Alexandra Ulsh

Blog Projects Talks Contact

Securely using .npmrc files in Docker images

Jun 25, 2018
9 minute read

Update - Docker build secrets

On February 24th, 2019 I published a follow-up post about using Docker build secrets instead of multi-stage builds. I recommend reading that post after this one!

Building Docker images with private npm packages

I recently completed a security audit of Docker images for a project. These images used `.npmrc` files (npm config files) to download private npm packages. By default, an `.npmrc` file contains a token with read/write access to your private npm packages. It looks like this:

```
//registry.npmjs.org/:_authToken=<npm token>
```

Most blog posts, Stack Overflow answers, and documentation recommend you delete `.npmrc` files from your `Dockerfile` after installing private npm packages. Many of these guides don't cover how to remove `.npmrc` files from intermediate images or npm tokens from the image commit history though. In fairness, most of these resources date from before Docker shipped multi-stage builds in Docker v17.05 in May 2017.

Multi-stage builds allow us to securely use `.npmrc` files in our Docker images. Only the intermediate images and commit history from the last build stage end up in our final Docker image. This enables us to `npm install` our private packages in earlier build stages without leaking our tokens in the final image.

Overview

In this blog post, I'll first describe the common ways people use `.npmrc` files insecurely in Docker images. For each scenario, I'll show how an attacker can exploit it to steal your npm access tokens. Finally, I'll explain how multi-stage builds enable you to securely use `.npmrc` files in your Docker images.

This blog post focuses on using Docker with Node.js and npm. The concepts I cover apply to any `Dockerfile` that uses tokens, passwords, or other secrets though.

I also created a companion GitHub repository for this blog post so you can follow along with my examples. You can check it out at https://github.com/alulsh/docker-npmrc-security.

I have revoked all npm tokens featured in all screenshots.

#1 - Leaving `.npmrc` files in Docker containers

If you fail to remove your `.npmrc` file from your `Dockerfile`, it will be saved in your Docker image. It will exist on the file system of any Docker container you create from that image.

Here's a `Dockerfile` where we create an `.npmrc` file but fail to delete it after running `npm install`:

```
FROM node:8.11.3-alpine
ARG NPM_TOKEN

WORKDIR /private-app
COPY . /private-app

RUN echo "//registry.npmjs.org/:_authToken=$NPM_TOKEN" > .npmrc
RUN npm install

EXPOSE 3000

CMD ["node","index.js"]
```

This `Dockerfile` creates the `.npmrc` file using an `NPM_TOKEN` environment variable that we pass in as a build argument (`ARG NPM_TOKEN`). To build it locally, first clone the companion GitHub repository at https://github.com/alulsh/docker-npmrc-security then:

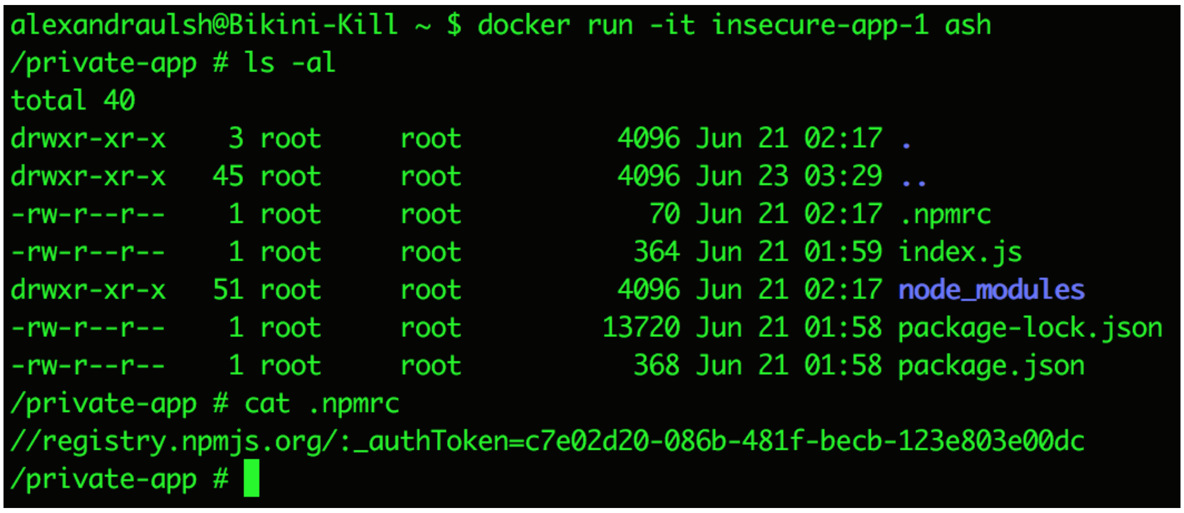
1. Run `npm token create --read-only` to create a read only npm token.
2. Run `export NPM_TOKEN=<npm token>` to set this token as an environment variable.
3. Run `docker build . -f Dockerfile-insecure-1 -t insecure-app-1 --build-arg NPM_TOKEN=$NPM_TOKEN`.

Stealing `.npmrc` files from Docker containers

If an attacker compromises your Docker container or manages to execute arbitrary commands on your application servers, they can steal your npm tokens by running `ls -al && cat .npmrc`.

You can try it out yourself:

1. Run `docker run -it insecure-app-1 ash` to start the container. We need to use `ash` instead of `bash` since we're running Alpine Linux.
2. Run `ls -al`. You should see an `.npmrc` file in the `/private-app` directory.
3. Run `cat .npmrc`.



```
alexandraulsh@Bikini-Kill ~ $ docker run -it insecure-app-1 ash
/private-app # ls -al
total 40
drwxr-xr-x    3 root     root          4096 Jun 21 02:17 .
drwxr-xr-x   45 root     root          4096 Jun 23 03:29 ..
-rw-r--r--    1 root     root            70 Jun 21 02:17 .npmrc
-rw-r--r--    1 root     root           364 Jun 21 01:59 index.js
drwxr-xr-x   51 root     root          4096 Jun 21 02:17 node_modules
-rw-r--r--    1 root     root         13720 Jun 21 01:58 package-lock.json
-rw-r--r--    1 root     root           368 Jun 21 01:58 package.json
/private-app # cat .npmrc
//registry.npmjs.org/:_authToken=c7e02d20-086b-481f-becb-123e803e00dc
/private-app # 
```

Stealing npm tokens from a running Docker container

#2 - Leaving `.npmrc` files in Docker intermediate images

Most guides recommend deleting your `.npmrc` file after running `npm install` in your `Dockerfile`. For example:

```
ARG NPM_TOKEN

RUN echo "//registry.npmjs.org/:_authToken=$NPM_TOKEN" > .npmrc
RUN npm install
RUN rm -f .npmrc
```

Fortunately, this does remove the `.npmrc` file from the top layer of your Docker image and from any containers. If an attacker compromised your Docker container they would not be able to steal your npm token.

Unfortunately, each `RUN` instruction creates a separate layer (also called intermediate image) in a Docker image. Multiple layers make up a Docker image. In the above `Dockerfile`, the `.npmrc` file is stored in the layer created by the `RUN echo "//registry.npmjs.org/:_authToken=$NPM_TOKEN" > .npmrc` instruction.

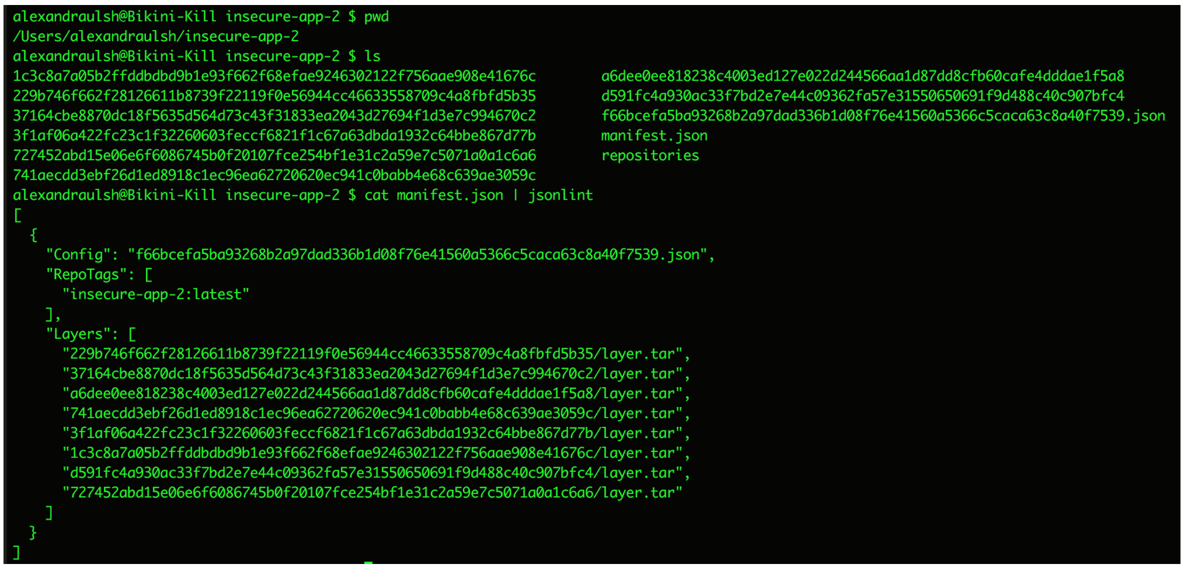
Stealing `.npmrc` files from intermediate images

To view the layers of your Docker image an attacker would need to either compromise your Docker daemon or obtain a copy of your Docker image. You likely won’t publish Docker images with private source code to public Docker registries. You might distribute these Docker images to contractors, consultants, and customers though. These third parties need your private source code but they do not need your npm tokens.

Here’s how an attacker or third-party could steal `.npmrc` tokens from layers in your Docker images:

1. Build the example Docker image with `docker build . -f Dockerfile-insecure-2 -t insecure-app-2 --build-arg NPM_TOKEN=$NPM_TOKEN`.
2. Run `docker save insecure-app-2 -o ~/insecure-app-2.tar` to save the Docker image as a tarball.

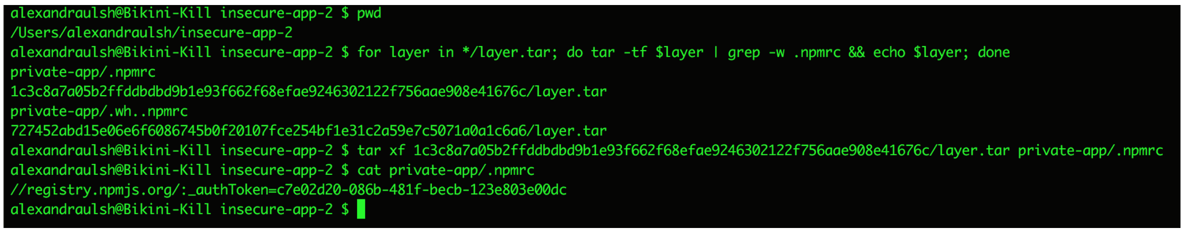
3. Run `mkdir ~/insecure-app-2 && tar xf ~/insecure-app-2.tar -C ~/insecure-app-2` to untar to `~/insecure-app-2`.
4. Run `cd ~/insecure-app-2`. For fun, try running `ls` and `cat manifest.json` in this directory to view all of the layers.



```
alexandraulsh@Bikini-Kill insecure-app-2 $ pwd
/Users/alexandraulsh/insecure-app-2
alexandraulsh@Bikini-Kill insecure-app-2 $ ls
1c3c8a7a05b2ffddbdbd9b1e93f662f68efae9246302122f756aae908e41676c        a6dee0ee818238c4003ed127e022d244566aa1d87dd8cfb60cafe4dddae1f5a8
229b746f662f28126611b8739f22119f0e56944cc46633558709c4a8fbfd5b35        d591fc4a930ac33f7bd2e7e44c09362fa57e31550650691f9d488c40c907bfc4
37164cbe8870dc18f5635d564d73c43f31833ea2043d27694f1d3e7c994670c2        f66bcefa5ba93268b2a97dad336b1d08f76e41560a5366c5caca63c8a40f7539.json
3f1af06a422fc23c1f32260603feccf6821f1c67a63dbda1932c64bbe867d77b        manifest.json
727452abd15e06e6f6086745b0f20107fce254bf1e31c2a59e7c5071a0a1c6a6        repositories
741aecdd3ebf26d1ed8918c1ec96ea62720620ec941c0babb4e68c639ae3059c
alexandraulsh@Bikini-Kill insecure-app-2 $ cat manifest.json | jsonlint
[
  {
    "Config": "f66bcefa5ba93268b2a97dad336b1d08f76e41560a5366c5caca63c8a40f7539.json",
    "RepoTags": [
      "insecure-app-2:latest"
    ],
    "Layers": [
      "229b746f662f28126611b8739f22119f0e56944cc46633558709c4a8fbfd5b35/layer.tar",
      "37164cbe8870dc18f5635d564d73c43f31833ea2043d27694f1d3e7c994670c2/layer.tar",
      "a6dee0ee818238c4003ed127e022d244566aa1d87dd8cfb60cafe4dddae1f5a8/layer.tar",
      "741aecdd3ebf26d1ed8918c1ec96ea62720620ec941c0babb4e68c639ae3059c/layer.tar",
      "3f1af06a422fc23c1f32260603feccf6821f1c67a63dbda1932c64bbe867d77b/layer.tar",
      "1c3c8a7a05b2ffddbdbd9b1e93f662f68efae9246302122f756aae908e41676c/layer.tar",
      "d591fc4a930ac33f7bd2e7e44c09362fa57e31550650691f9d488c40c907bfc4/layer.tar",
      "727452abd15e06e6f6086745b0f20107fce254bf1e31c2a59e7c5071a0a1c6a6/layer.tar"
    ]
  }
]
```

Layers in a Docker image

1. Run `for layer in */layer.tar; do tar -tf $layer | grep -w .npmrc && echo $layer; done`. [Credit goes to this StackOverflow answer](#) for that one liner. You should see a list of layers with `.npmrc` files.
2. Run `tar xf <layer id>/layer.tar private-app/.npmrc` to extract `private-app/.npmrc` from the layer tarball. In my case, I needed to run `tar xf 1c3c8a7a05b2ffddbdbd9b1e93f662f68efae9246302122f756aae908e41676c/layer.tar private-app/.npmrc`.
3. Run `cat private-app/.npmrc` to view the `.npmrc` file and npm token.



```
alexandraulsh@Bikini-Kill insecure-app-2 $ pwd
/Users/alexandraulsh/insecure-app-2
alexandraulsh@Bikini-Kill insecure-app-2 $ for layer in */layer.tar; do tar -tf $layer | grep -w .npmrc && echo $layer; done
private-app/.npmrc
1c3c8a7a05b2ffddbdbd9b1e93f662f68efae9246302122f756aae908e41676c/layer.tar
private-app/.wh..npmrc
727452abd15e06e6f6086745b0f20107fce254bf1e31c2a59e7c5071a0a1c6a6/layer.tar
alexandraulsh@Bikini-Kill insecure-app-2 $ tar xf 1c3c8a7a05b2ffddbdbd9b1e93f662f68efae9246302122f756aae908e41676c/layer.tar private-app/.npmrc
alexandraulsh@Bikini-Kill insecure-app-2 $ cat private-app/.npmrc
//registry.npmjs.org/:_authToken=c7e02d20-086b-481f-becb-123e803e00dc
alexandraulsh@Bikini-Kill insecure-app-2 $ 
```

Stealing .npmrc files and npm tokens from Docker layers

#3 - Leaking npm tokens in the image commit history

More security conscious guides are aware of the Docker layer problem. They advocate creating and deleting the `.npmrc` file in the same `RUN` instruction or layer. Other guides recommend using the `--squash` flag when running `docker build`. Here's a `Dockerfile` where we create and delete our `.npmrc` file in the same layer:

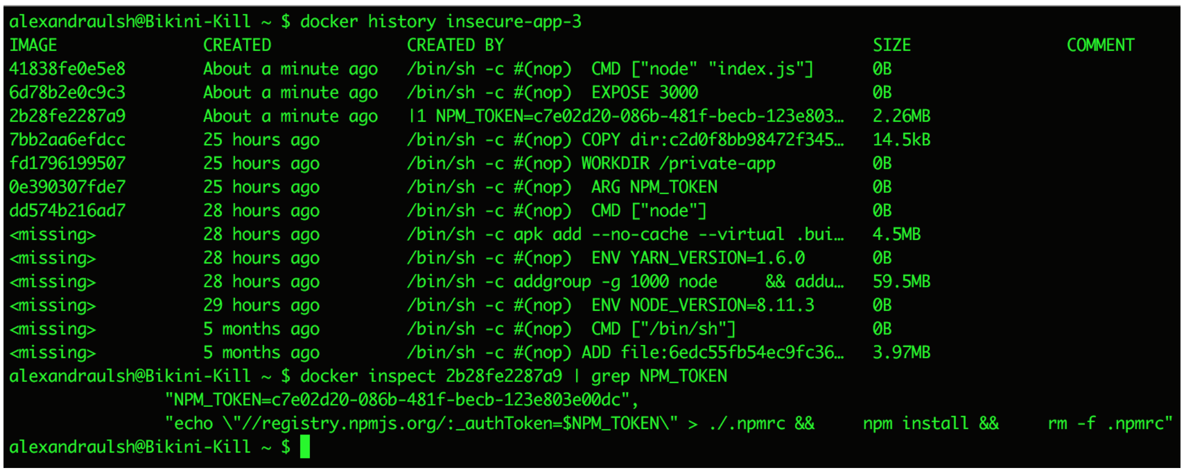
```
ARG NPM_TOKEN

RUN echo "//registry.npmjs.org/:_authToken=$NPM_TOKEN" > .npmrc && \
    npm install && \
    rm -f .npmrc
```

Both approaches prevent `.npmrc` files from being saved in layers. Unfortunately, the npm token is still visible in the commit history of the Docker image. The Docker image build process logs the plaintext values for build arguments (`ARG NPM_TOKEN`) into the commit history of an image. For this reason, the official Docker documentation on Dockerfiles warns that you should not use build arguments for secrets.

Stealing npm tokens from Docker image commit histories

Similar to viewing Docker layers, to view your image commit history an attacker or third-party would need to compromise your Docker daemon or obtain a copy of your Docker image. This “hack” is easier though and only requires one command - `docker history`. To try this yourself:

1. `docker build . -f Dockerfile-insecure-3 -t insecure-app-3 --build-arg NPM_TOKEN=$NPM_TOKEN`
2. `docker history insecure-app-3`



```
alexandraulsh@Bikini-Kill ~ $ docker history insecure-app-3
IMAGE               CREATED              CREATED BY                                      SIZE                COMMENT
41838fe0e5e8        About a minute ago   /bin/sh -c #(nop)  CMD ["node" "index.js"]      0B
6d78b2e0c9c3        About a minute ago   /bin/sh -c #(nop)  EXPOSE 3000                  0B
2b28fe2287a9        About a minute ago   |1 NPM_TOKEN=c7e02d20-086b-481f-becb-123e803…   2.26MB
7bb2aa6efdcc        25 hours ago         /bin/sh -c #(nop) COPY dir:c2d0f8bb98472f345…   14.5kB
fd1796199507        25 hours ago         /bin/sh -c #(nop) WORKDIR /private-app          0B
0e390307fde7        25 hours ago         /bin/sh -c #(nop)  ARG NPM_TOKEN                0B
dd574b216ad7        28 hours ago         /bin/sh -c #(nop)  CMD ["node"]                 0B
<missing>           28 hours ago         /bin/sh -c apk add --no-cache --virtual .bui…   4.5MB
<missing>           28 hours ago         /bin/sh -c #(nop)  ENV YARN_VERSION=1.6.0       0B
<missing>           28 hours ago         /bin/sh -c addgroup -g 1000 node     && addu…   59.5MB
<missing>           29 hours ago         /bin/sh -c #(nop)  ENV NODE_VERSION=8.11.3      0B
<missing>           5 months ago         /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           5 months ago         /bin/sh -c #(nop) ADD file:6edc55fb54ec9fc36…   3.97MB
alexandraulsh@Bikini-Kill ~ $ docker inspect 2b28fe2287a9 | grep NPM_TOKEN
                "NPM_TOKEN=c7e02d20-086b-481f-becb-123e803e00dc",
                "echo \"//registry.npmjs.org/:_authToken=$NPM_TOKEN\" > ./.npmrc &&     npm install &&     rm -f .npmrc"
alexandraulsh@Bikini-Kill ~ $ 
```

Stealing npm tokens from Docker image commit histories

Solution - Multi-stage builds

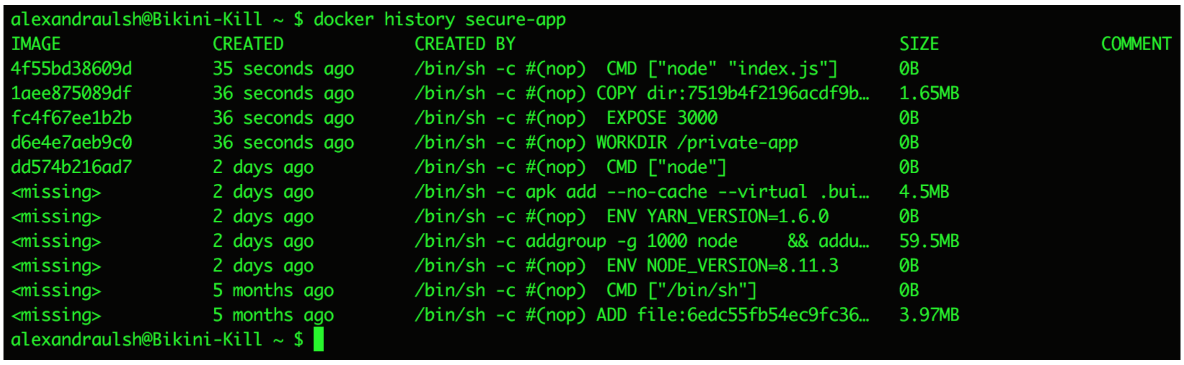
We can protect our build arguments from leaking with multi-stage builds. Only the final build in a multi-stage build will show up in the commit history of the final Docker image.

In the first stage of the build, we’ll use our `NPM_TOKEN` build argument to `npm install` our project. We can then copy our built Node application from the first stage to the second stage build using the `COPY` instruction.

Most multi-stage build tutorials and examples show two different base images. You can use the same base image for multi-stage builds. In this example, I use `node:8.11.3-alpine` as my base image for both stages.

```
# First build
FROM node:8.11.3-alpine AS build
ARG NPM_TOKEN

WORKDIR /private-app
COPY . /private-app

RUN echo "//registry.npmjs.org/:_authToken=$NPM_TOKEN" > .npmrc && \
    npm install --production && \
    rm -f .npmrc

# Second build
FROM node:8.11.3-alpine
WORKDIR /private-app

EXPOSE 3000
COPY --from=build /private-app /private-app

CMD ["node","index.js"]
```

To build this image, run `docker build . -f Dockerfile-secure -t secure-app --build-arg NPM_TOKEN=$NPM_TOKEN`. To view the history, run `docker history secure-app`.



```
alexandraulsh@Bikini-Kill ~ $ docker history secure-app
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
4f55bd38609d        35 seconds ago      /bin/sh -c #(nop)  CMD ["node" "index.js"]      0B
1aee875089df        36 seconds ago      /bin/sh -c #(nop) COPY dir:7519b4f2196acdf9b…   1.65MB
fc4f67ee1b2b        36 seconds ago      /bin/sh -c #(nop)  EXPOSE 3000                  0B
d6e4e7aeb9c0        36 seconds ago      /bin/sh -c #(nop) WORKDIR /private-app          0B
dd574b216ad7        2 days ago          /bin/sh -c #(nop)  CMD ["node"]                 0B
<missing>           2 days ago          /bin/sh -c apk add --no-cache --virtual .bui…   4.5MB
<missing>           2 days ago          /bin/sh -c #(nop)  ENV YARN_VERSION=1.6.0       0B
<missing>           2 days ago          /bin/sh -c addgroup -g 1000 node     && addu…   59.5MB
<missing>           2 days ago          /bin/sh -c #(nop)  ENV NODE_VERSION=8.11.3      0B
<missing>           5 months ago        /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           5 months ago        /bin/sh -c #(nop) ADD file:6edc55fb54ec9fc36…   3.97MB
alexandraulsh@Bikini-Kill ~ $
```

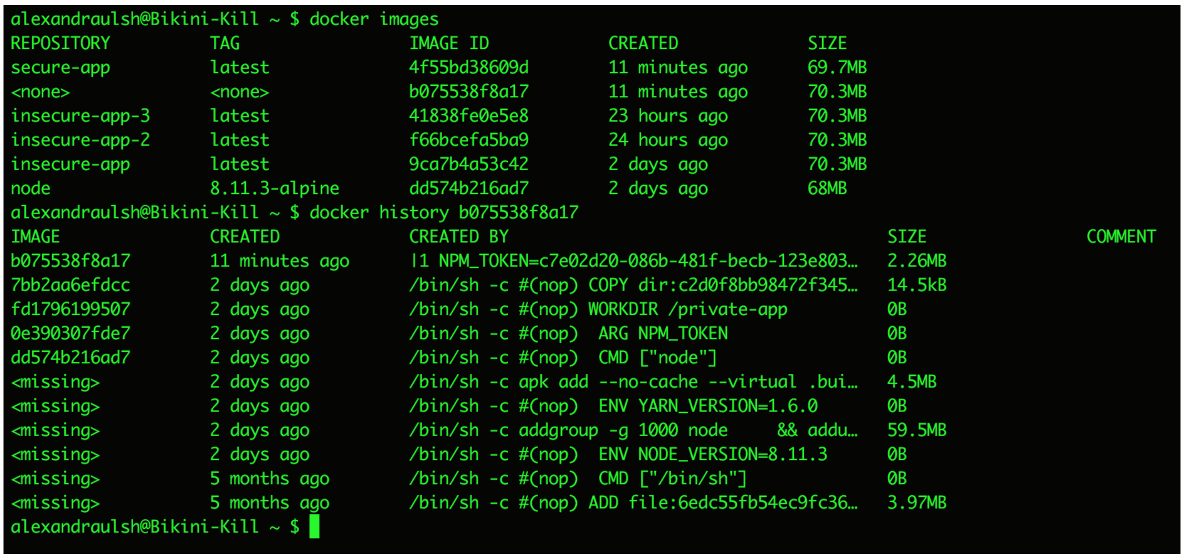
Secure Docker commit history thanks to multi-stage builds

Our npm tokens no longer leak in the commit history!

Delete untagged images from multi-stage builds

Multi-stage builds create untagged images of earlier build stages in our local Docker daemon for the Docker build cache. The commit history of these images leaks plaintext build arguments. You should delete these images after you finish your builds.

Run `docker history` on these untagged images to steal npm tokens from them. Run `docker rmi <image id>` to delete them.



```
alexandraulsh@Bikini-Kill ~ $ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
secure-app          latest              4f55bd38609d        11 minutes ago      69.7MB
<none>              <none>              b075538f8a17        11 minutes ago      70.3MB
insecure-app-3      latest              41838fe0e5e8        23 hours ago        70.3MB
insecure-app-2      latest              f66bcefa5ba9        24 hours ago        70.3MB
insecure-app        latest              9ca7b4a53c42        2 days ago          70.3MB
node                8.11.3-alpine       dd574b216ad7        2 days ago          68MB
alexandraulsh@Bikini-Kill ~ $ docker history b075538f8a17
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
b075538f8a17        11 minutes ago      |1 NPM_TOKEN=c7e02d20-086b-481f-becb-123e803…   2.26MB
7bb2aa6efdcc        2 days ago          /bin/sh -c #(nop) COPY dir:c2d0f8bb98472f345…   14.5kB
fd1796199507        2 days ago          /bin/sh -c #(nop) WORKDIR /private-app          0B
0e390307fde7        2 days ago          /bin/sh -c #(nop)  ARG NPM_TOKEN                0B
dd574b216ad7        2 days ago          /bin/sh -c #(nop)  CMD ["node"]                 0B
<missing>           2 days ago          /bin/sh -c apk add --no-cache --virtual .bui…   4.5MB
<missing>           2 days ago          /bin/sh -c #(nop)  ENV YARN_VERSION=1.6.0       0B
<missing>           2 days ago          /bin/sh -c addgroup -g 1000 node     && addu…   59.5MB
<missing>           2 days ago          /bin/sh -c #(nop)  ENV NODE_VERSION=8.11.3      0B
<missing>           5 months ago        /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           5 months ago        /bin/sh -c #(nop) ADD file:6edc55fb54ec9fc36…   3.97MB
alexandraulsh@Bikini-Kill ~ $
```

Npm tokens in untagged images from multi-stage builds

Assessing risk

If an attacker compromises your containers or Docker daemon, you'll likely be dealing with bigger issues than your `.npmrc` files. Removing npm tokens from your Docker images means you'll be dealing with one less problem though.

Depending on your threat model, deleting `.npmrc` files from your Docker images and containers may be all you need to do to mitigate most of your risk. If you distribute your Docker images to third party contractors, consultants, or customers you should use multi-stage builds to remove any secrets. This also applies if you publish images to Docker registries.

Multi-stage builds are easy to use and have little to no downsides. They can significantly improve the security, performance, and readability of your Docker images. If you use secrets to build your Docker images I recommend multi-stage builds even if you don't distribute your Docker images and aren't worried about the security of your Docker daemon.

Raising awareness of multi-stage builds

The Docker community is aware of the lack of a built-in way to manage secrets in Dockerfiles. They're currently working on ways to improve using secrets in Docker. In the meantime multi-stage builds allow us to securely use `.npmrc` files or other secrets in our Docker builds. They also improve the readability of our Dockerfiles and decrease the size of our images.

Most of the guides for using `.npmrc` files in Docker images date from before multi-stage builds in May 2017. I'm currently drafting a pull request to the official npm documentation to update their guidance to use multi-stage builds. I'll update this blog post with a link to the pull request as well as updates on its status!

Update - 22:40 EST June 25th, 2018 - I submitted an issue and a separate pull request to https://github.com/npm/docs with new guidance about multi-stage builds.