CSE 40567 / 60567: Computer Security

Software Security 3
Homework #4 Due Tonight at 11:59PM Eastern Time

See **Assignments Page** on the course website for details
Mid-term Course Instructor Feedback (CIF)

10 Points of Extra Credit!
Return to Gabe by next Tuesday
Mid-term: In-Class March 9th

Questions?
Role of x.500 in LDAP

In essence, LDAP is just an access protocol to an x.500 directory service.

LDAP is commonly directly implemented in X.500 servers.
Pluggable Authentication Modules (PAM)

- Provides authentication modules for applications
  - Solves problem of developers writing their own authentication modules
  - Suite of shared libraries with common configurations

- During authentication, program invokes library routine `pam_authenticate`
  - The routine accesses configuration files in `/etc/pam.d`
  - Example: `sshd` will access `/etc/pam.d/sshd`
# Disallow non-root logins when /etc/nologin exists.
account  required  pam_nologin.so

# Uncomment and edit /etc/security/access.conf if you need to set complex
# access limits that are hard to express in sshd_config.
# account  required  pam_access.so

# Standard Un*x authorization.
@include common-account

# Standard Un*x session setup and teardown.
@include common-session

# Print the message of the day upon successful login.
session  optional  pam_motd.so # [1]

# Print the status of the user's mailbox upon successful login.
session  optional  pam_mail.so standard noenv # [1]

# Set up user limits from /etc/security/limits.conf.
session  required  pam_limits.so

# Set up SELinux capabilities (need modified pam)
# session  required  pam_selinux.so multiple

# Read environment variables from /etc/environment and
# /etc/security/pam_env.conf.
session  required  pam_env.so # [1]
# In Debian 4.0 (etch), locale-related environment variables were moved to
# /etc/default/locale, so read that as well.
session  required  pam_env.so user_readenv=1 envfile=/etc/default/locale
PAM Configuration Files

(1) account (2) required (3) pam_nologin.so

- First field describes the auth. related mechanism treated by the line
- Second field controls the calling of the modules
  - required means failure of the module makes authentication fail
- Third field is the name of the module (dynamic library)
### PAM Configuration Files

<table>
<thead>
<tr>
<th>Type</th>
<th>Requirement</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>required</td>
<td>pam_nologin.so</td>
</tr>
<tr>
<td>session</td>
<td>optional</td>
<td>pam_motd.so</td>
</tr>
<tr>
<td>session</td>
<td>optional</td>
<td>pam_mail.so std noenv</td>
</tr>
<tr>
<td>session</td>
<td>required</td>
<td>pam_limits.so</td>
</tr>
<tr>
<td>session</td>
<td>required</td>
<td>pam_env.so</td>
</tr>
<tr>
<td>session</td>
<td>required</td>
<td>pam_env.so user_readenv=1 envfile=/etc/default/locale</td>
</tr>
</tbody>
</table>

- **Stacking:** modules are invoked successively
- Configuration determines the order
  - Caller can make no assumptions about how the modules work
  - Authentication is in effect hidden from the application using PAM
SELinux

What if we uncomment this option?

# Set up SELinux capabilities (need modified pam)
# session  required    pam_selinux.so multiple

• Gives users and administrators more access control than the base OS provides

• Access can be constrained on such variables as which users and applications can access which resources

• Access controls are determined by a policy
  ‣ Can’t be changed by careless users or misbehaving applications

http://selinuxproject.org/
SELinux

- Adds finer granularity to access controls
  - Control fundamental OS operations: you specify who can unlink, append only, move a file, etc.
  - Specify access for network resources and interprocess communication (IPC)
- **SELinux users** are not equivalent to Linux users
  - They cannot change via su or sudo
  - Many Linux users will use the same SELinux user
  - SELinux users that are generic have the suffix "_u", such as user_u.
SELinux

- **SELinux roles** are defined by the policy
  - Examples: unprivileged user, web administrator, database administrator
  - Objects have the role `object_r`
  - Roles have the suffix "_r", such as `user_r`

- **SELinux types** are the primary means of determining access
  - A type has the suffix "_t", such as `user_t`
SELinux context framework

**Contexts** are attributes used to determine if access should be allowed between a process and an object.

Contexts consist of 3 required fields and 1 optional field:

```
user:role:type:range ← optional
```

Example with required fields:
```
system_u:system_r:xserver_t
```

Adding optional multi-level security:
```
system_u:system_r:xserver_t:SystemLow-SystemHigh
```
SELinux object classes and rules

**Object classes** are used in the policy and in access decisions to more finely specify what access is allowed

*file* object class has the permissions *create*, *read*, *write*, and *unlink*

*unix_stream_socket* object class has the permissions *create*, *connect*, and *sendto*

**Rules** bring all of these elements together:

allow user_t user_home_t:file { create read write unlink };

More on file system security coming right up…
Windows authentication model

• Security Support Provider Interface (SSPI)
  - Generic authentication framework for security-related operations in Windows
  - SSPI provides functions to Security Support Providers (DLLs)
    ‣ Analogous to PAM in Linux
    ‣ Authentication for OS and Apps
Windows SSPs

- **NT Lan Manager (NTLM)** - challenge/response authentication for non-domain authentication (SMB/CIFS) and legacy auth.

- **Kerberos** - Preferred for mutual client-server domain authentication

- Negotiate - single sign-on capability

- Secure Channel (aka SChannel) - encrypted tunneling for data (via AES)

- Digest SSP - Challenge/response based HTTP and SASL authentication between Windows and non-Windows systems without Kerberos

- Credential (CredSSP) - SSO and Network Level Authentication for Remote Desktop Services

- Distributed Password Authentication (DPA) - Internet authentication using digital certificates

- Public Key Cryptography User-to-User (PKU2U) - Peer-to-peer auth. using digital certificates between systems that are not part of a domain
Windows access control model

- Access Control Lists (ACLs)
  - Users can either be whitelisted or blacklisted
  - Security policy is set by group, not user
    - Primary method of centralized config. and control
    - Group policies can be associated with sites, domains or organizations

Facilitated via Active Directory
File System Security
Filesystems with multiple users

- Confidentiality and integrity of files must be satisfied
- Multi-user operating systems have file systems that provide permissions
- Permissions can be at the user, group or universal level
UNIX file permissions

An access control model that has stood the test of time
Permission attributes

- **r** read  
  read a file or list a directory's contents

- **w** write  
  write to a file or directory

- **x** execute  
  execute a file or recurse a directory tree

- **s** suid/sgid  
  run executable with perms. of user or group

- **t** sticky bit  
  owners have precedence for directory actions
Octal notation for permissions

<table>
<thead>
<tr>
<th>#</th>
<th>Permission</th>
<th>r</th>
<th>w</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>read, write and execute</td>
<td>r</td>
<td>w</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>read and write</td>
<td>r</td>
<td>w</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>read and execute</td>
<td>r</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>read only</td>
<td>r</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>write and execute</td>
<td>-</td>
<td>w</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>write only</td>
<td>-</td>
<td>w</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>execute only</td>
<td>-</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>none</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Permissions are set with the `chmod(1)` command
suid attribute

-\texttt{rwsr-xr--}\ 1\ root\ dip\ 321552\ Apr\ 21\ 2015\ pppd

- set-user-id (suid) attribute means a program is run with the privilege of the owner, and not the user invoking it

- Can be used safely in some circumstances
  - Example: creation of a normal user account for a specific piece of software several users need common access to

- Extremely dangerous to use when ownership is associated with privileged accounts
suid pitfalls

• Programmer is in a rush and makes a program suid root
  ▸ What are the implications of this?

• Difficult to track down who is invoking suid files

• Figuring out the interaction between suid files and ACLs enforced by filesystem is complicated
sgid attribute and pitfalls

• set-group-id (sgid) attribute means a program is run with the privilege of the group associated with that program, and not the user invoking it

• Programmer is in a rush and makes a program sgid root
  ‣ What are the implications of this?
Sticky bit

drwxrwxrwt 12 root root 40960 Jan 21 12:39 /tmp

• File system treats files in a directory in such a way that only the file’s owner (or superuser) can rename or delete the file

• Without sticky bit: any user with write and execute privileges can intentionally or unintentionally delete another user’s files in a directory

• Commonly used to protect scratch spaces
An old trick: hidden directories

Hidden files and directories are a convenient way to store configuration files in the root of a home directory:

```
walter@eve:~$ ls -a
.
..  current directory
.bash_history
directory above the current one
.bash_logout
.bashrc
```

Attacker creates a directory called . . .

Does anybody notice?
Windows file permissions

• NTFS support permissions on files and folders
• GUI applications like Windows Explorer are most often used to set access control
  ‣ Right click file or folder, select “Properties,” then click the Security Tab

Image Credit: http://www.ghacks.net/2011/05/24/ntfs-permissions-tools/
Basic NTFS file and folder permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning for Folders</th>
<th>Meaning for Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Permits viewing and listing of files and subfolders</td>
<td>Permits viewing or accessing of the file's contents</td>
</tr>
<tr>
<td>Write</td>
<td>Permits adding of files and subfolders</td>
<td>Permits writing to a file</td>
</tr>
<tr>
<td>Read &amp; Execute</td>
<td>Permits viewing and listing of files and subfolders as well as executing of files; inherited by files and folders</td>
<td>Permits viewing and accessing of the file's contents as well as executing of the file</td>
</tr>
<tr>
<td>List Folder Contents</td>
<td>Permits viewing and listing of files and subfolders as well as executing of files; inherited by folders only</td>
<td>N/A</td>
</tr>
<tr>
<td>Modify</td>
<td>Permits reading and writing of files and subfolders; allows deletion of the folder</td>
<td>Permits reading and writing of the file; allows deletion of the file</td>
</tr>
<tr>
<td>Full Control</td>
<td>Permits reading, writing, changing, and deleting of files and subfolders</td>
<td>Permits reading, writing, changing and deleting of the file</td>
</tr>
</tbody>
</table>

Image credit: Microsoft MSDN
## Special NTFS file and folder permissions

<table>
<thead>
<tr>
<th>Special permissions</th>
<th>Full Control</th>
<th>Modify</th>
<th>Read &amp; Execute</th>
<th>List Folder Contents (folders only)</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traverse Folder/Execute File</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Folder/Read Data</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Read Attributes</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Read Extended Attributes</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Create Files/Write Data</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Create Folders/Append Data</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Write Attributes</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Write Extended Attributes</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Delete Subfolders and Files</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Permissions</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Change Permissions</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Take Ownership</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronize</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Image credit: Microsoft TechNet

Encrypting a drive

• Two ways to do this
  ‣ Disk Encryption
  ‣ File System Encryption

• Addresses possibility of an attacker circumventing OS filesystem controls by reading the data via external means

• In practice, the implementation and strength of these approaches is quite different
Disk Encryption

• aka Full Disk Encryption (FDE)
• Protects individual disk blocks
• Each block (typically 512 or 2,048 bytes) is encrypted
  ‣ CBC Mode
    ‣ Block number is used as the IV
    ‣ Includes blocks on the free list
• Encryption is agnostic to operating system file formats
Disk Encryption Implementations

• Can be done via the OS or by the disk hardware

• Software: Bitlocker (Windows), FileVault (OS X), eCryptfs (Linux), softraid (OpenBSD)

• Hardware: Hitachi, Micron, Seagate, Samsung, and Toshiba offer TCG OPAL SATA drives
  ‣ Key management takes place in the disk controller
  ‣ 128- or 256-bit encryption
  ‣ Authentication requires the CPU via software pre-boot authentication environment or a BIOS password
File System Encryption

- Protects individual files
  - Meta-data are exposed, including file size access patterns, and more
    ‣ **Leaks information** versus Disk Encryption
- Most useful for protecting remote file systems
  - Client-side unlocking difficult (how to you handle the free list?)
  - Space must be specified and allocated for the entire file system at creation time
- Possibility of different keys for different subtrees, held by different users
File System Encryption Implementations

- Common options
  - EFS extension of NTFS (Windows)
  - Transparent encryption extension of EXT4 (Linux)
  - Transparent encryption extension of F2FS (Linux)
Exploiting Bugs in Software
Software bugs have a profound impact on security

1. Buffer overflows
   ‣ Exceeding memory bounds can have unanticipated consequences

2. Integer manipulation attacks
   ‣ Overflows, underflows, wrap-around, or truncation can alter the execution flow of the stack

3. Format strings attacks
   ‣ Your printf() calls could be dangerous

4. Race conditions
   ‣ Happen when a transaction is carried out in two or more stages
Refresher on memory allocation in C

Much of today’s application programming is done in high-level languages like python, php, C# and Java, where memory management is transparent.

However, C/C++ is still the dominant language for systems programming.

**Advantage and disadvantage of C/C++:** provides low-level access to memory and constructs that map to machine instructions.