Cracking a Simple Passworded File: A Beginner's Guide to Security

By **Bilal Ahmad Khan** - May 2, 2016

Digital security has become very important in the past few years, with so much software being hacked everyday. This new trend has called attention to the need for better security with additional layers. This article is a beginner's guide to know how a simple file coded with c language, secured with a password, can be hacked. I will be running Ubuntu 14.04 LTS for the enitirety of this tutorial. **All the files used in this tutorial can be found on Github.**

Lets start with running the file to see what it does:

```
 $ ./crackme2
Access Denied
```

If you have been following my previous blog posts, the files used to show "usage" as to where we should write the password to get access; but this time things are a little bit different and we don't even know where to write the password. Anyway, let's run the file command to check if our file is stripped or not.

The File Command Determines The Filetype. Three Sets Of Tests Are Performed In This Order: Filesystem Tests, Magic Number Tests, And Language Tests.

```
 $ file crackme2
crackme2: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically li
```

Luckily, it's not stripped, which means our file still contains symbols. Lets start with doing an ltrace on this file.

Ltrace Is A Program That Simply Runs The Specified Command Until It Exits. It Intercepts And Records The Dynamic Library Calls Which Are Called By The Executed Process And The Signals Which Are Received By That Process. It Can Also Intercept And Print The System Calls Executed By The Program.

```
 $ ltrace ./crackme2
__libc_start_main(0x40087d, 1, 0x7ffc72840328, 0x400a70 <unfinished …>
strncmp("XDG_SESSION_ID=4", "Passw0rd=", 9) = 8
strncmp("TERM=xterm", "Passw0rd=", 9) = 4
strncmp("SHELL=/bin/bash", "Passw0rd=", 9) = 3
strncmp("SSH_CLIENT=10.0.2.2 23512 22", "Passw0rd=", 9) = 3
strncmp("OLDPWD=/home/vagrant/github/holb"…, "Passw0rd=", 9) = -1
strncmp("SSH_TTY=/dev/pts/0", "Passw0rd=", 9) = 3
strncmp("USER=vagrant", "Passw0rd=", 9) = 5
strncmp("LS_COLORS=rs=0:di=01;34:ln=01;36"…, "Passw0rd=", 9) = -4
strncmp("MAIL=/var/mail/vagrant", "Passw0rd=", 9) = -3
strncmp("PATH=/usr/local/sbin:/usr/local/"…, "Passw0rd=", 9) = -32
strncmp("PWD=/home/vagrant/github/holbert"…, "Passw0rd=", 9) = -10
strncmp("LANG=en_US.UTF-8", "Passw0rd=", 9) = -4
strncmp("NODE_PATH=/usr/lib/nodejs:/usr/l"…, "Passw0rd=", 9) = -2
strncmp("SHLVL=1", "Passw0rd=", 9) = 3
strncmp("HOME=/home/vagrant", "Passw0rd=", 9) = -8
strncmp("passw0rd=bilal", "Passw0rd=", 9) = 32
strncmp("LOGNAME=vagrant", "Passw0rd=", 9) = -4
strncmp("SSH_CONNECTION=10.0.2.2 23512 10"…, "Passw0rd=", 9) = 3
strncmp("LESSOPEN=| /usr/bin/lesspipe %s", "Passw0rd=", 9) = -4
strncmp("XDG_RUNTIME_DIR=/run/user/1000", "Passw0rd=", 9) = 8
strncmp("LESSCLOSE=/usr/bin/lesspipe %s %"…, "Passw0rd=", 9) = -4
strncmp("_=/usr/bin/ltrace", "Passw0rd=", 9) = 15
puts("Access Denied"Access Denied
) = 14
+++ exited (status 1) +++
```

From the looks of it, this program is accessing all the environment variables such as $HOME, $PWD, and trying to compare first 9 characters with Passw0rd=. We can easily recognize that this binary file is trying to search for an environment variable by the name Passw0rd. Lets have one with a random value and do an ltrace again.

```
 $ export Passw0rd=hey
$ ltrace ./crackme2
__libc_start_main(0x40087d, 1, 0x7fff93c86f38, 0x400a70 <unfinished …>
strncmp("XDG_SESSION_ID=4", "Passw0rd=", 9) = 8
strncmp("TERM=xterm", "Passw0rd=", 9) = 4
strncmp("SHELL=/bin/bash", "Passw0rd=", 9) = 3
strncmp("SSH_CLIENT=10.0.2.2 23512 22", "Passw0rd=", 9) = 3
strncmp("OLDPWD=/home/vagrant/github/holb"…, "Passw0rd=", 9) = -1
strncmp("SSH_TTY=/dev/pts/0", "Passw0rd=", 9) = 3
strncmp("passw0rd=bilal", "Passw0rd=", 9) = 32
strncmp("USER=vagrant", "Passw0rd=", 9) = 5
strncmp("LS_COLORS=rs=0:di=01;34:ln=01;36"…, "Passw0rd=", 9) = -4
strncmp("MAIL=/var/mail/vagrant", "Passw0rd=", 9) = -3
strncmp("PATH=/usr/local/sbin:/usr/local/"…, "Passw0rd=", 9) = -32
strncmp("PWD=/home/vagrant/github/holbert"…, "Passw0rd=", 9) = -10
strncmp("LANG=en_US.UTF-8", "Passw0rd=", 9) = -4
strncmp("NODE_PATH=/usr/lib/nodejs:/usr/l"…, "Passw0rd=", 9) = -2
strncmp("SHLVL=1", "Passw0rd=", 9) = 3
strncmp("HOME=/home/vagrant", "Passw0rd=", 9) = -8
strncmp("Passw0rd=hey", "Passw0rd=", 9) = 0
MD5_Init(0x7fff93c86da0, 0x400b06, 9, 6) = 1
strlen("hey") = 3
MD5_Update(0x7fff93c86da0, 0x7fff93c88f2c, 3, 0x7fff93c88f2c) = 1
```

```
MD5_Final(0x7fff93c86e00, 0x7fff93c86da0, 0x7fff93c86da0, 6) = 1
sprintf("60", "%02x", 0x60) = 2
sprintf("57", "%02x", 0x57) = 2
sprintf("f1", "%02x", 0xf1) = 2
sprintf("3c", "%02x", 0x3c) = 2
sprintf("49", "%02x", 0x49) = 2
sprintf("6e", "%02x", 0x6e) = 2
sprintf("cf", "%02x", 0xcf) = 2
sprintf("7f", "%02x", 0x7f) = 2
sprintf("d7", "%02x", 0xd7) = 2
sprintf("77", "%02x", 0x77) = 2
sprintf("ce", "%02x", 0xce) = 2
sprintf("b9", "%02x", 0xb9) = 2
sprintf("e7", "%02x", 0xe7) = 2
sprintf("9a", "%02x", 0x9a) = 2
sprintf("e2", "%02x", 0xe2) = 2
sprintf("85", "%02x", 0x85) = 2
strcmp("d8578edf8458ce06fbc5bb76a58c5ca4"…, "6057f13c496ecf7fd777ceb9e79ae285
puts("Access Denied"Access Denied
) = 14
+++ exited (status 1) +++
```

Woa! This opened a whole new door for us. So after finding out the Passw0rd environment variable, it gets the string length for the value of that environment variable, and from the looks of it calculates an MD5 hash and compares it with a predefined md5 hash in the program.
Let’s convert our input password to an MD5 hash to confirm it. Just a simple google search would land you with so many md5 hash calculators, you can even do it using bash:

```
 $ printf "hey" | md5sum | cut -d' ' -f1
6057f13c496ecf7fd777ceb9e79ae285
```

So this MD5 hash is exactly what you can see in the ltrace output for our input password, and it is comparing it with a predefined md5 hash, **d8578edf8458ce06fbc5bb76a58c5ca4,** using strcmp. Now we need some way to reverse this to a string.

> *It Is Not Possible To Retrieve The Original String From A Given MD5 Hash Using Only Mathematical Operations, But There Is A Way To Decrypt A MD5 Hash, Using A Dictionary Populated With Strings And Their MD5 Equivalent (Or We Can Try Brute Forcing As Well). As Most Users Use Very Simple Passwords (Like “123456”, “Password”, “Abc123”, Etc), MD5 Dictionaries Make Them Very Easy To Convert Them Back To A String. MD5 Reverse Dictionary Having Several Millions Of*

Entries Can Be Used To Convert An MD5 Hash To A String.

Let's use this website and try converting this hash to a string. It results in an output of "qwerty". Lets try our luck with this password.

```
 $ export Passw0rd=qwerty
$ ./crackme2
Access Granted
```

Tada! We finally have the password and we didn't even need tools like gdb for it.

Taking it one step further, we can write a simple ruby script containing only lowercase alphabets to try to brute-force the MD5 string. *This script assumes the password string contains only lower-case letters.*

```
 #!/usr/bin/ruby
require 'digest/md5'
?a.upto('zzzzzzz') { |string|
md5=Digest::MD5.hexdigest(string)
puts "Trying: #{string}"
if md5 == "d8578edf8458ce06fbc5bb76a58c5ca4"
   puts "nThe string is #{string}n"
   exit
end
}
```

Run this script, and then go watch a movie or sleep. After quite a bit of time, this script will tell us that the string for that specific MD5 hash is "qwerty".
P.S. We can also use an equivalent bash script for brute forcing, but as it turns out it was too slow. I have uploaded the bash brute force script in my github repo as well in the link given above.

This article was contributed by a student at Holberton School and should be used for educational purposes only.

Bilal Ahmad Khan