Security Aspects

Advanced Operating Systems and Virtualization
Alessandro Pellegrini
A.Y. 2018/2019

Basic Security Aspects

1. Systems must be usable by legitimate users only
2. Access is granted on the basis of an authorization, and according to the rules that are established by some system administrator
   – As for point 1, an unusable system is useless
   – However, in several scenarios the attacker might only tailor system non-usability by legitimate users (so called DOS – Denial of Service attacks)

Baseline Security Approaches

- Cryptography
- Authentication / Capabilities
- Security enhanced operating systems

- Each approach targets specific security aspects
- They should be combined together to improve the overall security of the system

Security Aspects Already Mentioned

- Address randomization
- Kernel-level stack protection
- Userspace Namespaces
- Sigreturn Oriented Programming
- Read only permission to critical data/code even when running in kernel mode
- Determination of the presence of critical instructions (e.g. those updating CR0 in x86 machines) upon module insertions (as in Linux)
- Minimization of the exposition of kernel layout data (`%pK` and `/proc/kallsyms`)

User Authentication

- Users login via passwords
- The passwords' database is stored within two distinct files:
  - `/etc/passwd`
  - `/etc/shadow`
- `/etc/passwd` is accessible to every user
- `/etc/shadow` is accessible only by root

/etc/passwd

- `/etc/passwd` has the following format:
  - `username:passwd:UID:GID:full_name:directory:shell`

`username:Npge08pfz4wuk:503:100:The User:/home/username:/bin/sh`

- `Np` represents the salt (16 bit) and `ge08pfz4wuk` is the encrypted password
- When using shadowing, `/etc/passwd` has the format:
  `username:x:503:100:full_name:/home/username:/bin/sh`
- `x` is a placeholder, hence `/etc/passwd` no longer contains passwords

/etc/shadow

- `/etc/shadow` has the format:
  `username:passwd:ult:can:must:note:exp:disab:reserved`

- where:
  1. `username` is the user
  2. `passwd` is the encrypted password
  3. `ult` are the days from 1/1/1970 since the last password change
  4. `can` day interval after which it is possible to change the password
  5. `must` day interval after which the password must be changed
  6. `Note` day interval after which the user is prompted for password update
  7. `exp` days after which the account is disabled if password expires
  8. `disab` days from 1/1/1970 after which the account will be disabled
  9. `reserved` no usage – a reserved field

User IDs in Unix

- The username is only a placeholder
- What discriminates which user is running a program is the UID
- The same is for GID
- Any process is at any time instant associated with three different UIDs/GIDs:
  - Real: this tells who you are
  - Effective: this tells what you can actually do
  - Saved: this tells who you can become again

UID/GID management system calls

- `setuid()`/`seteuid()`: available only to UID/EUID equal to 0 (root)
- `getuid()`/`geteuid()`: queries available to all users
- Similar services exist for managing GID
- `setuid()` is “non reversible” in the value of the saved UID: it overwrites all the three used IDs
- `seteuid()` is reversible and does not prevent restoring a saved UID
- An EUID-root user can temporarily become a different EUID user and then resume EUID-root identity
- UID and EUID values are not forced to correspond to those registered in `/etc/passwd`

`su` and `sudo`

- Both these commands are setuid-root
- They enable starting with the EUID-root identity
- If a correct input password is given by the user, they move the real UID to root or the target user (in case of `su`)
- After moving the UID to root, `sudo` executes the target command

Address-based Service Enabling

- Based on the concept of Access Control List (ACL)
- Addresses of enabled users are explicitly specified
- It is useful for services exposed on a network
- An approach used in architectures such as:
  - super-servers (e.g. **inetd**: the internet daemon, **xinetd**: the extended internet demon)
  - TCP containers (e.g. **tcpd**)
- Also used since ext3 File System
  - `setfacl` and `getfacl` commands

UNIX inetd

- It controls services running on specific port numbers
- Upon connection or request arrival, it starts the actual target service and redirects sockets to stdout, stdin, stderr
- Association between port number and actual service has been based on the file `/etc/services`, with format:
    - ……
    - `ftp-data            20/tcp`
    - `ftp                 21/tcp`
    - `telnet              23/tcp`
    - ……
- The **inetd** daemon was initially conceived as a means for resource usage optimization
- It has been then extended to cope with security

inetd Configuration

- Configuration information for **inetd** is typically kept by `/etc/inetd.conf`
- Each managed service is associated with one line structure as
  - Service name, as expressed in `/etc/services`
  - Socket type (e.g. stream)
  - Socket protocol (e.g. TCP)
  - Service flag (wait/nowait) which determines the execution mode (concurrent or not)
  - The user id to be associated with the running service instance (e.g. root)
  - The executable file path (e.g. `/usr/sbin/telnetd`) and its arguments (if any)

xinetd Features

- It provides an extension of **inetd** relying on
  - Address based access control
  - Time frame based access control
  - Full log of run-time events
  - DOS prevention by putting limitation on
    - Maximum number of per-service instances
    - Maximum number of total server instances
    - Log file size
    - Per machine source-connections
- Its configuration file is `/etc/xinetd.conf`
- It can be generated relying on the PERL utility `xconv.pl`

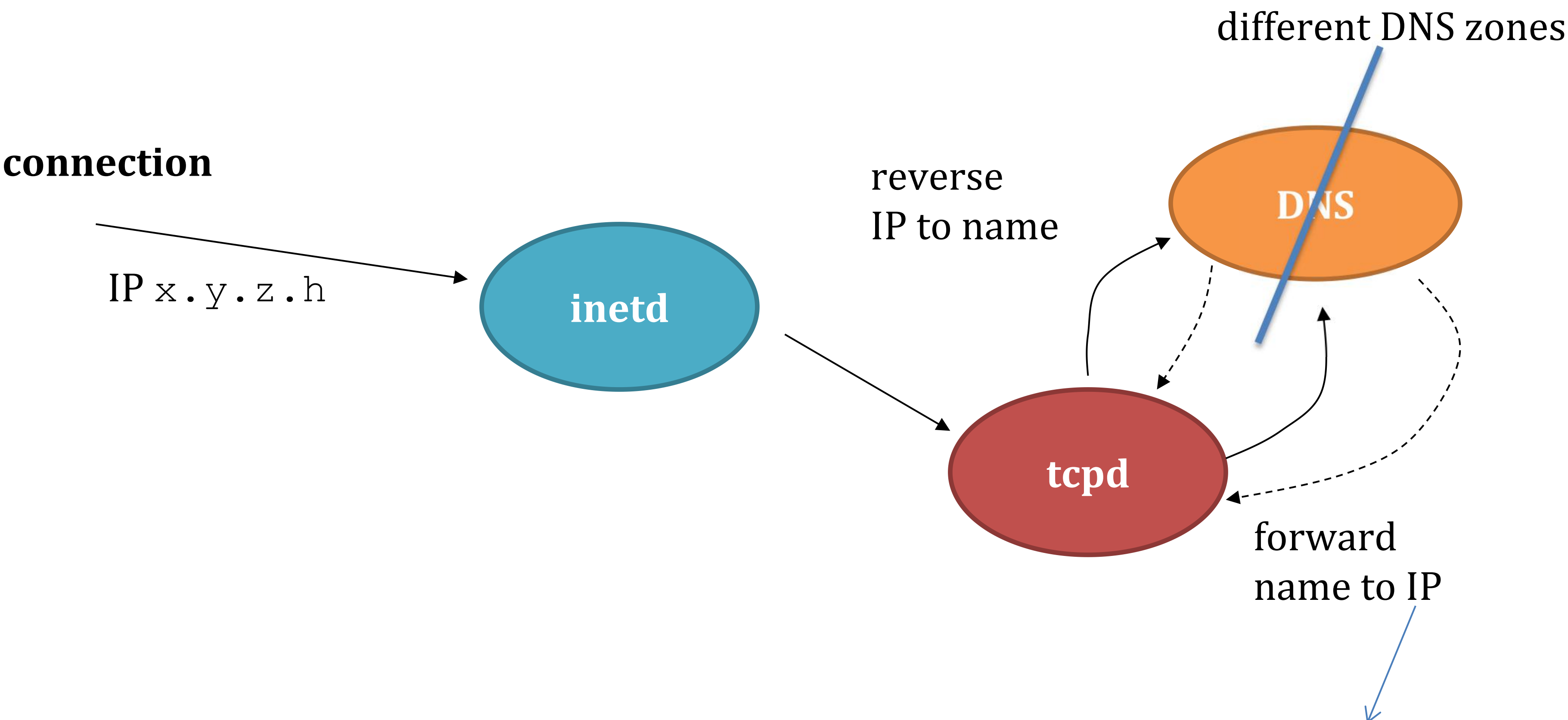
TCP daemons: tcpd

- The **tcpd** daemon wraps the services managed via **inetd**, so as to support access control rules
- **tcpd** is the actual server that is activated upon a request accepted by **inetd**
- **tcpd** receives as input the service specification
- Service management takes place by relying on rules coded in `/etc/hosts.deny` and `/etc/hosts.allow`
- Here we can find the specification of allowed or denied sources for a given service
- Each line is structured as `daemon_list : client_list`
- `ALL` is used to identify the whole set of managed services and all the hosts
- An example (access to all **inetd** services allowed from the local host)
  - `# /etc/hosts.allow`
  - `ALL:  127.0.0.1`

Reverse DNS tampering

- Usually host/domain specification occurs via symbolic names, rather than IP addresses
- Upon receiving a request/connection, **tcpd** checks with the source IP and executes a *reverse DNS* (rDNS) query to get the symbolic name of the source host
- An attacker can tamper with the reverse DNS query so as to reply with an allowed host/domain name
- To cope with this attack, **tcpd** typically performs both forwards DNS and reverse DNS queries so as to determine whether there is matching

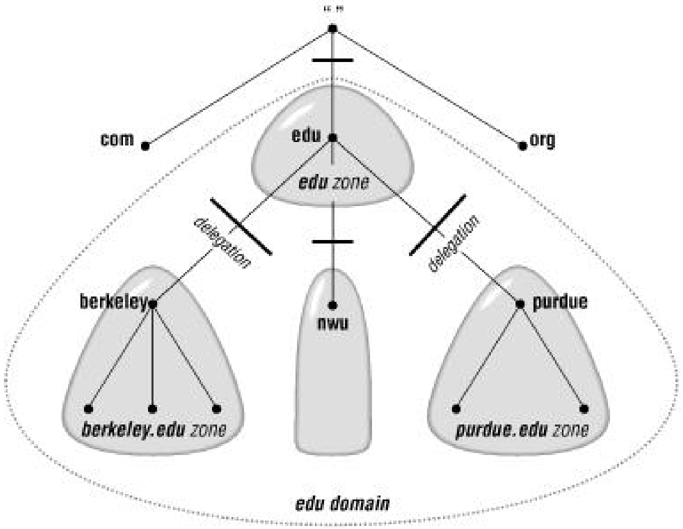
An example scheme



if equal to a `x.y.z.h` access is granted,
otherwise not

DNS Zones

- DNS is defined in zones
- The owner of a zone maps different addresses to different domain names in their zone
- writing www.example.com accesses the example.com zone, and associates via an A (alias) record the hostname www to a certain IP
- rDNS is based on the PTR record
- a PTR record is stored in a special zone called .in-addr.arpa. This zone is adminstrated by whoever owns the block of IP addresses.



Secure Operating Systems

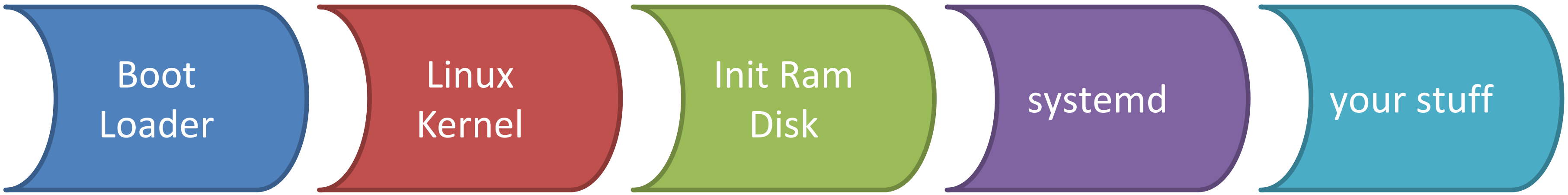
- A secure operating system is different from a conventional one because of the different granularity according to which we can specify resource access rules
- In this way, an attacker has lower possibility to make damages (e.g. in term of data access/manipulation) with respect to a conventional system
- Secure operating systems examples are:
  - SELinux (by NSA)
  - SecurLinux (by HP)
- Secure operating systems rely on the *Mandatory Access Control*

Security policies

- A security policy is named **discretionary** if ordinary users (including the administrator) are involved in the definition of security attributed (e.g. protection domains)

- A security policy is named **mandatory** if its logics and the actual definition of security attributes is demanded to a security policies' administrator (who is not an actual user/root of the system)

Boot Time Attacks



Startup Rootkits

Horse Pills

Services

Horse Pills

- A boot-time attack which is based on init scripts loaded into a ramdisk and the usage of namespaces
- An infected ramdisk can easily take control of the machine

Horse Pills

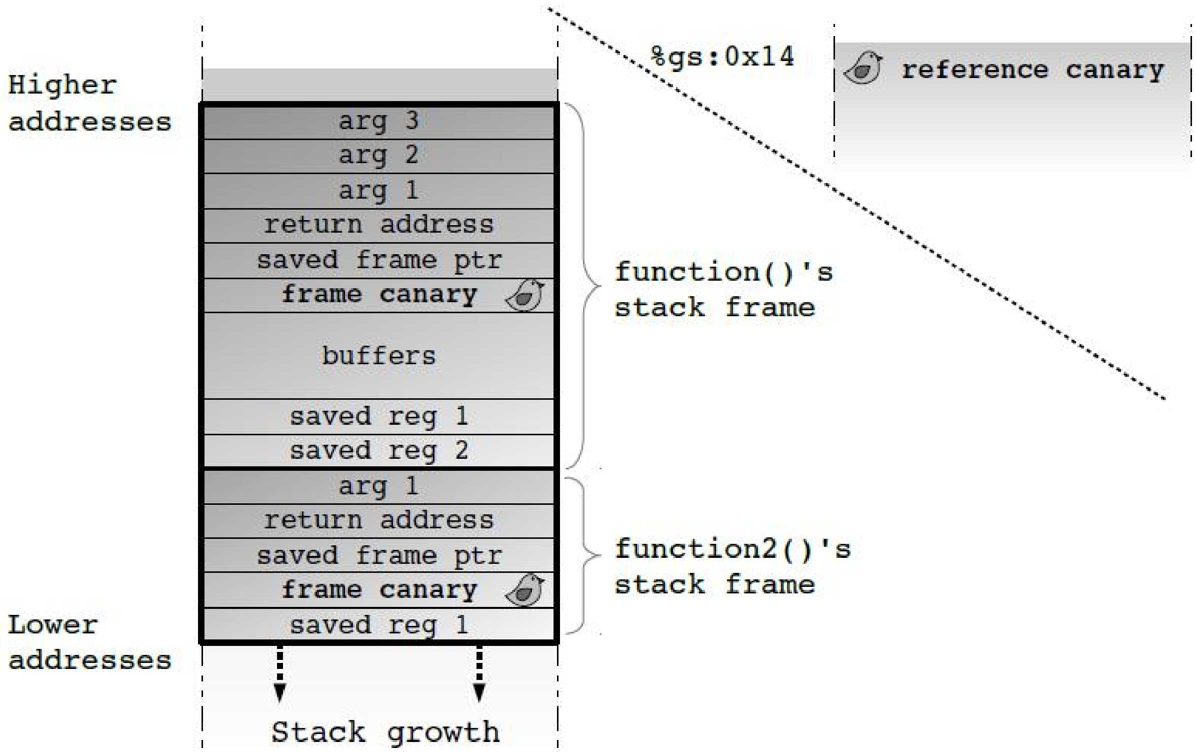
- What an infected ramdisk could do:
  - load modules
  - cryptsetup
  - find and mount rootfs
  - enumerate kernel threads
  - **`clone(CLONE_NEWPID, CLONE_NEWNS)`**
    - remount /proc
    - make fake kernel threads
    - clean up initrd
    - exec init
  - remount root
  - mount scratch space
  - `fork()`
    - hook initrd updates
    - backdoor shell
  - `waitpid()`
  - shutdown/reboot

Userspace System Internal Attacks

- An attack is said to be *internal* if it exploits an application that is installed and/or active in the system
- The attacker can either be an external user or one registered as a legitimate system user
- The classical internal attacks are:
  - Trojan horses
  - Login spoofing (ctrl+alt+del)
  - Logical bombs
  - Backdoors
  - Buffer overflows

Buffer overflow protection methods

- Stack randomization (upon `exec()` calls)
- Canary random tags as cross checks in the stack before returning



Non-Executable Address Space Regions

- x86_64 architectures provide page/region protection against instruction fetch
- The XD flag in the entries of the page tables
- This support was not available on i386 machines
- This is one reason why the `PROT_READ`/`PROT_EXEC` flags of `mmap()` are sometimes collapsed into the same protection semantic
- To enable instruction fetch from the stack on x86_64 you can use the "`-z execstack`" option of gcc

Exploit-based DOS: Ping of Death

- This attack appeared in 1996, and is based on an inconsistency within the IP protocol in common kernels
- IPv4 forbids a packet to be larger than 64 Kb
- IP allows for packet fragmentation, with reconstruction at the destination
- However, the offset of a fragment has been based on 16 bits within the header, so that we might specify a fragment that stands beyond the maximum packet bound
- In this case the operating system kernel writes the fragment out of the boundaries of the actual buffer selected for the receipt

Intrusion Detection Systems (IDS)

- Security can be improved, not definitely guaranteed
- We need systems able to detect that something wrong is going on
- This allows for:
  - Designing countermeasures for new attacks
  - Protect resources in case of an ongoing attack
- Intrusion detection systems (IDS) rely on two classical paradigms:
  - Anomaly Detection
  - Misuse Detection

Anomaly Detection

- This paradigm relies on the assumption that attacks are anomalous (infrequent), hence any anomalous event is assumed to represent an attack
- It is based on defining what are the admissible (normal) events, and in identifying any other event as an attack
- Events that are normal (but not identified as normal ones) can be identified as attacks (*false positives*)
- False positives can trigger countermeasures (e.g. system halt) that might not be actually required
- We might also experience *false negatives* in case an attack only relies on a sequence of admissible (normal) events

Misuse Detection

- It is based on a-priori identification of attack events which are registered into the IDS
- A true attack cannot be identified as such in case it is not coded in the a priori knowledge base, hence we can experience false negatives

Classical IDS types

- Honeypots
- File integrity checkers
  - Useful for libraries and modules
  - They can fail if the system is subverted
- Log checkers
  - Typically do not work in real time
- Network intrusion detection systems

Intrusion Prevention System: the Reference Monitor