Spencer Phippen

musl printf coverage report

My fuzzer was able to cover almost all of the musl vfprintf source. However, some portions were not covered – this document examines these sections of code and explains and justifies each portion.

First, here are the summary coverage statistics:

```
Lines executed:97.00% of 367
Branches executed:99.51% of 408
Taken at least once:92.89% of 408
Calls executed:97.01% of 67
```

Near the top of the internal function printf_core (line 469) is the following code:

```
if (cnt >= 0) {
  if (l > INT_MAX - cnt) {
    errno = EOVERFLOW;
    cnt = -1;
  } else cnt += l;
}
```

My fuzzer did not execute the bold lines above. That is because this code can only be executed if you try to execute a printf where the resulting output would be longer than INT_MAX characters. While this is not particularly difficult to do on a 64-bit machine, even through the snprintf interface (virtual memory and malloc is the key), my fuzzer used a fixed size buffer of size much less than INT_MAX, meaning it would never execute this line.

Near the bottom of musl's vfprintf (line 676) is the following code:

```
// FLOCK(f);
if (!f->buf_size) {
  saved_buf = f->buf;
  f->wpos = f->wbase = f->buf = internal_buf;
  f->buf_size = sizeof internal_buf;
  f->wend = internal_buf + sizeof internal_buf;
}
ret = printf_core(f, fmt, &ap2, nl_arg, nl_type);
if (saved_buf) {
  f->write(f, 0, 0);
  if (!f->wpos) ret = -1;
  f->buf = saved_buf;
  f->buf_size = 0;
  f->wpos = f->wbase = f->wend = 0;
}
```

```
// FUNLOCK(f);
```

My fuzzer did not execute the bold lines above. That is because this code will only be run when dealing with real musl file I/O – it deals with FILE structs that don't have an in-memory buffer for file data yet. Because we are testing the function through snprintf only, we cannot execute this code. However, if we tested with functions such as fprintf, we should be able to execute this code.