Security and security controls in operating systems

A quantitative approach
2015-02-16

Andreas Jonsson
andreas@romab.com

Robert Malmgren
rom@romab.com
I minute presentation

- Consultant in IT and infosec since 20+ years
- Working alot on with critical infrastrucutre protection, process control, SCADA security etc, but also in financial sector, government, etc
- Work covers everything from writing policies, requirement specs and steering documents to development, penetration testing, incident handling and forensics
Outline of talk

• Intro
• Background and basics
• Security problems & vulnerabilities
• Formal security models
• Example of operating systems and security
• Trends
Some short notes

- The focus is on general operating system used in general computers - COTS products
- *Embedded systems, code for micro controllers, etc often lack most fundamental security features*
- Some experimental OS’s and domain specific solutions have better-than-average security concepts and security controls, e.g. military grade usage
Background and basics
Modern software is normally formed into components, parts and layers in systems

Complex systems
- ...run multiple programs at once,
- ...have multiple users,
- ...store huge amounts of data,
- ...is interconnected via networks
Intro - foundation

- This is to build-in security into the foundation of the systems - the operating system
  - To identify and authorise users of the system
  - To allow for an environment where necessary basic controls are in place
  - To prevent unauthorised access to OS resources
Intro - *just the basic facts*

- All software is prone to bugs
- Some bugs will have an impact that can have security implications - data leaks, destruction of data, privilege escalations
Intro - *just the basic facts*

- Some bugs help to circumvent security mechanisms
- Some security designs are flawed, or build on flawed assumptions
Intro - *just the basic facts*

- Some bugs are undiscovered for some time, they lay latent
- Once discovered, they can be abused, if it is an *security vulnerability*, that can be *exploited*
- A discovered security bug, is sometime called a *0day*, until it is mitigated
Some concepts and principles

TCB - Trusted Computing Base

RBAC - Role Based Access Control

MAC - Mandatory Access Control

DAC - Discretionary Access Control

Principle of least privilege

Principle of least surprise
Operating system security

• Security problems in the operating system can affect the integrity of the system itself

• Someone else can control the system to their own liking - *pwnd*!

• Bugs in OS kernel can affect system integrity

• Security problems with the operating system can in turn affect the security in *applications* and *subsystems* (databases, middle ware, etc)

http://en.wikipedia.org/wiki/Pwn
### Capabilities and requirements

<table>
<thead>
<tr>
<th>Need</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protect a system resource</td>
<td>Prohibit malicious or unintentional access to system resources</td>
<td>System tables, direct access to I/O-units, memory protection</td>
</tr>
<tr>
<td>Authorization checks for usage of system calls and system resources</td>
<td>Provide controlled access to system, so that system maintain system integrity and provide continuous security to application and information</td>
<td>reference monitor</td>
</tr>
</tbody>
</table>
Some important concept

- Reference monitor
- Trusted Computing Base, TCB

**Principles for secure design**

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Economy of mechanism</strong></td>
<td>Keep the design as simple and small as possible</td>
</tr>
<tr>
<td><strong>Fail-safe defaults</strong></td>
<td>Base access decisions on permission rather than exclusion</td>
</tr>
<tr>
<td><strong>Complete mediation</strong></td>
<td>Every access to every object must be checked for authority</td>
</tr>
<tr>
<td><strong>Open design</strong></td>
<td>The design should not be secret</td>
</tr>
<tr>
<td><strong>Separation of privilege</strong></td>
<td>Technique in which a program is divided into parts which are limited to the specific privileges they require in order to perform a specific task</td>
</tr>
<tr>
<td><strong>Least privilege</strong></td>
<td>Every program and every user of the system should operate using the least set of privileges necessary to complete the job</td>
</tr>
<tr>
<td><strong>Least common mechanism</strong></td>
<td>Minimize the amount of mechanism common to more than one user and depended on by all users</td>
</tr>
<tr>
<td><strong>Psychological acceptability</strong></td>
<td>It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly</td>
</tr>
</tbody>
</table>

The classical ring model

**UNIX**

- Userland
- Kernel

**x86**

- Ring 3
- Ring 2
- Ring 1
- Ring 0
  - Kernel
  - Device drivers
  - Device drivers
  - Applications

Least privileges

Highest privileges

Interaction between application and OS

Exekverande process → Anrop till systemanrop → Fortsatt exekvering

Process i userland

Trap

Exekvera systemanrop

Kernel
Overview of operating system (1/2)
Overview of operating system (2/2)

Systemanrop in mot operativsystemsgrundläggningen

Operativsystemsgrundläggningar med basfunktioner

Gränssnitt mot hårdvara

<table>
<thead>
<tr>
<th>Nätverk</th>
<th>Minneshantering</th>
<th>Datalagring</th>
<th>CPU</th>
</tr>
</thead>
</table>
Problem with these pictures and concepts

- Layering violation
  - some software might skip a layer and call an underlaying layer directly and hence bypass controls

- In some scenarios attackers might come an unexpected way
  - Attacking from host operating system against guest operating systems in a virtual machine environment
Memory handling

- RAM memory is a central resource that in a controlled way must be shared and handled between operating system, applications and other components.
- Modern computer systems have hardware support for memory protection, e.g. MMU.
- OS support is required to use the hardware supported memory protection.
File system

• A file system is often a central component in a computer system w.r.t. security and protection

• Besides the actual file content, there is meta data that is of importance

• File owner, dates of creation/change/access, access information, security labels, etc

• Manipulation of meta data can in some cases be more serious security breach than the manipulation of the file content itself. Or a combo of both can be misleading and hide the fact that a file has been altered
<table>
<thead>
<tr>
<th>File system</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAT</td>
<td><em>No access control</em></td>
<td>Classic DOS</td>
</tr>
<tr>
<td>NTFS</td>
<td><em>Discretionary Access Control via ACL</em></td>
<td>Advanced possibilities to make controls</td>
</tr>
<tr>
<td>UFS</td>
<td><em>Discretionary Access Control, writing &amp; program execution for owner, group, “others”</em></td>
<td>Simple access controls</td>
</tr>
</tbody>
</table>
## Network file systems

<table>
<thead>
<tr>
<th>File system</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFSv3</td>
<td><em>Hostbaserad accesskontroll, uid</em></td>
<td>Trivial to circumvent</td>
</tr>
</tbody>
</table>
| NFSv4       | *Secure RPC, KRB5a, KRB5p, KRB5i* | Require a Kerberos server, KDC  
a= authentication  
i=integrity = calculate MAC  
p= privacy = encrypt packet |
| SMB/CIFS    | *KRB5a* | |

*KRB5a* denotes Kerberos for Windows-based file systems.
Comparing security in Operating systems (1/4)

• When in time was the system developed?
  • What was the state-of-the-art at that time?
  • What trends where currently in fashion?
Comparing security in Operating systems (2/4)

• Development methodologies

• Open Source or Closed Source?

• What support do one use to ensure that security is built into the product?

• How does one ensure that implementation is a correct representation of the design, that is a correct interpretation of the analysis?

"Given enough eyeballs, all bugs are shallow"
- Linus' Law
Comparing security in Operating systems (3/4)

But really, what good is this comparison?

Write more code = get higher salary?
Manage a 200K-SLOC project is cooler than a 5K-SLOC?

More code = more bugs?

More code = more security checks and advanced concepts like crypto, resilient failure checking built into everything?

But certainly, complexity is considered bad and evil in the context of security. And there is often a relation between complexity and size of program
Comparing security in Operating systems (4/4)

- What can one gain by having formal certification of operating systems, subsystems or application

- Trusted Computer System Evaluation Criteria (TCSEC), Common Criteria, etc

- More a theoretical excersice than of any real value?
Example of different protection solutions
## General example of control principles

<table>
<thead>
<tr>
<th>Security controls</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random numbers</td>
<td><em>Make a resource non-deterministic</em></td>
<td>File names, process ID’s, port numbers, session keys, transaction numbers, DNS query ID’s, timing</td>
</tr>
<tr>
<td>Encryption</td>
<td>Protection against eavesdropping or unauthorized access</td>
<td>network traffic, file content, disk partitions, memory pages, swap files/area</td>
</tr>
<tr>
<td>Hash values</td>
<td>Protection against unnotised changes,</td>
<td>passwords, checksums on files</td>
</tr>
<tr>
<td>Logs</td>
<td><em>Traces, error messages and dumps from systems and applications</em></td>
<td>Syslog, eventlog, audit, BSM</td>
</tr>
</tbody>
</table>
# General example of control principles

<table>
<thead>
<tr>
<th>Security controls</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler generated airbag - canary</td>
<td><em>Make sure buffer overflows don't get undetected</em></td>
<td>ProPolice, VisualStudio /GS</td>
</tr>
<tr>
<td>ASLR</td>
<td><em>Make sure it's hard to write code that knows of addresses to misuse. Where did that lib go?</em></td>
<td>Windows Vista/7/2008, OpenBSD, Linux, MacOSX, etc</td>
</tr>
<tr>
<td>DEP, NX, W^X</td>
<td><em>Make sure memory is not executable</em></td>
<td>Windows Vista/7/2008, OpenBSD, Linux, MacOSX, etc</td>
</tr>
</tbody>
</table>
### General example of control principles

<table>
<thead>
<tr>
<th>Security controls</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrubing, zeroing</td>
<td>Make sure that old data areas are cleaned before usage or returned to system</td>
<td>file systems, VM system</td>
</tr>
</tbody>
</table>
Examples of vulnerabilities and attacks
Where do attacks occur?

- Social engineering attacks
- Remote exploits
- User/admin errors
- Local exploits

Security layers:
- Host security
- Network security
- Human security
Examples of threats and attacks

**Confidentiality**
- Wrong file permissions
- Bypasswd security checks
- plain text in RAM

**System integrity**
- Manipulated system configuration
- Manipulated program binaries

**Data integrity**
- Manipulated user files
- Zapped system logs

**Availability**
- fork bombs
- SYN flood
- manipulated network packets
- intentional filling of disk partition
- unintentional filling of disk partition

**Other threats**
- fork bombs
- SYN flood
- manipulated network packets
- intentional filling of disk partition
- unintentional filling of disk partition
- manipulated user files
- Zapped system logs
Some examples of classic attacks (1/2)

- Ken Thompson’s trojanized C compiler
  - Modify the source code to the compiler to recognize if it recompile itself or the login program - insert backdoor in login
  - recompile compiler
  - remove source code changes and recompile the compiler
  - recompile the login program with the modified compiler
- No visible signs for humans or tools to see the backdoor in source code. Calls for binary inspection or decompilation.

Ken Thompson - TURING AWARD LECTURE: Reflections on Trusting Trust.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.5728&rep=rep1&type=pdf
Some examples of classic attacks (2/2)

• Create a symbolic link that is used to trick the system to overwrite an important file at a controlled point in time

```
ln -s /tmp/core /etc/passwd
```
# Example of attacks

<table>
<thead>
<tr>
<th>Attack method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rootkit</td>
<td>Replace parts of applications or kernel with attackers code.</td>
</tr>
<tr>
<td></td>
<td>Often contain built-in protection and deception parts to hide rootkit itself, as well as malicious code.</td>
</tr>
<tr>
<td></td>
<td>Often created / built upon modified original source code.</td>
</tr>
<tr>
<td></td>
<td>Name derives from earliest versions of threat that was created on UNIX systems</td>
</tr>
<tr>
<td>time-of-check-to-time-of-use (TOCTTOU)</td>
<td>Type of race-condition bug caused by (maliciously controlled) changes in a system between the checking of a condition (such as a security credential) and the use of the results of that check</td>
</tr>
</tbody>
</table>
# Example of attacks

<table>
<thead>
<tr>
<th>Attack method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer overflow</td>
<td>Attacks that allow an attacker to <em>deterministically alter the execution flow of a program by submitting crafted input to an application</em>. Executable code is written outside the boundaries of a memory buffer originally used for storing data. The executable parts is somehow made to execute, eg by manipulate return adress to be used when a function call is finished. Real world examples: OpenBSD IPv6 mbuf’s* remote kernel buffer overflow[1], windows kernel pool</td>
</tr>
<tr>
<td></td>
<td>Synonyms and variants: Buffer overrun, Stack smashing, Heap smashing, format string bugs, memory corruption attack</td>
</tr>
</tbody>
</table>

[1] [http://www.coresecurity.com/content/open-bsd-advisories](http://www.coresecurity.com/content/open-bsd-advisories)

* An mbuf is a basic unit of memory management in the kernel IPC subsystem
Example of attacks

• Attacks by attaching malicious hardware to buses and ports
  • Firewire and other DMA based methods to access memory of a computer (evil maid attacks)
  • UEFI attacks via Thunderbolt (thunderstruck attack)
  • Using JTAG interfaces to snoop & manipulate bus
• Removal of physical memory chips, (cold boot attacks)

Remember that there is a number of ways that all OS security controls can be bypassed, especially if the operating system is not running - a very good side-channel attack ;-)}
Attacks and counter measures

- Buffer overflow/memory corruption attacks
- Stack canaries
- More advanced buffer overflows, defeating canary
- Address Space Layout Randomization (ASLR)
- No-executable (NX, W^X) stacks
- Data Execution Prevention (DEP)
- Hijacking JIT compilers
- ROP attacks

Note - several of these counter measures does not work for protection within the kernel
Attacks and counter measures

- **Chaining of attacks** - combining a number of exploits to achieve goal
  - finding and abusing a number of different vulnerabilities might allow an attacker to achieve goals not possible with just one potent exploit
  - **Code execution in gadgets (ROP) + sandbox escape + elevation of privileges + execution of privileged code**
Security models
Security controls in operating systems

Discretionary access controls

Subject A can decide how an object created by subject A can be interacted with by subject B

Mandatory access controls

The System policy decide how object from subject A can be interacted with by subject B
Discretionary access control

**DAC**

- Exists in all COTS systems
- Conceptually work by having a subject (= user) have ownership of an object (e.g. file, process) and by being owner have right to control access rights to the object
- Used in UNIX, Windows NT derived OS’s, etc
Discretionary access control  

DAC

• Became a mandatory requirement to sell computers to DoD 1985

• First implemented in Windows NT 3.5 and Apple Mac in Mac OS X

• Simple concept to understand and administrate. Especially in (classic) UNIX where conceptually everything is a file in the filesystem
Role Based Access Control

RBAC

- Created to implement the least-privilege principle
- No users should have high privileges, all those should get these privileges from roles with clearly defined rights
Mandatory Access Controls

MAC

- Mandatory access controls were created to naturally implement the security policies of military organizations.
- Lots of resources and research have been spent in this area.
- But it has been hard to develop models and implementations that work well in real-life situations, especially in ordinary organizations.
The **BIBA Integrity model**

- Created by Kenneth Biba in 1977.
- Primarily goal is to maintain *system/data integrity*.
- Is built on a system policy where the following is legio:
  
  "no write up, no read down"

- Can be seen as *BLP upside-down*
BIBA Integrity model

- A subject can only write to its own level and to lower levels
- cannot write to higher labeled objects
- A subject can only read its own level and higher

The point is that objects on one level cannot be affected by corrupted data on a lower level
BIBA Integrity model

- Relatively simple policy, which simplifies implementation and administration
- Still hard to get a full working BIBA solution to gain acceptance in the real world
- Two example of where versions of BIBA is implemented in COTS is in Windows Vista and FreeBSD (TrustedBSD)

http://www.freebsd.org/cgi/man.cgi?mac_biba
BIBA Integrity model - in practise

- A dedicated system that runs a nameserver will be configured to use a biba policy

- System binaries and system directories, name server binaries, DNS zone information must be classified to belong to the high integrity label

- The subject, (in this case a users that runs the DNS server program, bind) is set to have a low integrity label
BIBA Integrity model
- in practise

• The result is that the running name server process cannot alter any data at all (config files, zone files) in case it get hacked, but is allowed to read all necessary information.

• To administer the name server, a separate user with an integrity label of equal level or higher (but that has some read issues), than the zone file must log in to edit it.
BIBA integrity model

root@freebsd# setfmac -R biba/high /var/named
root@freebsd# setpmac biba/low /etc/rc.d/named start

Note: temporary filer, pid-filer etc must also be given biba/low labels
BIBA in *Windows Vista*

- Used in few places, most notably in IE that is runned in low-privilege mode.
- Few tools to administer it
- Will stop working if UAC is disabled - and UAC is often disabled
- Have 4 integrity levels: low, medium, high and system
- Microsoft dumped “no read down” - which sort of destroys the model....

UAC = User Account Control
Drawbacks with BIBA

• Hard to get a policy that really works in real-life situations

• Hard to strictly follow the model. Almost all implementations of BIBA have extra verbs that isn’t part of original BIBA model, eg Freebsd’s BIBA/equal, which is a way to make an object or subject that is excluded from the policy

http://fuse4bsd.creo.hu/localcgi/man-cgi.cgi/mac_biba+4
Multilevel security - MLS

- Focus is on confidentiality of information and information flow, not system integrity
- Also known as the Bell-La Padula, BLP, model
- Simple general rule:
  "no write down, no read up"
- No write down is also known as the confinement property or *-property (star)
- Normally used in military style information management situations

http://www.albany.edu/acc/courses/ia/classics/belllapadula1.pdf
Multilevel security - MLS

- MLS is built upon that all subjects get a clearance level, which is then used to check the classification of an object.

<table>
<thead>
<tr>
<th>Level</th>
<th>user</th>
</tr>
</thead>
<tbody>
<tr>
<td>TopSecret</td>
<td>Andreas</td>
</tr>
<tr>
<td>Secret</td>
<td>Robert</td>
</tr>
<tr>
<td>classified</td>
<td></td>
</tr>
<tr>
<td>unclassified</td>
<td></td>
</tr>
</tbody>
</table>

Note: The security labels used in the example is used from the classical military style example. They can be arbitrarily things, like ”outside”, ”DMZ”, ”inside” etc.
Multilevel security - MLS

- A simple example of usage of MLS

- During some field work Robert writes an report rapport, classified as “secret”.

- Andreas and all users with clearance secret or higher can read, but not modify the report.
• If Andreas edit the report, his clearance is tainting the report, and the new classification is now "top secret".

• Information have a tendency to raise up in MLS system, since there are nothing that can happen that make is go downward.

• In the end, someone need to perform a manual reclassification work
Multilevel security - MLS

• Since DoD require MLS, a lot of COTS system have gotten MLS support

• TrustedAIX, TrustedSolaris, TrustedHP-UX

• A standard problem is that they are several releases behind the stock version of the operating system, since they need to be re-evaluated all the time
Multilevel security - MLS

- MLS is problematic, but does solve some hard-to-solve standard problems
- In a MLS system, a compromised web reader does not automatically get access to all files that a subject owns, just because it is run by the subject (user) that created/owns the files
- Encryption key files, secret reports, sensitive files
MLS in “modern” systems

• MLS exists in a number of different modern OS’s OS, e.g. FreeBSD, Solaris (trusted extensions), AIX 6.1 and Linux (SELinux enabled)

• Differences to the theoretical model is small, but important to make a practically useful system

MLS in “modern” systems

- MLS in Solaris 10 with *trusted extensions* is built on setting labels on container level, rather than subject and object level.

- In SELinux it is often not used since TE, *Type Enforcement* both is more flexible and simpler to use to protect the information.
POSIX 1.e

- IEEE standard defines *capabilities, acl, mac och labels*
- The IEEE work was ended 1999 after 13 years and was never completely finished
- Lots of implementations of security models in UNIX have its root in POSIX 1.e
LSM - Linux Security Module

- Was created by Crispin Cowan/imunix 2001
  - To avoid locking certain security models into the Linux Kernel

- Framework to implement security models in Linux with as few kernel changes as possible

- Also used to implement other security features, such as intrusion detection, etc

- Standard since 2.6 kernel

- Not completely different the MAC-modules in fbsd (trustedBSD) and kauth in netbsd

http://netbsd.gw.com/cgi-bin/man-cgi?kauth+9+NetBSD-current
http://en.wikipedia.org/wiki/Linux_Security_Modules
Apparmor

- Implemented using LSM for the Linux kernel
- Is built to create a white list for what application is allowed to do
- Implements part of posix i.e (capabilities)
- Mandatory

http://en.wikipedia.org/wiki/AppArmor
Apparmor

- Poison of choice in Ubuntu och SLES, instead of SELinux that competitors have chosen
- Much simpler policy language / configuration than other mandatory access controls
- Have a wizard functionality to create policies
# Apparmor - rules

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Any symbol besides /</td>
</tr>
<tr>
<td>*</td>
<td>any number of symbols besides /</td>
</tr>
<tr>
<td>**</td>
<td>* + /</td>
</tr>
<tr>
<td>[abc]</td>
<td>a, b, or c</td>
</tr>
<tr>
<td>[a-c]</td>
<td>a, b, or c</td>
</tr>
<tr>
<td>{ab,cd}</td>
<td>ab or cd</td>
</tr>
</tbody>
</table>
# Apparmor - rules

<table>
<thead>
<tr>
<th>Abbrev</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>read</td>
</tr>
<tr>
<td>w</td>
<td>write</td>
</tr>
<tr>
<td>ux</td>
<td>unconstrained execute</td>
</tr>
<tr>
<td>Ux</td>
<td>ux + scrubed env</td>
</tr>
<tr>
<td>px</td>
<td>disc profile execute, <code>change profil</code></td>
</tr>
<tr>
<td>Px</td>
<td>px + scrubed env</td>
</tr>
<tr>
<td>ix</td>
<td>inherit exec, <code>keep same profil</code></td>
</tr>
<tr>
<td>m</td>
<td>Allow PROT_EXEC with mmap(2)</td>
</tr>
<tr>
<td>l</td>
<td>link</td>
</tr>
</tbody>
</table>
Apparmor - example for firefox

```
/usr/lib/firefox/firefox.sh flags=(complain) {
    /bin/basename rmx,
    /bin/bash rmx,
    /bin/gawk rmx,
    /bin/netstat rmx,
    /dev/log w,
    /dev/null rw,
    /dev/tty rw,
    /dev/urandom r,
    /etc/fonts/** r,
    /etc/ld.so.cache rm
    /etc/localtime r,
    /etc/magic r,
    /etc/opt/gnome/** r,
    /etc/passwd r,
    /etc/resolv.conf r,
    /home/*/fontconfig/** r,
    /home/*/gconfd/* rw,
    /home/*/gconf/ r,
    /home/*/gconf/* rw,
    /home/*/gnome2_private/ w,
    /home/*/mozilla/** rw,
    /home/*.Xauthority r,
    /lib/id-2.5.so rmx,
    /lib/lib*.so* rm,
    /opt/gnome/lib/GConf/2/gconfd-2 rmx,
    /opt/gnome/lib/**.so* rm,
    /proc/meminfo r,
    /proc/net/ r,
    /proc/net/*/ r,
    /tmp/gconfd-*/ r,
    /tmp/gconfd-/** rw,
    /tmp/orbit-*/ w,
    /tmp/orbit-/** w,
    /tmp/ r,
    /usr/bin/file rmx,
    /usr/lib/browser-plugins/ r,
    /usr/lib/browser-plugins/** rm,
    /usr/lib/firefox/firefox-bin rmx,
    /usr/lib/firefox/firefox.sh r,
    /usr/lib/firefox/** r,
    /usr/lib/firefox/**.so* rm,
    /usr/lib/gconv/** r,
    /usr/lib/gconv/**so m,
    /usr/lib/lib*.so* rm,
    /usr/lib/locale/** r,
    /usr/lib/locale/** r,
    /usr/lib/locale/*so m,
    /usr/lib/share/** r,
    /var/cache/fontconfig/** r,
    /var/cache/libx11/compose/** r,
    /var/run/dbus/system_bus_socket w,
    /var/run/nscd/passwd r,
    /var/run/nscd/socket w,
    /var/tmp/ r,
} 
```

Note that this configure is very firefox and linux version specific
Apparmor - critics

- path-based instead of inod baserad
- The simplification wrt the wizarden, makes the simplification too much
- Only includes defined program, not the systemet as such or other programs
- Often is markedet to be more than it really is, e.g. RBAC
SElinux / Type Enforcement (te)

- **Type enforcement** is built on the concept that a subject is attached to a *domain* and that object is attached to *types*

- In a matrix one define how domain-to-domain and domain-to-type interaction is allowed.
SELinux

- In the SELinux there is a security matrix called policy which can be targeted, strict, permissive or enforcing.

- targeted - what is allowed besides that which is explicit prohibited

- strict - nothing is allowed beside that is explicitly allowed
SELinux

- SELinux is used to lock things down - primarily services, but can in theory lock down anything

- The focus on locking down services (e.g. network services) will result in that authorized users will not be locked down and gain advantages of any security controls from SELinux
SELinux

• Reference policy is maintained by tresys*
  - earlier by NSA

• Contain a few “trusted programs”,
  • e.g. su, sshd, login.

• These trusted programs must be able to perform so called *domain transitions*.

* https://github.com/TresysTechnology/refpolicy/wiki
TITLE: DEBIAN OPENSSH SELINUX PRIVILEGE ESCALATION VULNERABILITY

Severity: CRITICAL

Description:

Debian Linux can be configured to use SELinux extensions. OpenSSH may also be configured to use SELinux and to interface with the role-based privilege system.

Debian Linux is prone to an SELinux privilege-escalation vulnerability due to a flaw in its OpenSSH package.

Specifically, when remote users authenticate against a vulnerable OpenSSH server, their username can contain extra information, including the SELinux role they wish to use upon a successful login. Usernames containing a trailing `"&lt;role&gt;` will be parsed as the user requesting the `&lt;role&gt;` SELinux role; the system will improperly grant the role privileges to the user. This reportedly occurs without proper validation or privilege checking.

Successfully exploiting this issue allows attackers who can successfully authenticate against affected OpenSSH servers to gain access to any configured SELinux role. This may allow them elevated privileges, facilitating the complete compromise of affected computers.

Note that OpenSSH must be configured with `--with-selinux` for this vulnerability to be exposed.

Information regarding specific affected packages of OpenSSH running on Debian Linux is not available. Other derivative versions and operating systems may also be affected.

Affected Products:

- Debian Linux 4.0
- Debian Linux 4.0 alpha
- Debian Linux 4.0 amd64
- Debian Linux 4.0 arm
- Debian Linux 4.0 hppa
- Debian Linux 4.0 ia-32
- Debian Linux 4.0 ia-64

Important note to remember is that security code can add new security bugs
SELinux

- Is distributed in COTS Linux distributions such as RedHat and Fedora
- Is actively maintained by RedHat, Tresys, NSA and others
- The company Tresys is the maintainer of the reference policy and several selinux userland program
  - also sell separate policies for more program, tex razor
SELinux

- The model used to grant rights is extremely granular and powerful
- `exec_heap`, `exec_mem` are permissions in SELinux

- The SELinux advocate Russel Cooker have test boxes for anyone to use where root-login is allowed for anonymous users

SELinux

● Drawbacks with SELinux

● To create a flawless SELinux policy from scratch is very hard - often it is a copy-and-paste work from some existing policy, and thus might not really implement your intended design

● To maintain a SELinux policy is non-trivial, compare for example with apparmor

● Dependencies on trusted programs as well as classic data validation errors can result in security errors, as usual
GRsecurity

- Brainchild of Brad Spengler
- NOT based on the LSM concept
- Brad is a vocal critic of the LSM concept and have developed PoC attacks against LSM based security solutions
- It is released as a separate, non official, patch cluster to the Linux Kernel
- Some see the non-official status and ”hack” type of solution as unacceptable, e.g. Xorg
GRsecurity

- Badly supported by Linux distributions
- Almost always require that one compile a custom kernel, which can have problems on its own
- Have support for RBAC through automatic rule generation
Virtualization and isolation
Isolation, separation and virtualization

- chroot (no virtualization, just isolation)
- jails
- user mode linux, uml
- Virtual machines: Vmware, MS Virtual Server, Containers
- Hardware partitioning: Sun LDOMs, IBM LPAR
Overview of virtualization

<table>
<thead>
<tr>
<th>Application</th>
<th>Application</th>
<th>Application</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call to the operating system kernel</td>
<td>System call to the operating system kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating system kernels with basic functions</td>
<td>Operating system kernels with basic functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware interface</td>
<td>Hardware interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtualization functionality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating system kernels with basic functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware interface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>Memory management</td>
<td>Data storage</td>
<td>I/O</td>
</tr>
</tbody>
</table>

Applikationer

Gränssnitt mot hårdvara

Virtualiseringsfunktionalitet

Gränssnitt mot hårdvara
Pro’s and con’s with virtualization

- Isolation, and to have hardened and dedicated servers running specific services, are standard ways to minimize attack surface. Virtualization tools can help this.

- Various types of OS supported or application supported sandboxing is good as a way to get defense-in-depth.

- It's easy to believe that virtualization will automatically make things secure, and that there is no way to jump between guest os', but exploits have shown this not hold true, e.g. cloudburst.

http://www.immunityinc.com/documentation/cloudburst-vista.html
Some trends of interest
”real” OSs in new places

- **Linux**: phones & pads, embedded systems
- **Windows**: phones & pads, embedded systems
- **MacOSX** (iOS): phones, pads, embedded systems

...and embedded systems are used in really critical places, like industry or utility companies.
”real” OSs in new places

- Gaming consoles: WII, Xbox, PS3 use ”real OSs”
- A lot focus on security - they know they will be attacked
- Built in low-level hardware security
- Continuous study of #fail
- Hacked, hacked again, completely broken
”real” OSs in new places

2006: Hacked “LeCroy” oscilloscope at CERN (running Win XP SP2)
“real” OSs in new places

• “Same” attacks works in lots of places
• Not full OS’s on the new hosts, e.g. less protection makes it easier
• Always on, always reachable
• Device moves around, connects to unknown/rogue places, and easy to get physical access to them
On the high-level side

• When operating systems vendors and FOSS projects built-in better security and have more quality on their code, other low-hanging fruits are picked

• Attacks are moving toward applications, e.g. Media players, flash framework, rich-text-document-format-viewers

• ....and subsystems, e.g. database or integration software
On the low-level side

- More security critical functions are moved to hardware components
- But the cat-and-mouse game continues, e.g. successful hacking of TPM chips
- Bus snooping with JTAGs, goodfet, facedance, salea logic, etc

http://www.theregister.co.uk/2010/02/17/infineon_tpm_crack/
Summary
Important experience (1/3)

- OS-security is composed by several distinct parts. Failure in any of these parts with result in a security compromise.
- There are many ways to circumvent security controls.
- “Right tools for the right job”
- E.g. to use MLS on a packet filtering firewall can be something that one should consider if that is the right thing to do...
Important experience (2/3)

- Old and new security functions often co-exists in a system
- Can be complexed to understand the full consequence of a setup
- Often hard to use
- Is the counterpart to KISS

- RBAC is easy to understand, but hard to implement
  - 10 or 250 roles - wide or narrow privileges?
  - Often templates and profiler that is copied or referenced - changes to original definitions are not properly propagated, or in some cases erroneously propagated when they should not
Important experience (3/3)

- New trends challenge existing security concept or controls
- E.g. attacks goes from server -> client
- Unfortunately, many good models are not properly used in practise since they are very had to understand, work with and to administer
Questions?
• Heap spraying as an attack method to do buffer overruns. Common attacks include Javascript based implementations to be used in web browsers, or to attack Adobe flash

Additional information (2/3)

- Security in Cloud Computing
- US NIST government agency on cloud security
- Enisa report on Cloud security with recommendations
Additional information (3/3)

- Contact me if you’re interested in any of the work we have done with
  - Asterisk running contained with SELinux
  - FreeBSD w BIBA for different network services, e.g. bind
  - Free sandboxing, e.g. ironfox - our sandboxed firefox for MacOSX, ironadium - sandbox Adium, etc. See https://www.romab.com/ironsuite
Concepts mentioned during the class

- Kerkhoff’s principle - a ”rule of thumb” in crypto design - security must not rely on keeping the design/machine/source code secret

- SDL - software development life cycle, MS model for developing ”more secure code”

- PRNG - Pseudo Random Number Generator. The rand() function is not entirely ”random”, and an external observer can recreate the series of number created by rand if the ”seed value” used by rand is somehow extracted/observed/etc
Concepts mentioned during the class

- Vulnerability - some property of a piece of software that can be manipulated or used in unintended ways by an attacker, a.k.a. security bugs.

- Exploits - usage of a vulnerability. Exploit code is software snippets to use the vulnerability.

- Zero day exploits (0day). A previously ”unknown” (or at least publicly not published and wellknown) exploit for a security vulnerability.
Concepts mentioned during the class

• **Attack vector** - Different paths to reach an vulnerability. One path might be closed by a vendor patch, but another might still be there, if the root cause is not identified and fixed.

• **Reverse engineering.** To re-create the original design by observing the final result, in computer science - to re-create some source code by examining a binary.
Tools mentioned during the class

- IDA pro - Disassembler
- Hexray - Decompiler
- Ollydbg, windbg - Other disassemblers
- Bindiff - Advanced tool from zynamics to compare binaries, with call graphs etc. Not same as built-in windows tool with same name.