The Perils of Unauthenticated Encryption: Kerberos Version 4

Tom Yu
Sam Hartman
Ken Raeburn

Massachusetts Institute of Technology

06 February 2004
• Unauthenticated encryption in Kerberos version 4 creates a critical vulnerability.

• We implemented highly efficient chosen-plaintext attack to impersonate arbitrary principals

• Practical demonstration of importance of authenticating encryption

• Version 5 also differently vulnerable

• Ongoing revisions to version 5 fix these too
Authentication is More Important Than Confidentiality

- Unauthenticated encryption known to be dangerous

- Forging authentic ciphertext more useful than recovering plaintext

- Becoming someone else is more useful than knowing what someone said
Kerberos Vulnerabilities

• Kerberos version 4 has a critical authentication vulnerability allowing impersonation of arbitrary principals

• Caused by multiple design errors

• First specification of Kerberos version 5 (RFC 1510) has related (less serious) weaknesses.

• Upcoming revision to version 5 in IETF fixes even these.

• Despite improvements, obsolete version 4 remains in widespread use — protocols live longer than anticipated
The Version 4 Vulnerability

- Unauthenticated encryption of security-critical information

- Can forge credentials impersonating arbitrary principals

- Encryption oracle using legitimate protocol transactions

- Very efficient attack: $O(n)$ oracle queries to forge $n$ block-long ciphertext

- Successful attack may go completely unnoticed
Vulnerability is Symptom of Design Errors

• Designers of Kerberos version 4 failed to explicitly identify nonmalleability requirement

• Malleability in version 4 allows our attack

• Lack of good encryption abstraction contributed to problems

• Deterministic encryption scheme allows oracle

• Version 5 encryption thwarts oracle creation

• Make cryptographic assumptions explicit

• Create good encryption abstraction
RFC 1510 Flaws

- RFC 1510 uses encrypted plaintext checksums

- Message authentication can be subverted by using encryption oracle

- Designers should ensure that attackers can’t subvert message authentication by ciphertext surgery
Long-Lived Protocols

- Multiple application protocols built on top of Kerberos

- Deployment of security infrastructure expensive

- Resistance to change unless clear and present danger due to expense

- Use conservative design in security protocols

- Evaluate whether apparently theoretical weaknesses indicate more serious problems
Outline

- Kerberos History

- Design Shortcomings of Kerberos Version 4

- Kerberos Version 4 Protocol

- Block Encryption Oracle

- Ticket Ciphertext as Oracle

- Implementation Flaws

- Evolution of Kerberos Encryption
Kerberos History
Historical Overview

- Version 4 designed/deployed at MIT Project Athena (Miller et al. 1987; Steiner et al. 1988)

- Based on Needham-Schroeder (1978) symmetric-key

- Uses timestamps to mitigate replays (Denning 1981)

- Version 5, defined by RFC 1510

- Ongoing revision of version 5 in IETF
AFS leads to Widespread Deployment

- Andrew File System (AFS) developed at CMU

- AFS protocol uses Kerberos version 4

- Commerical AFS from Transarc/IBM

- Introduction of open-source version OpenAFS led to wider adoption of Kerberos version 4

- Reluctance to update AFS to use Kerberos version 5

- Our attack prompted rapid migration
Prior Work

- Formal correctness analyses (Bella and Riccobene 1997; Bella and Paulson 1998; Burrows et al. 1989)

- Deficiencies in encryption scheme (Bellovin and Merritt 1991; Stubblebine and Gligor 1992)

- Encryption oracle attacks against other protocols (Lowe 1996)
Evolution of Kerberos

- RFC 1510 fixes some flaws of version 4; still has some vulnerabilities

- Ongoing work (post-RFC 1510) in IETF
  - More explicit cryptographic abstraction
  - Strategies similar to recent work on SSH protocol (Bellare et al. 2002)
  - Fixes flaws in RFC 1510
Design Shortcomings of Kerberos
Version 4
Abstract Design Flaws

• Failure to make cryptographic assumptions explicit

• Needham-Schroeder implicitly requires nonmalleable encryption (Dolev et al. 2000)

• Kerberos version 4 fails to provide nonmalleability

• Concept of malleability not well-developed at time of design
Concrete Design Flaws

- DES in nonstandard Propagating Cipher Block Chaining (PCBC) mode

- Assumption: error propagation properties of PCBC sufficient to scramble plaintext after manipulation

- Constant Initialization Vector (IV)

- Use of PCBC for integrity via known values at end of plaintext

- PCBC as integrity check fails spectacularly

- PCBC insufficient against encryption oracle
Lack of Abstraction

- Dependency between integrity and message layout indicates lack of sufficient abstraction of encryption
- Separation of encryption and message details previously emphasized (Bellovin and Merritt 1991)
- Security of encryption should not depend on packet layout details
Our Attack Nearly Discovered Earlier

- Version 4 security assumptions do not include encryption oracle

- Existing chosen-plaintext attack (Voydock and Kent 1983) against CBC mode with fixed IV.

- Designers of version 4 were unaware

- Designers of version 5 nearly uncovered our attack on version 4 during design discussions

- Dismissed by (incorrect!) argument that first plaintext block is randomized

- Again, indicative of insufficiently abstracted encryption
Kerberos Version 4 Protocol
Dramatis Personae

- Trusted third party: Key Distribution Center (KDC)
- Client
- Server
Keys and Other Elements

- Client A’s long-term key: $k_a$

- Server B’s long-term key: $k_b$

- Ticket: ciphertext encrypted with $k_b$
  - Identifies client
  - Contains session key

- Credential: ticket and session key $k_{ab}$

- Ticket alone insufficient; also must prove knowledge of session key
Obtaining Credentials

• Two conceptual services in KDC
  – Authentication Service (AS)
  – Ticket Granting Service (TGS)

• The AS issues credentials encrypted using client’s long-term key $k_a$

• The TGS acts as a special application service for obtaining additional credentials

• Typically, client uses AS exchange to get ticket for TGS; permits single sign-on
AS Exchange

Here, client requests a ticket granting ticket (TGT) for later use with the TGS.

\[ A \rightarrow S : A, S \]
\[ S \rightarrow A : \{k_{as}, S, \{A, S, t_{s}, k_{as}\}k_{s}\}k_{a} \]

\( t_{s} \)  KDC’s timestamp
\( \{M\}k_{x} \)  \( M \) encrypted with key \( k_{x} \)
\( A \)  client name
\( S \)  TGS name
\( k_{a} \)  client long-term key
\( k_{s} \)  TGS long-term key
\( k_{as} \)  session key between client and TGS
\( \{A, S, t_{s}, k_{as}\}k_{s} \)  ticket
TGS Exchange

Here, client requests ticket for service $B$ from the TGS.

\[
A \rightarrow S : \{A, S, t_s, k_{as}\}k_s, \{A, t_a\}k_{as}, B \\
S \rightarrow A : \{k_{ab}, B, \{A, B, t'_s, k_{ab}\}k_b\}k_{as}
\]

$t_a$ client’s timestamp  
$B$ server name  
$k_b$ server long-term key  
$k_{as}$ session key between client and TGS  
$\{A, S, t_s, k_{as}\}k_s$ TGT  
$\{A, t_a\}k_{as}$ authenticator for TGS request  
$\{A, S, t'_s, k_{ab}\}k_b$ service ticket

Authenticator assures TGS that client has recent knowledge of session key $k_{as}$. 
Using Credential

Client sends application request

\[ A \rightarrow B : \{A, B, t'_s, k_{ab}\}k_b, \{A, t'_a\}k_{ab} \]

Authenticator \( \{A, t'_a\}k_{ab} \) proves to service \( B \) that \( A \) has recent knowledge of session key \( k_{ab} \).
AS Request

A $\rightarrow$ S

A, S

B
AS Reply

A, S

{kas, S, {A, S, ts, kas}ks}ka

A

S

B
TGS Request

\{A, S, t_s, k_a\}k_s, \{A, t_a\}k_a s, B
TGS Reply
Application Request

\[
\begin{align*}
\{A, S, t'_s, k_{as}\}k_s, \{A, t'\}_{k_{as}}B & \\
\{k_{ab}, B, \{A, B, t''_s, k_{ab}\}k_b\}k_{as} & \\
\{A, B, t''_s, k_{ab}\}k_b, \{A, t'_a\}k_{ab} & \end{align*}
\]
Kerberos Names

- Form of principal name has implications for attack

- Principal name is a triple
  \[
  \{\text{primaryname}, \text{instance}, \text{realm}\}
  \]

- Usually displayed like
  \[
  \text{primaryname} \cdot \text{instance}@$\text{realm}$
  \]

- Normal TGS principal name
  \[
  \text{krbtgt} . \text{realm}@\text{realm}
  \]

- Cross-realm TGS principal name
  \[
  \text{krbtgt} . \text{localrealm}@\text{foreignrealm}
  \]
Cross-realm Authentication

- Local KDC checks cross-realm TGT for client principal realm matching foreign realm; local KDC can’t normally issue ticket certifying wrong client realm

- Implementation flaws allow for circumvention of this check

- As designed, sharing cross-realm keys only implies trust that foreign realm trustworthy for its own principals

- Compromise of foreign realm only renders that realm’s principals untrustworthy

- Cryptographic flaws invalidate these trust assumptions

- Cross-realm TGS requests useful for inserting known plaintext
TGS Request for $S_2$

$\{A, S_1, t_{s_1}, k_{s_1}\} k_{s_1}, \{A, t_a\} k_{s_1}, S_2$
TGS Reply for $S_2$

\[
\{A, S_1, t_{s_1}, k_{as_1}\} k_{s_1}, \{A, t_a\} k_{as_1}, S_2
\]

\[
\{k_{as_2}, S_2, \{A, S_2, t'_{s_1}, k_{as_2}\} k_{s_2}\} k_{as_1}
\]
TGS Request for $B$

\[
\begin{align*}
\{A, S_1, t_{s_1}, k_{as_1}\} & \{A, t_a\} k_{as_1}, S_2 \\
\{k_{as_2}, S_2, \{A, S_2, t'_{s_1}, k_{as_2}\}\} k_{as_1} & \{A, S_2, t'_{s_1}, k_{as_2}\} k_{as_2}, \{A, t'_a\} k_{as_2}, B
\end{align*}
\]
TGS Reply for $B$

\[
\{A, S_1, t_{s_1}, k_{as_1}\}_{k_{s_1}}, \{A, t_a\}_{k_{as_1}}, S_2
\]

\[
\{k_{as_2}, S_2, \{A, S_2, t'_{s_1}, k_{as_2}\}_{k_{s_2}}\}_{k_{as_1}}
\]

\[
\{A, S_2, t'_{s_1}, k_{as_2}\}_{k_{s_2}}, \{A, t'_a\}_{k_{as_2}}, B
\]

\[
\{k_{ab}, B, \{A, B, t_{s_2}, k_{ab}\}_{k_b}\}_{k_{as_2}}
\]
Application Request to $B$
Block-Encryption Oracle
Block-Encryption Oracle

- Chosen plaintext allows block-encryption oracle in Kerberos version 4
- Oracle takes advantage of fixed or predictable IV
- Oracle uses structure of CBC or PCBC mode
Cipher Modes

- Cipher Block Chaining (CBC) mode

\[ C_{i+1} = k(P_{i+1} \oplus C_i) \]
\[ P_{i+1} = k^{-1}(C_{i+1}) \oplus C_i \]

- Propagating Cipher Block Chaining (PCBC) mode

\[ C_{i+1} = k(P_{i+1} \oplus C_i \oplus P_i) \]
\[ P_{i+1} = k^{-1}(C_{i+1}) \oplus C_i \oplus P_i \]

\[ C_0, C_1, \ldots, C_n \] ciphertext blocks
\[ P_0, P_1, \ldots, P_n \] plaintext blocks
\[ k(x) \] encryption of block \( x \) with key \( k \)
\[ k^{-1}(x) \] decryption of block \( x \) with key \( k \)
\[ x \oplus y \] bitwise exclusive-OR of \( x \) with \( y \)
CBC encrypt

\[ C_{i+1} = k(P_{i+1} \oplus C_i) \]

CBC decrypt

\[ P_{i+1} = k^{-1}(C_{i+1}) \oplus C_i \]
PCBC encrypt

\[ P_0 \rightarrow \oplus \rightarrow k \rightarrow \oplus \rightarrow C_0 \]
\[ \rightarrow \oplus \rightarrow \cdots \]

\[ C_0 \rightarrow \oplus \rightarrow k \rightarrow \oplus \rightarrow P_0 \]

\[ C_i + 1 = k(P_i + 1 \oplus C_i \oplus P_i) \]

PCBC decrypt

\[ C_0 \rightarrow \oplus \rightarrow k^{-1} \rightarrow \oplus \rightarrow P_0 \]
\[ \rightarrow \oplus \rightarrow \cdots \]

\[ P_0 \rightarrow \oplus \rightarrow k^{-1} \rightarrow \oplus \rightarrow C_0 \]

\[ P_i + 1 = k^{-1}(C_i + 1) \oplus C_i \oplus P_i \]
Generalized Feedback Modes

- CBC and PCBC can be generalized as feedback modes
  \[ C_i = k(P_i \oplus F_i) \]
  \[ P_i = k^{-1}(P_i) \oplus F_i, \]

- \( F_i \) is \( i \)-th feedback block; not necessarily transmitted

- CBC mode: \( F_{i+1} = C_i \)

- PCBC mode: \( F_{i+1} = C_i \oplus P_i \)
Predictable Feedback Makes an Oracle

To get encryption $X = k(M)$ of block $M$

- Find an $F_j$ that will remain the same when $P_j$ replaced with $P'_j$

- Choose $P'_j = M \oplus F_j$

- Now, the new ciphertext block $C'_j$ is the desired encryption of $M$

$$C'_j = k(P'_j \oplus F_j)$$

$$= k(M \oplus F_j \oplus F_j)$$

$$= k(M) = X.$$

- In well-designed protocol, attacker can’t create this oracle, since $F_j$ should not be predictable
Original Plaintext

\[ IV = F_0 \oplus P_0 = F_1 \oplus C_0 \oplus k \oplus P_1 = \cdots \]
Chosen Plaintext

\[ IV = F_0 \]

\[ C_0 = k \]

\[ C'_1 = k(M) \]

\[ P_0 \rightarrow F_1 \rightarrow P'_1 \rightarrow M \]

\[ k \rightarrow k \]
Constructing Desired Ciphertext

To get ciphertext blocks \( \{X_i\} \), whose plaintext blocks are \( \{M_i\} \), for each \( M_i \)

- Use oracle to perform block encryption \( X_i = k(M_i \oplus \Phi_i) \)

- \( \Phi_i \) is the feedback block, e.g., \( \Phi_{i+1} = M_i \oplus X_i \) in PCBC mode

- Choose plaintext block \( P'_j = M_i \oplus \Phi_i \oplus F_j \)

- This gives \( C'_j = k(M_i \oplus \Phi_i) \)
Ticket Ciphertext as Oracle
Kerberos Version 4 Ticket (pre-encryption)

- 1 byte flags namely, \texttt{HOST\_BYTE\_ORDER}
- string \texttt{pname} client’s name
- string \texttt{pinstance} client’s instance
- string \texttt{prealm} client’s realm
- 4 bytes \texttt{paddress} client’s address
- 8 bytes \texttt{session} session key
- 1 byte \texttt{life} ticket lifetime
- 4 bytes \texttt{time\_sec} KDC timestamp
- string \texttt{sname} service’s name
- string \texttt{sinstance} service’s instance
- \leq 7 bytes \texttt{null} null pad to 8 byte multiple

- \textit{flags} has only one meaningful bit (\texttt{HOST\_BYTE\_ORDER} — KDC’s byte order)

- “\textit{string}” fields are NUL-terminated ASCII, max 40 chars
Chosing Plaintext in Ticket

- Hold one aligned block of client name constant

- Vary the following block as chosen plaintext for oracle

- First byte (*flags*) usually constant

- IV is constant, but unknown (it’s the key)

- Client name and instance easily controlled
TGS Attack Scenario

- Attacker controls realm \( A \)
- Target realm \( B \)
- Attacker knows key of TGS principal \( \text{krbtgt}.B@A \), which has same key as \( \text{krbtgt}.A@B \)
- Attacker creates cross-realm ticket with client principal \( a234567xxxxxxx@A \)
  - “a234567” arbitrary and held constant
  - “xxxxxxx” is the \( P_1' \) varied to produce desired ciphertext block
- Attacker uses cross-realm ticket to get ticket for target service
TGS Attack Scenario (cont’d)

Resulting initial two blocks of plaintext of ticket issued by realm B KDC are

\[
\begin{array}{c|c|c}
\text{flags} & \text{a234567} & \text{XXXXXXXX} \\
\end{array}
\]

\[\leftarrow P_0 \rightarrow \leftarrow P'_1 \rightarrow\]

- \(C_0\) remains constant, due to constant IV and constant \(P_0\)

- \(F_1\) also remains constant

- \(C'_1\) contains desired ciphertext block when \(P'_1\) is varied

- Choose \(P'_1 = M_i \oplus \Phi_i \oplus F_1\) to get \(C'_1 = X_i\)
TGS Attack Scenario Limitations

- May need to choose different $P_0$ if $P'_1$ needs to contain too many NUL characters

- First ciphertext block can be problematic
  - Can sniff initial ciphertext block
  - Can submit initial substring of target principal, with restrictions due to NUL characters
  - Cross-protocol attack
Alternate Attack Scenario

- Knowledge of sufficient number of keys in target realm allows using AS exchange as oracle

- Particularly effective if attacker can create principals in target realm

- MIT implementation allows less-privileged administrators to create/change keys for host-based service

- Typically of form `rcmd.hostname@realm`

- Used for authenticating remote logins

- May need to create as few as $n$ principals for $n$ blocks of ciphertext
Alternate Attack Scenario Limitations

- Not very useful in version 4-only environment

- Few interesting client principals begin with “rcmd”

- Cross-protocol attack can work, though

- Can also get initial ciphertext block by sniffing
Implementation Flaws
Implementation flaws make certain additional attacks possible

- MIT implementation of version 4 has lax checking for cross-realm TGT issuance, allowing “hopping” between realms, which is normally not permitted

- MIT implementation of version 5 shares keys with its version 4 backwards-compatibility mode, allowing cross-protocol attack
Realm Hopping

- MIT implementation of version 4 will issue “useless” tickets that would be rejected by the target realm’s KDC

<table>
<thead>
<tr>
<th>client</th>
<th>using TGS</th>
<th>requested service</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>clienta@A</td>
<td>krbtgt.A@A</td>
<td>krbtgt.B@A</td>
<td>issued</td>
</tr>
<tr>
<td>clienta@A</td>
<td>krbtgt.B@A</td>
<td>krbtgt.C@B</td>
<td>issued</td>
</tr>
<tr>
<td>clienta@A</td>
<td>krbtgt.C@B</td>
<td>krbtgt.D@C</td>
<td>rejected</td>
</tr>
<tr>
<td>clienta@A</td>
<td>krbtgt.B@A</td>
<td>krbtgt.B@B</td>
<td>issued</td>
</tr>
<tr>
<td>clienta@A</td>
<td>krbtgt.B@B</td>
<td>anything@B</td>
<td>rejected</td>
</tr>
</tbody>
</table>

- Normally harmless, this allows use of forged tickets for krbtgt.B@A to run an oracle on key for krbtgt.C@B

- Recursive realm compromise possible; must forge $O(c^n)$ tickets to compromise a realm $n$ hops away
Realm Hopping (cont’d)

- Can shortcut by forging tickets of a realm administrator

- Can also use forged tickets for `krbtgt.B@A` to run an oracle on key for `krbtgt.B@B`. 
Cross-Protocol Attack

- Using a key for multiple cryptographic purposes can be a vulnerability

- MIT implementation of version 5 can allow KDC to issue version 4 tickets for backwards compatibility

- Same key used for version 4 and version 5 tickets for a principal

- Can use version 4 ticket oracle to forge ciphertext for version 5 ticket
Cross-Protocol Attack (cont’d)

- RFC 1510 uses single-DES

- Encoded plaintext of version 5 ticket prior to encryption is

  \[
  \text{confounder} | \text{checksum} | \ldots \text{data} \ldots | \text{pad}
  \]

- \textit{confounder} is a block of random bits

- \textit{checksum} is not keyed

- For checksums MD4 and MD5, IV is block of zeros

- For CRC-32 checksum, key used as IV
Cross-Protocol Attack (cont’d)

- Forge complete encoded plaintext of version 5 ticket, including checksum

- Use version 4 oracle to encrypt

- CRC-32 checksum makes first block slightly tricky; can use any initial ciphertext block whose plaintext is known, as receiver doesn’t know what random confounder value to expect
Evolution of Kerberos Encryption
RFC 1510

- Improvement over version 4

- No more PCBC

- Confounder prevents using ciphertext as oracle

- Plaintext checksum allows use of version 4 block-encryption oracle to forge ciphertext

- Confounder also prevents cut-and-paste attacks (Bellovin and Atkins, private communication, 1999)
Ongoing Revision to Kerberos Version 5

- Repairs many flaws in RFC 1510

- Increases encryption abstraction; moves encryption specification to separate document

- Stronger ciphers (triple-DES, AES, etc.)

- Uses HMAC for integrity checking — can’t be forged with encryption oracle
Revised Ciphertext (post-RFC 1510)

- Ciphertext output
  
  \[ \text{encrypt}(k_e, \text{plaintext}) | \text{HMAC}(k_c, \text{plaintext}) \]

- Encoded plaintext
  
  \[ \text{confounder} | \text{data} | \text{pad} \]

- \( k_e \): derived key exclusively for encrypting

- \( k_c \): derived key exclusively for HMAC

- Both derived via one-way function from key exchanged in protocol
Conclusions

• Critical vulnerabilities in Kerberos version 4 provide examples of errors in cryptographic protocol design

• Clearly identify role of encryption in protocols

• Use good abstraction of encryption to avoid cross-dependencies between message layout and encryption

• Avoid deterministic encryption

• Avoid using one key for multiple purposes

• Protocols live longer than expected