Dominic Furano
April 2014

musl vfprintf Code Coverage Report

gcov Coverage as of April 3, 2014

```
gcov vfprintf.c
File 'vfprintf.c'
Lines executed:91.30% of 391
vfprintf.c:creating ‘vfprintf.c.gcov'
```

uncovered code

```
176:  for (; l >= sizeof pad; l -= sizeof pad)
177:        out(f, pad, sizeof pad);
```

The branch that executes this for loop is never taken. I believe this is because I never created a scenario where the padding for a char format specifier was greater than 256.

```
470:  if (l > INT_MAX - cnt) {
471:        errno = EOVERFLOW;
472:        cnt = -1;
```

This branch is impossible to cover without creating a format specifier count that is larger than INT_MAX.

```
499:  if (isdigit(s[1]) && s[2]=='$') {
500:        l10n=1;
501:        nl_type[s[1]-'0'] = INT;
502:        w = nl_arg[s[1]-'0'].i;
503:        s+=3;

513:  if (isdigit(s[2]) && s[3]=='$') {
514:        nl_type[s[2]-'0'] = INT;
515:        p = nl_arg[s[2]-'0'].i;
516:        s+=4;
```

I never engineered a test which exercises the printf functionality of redirecting arguments in a format string. e.g. `printf(“%1$d”, 8);`

```
536:  if (st==NOARG) {
537:        if (argpos>=0) return -1;
```

This if statement never falls through because there is no fuzzing case I designed with no arguments.

```
594:  if (!arg.i && !p) {
595:        a=z;
596:        break;
```

I am not sure what this if statement means without deeper examination.

```
604:  case 'm':
605:        if (1) a = strerror(errno); else
```

No ‘m’ fuzzing case was engineered.

```
623:  if (l<0) return -1;
624:        p = i;
625:        pad(f, ' ', w, p, fl);
626:        ws = arg.p;
627:        for (i=0; i<0U+p && *ws && i+(l=wctomb(mb, *ws++))<=p; i
                  +=l)
628:        out(f, mb, l);
629:        pad(f, ' ', w, p, fl^LEFT_ADJ);
630:        l = w>p ? w : p;
631:        continue;
```

This code is never executed because l is never less than 0.

```
677:  if (!f->buf_size) {
678:        saved_buf = f->buf;
679:        f->wpos = f->wbase = f->buf = internal_buf;
680:        f->buf_size = sizeof internal_buf;
681:        f->wend = internal_buf + sizeof internal_buf;
682:  }

684:  if (saved_buf) {
685:        f->write(f, 0, 0);
686:        if (!f->wpos) ret = -1;
687:        f->buf = saved_buf;
688:        f->buf_size = 0;
689:        f->wpos = f->wbase = f->wend = 0;
690:  }
```

A buffer is provided in the fuzzing code for the musl function used. So the buffer test and cleanup is not needed below.