OS Security

Authentication

Radboud University Nijmegen, The Netherlands

Winter 2014/2015

What does an OS do?

Definition

An *operating system (OS)* is a computer program that manages access of processes (programs) to shared ressources.

What does an OS do?

Definition

An *operating system (OS)* is a computer program that manages access of processes (programs) to shared ressources.

Examples of shared resources

- Memory
- Input and Output (I/O) including
  - Files on the harddrive
  - Network
- Computation cycles on the processor(s)
- Peripheral hardware (keyboard, screen, . . .)

What does that mean for security?

- Operating system needs to decide whether processes are allowed to perform certain operations
- Obvious security disasters:
  - One process reading the memory of another process
  - A process reading a “secret” file
  - A process preventing other processes from operating
  - One process reading (keyboard) input meant for another process

Wait, what about users?

- Is the process with ID 4321 allowed to read the file `/home/peter/os-security/exam-2014.pdf`?

Wait, what about users?

- Is the process with ID 4321 allowed to read the file `/home/peter/os-security/exam-2014.pdf`?
- Is user `peter` allowed to read the file `/home/peter/os-security/exam-2014.pdf`?

Wait, what about users?

- Is the process with ID 4321 allowed to read the file `/home/peter/os-security/exam-2014.pdf`?
- Is user `peter` allowed to read the file `/home/peter/os-security/exam-2014.pdf`?
- Need to map between a user (human) and a certain operation

Wait, what about users?

- Is the process with ID 4321 allowed to read the file `/home/peter/os-security/exam-2014.pdf`?
- Is user `peter` allowed to read the file `/home/peter/os-security/exam-2014.pdf`?
- Need to map between a user (human) and a certain operation

Definition

Authentication is the means by which it is determined that a particular entity (typically a human) intends to perform a given operation.

Wait, what about users?

- Is the process with ID 4321 allowed to read the file `/home/peter/os-security/exam-2014.pdf`?
- Is user `peter` allowed to read the file `/home/peter/os-security/exam-2014.pdf`?
- Need to map between a user (human) and a certain operation

Definition

Authentication is the means by which it is determined that a particular entity (typically a human) intends to perform a given operation.

- Typically perform *user authentication* as a login procedure
- Start a shell mapped to the logged-in user
- A shell is (basically) an interface to run other programs
- All programs run from this shell are mapped to the logged-in user

Problems of authentication

- Authentication is not necessarily perfect. Have to balance
  - *Fraud rate*: authentication that passed, but should have failed
  - *Insult rate*: authentication that failed, but should have passed

Problems of authentication

- Authentication is not necessarily perfect. Have to balance
  - *Fraud rate*: authentication that passed, but should have failed
  - *Insult rate*: authentication that failed, but should have passed
- User authentication does not catch everything:
  - Programs may perform operations that are not requested (or intended) by the user
  - Programs may not perform operations that are requested (or intended) by the user

Problems of authentication

- Authentication is not necessarily perfect. Have to balance
  - *Fraud rate*: authentication that passed, but should have failed
  - *Insult rate*: authentication that failed, but should have passed
- User authentication does not catch everything:
  - Programs may perform operations that are not requested (or intended) by the user
  - Programs may not perform operations that are requested (or intended) by the user
- Can perform *operation authentication*:
  - Ensure that a given operation is performed on request of a given user
  - Only feasible for very important operations

Problems of authentication

- Authentication is not necessarily perfect. Have to balance
  - *Fraud rate*: authentication that passed, but should have failed
  - *Insult rate*: authentication that failed, but should have passed
- User authentication does not catch everything:
  - Programs may perform operations that are not requested (or intended) by the user
  - Programs may not perform operations that are requested (or intended) by the user
- Can perform *operation authentication*:
  - Ensure that a given operation is performed on request of a given user
  - Only feasible for very important operations
- Worst-case of authentication going wrong: *impersonation*
  - Authenticating as somebody else lets you perform all operations that this user is allowed to do
  - Authenticating as anybody else lets you perform arbitrary operations

User authentication

- Can authenticate through
  - *something you know* (typically a password)
  - *something you have* (typically a card or token)
  - *something you are* (biometrics)
- Multi-factor authentication combines two (or more) means of authentication

The user `root`

- UNIX and Linux have a special *superuser* called `root`
- The user ID of `root` is always 0

The user `root`

- UNIX and Linux have a special *superuser* called `root`
- The user ID of `root` is always 0
- `root` may access all files

The user `root`

- UNIX and Linux have a special *superuser* called `root`
- The user ID of `root` is always 0
- `root` may access all files
- `root` may change permissions on all files

The user `root`

- UNIX and Linux have a special *superuser* called `root`
- The user ID of `root` is always 0
- `root` may access all files
- `root` may change permissions on all files
- `root` may bind programs to network sockets with port number smaller than 1024

The user `root`

- UNIX and Linux have a special *superuser* called `root`
- The user ID of `root` is always 0
- `root` may access all files
- `root` may change permissions on all files
- `root` may bind programs to network sockets with port number smaller than 1024
- `root` may "impersonate" any other user
- A process belonging to `root` may change its user ID to that of another user
- Once a process has changed from user ID 0 to another user ID, there is no way back

The user `root`

- UNIX and Linux have a special *superuser* called `root`
- The user ID of `root` is always 0
- `root` may access all files
- `root` may change permissions on all files
- `root` may bind programs to network sockets with port number smaller than 1024
- `root` may "impersonate" any other user
- A process belonging to `root` may change its user ID to that of another user
- Once a process has changed from user ID 0 to another user ID, there is no way back
- There are still certain actions that a program run by `root` cannot do (more next lecture)

The user `root`

- UNIX and Linux have a special *superuser* called `root`
- The user ID of `root` is always 0
- `root` may access all files
- `root` may change permissions on all files
- `root` may bind programs to network sockets with port number smaller than 1024
- `root` may "impersonate" any other user
- A process belonging to `root` may change its user ID to that of another user
- Once a process has changed from user ID 0 to another user ID, there is no way back
- There are still certain actions that a program run by `root` cannot do (more next lecture)
- **Security nightmare:** an attacker who gets `root` access

The Linux login procedure

- First process started after OS bootup is called `init`
- Main job of `init` is to start other processes

The Linux login procedure

- First process started after OS bootup is called `init`
- Main job of `init` is to start other processes
- `init` starts a program called `getty`
- `getty` stands for "get terminal" or "get teletypewriter"

The Linux login procedure

- First process started after OS bootup is called `init`
- Main job of `init` is to start other processes
- `init` starts a program called `getty`
- `getty` stands for "get terminal" or "get teletypewriter"
- `getty` starts `login`
- `init`, `getty` and `login` all run as root
- `login` prompts for username and password

The Linux login procedure

- First process started after OS bootup is called `init`
- Main job of `init` is to start other processes
- `init` starts a program called `getty`
- `getty` stands for "get terminal" or "get teletypewriter"
- `getty` starts `login`
- `init`, `getty` and `login` all run as root
- `login` prompts for username and password
  - Bad password: `login` exits, `init` starts new `getty`
  - Good password: `login` changes to new user and executes a shell

Attacks against passwords and countermeasures

- **Guessing attack:** Avoid short passwords and passwords from a dictionary

Attacks against passwords and countermeasures

- **Guessing attack:** Avoid short passwords and passwords from a dictionary
- **Over-the-shoulder attack:** password aging (replace passwords after a certain time), user training

Attacks against passwords and countermeasures

- **Guessing attack:** Avoid short passwords and passwords from a dictionary
- **Over-the-shoulder attack:** password aging (replace passwords after a certain time), user training
- **Automated on-line guessing:** Limit the number and/or rate of retries, report suspicious number of retries

Attacks against passwords and countermeasures

- **Guessing attack:** Avoid short passwords and passwords from a dictionary
- **Over-the-shoulder attack:** password aging (replace passwords after a certain time), user training
- **Automated on-line guessing:** Limit the number and/or rate of retries, report suspicious number of retries
- **Read the password file:** Use a (salted) one-way hash, prevent users from reading the file

Attacks against passwords and countermeasures

- **Guessing attack:** Avoid short passwords and passwords from a dictionary
- **Over-the-shoulder attack:** password aging (replace passwords after a certain time), user training
- **Automated on-line guessing:** Limit the number and/or rate of retries, report suspicious number of retries
- **Read the password file:** Use a (salted) one-way hash, prevent users from reading the file
- **Automated offline attacks:** Use a slow one-way hash, good passwords

Attacks against passwords and countermeasures

- **Guessing attack:** Avoid short passwords and passwords from a dictionary
- **Over-the-shoulder attack:** password aging (replace passwords after a certain time), user training
- **Automated on-line guessing:** Limit the number and/or rate of retries, report suspicious number of retries
- **Read the password file:** Use a (salted) one-way hash, prevent users from reading the file
- **Automated offline attacks:** Use a slow one-way hash, good passwords
- **Spoofing attacks** (present a fake login window): Trusted path for login

Attacks against passwords and countermeasures

- **Guessing attack:** Avoid short passwords and passwords from a dictionary
- **Over-the-shoulder attack:** password aging (replace passwords after a certain time), user training
- **Automated on-line guessing:** Limit the number and/or rate of retries, report suspicious number of retries
- **Read the password file:** Use a (salted) one-way hash, prevent users from reading the file
- **Automated offline attacks:** Use a slow one-way hash, good passwords
- **Spoofing attacks** (present a fake login window): Trusted path for login
- **Eavesdropping attacks** (key logging, acoustic attacks): physical security

/etc/passwd

- Linux uses the file `/etc/passwd` to store user login information
- Each line has 7 fields, separated by ':', e.g.:

  ```
  peter:x:1000:1000:Peter Schwabe,,,:/home/peter:/bin/bash
  ```

/etc/passwd

- Linux uses the file `/etc/passwd` to store user login information
- Each line has 7 fields, separated by ':', e.g.:

  ```
  peter:x:1000:1000:Peter Schwabe,,,:/home/peter:/bin/bash
  ```

- 1. field: Username

/etc/passwd

- Linux uses the file `/etc/passwd` to store user login information
- Each line has 7 fields, separated by ':', e.g.:

  ```
  peter:x:1000:1000:Peter Schwabe,,,:/home/peter:/bin/bash
  ```

- 1. field: Username
- 2. field: Password information, 'x' means that the password hash is separately stored in `/etc/shadow`

/etc/passwd

- Linux uses the file `/etc/passwd` to store user login information
- Each line has 7 fields, separated by ':', e.g.:

  ```
  peter:x:1000:1000:Peter Schwabe,,,:/home/peter:/bin/bash
  ```

- 1. field: Username
- 2. field: Password information, 'x' means that the password hash is separately stored in `/etc/shadow`
- 3. field: User ID (assigned to every process started by the user)

/etc/passwd

- Linux uses the file `/etc/passwd` to store user login information
- Each line has 7 fields, separated by ':', e.g.:

  ```
  peter:x:1000:1000:Peter Schwabe,,,:/home/peter:/bin/bash
  ```

- 1. field: Username
- 2. field: Password information, 'x' means that the password hash is separately stored in `/etc/shadow`
- 3. field: User ID (assigned to every process started by the user)
- 4. field: Group ID (more later)

/etc/passwd

- Linux uses the file `/etc/passwd` to store user login information
- Each line has 7 fields, separated by ':', e.g.:

  ```
  peter:x:1000:1000:Peter Schwabe,,,:/home/peter:/bin/bash
  ```

- 1. field: Username
- 2. field: Password information, 'x' means that the password hash is separately stored in `/etc/shadow`
- 3. field: User ID (assigned to every process started by the user)
- 4. field: Group ID (more later)
- 5. field: Comment describing the user

/etc/passwd

- Linux uses the file `/etc/passwd` to store user login information
- Each line has 7 fields, separated by ':', e.g.:

  `peter:x:1000:1000:Peter Schwabe,,,:/home/peter:/bin/bash`

- 1. field: Username
- 2. field: Password information, 'x' means that the password hash is separately stored in `/etc/shadow`
- 3. field: User ID (assigned to every process started by the user)
- 4. field: Group ID (more later)
- 5. field: Comment describing the user
- 6. field: Home directory
- 7. field: Login program (set to `/bin/false` or `/usr/sbin/nologin` for users that are not allowed to log in)

/etc/shadow

- Traditionally `/etc/passwd` stored users' password hashes
- Disadvantage: every user can read all hashes
- Easy to run offline (dictionary) attacks for every user

/etc/shadow

- Traditionally `/etc/passwd` stored users' password hashes
- Disadvantage: every user can read all hashes
- Easy to run offline (dictionary) attacks for every user
- Better approach: store password hashes in `/etc/shadow`
- `/etc/shadow` is readable only for `root`
- Most important information per entry (line)
  - Username
  - Password hash + salt (+algorithm)
  - Password expiration information

/etc/shadow

- Traditionally `/etc/passwd` stored users' password hashes
- Disadvantage: every user can read all hashes
- Easy to run offline (dictionary) attacks for every user
- Better approach: store password hashes in `/etc/shadow`
- `/etc/shadow` is readable only for `root`
- Most important information per entry (line)
  - Username
  - Password hash + salt (+algorithm)
  - Password expiration information
- Use '`*`' or '`!`' in the password field to lock the password
- Locking a password is different from using `/bin/false` as login program
- There may be other means to authenticate than the password

Password hashing algorithms

- Traditionally Linux used `crypt` for password hashing
- Truncate the password to 8 characters, 7 bits each
- Encrypt the all-zero string with modified DES with this 56-bit key
- Iterate encryption for 25 times (later: up to $2^{24} - 1$)
- Incorporate a 12-bit (later: 24-bit) salt

Password hashing algorithms

- Traditionally Linux used `crypt` for password hashing
- Truncate the password to 8 characters, 7 bits each
- Encrypt the all-zero string with modified DES with this 56-bit key
- Iterate encryption for 25 times (later: up to $2^{24} - 1$)
- Incorporate a 12-bit (later: 24-bit) salt
- Use *modified* DES to prevent attacks with DES hardware
- Originally computing the hash cost ≈ 1 second
- Too weak nowadays to offer strong protection

Password hashing algorithms

- Traditionally Linux used `crypt` for password hashing
- Truncate the password to 8 characters, 7 bits each
- Encrypt the all-zero string with modified DES with this 56-bit key
- Iterate encryption for 25 times (later: up to $2^{24} - 1$)
- Incorporate a 12-bit (later: 24-bit) salt
- Use *modified* DES to prevent attacks with DES hardware
- Originally computing the hash cost ≈ 1 second
- Too weak nowadays to offer strong protection
- Sucessors: MD5, `bcrypt` (based on Blowfish), SHA-2

Password hashing algorithms

- Traditionally Linux used `crypt` for password hashing
- Truncate the password to 8 characters, 7 bits each
- Encrypt the all-zero string with modified DES with this 56-bit key
- Iterate encryption for 25 times (later: up to $2^{24} - 1$)
- Incorporate a 12-bit (later: 24-bit) salt
- Use *modified* DES to prevent attacks with DES hardware
- Originally computing the hash cost ≈ 1 second
- Too weak nowadays to offer strong protection
- Sucessors: MD5, `bcrypt` (based on Blowfish), SHA-2
- Password hash string indicates which algorithm to use:
  - **1:** MD5;
  - **$2a$, $2b$, $2x$, $2y$:** variants of `bcrypt`
  - **5:** SHA-256; **6:** SHA-512

Password hashing algorithms

- Traditionally Linux used `crypt` for password hashing
- Truncate the password to 8 characters, 7 bits each
- Encrypt the all-zero string with modified DES with this 56-bit key
- Iterate encryption for 25 times (later: up to $2^{24} - 1$)
- Incorporate a 12-bit (later: 24-bit) salt
- Use *modified* DES to prevent attacks with DES hardware
- Originally computing the hash cost ≈ 1 second
- Too weak nowadays to offer strong protection
- Sucessors: MD5, `bcrypt` (based on Blowfish), SHA-2
- Password hash string indicates which algorithm to use:
  - **1:** MD5;
  - **$2a$, $2b$, $2x$, $2y$:** variants of `bcrypt`
  - **5:** SHA-256; **6:** SHA-512
- Maybe better algorithm in the future, see https://password-hashing.net/

How about Windows?

- Traditionally, Windows uses the LM hash (for "LanMan hash" or "LAN manager hash")
- Algorithm for LM hash:
  1. Restrict password to 14 characters

How about Windows?

- Traditionally, Windows uses the LM hash (for "LanMan hash" or "LAN manager hash")
- Algorithm for LM hash:
  1. Restrict password to 14 characters
  2. Convert password to all-uppercase

How about Windows?

- Traditionally, Windows uses the LM hash (for "LanMan hash" or "LAN manager hash")
- Algorithm for LM hash:
  1. Restrict password to 14 characters
  2. Convert password to all-uppercase
  3. Pad to 14 bytes

How about Windows?

- Traditionally, Windows uses the LM hash (for "LanMan hash" or "LAN manager hash")
- Algorithm for LM hash:
  1. Restrict password to 14 characters
  2. Convert password to all-uppercase
  3. Pad to 14 bytes
  4. Split into two 7-byte halves

How about Windows?

- Traditionally, Windows uses the LM hash (for "LanMan hash" or "LAN manager hash")
- Algorithm for LM hash:
  1. Restrict password to 14 characters
  2. Convert password to all-uppercase
  3. Pad to 14 bytes
  4. Split into two 7-byte halves
  5. Use each of the halves as a DES key

How about Windows?

- Traditionally, Windows uses the LM hash (for "LanMan hash" or "LAN manager hash")
- Algorithm for LM hash:
  1. Restrict password to 14 characters
  2. Convert password to all-uppercase
  3. Pad to 14 bytes
  4. Split into two 7-byte halves
  5. Use each of the halves as a DES key
  6. Encrypt the fixed ASCII string `KGS!@#$%`

How about Windows?

- Traditionally, Windows uses the LM hash (for "LanMan hash" or "LAN manager hash")
- Algorithm for LM hash:
  1. Restrict password to 14 characters
  2. Convert password to all-uppercase
  3. Pad to 14 bytes
  4. Split into two 7-byte halves
  5. Use each of the halves as a DES key
  6. Encrypt the fixed ASCII string `KGS!@#$%`
  7. Concatenate the two ciphertexts to obtain the LM hash

LM Hash weaknesses

- 14 printable ASCII characters give $\approx 2^{92}$ passwords

LM Hash weaknesses

- 14 printable ASCII characters give $\approx 2^{92}$ passwords
- Can crack the halves independently: 2^{46} for each half

LM Hash weaknesses

- 14 printable ASCII characters give $\approx 2^{92}$ passwords
- Can crack the halves independently: 2^{46} for each half
- All characters converted to upper case: 2^{43} for each half

LM Hash weaknesses

- 14 printable ASCII characters give $\approx 2^{92}$ passwords
- Can crack the halves independently: 2^{46} for each half
- All characters converted to upper case: 2^{43} for each half
- No salt, rainbow tables are feasible

LM Hash weaknesses

- 14 printable ASCII characters give $\approx 2^{92}$ passwords
- Can crack the halves independently: 2^{46} for each half
- All characters converted to upper case: 2^{43} for each half
- No salt, rainbow tables are feasible
- Passwords shorter than 8 characters produce hash ending in `0xAAD3B435B51404EE`

LM Hash weaknesses

- 14 printable ASCII characters give $\approx 2^{92}$ passwords
- Can crack the halves independently: 2^{46} for each half
- All characters converted to upper case: 2^{43} for each half
- No salt, rainbow tables are feasible
- Passwords shorter than 8 characters produce hash ending in `0xAAD3B435B51404EE`
- Cracking LM hashes is fairly easy, there are even online services, e.g., http://rainbowtables.it64.com/

NT hashes

- LM hash weaknesses were addressed by NT hash (or NTLM)
- NTLMv1 uses MD4 to hash passwords
- NTLMv2 uses MD5 to hash passwords
- Passwords are still unsalted

NT hashes

- LM hash weaknesses were addressed by NT hash (or NTLM)
- NTLMv1 uses MD4 to hash passwords
- NTLMv2 uses MD5 to hash passwords
- Passwords are still unsalted
- Until Windows XP, LM hashes were still enabled by default for backwards compatiblity

NT hashes

- LM hash weaknesses were addressed by NT hash (or NTLM)
- NTLMv1 uses MD4 to hash passwords
- NTLMv2 uses MD5 to hash passwords
- Passwords are still unsalted
- Until Windows XP, LM hashes were still enabled by default for backwards compatiblity
- Today, Windows uses multiple different approaches for passwords

NT hashes

http://www.hotforsecurity.com/blog/windows-8-stores-logon-passwords-in-plain-text-3914.html

Authentication by “what you have”

- Very common in the “physical world”, e.g., keys
- Digital world: Smart cards, USB tokens
- Private keys (e.g., for SSH)

Authentication by “what you have”

- Very common in the “physical world”, e.g., keys
- Digital world: Smart cards, USB tokens
- Private keys (e.g., for SSH)
- Can easily combine with password, e.g. on SSH private keys

Authentication by “what you have”

- Very common in the “physical world”, e.g., keys
- Digital world: Smart cards, USB tokens
- Private keys (e.g., for SSH)
- Can easily combine with password, e.g. on SSH private keys

Attacks and countermeasures

- **Stealing (or finding):** Protect possession

Authentication by "what you have"

- Very common in the "physical world", e.g., keys
- Digital world: Smart cards, USB tokens
- Private keys (e.g., for SSH)
- Can easily combine with password, e.g. on SSH private keys

Attacks and countermeasures

- **Stealing (or finding):** Protect possession
- **Copying:** Tamper-proof hardware, holograms, anti-counterfeiting techniques

Authentication by "what you have"

- Very common in the "physical world", e.g., keys
- Digital world: Smart cards, USB tokens
- Private keys (e.g., for SSH)
- Can easily combine with password, e.g. on SSH private keys

Attacks and countermeasures

- **Stealing (or finding):** Protect possession
- **Copying:** Tamper-proof hardware, holograms, anti-counterfeiting techniques
- **Replay attack:** device-dependent, use challenge-response

Authentication by “what you are”

- Fingerprint (fake fingerprint, cut off finger)
  http://www.heise.de/video/artikel/iPhone-5s-Touch-ID-hack-in-detail-1966044.html

Authentication by “what you are”

- Fingerprint (fake fingerprint, cut off finger)
  http://www.heise.de/video/artikel/iPhone-5s-Touch-ID-hack-in-detail-1966044.html
- Retina scans

Authentication by "what you are"

- Fingerprint (fake fingerprint, cut off finger)
  http://www.heise.de/video/artikel/iPhone-5s-Touch-ID-hack-in-detail-1966044.html
- Retina scans
- Voice match (distorted by cold, defeated by recordings)

Authentication by "what you are"

- Fingerprint (fake fingerprint, cut off finger)
  http://www.heise.de/video/artikel/iPhone-5s-Touch-ID-hack-in-detail-1966044.html
- Retina scans
- Voice match (distorted by cold, defeated by recordings)
- Handwriting (low accuracy, easy to fake)

Authentication by "what you are"

- Fingerprint (fake fingerprint, cut off finger)
  http://www.heise.de/video/artikel/iPhone-5s-Touch-ID-hack-in-detail-1966044.html
- Retina scans
- Voice match (distorted by cold, defeated by recordings)
- Handwriting (low accuracy, easy to fake)
- Keystroking, timing of keystrokes

Pluggable authentication modules

- Local login is not the only program that needs user authentication:
  - SSH (remote login)
  - Graphical login (GDM, LightDM)
  - Screen locks (screensaver)
  - su and sudo (more next lecture)

Pluggable authentication modules

- Local login is not the only program that needs user authentication:
  - SSH (remote login)
  - Graphical login (GDM, LightDM)
  - Screen locks (screensaver)
  - su and sudo (more next lecture)
- Idea: Centralize authentication, make functionality available through library
- This is handled by Pluggable Authentication Modules (PAM)

Pluggable authentication modules

- Local login is not the only program that needs user authentication:
  - SSH (remote login)
  - Graphical login (GDM, LightDM)
  - Screen locks (screensaver)
  - su and sudo (more next lecture)
- Idea: Centralize authentication, make functionality available through library
- This is handled by Pluggable Authentication Modules (PAM)
- Add a new module (e.g., for fingerprint authentication), directly available to all PAM enabled programs

PAM design

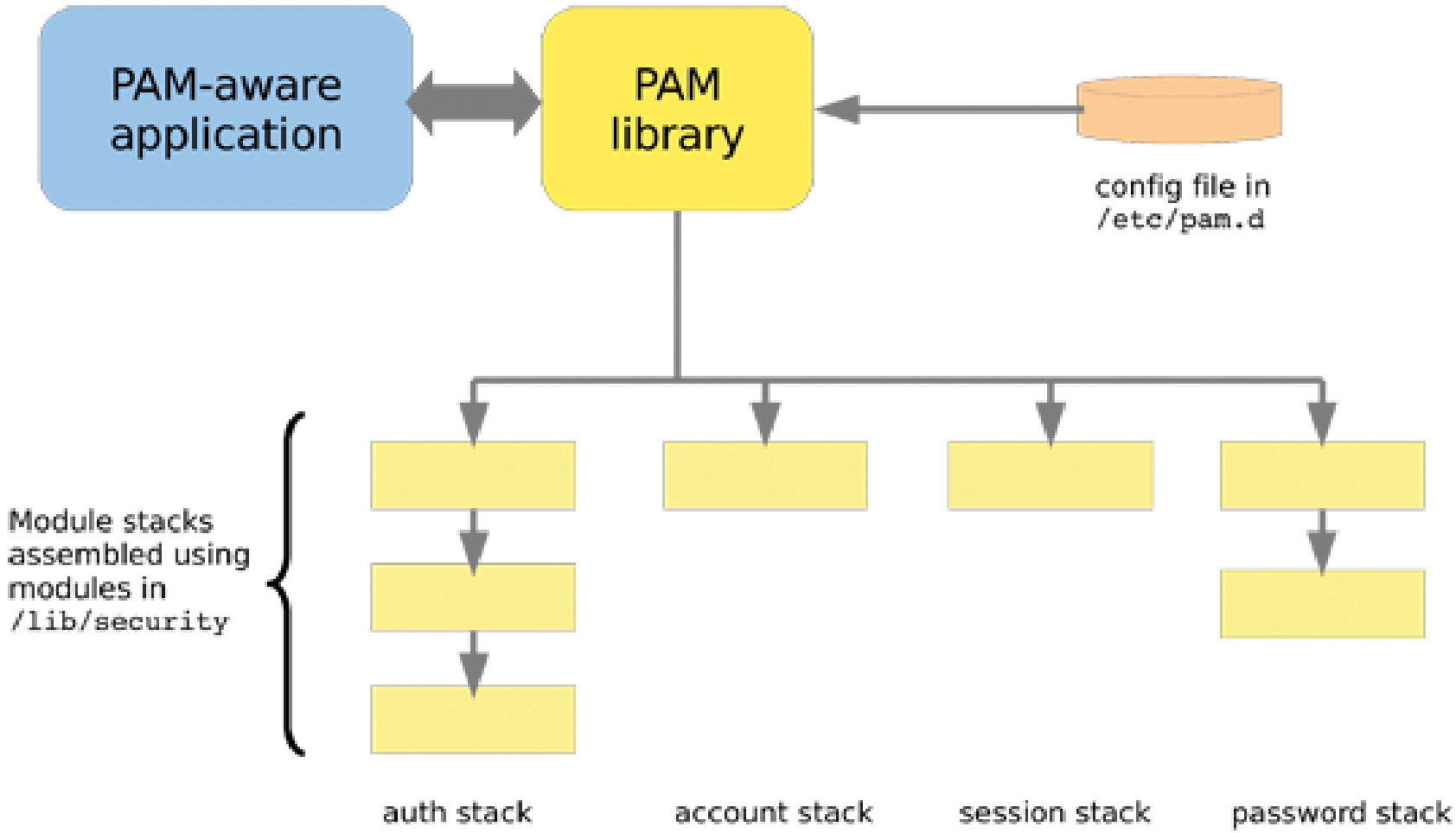


Image from http://www.tuxradar.com/content/how-pam-works

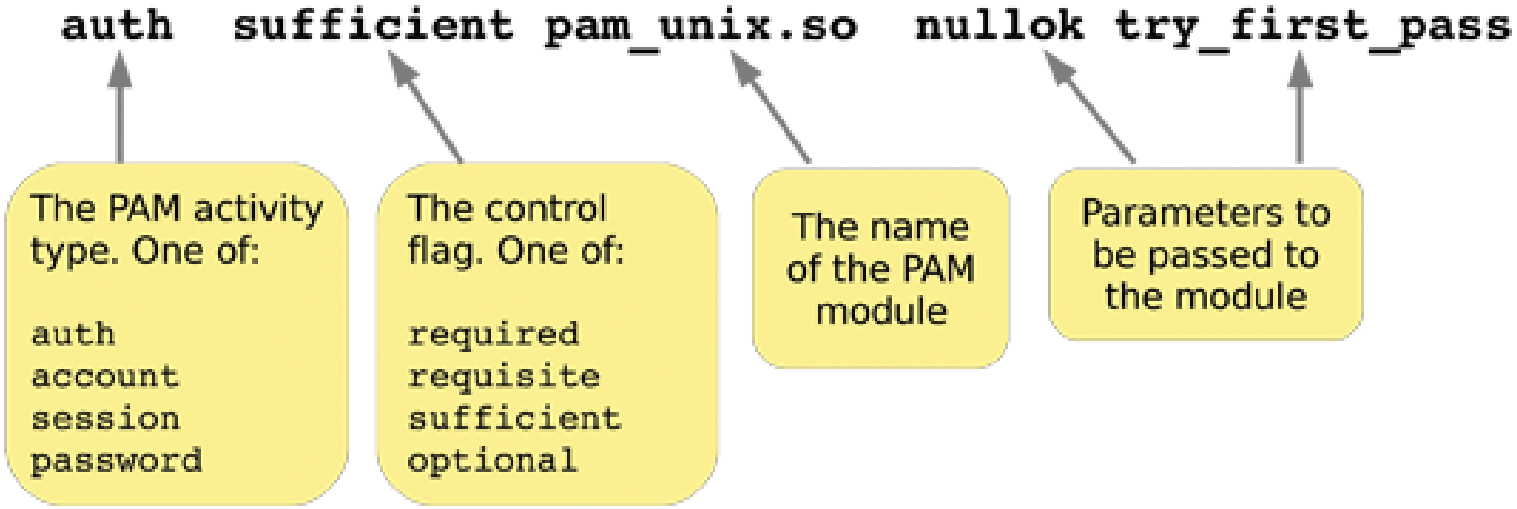
PAM activities

PAM knows 4 different authentication-related *activities*:

- **auth:** The activity of user authentication; typically by password, but can also use tokens, fingerprints etc.
- **account:** After a user is identified, decide whether he is allowed to log in. For example, can restrict login times.
- **session:** Allocates resources, for example mount home directory, set resource usage limits, print greeting message with information.
- **password:** Update the user's credentials (typically the password)

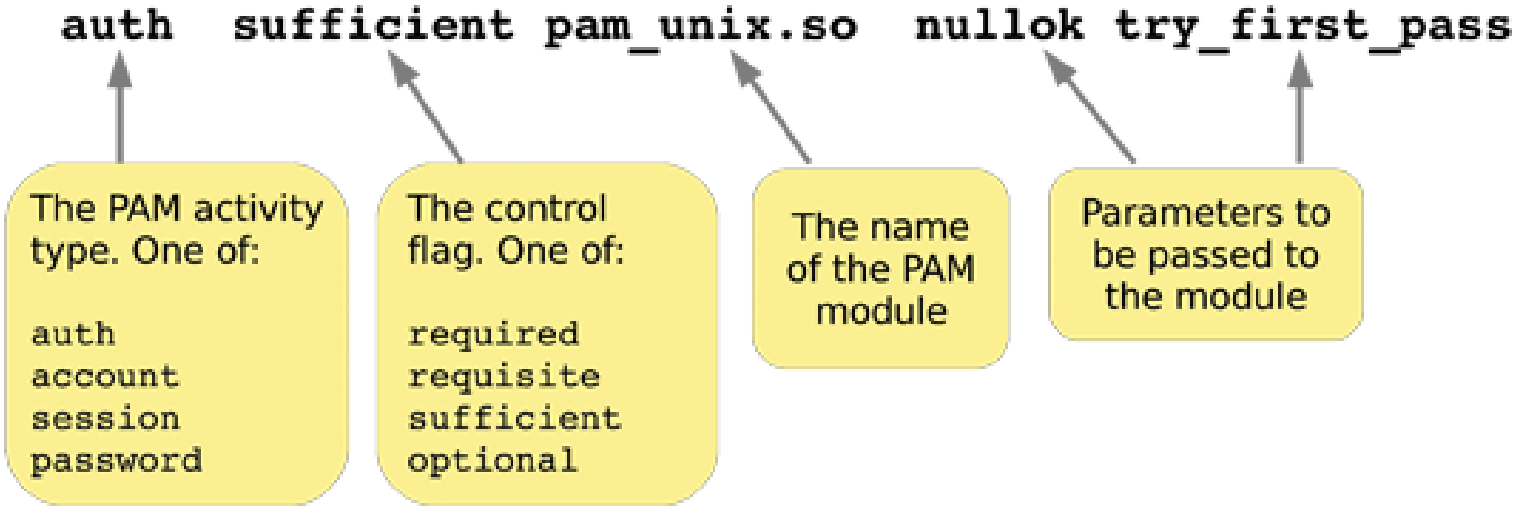
PAM configuration syntax

Configuration for program `progname` is in `/etc/pam.d/progname`



PAM configuration syntax

Configuration for program `progname` is in `/etc/pam.d/progname`



PAM control flags

- **requisite:** if module fails, immediately return failure and stop
- **required:** if module fails, return failure but continue
- **sufficient:** if module passes, return pass and stop
- **optional:** pass/fail result is ignored

Image source: http://www.tuxradar.com/content/how-pam-works

Examples of PAM modules

Name	Activities	Description
`pam_unix`	auth, session, password	Standard UNIX authentication through `/etc/shadow` passwords
`pam_permit`	auth, account, session, password	Always returns true
`pam_deny`	auth, account, session, password	Always returns false
`pam_rootok`	auth	Returns true iff you're root
`pam_warn`	auth, account, session, password	Write a log message to the system log
`pam_cracklib`	password	Perform checks of the password strength

Some PAM config examples

- Prevent all users from using `su` (`/etc/pam.d/su`)

```
auth        requisite   pam_deny.so
```

Some PAM config examples

- Prevent all users from using `su` (`/etc/pam.d/su`)

```
auth        requisite   pam_deny.so
```

- Prevent non-root users to halt (`/etc/pam.d/halt`)

```
auth        sufficient  pam_rootok.so
auth        required    pam_deny.so
```

Some PAM config examples

- Prevent all users from using `su` (`/etc/pam.d/su`)

```
auth        requisite   pam_deny.so
```

- Prevent non-root users to halt (`/etc/pam.d/halt`)

```
auth        sufficient  pam_rootok.so
auth        required    pam_deny.so
```

- Enforce passwords with at least 10 characters and at least 2 special characters, use SHA-512 for password hash (`/etc/pam.d/passwd`):

```
password    required    pam_cracklib.so minlen=10 ocredit=-2
password    required    pam_unix.so      sha512
```

Authentication over the network

- Large corporate networks want to keep user information central
- User is added to one central directory, can log into any machine

Authentication over the network

- Large corporate networks want to keep user information central
- User is added to one central directory, can log into any machine
- Various "simple" ways to set up the protocol:
  - Client sends password, server hashes and compares
  - Client sends hash, server compares
  - Server sends hash, client compares

Authentication over the network

- Large corporate networks want to keep user information central
- User is added to one central directory, can log into any machine
- Various "simple" ways to set up the protocol:
  - Client sends password, server hashes and compares
  - Client sends hash, server compares
  - Server sends hash, client compares
- Also more complex ways, e.g., challenge-response

Authentication over the network

- Large corporate networks want to keep user information central
- User is added to one central directory, can log into any machine
- Various "simple" ways to set up the protocol:
  - Client sends password, server hashes and compares
  - Client sends hash, server compares
  - Server sends hash, client compares
- Also more complex ways, e.g., challenge-response
- Possible disadvantage of central login server: single point of failure

NTLM and "pass the hash"

- Microsoft's LM and NTLM network authentication can send hash from the client, server compares hashes
- Attacker only needs to obtain the password *hash*
- The whole point of storing password hashes is gone
- Essentially, the hash becomes the password

NTLM and “pass the hash”

- Microsoft’s LM and NTLM network authentication can send hash from the client, server compares hashes
- Attacker only needs to obtain the password *hash*
- The whole point of storing password hashes is gone
- Essentially, the hash becomes the password
- This attack is known as “pass the hash” attack

NTLM and “pass the hash”

- Microsoft’s LM and NTLM network authentication can send hash from the client, server compares hashes
- Attacker only needs to obtain the password *hash*
- The whole point of storing password hashes is gone
- Essentially, the hash becomes the password
- This attack is known as “pass the hash” attack
- Conveniently automated in `metasploit`

NTLM and “pass the hash”

- Microsoft’s LM and NTLM network authentication can send hash from the client, server compares hashes
- Attacker only needs to obtain the password *hash*
- The whole point of storing password hashes is gone
- Essentially, the hash becomes the password
- This attack is known as “pass the hash” attack
- Conveniently automated in `metasploit`
- Almost any larger Windows network still has NTLM somewhere

NIS

- Network Information Service (NIS) invented by Sun
- Centrally administer users and hosts
- Server sends hash to the client, client compares
- Essentially, the advantage of `/etc/shadow` is lost
- NIS is still in use today, but not very common anymore

LDAP

- The Lightweight Directory Access Protocol (LDAP) is a network directory information protocol
- Developed by the IETF
- Includes means for user authentication

LDAP

- The Lightweight Directory Access Protocol (LDAP) is a network directory information protocol
- Developed by the IETF
- Includes means for user authentication
- Different modes involve sending the password to the server

LDAP

- The Lightweight Directory Access Protocol (LDAP) is a network directory information protocol
- Developed by the IETF
- Includes means for user authentication
- Different modes involve sending the password to the server
- Use these modes only over a TLS connection

LDAP

- The Lightweight Directory Access Protocol (LDAP) is a network directory information protocol
- Developed by the IETF
- Includes means for user authentication
- Different modes involve sending the password to the server
- Use these modes only over a TLS connection
- Even better: integrate LDAP with Kerberos

Kerberos

- State-of-the-art network authentication protocol
- Originally developed at MIT
- Two main versions: v4 (with some security problems) and v5
- Uses challenge-response, symmetric (and asymmetric) crypto

Kerberos

- State-of-the-art network authentication protocol
- Originally developed at MIT
- Two main versions: v4 (with some security problems) and v5
- Uses challenge-response, symmetric (and asymmetric) crypto
- Included in most UNIX/Linux variants
- Together with LDAP forms the basis of Microsoft's Active Directory

Kerberos

- State-of-the-art network authentication protocol
- Originally developed at MIT
- Two main versions: v4 (with some security problems) and v5
- Uses challenge-response, symmetric (and asymmetric) crypto
- Included in most UNIX/Linux variants
- Together with LDAP forms the basis of Microsoft's Active Directory
- More in the lecture "Cryptography" next semester