Network Security: Protocol Flaws, Kerberos

Tuomas Aura, Microsoft Research, UK

Outline

- Classic key-exchange protocols and flaws
- Advanced protocol properties
- Kerberos authentication
- Kerberos in Windows domains

Classic key-exchange protocols and flaws

Needham-Schroeder secret-key protocol

- The first secret-key key-exchange protocol 1978
- Kerberos is based on this protocol
- Trusted server T shares a secret master key with everyone. Encrypts a ticket and session key with master keys:
  1. A → T: A, B, N_{A1}
  2. T → A: $E_{TA}(N_{A1}, B, K_{ses}, ticket_{AB})$
  3. A → B: $ticket_{AB}$, $E_{ses}(N_{A2})$
  4. B → A: $E_{ses}(N_{A2}-1,\ N_B)$
  5. A → B: $E_{ses}(N_B-1)$

  K_{TA}, K_{TB} = A's and B's master keys
  K_{ses} = session key selected by T
  $ticket_{AB} = E_{KTB}(K_{ses}, A)$
- Encryption is assumed to also protect integrity. Nowadays, we would use standard MACs but they did not exist in the 70s. For example:
  $E_K(M) = AES\text{-}CBC_K(M, HMAC\text{-}SHA\text{-}1_K(M))$

Needham-Schroeder analysis

- The protocol again:
  1. A → T: A, B, N_{A1}
  2. T → A: $E_{TA}(N_{A1}, B, K_{ses}, ticket_{AB})$ // $ticket_{AB} = E_{TB}(K_{ses}, A)$
  3. A → B: $ticket_{AB}$, $E_{Kses}(N_{A2})$
  4. B → A: $E_{Kses}(N_{A2}-1,\ N_B)$
  5. A → B: $E_{Kses}(N_B-1)$
- T encrypts session key under A's and B's master keys
- Authenticators for key confirmation
- N_{A1} guarantees freshness of ticket and session key to A
- N_{A2} and N_B guarantee freshness of authenticators to A and B
- But no freshness of the ticket to B…

Needham-Schroeder vulnerability

- The protocol again:
  1. A → T: A, B, N_{A0}
  2. T → A: $E_{TA}(N_{A0}, B, K_{ses}, ticket_{AB})$ // $ticket_{AB} = E_{TB}(K_{ses}, A)$
  3. A → B: $ticket_{AB}$, $E_{Kses}(N_A)$
  4. B → A: $E_{Kses}(N_A-1,\ N_B)$
  5. A → B: $E_{Kses}(N_B-1)$
- Vulnerability discovered by Denning and Sacco 1981
  - Freshness of the ticket not guaranteed to B
  - Assume attacker compromises an old session key and has sniffed the corresponding ticket, or
  - Assume attacker compromises A's master key K_{TA}. A's master key is replaced but, before that, the attacker manages to obtain a ticket for B
- Replay attack by C who knows an old session key:
  3'. C(A) → B: $ticket_{AB\text{-}old}$, $E_{KAB\text{-}old}(N_A)$ // $ticket_{AB\text{-}old} = E_{KTB\text{-}old}(K_{AB\text{-}old}, A)$
  4'. B → C(A): $E_{KAB\text{-}old}(N_A-1,\ N_B)$
  5'. C(A) → B: $E_{KAB\text{-}old}(N_B-1)$
- How to fix?

Denning-Sacco protocol

- Public-key key exchange 1981; flaw found in 1994
- A obtains certificates from trusted server T for both A's and B's public keys
  1. A → T: A, B
  2. T → A: $Cert_A$, $Cert_B$
  3. A → B: $E_B(T_A, K_{ses}, S_A(T_A, K_{ses}))$, $Cert_A$, $Cert_B$

  K_{ses} = session key selected by A

  $Cert_A = A, PK_A, S_T(A, PK_A)$
- Analysis:
  - Expiration time missing from certificates → need to be added!
  - Public-key encryption for secrecy of K_{AB} → ok
  - Public-key signature for authenticity of K_{AB} → but what exactly is authenticated?
  - Time stamp for freshness of session key → what about quick replays?

7

Denning-Sacco vulnerability

- The protocol again:
  1. A → T: A, B
  2. T → A: $Cert_A$, $Cert_B$
  3. A → B: $E_B(T_A, K_{ses}, S_A(T_A, K_{ses}))$, $Cert_A$, $Cert_B$
- The signed message $S_A(T_A, K_{AB})$ does not contain all possible information.
- Q: What is missing?

  A: The signed message is not bound to the identity of B
- Q: Does it matter when only B can decrypt the message?

  A: B could be bad!

  → B can forward the last message to anyone else, e.g. to C:

  3'. B(A) → C: $E_C(T_A, K_{AB}, S_A(T_A, K_{AB}))$, $Cert_A$, $Cert_C$

  C thinks it shares K_{ses} with A but it is really shared with B
- Legitimate user B can impersonate legitimate users → insider attack
- How to fix?

8

Needham-Schroeder public-key protocol

- The first public-key key-exchange protocol 1978; flaw found in 1995 [Lowe95]
- A and B know each other's public encryption keys or obtain certificates from a trusted server T as needed
- A and B exchange key material:
  1. A → B: $E_B(N_A, A)$
  2. B → A: $E_A(N_A, N_B)$
  3. A → B: $E_B(N_B)$

  N_A, N_B = secret nonces

  $K_{ses} = h(N_A, N_B)$
- Analysis:
  - Session key secret and fresh, entity authentication ok
  - Session key not bound to A

9

Needham-Schroeder public-key vulnerablity

- The protocol again:
  1. A → B: $E_B(N_A, A)$
  2. B → A: $E_A(N_A, N_B)$
  3. A → B: $E_B(N_B)$

  $K_{ses} = h(N_A, N_B)$
- A authenticates to C. C can forward the authentication of A to B:

1.	A → C:	$E_C(N_A, A)$		
1'.	C(A) → B:		$E_B(N_A, A)$	// Re-encrypt for B
2'./2.	B → A:	$E_A(N_A, N_B)$		// Allow through
3.	A → C:	$E_C(N_B)$		
3'.	C(A) → C:		$E_B(N_B)$	// Re-encrypt for B

  C thinks it shares K_{ses} with A but also B knows it (for A, everything is ok… in a way)
- Insider attack: legitimate user B impersonates another user A
- How to fix?
  - Hint: what could be added to each message? Does it help?

10

Wide-mouth-frog protocol

- Toy protocol but interesting
- A and B share secret master keys with trusted server T. T delivers session keys:
  1. A → T: A, $E_{K_{TA}}(T_A, B, K_{ses})$
  2. T → B: $E_{K_{TB}}(T_T, A, K_{ses})$

  K_{TA}, K_{TB} = A's and B's master keys shared with T

  T_A, T_T = time stamps

  K_{ses} = session key selected by A
- Analysis:
  - Encryption must protect integrity → implement with a MAC
  - Otherwise, looks pretty good… but there is a subtle issue with the time stamps

11

Wide-mouth-frog protocol

- The protocol again:
  1. A → T: A, $E_{TA}(T_A, B, K_{ses})$
  2. T → B: $E_{TB}(T_T, A, K_{ses})$

  K_{TA}, K_{TB} = A's and B's master keys shared with T

  T_A, T_B = time stamps

  K_{ses} = session key selected by A
- Attack:

1.	A → T:	A, $E_{TA}(T_A, B, K_{ses})$
2.	T → B:	$E_{TB}(T_T, A, K_{ses})$
1.	C(B) → T:	A, $E_{TA}(T_{A2}, B, K_{ses})$
2.	T → C(A):	$E_{TB}(T_{T2}, A, K_{ses})$
1.	C(A) → T:	A, $E_{TA}(T_{A3}, B, K_{ses})$
2.	T → C(B):	$E_{TB}(T_{T3}, A, K_{ses})$
1.	C(A) → T:	A, $E_{TA}(T_{A4}, B, K_{ses})$
2.	T → C(B):	$E_{TB}(T_{T4}, A, K_{ses})$
…	…	…

- Attacker can keep the session key alive for ever
- Problem: authenticated messages 1 and 2 can be confused with each other
- How to fix?

  Add type tags to make the meaning of messages unambiguous

12

What is a protocol flaw?

- Researchers like to present spectacular attacks and flaws but the reality is often less black and white
- Limitation on the applicability of the protocol:
  - Do insider attacks matter?
  - Multiple parallel executions by the same principal?
  - Is the protocol used for its original purpose or for something different?
- Requirements for implementations:
  - Encryption mode in old protocols must protect integrity (checksum or non-malleable encryption)
  - Signed and MAC'd messages must include type tags and be parsed unambiguously
- New requirements arise over time:
  - Man-in-the-middle attacks
  - DoS protections?
  - Identity protection?
  - Has the protocol become the weakest link after improvements to other parts of the system?

13

Advanced protocol properties

14

Station-to-station (STS) protocol

- Variation of the original STS
- Signed ephemeral Diffie-Hellman:
  1. A → B: g, p, g^x
  2. B → A: g^y, $E_{K_{ses}}(g^y, g, p, g^x, S_B(g^y, g, p, g^x), Cert_B)$
  3. A → B: $E_{K_{ses}}(g, p, g^x, g^y, S_A(g, p, g^x, g^y), Cert_A)$

  $K_{ses} = h(g^{xy})$
- What could be wrong?
- What does the encryption $E_K(\ldots)$ achieve?
- No known flaws (and STS has been well analyzed)
- Encryption with the DH session key → identity protection
  - Why does it need to be ephemeral DH?

15

My modified STS

- Add a cookie exchange:
  1. A → B: N_A
  2. B → A: N_A, N_B
  3. A → B: N_A, N_B, g, p, g^x
  4. B → A: g^y, $E_K(g^y, g, p, g^x, S_B(g^y, g, p, g^x))$, $Cert_B$
  5. A → B: $E_K(g, p, g^x, g^y, S_A(g, p, g^x, g^y))$, $Cert_A$

  $K = h(g^{xy})$
- What does this modification achieve?
- Responder B verifies A's IP address before the DH computation → prevents CPU-exhaustion attacks with a spoofed attacker IP address
- Even better: make B stateless between messages 2 and 3: $N_B = h(A,B,N_A,N_{B\text{-periodic}})$ where $N_{B\text{-periodic}}$ changes every minute
- What is the cost of this defence? Is it worth it?

16

Kerberos authentication

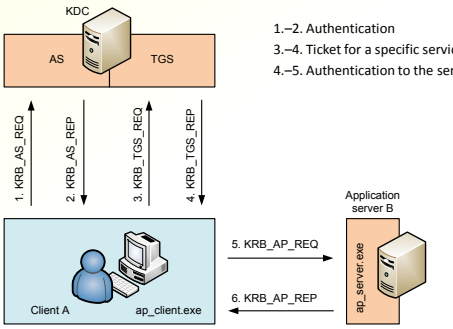
17

Kerberos

- Shared-key protocol for user login authentication
- Kerberos v4 1988-
- Kerberos v5 1993- [RFC 4120]
  - Updated protocol and algorithms
  - ASN.1 BER message encoding
  - Implemented in Windows 2000 and later

18

Kerberos architecture (1)



1.–2. Authentication
3.–4. Ticket for a specific service
4.–5. Authentication to the service

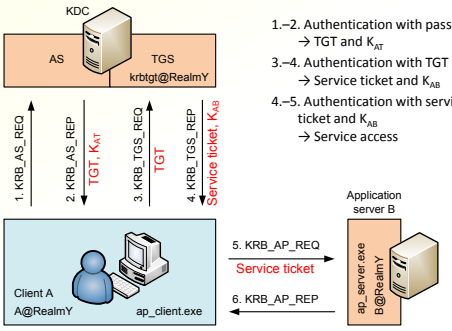
19

Kerberos terminology

- Client/server computing model
  - Authentication for remote login sessions
  - Users and services are principals
- Key distribution center (KDC)
  - Two components: authentication server (AS) and ticket-granting server (TGS)
  - Trusted by all principals
- KDC shares a master key with each principal
  - Long-term secret that is used only for initial authentication
  - Usually derived by hashing a password [RFC3961]
- When user logs in, his workstation uses the password to obtain a ticket-granting-ticket (TGT) from AS
- When client needs to access remote services, it uses TGT to request a service ticket from TGS for each server
  (Note how the two-step process could be generalized to more steps)

20

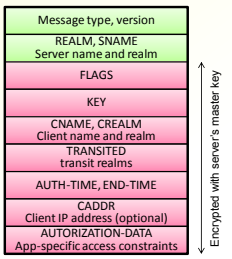
Kerberos architecture (2)



1.–2. Authentication with password
→ TGT and K_{AT}
3.–4. Authentication with TGT and K_{AT}
→ Service ticket and K_{AB}
4.–5. Authentication with service ticket and K_{AB}
→ Service access

21

Kerberos ticket



- Same format for both TGT and service ticket
- Credentials = ticket + key
- ASN.1 encoding in Kerberos v5
- "Encryption" actually means encryption and a MAC

- Flags:
  - FORWARDABLE, FORWARDED, PROXIABLE, PROXY, MAY-POST-DATE, POSTDATED, INVALID, RENEWABLE, INTINIAL, PRE-AUTHENT, HW-AUTHENT
  - INITIAL flag indicates TGT

22

Protocol details

- Initial login of user A:
  1. A → AS: Preauthentication, A, TGS, N_{A1}, $Addr_A$
  2. AS → A: A, TGT, E_{K_A} ($K_{A\text{-}TGS}$, N_{A1}, TGS, $Addr_A$)
- Ticket request:
  3. A → TGS: TGT, $Authenticator_{A\text{-}TGS}$, B, N_{A2}, $Addr_A$
  4. TGS → A: A, Ticket, $E_{K_{A\text{-}TGS}}$ (K_{AB}, N_{A2}, B, $Addr_A$)
- Authentication to server B:
  5. A → B: Ticket, $Authenticator_{AB}$
  6. B → A: AP_REP

K_A , K_{TGS}, K_B = master keys of A, TGS and B
$K_{A\text{-}TGS}$ = shared key for A and TGS
K_{AB} = shared key for A and B
TGT = B, $E_{K_{TGS}}$ (INITIAL, $K_{A\text{-}TGS}$, A, T_{auth}, $T_{expiry1}$, $Addr_A$))
Ticket = B, E_{K_B}(K_{AB}, A, T_{auth}, $T_{expiry2}$, $Addr_A$))
Preauthentication = E_{K_A} (^{1} T_A)
$Authenticator_{A\text{-}TGS}$ = $E_{K_{A\text{-}TGS}}$ (^{2} T_A)
$Authenticator_{AB}$ = $E_{K_{AB}}$ (^{3} T_A)
AP_REP = $E_{K_{AB}}$(^{4} T_A)
$Addr_A$ = A's IP addresses

Notes:

^{1234}) ASN.1 encoding adds type tags to all messages

Encryption mode protects message integrity

23

Crypto algorithms

- Algorithms in older implementations were complex and potentially weak, e.g.:
  - DES encryption
  - CRC-32 encrypted with DES in CBC mode for integrity
- Latest algorithm specification [RFC3961] recommends AES and HMAC
- Encryption mode must protect message integrity
  - Can be implemented by appending an HMAC

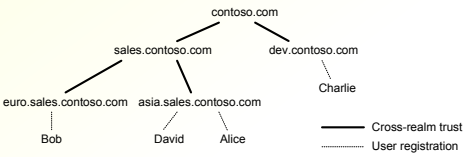
24

Kerberos realms

- Users and services registered to one KDC form a realm
  - name@realm, e.g. A@X, alice@asia.sales.contoso.com
- Cross-realm trust:
  - Two KDCs X and Y share a key (krbtgt@Y is registered in KDC X and krbtgt@X is registered in KDC Y)
  - KDCs believe each other to be honest and competent, at least to name users in their own realm
- Cross-realm authentication:
  - Client A@X requests from TGS at realm X a ticket for TGS at realm Y
  - The ticket is encrypted for krbtgt@Y (i.e. TGS at realm Y)
  - Client A@X requests from TGS at realm Y a ticket for server B@Y
  - Local policy at each KDC about when to honor tickets from other realms
  - Local policy at B@Y about whether to allow access to users from other realms
- Possible to transit multiple realms → TRANSITED field in the ticket lists intermediate realms
  - Local policy at each server about which transit realms are allowed

25

Realm hierarchy



- Large organizations can have a realm hierarchy
  - Hierarchy follows internet names
    → easy to find a path between realms
    → can filter cross-realm requests based on the names
  - Can add shortcut links or create even a fully connected graph between KDCs
  - E.g. Windows domain hierarchy

26

Password guessing attacks

- Kerberos is vulnerable to password guessing:
  - Sniffed KRB_AS_REQ or KRB_AS_REP can be used to test candidate passwords → offline brute-force password guessing
  - In Kerberos v4, anyone could request a password-encrypted TGT from AS → easy to obtain material for password cracking
  - Preauthentication in Kerberos v5 prevents active attacks to obtain material for password cracking → must sniff it
- Active vs. passive attacks
  - Misleading thinking:
    Attacker who can perform only passive attacks is weaker → vulnerability to weaker attackers is more serious → vulnerability to passive attacks is more serious
  - Reality:
    Active attacks can often be initiated by the attacker while passive attacks require attacker to wait for something to sniff → vulnerability to such active attacks is more serious

27

PKINIT

- Goal: take advantage of an existing PKI to bootstrap authentication in Kerberos
- Replaces the KRB_AS_REQ / KRB_AS_REP exchange with a public-key protocol
  - Public-key authentication and encryption to obtain TGT
  - Continue with standard Kerberos → transparent to TGS and application servers
- No password, so not vulnerable to password guessing
- Uses DSS signatures and ephemeral DH
- Windows 2000 and later, no standard [RFC 4556]

28

Using the session key

- Authentication at the beginning of a session is of little value unless session data is protected with the session keys
  - Attacker could not initiate sessions but is could sniff, modify and spoof session data
- Applications can use the Kerberos session key K_{AB} in any way they want
  - KRB_AP_REQ and KRB_AP_REP may include further key material (subkeys) that is sent encrypted under K_{AB}
- Kerberos provides special messages for integrity protection and encryption:
  - KRB_SAFE: data, T_A, SN, $addr_A$, $addr_B$, $MAC_{KAB}(\ldots)$
  - KRB_PRIV: E_{KAB}(data, T_A, SN, $addr_A$, $addr_B$)
  - Access to these functions happens often through GSSAPI (called SSPI in Windows)
- Another message KRB_CRED for sending credentials (ticket and secret key) for the purpose of delegation
- Applications need to be "Kerberized" to use Kerberos for authentication

29

Delegation

- Server may need to perform tasks independently on the client's behalf, e.g.
  - Recursive RPC
  - Agents operating when the user is no longer logged in
  - Batch processing at night
- Alice can give her TGT or service ticket and key to David
- Controlling the use of delegated rights:
  - Delegate only a service ticket, not TGT
  - Ticket may specify allowed the client IP addresses
  - Authorization-data field in ticket may contain app-specific restrictions
- Flags related to delegation:
  - FORWARDABLE flag in TGT: can be used to obtain a new TGT with different IP addresses
  - PROXIABLE flag in TGT: can be used to obtain service tickets with a different IP address
- Delegation of identity
  - When B has A's ticket and key, B can act as A and nobody can tell the difference → difficult to audit access; similar to sharing passwords

30

Kerberos in Windows domains

Thanks to Dieter Gollmann

Windows access control summary

- Two kinds of access rights: privileges and permissions
- The O/S stores security attributes for each processes (subject) in a token
- Token contains a list of privileges and a list of SIDs (i.e. user and group identifiers).
- The privileges are the union of all privileges assigned to the SIDs on the local machine. The list is created at login time
- Permissions are decided by comparing the list of SIDs against a DACLs on an object

Accessing objects across network

- Alice is logged on her local machine (client) and wants to access resources (e.g. email) on a remote machine (email server)
- Resources on the server are managed by a Windows service (daemon process) on the server
- Alice is running software (e.g. email client) that uses remote procedure calls (RPC) to access the remote resources on the server
- How does Windows allow and control access to such remote resources?

Network credentials

- Alice's user name, SID and network credentials (password) are cached on the client
- Alice's processes can use her network credentials for remote login

→ Authenticated access to network servers is mostly transparent to Alice

- Some applications ask for a different user name and credentials and store them separately
- Authentication protocols like Kerberos do *not* reveal the credentials (password) to the server, only temporary keys

Observations

- The service running on the server controls access to stored emails there
- Alice trusts the client machine to store her password, and her client software to use it for remote login
  - Thus, Alice must have high confidence in the client machine and the software she runs there
- Alice's password is used to authenticate Alice to the server. However, the server does not learn the password and cannot later pretend to be Alice
  - Thus, Alice only trusts the server to manage her email. She does not need to trust the server for anything else
- The server requires Alice to login just as if she were at the server console
  - The server does not trust the client machine at all (cf. Unix trusted hosts mechanism)

Tokens and remote access

- Tokens are meaningful only to the local machine and cannot be sent over network
  - The server does not trust the client machine to tell who Alice is and which groups she belongs to
- Instead, the client authenticates Alice to the server using her network credentials. The server creates a new login session and a new token (on the server) for Alice
- The service may now assign the token to a process or thread (impersonation) or implement its own access control based on the token contents

Network authentication

- Windows supports two authentication protocols:
  - NTLM: legacy protocol from Windows NT
  - Kerberos V5: implements RFC 1510
- The authentication protocols also
  - Provide the server with Alice's user and group SIDs
  - Produce a session key for protecting data between the client and server
- The session protocol is different for network logon, RPC, COM. Encryption and authentication are controlled by applications

37

Kerberos in Windows

- Realm = Windows domain
- Realm hierarchy = domain hierarchy
- KDC = domain controller (DC)
  - Information about users is stored in active directory (AD)

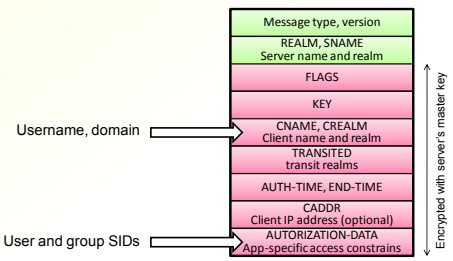
38

Kerberos and SIDs

- Kerberos authenticates 'principals', but which principals should be authenticated?
  - User name and a domain name (e.g. EUROPE\tuomaura)? The appropriate fields in the ticket for this are CNAME and CREALM
  - Principals according to the access control model? Windows puts the user SID and group SIDs in the optional field authorization-data
- Controversy over proprietary extensions, interoperability and standards compliance
- General remark on standards: options are there to be used but cause incompatibilities

39

Kerberos ticket in Windows



40

Delegating Kerberos credentials

- Alice needs a service from Bob, where Bob has to access servers on her behalf
  - For example, a print server needs to access Alice's email and files on file server to complete her printing jobs
- Alice applies for a proxyable ticket for the relevant servers and gives the ticket and corresponding session key to Bob

41

Exercises

- How to attack the Needham-Schroeder secret-key protocol if the encryption is no integrity-protecting, e.g. $E_K(M)$ = AES-ECB$_K$(M) ?
- Read about the Yahalom and Otway-Rees protocols. Can you find any flaws by yourself?
- Model the Needham-Schroeder shared-key protocol in Proverif
- How would you model identity or DoS protections with a tool like Proverif? (This is a difficult question.)
- Can you spot any (potential) vulnerabilities in the integrity algorithms used by older Kerberos implementations? See [RFC1510]
- Find source code for a Kerberized client/server application (e.g. telnet) and see how it accesses Kerberos services
- Why is Kerberos used on the intranets and TLS/SSL on the Internet? Could it be the other way?

42