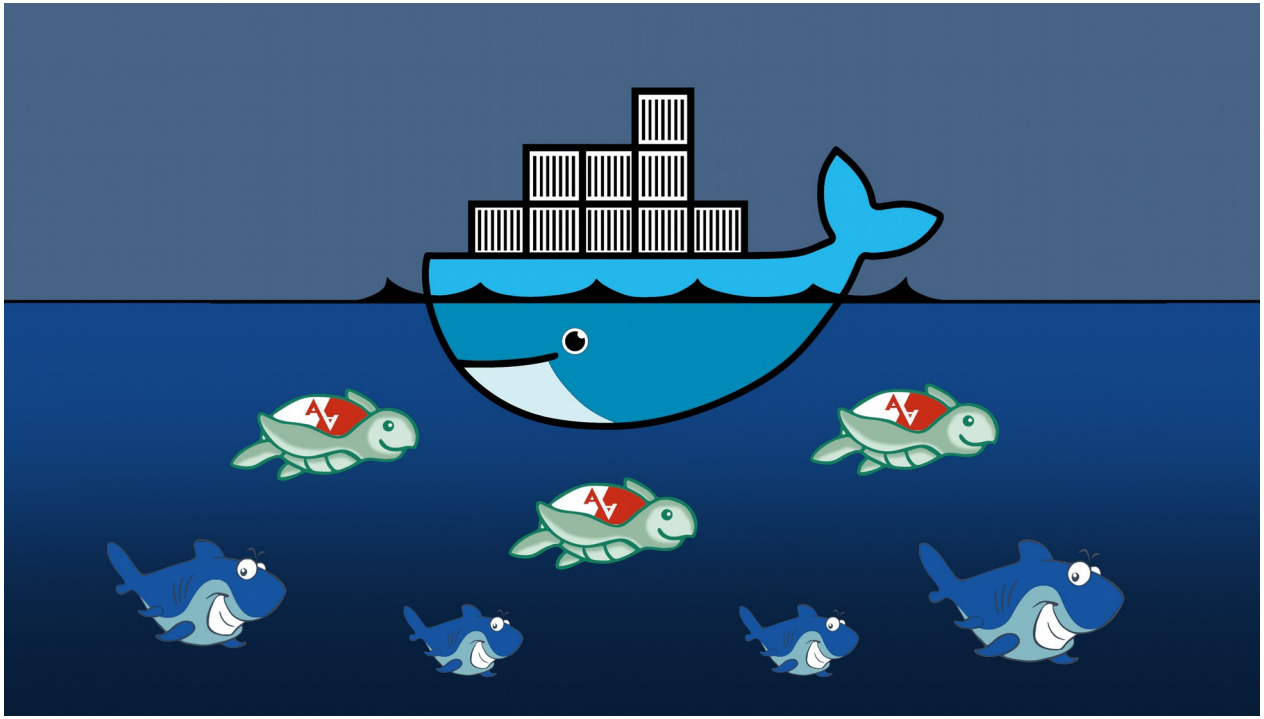
Security on Docker
Automatic security hardening of Docker containers using MAC, specialized in defending isolation



Diploma Thesis

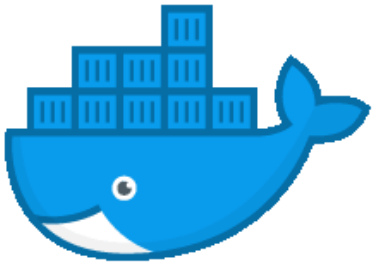
Fani Dimou

National Technical University of Athens
CSLab

Docker

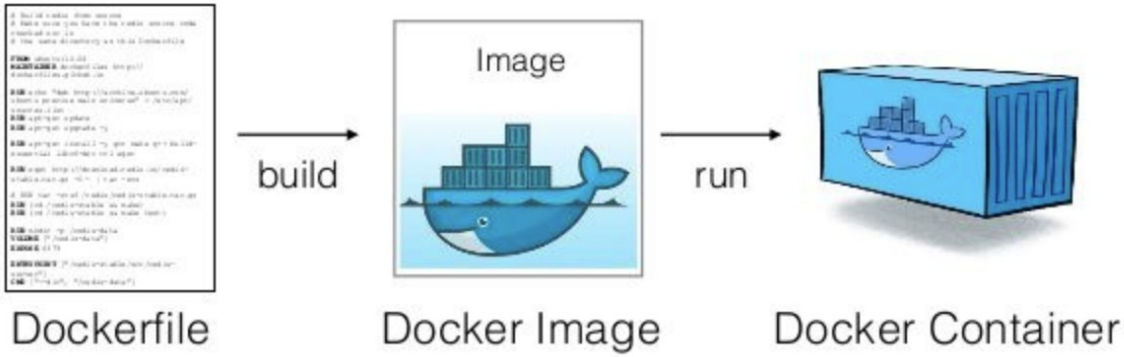
What is Docker?

A software, released in 2013, that performs operating-system-level virtualization. It is not a standalone software, but a platform to run and manage software packages called containers.



How it works?

It packages an application and all its dependencies together in the form of a docker container.



National Technical University of Athens CSLab

Docker

Why all the fuss?

Containers pre-existed docker, but docker presented a new way to use them easily.

Docker VS Virtual Machines

- Portability: Packages up code and all its dependencies, applications run in any infrastructure
- Speed: No OS to boot; applications are up in seconds
- Efficiency: Less OS overhead (CPU, RAM, power), lightweight, no wasting resources
- Easy to use

What about Security?

Containers are not as secure as VMs.
Security is a compromise we have to make…

We focus on **isolation** among host-container and container-container.

Security on Docker

How to secure a docker container?

- Use best practices (Dockerfiles, configuring containers etc)
- Seccomp
- SELinux
- AppArmor

AppArmor (Application Armor)
- Linux security module
- Implements Mandatory Access Control
- Supported by docker (security-opt option)
- Uses profiles to create a policy for an application
- Contains rules to allow and deny actions and access
- Path oriented
- Human readable profiles

Thesis contribution

1. Developed an open source software, SecureWilly, that automatically creates AppArmor profiles for any docker project in order to secure the containers and preserve the isolation.

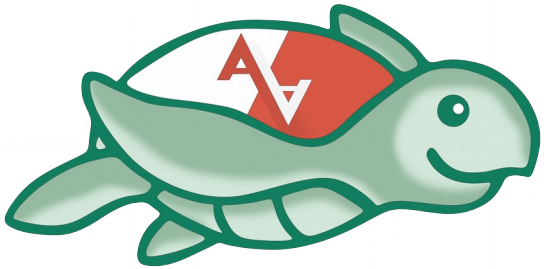
2. Extensive research on the vulnerable features of docker that can lead to attacks that violate isolation.

3. Implementation of several examples of breakout container attacks, in the context of ethical hacking, in order to assist security.

4. Alerting user about the vulnerabilities detected in the docker project, such as privileged mode or entering host's namespaces.

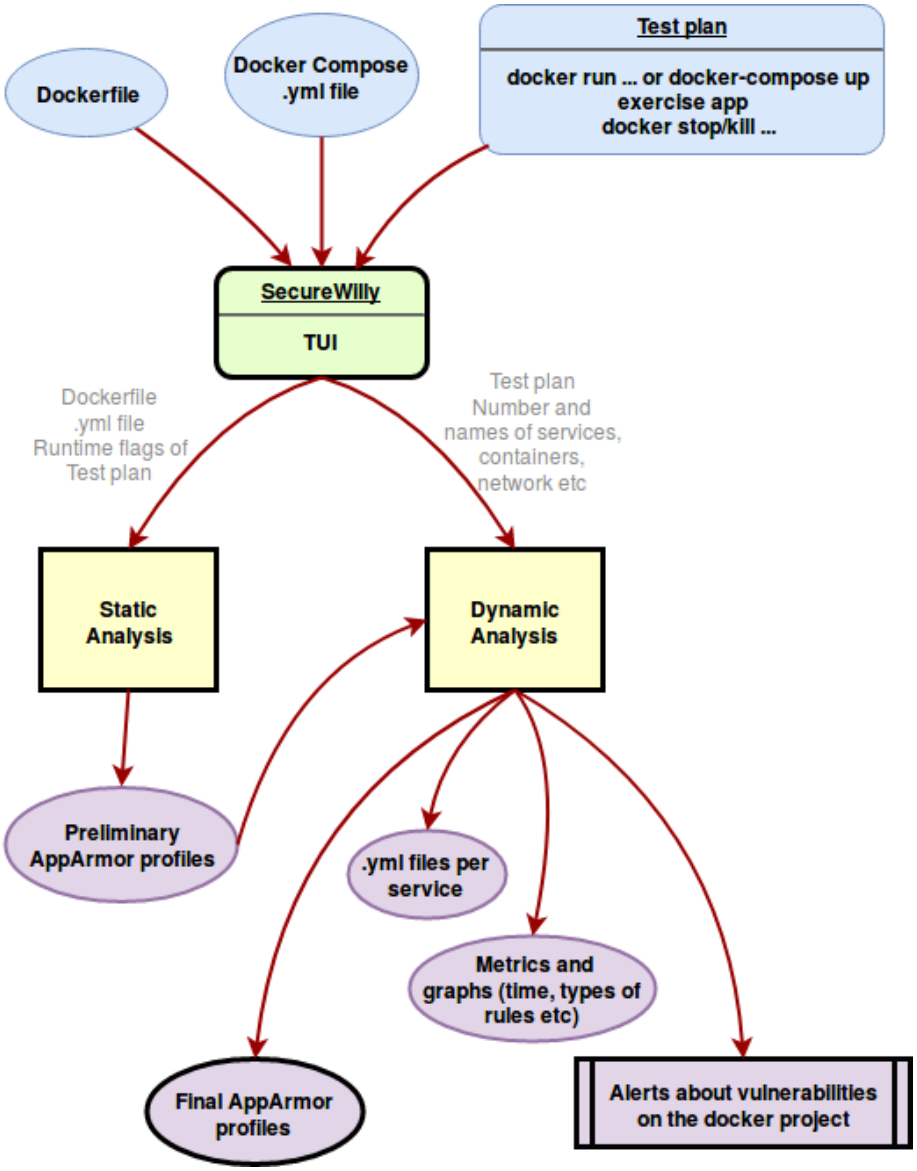
5. Creation of AppArmor profiles for an instance of Nextcloud platform (two profiles were created, one for the application of Nextcloud and one for the database that it uses), as experimental evaluation of SecureWilly.

SecureWilly



- Open source software
- Creates automatically AppArmor profiles for a docker project
- Handles single service and multi-service docker projects
- Service oriented; profiles adjusted to the task of each service, considering their co-operation too → specific and strict rules for each service
- Follows the Principle of Least Privilege → Contains the minimum set of rules needed in order to provide efficiency and security
- Efficient profiles, allowing user to complete the tasks specified
- Secure profiles, as any redundant actions are blocked

SecureWilly



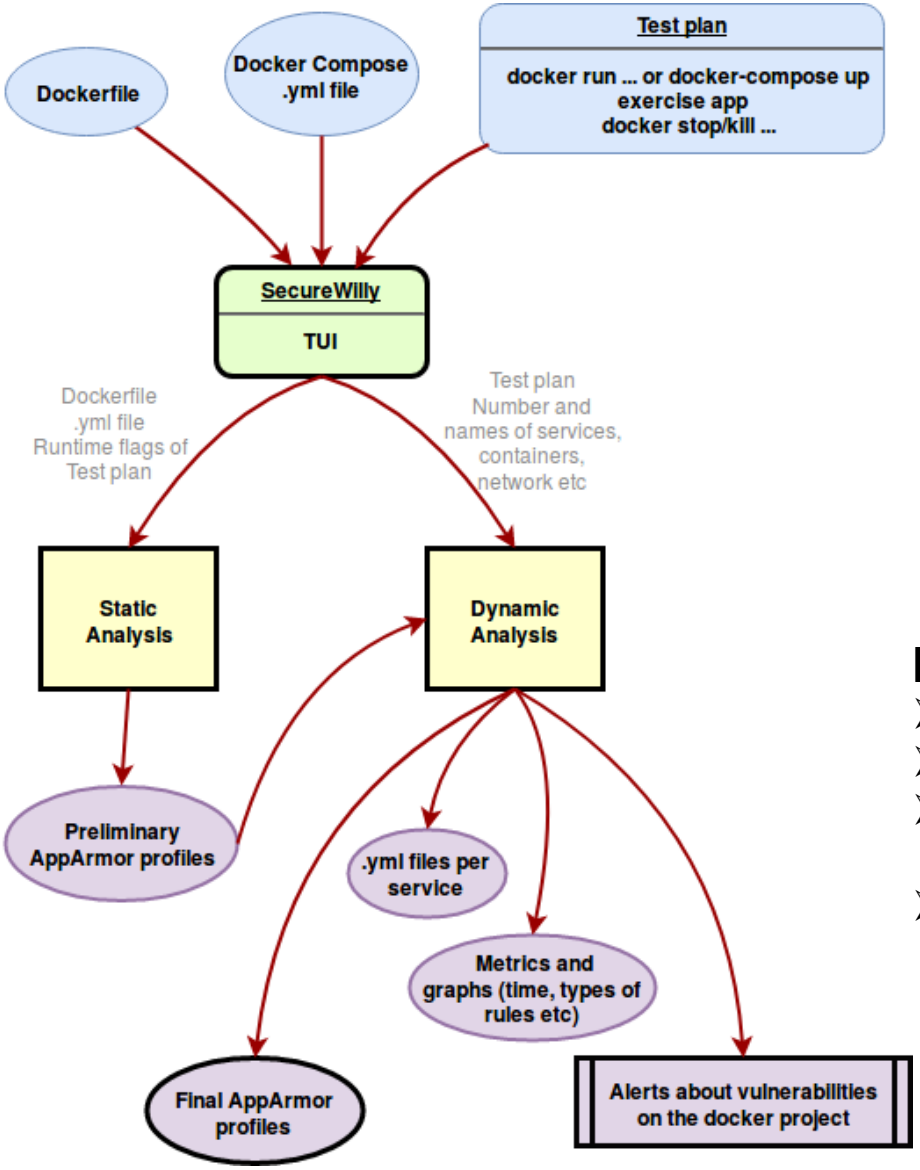
Input:

- Services: amount and names
- Dockerfile(s) path(s) / N
- Docker-compose.yml path / N
- Internal network: name / N
- Test plan:
  - Start container commands: docker start, docker run, docker-compose up
  - Exercise app's functionality
  - Stop container commands: docker stop, docker kill

Output:

- AppArmor profiles per service
- Alerts on vulnerabilities detected
- yml files per service (secondary)
- Metrics and graphs about time and the role of each service (secondary)

SecureWilly



Phase 1

Static Analysis — Dynamic Analysis

Static Analysis

- Static parser exports rules from the initiative code of the docker project.
- Creates a preliminary profile that will be used in Dynamic Analysis.

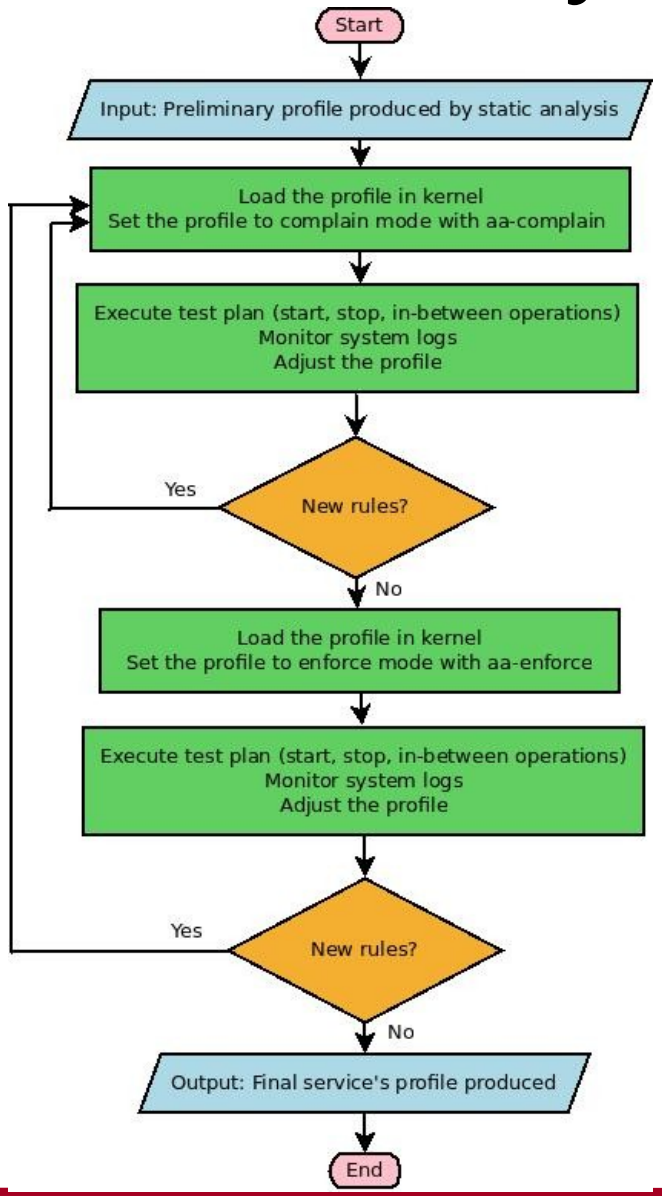
Dynamic Analysis

- Loads the preliminary profile in kernel.
- Runs docker project
- Dynamic parser monitors system logs and extracts new rules
- Updates profile until there are no new rules.

Static Analysis

- Dockerfile: The “recipe” to build a docker image
  - ~~VOLUME directories~~
  - EXPOSE ports
  - USER & RUN useradd
  - RUN chmod file
- Docker Compose: Included runtime parameters missing from Dockerfile
  - Volumes
  - Expose
  - Ports
  - Capabilities add/drop
  - Ulimits
  - Devices
- Runtime flags: Flags given in docker run commands in test plan
  - volumes: -v <source host path>:<container’s mountpoint>
  - expose: –expose <port>
  - ports: -p <host’s port>:<container’s port>
  - cap add: –cap-add <capability|ALL>
  - cap drop: –cap-drop <capability|ALL>
  - ulimits: –ulimit <resource type>=<soft>:<hard>
  - devices: –device <host’s device path>:<container’s device path>

Dynamic Analysis

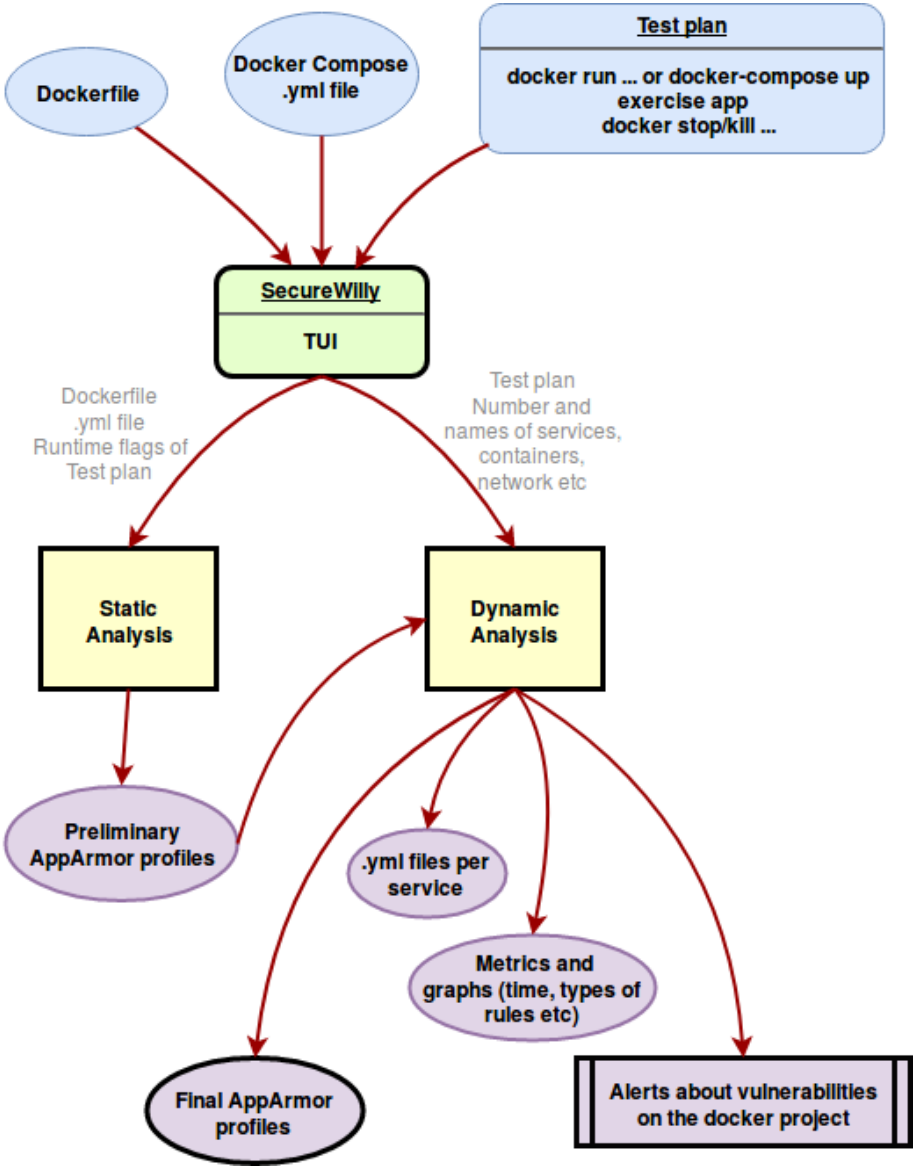


Complain mode
Allow every action, write denials to system logs

Enforce mode
Block any denied action according to the profile

- Loads the preliminary profile in kernel.
- Set it to complain mode
- Run test plan
- Monitors system logs and extracts new rules
- Updates profile until there are no new rules.
- Do the same for enforce mode

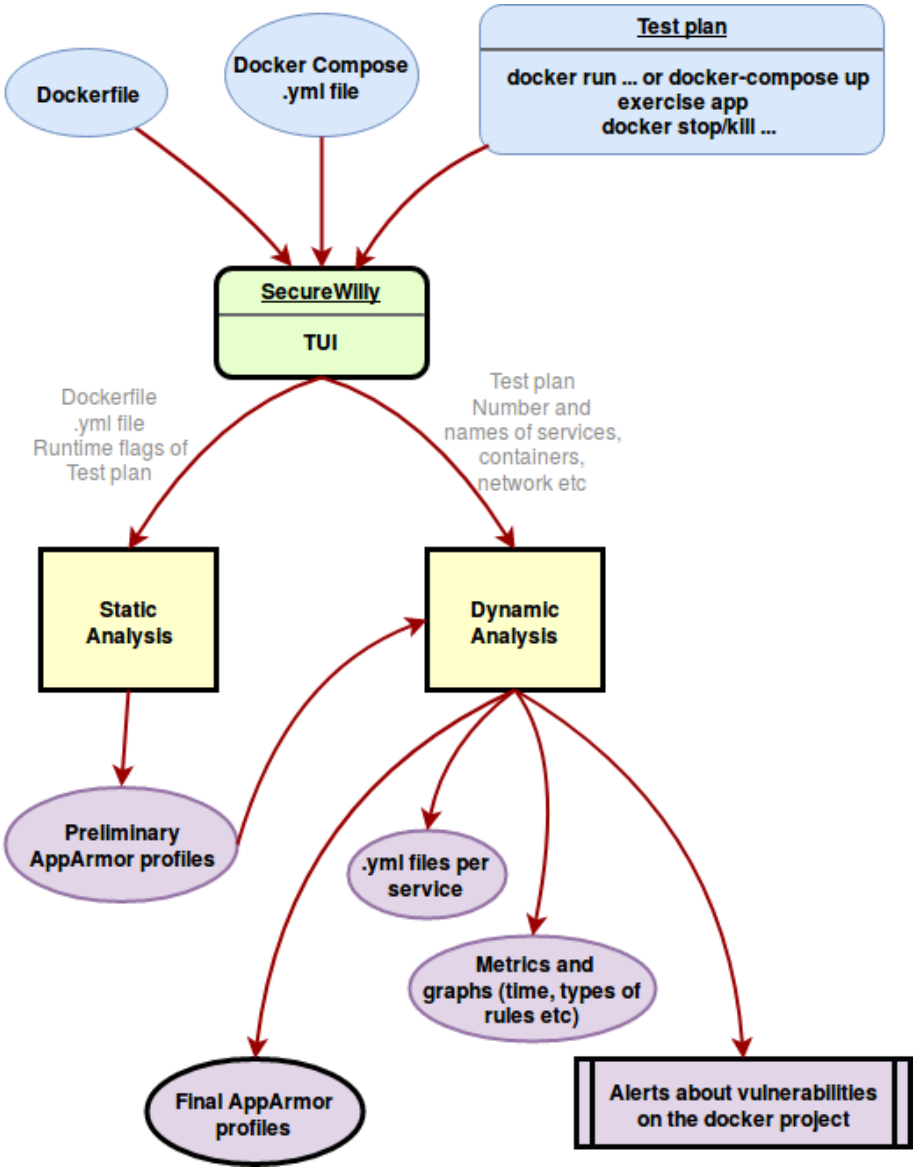
SecureWilly



Demo

National Technical University of Athens
CSLab

SecureWilly



Phase 1 provided:

- Efficient profiles → Adjusted to the task described through the test plan
- Specific and strict profiles → Least possible rules

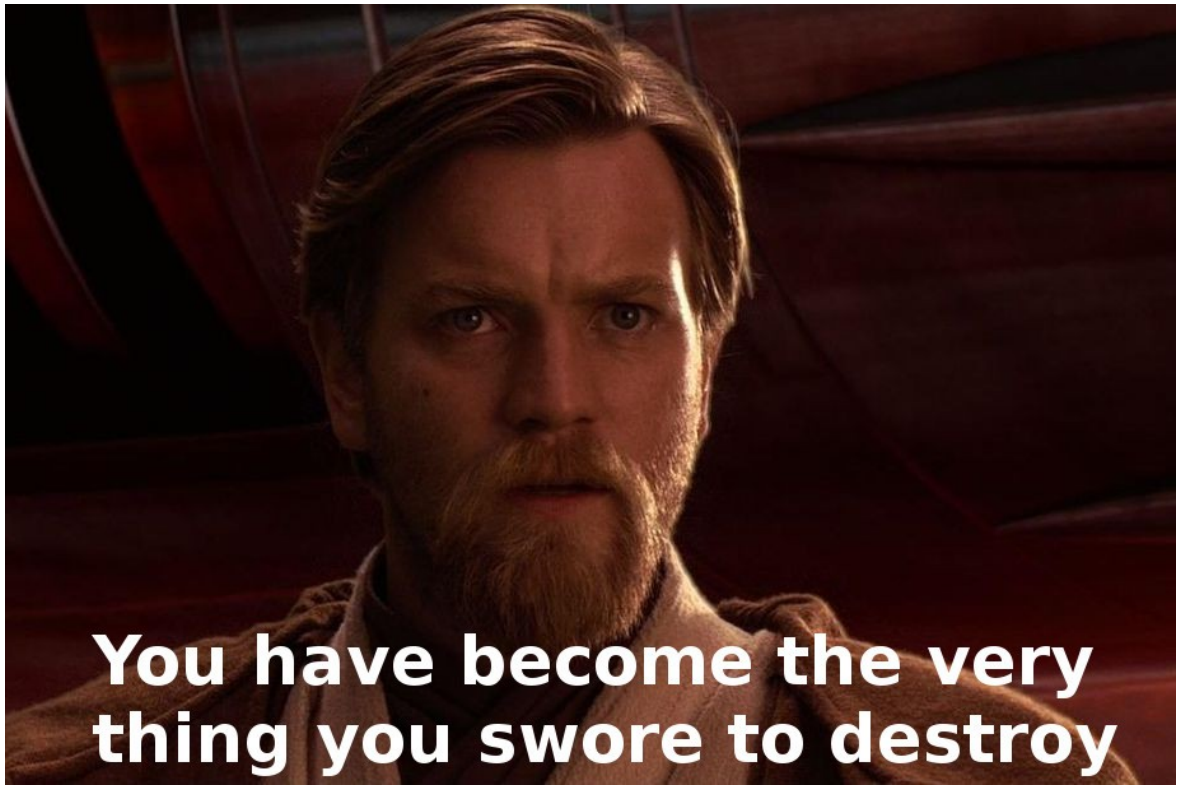
But...

- Are they secure enough?
- Can they protect us from an attack?
- How could somebody attack isolation?

Attacks and Vulnerabilities

Phase 2

- Detect docker vulnerabilities
- Ethical hacking: implement container breakout attacks
- Reverse engineering to find rules preventing them



National Technical University of Athens
CSLab

Attacks and Vulnerabilities

1. Running as root

If no User Namespaces are enabled in docker:
Host root == Container root

If running as root on your server is not a best practice, for the same reasons you shouldn't run anything as root in a container on your server.

To root or not to root?
➔ Don't use root.
➔ If you need it, try using a user that looks like root, using user namespaces or kernel capabilities.
➔ If this is not enough, use root but make sure you have extra walls of security.

Attacks and Vulnerabilities

2. Kernel Capabilities

Adding a lot of capabilities, could lead to privilege escalation and eventually root.

Some capabilities are more crucial than others: SYS_ADMIN, SYS_CHROOT, SETUID etc

privileged_mode: Gives all capabilities to the container and also lifts all the limitations enforced by the device cgroup controller.

➔AppArmor profiles cannot block actions of a privileged container
➔SecureWilly alerts user about any containers that use privileged mode

Attacks and Vulnerabilities

3. Disabling Namespaces

Namespaces are a feaure that partitions kernel resources and provide resource isolation.

In Docker, all namespaces are enabled by default, except for user ns.

There are some flags that allow a container to enter host's namespaces: --pid=host, --net=host, –uts=host, –ipc=host, –userns=host (if user namespaces are enabled).

What about mount namespace?
Volumes: Sharing host's filesystem by mounting directories in runtime.

➔Combining several of these flags could lead to container breakout attacks. Red hat talks about super privileged containers when mount and one of the other flags are used
➔SecureWilly alerts user about any containers that enters a host namespace

Attacks and Vulnerabilities

4. Nsenter tool

Nsenter breaks into the namespaces of other processes, as soon as it has their pids.
It does not drop capabilities, so a new shell inside a running process can harm it in great extent, given the right capabilities.

A container that uses nsenter can commit:

- Breakout to host
- Breakout to another running container (needs access to docker daemon to find the container id of the target container → get pid of container's process)

➔AppArmor profiles protect a container from nsenter attacks committed by another container

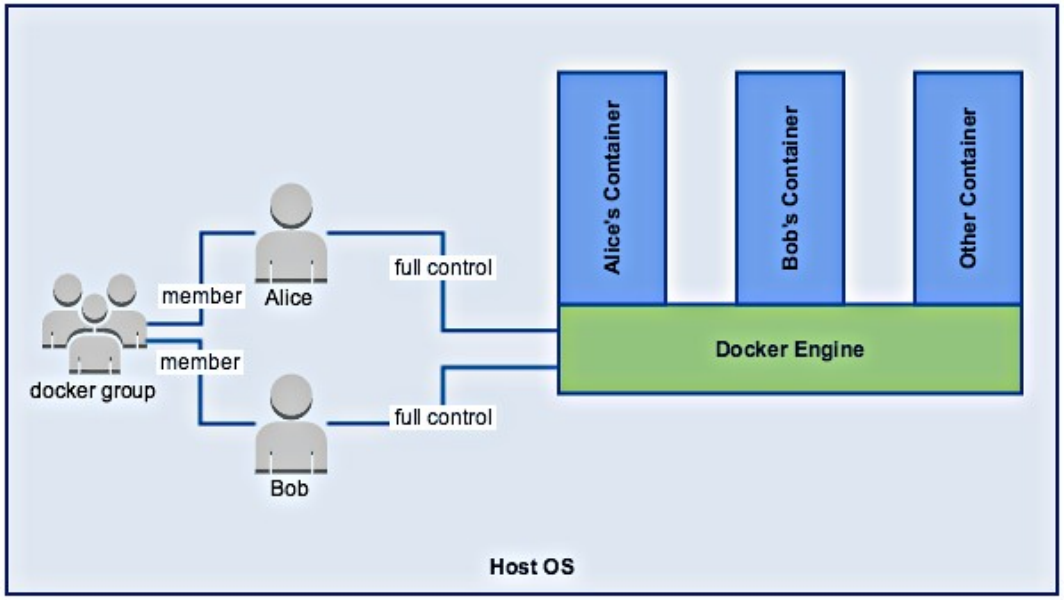
Demo

Attacks and Vulnerabilities

5. Access to Docker Daemon

Who can access the docker daemon?

- Members of docker group → owner group of docker.sock
- User privilege elevation: Non root member of docker group acts as root on host
- Container privilege elevation: A "regular" container could evolve to a "powerful" one. (needs 1 member of docker group + mount docker.sock)



- **Alice** runs **non root and no member of docker group** user on container, **mounts docker.sock**
- **Bob** (or a host user) member of docker group, **enters Alice's container as root** and **adds her user to docker group**.
- **Alice re-logins** and is **member of docker group**.

➔SecureWilly drops setgid and setuid capabilities if docker.sock is mounted, to block this attack.

Demo

National Technical University of Athens CSLab

Experimental Evaluation

Experimental Setup

We used the following docker projects to create AppArmor profiles and evaluate SecureWilly:

- Benchmarks from a cloud benchmark suite, CloudSuite.
  i. Media Streaming benchmark (services: server, client, dataset)
  ii. Data Caching benchmark (services: server, client)

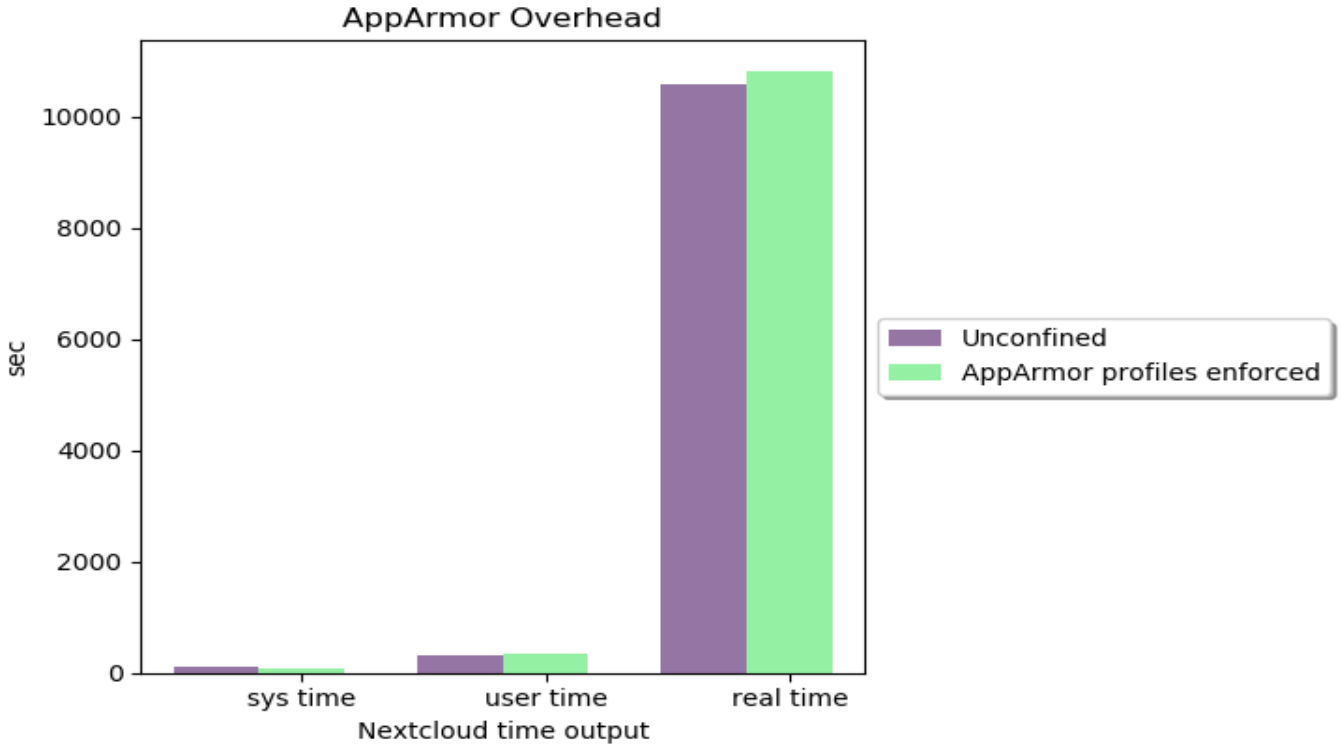
- Nextcloud: a suite of client-server software for creating and using file hosting services to have total control of private data.
  Used a Nextcloud docker instance consisting of a Nextcloud server service and a database service mariadb.

Experimental Evaluation

AppArmor related

1. AppArmor overhead

- Small overhead
- Almost not noticeab
- Confirms AppArmor intention



2. SecureWilly vs Genprof (official AppArmor generating profile tool):

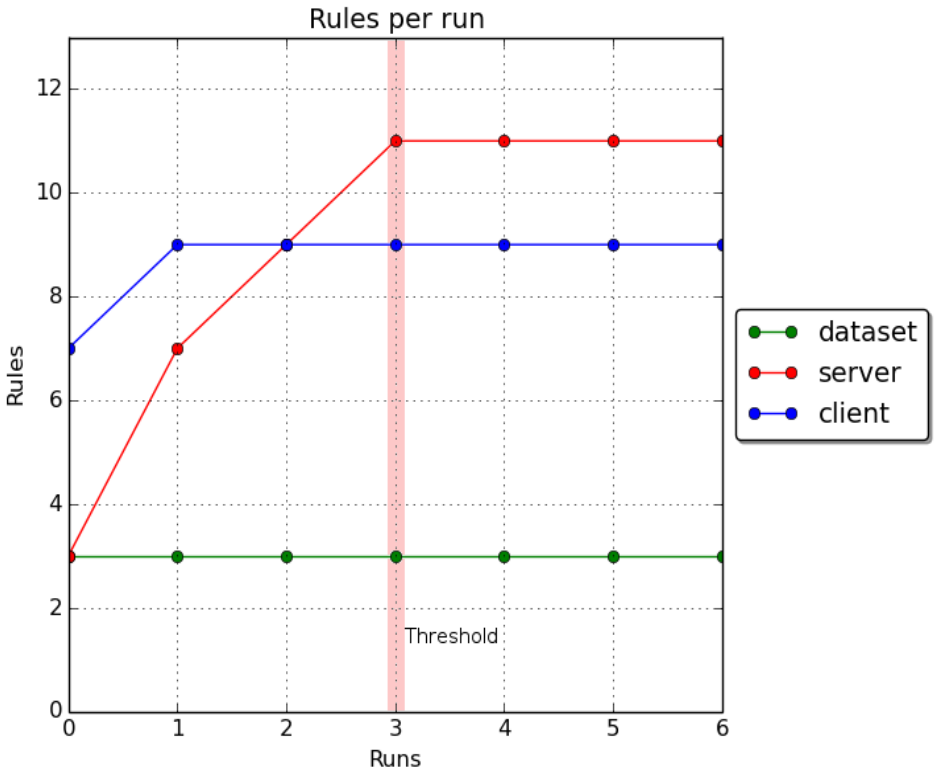
- Genprof is designed for host's applications → missing file rule, which is needed for a container to access its filesystem on host
- Host oriented behaviour – file and path accesses on host
- Include profiles → Violates the principle of least privilege, may include redundant rules

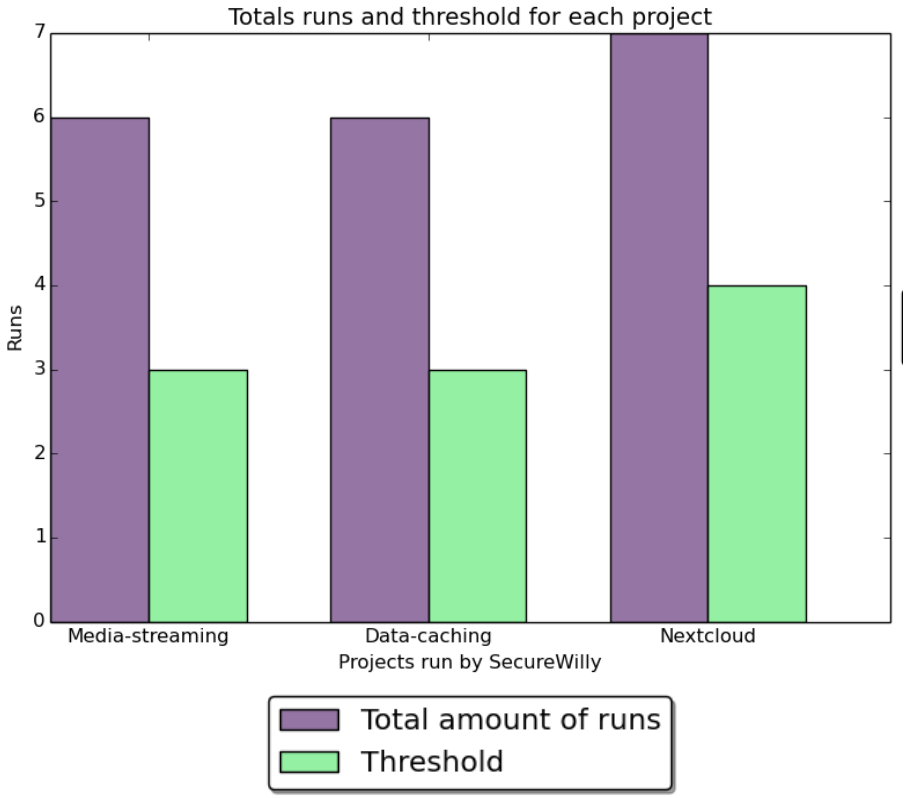
Experimental Evaluation

Performance

$$T(n) = \mathcal{O}(1) + m * f(n) + c \Rightarrow T(n) = m * f(n) + d \Rightarrow T(n) = m * f(n) \Rightarrow T(n) = f(n)$$

Comparing time is useless → Compare amount of runs of test plan.



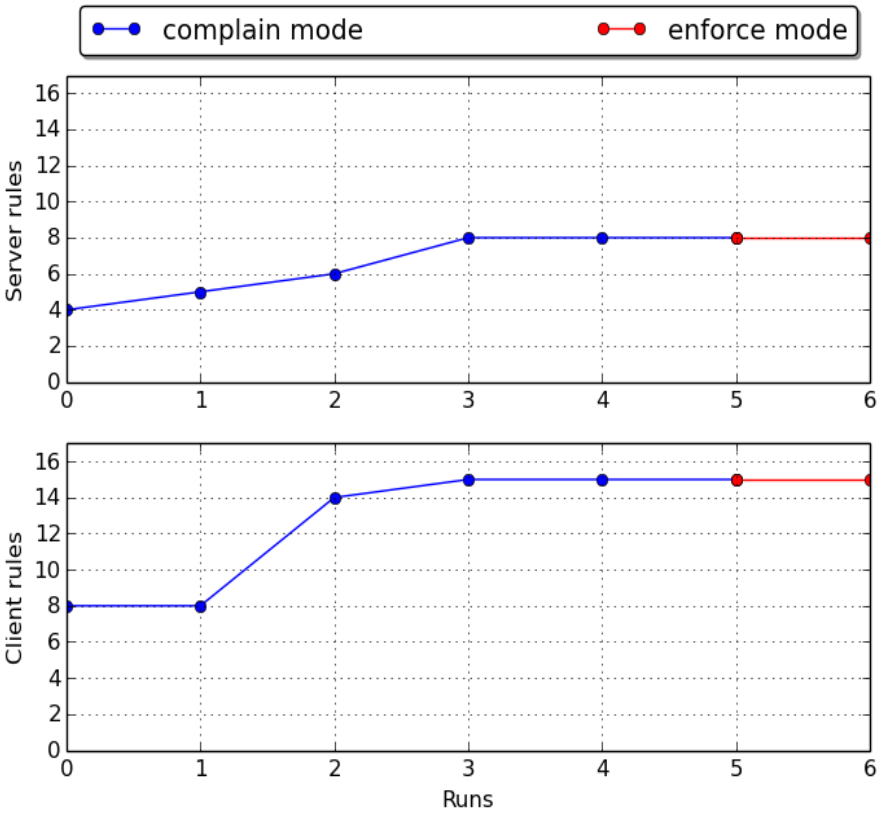


Media streaming benchmark

Experimental Evaluation

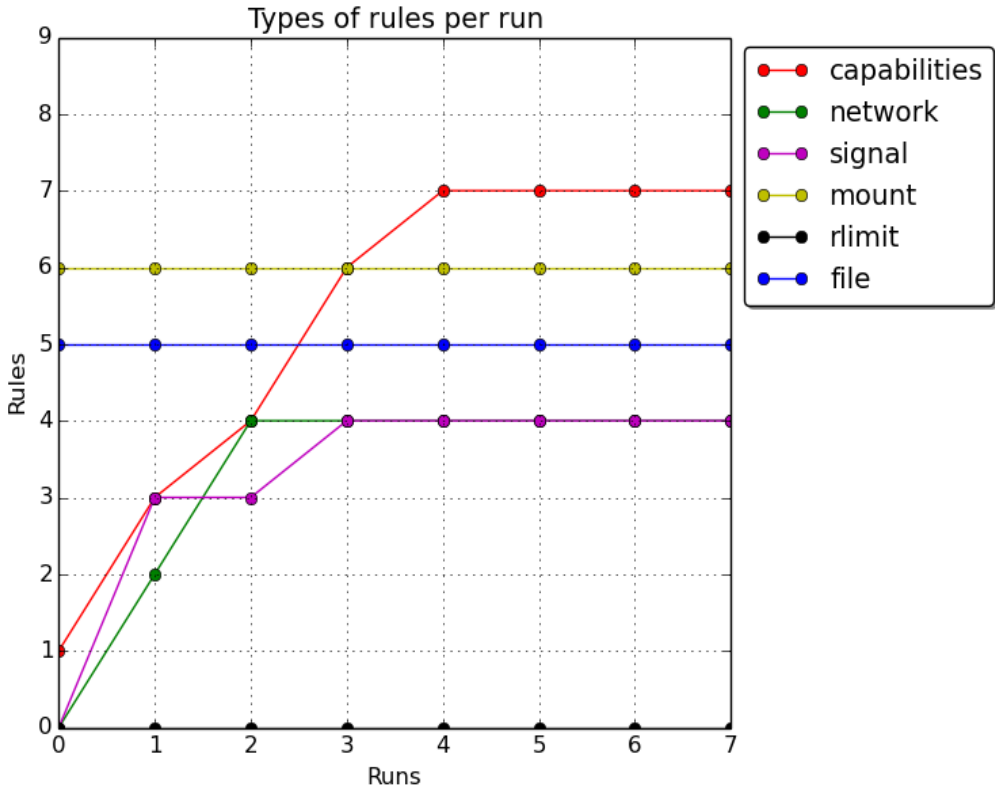
Functionality

1st Functionality testing:
Enforce mode inside Dynamic Analysis



Data caching benchmark

2nd Functionality testing:
Confirming service's role through profile rules

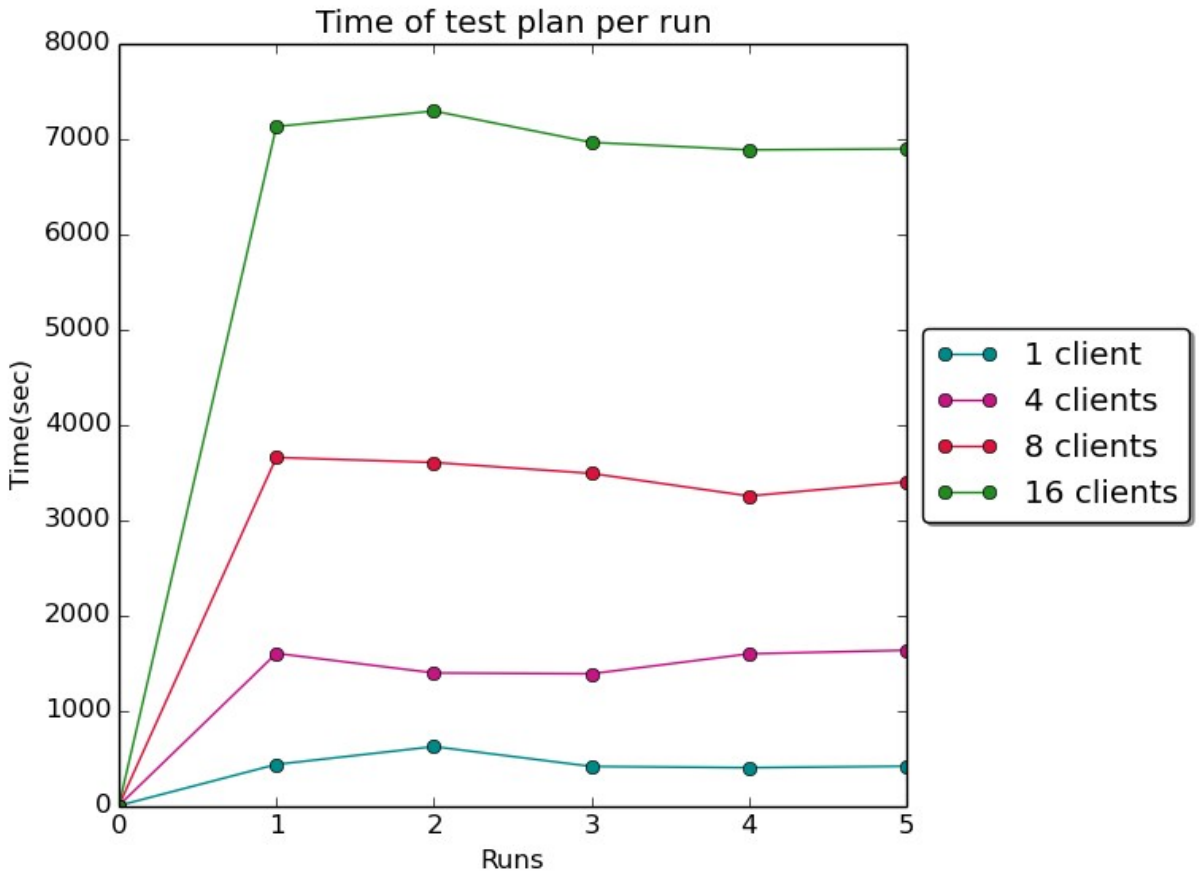


Nextcloud - server

Experimental Evaluation

Scalability

If the given docker project is scalable, so is SecureWilly.

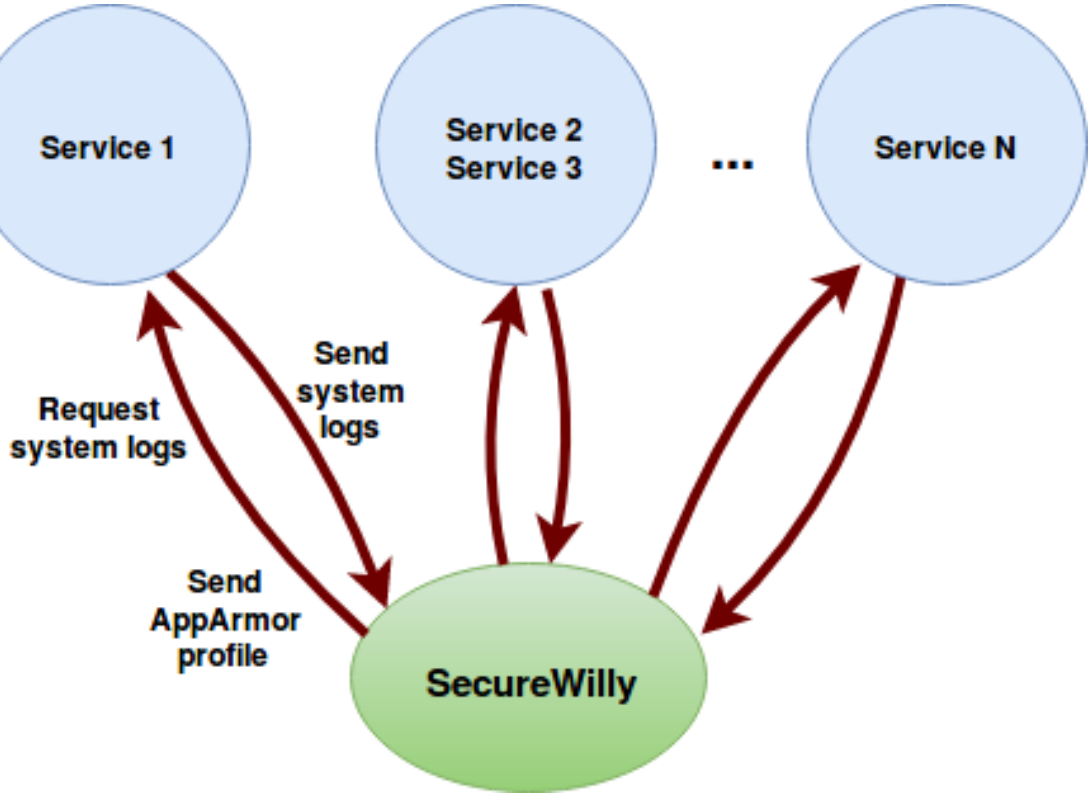


Media streaming benchmark

Experimental Evaluation

Distributed Systems

- Plans for handling distributed services
- The way SecureWilly approaches a docker project makes the development of this option possible



Future Work

Features to fix and integrate:

- Extract more rules in static analysis
- Include other syntax forms of Dockerfile instructions and Docker Compose options to static parser.py
- More rules to prevent attacks
- Alerts about root user in containers
- More flexible User Interface
- Support interactive test plans of the project, not only script commands
- Option of user manually adding rules, through a configuration file
- Change python scripts into executables (maybe write programs in Go) so that python interpreter is not a requirement
- Fix the conflict of multiple containers using the same image, when a docker-compose file is not provided
- Fix clearing volumes by detecting docker project's volumes and deleting them, instead of using docker volume prune
- CI/CD to build releases on GitHub, through Travis CI, which supports open source projects and Docker
- Support distributed services

Future Work

Spread the word...

- GSoC accepted to work on NextCloudPi project (expand features on a ready to use Nextcloud image)
- One of the tasks I proposed is to use SecureWilly to produce AppArmor profiles for all docker containers used in the project.

Questions

Thank you :)