8. Scalable Authentication
ENEE 757 | CMSC 818V

Prof. Tudor Dumitraș
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/757
https://www.facebook.com/SDSAtUMD

Today’s Lecture

• Where we’ve been
  – Security basics
  – Passwords
  – Biometrics

• Where we’re going today
  – Scalable authentication and Kerberos
  – Introduction to authorization logic

• Where we’re going next
  – More applications of authorization logic
Many-to-Many Authentication

How do users prove their identities when requesting services from machines on the network?

Naïve solution: every server knows every user’s password
- Insecure: break into one server ⇒ compromise all users
- Inefficient: to change password, user must contact every server

Access Control
- Principal makes a request for an object
- Reference monitor grants or denies the request

Ex: File editor Send file File server
Ex: Host Forward packet Firewall

- **Authentication**: Determining who made request
- **Authorization**: Determining whether requestor is trusted to access an object
  - This is the decision that the reference monitor must make
Authenticating a Channel

• Each request arrives on some channel, e.g.,
  – Kernel call from a user process
  – Network connection
  – A channel defined by a cryptographic key
• Reference monitor must authenticate the channel, i.e., determine whom the request is from
• Easy in a centralized system
  – OS implements all channels and knows the principal responsible for each process
• Harder in a distributed system
  – Request may have traversed different, not-equally-trusted machines
  – Different types of channels
  – Some parts of the system may be faulty or broken

Requirements

• Security
  – ... against attacks by passive eavesdroppers and actively malicious users

• Transparency
  – Users shouldn’t notice authentication taking place
  – Entering password is Ok, if done rarely

• Scalability
  – Large number of users and servers
Threats

• User impersonation
  – Malicious user with access to a workstation pretends to be another user from the same workstation

• Network address impersonation
  – Malicious user spoofs network address of his workstation to impersonate another workstation

• Eavesdropping, tampering, replay
  – Malicious user eavesdrops, tampers, or replays other users’ conversations to gain unauthorized access

Who Can You Trust?

• Where is the request “from”?
  – The user? The workstation? The application? All of the above?
  – Which of these things do you trust?
Our Approach to Studying the Problem

- Explain authentication and access control using an authorization logic
  - We’re going to use a simpler version

- The logic forces us to make assumptions explicit and teaches us how to think about access control

- Logic helps us to reason about principals and the statements they make

- Principals can be
  - Keys
  - People
  - Machines
  - Principals in roles
  - Groups
  - ...

Trusted Computing Base (TCB)

- Logic will help us identify the “trusted computing base”, i.e., the collection of hardware and software that security depends on
  - Compromise or failure of a TCB element may result in an incorrect “Yes” access-control decision

- Thus, TCB should be as small as possible
  - Must be carefully tested, analyzed and protected

- Benign failure of an untrusted (non-TCB) element may produce more “No” answers, not more “Yes” ones
  - This is called “fail secure” or “fail safe”

- Ex: An untrusted server holding a digitally signed credential
  - Failure prevents credential from being retrieved (more “Nos”)
  - Cannot undetectably modify the credential (due to the signature)
A Logic of Authorization

- The logic is inhabited by
  - Terms that denote principals and strings
  - Formulas that are either “true” or “false”

- Terms:
  \[ t ::= s \mid p \]
  \[ p ::= \text{key}(s) \mid p.s \]
  where \( s \) ranges over strings and \( p \) over principals

- Formulas:
  \[ \phi ::= s \text{ signed } \phi' \mid p \text{ says } \phi' \]
  \[ \phi' ::= \text{action}(s) \mid p \text{ speaksfor } p \mid \text{delegate}(p, p, s) \]
  where \( s \) ranges over strings and \( p \) over principals

A Logic of Authorization – cont’d

- Inference rules

\[
\frac{\text{keystr signed } F}{\text{key(keystr) says } F} \quad \text{(says-I)}
\]

\[
\frac{\text{A says } (A.S \text{ says } F)}{A.S \text{ says } F} \quad \text{(says-LN)}
\]
A Logic of Authorization – cont’d

• Inference rules

If $F$ is true, then we can consider that any principal says $F$

\[
\frac{F}{\text{A says } F} \quad \text{(says-12)}
\]

Modus ponens

\[
\frac{\text{A says } (F \rightarrow G) \quad \text{A says } F}{\text{A says } G} \quad \text{(impl-E)}
\]

A delegates authorization to B

\[
\frac{\text{A says } (B \text{ speaks for } A) \quad B \text{ says } F}{\text{A says } F} \quad \text{(speaksfor-E)}
\]

Limited delegation: A authorizes B to approve action $U$

\[
\frac{\text{A says } (B \text{ speaks for } A.S) \quad B \text{ says } F}{A.S \text{ says } F} \quad \text{(speaksfor-E2)}
\]

\[
\frac{\text{A says delegates } (A, B, U) \quad B \text{ says action } (U)}{\text{A says action } (U)} \quad \text{(delegate-E)}
\]
Authenticated Channels

- Reference monitor receives request \textit{C says action(s)} on channel C.
- Reference monitor must prove that \textit{A says action(s)} for some named principal A on the access control list.
- Credentials typically come from “certification authorities” (CA)
  (more on this later).
- CA issues statements of the form

\[ K_{CA} \text{ signed } (\text{key}(K_A) \text{ speaksfor } \text{key}(K_{CA}).A) \]

- Example: \( K_{\text{Verisign}} \text{ signed } (\text{key}(K_{\text{Microsoft}}) \text{ speaksfor } \text{key}(K_{\text{Verisign}}).\text{Microsoft}) \)

A Certification Authority

\[ K_{CA} \text{ signed } (\text{key}(K_A) \text{ speaksfor } \text{key}(K_{CA}).A) \]

\[ K_A \text{ signed action(resource)} \]

\text{Infers } \text{key}(K_{CA}).A \text{ says action(resource)} \]
An Example Proof

1. $K_{CA}$ signed (key($K_A$) speaksfor key($K_{CA}).A$)
2. $K_A$ signed action(resource)

3. key($K_{CA}$) says (key($K_A$) speaksfor key($K_{CA}).A$) says-I (1)
4. key($K_A$) says action(resource) says-I (2)
5. key($K_{CA}).A$ says action(resource) speaksfor-E2 (3,4)

Kerberos

- User should not be able to access server without first proving his identity to authentication service
- Ticket proves that user has authenticated
  - Authentication service encrypts some information with a key known to the server (but not the user!)
  - The only thing the user can do is pass the ticket to the server
  - Server decrypts the ticket and verifies information
Meet Kerberos

What Should a Ticket Include?

- User name
- Server name
- Address of user’s workstation
  - Otherwise, a user on another workstation can steal the ticket and use it to gain access to the server
- Ticket lifetime
- A few other things (session key, etc.)
Two-Step Authentication

- Prove identity once to obtain a special TGS ticket
- Use TGS to get tickets for any network service

Symmetric Keys in Kerberos

- $K_c$ is long-term key of client C
  - Derived from the user’s password
  - Known to the client and the key distribution center (KDC)
- $K_{TGS}$ is long-term key of TGS
  - Known to KDC and the ticket granting service (TGS)
- $K_v$ is long-term key of network service V
  - Known to V and TGS; each service V has its own long-term key
- $K_{c,TGS}$ is short-term session key between C and TGS
  - Created by KDC, known to C and TGS
- $K_{c,v}$ is short-term session key between C and V
  - Created by TGS, known to C and V
“Single Sign-On” Authentication

User

- Password

Password

K_c

Decryption

K_c

Encrypted with
K_c and obtains
K_TGS and
ticket_TGS

Implicit authentication:
only someone who knows K_c can decrypt

• Client only needs to obtain TGS ticket **once** (say, every morning)
• Ticket is encrypted; client cannot forge it or tamper with it

kinit program (client)

ID_c, ID_TGS, time_c

Encrypt (K_TGS, ID_TGS, time_KDC, ticket_TGS)

Key Distribution Center (KDC)

Encrypt (K_TGS, ID_c, Addr_c, ID_TGS, time_TGS, lifetime)

Client will use this unforgeable ticket to get other tickets without re-authenticating

1. key(K_TGS) says (key(K_TGS, TGS) speaksfor key(K_TGS, C))

Key = K_TGS

Key = K_c

All users must pre-register their passwords with KDC

Obtaining a Service Ticket

User

Knows K_TGS and ticket_TGS

System command, e.g. "lpr –Pprint"

Encrypted(K_TGS(ID_c, Addr_c, time_c))

Proves that client knows key K_TGS contained in encrypted TGS ticket

ID_c, ticket_TGS, auth_TGS

1. key(K_TGS) says (key(K_TGS, TGS) speaksfor key(K_TGS, C))

Speaksfor-E2(1,2):

2. key(K_TGS) says (key(K_TGS, TGS) speaksfor key(K_TGS, C))

On ACL for service V

ID_c

Encrypt(K_TGS(ID_c, ID_TGS, time_TGS, lifetime, ticket_TGS)

Fresh key to be used between client and service

Encrypt(K_TGS(ID_c, ID_TGS, time_TGS, lifetime, ticket_TGS)

Client will use this unforgeable ticket to get access to service V

1. key(K_TGS) says (key(K_TGS, TGS) speaksfor key(K_TGS, C))

2. key(K_TGS) says (key(K_TGS, TGS) speaksfor key(K_TGS, C))

3. key(K_TGS) says (key(K_TGS, TGS) speaksfor key(K_TGS, C))

4. key(K_TGS) says (key(K_TGS, TGS) speaksfor key(K_TGS, C))

• Client uses TGS ticket to obtain a service ticket and a short-term session key for each network service (printer, email, etc.)
Obtaining Service

5. $\text{key}(K_{c,v})$ says $(\text{Addr}_c, \text{speaksfor key}(K_{c,v}).C)$

Encrypt$_c$(ID$_c$, Addr$_c$, time$_c$)

Proves that client knows key $K_{c,v}$ contained in encrypted ticket

4. $\text{key}(K_{c,v})$ says $(\text{key}(K_{c,v}).\text{speaksfor key}(K_{c,v}).C)$

6. Addr$_c$ says $\text{lpr} -P\text{print}$

Speaksfor-E(Speaksfor-E2(4,5), 6):

7. $\text{key}(K_{c,v}).C$ says $\text{lpr} -P\text{print}$

V accepts requests from $\text{key}(K_{c,v}).C$

Encrypt$_c$ (time$_c$+1)

• For each service request, client uses the short-term key for that service and the ticket he received from TGS

Sources

• Various slides from Mike Reiter and Vitaly Shmatikov

Review of Lecture

• What did we learn?
  – Kerberos
  – Authorization logic

• Paper discussion: “Designing an Authentication System”
  – Discussion lead: BumJun
  – Scribe: Octavian

• What’s next?
  – More applications of authorization logic