nhttpd 从目录穿越到远程代码执行漏洞分析(CVE-2019-16278)

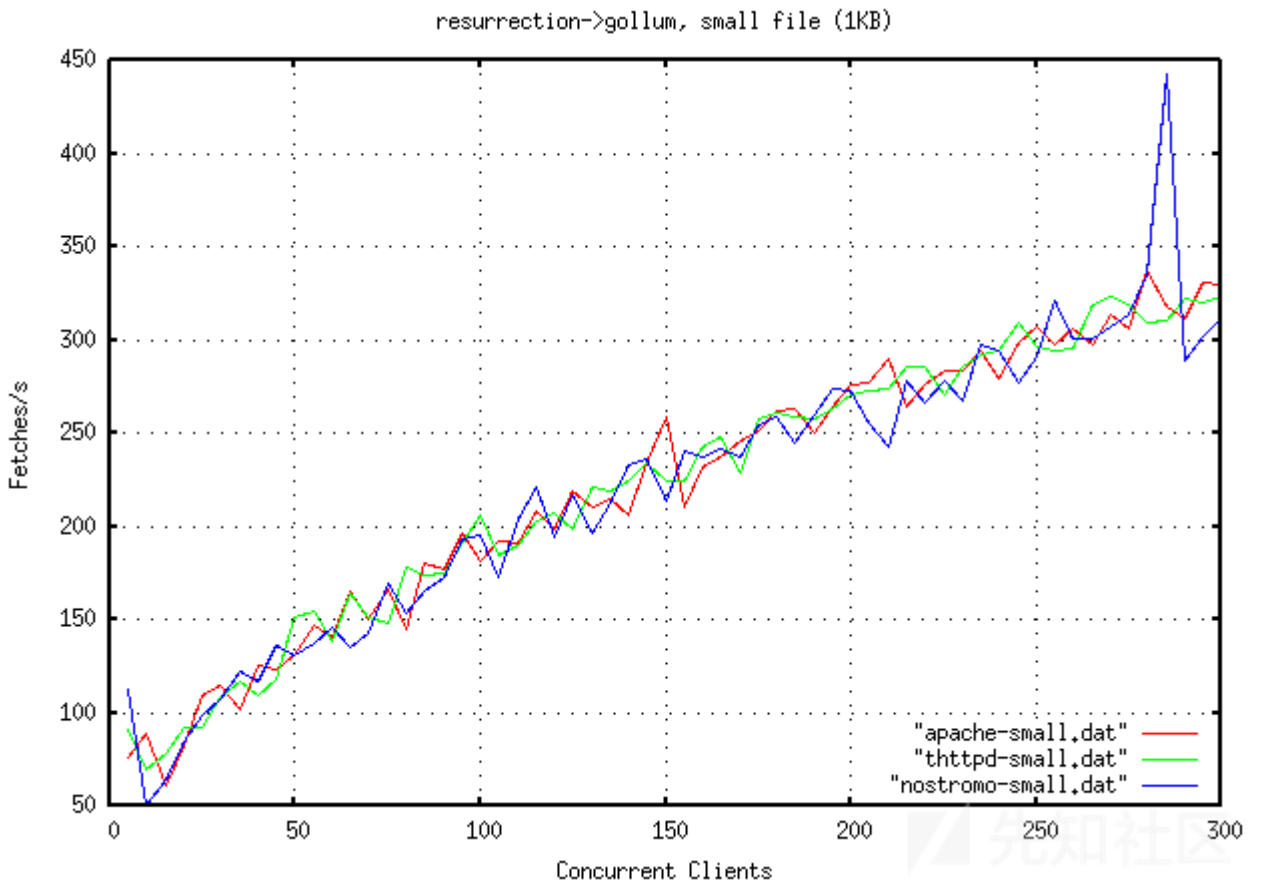
熊本熊本熊 / 2019-11-15 09:29:49 / 浏览数 5328 安全技术 漏洞分析 顶(0) 踩(0)

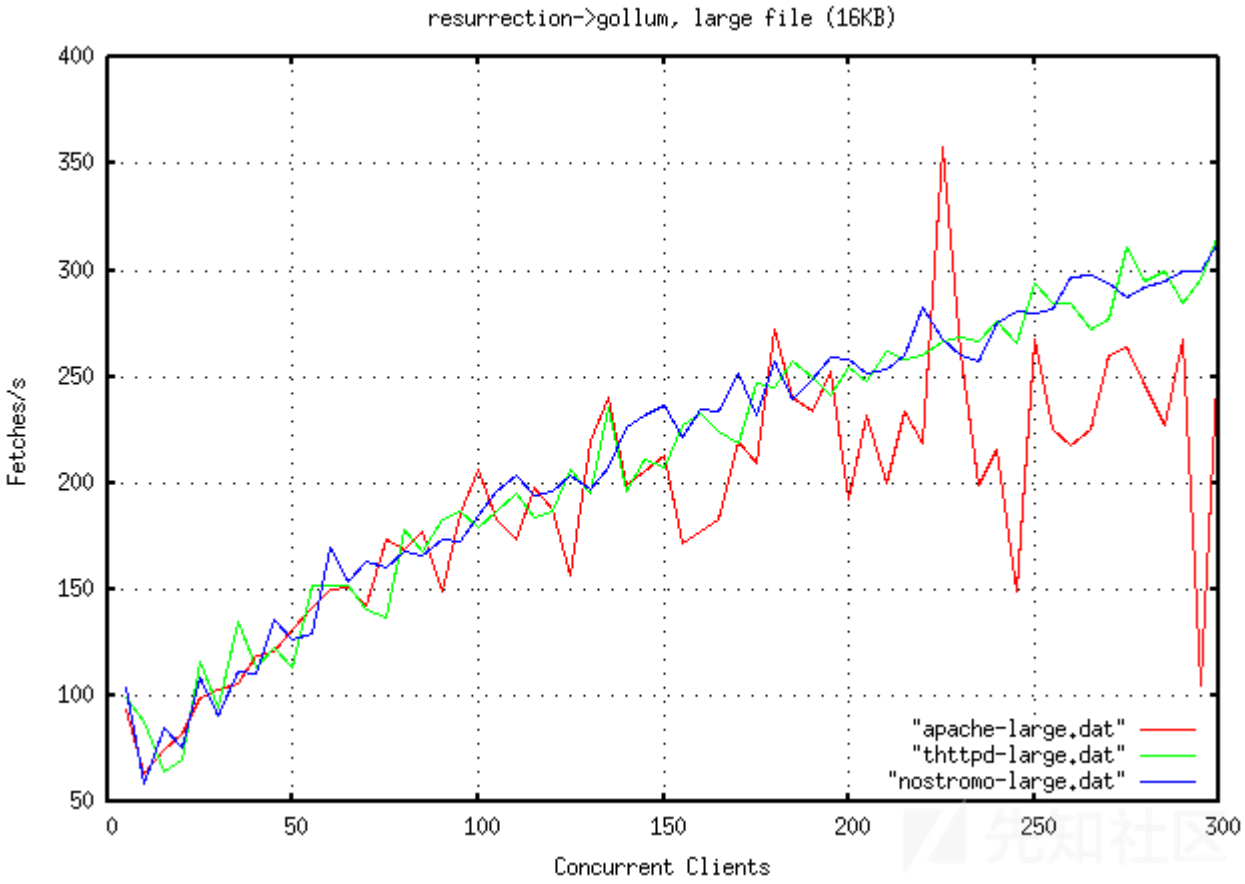
前言

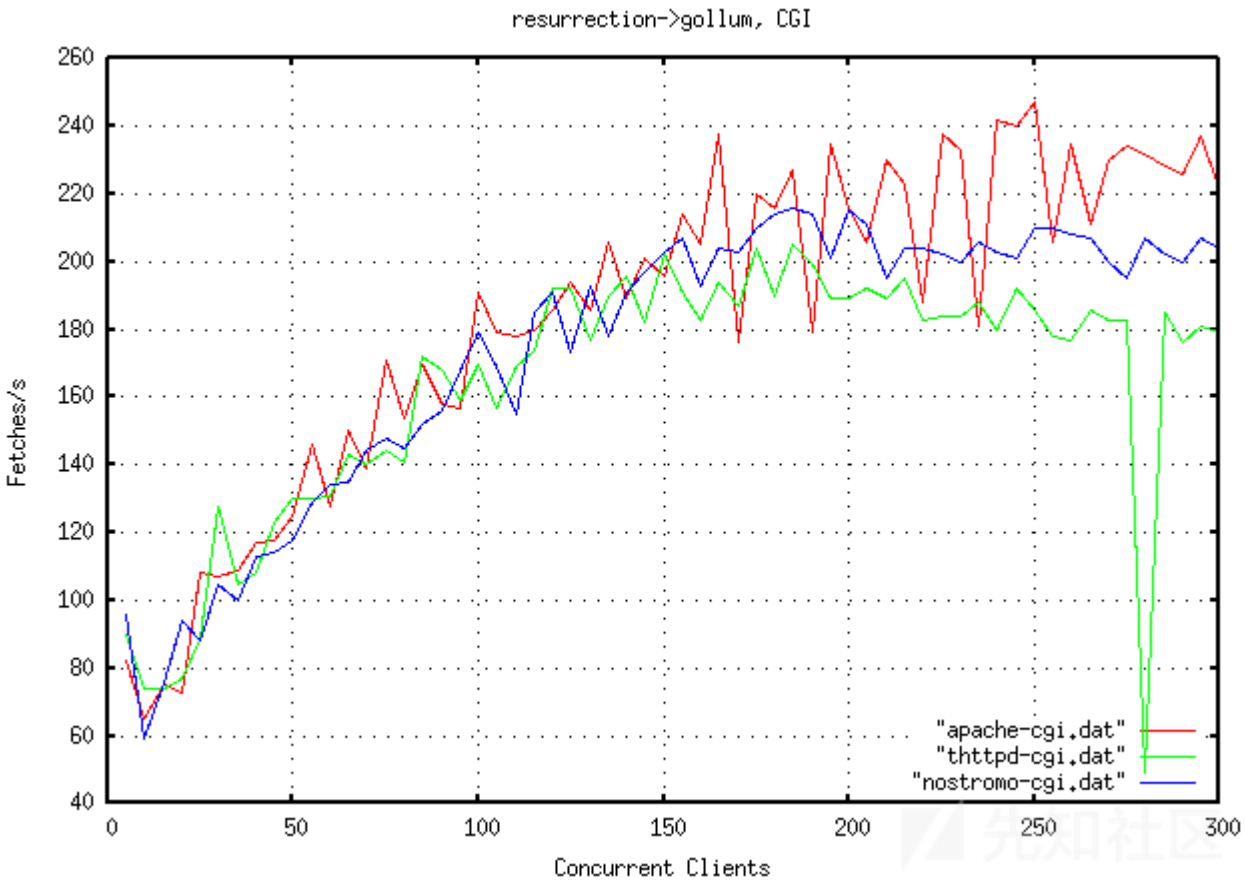
nhttpd是Marcus Glocker设计的开源Web服务器，也称为NostromoWeb服务器。

nhttpd通过select(2)处理正常的并发连接，但是为了提高效率（例如列目录和CGI执行），它使用fork(2)以提高效率。

下图为nostromo以及apache和thttpd在处理1KB文件，16KB文件、执行CGI时的性能比较







漏洞描述

nhttpd 1.9.6及之前版本中存在路径穿越漏洞。攻击者可利用该漏洞访问web server路径之外的文件。

漏洞分析

目录穿越

在main.c中，处理header处存在如下代码

```
1564            /* and process every single header */
1565            for (i = 0; i < k; i++) {
1566                s = 0;
1567                r = 1;
1568                s = http_verify(header[i], sizeof(header[i]),
1569                    c[sdnow].ip, sdnow, i);
1570                if (s == 1)
1571                    r = http_proc(header[i], body, i, size,
1572                        sdnow);
1573                if (r == 0)
1574                    goto quit;
1575                /* on error abort processing */
1576                if (r == -1) {
1577                    strcutl(tmp, header[i], 1, sizeof(tmp));
1578                    h = http_head(http_s_500, tmp,
1579                        c[sdnow].ip, 0);
1580                    b = http_body(http_s_500, "", h, 0);
1581                    c[sdnow].pfdo++;
1582                    c[sdnow].pfdn[i] = 1;
1583                    c[sdnow].pfdh[i] = strdup(b);
1584                    c[sdnow].x_ful[i] = 1;
1585                    c[sdnow].x_chk[i] = 0;
1586                    c[sdnow].x_sta = 0;
1587                    free(h);
1588                    free(b);
1589                }
1590            }
```


程序将会调用http_verify对每个header进行有效性验证，如上图1568行

当http_verify函数对header有效性验证通过后，会调用http_proc对header进行处理，以及后续操作

跟入http_verify方法中

```
94  /*
95   * http_verify()
96   *  verify if incoming header is valid
97   * Return:
98   *  0 = invalid header, 1 = valid header
99   */
100 int
101 http_verify(char *header, const int header_size, const char *cip, const int sfd,
102     const int hr)
103 {
104     int     r, proto;
105     char        *h, *b, line[1024], protocol[16];
106     time_t      tnow;
107     struct tm   *t;
108
109     r = proto = 0;
110
```


在header有效性验证环节(http_verify)中，http_verify方法验证了header是否可以被url解码、method是否效、以及是否存在"/../"目录变量这样的字符串等内容

```
111     /* check if header URI needs to be decoded */
112     if (http_decode_header_uri(header, header_size) == -1) {
113         h = http_head(http_s_400, "-", cip, 0);
114         b = http_body(http_s_400, "", h, 0);
115         c[sfd].pfdo++;
116         c[sfd].pfdn[hr] = 1;
117         c[sfd].pfdh[hr] = strdup(b);
118         c[sfd].x_ful[hr] = 1;
119         c[sfd].x_chk[hr] = 0;
120         c[sfd].x_sta = 0;
121         free(h);
122         free(b);
123         return (0);
124     }
```


上图：header是否可以被url解码http_decode_header_uri，该方法有三种可能的返回值，-1(error) / 0 (不需要解码)/ 解码后的值。-1则返回400状态码

```
126     /* check for valid method */
127     if (strcutl(line, header, 1, sizeof(line)) > 0) {
128         if (!strncasecmp("GET ", line, 4))
129             r = 1;
130         else if (!strncasecmp("POST ", line, 5))
131             r = 1;
132         else if (!strncasecmp("HEAD ", line, 5))
133             r = 1;
134     }
135     if (r == 0) {
136         h = http_head(http_s_501, line, cip, 0);
137         b = http_body(http_s_501, "", h, 0);
138         c[sfd].pfdo++;
139         c[sfd].pfdn[hr] = 1;
140         c[sfd].pfdh[hr] = strdup(b);
141         c[sfd].x_ful[hr] = 1;
142         c[sfd].x_chk[hr] = 0;
143         c[sfd].x_sta = 0;
144         free(h);
145         free(b);
146         return (r);
147     }
```


上图：判断header 是否是GET/POST/HEAD method中的一个，不是则返回501状态码

```
149     /* check for valid uri */
150     if (strstr(header, "/../") != NULL) {
151         h = http_head(http_s_400, line, cip, 0);
152         b = http_body(http_s_400, "", h, 0);
153         c[sfd].pfdo++;
154         c[sfd].pfdn[hr] = 1;
155         c[sfd].pfdh[hr] = strdup(b);
156         c[sfd].x_ful[hr] = 1;
157         c[sfd].x_chk[hr] = 0;
158         c[sfd].x_sta = 0;
159         free(h);
160         free(b);
161         return (0);
162     }
```


上图：判断header中是否存在"/../",存在则返回400状态码

在通过http_verify校验后，理论上是不允许出现"/../"这样导致目录穿越漏洞的字符串，导致漏洞产生的原因，需要接下来往后看。

通过验证的header，紧接着通过http_proc处理，如下图1571行

```
1564            /* and process every single header */
1565            for (i = 0; i < k; i++) {
1566                s = 0;
1567                r = 1;
1568                s = http_verify(header[i], sizeof(header[i]),
1569                    c[sdnow].ip, sdnow, i);
1570                if (s == 1)
1571                    r = http_proc(header[i], body, i, size,
1572                        sdnow);
1573                if (r == 0)
```


在该方法的290行处，可见使用http_header方法进行header解析

```
269 http_proc(const char *header, char *body, const int hr, const int blen,
270     const int sfd)
271 {
272     DIR     *odir;
273     int     cpid, file, fds1[2], fds2[2];
274     int     i, r, s, rp, rt, sp, st, len, size, dirents;
275     us_int      x_nph, x_fork, x_cpage;
276     char        **dirsort, *b, *h, *x = NULL;
277     char        ch[1], image[128], tmp[1024], full[1024], status[1024];
278     char        buf[BS], cpage[1024], *cgiarg[2], *cgienv[64];
279     struct tm   *t = NULL;
280     struct stat sta;
281     struct dirent   *rdir;
282     struct header   *rh;
283     struct timeval  tv;
284     fd_set      set_r;
285
286     r = 1;
287     sp = st = x_nph = x_fork = x_cpage = 0;
288
289     /* parse request header */
290     if ((rh = http_header(header, NULL, 0, blen, sfd)) == NULL)
291         return (-1);
```


http_header方法解析传入的header并在结构中返回我们的响应，如下图

```
1458  struct header *
1459  http_header(const char *header_data, const char *force_status,
1460      const int force_status_nr, const int blen, const int sfd)
1461  {
1462      intmax_t    fsize;
1463      int     file, i, r;
1464      char        line[1024], option[1024];
1465      char        alias[1024], cgi_full[1024];
1466      char        docroot[1024], file_path[1024];
1467      char        acc_base64[1024], acc_file[1024];
1468      char        acc_userpw[1024], acc_pw[1024];
1469      char        tmp[1024], status[128];
1470      char        *x, *type = NULL;
1471      time_t      tnow;
1472      struct tm   *t;
1473      struct stat s;
1474      struct header   *h = NULL;
1475
```


在该方法中存在如下代码

```
1506      /* parse request line */
1507      strcutl(line, header_data, 1, sizeof(line));
1508      strcutw(h->rq_method, line, 1, sizeof(h->rq_method));
1509      strcutw(h->rq_uri, line, 2, sizeof(h->rq_uri));
1510      strcutw(h->rq_protocol, line, 3, sizeof(h->rq_protocol));
1511
1512      /* set protocol depended flags */
1513      if (!strcasecmp(h->rq_protocol, http_fv_pr1))
1514          h->x_chk = 1;
1515
```


如上图1507行，http_header方法调用strcutl方法对header_data进行解析

strcutl方法中60行处，存在如下图代码

```
59      for (j = 0; src[i] != '\n' && src[i] != '\0' && j != dsize - 1; i++) {
60          if (src[i] == '\r')
61              continue;
62          dst[j] = src[i];
63          j++;
64      }
65
```


此处代码的本意，是通过'\n'或'\r\n'切分传入的header字符串

这里解释一下"\r"、"\n"以及"\r\n"：

CR：Carriage Return，对应ASCII中转义字符\r，表示回车

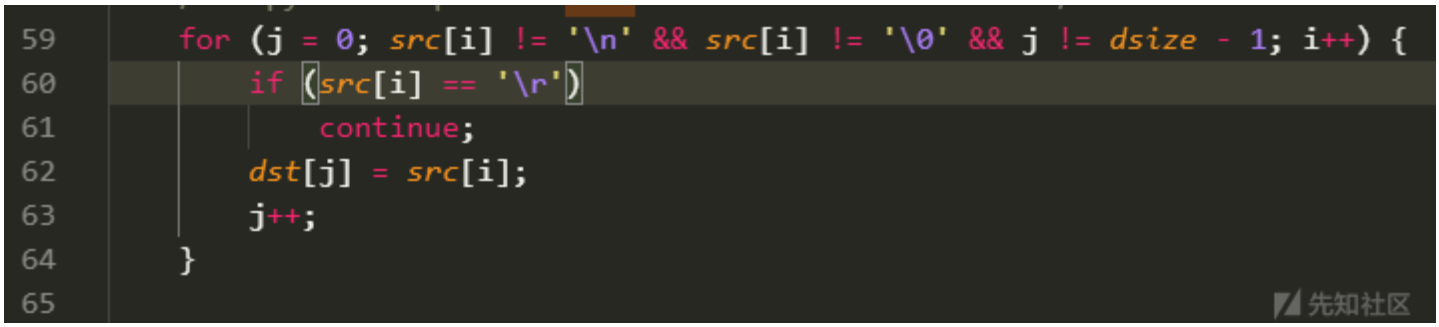
LF：Linefeed，对应ASCII中转义字符\n，表示换行

CRLF：Carriage Return & Linefeed，\r\n，表示回车并换行

由于操作系统的不同,Windows以及Linux等采用不同的换行方式：

Windows操作系统采用两个字符来进行换行，即CRLF(\r\n)；

Unix/Linux/Mac OS X操作系统采用单个字符LF(\n)来进行换行



上图代码的本意，是通过'\n'或'\r\n'切分传入的header字符串，当for循环中原始header中出现'\n'，则认为是一行数据的截至，停止对dst变量进行赋值，此时dst变量中存

但是在windows中，会使用\r\n进行换行，数据中的header会以如下形式存在：

"line1\r\nline2"

为了在切割header行时，为了不把"\r"混入line1中，即避免存在"line1\r"情况的出现，程序通过判断原始header中逐位的字符是否是"\r"，若该处字符是"\r"，则continue，

因此，程序在用这种方式处理"line1\r\nline2"这样的windows下的数据时，第一行的dst值会是"line1",而不会是"line1\r"

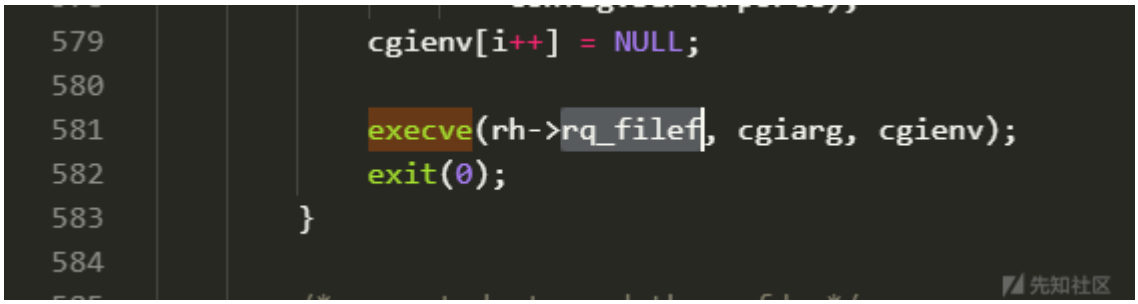
但是开发者仅仅考虑在处理windows
换行符("\r\n")换行时的问题，并简单的将"\r"去除，并没有意识到攻击者可能构造的特殊情况，例如：

"/..\r/..\r/..\r/"

这样的payload，在经过程序处理时，可以安全的通过http_verify对"/../"的校验。然而这人个payload在strcutl方法中，"\r"会被去除，最终dst值为/../../../。这样的dst数据

RCE

http_proc方法中存在execve函数，如下图



execve()的第一个参数指定准备载入当前进程空间的新程序的路径名，第二个参数指定了传给新进程的命令行参数，最后一个参数指定了新程序的环境列表。

而此处，rh->rq_filef可以通过目录遍历，通过"/../../../../bin/sh",指定web服务目录之外的bin/sh文件，造成远程代码执行漏洞的产生

修复

```
269 http_proc(const char *header, char *body, const int hr, const int blen,
270     const int sfd)
271 {
272     DIR     *odir;
273     int     cpid, file, fds1[2], fds2[2];
274     int     i, r, s, rp, rt, sp, st, len, size, dirents;
275     us_int      x_nph, x_fork, x_cpage;
276     char        **dirsort, *b, *h, *x = NULL;
277     char        ch[1], image[128], tmp[1024], full[1024], status[1024];
278     char        buf[BS], cpage[1024], *cgiarg[2], *cgienv[64];
279     struct tm   *t = NULL;
280     struct stat sta;
281     struct dirent   *rdir;
282     struct header   *rh;
283     struct timeval  tv;
284     fd_set      set_r;
285
286     r = 1;
287     sp = st = x_nph = x_fork = x_cpage = 0;
288
289     /* parse request header */
290     if ((rh = http_header(header, NULL, 0, blen, sfd)) == NULL)
291         return (-1);
```


官方给出的修复方案如上，这里不仅判断了当前字符是"\r"，更是判断了"\r"后一位字符是否是"\n"。也就是说，只有"\r\n"这样的windows换行符出现时，"\r"会被去除原pa
/\r../..\r/"经过此次修复，最终解析后的dst值仍为"/\r../..\r/"

如果构造payload"/\r\n../..\r\n/",虽然可以去除"\r",但由于随后的"\n"导致for循环的终止，此时dst值为"/",仍不能满足目录穿越

上一篇：npm模块名中的命令注入漏洞分享 下一篇：记一次ueditor老版本的非常规...

1. 0 条回复

- 动动手指，沙发就是你的了！