Security

Stefan Kalkowski

Dresden, 2009-01-13
So far ... 

• Basics  
  ▪ Threads, Memory, Communication  
• Real time  
• Device Drivers  
• Resource Management  
• Virtualization / Legacy Container  
• **Security**
Outline

• Security
  • Detailed example: Mikro-SINA
  • Some definitions
  • Naming & capabilities
  • Security policies
  • Distributed & stacked policies
  • Bastei demo
Mikro-SINA
• VPN based security architecture by 'Bundesamt für Sicherheit in der Informationstechnik'
• IPSec and PKI
• Intrusion detection response
• Complex real world scenario
• Minimized & hardened Linux
Linux complexity

• Linux is complex!
• SLOC for kernel 2.6.18
  • All: 4,983,723
  • Architecture specific: 817,880
  • x86 specific: 55,463
  • Driver code: 2,365,256
  • Common: 1,800,587
• Running kernel > 2 millions lines of code
• Minimized & hardened version > 500,000 LOC
Project Mikro-SINA

• Goals:
  • Reduce Trusted Computing Base of VPN gateway
  • Enable high level evaluation (e.g.: Common Criteria)

• Objectives:
  • Confidentiality and integrity of sensitive network data inside the VPN

• Exploit microkernel features
IPSec – the heart of VPN security

- IPSec is a protocol suite for securing the Internet protocol
- Authentication header
  - Guarantees integrity and authentication
- Encapsulating Security Payload
  - Also confidentiality
- Two different modes
  - Tunnel vs. transport

Data Link Layer
IPSec
IP
TCP/UDP
Application

TU Dresden, 2008-01-13
MOS - Security
Slide 9 von 53
Easy approach

- Use paravirtualized L4Linux as a tool
- Identify the security relevant parts in the insecure application and separate them
- 'Viaduct' is used for authentication, key management and en- and decryption
• IP packets must be passed to the Viaduct
• Use TUN/TAP driver in Linux to tunnel IP
Fragmentation of IP packets

- Encrypted IP packets get fragmented
  - Problem for decryption
- IP packets have to be unfragmented before they are passed to the Viaduct
Let Linux do the dirty work ...

- Register new transport protocol
- IP stack will unfragment IP packets
What about confidentiality?

- L4Linux is untrusted
  - but handles encrypted and unencrypted data
- Use a second L4Linux instance
  - Each L4Linux instance drives its own ethernet card exclusively
Mikro-SINA so far

- 'Outer' L4Linux instance handles encrypted data only
- 'Inner' L4Linux instance handles unencrypted data and cannot communicate to the other instance or the 'outgoing' network device
Mikro-SINA: Multi Level Security

- Different security levels for data from different organizational levels
- Use an own L4Linux instance for each level, that handle the unencrypted data
Reduce untrusted code size

- Do we need two L4Linux instances for a VPN gateway?
  - Linux user level, process management, scheduling, file systems etc. are not necessary
- Use FLIPS (Flexible IP Stack): standalone TCP/IP stack
Techniques

• Fine grained isolation
  • Microkernel feature
  • Enables implementation of principle of least privilege

• Minimize complexity of TCB for security relevant code
  • Split applications in security relevant and nonrelevant parts
  • Use trusted wrappers (e.g.: Viaduct)
  • Minimize common code (microkernel approach)

• Usage of legacy code
Terms
System's security – a wide field

- Confidentiality, integrity and availability
- Security policies and mechanisms
- Cryptography: ciphers and keys
- Identity and authentication
- Access / information flow control
- Resource accounting
- Trust / assurance
- Formal methods: verification and evaluation
- Auditing and intrusion detection
Secure System*:  
- A secure system is a system that starts in an authorized state and cannot enter an unauthorized state.

Security Policy*:  
- A security policy is a statement that partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized, or nonsecure, states.

* Matt Bishop: Computer Security – Art and Science
Security Policy:
- A security policy is a statement of what is, and what is not allowed.
- e.g.: SELinux policy, /etc/exports

Security Mechanism:
- A security mechanism is a method, tool, or procedure for enforcing a security policy
- e.g.: Capabilities, ACLs, MMU ...
“Every program and every user of the system should operate using the least set of privileges necessary to complete the job.”

(Saltzer and Schroeder)
Naming
Naming & policy

- Naming issues are highly coupled with security
- Symbolic name resolution is highly policy dependent (files, objects, uri ...)
- References and access control belong together
- Names are resources, that have to be controlled
Local vs. global name spaces

- **Global name spaces:**
  - All instances share the same view
  - Classical in monolithic systems
  - Easy to configure

- **Local name spaces:**
  - Instances have a private name space
  - Forwards *principle of least privilege*
  - Common examples: BSD jails or chroot
Problems with global names

- Example: L4 thread ids are globally visible
  - Everyone can send IPC to everyone
  - Services need to care of access control
  - Denial of Service attacks are possible
  - No full isolation

- Simple solution: using local names
Local names example: Plan 9

- Developed by Bell Labs in the late 1980’s
- Distributed system, one UNIX out of a lot of systems
- Main features:
  - All resources are named and accessed like files
  - Network protocol 9P for remote file access
  - Per process, private hierarchical file name space
Local names example: Plan 9

- Services export file hierarchies
- Processes *mount* services they use into their own name space
- Processes might inherit the name space of their parent process
- In addition processes can use *bind* to duplicate paths in the file hierarchy
• Capabilities
  • Designate a specific object (e.g.: kernel object) and give certain access rights to that object
  • Possession of a capability is sufficient to access the concerning object
  • Can be implemented by using hardware support, memory protection mechanisms or cryptography
Capability systems

- Cambridge CAP computer
- IBM System/38
- KeyKOS, EROS, Coyotos
- Mach / GNU Hurd
- Amoeba
- TU-Dresden: L4.sec, Bastei, NOVA ...
Capability properties

• Capability models differ:
  • Originally: possession of a capability is sufficient to further delegate that capability
    -> complicates access control
  • Today: most capability systems have separate privileges for capability propagation

• Capability Lists vs. Access Control Lists:
Capabilities in practice

**L4.Sec (Florence):**

- Local names in a task local capability space translate to capabilities
- Capabilities reference kernel objects (e.g.: endpoints)
- Capabilities can be obtained by creating an object or by mapping/copying
- An additional identifier the *badge* is associated with each endpoint capability
- Badges enable the receiver to distinguish sender capabilities from each other
Capabilities to Endpoints

Make thread implementation details transparent

Client 1

Client 2

Server

Kernel

1456

1208

Badges

Objects
Models
Formal security model: Bell-LaPadula

- Subjects and objects have a *security label*, consisting of a *security level* and *category set*.
- A security label *dominates* another one, if its security level is greater or equal than the other one and its category set is a superset of the other one.

```
{National, Foreign}
{National}
{Foreign}
{}
```

- Top Secret
- Secret
- Confidential
- Unclassified
• Example: Label $L_1: (\text{Topsecret, \{National\}})$ dominates $L_2 (\text{Secret, \{\}})$

• Simple Security Condition: S can read O if S dominates O (no reads up)

• *-Property: S can write to O if and only if O dominates S (no writes down)

• Also known as multi-level security
Bell-LaPadula summary

• **Information flow** policy, that preserves confidentiality

• Very simple model, proof of model's security properties is trivial, practical proof is hard

• No integrity concerns in the model (use Biba)

• Shortcomings:
  - Some system's software (e.g. device drivers) has to be used by all different security levels
  - These parts are outside of the model
Decentralized information flow control

• Difficulty to incorporate application-specific policies in the system

• Shortcomings of centralized labels (like Bell-LaPadula)

➔ Solution: Decentralized label model
  • Applications can allocate tags for its objects
  • Each process possesses a set of tags it has potentially accessed – its label
  • Tainted processes can't access non-tainted
  • Examples: JFlow, Asbestos, HiStar ...
Access Control

• Information flow states how data can flow through the system
• In contrast access control states who can access what using which operations
• Prominent example: Access Control Matrix

**Discretionary access control:**
  • Privileged instance (e.g.: owner) related to an object decides who is allowed to access it

**Mandatory access control:**
  • System's rules, that cannot be altered by an individual user
• Type Enforcement
  • Abstracts from certain instances
  • Subjects and objects have *types*
  • Types can be compounded to other types
  • Explicit rules state what types can access (read, write) what other types

• SELinux is an example for TE in practice
SELinux

- Developed by the NSA
- Based on Linux Security Modules (LSM)
- Beside TE it provides Role-based AC (RBAC)
- Linux functionality leads to a complex policy
- Standard (NSA) policy
  - > 300 domains
  - \( \sim 50,000 \) rules
  - Tools might help to keep the overview
- Problem: monolithic kernel leads to monolithic reference monitor with one big policy inside
Multiple reference monitors

- Every task dominates its children with respect to *sessions* outside the child's own subtree

- Eases up multilateral security
  - Every organization unit can integrate its own policy
Bastei
Bastei

- Userland experiment within the ROBIN project

- Features:
  - Strict hierarchical structure
  - Core node provides basic services representing system's resources
  - Parent node acts as name server
  - Nodes have to pay for services they use

- Bastei is called Genode now, an commercial open source & community project:
  http://www.genode.org
• Announcing services:

- GUI
- Init
- Core

```
announce("GUI", root_cap)
```
Bastei: Open a session

• Using service:

```plaintext
session("GUI", "input=yes, label=xterm")

session("GUI", "label=bob->xterm, input=yes")

session("input=false, label=bob->xterm")
```
Demo
Summary

• How to construct safe systems with less effort
• Tools: application splitting, para-virtualization
• Capabilities
• Security policies:
  • A little bit about formal models (more in 'Distributed Systems' lecture)
  • Stacking of security policies (Bastei)
References


References

- Petros Efstatopoulos: 'Labels and Event Processes in the Asbestos Operating System' SOSP 2005
Next Lesson

- Following lesson
  - Practical exercise
  - Hacking Bastei

- Next week (20.01.)
  - Security part II