Do Containers fully 'contain' security issues?

A closer look at Docker and Garden.

Farshad Abasi / Mirai Security Inc.

FWDSEC»

Farshad Abasi

- Based in: Vancouver, BC, Canada
- Co-founder: Mirai Security
- CTO/CISO: Machool Technologies
- Instructor: British Columbia Institute of Technology
- News correspondent: CFAX AM1070 (Victoria)
- Board member: BSides Vancouver / MARS
- Avid music fan!

Overview

- Computing in “The Cloud”
- What are Containers?
- The Supporting Cast
- Containerization vs. Virtualization
- Behind The Scenes: Docker
- Behind The Scenes: Garden
- Key Security Concerns
- Known Vulnerabilities
- Safe Container Practice
- Conclusion
- Q&A

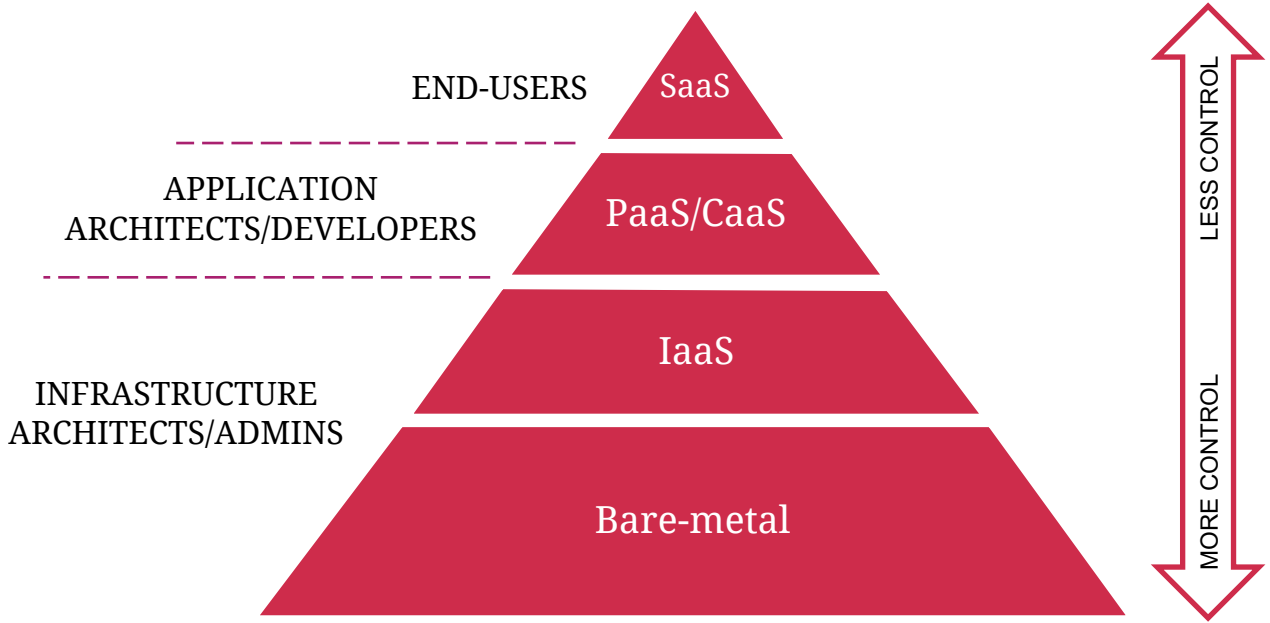
FWDSEC


Computing in “The Cloud”

“a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

- SP 800-145, The NIST Definition of Cloud Computing

- Cloud computing requires a workload isolation mechanism
  - Physical separation (aka Bare-metal Cloud)
    - Addresses covert channels & defects in H/W protection mechanisms
  - Virtualization (aka Hardware Virtualization)
    - Used to provide IaaS
    - Vulnerabilities related to CPU or hypervisor can have impact
  - Containerization (aka OS-level virtualization)
    - Used to deliver CaaS or PaaS
  - Multi-user accounts within a single application
    - In other words: SaaS


END-USERS
SaaS
APPLICATION
ARCHITECTS/DEVELOPERS
PaaS/CaaS
IaaS
INFRASTRUCTURE
ARCHITECTS/ADMINS
Bare-metal
LESS CONTROL
MORE CONTROL

FWDSEC

What are Containers?

FWDSEC

- Containerization (aka OS-Level Virtualization)
  - Multiple isolated user-space instances sharing a kernel
  - Each instance = a container
    - aka partition, virtualization engine (VE) or jail (e.g. FreeBSD jail or chroot jail)
    - seems like a regular machine from the inside
- Encapsulate applications + act as interface to the surrounding system
- Kernel provided resource-management
  - limits impact of a container's activities on others

- Chroot
  - Has been around since Version 7 Unix (‘79)
  - Isolates filesystem views for process + its children
  - Does not isolate other resources (e.g. networking, processes, etc.)
  - Root account has full view/access
- Modern containers use:
  - Namespaces to separate resources
    - PID (process ID), Mount, Network, UTS (hostname + NIS domain name), IPC, and User namespaces (arguably most important, not always implemented)
  - Cgroups to meter and limit resources, control access to device node (/dev/*), and perform crowd control

FWDSEC

More on namespaces

- **User**
  - Namespace for UIDs and GIDs
  - A given user ID (e.g. 0 or root) inside the container will map to a different user ID on the host
  - Fairly new and there have been known vulnerabilities at the start
  - System calls and security logic should be namespace aware and check the capability in the correct namespace
    - incorrect checking may lead to “CLONE_NEWUSER|CLONE_FS” root exploit type vulnerabilities
  - Exploits are still possible even with user namespaces enabled
- **Network**
  - provides a separated net stack for each container
  - Widely used (“NET_RAW abuse” explores typical flaws in this namespace)

More on namespaces

- **PID**
  - Process ID namespace, allowing each container to have a fully isolated process tree (with an ‘init’ process the runs as PID 1 in its namespace)
  - The PID will be different inside the container than on the host (vulnerabilities have shown that this info can leak)
- **Mount**
  - Used to separate the filesystem for each container (pivot_root sys call is used in conjunction)
- **IPC**
  - Deals with SystemV IPC and POSIX message queues
- **UTS**
  - System identifiers, used to provide container specific hostnames

Linux Security features used to secure containers

- Cgroups (aka control groups)
  - Limit, accouns for, and isolate the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes
  - Can also be used along with iptables for traffic shaping
- Capabilities
  - Allow the breakdown of root role into pieces that can be granted to non-privileged processes to perform privileged actions
  - The set assigned to a process carries forward to child processes

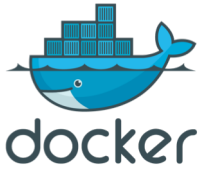
Linux Security features used to secure containers (cont'd)

- MAC (Mandatory Access Control)
  - Limit the actions of a program
  - Hooks provided by Linux Security Modules (LSMs) such as AppArmor
  - Both Docker and LXC both have this enabled
  - SELinux is also an option, but not as well supported and documented
- Seccomp: used to filter system calls
  - "strict" mode only allows a small set of system calls (can't be modified)
  - In "filter mode" filters are written using BPF (Berkeley Packet Filter) programs allowing more finely-grained policies
  - Docker supports seccomp-bpf
    - ptrace vulnerability bypasses seccomp

Container uses:

- Allocation of finite hardware resources (same as H/W virtualization)
- Improved hardware independence and added resource management
  - Can package an app and its dependencies in a container
- Hosting API/microservices
- Automates deployment of applications
  - Distribution method for software to guarantee reproducibility
  - DevOps tools for testing and deploying code
- Building blocks for a PaaS
- Common in virtual hosting
- Security??

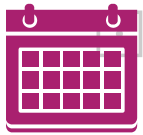
- Modern implementations make containers easier to use
  - Docker Engine, Cloud Foundry Garden
- Some are designed to run multiple processes and services, some only run a single service
  - Operating system containers vs. application containers



FWDSEC

The Supporting Cast

FWDSEC



- Container schedulers
  - Allow scale-out, load balancing, adding storage or network resources, maintaining HA and recoverability
  - E.g. Docker Swarm, Kubernetes, CF Diego

- Container packaging and staging
  - E.g. OpenShift S2I, CF buildpacks
- IaaS orchestration
  - E.g. Red Hat Ansible, CF BOSH
- Platform as a Service (PaaS)
  - E.g. OpenShift, Cloud Foundry
- Tools evolved at the same time independently

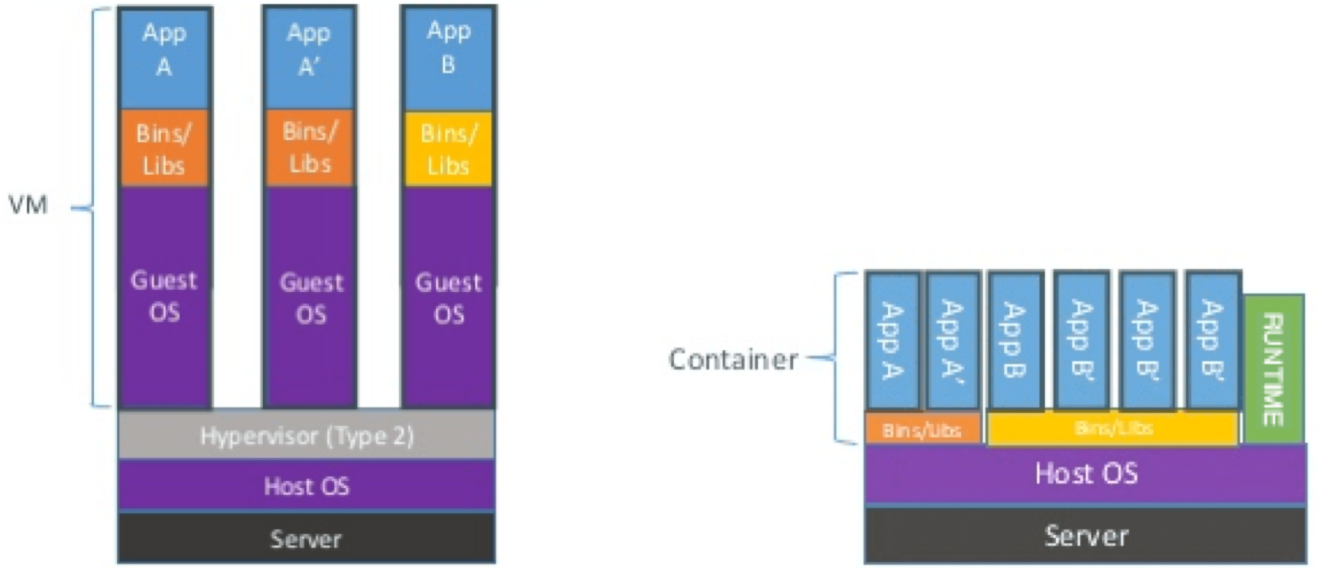
FWDSEC

Containerization vs. Virtualization

FWDSEC

- Containerization
  - Containers share the same OS kernel
    - Cannot use containers with different operating systems
  - Faster, lightweight, more portable, scale more efficiently
    - No H/W emulation
  - Do not provide the same level of isolation as virtualization
- Virtualization
  - Mature with an extensive ecosystem
  - Allows for mixed kernels on the same platform
  - Host emulates the hardware provided to the VM
    - looks like it is running on separate hardware
  - Hypervisor is the security boundary: More secure

Containerization vs. Virtualization



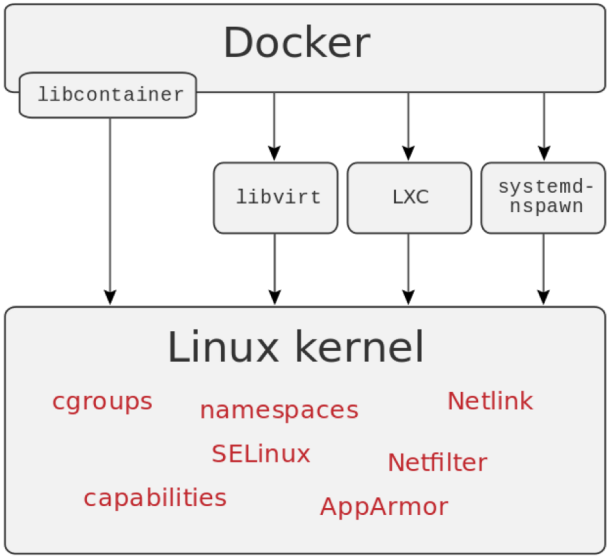
Containers share OS kernel and (possibly) binaries and libraries

FWDSEC

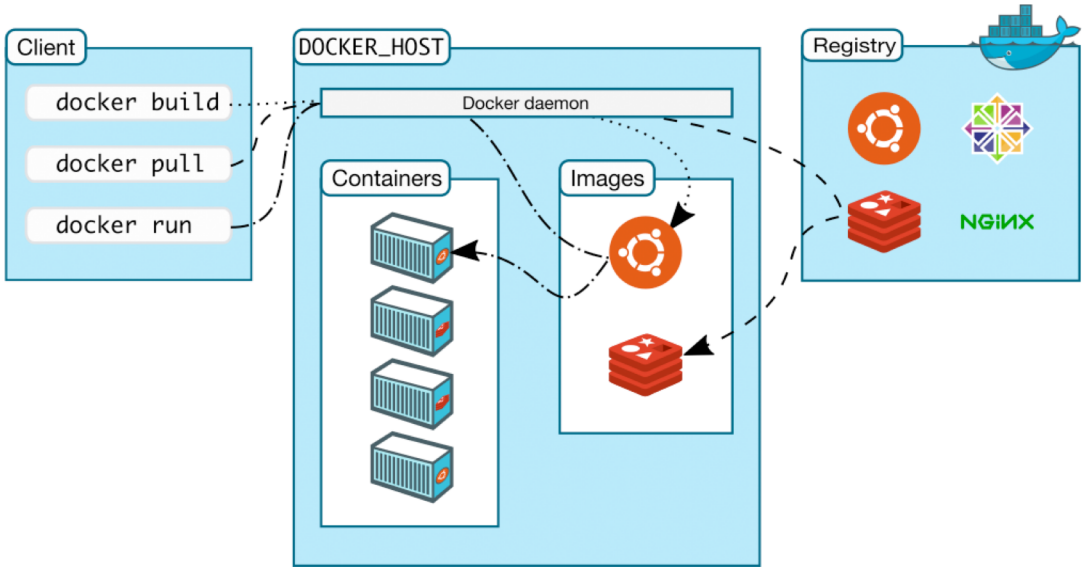
Behind the Scenes: Docker

FWDSEC

- Accesses the Linux Kernel virtualization in many ways
  - Pre-version 0.9, LXC was the default execution environment
  - Post-version 0.9, runC (aka libcontainer) written in Go, Open-source, allows for direct use of Linux virtualization facilities
  - Supports abstracted virtualization interfaces
- Actions done to Docker base images
  - UnionFS layers are created
    - allow for recreation
- Docker daemon runs as root!

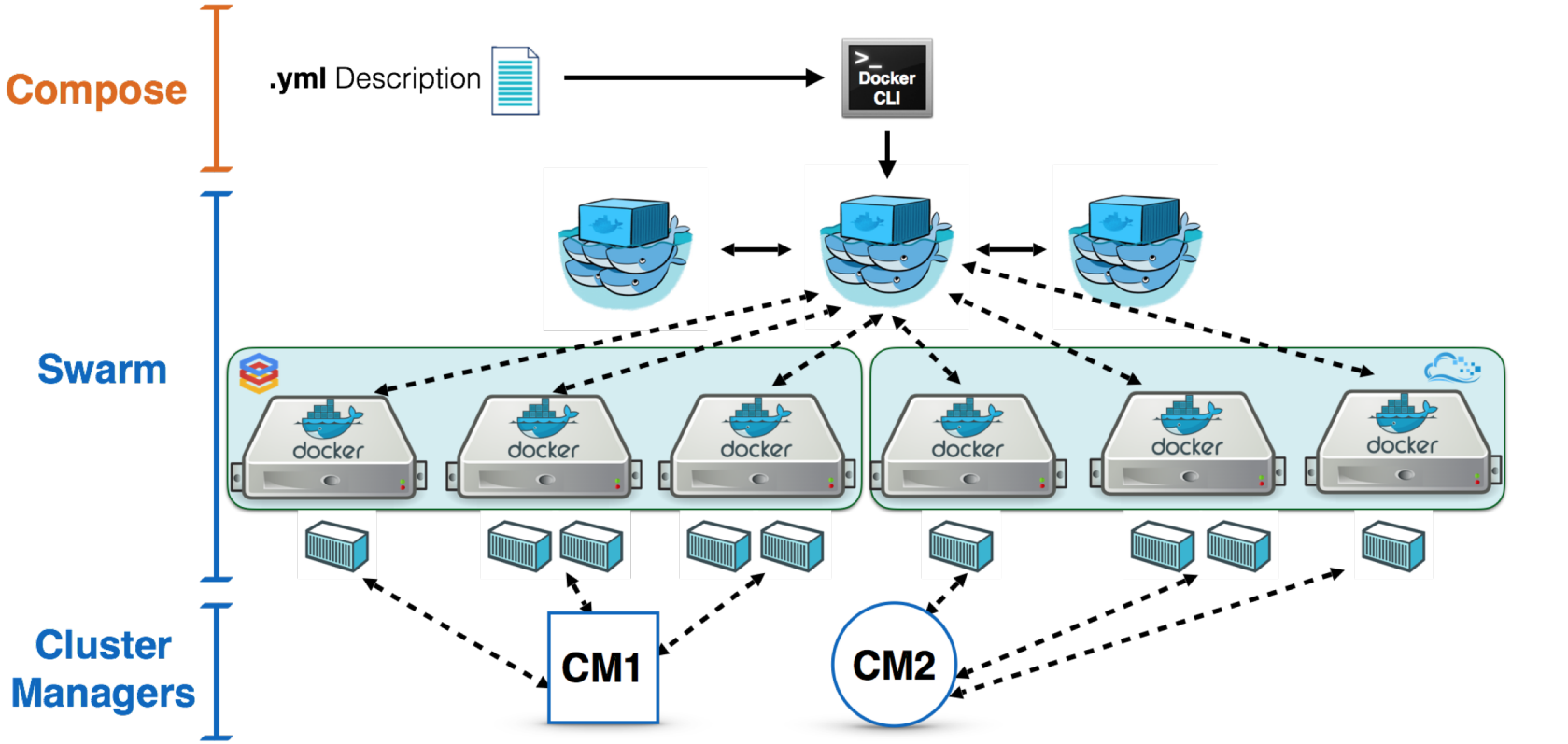


FWDSEC



Docker Engine Architecture

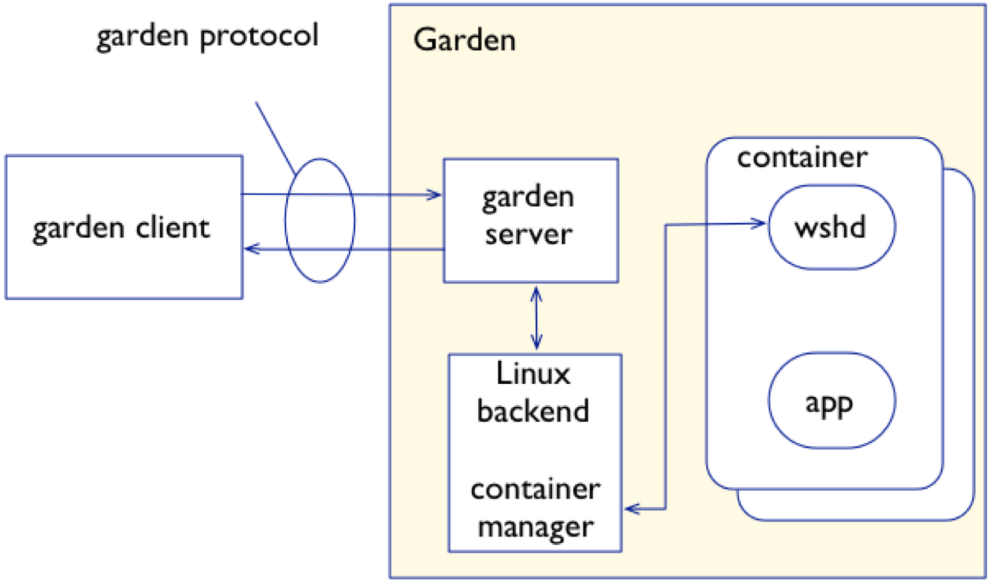
Behind The Scenes: Docker



FWDSEC

Behind the Scenes:
Garden

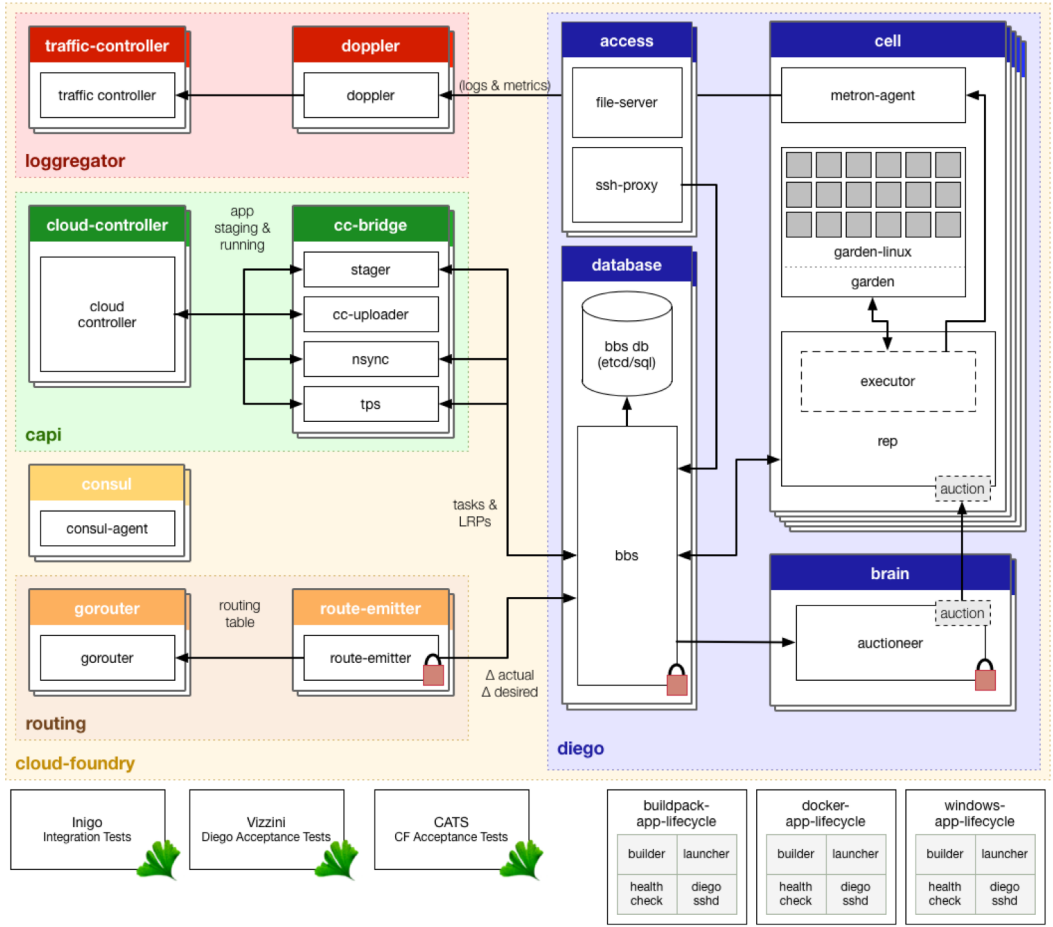
FWDSEC



Garden Architecture

- Garden = Warden re-written in Go
  - Pluggable back-ends for different platforms
- Garden-runC back-end
  - Linux-specific implementation of the Garden interface using Open Container Interface (OCI) standard
  - AppArmor is used for all unprivileged containers
  - Seccomp whitelisting restricts access to set of system calls
- ‘wshd’ (Warden Shell Daemon)
  - Root process for managing containers and launching apps
  - Streams standard output and error back to client

Behind The Scenes: Garden



FWDSEC

Behind The Scenes: Garden vs. Docker

Feature	Garden	Docker
Resource isolation and control	CPU shares memory + swap network bandwidth disk size quota	CPU shares CPU sets memory memory swap block device bandwidth
Dynamic resource management	Feature supported but not used	Not supported.
Image management	Only whole images can be reused to create new containers.	Layered—allows for reusing separate layers.
Linking containers	no	yes
Exposing ports	Multiple ports per container	Multiple ports per container

FWDSEC

Key Security Concerns

FWDSEC

- User namespace support
  - Supported in Garden and Docker
    - Not enabled by default in Docker
  - Without it, processes running as root inside a container can have root access everywhere
  - Has to be configured properly and not break programs
- Namespaces do not cover everything in Linux
- Imperfect design and code
  - Increases attack surface
  - Namespace leaks present in many containers
  - Many containers have had little security analysis
- Docker's images verification is not 100%

FWDSEC

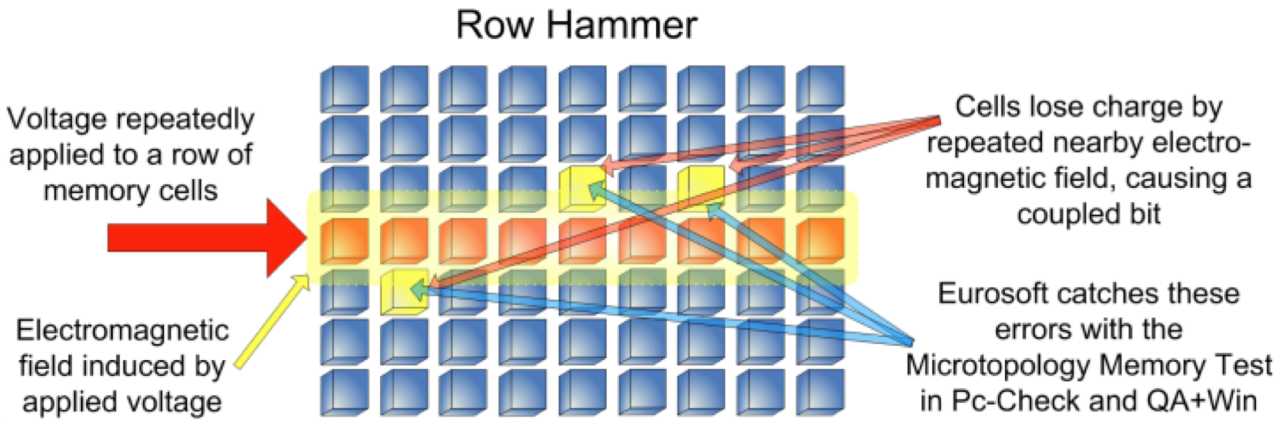
- Shared Kernel = any vulnerability in Kernel can impact all containers
  - OS Kernels provide much more functionality than Hypervisors: larger attack surface
  - Kernels will always have vulnerabilities and containers directly expose it to programs
- Containers were not designed to ‘contain’ security issues

Covert channels

- Provide capability to transfer information between processes that are not allowed to communicate
- Storage channels
  - Communicate by modifying a "storage location"
- Timing channels
  - Perform operations affecting response time observed by the receiver
- Difficult to completely prevent on the same processor
- Techniques for locating potential covert channels:
  - Analyzing the resources of a system
  - Source-code level analysis
- Possibility can be reduced by careful design and analysis

Defect in hardware protection mechanisms

- Computer hardware is complex
- DRAM Rowhammer bug used to gain kernel privilege
  - Enables change to values in other programs/kernel
- Exploit demonstrated by Google on a variety of systems



FWDSEC

Known Vulnerabilities

FWDSEC

Docker: 14 vulnerabilities identified so far

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	Other
2014	6		2							1		1			2
2015	3										1	1			1
2016	2									1		1			
2017	3	1													2
Total	**14**	1	2							2	1	3			5
% Of All		7.1	14.3	0	0	0	0	0	0	14.3	7.1	21.4	0	0	35.7

PCF/Garden: 11 vulnerabilities identified so far

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	Other
2016	4											2	1		1
2017	7	1	1			1	1				1				2
Total	11	1	1			1	1				1	2	1		3
% Of All		9.1	9.1	0	0	9.1	9.1	0	0	0	9.1	18.2	9.1	0	27.3

* Source: www.cvedetails.com

FWDSEC

“Containing” Potential Harm

FWDSEC»

- Group containers on a given VM based classic segregation principles
  - Use a risk based approach, consider impact and likelihood
- Services should be run as unprivileged
- Privilege should be dropped as soon as not needed
- Treat root inside a container as if it is root outside the container
- Only run containers from trusted parties
- Follow a layered defense approach
  - Use AppArmor or SELinux

- Standardize and verify hardened host OS
- Scan containers for vulnerabilities
  - OSS modules, licensing, malware, correct configuration
- Measure containers and sign
  - Analyze, sandbox, build profile
  - Confirm signatures at boot: Confirm host OS integrity
- Monitor and detect anomalous behavior
  - Alert, log, or prevent
- Analyze usage logs to:
  - Identify weaknesses, adapt patterns
  - Share learning across different instances

FWDSEC

Conclusion

FWDSEC

- Containers are very useful
  - help ease software management
- Work is being done to address security
  - e.g. implementation of user namespaces
- Should be used with caution
  - processes in the container should not be given privileged access
  - good for deploying apps that are trusted (e.g. same vendor)
  - other mechanisms such as SELinux, seccomp, AppArmor, and separate user accounts should be used in conjunction

Q&A

Contact: Farshad Abasi
f.abasi@fwdsec.com

FWDSEC