Force, 1993.

[24] Messages exchanged on the `krb-protocol@athena.mit.edu` email list, April 1989.

[25] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *Advances in Cryptology – CRYPTO 2001*, pages 310–331. LNCS 2139, Springer-Verlag, 2001.

[26] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. Internet Request for Comments 2104, Internet Engineering Task Force, 1997.

[27] G. Lowe. Some new attacks upon security protocols. In *Proceedings of the 9th Computer Security Foundations Workshop*. IEEE, 1996.

[28] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. Project Athena technical plan, section E.2.1, Massachusetts Institute of Technology, 1987.

[29] J. H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, May 1988.

[30] National Institute of Standards and Technology. DES modes of operation. FIPS publication 81, U.S. Department of Commerce, December 1980.

[31] National Institute of Standards and Technology. Data encryption standard. FIPS publication 46-3, U.S. Department of Commerce, October 1999.

[32] National Institute of Standards and Technology. Advanced encryption standard. FIPS publication 197, U.S. Department of Commerce, November 2001.

[33] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[34] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos network authentication service (v5). Internet Draft `draft-ietf-krb-wg-kerberos-clarifications-04.txt`, 2003.

[35] K. Raeburn. AES encryption for Kerberos 5. Internet Draft `draft-raeburn-krb-rijndael-krb-05.txt`, 2003.

[36] K. Raeburn. Encryption and checksum specifications for Kerberos 5. Internet Draft `draft-ietf-krb-wg-crypto-06.txt`, 2003.

[37] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter Conference*, pages 191–202, February 1988.

[38] S. G. Stubblebine and V. D. Gligor. On message integrity in cryptographic protocols. In *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 85–104. IEEE, May 1992.

[39] S. Vaudenay. Security flaws induced by CBC padding – Applications to SSL, IPSEC, WTLS .... In *Advances in Cryptology – EUROCRYPT 2002*, pages 534–545. LNCS 2332, Springer-Verlag, 2002.

[40] V. L. Voydock and S. T. Kent. Security mechanisms in high-level network protocols. *Computing Surveys*, 15(2):135–171, June 1983.