9. Authorization Logic
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
  – Authorization logic

• Where we’re going next
  – Web authentication
Group Projects

• Group project proposal
  – Post proposal on Piazza by end of today
  – Indicate team members

• Work for Checkpoint #1 (October 26)
  – Survey of related work: What gap is your project addressing?
  – Propose a concrete list of tasks for checkpoint #1
    • Post as part of project proposal or as a Piazza followup

• Projects that did not establish feasibility during the pilot phase
  – These projects are now behind schedule
  – Aim to catch up with the other teams by Checkpoint #1
  – Target: write workshop-quality paper

Observations from Past Projects

• Don’t just show random plots and results, or reference literature without showing how it applies to the problem

• Do present a coherent experimental design
  – Research questions that make sense for the problem studied
  – A set of experiments to investigate these ideas
  – Results that support or refute each research question
    • Depth is more important than breadth
    • Discuss negative results – important to understand what doesn’t work

• Do explain clearly what you are doing
  – Do state the project goals
  – Don’t reference terms not defined or techniques not introduced/motivated

• Do show meaningful effort in order to decide where to go next
  – If the only feedback I can give is “do more work”: wasted opportunity
Observations from Past Projects – cont’d

• I will ask you if you are on track to write a workshop paper
  – Question is not rhetorical
  – Of course you need to investigate more until you find the best approach
  – You may also find a negative result: the approach you were trying to develop doesn’t work
  – However, you must show that you have explored the design space exhaustively
  – The negative result can’t be because you haven’t really tried

• Think about what grade you would give yourself, given the amount of effort you dedicated to the project

• Team members will not necessarily receive the same grade
  – Grades will reflect each student’s effort

Authorization (a.k.a. Access Control)

• We’ve seen:
  – 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
Authorization Logic
[Lampson, Abadi, Burrows and Wobber, TOCS'92]

• Logical framework for authentication and access control decisions
  – Systems: Taos, Nexus, Grey, Penumbra
  – We'll use a simplified version in this class

• Principals:
  – Keys
  – People
  – Machines
  – Principals in roles
  – Groups
  – ...

• Each request arrives on some channel:
  – Kernel call from a user process
  – Network connection
  – A channel defined by a cryptographic key

Authenticating a Channel

• 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
Remember Nexus?

- Where is the request “from”?
  - The user? The workstation? The application? The network channel? The OS? All of the above?
  - Which of these principals do you trust?
- Many components manipulate user data
  - Want to reason about security properties based on all these components

The Logic

- Terms: denote principals and strings
  - String examples:
    - A public key
    - A request for performing an action
  - Principal examples:
    - key(keystr): a principal that corresponds to the keystr string
    - p.s: the principal referred to as s by p

- Formulas: have a truth value (true or false)
  - action(s): request to perform the action described by string s
  - keystr signed msg
  - p says msg
  - p speaksfor q
  - delegate(p, q, s)
The Logic – cont’d

- Inference rules

\[
\text{keystr} \quad \text{signed} \quad F
\]

\[
\text{key(keystr)} \quad \text{says} \quad F
\]

\[
A \quad \text{says} \quad (A.S \quad \text{says} \quad F)
\]

\[
A.S \quad \text{says} \quad F
\]

Translate digital signatures into says statements

Note the difference from:
- B.S says F
- S says F

The Logic – cont’d

- Inference rules

\[
F
\]

\[
A \quad \text{says} \quad F
\]

\[
A \quad \text{says} \quad (F \rightarrow G) \quad A \quad \text{says} \quad F
\]

\[
A \quad \text{says} \quad G
\]

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

Modus ponens

impl-E
The Logic – cont’d

- Inference rules

\[
\begin{align*}
A \text{ says } (B \text{ speaks for } A) & \quad B \text{ says } F \\
A \text{ says } F
\end{align*}
\]

(speaksfor-E)

\[
\begin{align*}
A \text{ says } (B \text{ speaks for } A.S) & \quad B \text{ says } F \\
A.S \text{ says } F
\end{align*}
\]

(speaksfor-E2)

\[
\begin{align*}
A \text{ says delegates } (A, B, U) & \quad B \text{ says action } (U) \\
A \text{ says action } (U)
\end{align*}
\]

(delegate-E)

---

Authenticating a Channel

- Reference monitor receives a request \(C \text{ says action(s)}\)
- An access-control list usually specifies named principals
- Thus, reference monitor must collect credentials to prove that \(A \text{ says action(s)}\) for some \(A\) on the access control list
- Credentials typically correspond to statements of the form

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

- Two general methods
  - **Push**: The sender on the channel \(C\) collects \(A\)’s credentials and presents them to authenticate the channel to the receiver.
  - **Pull**: The receiver looks up \(A\) in some database to get credentials for \(A\) when it needs to authenticate the sender.
How to Implement an Authenticated Channel

- Credentials come from certification authorities (CA)
  \[ K_{CA} \text{ signed } (key(K_A) \text{ speaks for } key(K_{CA}).A) \]

- Two implementation approaches: \( K_{CA} \) can be a public key or a symmetric key

- Symmetric cryptography
  - Shared secret between sender and receiver: authentication messages must be sent and received in real time
  - Need an online relay
  - Example: Kerberos Ticket Granting Service translates credentials issued by the Key Distribution Center into messages for various kerberized services

- Public key cryptography
  - Broadcast channel: you can send a signed message without knowing who will receive it
  - Can store signed certificates locally
  - Example: Web certification authorities

Proof Needed to Authenticate the Channel

- Reference monitor receives a request \( K_A \text{ signed } \text{action(resource)} \)
  - Request arrives on the channel defined by the cryptographic key \( K_A \)
  - Reference monitor also has a credential \( K_{CA} \text{ signed } (key(K_A) \text{ speaks for } key(K_{CA}),A) \)

- Proof
  1. \( K_{CA} \text{ signed } (key(K_A) \text{ speaks for } key(K_{CA}),A) \)
  2. \( K_A \text{ signed } \text{action(resource)} \)
  3. \( key(K_{CA}) \text{ says } (key(K_A) \text{ speaks for } key(K_{CA}),A) \) \text{ says-I(1)}
  4. \( key(K_A) \text{ says } \text{action(resource)} \) \text{ says-I(2)}
  5. \( key(K_{CA}),A \text{ says } \text{action(resource)} \) \text{ speaks-for-E2(3,4)}

- Note that the authenticated principal is \( key(K_{CA}),A \) (not \( A \))
Groups

• A group is a principal whose members speak for it

• Simplest way to define a group $G$ is for a defining CA to issue certificates

  $\text{key}(K_{CA})$ says $P_1$ speaks for $\text{key}(K_{CA}).G$

  $\text{key}(K_{CA})$ says $P_2$ speaks for $\text{key}(K_{CA}).G$

  ...

  for group members $P_1, P_2, ...$

Example Proof

Tudor says (BumJun speaks for Tudor.Students)  BumJun says action(AW 3425)
Tudor says delegate(Tudor, Tudor.Students, AW 3425)  Tudor.Students says action(AW 3425)

Tudor says action(AW 3425)

Stored in the reference monitor.
Part of the TCB.
Traditional Access Control Lists

Implicitly known to the reference monitor.
BumJun speaks for Tudor.Students

\[\text{delegate(Tudor, Tudor.Students, AVW 3425)}\]

Not signed (part of TCB)

Received in the request.
BumJun says action(AVW 3425)

Tudor.Students says action(AVW 3425)

Tudor says action(AVW 3425)
Stored in the reference monitor.
Part of the TCB.

A “Pull” Approach

Retrieved by reference monitor.
Tudor says (BumJun speaks for Tudor.Students)

Tudor says delegate(Tudor, Tudor.Students, AVW 3425)

Credentials stored on server that is not part of TCB

Received in the request.
BumJun says action(AVW 3425)

Tudor.Students says action(AVW 3425)

Tudor says action(AVW 3425)
Stored in the reference monitor.
Part of the TCB.
A “Push” Approach

Received in the request.

Tudor says (BumJun speaks for Tudor.Students)

Tudor says delegate(Tudor, Tudor.Students, AVW 3425)

BumJun says action(AVW 3425)

Tudor says delegate(Tudor, Tudor.Students, AVW 3425)

Tudor.Students says action(AVW 3425)

Tudor says action(AVW 3425)

Stored in the reference monitor.
Part of the TCB.

A “Proof Carrying” Approach

[Appel & Felten, CCS'99]

Received in the request.

Tudor says (BumJun speaks for Tudor.Students)

Tudor says delegate(Tudor, Tudor.Students, AVW 3425)

BumJun says action(AVW 3425)

Tudor.Students says action(AVW 3425)

Tudor says action(AVW 3425)

Stored in the reference monitor.
Part of the TCB.
A Principal in a Role

• Suppose a principal wants to **limit** its authority
  – Tudor “as” GamePlayer
  – Tudor “as” SysAdmin

• Intuition: A “as” R should be weaker than A
• A can accomplish this by enabling statements of the form

  \[ A.R \textit{ says } F \]

...to be created

---

Programs as an Application of Roles

• Acting in a role is like acting according to some program
• If host H is running program with binary code I, then H can make

  \[ H.I \textit{ says } F \]
  – for a statement F made by the process running I

• Instead of using the whole program I, H can instead make

  \[ H.D \textit{ says } F \]
  – where D = h(I) for h a cryptographic hash function, and using

  \[ D \textit{ speaksfor } P \]
  – where P is the program name
Loading Programs Securely

• To load program named $P$, on host $H$
  — Creates a process $pr$
  — Reads text $I$ of file $P$ from the file system
  — Finds credentials for $D$ speaksfor $P$ and checks $h(I) = D$
  — Copies $I$ into $pr$
  — Gives $pr$ ability to write to channel $C$
  — Emit: $H$ says ($C$ speaksfor $H.P$)

• Now $pr$ can issue requests on channel $C$
  — Will be granted if $H.P$ is on ACL

Parasitic Virus Control

• Some viruses alter texts of programs in the file system
  — If $I'$ is the infected program text, then $D' = h(I')$ will be different from $D = h(I)$, and so $D$ speaksfor $P$ will not apply

• Certification authority CA can issue certificates

  \[
  \begin{align*}
  &K_{CA} \text{ signed } P \text{ speaksfor key}(K_{CA}).\text{trustedSW} \\
  &K_{CA} \text{ signed } N \text{ speaksfor key}(K_{CA}).\text{trustedNodes} \\
  &K_{CA} \text{ signed } (P \text{ speaksfor key}(K_{CA}).\text{trustedSW} \rightarrow \\
  &\hspace{1cm} (N \text{ speaksfor key}(K_{CA}).\text{trustedNodes} \rightarrow \\
  &\hspace{2cm} N.P \text{ speaksfor key}(K_{CA}).\text{trustedNodeAndSW}))
  \end{align*}
  \]
  — where trustedSW, trustedNodes, and trustedNodeAndSW are group names, $P$ is a program name, and $N$ is a node name

• The CA may revoke program $P$'s credential: the ACLs do not have to change
Secure Booting

- ‘trustedNodes’ should be computers that
  - run operating systems validated before booting
  - validate other software before loading it

- Validating OS during boot is like validating other software
  - Machine W holds h(I) in boot ROM, where I is OS image
    - i.e., h(I) speaks for OS

- To create a channel C such that C speaks for W.OS, W can
  - Generate a new signature key pair <K_{W.OS}, K^{-1}_{W.OS}>, and
  - Give K^{-1}_{W.OS} to OS, along with K_{W} signed key(K_{W.OS}) speaks for key(K_{W}).OS

- Private key for K_{W} must be protected in secure hardware (e.g. TPM)
  - Otherwise, OS can use it to sign credentials

TCG Integrity Challenge and Response

- Each TPM is equipped with a different private key K^{-1}_{TPM} and a certificate
  \[ K_{TPM} \text{ signed (key}(K_{TPM}) \text{ speaks for key}(K_{TPM}).\text{TrustedProcessingModules) } \]
  - signed by a “trusted platform module entity” (TPME)
  - TrustedProcessingModules is a group

- Remote machine can query TPM for contents of PCRs

- TPM responds with signed PCR values
  - Think of it as signed with K_{TPM}
    \[ K_{TPM} \text{ signed PCRvals = ... } \]
  - (In reality, is not signed with K_{TPM} but another Attestation Identification Key (AIK) is used to enhance privacy)
Review of Lecture

- What did we learn?
  - Authorization logic
  - Inference rules
  - Applications

- Sources
  - Mike Reiter

- What’s next?
  - Web authentication

- Deadline reminder
  - Post group project proposal by end of today