git_connect_git(): forbid newlines in host and path

Browse files

```
When we connect to a git:// server, we send an initial request that
looks something like:

  002dgit-upload-pack repo.git\0host=example.com

If the repo path contains a newline, then it's included literally, and
we get:

  002egit-upload-pack repo
  .git\0host=example.com

This works fine if you really do have a newline in your repository name;
the server side uses the pktline framing to parse the string, not
newlines. However, there are many _other_ protocols in the wild that do
parse on newlines, such as HTTP. So a carefully constructed git:// URL
can actually turn into a valid HTTP request. For example:

  git://localhost:1234/%0d%0a%0d%0aGET%20/%20HTTP/1.1 %0d%0aHost:localhost%0d%0a%0d%0a

becomes:

  0050git-upload-pack /
  GET / HTTP/1.1
  Host:localhost

  host=localhost:1234

on the wire. Again, this isn't a problem for a real Git server, but it
does mean that feeding a malicious URL to Git (e.g., through a
submodule) can cause it to make unexpected cross-protocol requests.
Since repository names with newlines are presumably quite rare (and
indeed, we already disallow them in git-over-http), let's just disallow
them over this protocol.

Hostnames could likewise inject a newline, but this is unlikely a
problem in practice; we'd try resolving the hostname with a newline in
it, which wouldn't work. Still, it doesn't hurt to err on the side of
caution there, since we would not expect them to work in the first
place.

The ssh and local code paths are unaffected by this patch. In both cases
we're trying to run upload-pack via a shell, and will quote the newline
so that it makes it intact. An attacker can point an ssh url at an
arbitrary port, of course, but unless there's an actual ssh server
there, we'd never get as far as sending our shell command anyway.  We
_could_ similarly restrict newlines in those protocols out of caution,
but there seems little benefit to doing so.

The new test here is run alongside the git-daemon tests, which cover the
same protocol, but it shouldn't actually contact the daemon at all.  In
theory we could make the test more robust by setting up an actual
repository with a newline in it (so that our clone would succeed if our
new check didn't kick in). But a repo directory with newline in it is
likely not portable across all filesystems. Likewise, we could check
git-daemon's log that it was not contacted at all, but we do not
currently record the log (and anyway, it would make the test racy with
the daemon's log write). We'll just check the client-side stderr to make
sure we hit the expected code path.

Reported-by: Harold Kim <h.kim@flatt.tech>
Signed-off-by: Jeff King <peff@peff.net>
Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

master
v2.39.0 ··· v2.30.1


peff authored and **gitster** committed on Jan 7, 2021 1 parent 041bc65 commit a02ea577174ab8ed18f847cf1693f213e0b9c473

Showing **2 changed files** with **7 additions** and **0 deletions**.

Split Unified

2 connect.c

```
1063 1063                 target_host = xstrdup(hostandport);
1064 1064
1065 1065         transport_check_allowed("git");
     1066 +      if (strchr(target_host, '\n') || strchr(path, '\n'))
     1067 +              die(_("newline is forbidden in git:// hosts and repo paths"));
1066 1068
1067 1069         /*
1068 1070          * These underlying connection commands die() if they
```

5 t/t5570-git-daemon.sh

```
102 102         )
103 103   '
104 104
     105 + test_expect_success 'client refuses to ask for repo with newline' '
     106 +       test_must_fail git clone "$GIT_DAEMON_URL/repo$LF.git" dst 2>stderr &&
     107 +       test_i18ngrep newline.is.forbidden stderr
     108 + '
     109 +
```

```
105  110  test_remote_error()
106  111  {
107  112          do_export=YesPlease
```

0 comments on commit `a02ea57`

Please sign in to comment.