Systems Security
Chapter 5: Security Architectures

Winfried E. Kühnhauser
Summer Term 2016

Winfried E. Kühnhauser
CSI
Ilmenau University of Technology
www.tu-ilmenau.de
Roadmap

1. Security Requirements
3. Security Mechanisms
4. Security Architectures
**Terms**

**TCB** (*Trusted Computing Base*)
The set of functions of a system that are necessary and sufficient for implementing its security properties

**Security Architecture**
The part of a system architecture that implements its TCB

**Security Mechanisms**
Algorithms and data structures that implement functions of a TCB
Security architectures have been around for a long time …
Architecture Components
- Buildings, walls, windows, doors, moats, draw bridges

Architecture
- Component arrangement and interaction

Goal of a Security Architecture
Build a stronghold such that security policies can be enforced
- Presence of necessary components/mechanisms
- Total mediation
- Tamperproofness
→ architecture design principles
An Ugly Example
Access control implementation in Linux

Monolithic OS architecture

Application Programmer’s Interface (API)
... (approx. 250 system calls)

open(filename, openmode)
{ ino=namei(filename);
  switch ino.objecttype of
    file:
      if (filesystem.checkpermission(ino,openmode))
        { fd=filesystem.open(ino, ...) } ...
    socket: ...
}

: involved system components

Check your trust in
  - Completeness of control (and its verification!)
  - Tamperproofness (and its verification!)
  - Correctness (and its verification!)

§ Completeness of control (and its verification!)
§ Tamperproofness (and its verification!)
§ Correctness (and its verification!)

... (approx. 250 system calls)
Problem Areas

- PDPs (policy decision points)
- PEPs (policy enforcement points)

are

- Distributed among many OS components
- Not robust
  - Not isolated from errors within the entire OS implementation
  - Especially in dynamically loaded OS modules
... and that is not all

Linux set-user-id programs

- Execute security-sensitive operations
- Adopt different effective user id during execution
- Thus have different privileges
- Especially those of user id “root”

→ each such program is part of the TCB!

e.g.

- /bin/mount
- /opt/google/chrome/chrome-sandbox (Author: Google!)
- /usr/lib/virtualbox/VirtualBox (Author: Oracle!)

Still wondering about today’s security problems?
Problem

■ OSes/Middleware are big
■ Only a small part belongs to the TCB

→ Security architecture design such that TCB functions are collected in an
  ◆ not bypassable (complete control)
  ◆ isolated (tamperproofness)
  ◆ trustworthy (correctness)
  core
5.1 Architecture Design Principles

Goal
- Complete
- Tamperproof
- Provable

control of all security-relevant actions in a system

Approach
Definitions of fundamental architecture design principles
There Exists an Architecture Component that is

(RM 1) Involved in any subject/object interactionen
   → total mediation property

(RM 2) Well isolated from the rest of the system
   → tamperproofness

(RM 3) Small and well-structured enough to analyze correctness by formal methods
   → verifyability

A security architecture component built along these principles:
“Reference Monitor“
Idea

OS Application Programmer’s Interface (API)

... (approx. 250 system calls)

open(filename, openmode)
{
  ino=namei(filename);
  switch ino.objecttype of
    file:
      if (filesystem.checkpermission(ino, openmode))
        { fd=filesystem.open(ino, ...); ... }
      socket: ...
}

5.1 Architecture Design Principles
Reference Monitor

■ Core component of a TCB

■ Typically encloses
  ◆ Security policy implementation(s) (PDP)
    • Model state (e.g. ACM, subject set, entity attributes)
    • Model logic (e.g. authorization scheme)
  ◆ Enforcement mechanisms: PEPs

■ Typically excludes (due to complexity and size, RM 3)
  ◆ Authentication
  ◆ Cryptographic mechanisms
  ◆ Sometimes also model state (e.g. ACLs)
Consequences of (RM 3) for TCBs

- Small size, few functions
- Low complexity, simple functions
- Strong isolation
- Precise Perimeter
Highway to Hell: What is the
- TCB
- TCB Implementation
- Security architecture
- Observance of reference monitor principles
of
- Contemporary IT Systems
- Policy-controlled OSes
- Policy-controlled Applications
- Policy-controlled Middleware
- Microkernel Architectures
Policy-controlled OS – Monolithic OS Kernel
System-level Policy

Implementation Layers

5.1 Architecture Design Principles
Policy-controlled OS – Monolithic OS Kernel

TCB – Functional View

OS

Application Process
Application Process
Application Process
Application Process

File System

Policy Server

...
Policy-controlled OS – Monolithic OS Kernel

TCB – Implementation View

Implementation Layers
Policy-controlled OS – Microkernel Architecture (*Nizza*)
System-level Policy
Policy-controlled OS – Microkernel Architecture \((Nizza)\)

**TCB – Functional View**

- Microkernel
- Trusted Loader
- Trusted Name Server
- Trusted GUI
- Security Policy

Paravirtualized Legacy OS

Secure Application

Legacy Application
Policy-controlled OS – Microkernel Architecture (Nizza)

TCB – Implementation View

Paravirtualized Legacy OS

Microkernel

- Trusted Loader
- ... Trusted Name Server
- Trusted GUI
- Security Policy

Legacy Application

Secure Application

trusted
name server
trusted
gui
security
policy
Policy-controlled Application
Middleware-level Policy

\[ S = \{s_1, s_2, s_3\}, O = \{o_1, o_2, o_3\} \]

\[ \text{command} \ read(s, o) \]
\[ \text{if} \ read \in m(s, o) \]
\[ \text{enter} \ (s, o) \text{in } h; \]
\[ \forall o_i \in O, (o_i, o) \in C: \text{delete} \ read \text{from } m(s, o_i); \]
\[ \forall o_i \in O, o_i \neq o: \text{delete} \ write \text{in } m(s, o_i); \]
\[ \text{end if} \]
\[ \text{end} \]
AxisEngine

<table>
<thead>
<tr>
<th>Handler 1</th>
<th>Handler 2</th>
<th>\ldots</th>
<th>Handler n</th>
<th>Transport Sender</th>
</tr>
</thead>
</table>

\[ \text{instanciates} \]

\[ \text{out flow} \]

\[ \text{SOAP Request (by HTTP, JMS ...)} \]

\[ \text{instanciates} \]

\[ \text{MC}_{\text{req}} \]

\[ \text{instanciates} \]

\[ \text{MC}_{\text{resp}} \]

\[ \text{AxisEngine} \]

\[ \text{instanciates} \]

\[ \text{in flow} \]

\[ \text{MC}_{\text{resp}} \]

\[ \text{SOAP Reply} \]
5.1 Architecture Design Principles
TCB – Functional View

AxisEngine

Instanciates

OutInAxisOperation

Instanciates

Client

Stub

MC_{req}

MC_{resp}

SOAP Request
(by HTTP, JMS ...)

SOAP Reply

Handler 1

Handler 2

... Handler n

Transport Sender

in flow

out flow

TCB – Functional View

AxisEngine

Instanciates

OutInAxisOperation

Instanciates

Client

Stub

MC_{req}

MC_{resp}

SOAP Request
(by HTTP, JMS ...)

SOAP Reply

Handler m

... Handler 2

Handler 1

in flow

out flow
5.1 Architecture Design Principles
TCB – Implementation View
5.1 Architecture Design Principles
Policy-controlled Application

Application-level Policy

\[ S = \{ s_1, s_2, s_3 \}, O = \{ o_1, o_2, o_3 \} \]

... command \( read(s, o) \) if \( read \in m(s, o) \) enter \( s, o \) in \( h \); \( \forall o_i \in O, (o_i, o) \in C \) : delete \( read \) from \( m(s, o_i) \); \( \forall o_i \in O, o_i \neq o \) : delete \( write \) in \( m(s, o_i) \); end if end ...

5.1 Architecture Design Principles
Policy-controlled Application

TCB – Functional View

\[ S = \{s_1, s_2, s_3\}, O = \{o_1, o_2, o_3\} \]

...command read (s, o) if read \( \in m(s, o) \) enter (s, o) in h; \( \forall o_i \in O, (o_i, o) \in C \) delete read from \( m(s, o_i) \); \( \forall o_i \in O, o_i \neq o \) delete write in \( m(s, o_i) \); end if end

5.1 Architecture Design Principles
Policy-controlled Application

TCB – Implementation View

\[
\begin{align*}
S &= \{s_1, s_2, s_3\}, \\
O &= \{o_1, o_2, o_3\}, \\
\text{command} &\text{read}(s, o) \\
\text{if} &\text{read} \in m(s, o) \\
\text{enter} &\text{read} \text{in } h; \\
\forall o_i \in O, &\text{if } o_i \neq o \text{ then delete write in } m(s, o_i); \\
\end{align*}
\]
State of the Art

- Numerous rather weak implementations in
  - Commodity OSes
  - Middleware platforms
  - Applications

- Stronger approaches
  - In microkernel OSes (e.g. L4/Nizza)
  - In recent OSes (e.g. SELinux, Trusted BSD, Open Solaris)
Problem

Implementation of OS TCB such that

- Reference monitor principles
- The manyfold security requirements of applications are satisfied
5.2.1 Nizza
(TU Dresden)

Goals

- Security architecture with trustworthy TCB
  - Total access mediation (RM 1)
  - Tamperproof implementation (RM 2)
  - Small implementation (RM 3)

- Maintain functionality of
  - Contemporary legacy OSs
  - Legacy Applications
Concepts

- Reference monitor principles: Separation of
  - OS
  - Applications
  into security-critical and non-critical components
  → precise identification of a (minimal) TCB

- Maintain functionality
  - Paravirtualization of standard OSes
OS Layer

- Trustworthy microkernel
- Trustworthy basic services
- Not trustworthy (paravirtualized) legacy OS
Application Layer

- Vulnerability rises with growing complexity
  → reduce vulnerability of security-critical code by
- separation
- isolation of critical components

Example Email Client

- Non-critical: reading/writing emails
- Critical: signing emails
Putting it all Together

Email Client

Legacy Operating System

Network Driver  ...  Loader  GUI  Secure Storage

Microkernel

Enigmail Signer
Size of TCB: 105.000 LOC

- Microkernel (L4/Fiasco): 15.000 LOC
- Trustworthy basic services: 35.000 LOC
- Enigmail Signer: 55.000 LOC
Results

- Code size of TCB reduced by 2 orders of magnitude (100,000 LOC vs. 10,000,000 LOC)
- Functionality of legacy OSes and applications preserved
- Moderate performance penalties
5.2.2 Security Enhanced Linux

Goal
Security-aware OS
■ State-of-the-art OS
■ State-of-the-art security paradigms

Idea
■ Policy-controlled (Linux) OS kernel

Approach
SELinux =
■ Linux (DAC, IBAC)
■ MAC, RBAC, ABAC, TAM, MLS
Security Policies in SELinux (see also chap. 3.3.2)

- Application domain: AC policies for OSes
- Implementation by new OS abstractions (unheard of for > 40 years)
- Not entirely unlike process abstraction
  - Specification: process $\leftrightarrow$ security policy
    - Program: algorithm in formal language (C++, Java, ...)
    - Security model: rules in formal language (HRU, BLP, Skippy, ...)
  - RTE: process $\leftrightarrow$ security policy
    - Process management: RTE for application-level programs
    - Security Server: RTE for kernel-level policies
Policy-aware Security Server (*policy decision point*, PDP)
- Policy RTE in kernel’s protection domain

Interceptors (*policy enforcement points*, PEPs)
- Total interaction control in object managers (part of *Linux Security Module Framework*)
Reference Monitor Principles

- Total mediation of security-relevant interactions
  → placement of PEPs (LSMs)
- Tamperproofness of policy implementation
  → integration into kernel module (security server)

Policy Support

Remember: Security policies require

- Authenticity of entities: Unique subject/object identifiers
  → security identifier (SID)
- Policy-specific entity attributes
  → security context
- Approach: extensions to process/file/socket…-management
Authenticity of Entities

Object managers help: implement injective mapping $S \cup O \rightarrow \text{SID}$

- SID created by security server
- Mapping of SIDs to objects by object managers
Entity Attributes

Security policy implements injective mapping SID → security context

- security contexts creation according to policy-specific labeling rules
- Entry in SID→security context mapping table
Security Context contains

- Standard entity attributes such as
  - User Id
  - Role
  - Type (user_t, tomCat_t, ...)
  - Class (process, file, ...)

- Policy-specific entity attributes such as
  - Confidentiality/clearance level (BLP label)
Persistent Entities

Problem: Security contexts of persistent Entities

- Policies not aware of persistency of entities
  → persistency of security contexts is job of object managers

- Layout of object metadata (e.g. inodes in a file system) is file system standard
  → security contexts cannot be integrated in (policy-independent) inodes
Solution

- Persistent objects additionally have (OM-local) **persistent** SID: “PSID”
- OMs map these to SID
- 3 invisible storage areas (≈ files) in persistent memory implementing
  - Security context of file system itself (label)
  - Bijective mapping inode → PSID
  - Bijective mapping PSID → security context
Access Vector Cache (AVC)

- Located in object managers (user level) resp. in Security Server (kernel level)
- Caches access decisions
Motivation
- Old and weak security mechanisms in contemporary commodity OSs

Consequence
- Insufficient and inadequate specification paradigms
- Numerous vulnerabilities

Approach
New OS abstraction: security policies
→ policy-controlled OS: DAC & MAC, IBAC, RBAC, ABAC, TAM, MLS
Compliance with Reference Monitor Principles

1. Total Mediation Property (placement of PEPs)

   well ...  

   (currently) done manually
Compliance with Reference Monitor Principles

2. Tamperproofness of Policy Implementation

Fundamental problem in monolithic (SAS-) architectures

→ TCB implementation vulnerable to entire OS kernel code

- Security server
- All object managers
- Memory management
- IPC implementation
- I/O system

It can be done:

_hat Nizza
Compliance with Reference Monitor Principles

3. Verifiability

- Size and complexity of policy (reference policy ≈ 50,000 rules) → analysis tools
- Policy’s RTE claim to be universal
- Completeness of PEPs
- Policy isolation
5.3 Security Architectures of Distributed Systems

Scenario

- Client
- Object
- Distributed Middleware Platform
- OS Layer
Integration of Security Policies

Client

Client Stub with Proxy, here without attached policy

... writeAssignment(MyHomework);
...

... 
Msg.Subject=MyCredentials;
Msg.Method_Id=writeAssignment_id;
Msg.Params= MyHomework;
send(Server, Msg);
...

Msg: MyCredentials
   writeAssignment_id
   MyHomework
Policy Object

class OpenUniversityPolicy
{
  public:
    bool check(MsgType Msg)
    {
      if (s=CheckCredentials(Msg.MyCredentials))
      {
        o=Msg.MyHomework;
        switch (Msg.Method_Id)
        {
          case writeAssignment_id:
          {
            if (execWH_Right ∈ m[s,o])
              m[s,o]+=execRS_Right;
            return (execWH_Right ∈ m[s,o]);
          }
          case readSample_id:
          {
            if (execRS_Right ∈ m[s,o])
              m[s,o]-= execWH_Right;
            return (execRS_Right ∈ m[s,o]);
          }
          default: return false;
        }
      }
      else return false;
    }
  private:
    rightsetType m[SubjectType,ObjectType];
}
Integration of Security Policies

Example: Apache Axis-2 Webserver

5.3.2 Web Services
5.3.2 Web Services

AxisEngine

AxisServlet

HTTP/JMS
Transport Utilities

MC_{req}

MC_{resp}

SOAP Request

SOAP Reply

MC_{req}

MC_{resp}

Message Receiver

Web Service

AxisEngine

Handler 1

Handler 2

... 

Handler n

in flow

Handler m

... 

Handler 2

Handler 1

out flow
Summary Web Services SA

- PDPs?
- PEPs?
- Size of TCB?
- Reference Monitor Principles?
5.3.3 Kerberos

Authentication and authorization architecture for distributed systems with closed user groups

Scenario

■ Distributed system run by single organization
■ Workstations and Servers
■ 2 Kerberos servers
  ◆ Authentication Server (AS)
  ◆ Authorization Server (TGS)

■ MIT Athena project, started 1986
■ Concepts are part of CORBA‘s Basic CORBA Security Architecture specification
■ Originally created for: 650 workstations, 65 servers, 5000 users (1988)
Architecture Components

- **Authentication Server (AS)**
  - Authenticates users
    - Based on (symmetric) key shared between user and AS
    - Result: id card ("authenticator")
  - Authorizes use of TGS
    - Based on key shared between AS and TGS
    - Result: ticket (capability) for TGS

- **Ticket Granting Server (TGS)**
  - Issues tickets (capabilities) for all servers
    - Based on key shared between TGS and respective server
    - Result: ticket(s) for server(s)

- **Kerberos database**
  - Contains for each user and server a mapping
    user/server name → authentication key
  - Used by AS
  - Is multiply replicated (availability, scalability)
1. Authentication, request for TGS ticket
2. Id card, TGS-Ticket
3. Request for further server tickets
4. Server tickets
5. Service request: s decide based on
   - Id card of client
   - Server ticket
   - Local ACLs
Tickets

- Specify right of one client to use one server (personalized capability)
- Issued by *Ticket Granting Server*
- Limited lifetime (to make cryptographic attacks difficult)
  - \( \approx 1 \) day; balance between secure and convenient
    - Short: inconvenient but more secure (if ticket is stolen it soon expires)
    - Long: insecure but more convenient (no need for frequent renewal)
- Can be used multiply while valid
- Are sealed by TGS with key of server

\[
T_{\text{Client/Server}} = \{ \text{Client, Server, Client.Networkaddress, Timestamp, Lifetime, SessionKey}_{\text{Client/Server}} \}^K_{\text{TGS/Server}}
\]
Provisions against Misuse

- Tampering by client to fabricate rights for different servers
  \[\rightarrow\] provision: guarantee of integrity by MAC using \(K_{TGS/Server}\)

- Intercept ticket, use by third party
  \[\rightarrow\] provision: personalization by
    - Name and network address of client together with
      - Limited lifetime
      - Authenticator of client \(\rightarrow\)

\[T_{\text{Client/Server}} = \{\text{Client, Server, Client.Networkaddress, Timestamp, Lifetime, SessionKey}_{\text{Client/Server}}\}_{K_{TGS/Server}}\]
Inside Kerberos Authenticators

- **Authenticators**
  - Proof of identity of client to server
  - Fabricated by client using session key Client/Server; can be done only by
    - TGS (trust: it doesn’t do that)
    - Server (why should it?)
    - Client (without help by AS, because client knows session key (see below))
  - Can be used only once
    → prevent replay attacks by checking freshness
Inside Kerberos Authenticators

Note to Implementation

- Random/meaningless Nonce (instead of time stamp)
  - Precise implementation of single use expensive
    → bookkeeping in each server, lifelong

- Time stamp as Nonce
  - Provides limited (very short) lifetime authenticator
  - Requires synchronized clocks (for security reasons)

\[
A_{\text{Client}} = \{\text{Client, Client.Networkaddress, Timestamp}\} \text{ SessionKey}_{\text{Client/Server}}
\]
The Complete Process

Single Steps:
1. Alice tells her name

2. Alice´s workstation requests authentication

Alice

name, password

auth. request

TGS ticket with TGS session key

Kerberos AS

Alice

Alice

Alice, TGS

Kerberos AS

Alice, TGS session key

Alice, TGS

Kerberos AS

Alice, TGS

Kerberos AS
3. The AS

- Creates a fresh session key for Alice´s communication with the TGS:
  \[ SessionKey_{Alice/TGS} \]

- Creates freshness timestamp

- Creates Alice´s ticket for TGS and encrypts it with \( K_{AS/TGS} \) (so Alice cannot modify it):
  \[ Ticket_{Alice/TGS} = \{Alice, TGS, ..., SessionKey_{Alice/TGS}\}K_{AS/TGS} \]

- Encrypts everything with \( K_{Alice/AS} \) (so only Alice can read the session key and the TGS-Ticket)
4. Alice’s workstation

- Now has \( \{TGS, Timestamp, Lifetime, SessionKey_{Alice/TGS}, Ticket_{Alice/TGS}\}_{K_{Alice/AS}} \)
- Request Alice’s password:
  - Computes from password \( K_{Alice/AS} \) using a cryptographic hash function
  - Decrypts therewith message from AS

Result: Alice’s workstation has
- Session key for TGS session: \( SessionKey_{Alice/TGS} \)
- Ticket for TGS: \( Ticket_{Alice/TGS} \)
Bidirectional Authentication

2 Steps
1. Authentication of client to server
2. Authentication of server to client (optional)
1. Authentication of Client

Assumptions

- Alice has session key
- Alice has server ticket

1.a) Alice assembles authenticator

\[ A_{Alice} = \{Alice, Alice's\ network\ address,\ timestamp\}^{SessionKey_{Alice/Server}} \]

Only Alice can do that, because only she knows SessionKey_{Alice/Server}

1.b) Server decrypts ticket and thus gets session key; thus it can decrypt A_{Alice} and check:

- Freshness
- Compliance of names in ticket and authenticator
- Origin of message (as told by network interface) and network address in authenticator
2. Authentication of Servers

- Can only be done by principal that knows $\text{SessionKey}_{\text{Alice/Server}}$
- This can only be the server that is able to extract the session key from the ticket

$$\left\{\text{Timestamp} + 1\right\} \text{SessionKey}_{\text{Alice/Server}}$$

$$\text{Ticket}_{\text{Alice/Server}} = \{\text{Alice, Server, ..., SessionKey}_{\text{Alice/Server}} \}_K_{\text{TGS/Server}}$$
Tickets

- Are valid for a pair (client, server)
- Are issued (but for TGS-Ticket itself) only by TGS

Ticket request to TGS: (server, TGS ticket, authenticator)

TGS:

- Checks authenticator
- Generates session key for client and server: \( \text{SessionKey}_{\text{Alice/Server}} \)
- Generates ticket: \( \text{Ticket}_{\text{Alice/Server}} \)
- Encrypts both using shared session key

\[ \{ \text{Server}, \text{SessionKey}_{\text{Alice/Server}}, \text{Ticket}_{\text{Alice/Server}} \} \text{SessionKey}_{\text{Alice/TGS}} \]
Differences to non-distributed security architectures?

The architectures‘ achievements?

PDPs und PEPs?

Size of TCB?

RM principles?
Distributed Authentication and Authorization Architecture

- Security servers
  - Authentication
  - Authorization

- Cryptographic mechanisms for
  - Authentication
  - C+I of communication
  - I+A of tickets and authenticators

- Kerberos TCB: OS +
  - AS
  - TGS
  - Kerberos database
  - All PDPs and PEPs on the servers
  - Time server (NTP: signed time stamps!)
  - Partly also clients (are trusted to assemble correct authenticators)
5.4 Summary

TCBs
- Huge
- Inhomogeneous
- Distributed
- Hard to identify precisely

Security Architectures’ Problem
- Compliance to reference monitor principles

Case Studies
- Nizza
- SELinux
- CORBA, Web Services, Kerberos

→ Huge challenges!
Yes, we can

- Security Requirements
- Security Policies Modeling and Specification
- Security Mechanisms
- Security Architectures

Vulnerabilities and Risks
Policy Engineering
- Role of security policies
- Modeling
- Analysis
- Specification
Implementation
- TCBs
- Reference Monitors
- Security Architectures
What we did not look at but still is important

- Network security
- Cryptography: theoretical foundations and algorithms
- Technical aspects: firewalls, virus scanners, IDSs, ...
- Management aspects: organizational structures, backup procedures, ...
Policy Engineering
"Things should be fun to use"

- Security models
  (D)RBAC, (D)ABAC ... ; what comes next?
- Methods and tools
  model analysis, specification, implementation
- Multi-policy systems and policy coordination

Security Architectures

- Small, precisely perimetered TCBs
- Reference monitors that live up to their name
Research Seminar

- on Model Engineering

Master Theses

- Design and Implementation of Components for the Security Policy Engineering Workbench WorSE
  - Security Models
  - Analysis Algorithms
  - Specification Languages