镜像安全检测工具——Clair
检测原理及部署使用

Part1 镜像结构

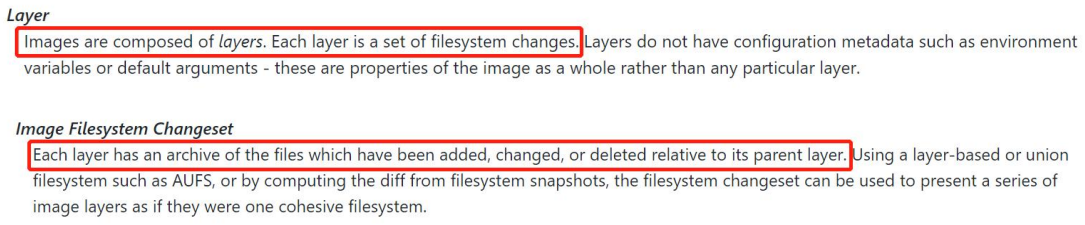
Clair 是一款对容器组件安全性进行静态检测的开源工具，其核心模块由 Red Hat 开发并开源在 github 上。为了较深入地了解 Clair 的原理，我们需要先从它的检测对象——镜像（image）的结构入手。

根据 Clair github 库中 ReadMe 介绍，Clair 目前支持满足 OCI 规范和 docker 规范的镜像的检测。本文将以 docker 镜像为例进行介绍。

Docker 镜像（image）是以层次（layer）结构组织的，layer 是组成 image 的单位，上下层 layer 之间还存在构建时的依赖关系。在谈论 docker 镜像时将会涉及到以下一些名词：

- Layer
- Image JSON
- Image Filesystem Changeset
- Layer DiffID
- Layer ChainID
- ImageID

其中 Layer 已简要介绍过，Layer 是组成 image 的单位，在其中包含了构成整体 image 的信息。Image Filesystem Changeset 记录了文件系统的变化。根据官方文档描述每个 layer 都是一组 filesystem changeset。



Layer

Images are composed of *layers*. Each layer is a set of filesystem changes. Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.

Image Filesystem Changeset

Each layer has an archive of the files which have been added, changed, or deleted relative to its parent layer. Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem.

简单来说，对一个空的目录由下而上逐层实现 layer 中的文件变更，最后呈现的结果应与镜像当前的状态相同。

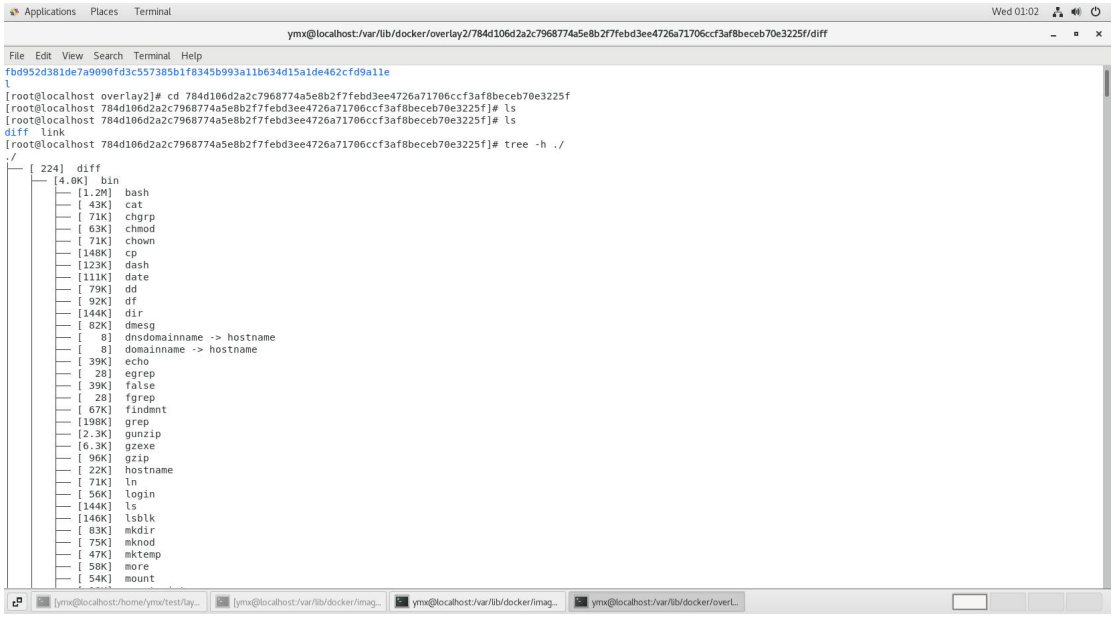
Image JSON 是对镜像整体的说明信息，创建人、创建时间等，其中还包括了对 Layer 的索引。

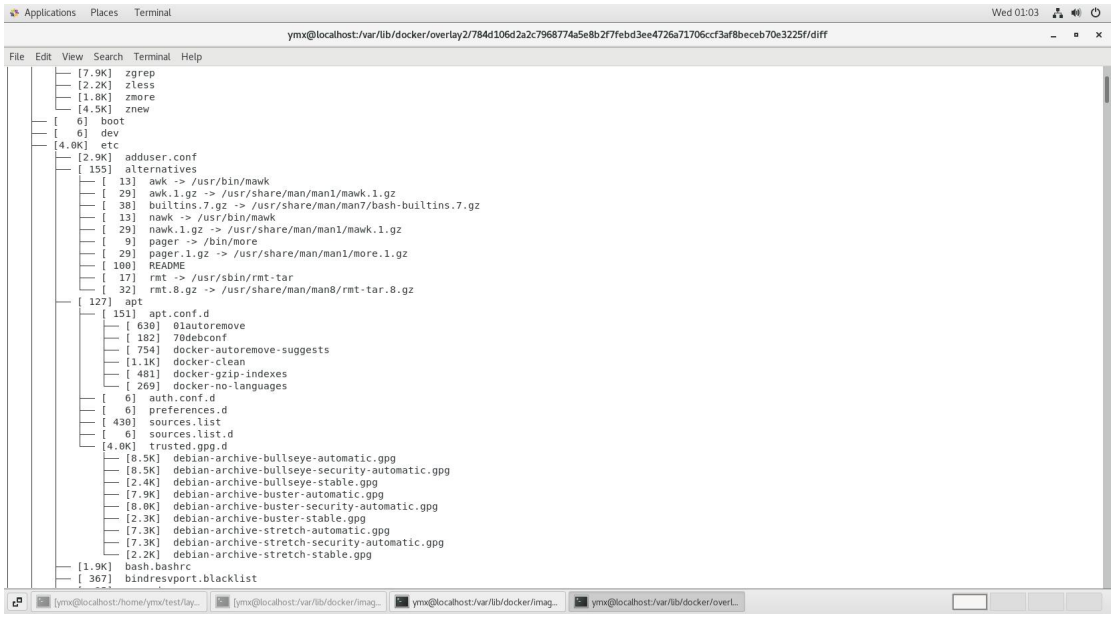
```
{
    "created": "2015-10-31T22:22:56.015925234Z",
    "author": "Alyssa P. Hacker &ltalyspdev@example.com&gt",
    "architecture": "amd64",
    "os": "linux",
    "config": {
        "User": "alice",
        "Memory": 2048,
        "MemorySwap": 4096,
        "CpuShares": 8,
        "ExposedPorts": {
            "8080/tcp": {}
        },
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
            "FOO=docker_is_a_really",
            "BAR=great_tool_you_know"
        ],
        "Entrypoint": [
            "/bin/my-app-binary"
        ],
        "Cmd": [
            "--foreground",
            "--config",
            "/etc/my-app.d/default.cfg"
        ],
        "Volumes": {
            "/var/job-result-data": {},
            "/var/log/my-app-logs": {},
        },
        "WorkingDir": "/home/alice"
    },
    "rootfs": {
      "diff_ids": [
        "sha256:c6f988f4874bb0add23a778f753c65efe992244e148a1d2ec2a8b664fb66bbd1",
        "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef"
      ],
      "type": "layers"
    },
    "history": [
      {
        "created": "2015-10-31T22:22:54.690851953Z",
        "created_by": "/bin/sh -c #(nop) ADD file:a3bc1e842b69636f9df5256c49c5374fb4eef1e281fe3f282c65fb853ee171c5 in /"
      },
      {
        "created": "2015-10-31T22:22:55.613815829Z",
        "created_by": "/bin/sh -c #(nop) CMD [\"sh\"]",
        "empty_layer": true
      }
    ]
}
```


Layer DiffID、Layer ChainID、ImageID 分别是指代 Layer 和 Image 的编号，其中 Layer DiffID 指代某一个 Layer 而 Layer ChainID 指代一组 Layer。

将一个镜像拉取到本地后，我们很容易通过 docker images 命令获取镜像的 ID 值。Docker 的文件系统是将索引与数据分开存储的，我们可以根据 ImageID 获取 LayerID，再根据 LayerID 最终找到 Layer 中存储的数据。

本文中以 Clair 检测 python：latest 容器为例，该 python 镜像中的一个 Layer 存储的数据如图：





通过 imageID 获得 layer 数据的具体方法可以参考阅读 https://www.keepnight.com/archives/298/#menu_index_5

Part2 Clair 应用结构及检测原理

本文中试验部署的 Clair 应用由 clair-scanner 客户端、Clair 服务端及调用的底层模块 ClairCore 构成。

Clair-scanner 作为客户端，采用 cli 架构，主要实现命令行交互、组织检测报告格式、与服务端通信的功能。

Clair 服务端调用 ClairCore 核心模块，通过 LibIndex 包实现从镜像中解析

出组件，再通过 LibVuln 包实现组件与漏洞的对应，并且 LibVuln 包还可以实现漏洞库的自动更新。

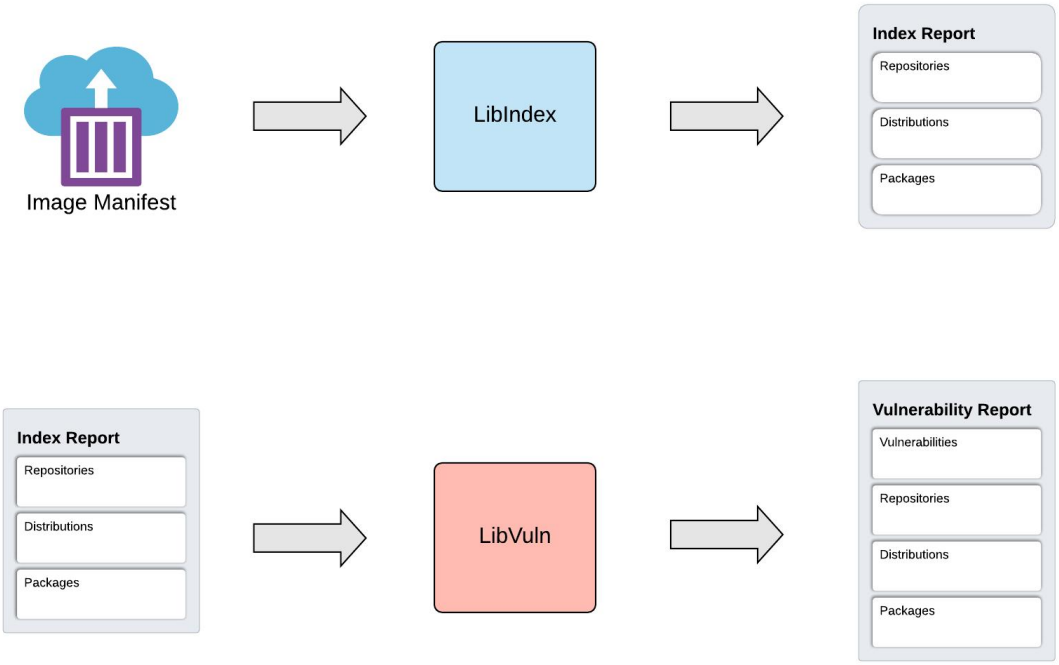


图 2-1 LibIndex 模块与 LibVuln 模块原理示意图

2.1 clair-scanner 功能实现

Clair-scanner 作为客户端，主要实现两个功能。第一，构建 HTTP 客户端向服务端发送待检测 Layer 的索引信息。第二，接收服务端的返回信息并解析成检测报告的格式返回给用户。主干函数为定义在 scanner.go 中的 scan 函数。

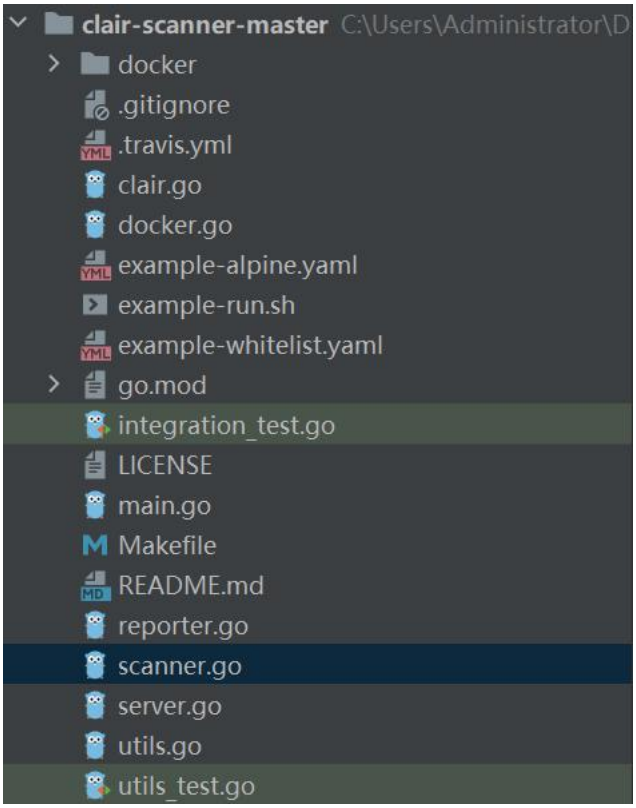


图 2-1-1 clair-scanner 项目目录结构

```
// scan orchestrates the scanning process of an image
func scan(config scannerConfig) []string {
    //Create a temporary folder where the docker image layers are going to be stored
    tmpPath := createTmpPath(tmpPrefix) ①
    defer os.RemoveAll(tmpPath)

    saveDockerImage(config.imageName, tmpPath) ②
    layerIds := getImageLayerIds(tmpPath)

    //Start a server that can serve Docker image layers to Clair
    server := httpFileServer(tmpPath)
    defer server.Shutdown(context.TODO()) ③

    //Analyze the layers
    analyzeLayers(layerIds, config.clairURL, config.scannerIP)
    vulnerabilities := getVulnerabilities(config, layerIds) ④

    if vulnerabilities == nil {
        return nil // exit when no features
    }

    ⑤
    //Check vulnerabilities against whitelist and report
    unapproved := checkForUnapprovedVulnerabilities(config.imageName, vulnerabilities, config.whitelist, c
    ⑥
    // Report vulnerabilities
    reportToConsole(config.imageName, vulnerabilities, unapproved, config.reportAll, config.quiet)
```

图 2-1-2 scanner.go 中 scan 函数

①createTmpPath 定义在 utils.go 中，创建一个临时路径用于存储镜像 layer 文件。

②saveDockerImage 定义在 docker.go 中，将镜像存储到临时路径。

getImageLayerIds 定义在 docker.go 中，从表示镜像的 manifest.json 文件中解析出 Layer ID。

③httpFileServer 定义在 server.go 中，构建一个 HTTP 客户端。

④analyzeLayers 定义在 clair.go 中，将待检测的 LayerID 和 Clair 服务端 url 及 IP 作为参数，通过③中构建的 HTTP client 发送给 Clair 服务端，对 Layer 进行检测。

getVulnerbilities 定义在 clair.go 中，根据 LayerID，从 HTTP client 客户端中解析出漏洞信息。

⑤checkForUnapprovedVulnerabilities 定义在 scanner.go 中，根据用户自定义的漏洞白名单对④中漏洞信息进行筛选。

⑥reportToConsole 定义在 reporter.go 中，将筛选过后的漏洞信息组织成检测报告的形式，打印在屏幕上。

2.2 ClairCore LibIndex 模块实现镜像中组件检索

如 Part1 中介绍，Layer 是镜像（Image）的基本组成单位。ClairCore 中为表示 Layer 定义了结构体，并且 Layer 结构体又作为 Manifest 结构体的属性。一个 Manifest 结构体即可表示一个镜像。

clair-main - C:\Users\Administrator\go\pkg\mod\github.com\quay\claircore@v1.4.7\layer.go

```go
// Layer is a container image filesystem layer. Layers are stacked
// on top of each other to comprise the final filesystem of the container image.
type Layer struct {
    // Hash is a content addressable hash uniqely identifying this layer.
    // Libindex will treat layers with this same hash as identical.
    Hash    Digest              `json:"hash"`
    URI     string              `json:"uri"`
    Headers map[string][]string `json:"headers"`

    // path to local file containing uncompressed tar archive of the layer's content
    localPath string
}
```

图 2-2-1 Layer 结构体定义

如图 2-2-1 所示，localPath 属性中为未压缩的 Layer 内容的原始数据，其中包含我们需要的组件的名称及版本等信息。

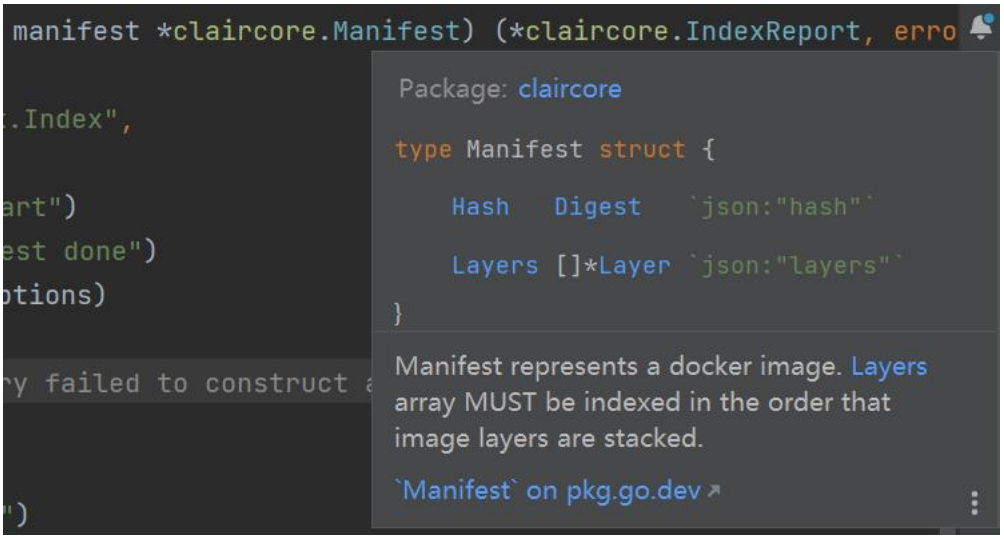


图 2-2-2 Manifest 结构体定义，可见 Layer 为其属性

在 ClairCore 的 LibIndex 模块下，有 Index 函数，将 Manifest 作为参数，最终输出 IndexReport。IndexReport 以 Json 格式组织，其中包含了镜像的标识码 manifest_hash 和镜像中包含的组件 packages 的信息。IndexReport 将作为下一个阶段的输入，传入 LibVuln 模块以检索出组件对应的漏洞。

图 2-2-3 IndexReport 结构

clair-main - ...\Administrator\go\pkg\mod\github.com\quay\claircore@v1.4.7\libindex\libindex.go

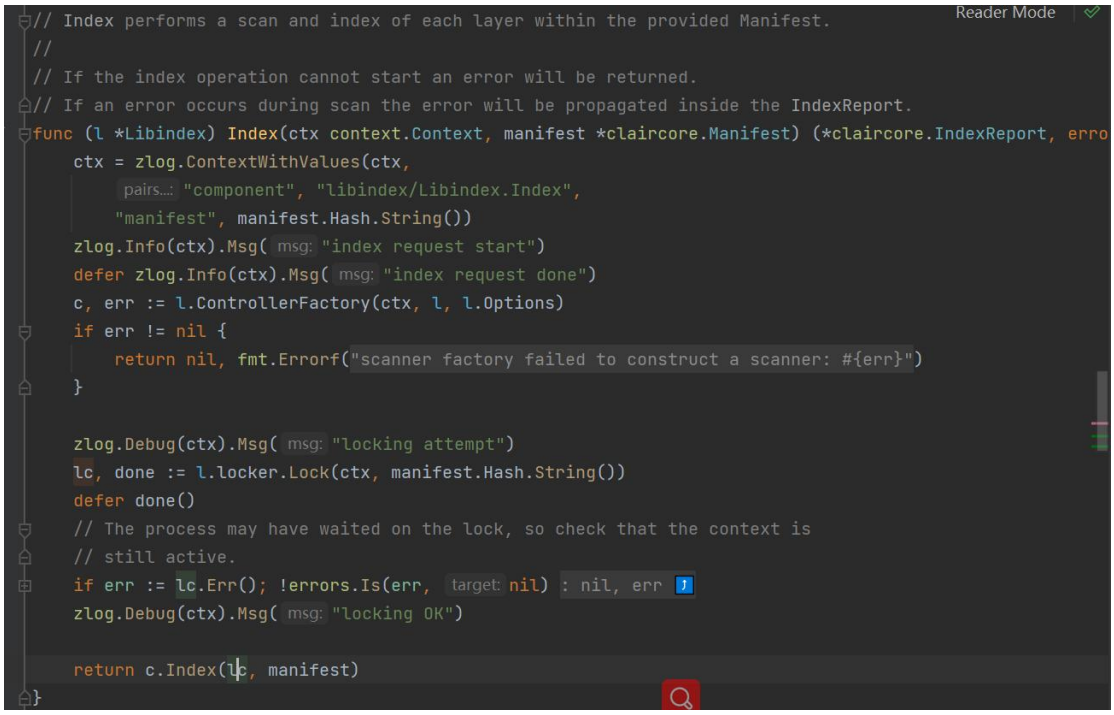


图 2-2-4 libindex.go 文件中的 Index 函数

函数在返回值中调用 controller 模块中的 Index 函数。Controller 模块中的 index 函数同样将 manifest 作为参数，最终返回 controller 结构体中的 report 属性。Report 属性中包含扫描结果。

图 2-2-5 Controller 模块中的 Index 函数，返回 Controller 结构体中的 report 属性

```go
// Controller is a control structure for scanning a manifest.
//
// Controller is implemented as an FSM.
type Controller struct {
    // holds dependencies for a indexer.controller
    *indexer.Opts
    // the manifest this controller is working on. populated on Scan() call
    manifest *claircore.Manifest
    // the result of this scan. each stateFunc manipulates this field.
    report *claircore.IndexReport
    // a fatal error halting the scanning process
    err error
    // the current state of the controller
    currentState State
}
```

图 2-2-6 Controller 结构体定义

2.3 ClairCore LibVuln 数据结构定义

ClairCore 的 LibVuln 模块下包含 Libvuln 结构体的定义，由其定义可知 Libvuln 通过 driver.Matcher 实现组件与漏洞的对应，通过 updates.Manager 实现漏洞库的更新。

```go
// Libvuln exports methods for scanning an IndexReport and created
// a VulnerabilityReport.
//
// Libvuln also runs background updaters which keep the vulnerability
// database consistent.
type Libvuln struct {
    store           datastore.MatcherStore
    locker          LockSource
    pool            *pgxpool.Pool
    matchers        []driver.Matcher
    enrichers       []driver.Enricher
    updateRetention int
    updaters        *updates.Manager
}
```

图 2-3-1 Libvuln 结构体定义

2.4 ClairCore LibVuln 实现组件与漏洞的对应

在 driver\matcher.go 中可以找到 Matcher 接口的定义，在接口中定义了 Vulnerable 函数，它将 IndexRecord 和 Vulnerability 作为参数，并返回一个

布尔值。IndexRecord 是组成 IndexReport 的单位，其中包含一个组件的信息。Vulnerability 包含一个漏洞信息。返回的布尔值表示该组件是否有该漏洞。

clair-main - ...\go\pkg\mod\github.com\quay\claircore@v1.4.7\libvuln\driver\matcher.go

```go
// Matcher is an interface which a Controller uses to query the vulnstore for vulnerabilities.
type Matcher interface {
    // a unique name for the matcher
    Name() string
    // Filter informs the Controller if the implemented Matcher is interested in the provided IndexRecord.
    Filter(record *claircore.IndexRecord) bool
    // Query informs the Controller how it should match packages with vulnerabilities.
    // All conditions are logical AND'd together.
    Query() []MatchConstraint
    // Vulnerable informs the Controller if the given package is affected by the given vulnerability.
    // for example checking the "FixedInVersion" field.
    Vulnerable(ctx context.Context, record *claircore.IndexRecord, vuln *claircore.Vulnerability) (bool, e
}
```

图 2-4-1 matcher.go 中 Matcher 接口的定义

通过对 Vulnerable 函数的实现可以完成组件与漏洞的映射。

Clair 的组件漏洞主要有以下几个来源：

- Alpine
- Aws
- Debian
- Oracle
- Photon
- Python
- Rhel
- Suse
- Ubuntu

根据不同的来源，ClairCore 分别实现了 Vulnerable 方法。

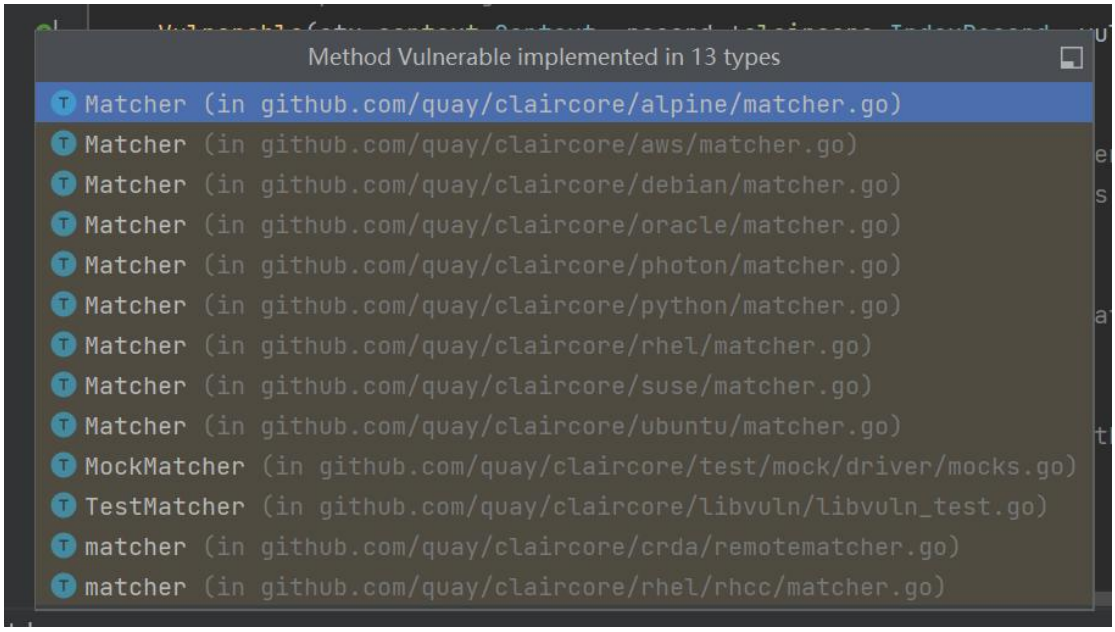


图 2-4-2 对 Vulnerable 函数接口的不同实现

以 alpine 为例。根据从 record 参数中获取的组件版本与漏洞修复的组件版本比较，判断当前版本的组件是否有风险，并返回一个布尔值。

```
// Vulnerable implements driver.Matcher.
func (*Matcher) Vulnerable(ctx context.Context, record *claircore.IndexRecord, vuln *claircore.Vulnerability
    v1, err := version.NewVersion(record.Package.Version)
    if err != nil { return false, nil }

    v2, err := version.NewVersion(vuln.FixedInVersion)
    if err != nil { return false, nil }

    if vuln.FixedInVersion == "" { return true, nil }

    if vuln.FixedInVersion == "0" { return false, nil }

    if v1.LessThan(v2) { return true, nil }

    return false, nil
}
```

图 2-4-3 alpine 包中对 Vulnerable 接口的实现

根据返回的布尔值，LibVuln 判断当前组件是否存在该漏洞，并将漏洞信息以 Json 格式拼接在 IndexReport 后形成 VulnerabilityReport 返回给客户端。由客户端解析生成检测报告。

图 2-4-4 LibVuln 模块生成的 JSON 格式漏洞信息

2.5 ClairCore LibVuln 实现漏洞库的更新

在 Libvuln 模块中有 Updater 接口及它的两个属性 Fetcher、Parser 的接口定义。如 2.4 中提到，对不同的漏洞来源，Clair 分别实现了 Fetcher、Parser。

clair-main - ...\go\pkg\mod\github.com\quay\claircore@v1.4.7\libvuln\driver\updater.go

```go
// Updater is an aggregate interface combining the method set of a Fetcher and a Parser
// and forces a Name() to be provided
type Updater interface {
    Name() string
    Fetcher
    Parser
}
```

图 2-5-1 Updater 接口的定义

以 aws 为例，Fetch 方法根据参数中的 Fingerprint（标识一个漏洞库）返回一个 io 流。Parse 方法读取 io 流中的数据解析成 Vulnerability 的格式更新到漏洞库中。

```go
func (u *Updater) Fetch(ctx context.Context, fingerprint driver.Fingerprint) (io.ReadCloser, driver.Fingerp
    ctx = zlog.ContextWithValues(ctx, pairs...: "component", "aws/Updater.Fetch")
    if u.c == http.DefaultClient { // OK: checking for log purposes
        zlog.Warn(ctx).Msg( msg: "DefaultClient used, this is almost certainly wrong")
    }
    client, err := NewClient(ctx, u.c, u.release)
    if err != nil { return nil, "", fmt.Errorf("failed to create client: %v", err) }

    tctx, cancel := context.WithTimeout(ctx, defaultOpTimeout)
    defer cancel()
    repoMD, err := client.RepoMD(tctx)
    if err != nil { return nil, "", fmt.Errorf("failed to retrieve repo metadata: %v", err) }
```

图 2-5-2　aws 包中的 Fetch 方法

```go
func (u *Updater) Parse(ctx context.Context, contents io.ReadCloser) ([]*claircore.Vulnerability, error) {
    var updates alas.Updates
    dec := xml.NewDecoder(contents)
    dec.CharsetReader = xmlutil.CharsetReader
    if err := dec.Decode(&updates); err != nil { return nil, fmt.Errorf("failed to unmarshal updates xml: %v", err
    dist := releaseToDist(u.release)

    vulns := []*claircore.Vulnerability{}
    for _, update := range updates.Updates {
        issued, err := time.Parse( layout: "2006-01-02 15:04", update.Issued.Date)
        if err != nil { return vulns, err }
        partial := &claircore.Vulnerability{
            Updater:            u.Name(),
            Name:               update.ID,
            Description:        update.Description,
            Issued:             issued,
            Links:              refsToLinks(update),
            Severity:           update.Severity,
            NormalizedSeverity: NormalizeSeverity(update.Severity),
            Dist:               dist,
        }
```

图 2-5-3　aws 包中的 Parse 方法

Updater 的一些配置信息可由 manager 结构提供，包括自动执行的时间间隔 interval。 如果设置了 interval，则每隔设置的时间间隔执行 manager 类的 start 方法，以更新漏洞库。

```go
// Manager oversees the configuration and invocation of vulnstore updaters.
//
// The Manager may be used in a one-shot fashion, configured to run background
// jobs, or both.
type Manager struct {
    // provides run-time updater construction.
    factories map[string]driver.UpdaterSetFactory
    // max in-flight updaters.
    batchSize int
    // update interval used once Manager.Start is invoked, otherwise
    // this field is not used.
    interval time.Duration
    // configs provided to updaters once constructed.
    configs Configs
    // instructs manager to run gc and provides the number of
    // update operations to keep.
    updateRetention int

    locks  LockSource
    client *http.Client
    store  datastore.Updater
}
```

图 2-5-4 Manager 结构体的定义

```go
// Start will run updaters at the given interval.
//
// Start is designed to be ran as a goroutine. Cancel the provided Context
// to end the updater loop.
//
// Start must only be called once between context cancellations.
func (m *Manager) Start(ctx context.Context) error {
    ctx = zlog.ContextWithValues(ctx, pairs...: "component", "libvuln/updates/Manager.Start")

    if m.interval == 0 : fmt.Errorf("manager must be configured with an interval to start") 1

    // perform the initial run
    zlog.Info(ctx).Msg( msg: "starting initial updates")
    err := m.Run(ctx)
    if err != nil {
        zlog.Error(ctx).Err(err).Msg( msg: "errors encountered during updater run")
    }

    // perform run on every tick
    zlog.Info(ctx).Str( key: "interval", m.interval.String()).Msg( msg: "starting background updates")
    t := time.NewTicker(m.interval)
```

图 2-5-5 Manager 类的 start 方法，用于在设置了时间间隔的情况下执行 updaters

Part3 clair-scanner 的部署及使用

Clair-scanner 是由 github 用户 Armin Coralic 开发的 clair 客户端，使用了 cli 框架，通过与本地搭建的 clair-local-scan 和 clair-db 通信完成扫描工

作。

Clair-local-scan 项目中主要包含一个构建 Clair 镜像的 dockerfile，将 Clair 构建部署到本地。

Clair 作为服务端接收 clair-scanner 的请求，调用底层的 ClairCore，实现功能。

Clair-scanner 的优势在于，它在 Clair 基础上添加了用户自定义漏洞白名单的功能。项目地址：https://github.com/arminc/clair-scanner

3.1 CentOS 7 环境下部署 clair-scanner

- 安装 git、golang

  yum install -y git golang

  查看 git、golang 版本

```
[root@localhost clair-scanner]# go version
go version go1.17.12 linux/amd64
[root@localhost clair-scanner]# git --version
git version 1.8.3.1
[root@localhost clair-scanner]#
```

  配置 go 环境变量到/etc/profile 中

```
#Golang environment settings
export GO111MODULE=on
export GOROOT=/usr/local/go  #golang install path
export GOPATH=/home/gopath  #golang workspace
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin    #golang exe files
export GOPROXY=https://goproxy.cn
```

```
[root@localhost clair-scanner]# source /etc/profile
```

  使环境变量生效

- 编译安装 clair-scanner

  # Clone the repo 下载源码

  git clone git@github.com:arminc/clair-scanner.git

  # Build and install 构建和安装

  cd clair-scanner

  make build

  make installLocal

  # Run 试运行

  ./clair-scanner -h

```
[root@localhost clair-scanner]# ./clair-scanner -h

Usage: clair-scanner [OPTIONS] IMAGE

Scan local Docker images for vulnerabilities with Clair

Arguments:
  IMAGE=""     Name of the Docker image to scan

Options:
  -w, --whitelist=""                    Path to the whitelist file
  -t, --threshold="Unknown"             CVE severity threshold. Valid values; 'Defcon1', 'Critical', '
High', 'Medium', 'Low', 'Negligible', 'Unknown'
  -c, --clair="http://127.0.0.1:6060"   Clair URL ($CLAIR_URL)
  --ip="localhost"                      IP address where clair-scanner is running on
  -l, --log=""                          Log to a file
  --all, --reportAll=true               Display all vulnerabilities, even if they are approved
  -r, --report=""                       Report output file, as JSON
  -q, --quiet=false                     Quiets ASCII table output
  --exit-when-no-features=false         Exit with status code 5 when no features are found for a parti
cular image
[root@localhost clair-scanner]#
```

- 拉取镜像

```
[root@localhost clair-scanner]# docker pull arminc/clair-db:latest
Trying to pull repository docker.io/arminc/clair-db ...
latest: Pulling from docker.io/arminc/clair-db
c9b1b535fdd9: Already exists
d1030c456d04: Already exists
d1d0211bbd9a: Already exists
07d0560c0a3f: Already exists
ce7fd4584a5f: Already exists
63eb0325fe1c: Already exists
b67486507716: Already exists
f58de2b85820: Already exists
ca982626dd56: Already exists
d79ba1567dc0: Pull complete
Digest: sha256:7343727174669850899c8c068e489fdf967730b9ab1a1164eacb50e595a084a4
Status: Downloaded newer image for docker.io/arminc/clair-db:latest
[root@localhost clair-scanner]# docker pull arminc/clair-local-scan:latest
Trying to pull repository docker.io/arminc/clair-local-scan ...
latest: Pulling from docker.io/arminc/clair-local-scan
70cdc8479188: Pull complete
53a74cf68280: Pull complete
34eb7ed801b8: Pull complete
Digest: sha256:d3b5354f81252a139b6debc3b0258ca4aa2306ca49e02cd904f0d2d41746a6ab
Status: Downloaded newer image for docker.io/arminc/clair-local-scan:latest
```

- 运行容器

```
[root@localhost clair-scanner]# docker run -p 5432:5432 -d --name clair-db arminc/clair-db:latest
6b2aa649a29f920c704cad22e3411a2100d0974efa17445fc24a85f44f3a374f
```

```
[root@localhost clair-scanner]# docker run -p 6060:6060 --link clair-db:postgres -d --name clair armin
c/clair-local-scan:latest
ff604c17483c3f499c73d30429314d63cac0998e7cd3bf484a4aca1ef34cb804
```

- 查看容器

```
[root@localhost clair-scanner]# docker ps
CONTAINER ID        IMAGE                             COMMAND                  CREATED              STA
TUS                 PORTS                             NAMES
ff604c17483c        arminc/clair-local-scan:latest    "/clair -config=/c..."   13 seconds ago       Up
12 seconds          0.0.0.0:6060->6060/tcp, 6061/tcp  clair
6b2aa649a29f        arminc/clair-db:latest            "docker-entrypoint..."   About a minute ago   Up
About a minute      0.0.0.0:5432->5432/tcp            clair-db
```

- 使用 Clair

  查询 docker0 ip 信息

```
[root@localhost clair-scanner]# ifconfig docker0 | grep inet
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:c0ff:fefb:2f54  prefixlen 64  scopeid 0x20<link>
```

3.2 clair-scanner 的使用

- **直接扫描容器**

```
[root@localhost clair-scanner]#
[root@localhost clair-scanner]#
[root@localhost clair-scanner]# ./clair-scanner --ip 172.17.0.1 python:latest
2022/09/21 19:06:44 [INFO] ▶ Start clair-scanner
2022/09/21 19:07:03 [INFO] ▶ Server listening on port 9279
2022/09/21 19:07:03 [INFO] ▶ Analyzing 360055bafd3c7d97d271aa22d8ebc05d4461b8bde45a6a7999036275d3be7393
2022/09/21 19:07:03 [INFO] ▶ Analyzing 29533d395a7bccddbe0632d76d3c60b372efb4fba678b39a5aebb3875a12f1fe
2022/09/21 19:07:03 [INFO] ▶ Analyzing 3ccbf9b3c8e435d0587fc65171bca63b8b2e71e7a6750f05cbc52ac21ffa805e
2022/09/21 19:07:03 [INFO] ▶ Analyzing 74e6f139551426a654040a5f505236e058a66b9cf63674cceae0458f75a01e24
2022/09/21 19:07:03 [INFO] ▶ Analyzing dbfffca89bc4a891f82b390b1054a0c861d5eec54522cc05b7593f2ea8fc4278
2022/09/21 19:07:03 [INFO] ▶ Analyzing ffa0e6787050b1423db8c4e39a8a3a7f42d4674effb5389c7a3ff3d0cba6aae8
2022/09/21 19:07:03 [INFO] ▶ Analyzing f604a76ee4c2b246cf979b3d6c3a5d11f0728f9044faa5e68fc8fe82c4865434
2022/09/21 19:07:03 [INFO] ▶ Analyzing c470c8c9a999f9b83557b32df3f591bf035fbfb1bb7f928cd677e6b4dff6e0ef
2022/09/21 19:07:03 [INFO] ▶ Analyzing 75030c12c71e2acf79819997f2144318dd095cd23b7b925dabff4a0406d0b665
2022/09/21 19:07:04 [WARN] ▶ Image [python:latest] contains 383 total vulnerabilities
2022/09/21 19:07:04 [ERRO] ▶ Image [python:latest] contains 383 unapproved vulnerabilities
+------------+-------------------------+--------------+---------------------+-------------------------------------------------------------------------------+
| STATUS     | CVE SEVERITY            | PACKAGE NAME | PACKAGE VERSION     | CVE DESCRIPTION                                                               |
+------------+-------------------------+--------------+---------------------+-------------------------------------------------------------------------------+
| Unapproved | Critical CVE-2019-19814 | linux        | 5.10.136-1          | In the Linux kernel 5.0.21, mounting a crafted f2fs                           |
|            |                         |              |                     | filesystem image can cause __remove_dirty_segment                             |
|            |                         |              |                     | slab-out-of-bounds write access because an                                    |
|            |                         |              |                     | array is bounded by the number of dirty types                                 |
|            |                         |              |                     | (8) but the array index can exceed this.                                      |
|            |                         |              |                     | https://security-tracker.debian.org/tracker/CVE-2019-19814                    |
+------------+-------------------------+--------------+---------------------+-------------------------------------------------------------------------------+
| Unapproved | Critical CVE-2015-20107 | python3.9    | 3.9.2-1             | In Python (aka CPython) through 3.10.4, the mailcap module                    |
|            |                         |              |                     | does not add escape characters into commands discovered                       |
|            |                         |              |                     | in the system mailcap file. This may allow attackers                          |
|            |                         |              |                     | to inject shell commands into applications that call                          |
|            |                         |              |                     | mailcap.findmatch with untrusted input (if they lack                          |
|            |                         |              |                     | validation of user-provided filenames or arguments).                          |
|            |                         |              |                     | https://security-tracker.debian.org/tracker/CVE-2015-20107                    |
+------------+-------------------------+--------------+---------------------+-------------------------------------------------------------------------------+
| Unapproved | High CVE-2021-29921     | python3.9    | 3.9.2-1             | In Python before 3.9.5, the ipaddress library mishandles                      |
```

此处配置的 IP 为 Clair 服务端的 IP。

- **使用白名单扫描容器**

```
generalwhitelist:
  CVE-2017-6055: XML
  CVE-2017-5586: OpenText
  CVE-2016-3709: libxml2
images:
  ubuntu:
    CVE-2017-5230: Java
    CVE-2017-5230: XSX
  alpine:
    CVE-2017-3261: SE
~
```

```
clair-scanner  docker.go  example-run.sh      go.mod              integration_test.go  main.go  python.report  redis.report  scanner.go   utils.go
[root@localhost clair-scanner]# ./clair-scanner -w example-whitelist.yaml --ip 172.17.0.1 python:latest
2022/09/21 19:21:14 [INFO] ▶ Start clair-scanner
2022/09/21 19:21:31 [INFO] ▶ Server listening on port 9279
2022/09/21 19:21:31 [INFO] ▶ Analyzing 360055bafd3c7d97d271aa22d8ebc05d4461b8bde45a6a7999036275d3be7393
2022/09/21 19:21:31 [INFO] ▶ Analyzing 29533d395a7bccddbe0632d76d3c60b372efb4fba678b39a5aebb3875a12f1fe
2022/09/21 19:21:31 [INFO] ▶ Analyzing 3ccbf9b3c8e435d0587fc65171bca63b8b2e71e7a6750f05cbc52ac21ffa805e
2022/09/21 19:21:31 [INFO] ▶ Analyzing 74e6f139551426a654040a5f505236e058a66b9cf63674cceae0458f75a01e24
2022/09/21 19:21:31 [INFO] ▶ Analyzing dbfffca89bc4a891f82b390b1054a0c861d5eec54522cc05b7593f2ea8fc4278
2022/09/21 19:21:31 [INFO] ▶ Analyzing ffa0e6787050b1423db8c4e39a8a3a7f42d4674effb5389c7a3ff3d0cba6aae8
2022/09/21 19:21:31 [INFO] ▶ Analyzing f604a76ee4c2b246cf979b3d6c3a5d11f0728f9044faa5e68fc8fe82c4865434
2022/09/21 19:21:31 [INFO] ▶ Analyzing c470c8c9a999f9b83557b32df3f591bf035fbfb1bb7f928cd677e6b4dff6e0ef
2022/09/21 19:21:31 [INFO] ▶ Analyzing 75030c12c71e2acf79819997f2144318dd095cd23b7b925dabff4a0406d0b665
2022/09/21 19:21:31 [WARN] ▶ Image [python:latest] contains 383 total vulnerabilities
2022/09/21 19:21:31 [ERRO] ▶ Image [python:latest] contains 382 unapproved vulnerabilities
+------------+-------------------------+--------------+---------------------+-------------------------------------------------------------------------------+
| STATUS     | CVE SEVERITY            | PACKAGE NAME | PACKAGE VERSION     | CVE DESCRIPTION                                                               |
+------------+-------------------------+--------------+---------------------+-------------------------------------------------------------------------------+
| Unapproved | Critical CVE-2015-20107 | python3.9    | 3.9.2-1             | In Python (aka CPython) through 3.10.4, the mailcap module                    |
|            |                         |              |                     | does not add escape characters into commands discovered                       |
|            |                         |              |                     | in the system mailcap file. This may allow attackers                          |
|            |                         |              |                     | to inject shell commands into applications that call                          |
|            |                         |              |                     | mailcap.findmatch with untrusted input (if they lack                          |
|            |                         |              |                     | validation of user-provided filenames or arguments).                          |
|            |                         |              |                     | https://security-tracker.debian.org/tracker/CVE-2015-20107                    |
+------------+-------------------------+--------------+---------------------+-------------------------------------------------------------------------------+
```

Clair-scanner 的白名单仅根据漏洞编号对漏洞进行剔除。

Part4 Clair-db 数据库结构

Clair 使用 Postgres 数据库存储数据，进入 clair-db 容器后，登陆 postgres 可以看到 clair 数据库的表结构。

```
[root@localhost clair-scanner]# docker exec -it clair-db /bin/bash
bash-5.0# psql -U postgres
psql (11.6)
Type "help" for help.

postgres=# 
```

默认用户名“postgres”登陆。

```
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(3 rows)
```

查看数据库

```
postgres=# \c postgres
You are now connected to database "postgres" as user "postgres".
postgres=#
```

使用数据库

```
postgres=# \dt
                    List of relations
 Schema |                  Name                  | Type  |  Owner
--------+----------------------------------------+-------+----------
 public | feature                                | table | postgres
 public | featureversion                         | table | postgres
 public | keyvalue                               | table | postgres
 public | layer                                  | table | postgres
 public | layer_diff_featureversion              | table | postgres
 public | lock                                   | table | postgres
 public | namespace                              | table | postgres
 public | schema_migrations                      | table | postgres
 public | vulnerability                          | table | postgres
 public | vulnerability_affects_featureversion | table | postgres
 public | vulnerability_fixedin_feature          | table | postgres
 public | vulnerability_notification             | table | postgres
(12 rows)
```

```
postgres=# \ds
                            List of relations
 Schema |                     Name                      |   Type   |  Owner
--------+-----------------------------------------------+----------+----------
 public | feature_id_seq                                | sequence | postgres
 public | featureversion_id_seq                         | sequence | postgres
 public | keyvalue_id_seq                               | sequence | postgres
 public | layer_diff_featureversion_id_seq              | sequence | postgres
 public | layer_id_seq                                  | sequence | postgres
 public | lock_id_seq                                   | sequence | postgres
 public | namespace_id_seq                              | sequence | postgres
 public | vulnerability_affects_featureversion_id_seq | sequence | postgres
 public | vulnerability_fixedin_feature_id_seq          | sequence | postgres
 public | vulnerability_id_seq                          | sequence | postgres
 public | vulnerability_notification_id_seq             | sequence | postgres
(11 rows)
```

其中 Type=table 的是表，Type=sequence 的是序列对象。序列对象是一种单行表，用于记录同名 table 中的数据条数。以 feature_id_sequence 为例：

```
postgres=# select * from feature_id_seq ;
 last_value | log_cnt | is_called
------------+---------+-----------
      42415 |      25 | t
(1 row)
```

Clair-db 中存储主要数据的为以下几张表：

```
postgres=# \d feature
                                  Table "public.feature"
    Column    |          Type          | Collation | Nullable |                Default
--------------+------------------------+-----------+----------+---------------------------------------
 id           | integer                |           | not null | nextval('feature_id_seq'::regclass)
 namespace_id | integer                |           | not null |
 name         | character varying(128) |           | not null |
Indexes:
```

此处 feature 指的就是 IndexReport 中的 package，name 存储的就是 package name，即组件名。同理 featureversion 表中存储的是 package version，即组件版本。

```
postgres=# \d vulnerability
                                    Table "public.vulnerability"
    Column    |           Type           | Collation | Nullable |                   Default
--------------+--------------------------+-----------+----------+-----------------------------------------
 id           | integer                  |           | not null | nextval('vulnerability_id_seq'::regclass)
 namespace_id | integer                  |           | not null |
 name         | character varying(128)   |           | not null |
 description  | text                     |           |          |
 link         | character varying(128)   |           |          |
 severity     | severity                 |           | not null |
 metadata     | text                     |           |          |
 created_at   | timestamp with time zone |           |          |
 deleted_at   | timestamp with time zone |           |          |
```

Vulnerability 表即是 Clair 的漏洞库。其中 name 字段存储漏洞编号（CVE 等），description 字段是对漏洞的文字介绍，link 字段是漏洞的来源网站连接，severity 为漏洞危险等级，metadata 字段存储 CVSS 评分数据。

测试时 Vulnerability 表中共 23w 条数据，表的大小是 135M，平均约每条数据 0.5KB。

各表之间通过外键相互关联，实现检测原理一节中提到的根据 IndexReport 中的 package 名称和版本（存储在 feature 和 featureversion 表中）从漏洞库（vulnerability 表）中检索出被测容器的漏洞信息，最终生成检测报告。检测报告格式如下：

STATUS	CVE SEVERITY	PACKAGE NAME	PACKAGE VERSION	CVE DESCRIPTION
^[[1;31mUnapproved^[[0m	Critical CVE-2019-19814	linux	5.10.136-1	In the Linux kernel 5.0.21, mounting a crafted f2fs filesystem image can cause __remove_dirty_segment slab-out-of-bounds write access because an array is bounded by the number of dirty types (8) but the array index can exceed this. https://security-tracker.debian.org/tracker/CVE-2019-19814
^[[1;31mUnapproved^[[0m	Critical CVE-2015-20107	python3.9	3.9.2-1	In Python (aka CPython) through 3.10.4, the mailcap module does not add escape characters into commands discovered in the system mailcap file. This may allow attackers to inject shell commands into applications that call mailcap.findmatch with untrusted input (if they lack validation of user-provided filenames or arguments). https://security-tracker.debian.org/tracker/CVE-2015-20107