0day.work

Just another security blog by Sebastian Neef

0day

CVE-2019-11360: BufferOverflow in iptables-restore v1.8.2

2019-07-11 8 minute read

This blogpost is about a BufferOverflow vulnerability which I found by fuzzing `iptables-restore` using AFL in March, 2019. It was fixed by the netfilter team in April 2019 and was assigned CVE-2019-11360 by MTIRE.

Fuzzing iptables-restore

Back in March I somehow came up with the idea of fuzzing iptables. I thought using `iptables-restore` as the target binary would be a good idea, because I could use AFL to supply iptables rules files to it.

You can save your current rules with `iptables-save > /tmp/rules` and restore them later (i.e. on boot) using `iptables-restore /tmp/rules`.

The test harness consisted of two such rules files that I took from my laptop and a server that I own. As iptables is the user-interface to the kernel's netfilter subsystem, the source code is open source and can be downloaded from the git.netfilter.org repository.

To be honest, I can't remember for how long the master and slave AFL instance was fuzzing on my fuzzing-VM, but I think it was not longer than 1-2 weeks until the first crash showed up.

BufferOverflow in add_param_to_argv

I downloaded the crash file and used GDB to run it against the AFL-instrumented `iptables-restore` binary. The crash occurred in the function `add_param_to_argv` in `iptables/xshared.c`.

Having the source code, I started looking at the function and trying to understand why the crash occurred and if it was exploitable.

The vulnerable code can be found in iptables v1.8.2 and looks like this:

```
1  void add_param_to_argv(char *parsestart, int line)
2  {
3          int quote_open = 0, escaped = 0, param_len = 0;
4          char param_buffer[1024], *curchar;
5
6          /* After fighting with strtok enough, here's now
7           * a 'real' parser. According to Rusty I'm now no
8           * longer a real hacker, but I can live with that */
9
10         for (curchar = parsestart; *curchar; curchar++) {
11                 if (quote_open) {
12                         if (escaped) {
13                                 param_buffer[param_len++] = *curchar;
14                                 escaped = 0;
15                                 continue;
16                         } else if (*curchar == '\\') {
17                                 escaped = 1;
18                                 continue;
19                         } else if (*curchar == '"') {
20                                 quote_open = 0;
21                                 *curchar = '"';
22                         } else {
23                                 param_buffer[param_len++] = *curchar;
24                                 continue;
```

```
25                      }
26              } else {
27                      if (*curchar == '"') {
28                              quote_open = 1;
29                              continue;
30                      }
31              }
32
33              switch (*curchar) {
34              case '"':
35                      break;
36              case ' ':
37              case '\t':
38              case '\n':
39                      if (!param_len) {
40                              /* two spaces? */
41                              continue;
42                      }
43                      break;
44              default:
45                      /* regular character, copy to buffer */
46                      param_buffer[param_len++] = *curchar;
47
48                      if (param_len >= sizeof(param_buffer))
49                              xtables_error(PARAMETER_PROBLEM,
50                                            "Parameter too long!");
51                      continue;
52              }
53
54              param_buffer[param_len] = '\0';
55
56              /* check if table name specified */
57              if ((param_buffer[0] == '-' &&
58                   param_buffer[1] != '-' &&
59                   strchr(param_buffer, 't')) ||
60                  (!strncmp(param_buffer, "--t", 3) &&
61                   !strncmp(param_buffer, "--table", strlen(param_b
62                      xtables_error(PARAMETER_PROBLEM,
63                                    "The -t option (seen in line %u
64                                    line, xt_params->program_name);
65              }
66
67              add_argv(param_buffer, 0);
68              param_len = 0;
```

```
69          }
70  }
```

The function is part of the argument parsing process of iptables, i.e. `-p tcp --dport "<some-argument-here>"`. It uses a fixed `param_buffer[1024]` on the stack and overflowing it leads to the crash.

A motivated reader will take a 5-minute break to look at the code and find the BoF himself ;-)

The `for`-loop goes over all characters of the string that is passed to the function through `char *parsestart`. In the lower part there is a `switch` blck which will match different characters and conduct a bounds check in the `default:` branch. If an argument is longer than the 1024-bytes `param_buffer`, then an error is thrown and the BoF prevented.

However, the upper part of the `for`-loop checks if an argument is quoted. If the `*curchar` equals `"` then `quote_open` is set to `1` in the `else` branch and the loop starts over (`continue`). In the next iteration, it will enter the `if (quote_open)` branch. As long as no matching `"` is found, the loop will stay in the upper part due to the `continue` statements:

```
1       param_buffer[param_len++] = *curchar;
2       continue;
```

That means that it will never reach the bounds check in the lower part and copy as many characters as the `*parsestart` has to the stack, thereby overflowing the `param_buffer` and allowing an attacker to rewrite the stack.

Proof of Concept

I started minifying the AFL-generated crash file and managed to create the following PoC iptables file which overwrites `RIP` with `0x414141414141`:

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp --dport "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaal
COMMIT
```



One could argue that the severity of the issue is not high, because one already needs root privileges to run `iptables-restore` in a standard debian setup. So if an attacker is capable of running `iptables-restore`, then she already can do more harm to your system anyway. If an administrator can be convinced to apply unknown iptables rules files, then her system is most likely already compromised.

Furthermore, I had to disable all well-known security features like `NX`, `Canaries` and `PIE` to build this PoC. This means, that building a real-world exploit probably takes much more effort and/or is quite hard:

```
$> checksec -f /usr/bin/iptables-restore
RELRO           STACK CANARY      NX            PIE             RPATH
Full RELRO      Canary found      NX enabled    PIE enabled     No RPA
```

All in all, I believe that this vulnerability can only be used for academic/educational purposes and has no particular real-world impact. [I'd really appreciate if you prove me wrong ;-)]

Patch

As iptables is open source, I decided to try and come up with a patch myself. I took the easiest and most obvious approach of simply doing a bounds check at the beginning of the loop:

```
fuzz@fuzzing:~/fuzzing/iptables/pkg-iptables/iptables# diff -Naur xsh
--- xshared.c.old       2019-04-13 20:50:07.871158901 +0100
+++ xshared.c   2019-04-13 20:42:59.216774862 +0100
@@ -443,6 +443,10 @@
          * longer a real hacker, but I can live with that */

         for (curchar = parsestart; *curchar; curchar++) {
+                if (param_len >= sizeof(param_buffer))
+                        xtables_error(PARAMETER_PROBLEM,
+                                      "Parameter too long!");
+
                 if (quote_open) {
                         if (escaped) {
                                 param_buffer[param_len++] = *curchar;
@@ -479,10 +483,6 @@
                 default:
                         /* regular character, copy to buffer */
                         param_buffer[param_len++] = *curchar;
-                        if (param_len >= sizeof(param_buffer))
-                                xtables_error(PARAMETER_PROBLEM,
-                                              "Parameter too long!");
-
                         continue;
                 }
```

I sent the patch to Pablo in hopes to make his life easier. The netfilter team decided to completely rewrite my patch and, regarding to my more proficient C-developer friends, make it more readable.

The final patch can be found in their git repo and looks like this:

```
diff --git a/iptables/xshared.c b/iptables/xshared.c
index fb186fb1..36a2ec5f 100644
--- a/iptables/xshared.c
+++ b/iptables/xshared.c
@@ -433,10 +433,24 @@ void save_argv(void)
        }
 }

+struct xt_param_buf {
+       char    buffer[1024];
+       int     len;
+};
+
+static void add_param(struct xt_param_buf *param, const char *curcha
+{
+       param->buffer[param->len++] = *curchar;
+       if (param->len >= sizeof(param->buffer))
+               xtables_error(PARAMETER_PROBLEM,
+                             "Parameter too long!");
+}
+
 void add_param_to_argv(char *parsestart, int line)
 {
-       int quote_open = 0, escaped = 0, param_len = 0;
-       char param_buffer[1024], *curchar;
+       int quote_open = 0, escaped = 0;
+       struct xt_param_buf param = {};
+       char *curchar;

        /* After fighting with strtok enough, here's now
         * a 'real' parser. According to Rusty I'm now no
@@ -445,7 +459,7 @@ void add_param_to_argv(char *parsestart, int line
        for (curchar = parsestart; *curchar; curchar++) {
                if (quote_open) {
                        if (escaped) {
-                               param_buffer[param_len++] = *curchar;
+                               add_param(&param, curchar);
                                escaped = 0;
                                continue;
                        } else if (*curchar == '\\') {
@@ -455,7 +469,7 @@ void add_param_to_argv(char *parsestart, int line
```

```
                                quote_open = 0;
                                *curchar = '"';
                        } else {
-                               param_buffer[param_len++] = *curchar;
+                               add_param(&param, curchar);
                                continue;
                        }
                } else {
@@ -471,36 +485,32 @@ void add_param_to_argv(char *parsestart, int li
                case ' ':
                case '\t':
                case '\n':
-                       if (!param_len) {
+                       if (!param.len) {
                                /* two spaces? */
                                continue;
                        }
                        break;
                default:
                        /* regular character, copy to buffer */
-                       param_buffer[param_len++] = *curchar;
-
-                       if (param_len >= sizeof(param_buffer))
-                               xtables_error(PARAMETER_PROBLEM,
-                                             "Parameter too long!");
+                       add_param(&param, curchar);
                        continue;
                }

-               param_buffer[param_len] = '\0';
+               param.buffer[param.len] = '\0';

                /* check if table name specified */
-               if ((param_buffer[0] == '-' &&
-                    param_buffer[1] != '-' &&
-                    strchr(param_buffer, 't')) ||
-                   (!strncmp(param_buffer, "--t", 3) &&
-                    !strncmp(param_buffer, "--table", strlen(param_b
+               if ((param.buffer[0] == '-' &&
+                    param.buffer[1] != '-' &&
+                    strchr(param.buffer, 't')) ||
+                   (!strncmp(param.buffer, "--t", 3) &&
+                    !strncmp(param.buffer, "--table", strlen(param.b
                        xtables_error(PARAMETER_PROBLEM,
```

```
86                                      "The -t option (seen in line %u
87                                      line, xt_params->program_name);
88                      }
89
90  -                   add_argv(param_buffer, 0);
91  -                   param_len = 0;
92  +                   add_argv(param.buffer, 0);
93  +                   param.len = 0;
94              }
95      }
```

Timeline

- 03/05/2019 - Report sent to netfilter's coreteam mailing list
- 03/22/2019 - Resent email to Pablo Neira Ayuso, because it was somehow lost
- 04/17/2019 - Provided the netfilter team an initial patch / diff
- 04/XX/2019 - MITRE assigned CVE-2019-11360
- 04/22/2019 - Pablo rewrites and posts the patch for review
- 04/30/2019 - Patch is merged into iptables-restore
- ??/??/???? - iptables v.1.8.3 released
- 07/11/2019 - Blogpost published

-=-

CVE Fuzzing Firewalls

RELATED CONTENT

CVE-2019-1000032: Memory corruption / DoS in nanosvg

CVE-2023-6294: popup-builder <= 4.2.6 Admin+ SSRF & File Read

CVE-2023-6295: so-widgets-bundle < 1.5 Admin+ Local File Inclusion

© 2025 Sebastian Neef

Impressum, Privacy Policy. This is a static site - it does not use any cookies. Built with Hugo. Theme **Stack** designed by Jimmy