CYENG 312/GECE 594 :
Trusted Operating System (OS)

Lecture 03: SELinux and AppArmor

Instructor: Shayan (Sean) Taheri, Ph.D.

Assistant Professor

The Department of Electrical and Cyber Engineering (ECE)

The Institute for Health and Cyber Knowledge (I-HACK)

The Gannon University (GU)

Personal Information

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Past Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

SELinux and AppArmor

- Two major Open Source Secure Oses: More Security Vs. More Usability
- SELinux
  - *Security-Enhanced Linux (SELinux) is a Linux feature that provides a variety of security policies for Linux kernel.*
  - *It is included with CentOS / RHEL / Fedora Linux, Debian / Ubuntu, Suse, Slackware and many other distributions.*
  - *For Strict security, but hard to use.*
  - *Developed by NSA.*
- AppArmor
  - *AppArmor (Application Armor) is another security software for Linux which maintained and released by Novell under GPL.*
  - *AppArmor was created as an alternative to SELinux.*
  - *AppArmor works with file paths.*
  - *AppArmor is default in OpenSUSE and Suse Enterprise Linux. It was first successfully packaged for Ubuntu Linux.*
  - *Easy to use, but to not strict in security.*
  - *Was called Subdomain, developed by Immunix.*
  - *Now maintained by Novell.*

SELinux and AppArmor (Contd.)

- SELinux Features
  - *Clean separation of policy from enforcement*
  - *Well-defined policy interfaces*
  - *Support for applications querying the policy and enforcing access control*
  - *Independent of specific policies and policy languages*
  - *Independent of specific security label formats and contents*
  - *Individual labels and controls for kernel objects and services*
  - *Caching of access decisions for efficiency*
  - *Support for policy changes*
  - *Separate measures for protecting system integrity (domain-type) and data confidentiality (multilevel security)*
  - *Very flexible policy*
  - *Controls over process initialization and inheritance and program execution*
  - *Controls over file systems, directories, files, and open file descriptors*
  - *Controls over sockets, messages, and network interfaces*
  - *Controls over use of "capabilities"*

SELinux and AppArmor (Contd.)

- AppArmor Features
  - *Full integration.*
  - *Easy deployment.*
  - *AppArmor includes a full suite of console and YaST-based tools to help you develop, deploy and maintain application security policies.*
  - *Protects the operating system, custom and third-party applications from both external and internal threats by enforcing appropriate application behavior.*
  - *Reporting and alerting. Built-in features allow you to schedule detailed event reports and configure alerts based on user-defined events.*
  - *Sub-process confinement. AppArmor allows you to define security policies for individual Perl and PHP scripts for tighter Web-server security.*

SELinux and AppArmor (Contd.)

- SELinux Pros and Cons
  - *Admin skill set (learning curve) – High*
  - *Complex and powerful access control mechanism – Yes*
  - *Detailed configuration required – Yes*
  - *GUI tools to write / modify rules set – Yes*
  - *CLI tools to write / modify rules set – Yes*
  - *Ease of use – No (often described as horrible to use)*
  - *Binary package – Available for most Linux distributions*
  - *System performance impact: None*
  - *Security Framework: Mandatory access controls using Flask*
  - *Auditing and logging supported – Yes*
  - *Typical user base – Enterprise users*
  - *Documentation – Well documented*

SELinux and AppArmor (Contd.)

- AppArmor Pros and Cons
  - Admin skill set (learning curve) – Medium
  - Complex and powerful access control mechanism – Yes.
  - Detailed configuration required – Yes.
  - GUI tools to write / modify rules set – Yes (yast2 and wizards).
  - CLI tools to write / modify rules set – Yes.
  - Ease of use – Yes (often described as less complex and easier for the average user to learn than SELinux).
  - Binary package – Available for Ubuntu / Suse / Opensuse and distros.
  - System performance impact – None.
  - Security Framework – Mandatory access controls.
  - Auditing and logging supported – Yes.
  - Typical user base – Enterprise users.
  - Documentation – Documented (mostly available from Opensuse and Suse enterprise Linux).

Access Control in SELinux: Type Enforcement

- Label based access control
  - **Domain** Identifier for process.
  - **Type** Identifier (label) for resources.
  - Controls permission between domain and type.
- Fine-grained access control
  - A method of controlling who can access certain data.
  - Compared to generalized data access control, also known as coarse-grained access control, fine-grained access control uses more nuanced and variable methods for allowing access.

SELinux: Configuration of policy

- The most important feature
  - What domain can access what access to what types?

Ex. Web server: domain httpd_t: Allowing access to homepage

- **allow specify domain, type, permission**
  - **allow** httpd_t web_contents_t file:{ read };

Domain **Type** **Permission**

- Assign label (= type) to resource

  /var/www(|/.*) **system_u**: object_r:web_contents_t

- Many lines of allows (10k-100k) are required
- Macro is Used: Bunch of allows is summarized by macro

Reminder - Macro

- It is a rule or pattern that specifies how a certain input should be mapped to a replacement output.
- Applying a macro to an input is known as macro expansion.
- The input and output may be a sequence of lexical tokens or characters, or a syntax tree.
- Macros are used to make a sequence of computing instructions available to the programmer as a single program statement, making the programming task less tedious and less error-prone.

Example of policy

- bind.te: allowing acces

```
type named_t;
type named_exec_t;
init_daemon_domain(named_t,named_exec_t)
...
...
kernel_read_kernel_sysctls(named_t)
kernel_read_system_state(named_t)
kernel_read_network_state(named_t)
kernel_tcp_recvfrom(named_t)
....
corenet_tcp_sendrecv_all_if(named_t)
corenet_raw_sendrecv_all_if(named_t)
corenet_udp_sendrecv_all_if(named_t)
corenet_tcp_sendrecv_all_nodes(named_t)
corenet_udp_sendrecv_all_nodes(named_t)
corenet_raw_sendrecv_all_nodes(named_t)
corenet_tcp_sendrecv_all_ports(named_t)
corenet_udp_sendrecv_all_ports(named_t)
corenet_non_ipsec_sendrecv(named_t)
corenet_tcp_bind_all_nodes(named_t)
corenet_udp_bind_all_nodes(named_t)
```

…293 lines
…100 kinds of macros

- bind.fc:assigning label

```
/etc/rndc.*             --      gen_context(system_u:object_r:named_conf_t,s0)
/etc/rndc\.key          --      gen_context(system_u:object_r:dnssec_t,s0)

/usr/sbin/lwresd                --      gen_context(system_u:object_r:named_exec_t,s0)
/usr/sbin/named         --      gen_context(system_u:object_r:named_exec_t,s0)
/usr/sbin/named-checkconf --    gen_context(system_u:object_r:named_checkconf_exec_t,s0)
/usr/sbin/r?ndc         --      gen_context(system_u:object_r:ndc_exec_t,s0)

/var/log/named.*                --      gen_context(system_u:object_r:named_log_t,s0)

/var/run/ndc            -s      gen_context(system_u:object_r:named_var_run_t,s0)
/var/run/bind(/.*)?             gen_context(system_u:object_r:named_var_run_t,s0)
/var/run/named(/.*)?            gen_context(system_u:object_r:named_var_run_t,s0)

ifdef(`distro_debian',`
/etc/bind(/.*)?                 gen_context(system_u:object_r:named_zone_t,s0)
/etc/bind/named\.conf   --      gen_context(system_u:object_r:named_conf_t,s0)
```

...45 labels

Difficult to understand

Access Control in AppArmor

- Easier than SELinux Implemented as Loadable Kernel Module (LKM).
- Recently, often compared with SELinux.
- Features:
  - Access control
    - ✓ Controls file and POSIX capability
    - ✓ Path name-based
      - Label is not used
    - ✓ Profile → “policy”
  - GUI Tools
    - ✓ Integrated in YaST
      - Generating profile
      - Log report
      - Not so important for embedded computing

AppArmor Path Name-Based Access Control

- Path name based:
  - Identify file with “path name”
  - Easy to understand
- Example:

/usr/sbin/httpd{

/var/www/** r,
}

→ /usr/sbin/httpd can read under /var/www

Permission to File

- Basic permission: r,w,x,l
  - r read
  - w : write
  - ix : execute
  - l : link (remove file)

POSIC Capability

- Controls capability
  - Capability
    - ✓ Important operation other than file access
    - ✓ Example:
      - – net_bind_service: bind well-known port
      - – net_raw: use raw socket
  - For detail: see $man capabilities
- The **Portable Operating System Interface (POSIX)** is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.
- POSIX defines both the system- and user-level application programming interfaces (API), along with command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems.
- POSIX is intended to be used by both application and system developers.

Configuration for Profile

- Simple, easy to understand

```
/usr/sbin/named {          -> path to exectable
 #include <abstractions/base>
 #include<abstractions/nameservice>
 capability net_bind_service,
 capability setgid,
 capability setuid,
<snip>
  /var/lib/named/** rwl,
  /var/run/named.pid wl,
}
```

Common

Capability

Access to file

Linux Security Module (LSM)

- Both use LSM for implementation
- LSM: Linux Security Module
  - Set of hooks in kernel to check security
  - It is included in mainline from LIDS 2.6 version
- Using LSM:
  - SELinux, AppArmor, LIDS (for 2.6 version)
- Not using
  - TOMOYO Linux, LIDS (for 2.4 version)

Difference Between SELinux and AppArmor

- Granularity of permission
  - SELinux:
    - ✓ File, network, IPC, POSIX, capability, etc.
  - AppArmor
    - ✓ File + POSIX capability
  - AppArmor can reach SELinux in theory, because both use LSM.

How to identify resource

- Fundamental difference
  - Affects security and usability
- Label based Vs. Path name based
  - Label: lower usability, higher security
    - Assign label to file
    - SELinux
  - Path name: higher usability, lower security
    - Identify file with path name
    - AppArmor, TOMOYO Linux
- Compare them by showing benefit and loss of pathname

Benefit of Path Name-Based

- High usability, easy to understand → No
- need to extend file system
  - Label base: File system have to be extended to store label.

- Implementing policy generation tool is easier.

- Nothing happens when index node (inode) number is changed.

Benefit of Path Name-Based in Policy Generation

- Example case:
  - PHP tried to write /var/www/html/write/test.txt → But, access denied by Secure OS
  - Have to generate policy from log
- SELinux

1) label under /var/www/html -> httpd_sys_content_t
2) Log says:
   - *httpd_t was denied to write to httpd_sys_content_t*
3) Generate policy from log:
   - ***allow httpd_t httpd_sys_content_t: file write;***
     - > ***allowing write access for whole "/var/www" resource!***

4) Unnecessary access is granted.

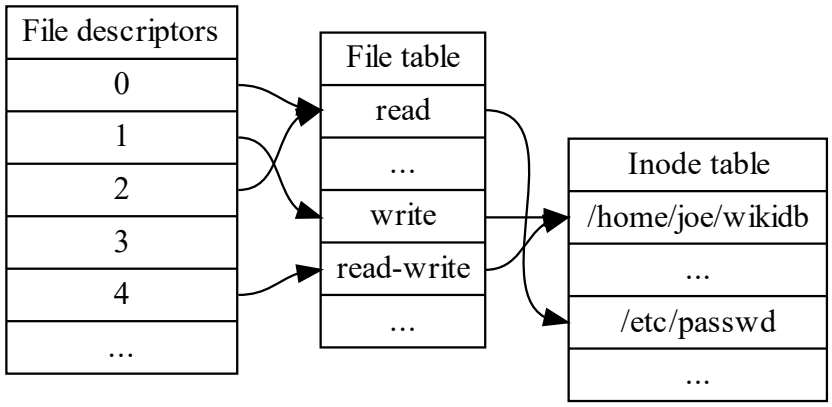
- AppArmor

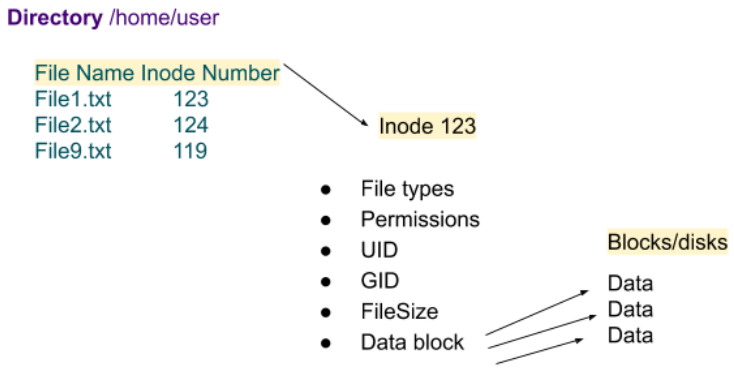
1) log says:
   - /usr/sbin/httpd is denied to write /var/www/html/write/test.txt
2) Generate policy (= profile) from log:
   - **/usr/sbin/httpd{/var/www/html/write/test.txt w}**

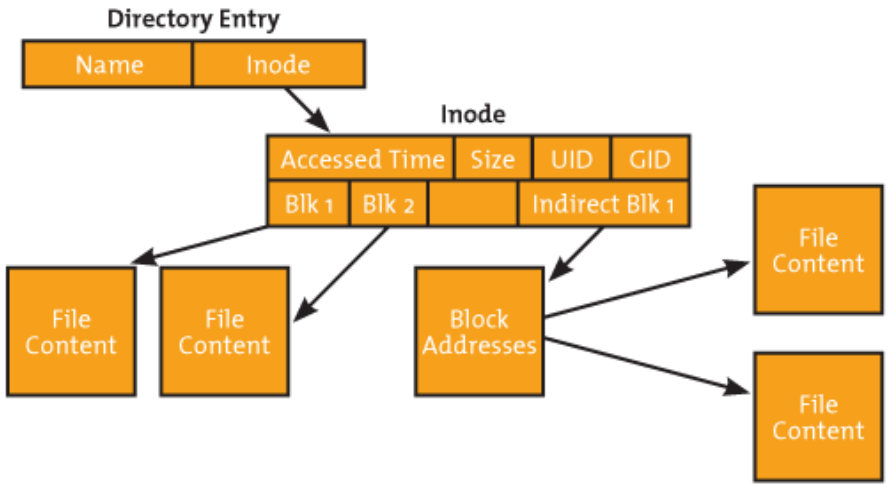
3) Unnecessary access is NOT granted → Benefit of Path Name-Based in AppArmor.

Reminder – "index node (inode)"

- The inode (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory.
- Each inode stores the attributes and disk block locations of the object's data.
- File-system object attributes may include metadata (times of last change, access, modification), as well as owner and permission data.
- A directory is a list of inodes with their assigned names. The list includes an entry for itself, its parent, and each of its children.







Benefit of Path Name Based in Change of inode Number

- Example: “/etc/mtab“.
- SELinux: Label is lost when inode number is changed
  - Label is associated with inode
    - /etc/mtab
    - **vi** (Visual Instrument, a text editor), **rpm** (Red Hat Package Manager, a command-line utility for managing packages) changes inode
  - Solution
    - “file type transition” configuration
      - Not easy for beginner
    - Some userland have to be extended, Example: rpm ,vi.
- AppArmor:
  - No problem!

Loss by Path Name-Based

- **Loss By**:
  - Information Flow Analysis
  - Temporary (tmp) Files

    → Who can access the information?
- Some people say path name-based security is broken because of:
- **Ex**: Information flow analysis to password information.
  - Initial State: Stored in → /etc/shadow
  - If hardlink is created to /etc/shadow, password information can be accessed via hardlink.
    - In computing, a hard link is a directory entry (in a directory-based file system) that associates a name with a file.
    - Thus, each file must have at least one hard link.
  - What happens in information flow analysis?
    - Have to traverse whole file tree to find hardlink.
    - What if more hardlink is created during traversal?
  - SELinux:
    - All you have to do is to check what kind of domain can access label for /etc/shadow.
    - Label is the same for hardlink.

Loss by Path Name-Based in Temporary Files

- When creating randomly named file under /tmp
- SELinux
  - Can identify such file by naming label such as httpd_tmp_t.
- AppArmor
  - How to identify randomly named files?
    - ✓ Result in allowing whole /tmp.

SELinux Policy Editor (SEEDIT)

- Tool that makes SELinux easy
- Open Source: http://seedit.sourceforge.net/
  - Originally developed by Hitachi Software
  - Included in Fedora repository

  Main Feature: Simplified Policy Description Language (SPDL)
- AppArmor-like syntax to write policy
  - Example:
    - domain httpd_t
    - program /usr/sbin/httpd;
    - allow /var/www/** r; → path name configuration
  - This is converted to SELinux policy syntax
    - type var_www_t; → label is generated
    - allow httpd_t var_www_t { file dir }: read;

SELinux Policy Editor (Contd.)

- Still different from AppArmor
- Inherit drawback from label-based access control
  - Change of inode
    - Generated policy is label-based
- Inherit good points from SELinux
  - Fine-grained permission (interprocess communication - IPC, network)
    - IPC refers specifically to the mechanisms an operating system provides to allow the processes to manage shared data.
    - Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests.
  - No patch to kernel
- **Experiment**: Porting SELinux/AppArmor to embedded devices to learn more.

Questions?