An In-Depth Look at Application Containers
Overview

• A Brief History and Overview of Containers
• Security Benefits(?) of Containers
• Container Vulnerability Management
• Responding to Container Attacks
Containers are not new, but...

Containers have been around for 30 years.
Similarly, computers have been around since 1882 when Babbage built it.
Container Adoption Challenges

- Networking is the second biggest factor when it comes to container adoption, according to the survey, but there is a wide range of other reasons.

- Although Kubernetes is one of the largest areas of container adoption, according to the survey, it is behind Docker, the largest container company on the globe.

- Although containers are being used across a broad range of application use cases, the survey shows a question of use cases and not just run in containers as a container.

- The top container challenges include:
  - Container adoption
  - Security
  - Application architecture evolution
  - Container threats
  - Application architecture evolution

- Biggest challenges of using containers:
  - Integrating them into existing IT environments: 55%
  - Relating them to existing environments: 55%
  - Adopting new and open standards: 47%
  - Having enough experienced staff to handle them: 24%

- Container adoption and usage:
  - 34% are in use today
  - 22% are planning to adopt
  - 23% are not planning to adopt
  - 21% are already using them

- Challenges with container adoption:
  - Benefits of containers
  - Container threats
  - Application architecture evolution

- Enterprise adoption of DevOps:
  - Not adopting DevOps: 17%
  - Adopting DevOps for pilots or lessons: 10%
  - Adopting DevOps for the whole company: 33%
Container Adoption Challenges

- Lack of experience using containers: 25% using, 39% not using
- Security: 29% using, 27% not using
- Monitoring & managing containers: 22% using, 26% not using
- Governance: 19% using, 25% not using
- Technology not mature: 14% using, 29% not using
- Resistance from ops: 14% using, 18% not using
- Resistance from dev: 11% using, 14% not using
Container (n):

- Software-based isolation, for processes controlling...
  - Process grouping
  - Resource usage
  - What actions a process can take
cgroups

/sys/fs/cgroup/

- blkio
  - blkio.io_merged
  - blkio.io_queue
  - blkio.io_service_bytes
  - blkio.io_service
  - blkio.io_service_time
  - blkio.io_scheduled
  - blkio.reset_stats
  - blkio.sectors
  - blkio.throttle.io_service_bytes
  - blkio.throttle.io_service
  - blkio.throttle.read_bps_device
  - blkio.throttle.read_iops_device
  - blkio.throttle.write_bps_device
  - blkio.throttle.write_iops_device
  - blkio.time
  - blkio.weight
  - blkio.weight_device
  - cgroup.clone_children
  - cgroup.event_control
  - cgroup.procs
  - lxc
    - notify_on_release
    - release_agent
    - tasks
- cpu
  - cgroup.clone_children
  - cgroup.event_control
  - cgroup.procs
  - cpuset
    - cpuset.cpu_exclusive
    - cpuset.cpus
    - cpuset.mm_exclusive
    - cpuset.mm_hardwall
    - cpuset.memory_migrate
    - cpuset.memory_pressure
    - cpuset.memory_pressure_enabled
    - cpuset.memory_swap_page
    - cpuset.memory_swap_slab
    - cpuset.mems
    - cpuset.sched_load_balance
    - cpuset.sched_relax_domain_level
    - lxc
      - notify_on_release
      - release_agent
      - tasks
- devices
  - cgroup.clone_children
  - cgroup.event_control
  - cgroup.procs
  - devices.allow
  - devices.deny
  - devices.list
  - lxc
    - notify_on_release
    - release_agent
    - tasks
- freezer
  - cgroup.clone_children
  - cgroup.event_control
  - cgroup.procs
  - lxc
    - notify_on_release
    - release_agent
    - tasks
- hugetlb
  - cgroup.clone_children
  - cgroup.event_control
  - cgroup.procs
  - hugetlb.2M.failcnt
  - hugetlb.2M.limit_in_bytes
  - hugetlb.2M.max_usage_in_bytes
  - hugetlb.2M.usage_in_bytes
  - lxc
    - notify_on_release
    - release_agent
    - tasks
- memory
  - cgroup.clone_children
  - cgroup.event_control
  - cgroup.procs
  - lxc
    - memory.failcnt
    - memory.force_empty
    - memory.limit_in_bytes
    - memory.max_usage_in_bytes
    - memory.mems.failcnt
    - memory.mems.limit_in_bytes
    - memory.mems.max_usage_in_bytes
    - memory.mems.usage_in_bytes
    - memory.move_charge_at_immigrate
    - memory.numa_stat
    - memory.oom_control
    - memory.soft_limit_in_bytes
    - memory.stat
    - memory.swappiness
    - memory.usage_in_bytes
    - memory.use_hierarchy
    - notify_on_release
    - release_agent
    - tasks
- perf_event
  - cgroup.clone_children
  - cgroup.event_control
  - cgroup.procs
  - lxc
    - notify_on_release
    - release_agent
    - tasks
Grouping and constraints
What does a process do?

• Auditing (read write audit events)
• File access (read, write, create, delete)
• Network access (bind, send, receive)
• Process management (fork, kill)
• Security (MAC)
• Debug (tracing)
• Administration (system config)
Capabilities

- CAP_AUDIT_CONTROL
- CAP_AUDIT_READ
- CAP_AUDIT_WRITE
- CAP_BLOCK_SUSPEND
- CAP_CHOWN
- CAP_DAC_OVERRIDE
- CAP_DAC_READ_SEARCH
- CAP_FOWNER
- CAP_FSETID
- CAP_IPC_LOCK
- CAP_IPC_OWNER
- CAP_KILL

- CAP_LEASE
- CAP_LINUX_IMMUTABLE
- CAP_MAC_ADMIN
- CAP_MAC_OVERRIDE
- CAP_MKNOD
- CAP_NET_ADMIN
- CAP_NET_BIND_SERVICE
- CAP_NET_BROADCAST
- CAP_NET_RAW
- CAP_SETGID
- CAP_SETFCAP
- CAP_SETPCAP
Capabilities

Worst to best:

- Run with --privileged=true
- Run with --cap-add ALL
- Run with --cap-drop ALL --cap-add <only needed>
- Run as non-root user, unprivileged

Useful: capabilities section of documentation container runtimes
Security Benefits of Containers and Microservices

• Smaller surface area*
• Shorter lifespan* – shorter period when open to attack
• More automated process – easier to recreate/redeploy*
Security Benefits of Containers and Microservices

• Containerized apps lend themselves to ”12 factor” design
Security *Disadvantages* of Containers and Microservices

- Relatively new technology
- Lots of moving parts
- Shorter lifespan – this makes investigations more difficult
Results of Twitter Survey

John Kinsella @johnlkinsella · Oct 12
#docker #container #security poll: Working on my #LFWWebinar talk, curious what folks see as their biggest container security issues? Others?

- 29% Image security (signing)
- 31% Host security
- 23% Vulnerability management
- 17% Container isolation

84 votes • Final results
Great White Breaches Cage

SHARK #GREATWHITE #GUADALUPEISLAND

Cory Scott @cory_scott · Oct 14

"Don't worry, service isolation and sandboxing will address the risks."
Why Isolate?

• Only as secure as your weakest link
• What happens if other departments are running in your private cloud?
• What happens if other customers are running in your bare metal CaaS?
Unikernels – Hardware Isolated Containers

• Build custom VM images to boot bare OS to support single (usually) container
Brief Overview of Container Orchestration
Why Orchestration?

• For “real” workloads:
  • How to launch 500 containers across 20 hosts?
    • Being aware of resources on each host
    • Getting storage and networking to right container on the right host
    • Distribution for speed, efficiency, cost, etc.
    • As part of a CI/CD process
  • How to do a rolling update of those 500 live containers to a new sw version?
Lots to Orchestrate

- Customer VM
  - VM Image Management
  - Networking
- Customer VM
  - Local Storage
  - NAS/SAN
Lots to Orchestrate

Customer VM
- VM Image Management
- Networking
- Local Storage
- NAS/SAN

Customer VM

Containers
- Container Image mgmt
- Container networking
- Container storage

Host
- Host Image Mgmt
- Host Networking
- Local Storage
- NAS/SAN
Image Security

• Where did an image come from?
• Is it an official image?
• Is it the right version?
• Has somebody modified it?
Host Security

• Follow standard hardening processes but only firewall host, not it’s containers

• A host itself shouldn’t be “exposed” – there should be no public attack surface. Administer via known private network

• One nasty exposure – privileged containers.
Vulnerability Management in a Container World
Managing Security Exposure in Containers
Smaller Image, Less Vulnerabilities

• Avoid "From: $bloatedDistribution" and similar

• Software can’t be vulnerable if it’s not installed.

An amazingly large percentage of public container images are based on bloated (500mb+) full distributions.
Why? Least Privilege

• We want the smallest image possible, when we load it across 100 hosts
• The smaller the image, the less exposure for potential vulnerabilities
• If the parent image has a vulnerability, everybody based on that parent has to re-spin their image
Seccomp

We need to build a list of system calls called by the program...

...that we want to succeed

- Guess (preferably educated)
- RTFM (thanks John!)
- Capture behavior – maybe /usr/sbin/strace
- Disassembly?
Plan For Container Attacks

• Before going to production, think about how you’d investigate an attack

• Containers are mostly ephemeral
• Collect logs at a central location
• Practice identifying and snapshotting problem containers
• Don’t forget about data backup/recovery
Thanks – Let’s continue the conversation!

@johnlkinsella

Slides posted at http://www.slideshare.net/jlkinsel
Addendum

• Classification and grouping of system calls
  http://seclab.cs.sunysb.edu/sekar/papers/syscallclassif.htm
Data Sources

Data and some graphics provided by:

• Container Adoption Challenges: RightScale 2016 State of the Cloud
• Layered container image: Ubuntu
• Namespaces and cgroups images: IBM