登录

CVE-2019-11229 Gitea RCE

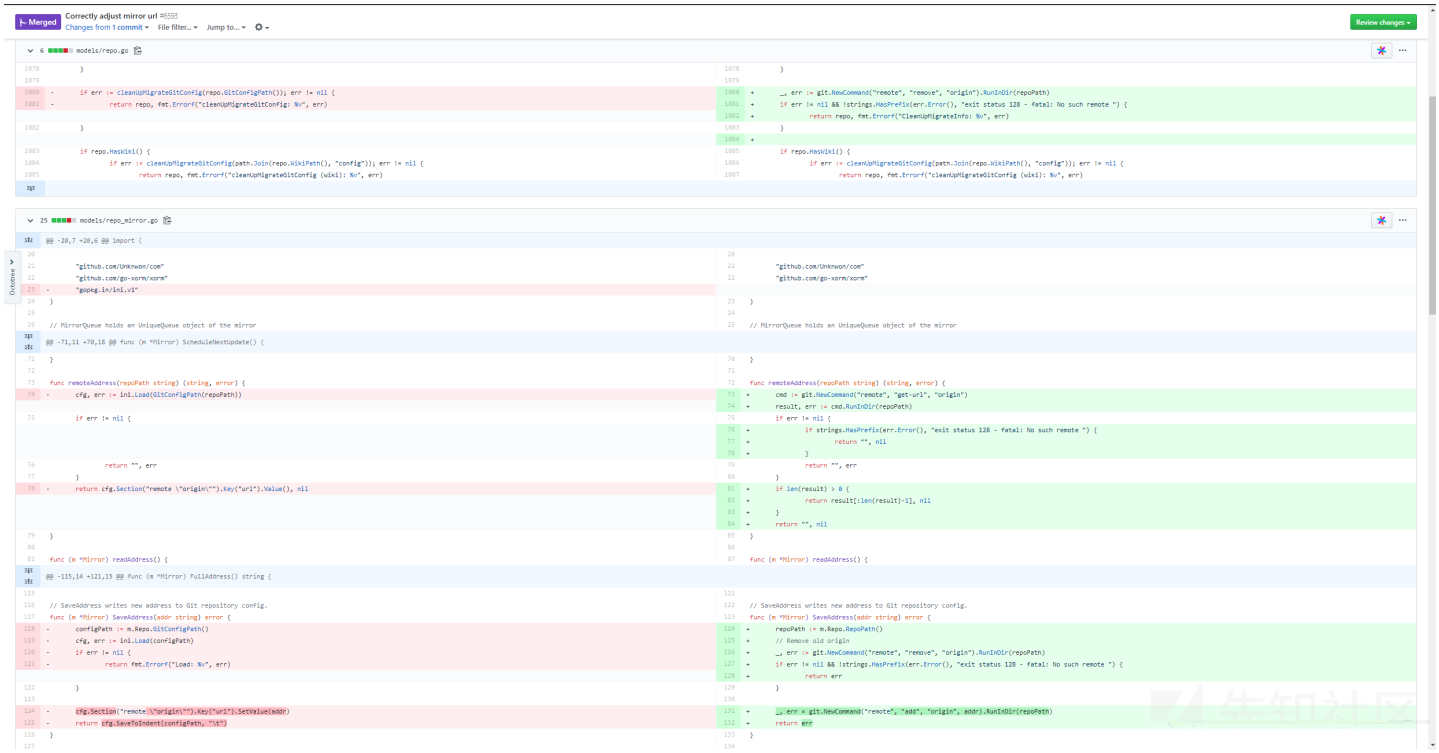
lz1y / 2019-07-26 09:46:00 / 浏览数 5204 安全技术 漏洞分析 顶(1) 踩(0)

首话

在四月十三号的时候，gitea推送了v1.7.6版本，更新日志说到修复了一个安全问题。之前都没有关注，直到这几天看到先知作者群的师傅们讨论了一下这个洞的利用手段，比

挖掘

首先看一看github的diff。
https://github.com/go-gitea/gitea/pull/6593/commits/7ee6794ad5266398c03b1c320804c6c2a6ce34ee

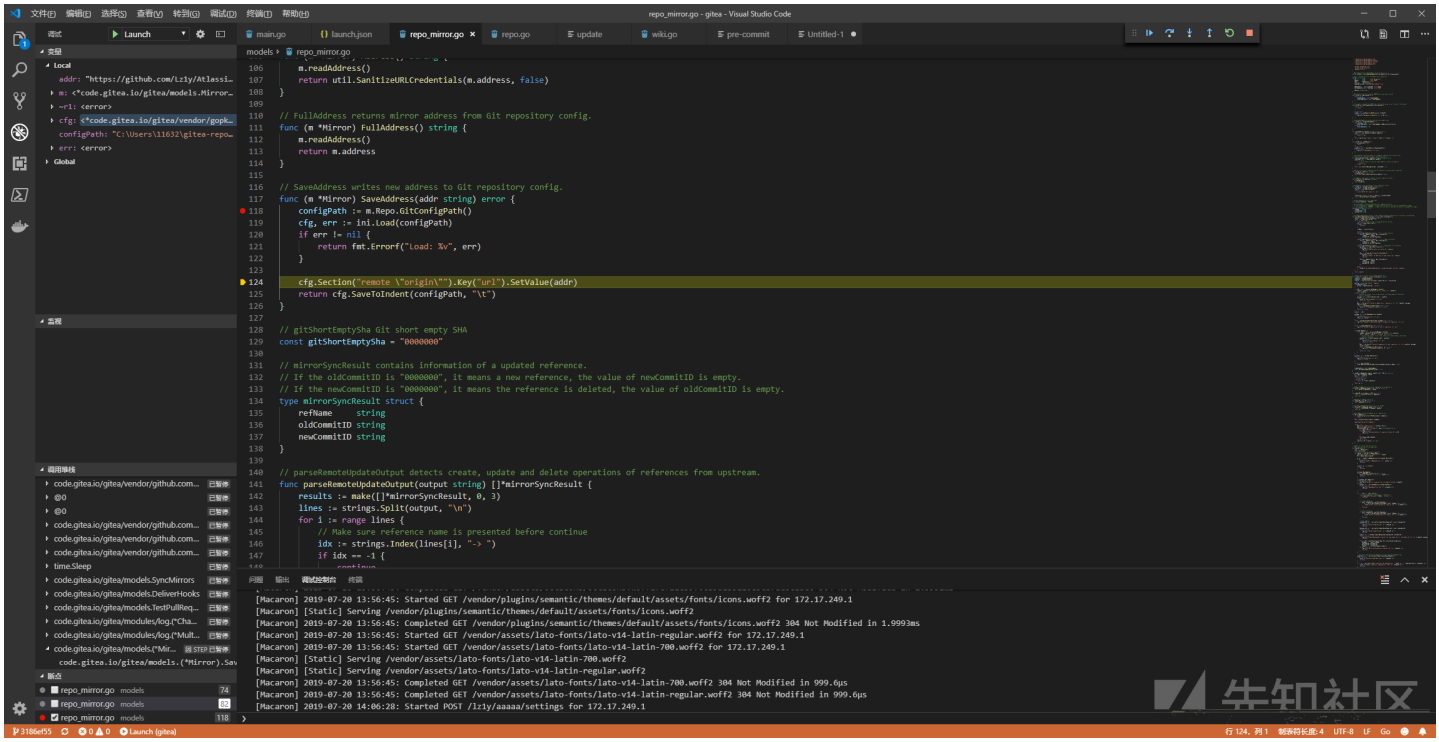


很明显修改点全是关于repo mirror的配置读写，也就是仓库镜像这个功能。其中让我怀疑的点有几个：

1. 删除了"gopkg.in/ini.v1"这个包
2. 添加了镜像地址的格式校验
3. 仅仅是配置读写，并没有命令注入

所以我认为"gopkg.in/ini.v1"这个包存在CRLF漏洞，在LoRexxar的帮助下，果不其然证实了的确存在，所以这个洞利用的第一步就是通过CRLF篡改配置文件。

动态跟一下，看看"gopkg.in/ini.v1"是怎么处理换行的：
根据diff，找到了SaveAddress这个函数，其中存在写配置操作，CRLF也应该是此时发生的。



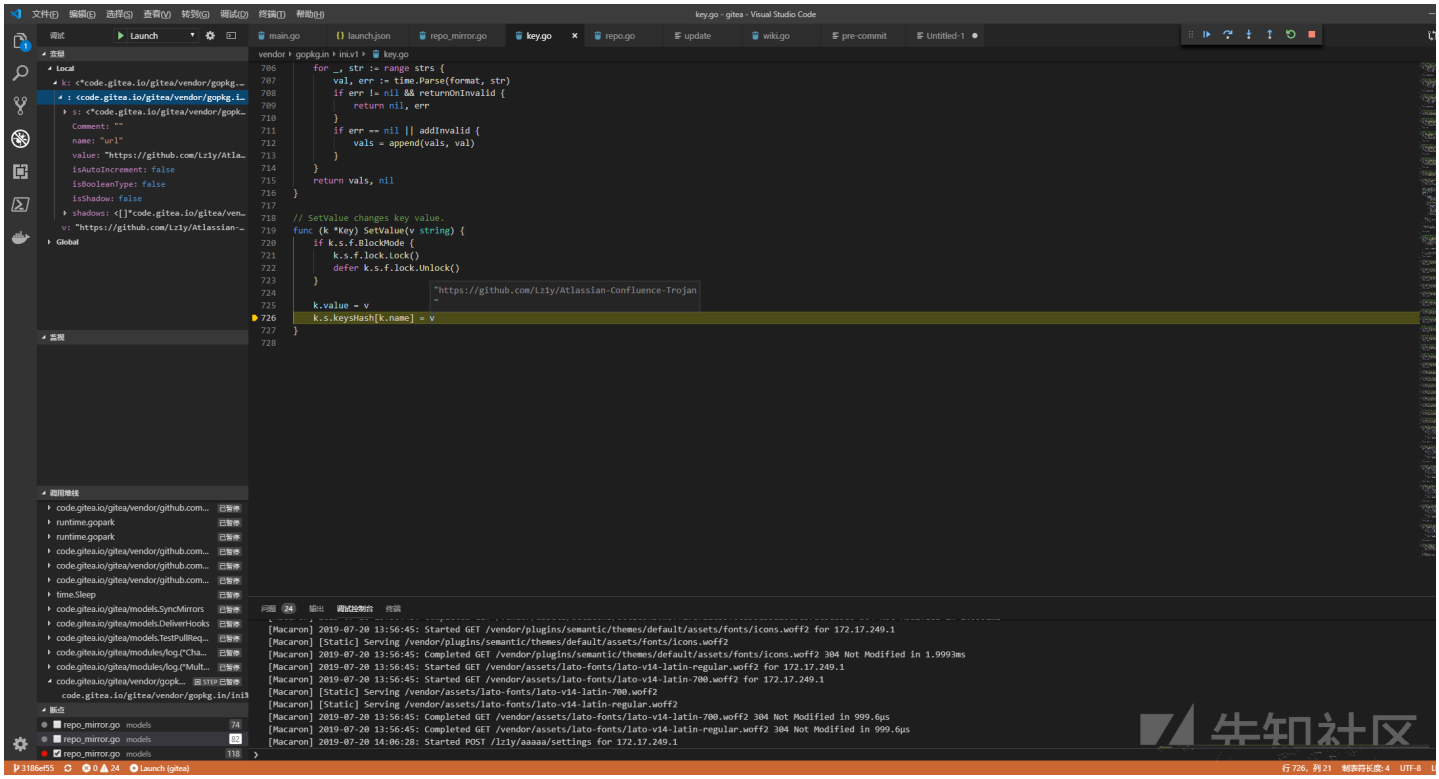
进入SetValue方法

key结构体：

```
type Key struct {
    s               *Section
    Comment         string
    name            string
    value           string
    isAutoIncrement bool
    isBooleanType   bool

    isShadow bool
    shadows  []*Key
}
```

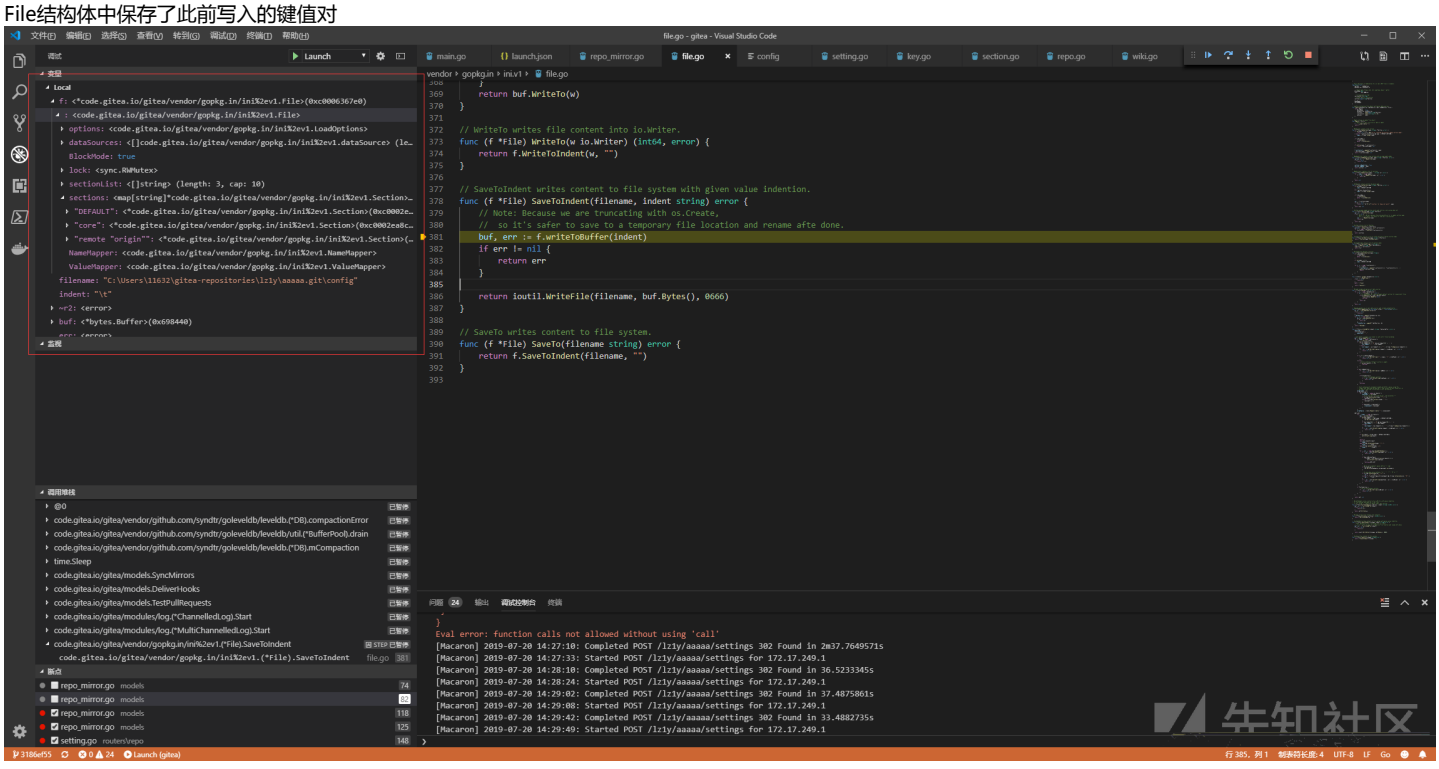
Section结构体:

```
type Section struct {
    f        *File
    Comment  string
    name     string
    keys     map[string]*Key
    keyList  []string
    keysHash map[string]string

    isRawSection bool
    rawBody      string
}
```

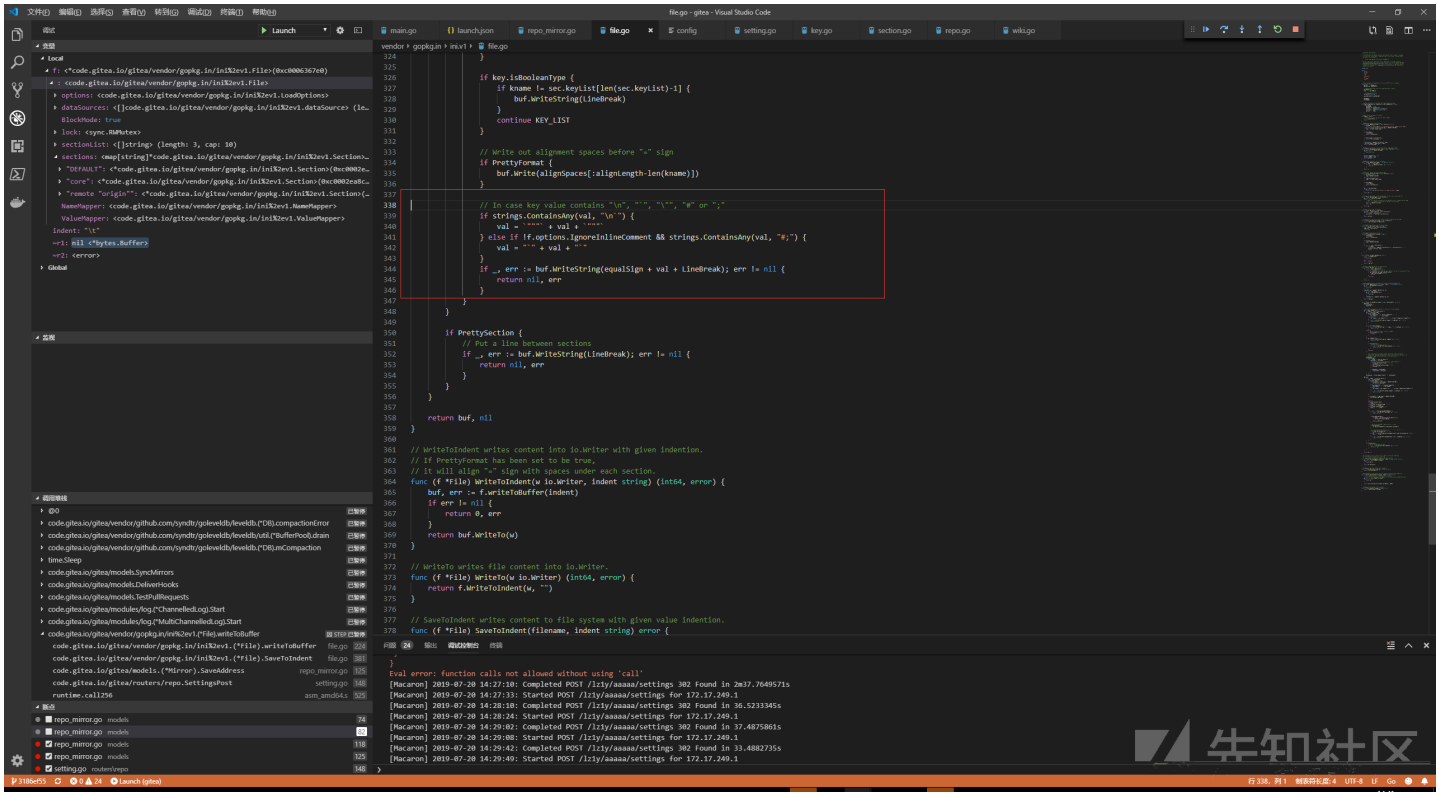


可以看到目前为止只是将我们输入的值写入了键值对，没有做任何处理。

接下来，进入方法SaveToIndent
File结构体中保存了此前写入的键值对

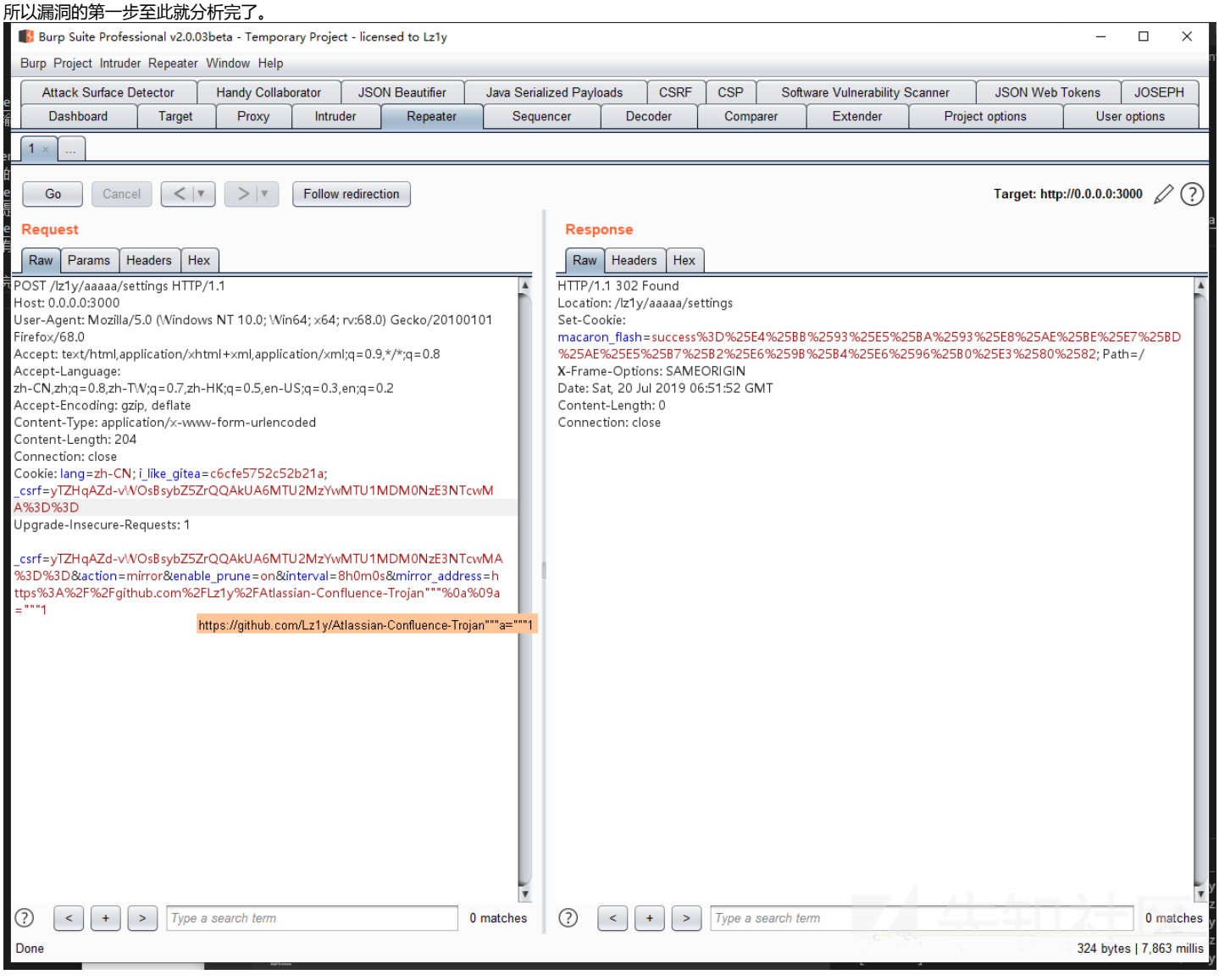


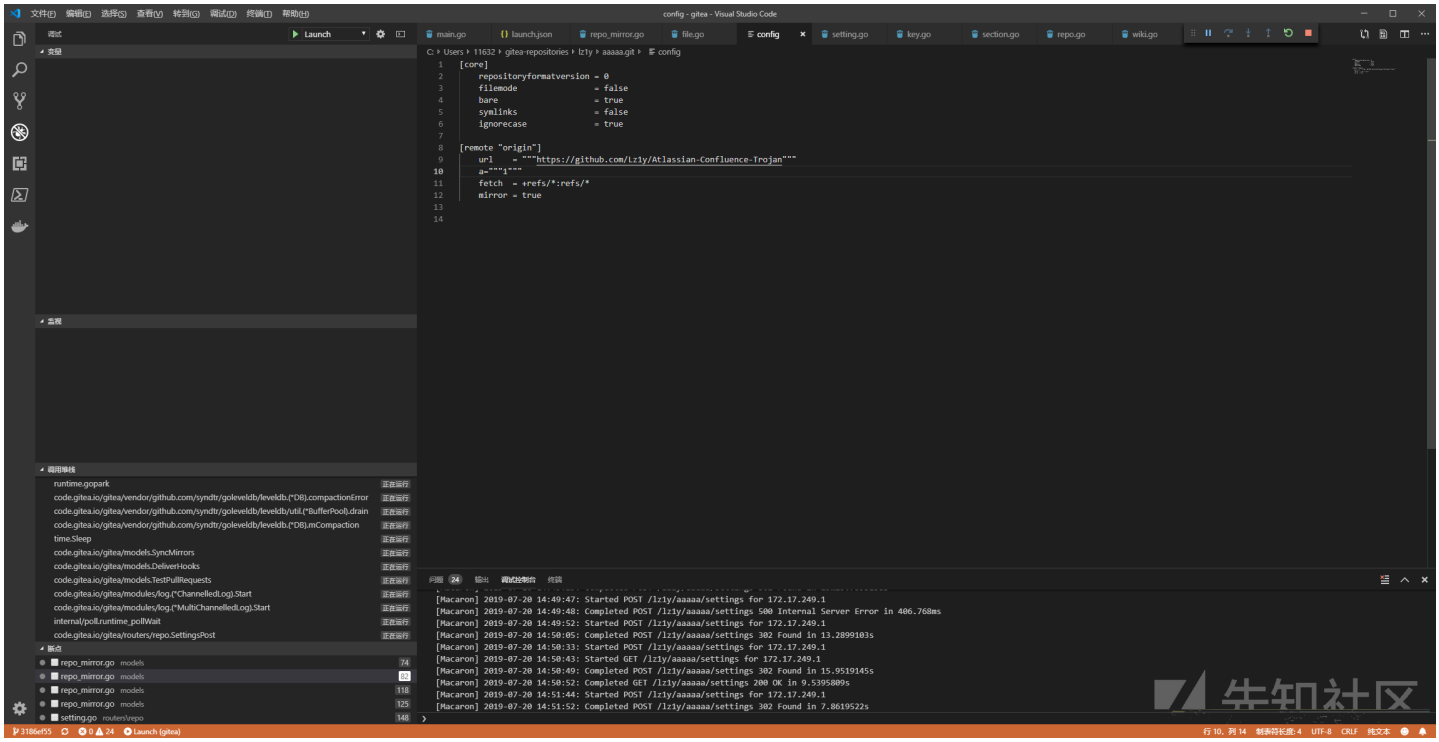
跟进writeToBuffer方法，看是如何将键值对解析并写入缓冲区的



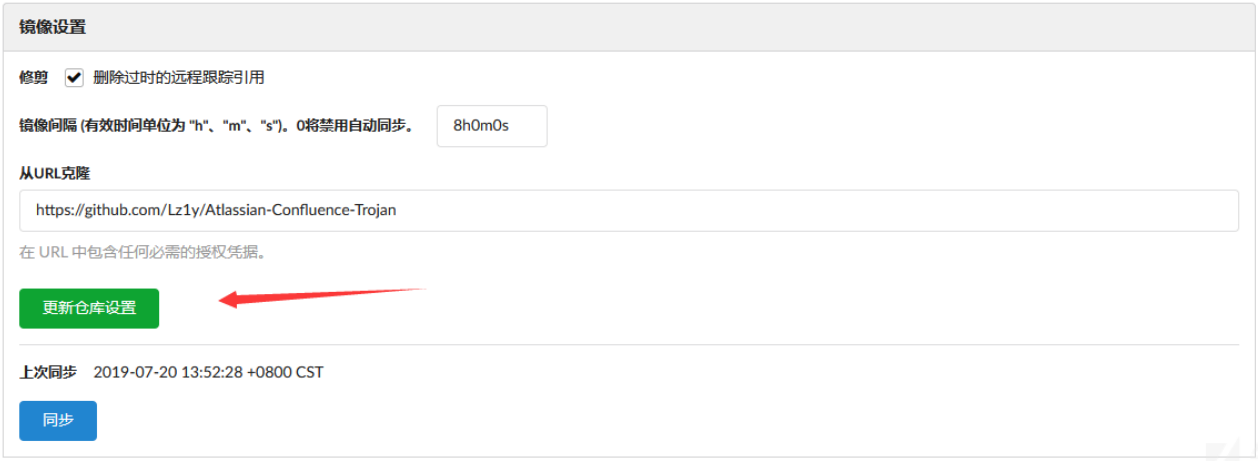
关键点就是这里，判断了是否有换行、注释等符号，有的话就加上"""，此处应该是怕字符串产生歧义，这里通过闭合，就能逃逸了。

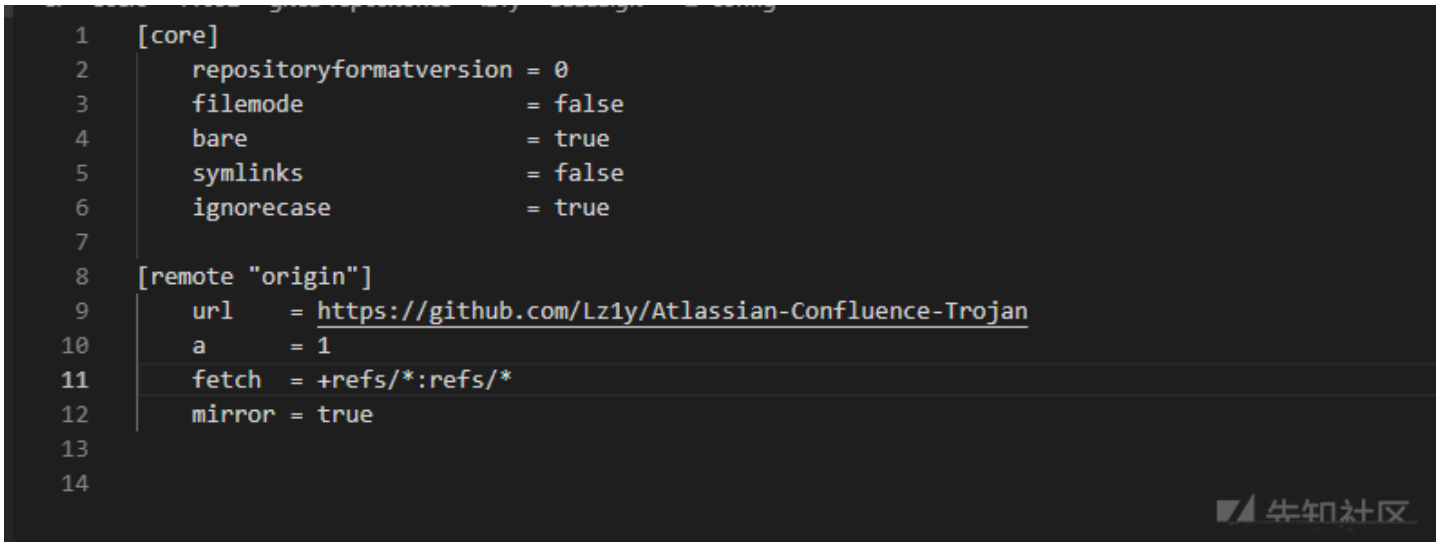
所以漏洞的第一步至此就分析完了。





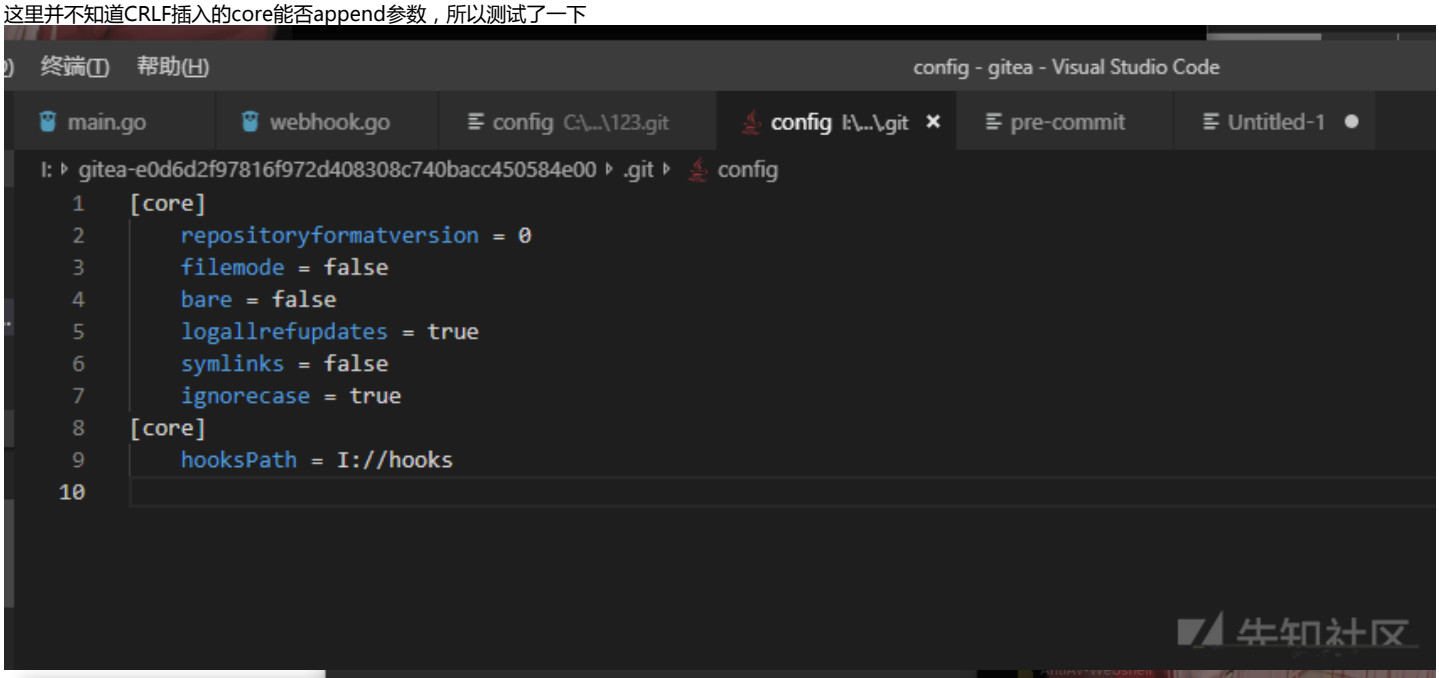
如果觉得不优雅的话，可以刷新后再保存一次，程序逻辑是读取配置为内存中的键值对，此时就过了一次格式优化了，然后再重新写入配置文件。所以会让格式好看很多。

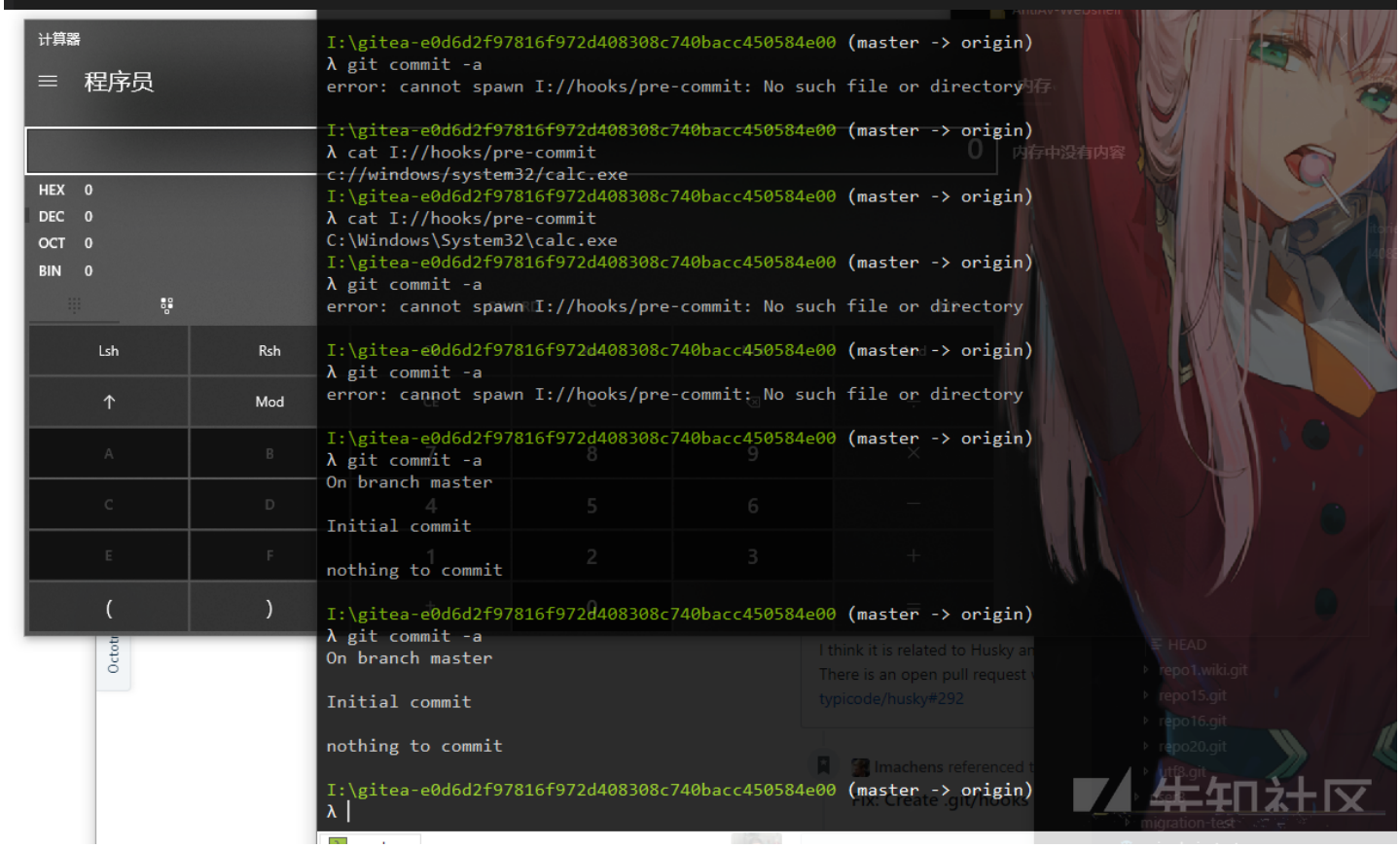




在找到了CRLF写配置文件后，需要做的就是找寻RCE点了。我第一时间想到的就是Hooks，Git及其衍生产品的漏洞总是离不开Hooks。
从 https://github.com/git/git/commit/867ad08a2610526edb5723804723d371136fc643
中得知，core.hooksPath为Hooks目录控制参数，所以我们只要想办法控制这个参数就有可能RCE了。

这里并不知道CRLF插入的core能否append参数，所以测试了一下





不过后来想到了，就算不能append参数，也可以利用之前的优化解析特性将core参数合并到一起。

接下来就是找寻能够控制的文件名，众所周知hooks的文件名必须是固定的。这里我卡了一会，一直想着要文件上传，不过其实利用临时仓库就可以控制文件内容了，可惜的

结果是没利用成功，转而通过读源码以及参考手册https://git-scm.com/docs/git-config
，找了一些能执行命令的点，例如core.pager，但是很奇怪，直接执行能够触发，通过go却无法执行命令。这里需要对git进行调试，由于是子进程调试所以一直没有时间做

在绝望的时候，LuckyCat师傅告诉我说，`core.gitProxy`能够执行程序，但是无法带上参数。
我认为想要利用这个的话，必须要上传脚本到服务器然后执行，同样因为没有可执行权限，应该是没有办法利用的。
不过我注意到了预期相近的一个参数

core.gitProxy

A "proxy command" to execute (as *command host port*) instead of establishing direct connection to the remote server when using the Git protocol for fetching. If the variable value is in the "COMMAND for DOMAIN" format, the command is applied only on hostnames ending with the specified domain string. This variable may be set multiple times and is matched in the given order; the first match wins.

Can be overridden by the `GIT_PROXY_COMMAND` environment variable (which always applies universally, without the special "for" handling).

The special string `none` can be used as the proxy command to specify that no proxy be used for a given domain pattern. This is useful for excluding servers inside a firewall from proxy use, while defaulting to a common proxy for external domains.

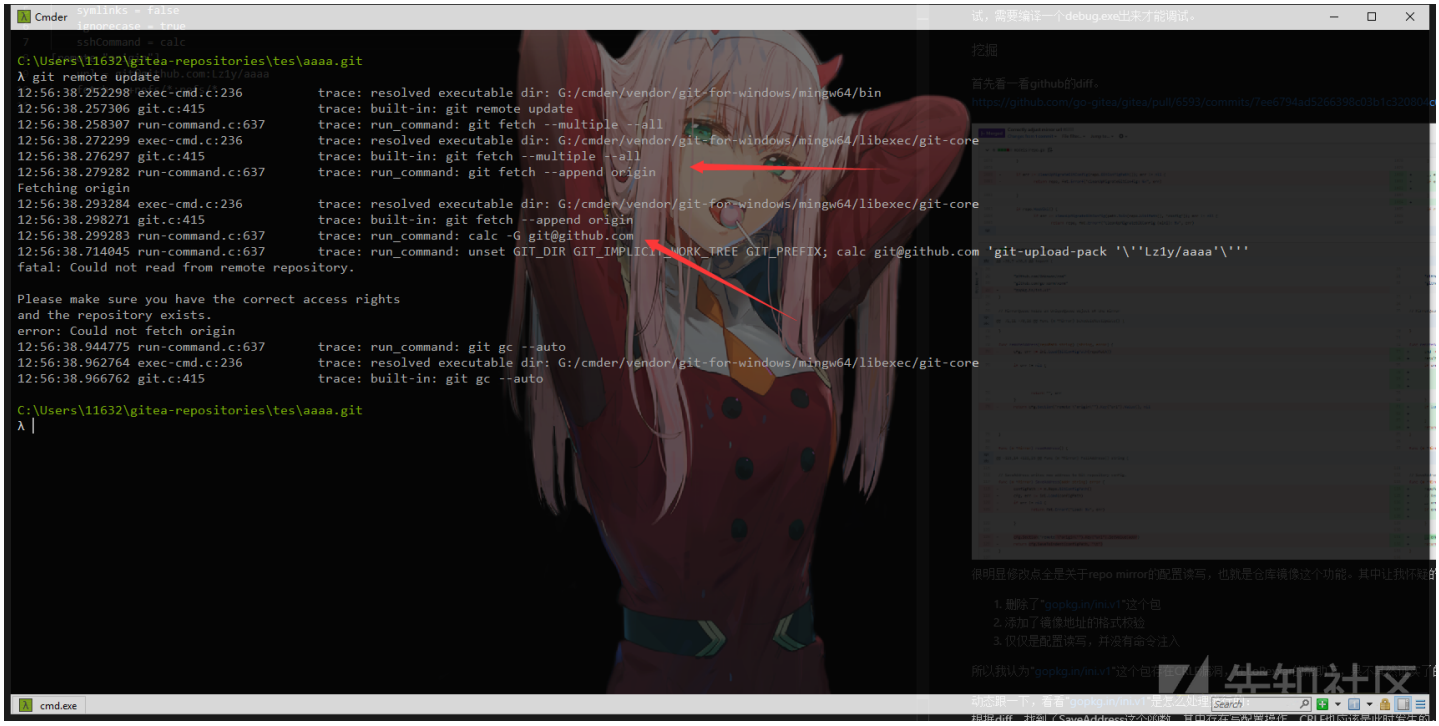
§ **core.sshCommand**

If this variable is set, `git fetch` and `git push` will use the specified command instead of `ssh` when they need to connect to a remote system. The command is in the same form as the `GIT_SSH_COMMAND` environment variable and is overridden when the environment variable is set.

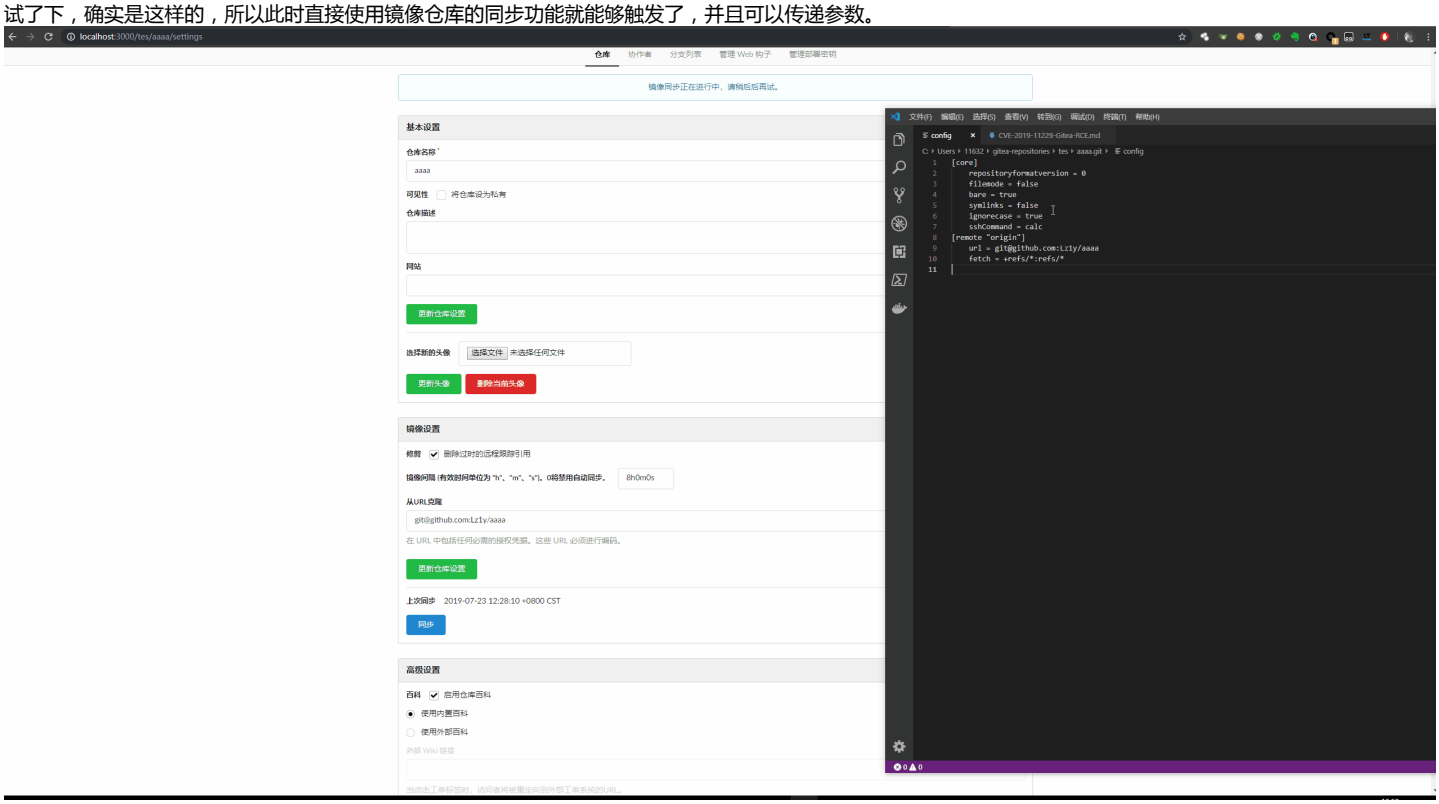
core.sshCommand根据描述就是，当调用ssh的时候，使用我们提供的值替换ssh。这里常用作填充ssh参数，方便ssh使用。所以只要控制此参数，并调用git push或者git fetch即可。
具体实现代码：https://github.com/git/git/blob/6e0cc6776106079ed4efa0cc9abace4107657abf/connect.c

```
if (protocol == PROTO_SSH) {
          char *ssh_host = hostandport;
          const char *port = NULL;
          transport_check_allowed("ssh");
          get_host_and_port(&ssh_host, &port);

          if (!port)
              port = get_port(ssh_host);

          if (flags & CONNECT_DIAG_URL) {
              printf("Diag: url=%s\n", url ? url : "NULL");
              printf("Diag: protocol=%s\n", prot_name(protocol));
              printf("Diag: userandhost=%s\n", ssh_host ? ssh_host : "NULL");
              printf("Diag: port=%s\n", port ? port : "NONE");
              printf("Diag: path=%s\n", path ? path : "NULL");

              free(hostandport);
              free(path);
              free(conn);
              strbuf_release(&cmd);
              return NULL;
          }
          conn->trace2_child_class = "transport/ssh";
          fill_ssh_args(conn, ssh_host, port, version, flags);
      } else {
```

可以看到当使用ssh协议的时候就会进入此分支。一开始还在找直接调用git fetch或者git push的触发点，这时候LuckyCat师傅说直接git remote update也行，其中调用了git fetch。



试了下，确实是这样的，所以此时直接使用镜像仓库的同步功能就能够触发了，并且可以传递参数。



结语

至此，利用链就完整了。近几年的几个gitea的大洞都是组合拳，玩起来相当有意思。另外，代码分析的并不是特别多，尤其是最后触发点。有点太靠运气了，不过时间实在
最后，希望有生之年我也能挖到233

点击收藏 | 0 关注 | 1

上一篇：堆入门---unlink的理解和各... 下一篇：论菜鸡pwn手如何在无网环境（ps...

1. 0 条回复
   - 动动手指，沙发就是你的了！

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