Docker – Insecurely Wrapping Secure Software.

Aleksa Sarai

SYD0x05

March 2015

A short(ish) spiel on how monolitic tools like Docker can create insecure systems through trivial vulnerabilities, especially if they are built atop secure software.

Aleksa Sarai

SYD0x05

March 2015

```
puts("Hello, s3ct41k5!");
```

`puts("Hello, s3ct41k5!");`

* Hello there, I'm Aleksa (aka `cyphar`).

`puts("Hello, s3ct41k5!");`

* Hello there, I'm Aleksa (aka `cyphar`).
* I've been interested in hacking Gibsons / writting cool software since I was a young 'un.

`puts("Hello, s3ct41k5!");`

* Hello there, I'm Aleksa (aka `cyphar`).
* I've been interested in hacking Gibsons / writting cool software since I was a young 'un.
* More relevant to this talk, I'm a maintainer of and active contributor to Docker (as well as `libcontainer`).

`puts("Hello, s3ct41k5!");`

* Hello there, I'm Aleksa (aka `cyphar`).
* I've been interested in hacking Gibsons / writting cool software since I was a young 'un.
* More relevant to this talk, I'm a maintainer of and active contributor to Docker (as well as `libcontainer`).
  * ...which means I have a somewhat unique view of the state of Docker's security.

`puts("Hello, s3ct41k5!");`

* Hello there, I'm Aleksa (aka `cyphar`).
* I've been interested in hacking Gibsons / writting cool software since I was a young 'un.
* More relevant to this talk, I'm a maintainer of and active contributor to Docker (as well as `libcontainer`).
  * ...which means I have a somewhat unique view of the state of Docker's security.
* **tl;dr**: If you thought that Docker was a cool way to use secure Linux kernel features, think again.

`puts("Hello, s3ct41k5!");`

* Hello there, I'm Aleksa (aka `cyphar`).
* I've been interested in hacking Gibsons / writting cool software since I was a young 'un.
* More relevant to this talk, I'm a maintainer of and active contributor to Docker (as well as `libcontainer`).
  * ...which means I have a somewhat unique view of the state of Docker's security.
* **tl;dr**: If you thought that Docker was a cool way to use secure Linux kernel features, think again.
* *Disclaimer*: I don't work for Docker, so anything I say (offensive or otherwise) isn't funded by that sweet, *sweet* Docker money.

Docker money?

Docker money?

Figure: "1 g0t th4t D0ck3r m0n3y!"

What is an Docker? How I can haz one?

What is an Docker? How I can haz one?

* Who's heard of Docker?

What is an Docker? How I can haz one?

* Who's heard of Docker / used Docker?

What is an Docker? How I can haz one?

* Who's heard of Docker / used Docker / worked on Docker?

What is an Docker? How I can haz one?

* Who's heard of Docker / used Docker / worked on Docker?
* Kick-ass new Linux technology that provides trivial (semi-)containerisation, consistent development environments (without the negatives of heavy VMs), makes unicorns real, etc.

What is an Docker? How I can haz one?

* Who's heard of Docker / used Docker / worked on Docker?
* ~~Kick-ass new Linux technology that provides trivial (semi-)containerisation, consistent development environments (without the negatives of heavy VMs), makes unicorns real, etc.~~
  * — not really, it's more of a wrapper of fairly new-ish Linux kernel technologies (cgroups and Linux namespaces).

What is an Docker? How I can haz one?

* Who's heard of Docker / used Docker / worked on Docker?
* ~~Kick-ass new Linux technology that provides trivial (semi-)containerisation, consistent development environments (without the negatives of heavy VMs), makes unicorns real, etc.~~
  * — not really, it's more of a wrapper of fairly new-ish Linux kernel technologies (cgroups and Linux namespaces).
* The reason why people (including me) rave about Docker so much is because it solves a problem that developers and sysadmins have been trying to solve since time immemorial:

What is an Docker? How I can haz one?

* Who's heard of Docker / used Docker / worked on Docker?
* ~~Kick-ass new Linux technology that provides trivial (semi-)containerisation, consistent development environments (without the negatives of heavy VMs), makes unicorns real, etc.~~
  * — not really, it's more of a wrapper of fairly new-ish Linux kernel technologies (cgroups and Linux namespaces).
* The reason why people (including me) rave about Docker so much is because it solves a problem that developers and sysadmins have been trying to solve since time immemorial:
  * How *exactly* do you ensure that you can deploy some software anywhere in your datacenter without undergoing dependency hell?

What is an Docker? How I can haz one?

* But none of that *really* matters for this talk. All that matters for the purposes of this talk is that Docker is wrapping some pretty damn cool Linux kernel features with a fairly nice interface.

What is an Docker? How I can haz one?

* But none of that *really* matters for this talk. All that matters for the purposes of this talk is that Docker is wrapping some pretty damn cool Linux kernel features with a fairly nice interface.
* . . . but I'd be glad to demo it if you like. `:P`

And herein lies the issue . . .

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?
* **tl;dr**?

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?
* **tl;dr**? `[+] w00t w00t g0t r00t :)`.

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?
* **tl;dr**? `[+] w00t w00t g0t r00t :).`
* Why?

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?
* **tl;dr**? `[+] w00t w00t g0t r00t :)`.
* Why? Because `/usr/bin/docker` is a daemon on your system running as `r00t` and provides an API to clients on the system. Vulnerabilities in Docker are vulnerabilities in a long-running `r00t`-level daemon.

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?
* **tl;dr**? `[+] w00t w00t g0t r00t :).`
* Why? Because `/usr/bin/docker` is a daemon on your system running as `r00t` and provides an API to clients on the system. Vulnerabilities in Docker are vulnerabilities in a long-running `r00t`-level daemon.
  * Even though the processes are containerised by Linux, if the wrappers are vulnerable the whole system becomes vulnerable.

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?
* **tl;dr**? `[+] w00t w00t g0t r00t :).`
* Why? Because `/usr/bin/docker` is a daemon on your system running as `r00t` and provides an API to clients on the system. Vulnerabilities in Docker are vulnerabilities in a long-running `r00t`-level daemon.
  * Even though the processes are containerised by Linux, if the wrappers are vulnerable the whole system becomes vulnerable.
  * *Aside: the kernel actually doesn't have any concept of a "container".*

And herein lies the issue . . .

* So, what happens if you find a trivial security vulnerability in Docker?
* **tl;dr**? `[+] w00t w00t g0t r00t :).`
* Why? Because `/usr/bin/docker` is a daemon on your system running as `r00t` and provides an API to clients on the system. Vulnerabilities in Docker are vulnerabilities in a long-running `r00t`-level daemon.
  * Even though the processes are containerised by Linux, if the wrappers are vulnerable the whole system becomes vulnerable.
  * *Aside: the kernel actually doesn't have any concept of a "container".*
* "What vulnerabilities?" I hear you say . . .

Symlinks!

Symlinks!

* Everyone loves symlinks, right?

Symlinks!

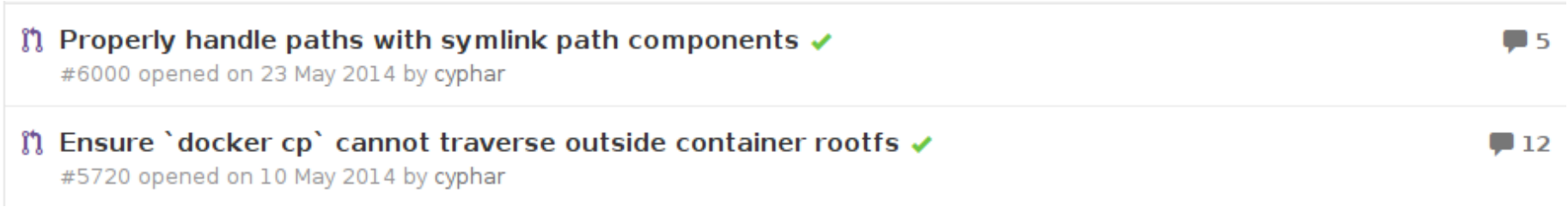
* Everyone loves symlinks, right? . . . *right guys*?

Symlinks!

* Everyone loves symlinks, right? . . . *right guys*?
* Turns out, symlinks are hard.

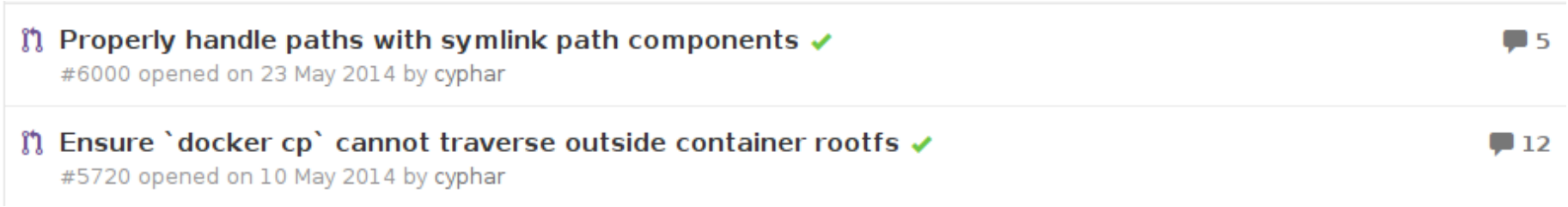
Symlinks!

* Everyone loves symlinks, right? . . . *right guys*?
* Turns out, symlinks are hard.
* Like, **really** hard.



Symlinks!

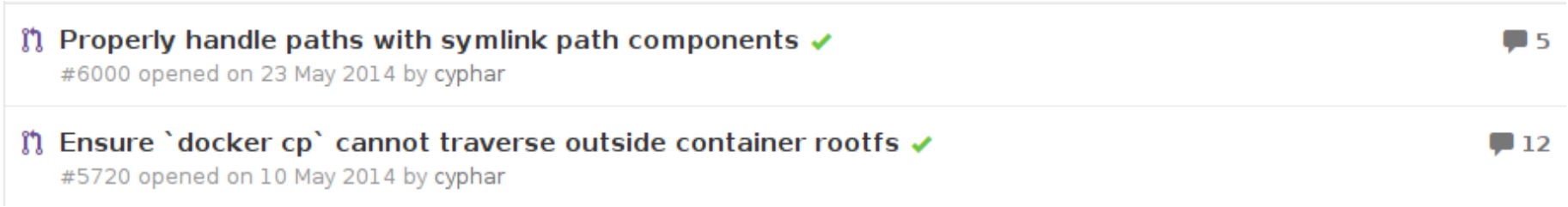
* Everyone loves symlinks, right? . . . *right guys*?
* Turns out, symlinks are hard.
* Like, **really** hard.



* The above are fixes for **two** vulnerabilities involving path sanitisation and symlinks.

Symlinks!

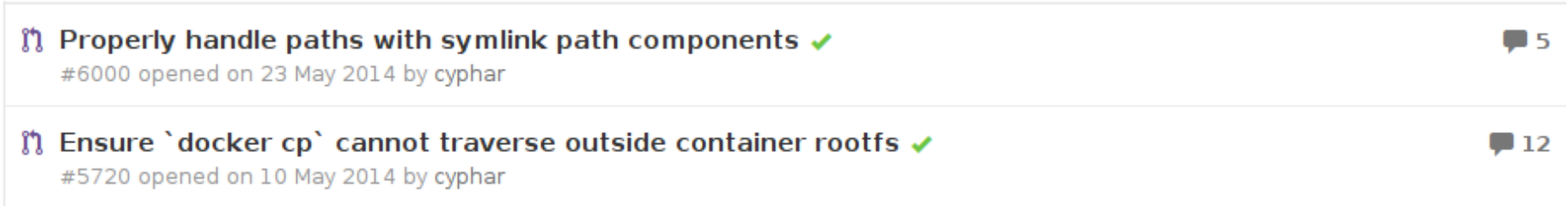
* Everyone loves symlinks, right? . . . *right guys*?
* Turns out, symlinks are hard.
* Like, **really** hard.



* The above are fixes for **two** vulnerabilities involving path sanitisation and symlinks.
  * They allow you to read **any** file as though you are `r00t`.

Symlinks!

* Everyone loves symlinks, right? . . . *right guys*?
* Turns out, symlinks are hard.
* Like, **really** hard.



* The above are fixes for **two** vulnerabilities involving path sanitisation and symlinks.
  * They allow you to read **any** file as though you are `r00t`.
* This vulnerability affects `docker<0.12.0`, so if you've updated in the last year you're not vulnerable.

Symlinks!

* So, how did the vulnerability work?

Symlinks!

* So, how did the vulnerability work?
* Docker provides the ability to copy files from a container using the `docker cp` command.

Symlinks!

* So, how did the vulnerability work?
* Docker provides the ability to copy files from a container using the `docker cp` command.
* However, the container path was not sanitised properly (see: not sanitised **at all**).

Symlinks!

* So, how did the vulnerability work?
* Docker provides the ability to copy files from a container using the `docker cp` command.
* However, the container path was not sanitised properly (see: not sanitised **at all**).
* So the following paths would allow you to copy `/etc/shadow` from the host filesystem:

Symlinks!

* So, how did the vulnerability work?
* Docker provides the ability to copy files from a container using the `docker cp` command.
* However, the container path was not sanitised properly (see: not sanitised **at all**).
* So the following paths would allow you to copy `/etc/shadow` from the host filesystem:
  1. `../../../../../../../etc/shadow`.

Symlinks!

* So, how did the vulnerability work?
* Docker provides the ability to copy files from a container using the `docker cp` command.
* However, the container path was not sanitised properly (see: not sanitised **at all**).
* So the following paths would allow you to copy `/etc/shadow` from the host filesystem:
  1. `../../../../../../../etc/shadow`.
  2. `/symlink/shadow` (where `/symlink` points to `/etc/`).

Symlinks!

* So, how did the vulnerability work?
* Docker provides the ability to copy files from a container using the `docker cp` command.
* However, the container path was not sanitised properly (see: not sanitised **at all**).
* So the following paths would allow you to copy `/etc/shadow` from the host filesystem:
  1. `../../../../../../../etc/shadow`.
  2. `/symlink/shadow` (where `/symlink` points to `/etc/`).
  3. `/symlink/shadow` (where `/symlink` points to `../../../../../../../etc/`).

Symlinks!

* So, how did the vulnerability work?
* Docker provides the ability to copy files from a container using the `docker cp` command.
* However, the container path was not sanitised properly (see: not sanitised **at all**).
* So the following paths would allow you to copy `/etc/shadow` from the host filesystem:
  1. `../../../../../../../etc/shadow`.
  2. `/symlink/shadow` (where `/symlink` points to `/etc/`).
  3. `/symlink/shadow` (where `/symlink` points to `../../../../../../../etc/`).
* There were a few others, but for some reason I've been unable to replicate them – even after `git bisecting` the repo.

Demo!

Moar symlinks! CVE-2014-9356.

Moar symlinks! CVE-2014-9356.

* Remember how I said symlinks are hard?

Moar symlinks! CVE-2014-9356.

* Remember how I said symlinks are hard?
* Well, here's a vulnerability present in `docker<1.3.3`.

Moar symlinks! CVE-2014-9356.

* Remember how I said symlinks are hard?
* Well, here's a vulnerability present in `docker<1.3.3`.
* Turns out that some other features of Docker wouldn't correctly scope symlinks and then you had read-write access to the host filesystem.

Moar symlinks! CVE-2014-9356.

* Remember how I said symlinks are hard?
* Well, here's a vulnerability present in `docker<1.3.3`.
* Turns out that some other features of Docker wouldn't correctly scope symlinks and then you had read-write access to the host filesystem.
* Looks like they didn't learn their lesson the first time . . .

Moar symlinks! CVE-2014-9356.

* So, how did this vulnerability work?

Moar symlinks! CVE-2014-9356.

* So, how did this vulnerability work?
* This was actually a general vulnerability, caused by broken path sanitisation in several places inside `Dockerfile` evaluation.

Moar symlinks! CVE-2014-9356.

* So, how did this vulnerability work?
* This was actually a general vulnerability, caused by broken path sanitisation in several places inside `Dockerfile` evaluation.
* One example of this vulnerability is that the `VOLUME` instruction would create a new `VFS` volume, but instructions like `COPY` would still traverse a symlinked volume path when referencing data in the volume:

Moar symlinks! CVE-2014-9356.

* So, how did this vulnerability work?
* This was actually a general vulnerability, caused by broken path sanitisation in several places inside `Dockerfile` evaluation.
* One example of this vulnerability is that the `VOLUME` instruction would create a new `VFS` volume, but instructions like `COPY` would still traverse a symlinked volume path when referencing data in the volume:
  1. Create a symlink as the volume path pointing to `/../../../../../../../<path>`.

Moar symlinks! CVE-2014-9356.

* So, how did this vulnerability work?
* This was actually a general vulnerability, caused by broken path sanitisation in several places inside `Dockerfile` evaluation.
* One example of this vulnerability is that the `VOLUME` instruction would create a new `VFS` volume, but instructions like `COPY` would still traverse a symlinked volume path when referencing data in the volume:
  1. Create a symlink as the volume path pointing to `/../../../../../../../<path>`.
  2. Expose the `VOLUME`.

Moar symlinks! CVE-2014-9356.

* So, how did this vulnerability work?
* This was actually a general vulnerability, caused by broken path sanitisation in several places inside `Dockerfile` evaluation.
* One example of this vulnerability is that the `VOLUME` instruction would create a new `VFS` volume, but instructions like `COPY` would still traverse a symlinked volume path when referencing data in the volume:
  1. Create a symlink as the volume path pointing to `/../../../../../../../<path>`.
  2. Expose the `VOLUME`.
  3. `COPY` any file to the volume and it gets written to `<path>` on the host.

Moar symlinks! CVE-2014-9356.

* So, how did this vulnerability work?
* This was actually a general vulnerability, caused by broken path sanitisation in several places inside `Dockerfile` evaluation.
* One example of this vulnerability is that the `VOLUME` instruction would create a new `VFS` volume, but instructions like `COPY` would still traverse a symlinked volume path when referencing data in the volume:
  1. Create a symlink as the volume path pointing to `/../../../../../../../<path>`.
  2. Expose the `VOLUME`.
  3. `COPY` any file to the volume and it gets written to `<path>` on the host.
* It was also possible to use `docker cp` to copy data from `<path>` on the host (this wasn't documented anywhere, I discovered it while setting up the demos).

Demo!

Other (fairly trivial) vulnerabilities.

Other (fairly trivial) vulnerabilities.

CVE-2014-9357 By overwriting the `xz` binary in the archive extraction `chroot`, you could escalate privileges and get `r00t` on the host. `docker=1.3.2`

Other (fairly trivial) vulnerabilities.

CVE-2014-9357 By overwriting the `xz` binary in the archive extraction `chroot`, you could escalate privileges and get `r00t` on the host. `docker=1.3.2`

CVE-2014-9358 Image IDs weren't properly sanitised when communicating with the registry or from `tar` image archives, so you had a path traversal again. `docker<1.3.3`

Takeaways.

Takeaways.

* None of the above vulnerabilities actually had anything to do with the Linux kernel features being wrapped.

Takeaways.

* None of the above vulnerabilities actually had anything to do with the Linux kernel features being wrapped.
  * They were all home-grown vulnerabilities in Docker's wrapping of said kernel features.

Takeaways.

* None of the above vulnerabilities actually had anything to do with the Linux kernel features being wrapped.
  * They were all home-grown vulnerabilities in Docker's wrapping of said kernel features.
  * As a result, you could escape the Linux containerisation because the wrappers were insecure.

Takeaways.

* None of the above vulnerabilities actually had anything to do with the Linux kernel features being wrapped.
  * They were all home-grown vulnerabilities in Docker's wrapping of said kernel features.
  * As a result, you could escape the Linux containerisation because the wrappers were insecure.
* I personally find it interesting how many of the vulnerabilities are related to path sanitisation.

Dangerous patterns.

Dangerous patterns.

* So far, most of the vulnerabilities required access to the Docker client on the host (i.e. read-write access to the `docker.sock` socket).

Dangerous patterns.

* So far, most of the vulnerabilities required access to the Docker client on the host (i.e. read-write access to the `docker.sock` socket).
  * . . . which would seem to make them useless for breaking **out** of a container without host access.

Dangerous patterns.

* So far, most of the vulnerabilities required access to the Docker client on the host (i.e. read-write access to the `docker.sock` socket).
  * . . . which would seem to make them useless for breaking **out** of a container without host access.
* As it turns out, there are a **bunch** of dangerous patterns that result in you being able to gain read-write access to the **host's** `docker.sock` socket.

Dangerous patterns.

* So far, most of the vulnerabilities required access to the Docker client on the host (i.e. read-write access to the `docker.sock` socket).
  * ...which would seem to make them useless for breaking **out** of a container without host access.
* As it turns out, there are a **bunch** of dangerous patterns that result in you being able to gain read-write access to the **host's** `docker.sock` socket.
* These dangerous patterns cause vulnerabilities that are traditionally host-facing (which include most of the ones I've covered) now become container-facing, escalating the vulnerabilities to the level of sandbox bypassing.

Dangerous patterns.

* The most worrying (and widespread) pattern of this kind is known as the "Docker plugin" system.

Dangerous patterns.

* The most worrying (and widespread) pattern of this kind is known as the "Docker plugin" system.
  * It is a very elaborate hack of bind-mounting the host's `docker.sock` into containers in order for some processes to manage some aspect of Docker. Needless to say, if those containerised processes get exploited, the whole system is exploited.

Dangerous patterns.

* The most worrying (and widespread) pattern of this kind is known as the "Docker plugin" system.
  * It is a very elaborate hack of bind-mounting the host's `docker.sock` into containers in order for some processes to manage some aspect of Docker. Needless to say, if those containerised processes get exploited, the whole system is exploited.
* There have also been several proposals (most recently #10296) which have tried to make the above hack an **out of the box feature**.

Dangerous patterns.

* The most worrying (and widespread) pattern of this kind is known as the "Docker plugin" system.
  * It is a very elaborate hack of bind-mounting the host's `docker.sock` into containers in order for some processes to manage some aspect of Docker. Needless to say, if those containerised processes get exploited, the whole system is exploited.
* There have also been several proposals (most recently #10296) which have tried to make the above hack an **out of the box feature**.

  wat.

Prevailing issues.

Prevailing issues.

* Docker still doesn't take advantage of all the kernel features for containerisation (specifically, it still doesn't use the `USER` namespace).

Prevailing issues.

* Docker still doesn't take advantage of all the kernel features for containerisation (specifically, it still doesn't use the `USER` namespace).
* Just reading through the source, there are still a **bunch** of cases where path-related code **still** isn't taking advantage of the right sanitisation wrappers.

Prevailing issues.

* Docker still doesn't take advantage of all the kernel features for containerisation (specifically, it still doesn't use the `USER` namespace).
* Just reading through the source, there are still a **bunch** of cases where path-related code **still** isn't taking advantage of the right sanitisation wrappers.
* There still hasn't been a proper security audit of the whole Docker source code (although some people have done their due diligence in finding the above vulnerabilities).

Prevailing issues.

* Docker still doesn't take advantage of all the kernel features for containerisation (specifically, it still doesn't use the `USER` namespace).
* Just reading through the source, there are still a **bunch** of cases where path-related code **still** isn't taking advantage of the right sanitisation wrappers.
* There still hasn't been a proper security audit of the whole Docker source code (although some people have done their due diligence in finding the above vulnerabilities).
* There are **no** ACLs in place for users that can write to `docker.sock`. If you can write to `docker.sock` you **are** `r00t`, `w00t w00t` and all.

Prevailing issues.

* Docker still doesn't take advantage of all the kernel features for containerisation (specifically, it still doesn't use the `USER` namespace).
* Just reading through the source, there are still a **bunch** of cases where path-related code **still** isn't taking advantage of the right sanitisation wrappers.
* There still hasn't been a proper security audit of the whole Docker source code (although some people have done their due diligence in finding the above vulnerabilities).
* There are **no** ACLs in place for users that can write to `docker.sock`. If you can write to `docker.sock` you **are** `r00t`, `w00t w00t` and all.
* *Aside*: Isn't it interesting that all of the core maintainers work for one company?

Securing Docker?

Securing Docker?

* While the future of Docker (from a security perspective) might sound fairly bleak, *it is possible to secure Docker with some best practices and other kernel security features*.

Best practices.

Best practices.

* Do **NOT**(!) run random containers on your system as `r00t`.

Best practices.

* Do **NOT**(!) run random containers on your system as `r00t`.
* Do not expose the Docker API, either by binding to a public TCP port or allowing random users to write to `docker.sock`.

Best practices.

* Do **NOT**(!) run random containers on your system as `r00t`.
* Do not expose the Docker API, either by binding to a public TCP port or allowing random users to write to `docker.sock`.
* Make sure that all the images you run are verified (or built by **you** from a verified image). And of course, make sure the verified image is signed.

More kernel features.

More kernel features.

* SELinux and AppArmor provide some further containerisation of Docker containers through kernel-level access control policies on kernel objects.

More kernel features.

* SELinux and AppArmor provide some further containerisation of Docker containers through kernel-level access control policies on kernel objects.
* Make sure you're running the latest stable kernel (with the GRSecurity and PAX patches applied), so you are less likely to get `l0cal r00t3d`.

More kernel features.

* SELinux and AppArmor provide some further containerisation of Docker containers through kernel-level access control policies on kernel objects.
* Make sure you're running the latest stable kernel (with the GRSecurity and PAX patches applied), so you are less likely to get `l0cal r00t3d`.
  * Remember, `r00t` in a Docker container is `r00t` outside of it, so the kernel lets you do any syscall tomfoolery you want.

More kernel features.

* SELinux and AppArmor provide some further containerisation of Docker containers through kernel-level access control policies on kernel objects.
* Make sure you're running the latest stable kernel (with the GRSecurity and PAX patches applied), so you are less likely to get `l0cal r00t3d`.
  * Remember, `r00t` in a Docker container is `r00t` outside of it, so the kernel lets you do any syscall tomfoolery you want.
* If you use the LXC execdriver, you can take advantage of the `USER` namespace.

More kernel features.

* SELinux and AppArmor provide some further containerisation of Docker containers through kernel-level access control policies on kernel objects.
* Make sure you're running the latest stable kernel (with the GRSecurity and PAX patches applied), so you are less likely to get `l0cal r00t3d`.
  * Remember, `r00t` in a Docker container is `r00t` outside of it, so the kernel lets you do any syscall tomfoolery you want.
* If you use the LXC execdriver, you can take advantage of the `USER` namespace.
  * — but don't use the LXC execdriver. There have been several cases where vulnerabilities found in both the native and LXC execdrivers have not been fixed in the LXC execdriver for several releases.

Securing Docker?

* Essentially, just follow the general best security practices even when using Docker. Docker is an ingredient, not a complete solution.

But I still love it.

But I still love it.

* But notwithstanding all of these issues, I still think that Docker is one of the coolest projects being worked on today.

But I still love it.

* But notwithstanding all of these issues, I still think that Docker is one of the coolest projects being worked on today.
* My only complaint is that the Docker development community needs to be more aware and should actively try to solve those issues.

But I still love it.

* But notwithstanding all of these issues, I still think that Docker is one of the coolest projects being worked on today.
* My only complaint is that the Docker development community needs to be more aware and should actively try to solve those issues.
* And actually, they are. `USER` namespaces are confirmed to land in `docker=1.6`. And people like myself are constantly working on finding and fixing security vulnerabilities.

The Big Picture™.

The Big Picture™.

* Okay, so maybe you don't use Docker and you don't do pentests of Dockerised systems.

The Big Picture™.

* Okay, so maybe you don't use Docker and you don't do pentests of Dockerised systems. That's fine! There is a bigger picture here.

The Big Picture™.

* Okay, so maybe you don't use Docker and you don't do pentests of Dockerised systems. That's fine! There is a bigger picture here.
* Attacking a system is all about attacking the lowest hanging fruit.

The Big Picture™.

* Okay, so maybe you don't use Docker and you don't do pentests of Dockerised systems. That's fine! There is a bigger picture here.
* Attacking a system is all about attacking the lowest hanging fruit.
* And insecure wrappers of secure software are the lowest hanging fruit in such systems. But they're also in the weird position that they are "trusted" to manage secure sofwtare.

The Big Picture™.

* Okay, so maybe you don't use Docker and you don't do pentests of Dockerised systems. That's fine! There is a bigger picture here.
* Attacking a system is all about attacking the lowest hanging fruit.
* And insecure wrappers of secure software are the lowest hanging fruit in such systems. But they're also in the weird position that they are "trusted" to manage secure sofwtare.
* So don't be fooled by the idea that a secure piece of software suddenly lends its security to the insecure wrappers that manage it.

The Bottom Line™?

The Bottom Line™?

Insecure wrappers of secure software create insecure systems which have the illusion of security.

Shameful Plugs.

Shameful Plugs.

* If you want to see more of my stuff, check out my:
  Website : `https://www.cyphar.com/`
  Twitter : `@lordcyphar`

Questions?

fin.