CS-527 Software Security

OS Security

Asst. Prof. Mathias Payer

Department of Computer Science
Purdue University

TA: Kyriakos Ispoglou

https://nebelwelt.net/teaching/17-527-SoftSec/

Spring 2017

Table of Contents

Unix security

- Everything is a file.
- Keep things simple.
- Separate between user-space and kernel space.
- Isolate on process level (use virtual memory, unprivileged processes).
- (But allow communication through shared memory or IPC.)

Unix privilege model

- Distinguish between users. Allow groups of multiple users.
- Each file has a set of three read/write/execute sets for users, groups, and all other users.
- Access permissions restrict access to files, directories, programs
- Extremely simple but allows layering and chaining.
- More complex privilege models can be mapped through layering across, e.g., multiple directories.

Unix process permissions

- Real User ID (RUID): inherits UID (if unchanged) of the parent process, allows tracking of who started process.
- Effective User ID (EUID): the UID the process currently used for policy decisions (e.g., file access).
- Saved User ID (SUID): allows restoring previous UID.
- Now, how do you get rid of privileges?
- `int setresuid(uid_t ruid, uid_t euid, uid_t suid);`
- Unprivileged user processes may change the real UID, effective UID, and saved set-user-ID, each to one of: the current real UID, the current effective UID or the current saved set-user-ID.
- Privileged processes (on Linux, those having the CAP_SETUID capability) may set the real UID, effective UID, and saved set-user-ID to arbitrary values.
- Similar mechanism for groups

Table of Contents

Compartmentalization

- Break large monolithic over-privileged software into smaller components.
- Develop "fault compartments", that each fail individually.
- Goal: when one compartment fails, the others can still function.
- Think of compartments in ships, if one is flooded it can be separated.

Isolation

- Ensure that individual components can only interact through well-defined channels (APIs).
- No other components at the same privilege level can interact with the component without going through the API.
- Needs some higher privileged component to control interaction.

Principle of Least Privilege

- A privilege gives an entity the permission to access a resource.
- Enforce minimal privileges for intended purpose.
- Assumes compartmentalization (break software into smaller components) and isolation (ensure that components can only interact along exposed API).
- Drop privileges when you no longer need them.

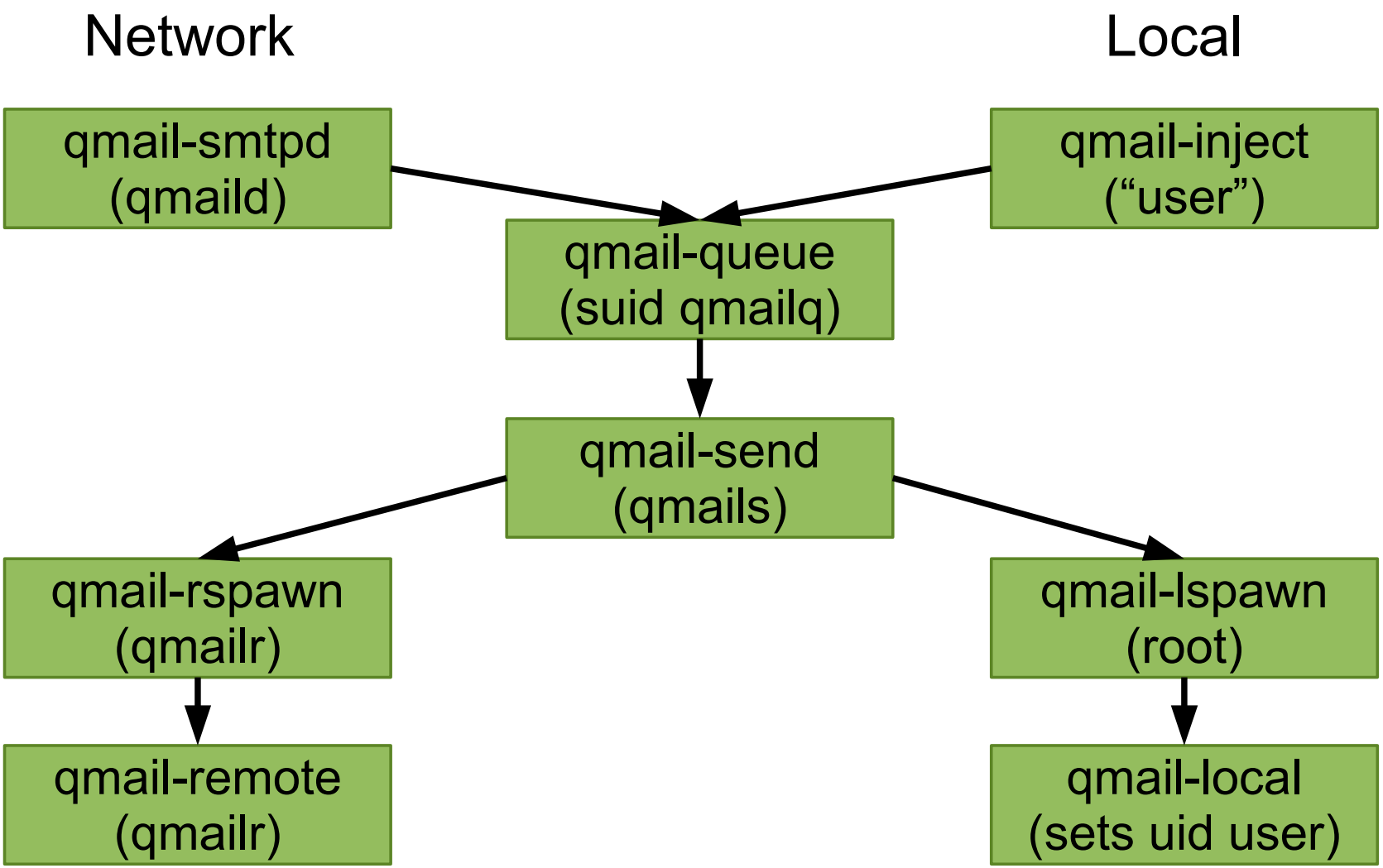
Example: mail agents

- Mail agents need to do a plethora of tasks: (i) send/receive data from the network, (ii) manage a pool of received/unsent messages, (iii) provide access to stored messages for each user.
- Two approaches: sendmail and qmail
- Sendmail uses a typical Unix approach with a large monolithic server and is known for the high complexity and previous security vulnerabilities.
- QMail uses a modern least privilege approach with a set of communicating processes.

Example: qmail

- Separate modules run under separate user IDs (isolation)
- Each user ID has only limited access to a subset of the resources (least privilege)
- Only one very small component runs as suid root
- Only one very small component running as root

Example: qmail



qmail components

qmaild/"user" : incoming email

suid qmaild : split message into contents and headers, signal qmail-send

qmail-send : send locally or remotely

qmail-lspawn : root, spawns qmail-local with ID of user

qmail-local : handles alias expansion, delivers locally, or signals qmail-queue if needed

qmail-remote : sends remote message

Table of Contents

Confining applications

- "Strongly" isolate processes from the rest of the system.
- Limit a process to a view of the file system `chroot` (Linux) and `jail` (FreeBSD): the process executes under a different root. What are the potential problems?
- An attacker can escape from a jail if there is a link to the outside (e.g., `..`).

System Call Interposition

- To change persistent state, an application must do system calls. So does an attack.
- System calls offer an interesting abstraction level
- Audit all the system calls and parameters, enforce access restrictions.
- Define a policy depending on what the program should do and define a least privilege policy of system calls that is required for the program to function.
- Can this be implemented through ptrace?
- ptrace suffers from TOCTTOU attacks (where an attacker-controlled thread changes the parameters after the check but before the execution of the system call).

Sandboxing

- The `seccomp` interface can be used to implement an efficient sandbox (e.g., Chrome uses `seccomp` on Linux).
- After executing `seccomp` only read, write, _exit, and sigreturn can be executed, all other system calls terminate the application.
- The sandbox first sets up I/O channels to a mediator, calls seccomp, and then can only request services from the trusted mediator.

Software-based Fault Isolation

- Application and untrusted code run in the same address space.
- The untrusted code may only read/write the untrusted data segment. How do you implement such a restriction?
- For each memory read/write the target address is masked:
  `and $0x00ff0000, %rax`
  `mov $0xc0fe0000, (%rax)`.
- Challenge for CISC architectures: jump to unaligned instructions:
  `mov $0x80cd01b0, (%rax)` → `mov $1, %al; int $0x80`
- NaCL solves the problem by aligning individual instructions.

Virtualizing applications: containers

- Full virtualization can be heavy-weight and incurs a lot of redundancy (the same kernel running multiple times, wasted disk/memory space).
- Containers solve this problem and allow the execution of "applications" in a virtual environment without replicated full virtual machines.
- Namespaces need to be separated: pid (processes), net (network), ipc (IPC), mnt (file system), uts (host name) namespaces.
- Control over the container through cgroups (controlling memory, CPU, and block I/O).
- AUFS, a special file system that allows different views of the same file system, adding permissions for containers and allowing to hide parts of the file system or have exclusive access to it.

Other isolation techniques

- More drastic isolation techniques ensure clear separation.
- Virtual Machine Introspection (VMI) assumes a trusted hypervisor that mediates inspections.
- An air gap physically separates machines.
- (An air gap may still not be enough!)
- A hard problem for isolation are side channels/covert channels.
- An even harder problem is specifying policies.

Table of Contents

Best practices

- Facts: (i) root can do everything, (ii) many programs run as root, (iii) most programs only execute few privileged actions. How can we reduce the security problem?
- Drop privileges as soon as possible (e.g., network service needs root privileges to bind to low port, drop privileges after binding).
- If you can't drop privileges: split into multiple components, drop privileges per component.
- Use more than one security mechanism.
- Secure the weakest link.
- Fail securely (and quickly).

Table of Contents

Time Of Check To Time Of Use Attacks

Exploits the fact that an attacker may change the environment *between* a check and the following operation.

```
1 // in a suid program
2 fl = "file";
3 if (access(fl, W_OK) != 0) {
4    exit(1);
5 }
6
7
8 fd = open(fl, O_WRONLY);
9 write(fd, buffer, sizeof(
      buffer));
```

```
1 symlink("/etc/passwd",
      "file");
```

Confused Deputy

- Privileged code may be accessed by unprivileged code through its exposed API. If the privileged code can be tricked tricked into abusing its privileges then this it is called a *confused deputy*. The confused deputy is a special form of privilege escalation.
- The confused deputy problem may arise when using ACL but not when using capabilities.
- The classic example is that of a pay-per-use compiler, where the compiler has access to a billing file. Users can specify input source files and output compiled files. An adversary may specify the billing file as output, overwriting the billing information.

Table of Contents

Summary

- Compartmentalize: break monolithic system into components.
- Isolation: Ensure that individual components can only interact through well-defined channels (APIs).
- Principle of least privilege: each of the components will only run with the minimal amount of permissions.
- System call interposition allows implementation of least privilege principle at the coarse-grained abstraction level of system calls.
- TOCTTOU and confused deputy attacks must be considered.
- Isolation comes in different flavours with different performance/protection/cost trade-offs.

Questions?

?