

Gitea 1.4.0未授权远程代码执行漏洞解析

[mss\\*\\*\\*\\*](#) / 2018-07-13 11:46:22 / 浏览数 4314 [技术文章](#) [技术文章 顶\(0\)](#) [踩\(0\)](#)

原文：<https://security.szurek.pl/gitea-1-4-0-unauthenticated-rce.html>

Gitea主页：

<https://gitea.io/en-US/>

引言

该文档也可从[GitHub](#)上下载。

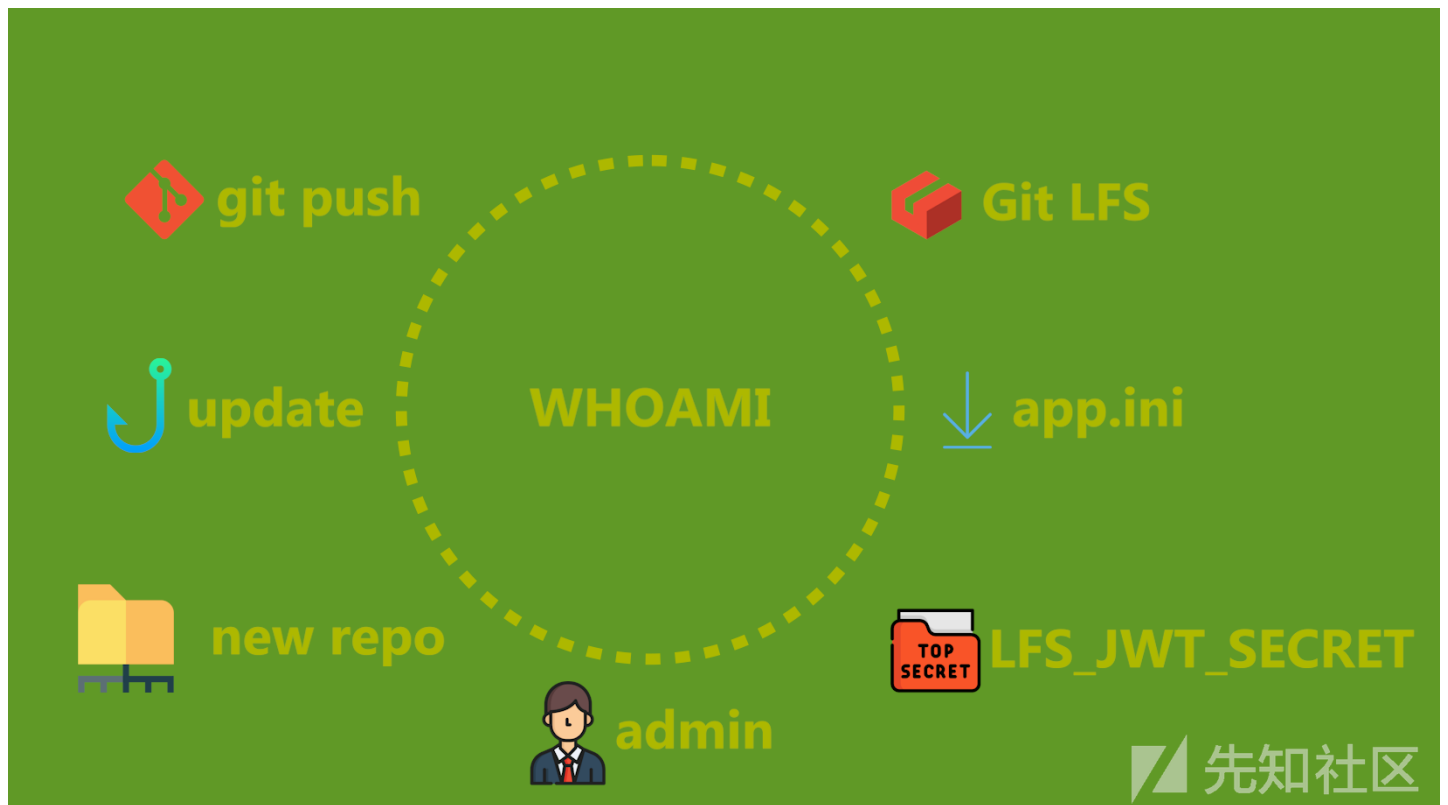
本文是介绍[Gitea](#)和[Gogs](#)内部漏洞的系列文章中的一部分。

您还可以在YouTube上观看解说视频：[Race condition and git hooks vs Gitea server](#)。

Gitea是一个利用Go语言编写的git服务器。

它不仅易于安装，同时还提供了许多有趣的选项。

下面演示的漏洞利用过程涉及多个要素，当它们组合使用时，就可以完全接管该服务器了。



首先，我们需要通过GIT LFS实现中的漏洞来获得app.ini文件的内容。

然后，从这个文件中读取用于为JWT令牌签名的SECRET。

这样一来，我们就能够发送伪造的用户会话文件了。

然后，使用新建的管理员会话创建一个新的存储库。注意，这里需要一个管理员帐户，因为只有管理员才有权修改git hook。

接着，设法把要在服务器上执行的恶意代码放入update hook中。

然后，只要把任意的源代码修改操作推送到存储库，就可以运行恶意代码了。

好了，漏洞利用的原理已经介绍过了，接下来，将详细讨论其中的各个要素。

被遗忘的关键词：return

PostHandler函数的作用是创建新的LFS对象。

表面上看，一切都很正常。例如，如果用户没有相应的权限，则会调用requireAuth函数，并设置相应的WWW-Authenticate头部以及401状态。

但是，当我们深入考察源代码时，就会发现这个函数即使正常使用的话，也会出现问题。

这里的问题在于缺少关键词return：如果有该关键词的话，一旦发生故障，立即终止PostHandler函数。

如果没有关键词return，将执行requireAuth函数，然后，程序会继续执行下一个操作，就这里来说，就是创建LFS对象。

这样的话，我们就可以绕过用户权限的验证机制了。换句话说，现在我们能够为任意的存储库创建任意的LFS对象。

```
func PostHandler(ctx *context.Context) {
    if !setting.LFS.StartServer {
        writeStatus(ctx, 404)
        return
    }

    if !MetaMatcher(ctx.Req) {
        writeStatus(ctx, 400)
        return
    }

    rv := unpack(ctx)

    repository, err := models.GetRepositoryByOwnerAndName(rv.User, rv.Repo)
    if err != nil {
        log.Debug("Could not find repository: %s/%s - %s", rv.User, rv.Repo, err)
        writeStatus(ctx, 404)
        return
    }

    if !authenticate(ctx, repository, rv.Authorization, true) {
        requireAuth(ctx)
        # !!!!! MISSING RETURN HERE
    }

    meta, err := models.NewLFSMetaObject(&models.LFSMetaObject{Oid: rv.Oid, Size: rv.Size, RepositoryID: repository.ID})
    if err != nil {
        writeStatus(ctx, 404)
        return
    }

    ctx.Resp.Header().Set("Content-Type", meta.MediaType)

    sentStatus := 202
    contentStore := &ContentStore{BasePath: setting.LFS.ContentPath}
    if meta.Existing && contentStore.Exists(meta) {
        sentStatus = 200
    }
    ctx.Resp.WriteHeader(sentStatus)

    enc := json.NewEncoder(ctx.Resp)
    enc.Encode(Represent(rv, meta, meta.Existing, true))
    logRequest(ctx.Req, sentStatus)
}
```

## 任意文件读取

---

getContentHandler函数的作用，是根据其Oid从LFS存储库中检索文件的内容。

首先，它会检查当前用户是否有权读取存储库。这就是我们需要使用公共可用存储库的原因——任何用户（甚至没有登录）都可以从中下载任何文件。

然后，使用ContentStore检索文件的路径。

接着，将LFS\_CONTENT\_PATH目录与oid参数连接起来。

然后，名为transformKey的函数将为该文件生成新的路径。

```
func transformKey(key string) string {
    if len(key) < 5 {
        return key
    }

    return filepath.Join(key[0:2], key[2:4], key[4:])
}
```

该路径构建方式是，先取标识符的前两个字符，然后添加一个反斜杠，再取标识符接下来的两个字符，然后又加入一个反斜杠，最后，取标识符的其余部分。

abcdefgh -> ab\cd\efgh

通过用点号替换oid参数，我们可以得到如下结果：

gitea\data\lfs\..\..\custom\conf\app.ini

在Windows平台上，../表示向上移动到父目录。这样的话，我们就能够读取app.ini文件的内容了。

为JWT令牌签名

---

在配置文件中，已经含有LFS\_JWT\_SECRET。

```
APP_NAME = Gitea: Git with a cup of tea
RUN_USER = root
RUN_MODE = prod

[security]
INTERNAL_TOKEN = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
INSTALL_LOCK   = true
SECRET_KEY     = 79jOlo4qSO

[database]
DB_TYPE = sqlite3
HOST     = 127.0.0.1:3306
NAME     = gitea
USER     = gitea
PASSWD   =
SSL_MODE = disable
PATH     = data/gitea.db
```

LFS\_JWT\_SECRET的值就是用于为JWT令牌签名的密钥。将LFS GIT文件发送到服务器时，系统会对这些令牌进行检查。

由于我们知道LFS\_JWT\_SECRET的值，所以，可以随心所欲地使用任意oid将任意文件发送到任意存储库。

接下来，我们将再次使用“4点”技巧来创建新的LFS对象。但是，这次我们将使用sessions目录的路径作为oid。

....data/sessions/1/1/11customsession

这个目录中的某些文件，存放的就是前登录用户的相关信息。

使用Gitea，我们可以向服务器发送一个名为i\_like\_gitea的cookie。

服务器将会检查该目录中是否存在名为cookie的文件。

如果有的话，服务器就会读取存储在会话中的当前用户信息。

之后，就可以使用一个伪造的管理员帐户发送我们自己的会话文件了。

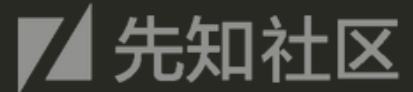
Why？听我细细道来。当我们检查允许在服务器上保存文件的函数时，就会发现它的工作原理与用于下载文件的函数完全相同。

它们之间只有一个不同之处，那就是.tmp字符串将添加到正在创建的文件的名称中。

```

24 // Get takes a Meta object and retrieves the content from the store,
25 // it as an io.Reader. If fromByte > 0, the reader starts from that b
26 func (s *ContentStore) Get(meta *models.LFSMetaObject, fromByte int64) (
27     path := filepath.Join(s.BasePath, transformKey(meta.Oid))
28
29     f, err := os.Open(path)
30     if err != nil {
31         return nil, err
32     }
33     if fromByte > 0 {
34         _, err = f.Seek(fromByte, os.SEEK_CUR)
35     }
36     return f, err
37 }
38
39 // Put takes a Meta object and an io.Reader and writes the content to
40 func (s *ContentStore) Put(meta *models.LFSMetaObject, r io.Reader) (
41     path := filepath.Join(s.BasePath, transformKey(meta.Oid))
42     tmpPath := path + ".tmp"
43
44     dir := filepath.Dir(path)
45     if err := os.MkdirAll(dir, 0750); err != nil {
46         return err
47     }
48

```



对我们这些攻击者而言，这意味着我们可以将文件发送到任何位置。但是，它的扩展名总是.tmp。

竞争条件

不幸的是，我们无法使用自己发送的会话，因为它被立即从服务器中删除。

实际上，关键字defer就是用来负责这一操作的——Put函数一旦完成其操作，就会立即删除其创建的文件。

defer.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // This will execute after put
7     defer fmt.Println("world")
8     // Put function content
9     fmt.Println("hello")
10 }
11
```

hello

world



为了克服这个限制，我们将搬出一种称为竞争条件的法宝。

将POST请求发送到服务器时，Content-Length头部将与待发送的数据一起传递过去。

它的作用是告诉服务器，用户打算传输多少数据。这样的话，服务器就可以知道用户当前处于数据传输的哪个阶段。

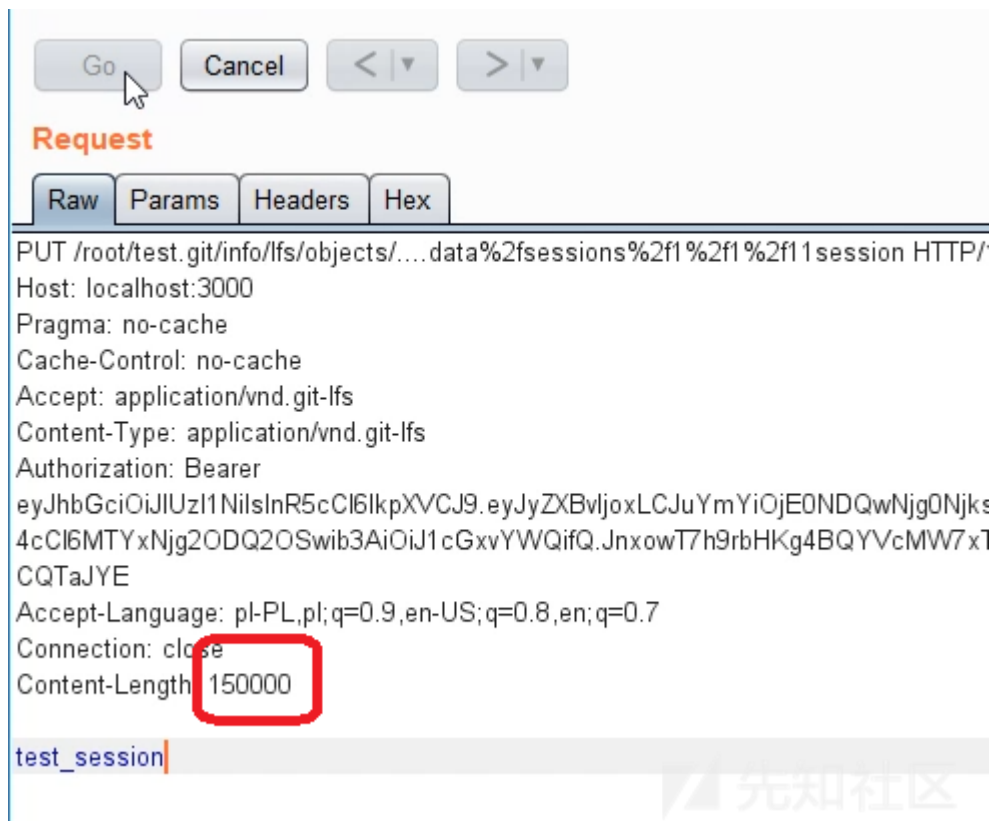
这里的技巧是将这个头部的值设置为一个非常大的数字。

服务器从用户接收数据后，会立即保存到文件中。

但是，该函数会等待传输完成，直到接收到的数据的大小等于Content-Length头部中给出的数字为止。

这样一来，我们的文件就不会被立即删除了。

相反，这样我们就可以争取到几秒钟的时间，如果用于利用我们的会话的话，这些时间就足够了。



Git hooks


现在，我们可以使用伪造的管理员帐户创建一个新的存储库。


接下来，就可以对存储库进行设置了。由于我们的身份是管理员，所以，我们可以访问Git hooks选项。

实际上，所谓hook就是位于各个存储库的.git/hooks目录中的脚本。


在对存储库执行操作时，系统就会执行这些脚本。

例如，在响应git push命令时，Git会执行update脚本。


 root / test


 Unwatch

1


 Star


0

 Code

 Issues

0

 Wiki

 Settings

Options

Collaboration

Webhooks

Git Hooks

Deploy Keys


Git Hooks

If the hook is inactive, sample content will be presented. Leaving content to an empty value will disable this hook.

Hook Name update

Hook Content

```
#!/bin/sh
whoami > objects/info/exploit
```



我们可以把需要执行的命令放到update脚本中，就这里来说，这些命令的运行结果将写入objects/info/exploit文件。

现在，我们只需要将新文件添加到我们的存储库，并通过git push命令将其发送到服务器就行了。

此时，服务器将执行update hook，并将该命令的结果写入名为exploit的文件中。

我们可以通过下载对象来显示该命令的结果：

<http://localhost:3000/root/test/objects/info/exploit>

看到了吧，在没有登录名和密码的情况下，我们照样能够在远程服务器上执行代码。

POC

[https://github.com/kacperszurek/exploits/blob/master/Gitea/gitea\\_lfs\\_rce.py](https://github.com/kacperszurek/exploits/blob/master/Gitea/gitea_lfs_rce.py)

点击收藏 | 0 关注 | 1

[上一篇：SSL Pinning Practice](#) [下一篇：python的lxml库的xxe防御](#)

1. 0 条回复
- 动动手指，沙发就是你的了！

[登录](#) 后跟贴

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)