CVE-2019-12527: Squid 远程命令执行漏洞

mss**** / 2019-08-27 08:57:00 / 浏览数 4620 安全技术 漏洞分析 顶(0) 踩(0)

原文地址：https://www.thezdi.com/blog/2019/8/22/cve-2019-12527-code-execution-on-squid-proxy-through-a-heap-buffer-overflow

在本文中，我们将为读者详细介绍安全人员最近在Web代理Squid中发现的一个缓冲区溢出漏洞。需要注意的是，远程的、未经身份验证的攻击者可以通过向目标服务器发送

[Squid是一款非常流行的开源Internet代理和Web缓存应用程序。它可用于降低Web服务器的带宽用量，过滤网络流量，并通过本地缓存常用资源来加速Web访问。此外，

基础知识

在Squid中，它为用户提供了缓存管理器接口`cachemgr.cgi`，可用于展示Web上Squid代理进程方面的统计信息。除了显示统计信息之外，缓存管理器还可用于管理缓存，

HTTP是由RFC
7230-7237及其他RFC中描述的一种请求/响应型通信协议。其中，请求将从客户端发送到服务器，之后，服务器又会将响应发送回客户端。HTTP请求由请求行、各种头部，

```
Request = Request-Line headers CRLF [message-body]
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
Headers = *[Header]
Header = Field-Name ":" Field-Value CRLF
```

其中，`CRLF`表示回车换行，而SP则表示空格字符。另外，各个参数可以放在Request-URI或消息正文中，以名称-值对的形式从客户端传递给服务器；具体放到哪里，要视乎

```
GET /my_webapp/mypaget.htm?param=1 HTTP/1.1
Host: www.myhost.com
```


HTTP支持多种身份验证方案，并且大多方案都会用到“Authorization”头部，其格式如下所示：

```
Authorization: <type> <credentials>
```


其中，<type>常见的取值包括“Basic”、“Digest”、“OAuth”和“Negotiate”。</type>

漏洞描述

安全研究人员发现，Squid中存在一个缓冲区溢出漏洞。当Squid收到针对`cachemgr`的传入请求时，将调用`CacheManager::ParseHeaders()`函数来解析请求的头部。这

类似地，当Squid用作FTP代理并且发送以"ftp"开头的`REQUEST-URI`请求时，将调用`HttpHeader::getAuth()`函数。

而`HttpHeader::getAuth()`函数会用到一个长8192字节的缓冲区decodeAuthToken，准确来说，使用base64_decode_update()函数对`<credentials>`进行base64解

远程攻击者可以通过向目标服务器发送精心设计的HTTP请求来利用这个漏洞。攻击者一旦得手，就能够以服务器进程的权限来执行任意代码；如果失败，攻击活动将导致服

源码分析

下面展示的是取自Squid 4.7版的源代码片段，其中已经加入了相应的代码注释。

下面代码来自`src/cache_manager.cc`文件：

```
void
CacheManager::ParseHeaders(const HttpRequest * request, Mgr::ActionParams &params)
{
    assert(request);

    params.httpMethod = request->method.id();
    params.httpFlags = request->flags;

    // Call the vulnerable function
    const char *basic_cookie = request->header.getAuth(Http::HdrType::AUTHORIZATION, "Basic");
[...Truncated for readability...]
}
```


下面的代码来自src/clients/FtpGateway.cc文件：

```
int
Ftp::Gateway::checkAuth(const HttpHeader * req_hdr)
{
    /* default username */
    xstrncpy(user, "anonymous", MAX_URL);

    // Call the vulnerable function
    const SBuf auth(req_hdr->getAuth(Http::HdrType::AUTHORIZATION, "Basic"));
    if (!auth.isEmpty()) {
        flags.authenticated = 1;
        loginParser(auth, false);
    }
[...Truncated for readability...]
}
```


下面的代码来自src/HttpHeader.cc文件：

```
const char *
HttpHeader::getAuth(Http::HdrType id, const char *auth_scheme) const
{
    const char *field;
    int l;
    assert(auth_scheme); // auth_scheme is "Basic"
    field = getStr(id); // id is "Authorization"

    if (!field) /* no authorization field */
        return NULL;

    l = strlen(auth_scheme);

    if (!l || strncasecmp(field, auth_scheme, l)) /* wrong scheme */
        return NULL;

    field += l;

    if (!xisspace(*field)) /* wrong scheme */
        return NULL;

/* skip white space */
for (; field && xisspace(*field); ++field);

if (!*field) /* no authorization cookie */
    return NULL;

static char decodedAuthToken[8192]; // Buffer of fixed size 8192
struct base64_decode_ctx ctx;
base64_decode_init(&ctx);
size_t decodedLen = 0;
// Call to function within which heap buffer overflow occurs
if (!base64_decode_update(&ctx, &decodedLen, reinterpret_cast<uint8_t*>
            (decodedAuthToken), strlen(field), field) ||
        !base64_decode_final(&ctx)) {
    return NULL;
    }
    decodedAuthToken[decodedLen] = '\0';
    return decodedAuthToken;
}
```


最后，下面的代码来自 `lib/base64.c`文件：

```c
int
base64_decode_update(struct base64_decode_ctx *ctx,
                     size_t *dst_length,
                     uint8_t *dst, // Fixed-sized 8192-byte buffer
                     size_t src_length, // Attacker-controlled length
                     const char *src) // Attacker controlled buffer
{
    size_t done;
    size_t i;

    // Iterate through each byte in base64-encoded data,
    // write it to dst. Potential buffer overflow.
    for (i = 0, done = 0; i<src_length; i++)
        switch(base64_decode_single(ctx, dst + done, src[i]))
        {
        case -1:
            return 0;
        case 1:
            done++;
        /* Fall through */
        case 0:
            break;
        default:
            abort();
        }

    assert(done <= BASE64_DECODE_LENGTH(src_length));

    *dst_length = done;
    return 1;
}
```


触发漏洞

以下数据包解码结果展示了从客户端发送给目标Squid代理的攻击请求的相关片段：

```
Frame 4: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
Ethernet II, Src: Vmware_b4:39:7d (00:50:56:b4:39:7d), Dst: Vmware_b4:0e:7f (00:50:56:b4:0e:7f)
Internet Protocol Version 4, Src: 172.16.9.9 (172.16.9.9), Dst: 172.16.9.230 (172.16.9.230)
Transmission Control Protocol, Src Port: 38215 (38215), Dst Port: ndl-aas (3128), Seq: 1, Ack: 1, Len: 1448
Hypertext Transfer Protocol
    GET cache_object://172.16.9.230/info HTTP/1.1\r\n
    Host: 172.16.9.230\r\n

0000 00 50 56 b4 0e 7f 00 50 56 b4 39 7d 08 00 45 00 .PV....PV.9}..E.
0010 05 dc aa f0 40 00 40 06 1f 1c ac 10 09 09 ac 10 ....@.@.........
0020 09 e6 95 47 0c 38 b1 26 9c 9d e6 b9 20 46 80 10 ...G.8.&.... F..
0030 01 f6 5b 75 00 00 01 01 08 0a 96 70 f5 f0 aa 43 ..[u.......p...C
0040 a4 b4 47 45 54 20 63 61 63 68 65 5f 6f 62 6a 65 ..GET cache_obje
0050 63 74 3a 2f 2f 31 37 32 2e 31 36 2e 39 2e 32 33 ct://172.16.9.23
0060 30 2f 69 6e 66 6f 20 48 54 54 50 2f 31 2e 31 0d 0/info HTTP/1.1.
0070 0a 48 6f 73 74 3a 20 31 37 32 2e 31 36 2e 39 2e .Host: 172.16.9.
0080 32 33 30 0d 0a 41 75 74 68 6f 72 69 7a 61 74 69 230..Authorizati
0090 6f 6e 3a 20 42 61 73 69 63 20 51 54 70 43 51 6b on: Basic QTpCQk
00a0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
00b0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
00c0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
00d0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
00e0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
00f0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0100 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0110 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0120 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0130 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0140 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0150 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0160 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0170 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0180 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
0190 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
01a0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
01b0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
01c0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
01d0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
[...Truncated for readability...]
05c0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
05d0 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b 4a 43 51 6b JCQkJCQkJCQkJCQk
05e0 4a 43 51 6b 4a 43 51 6b 4a 43 JCQkJCQkJC
```


如前所述，标准的Authorization头部的格式为：

```
Authorization: <type> <credentials>
```


在“Authorization:
Basic”后面，有一个字符串，其中只有一部分显示在了这个片段中。当对整个字符串进行base64解码时，解码结果会溢出8192字节的缓冲区。请注意，base64解码过程产生

关于漏洞补丁

Squid维护者已经通过最新的commit修复了这个漏洞。根据相关说明，我们发现他们是通过用SBuf替换用来存放解码后的base64令牌的具有固定大小的缓冲区来修复这个安

上一篇：X-NUCA 2019 线上赛 W... 下一篇：从国赛决赛的webpwn到Delc...

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