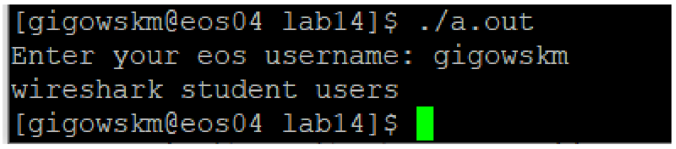
CIS 452
Lab 14 Report

Ashley Hendrickson
Muna Gigowski
Fall 2019

Unvalidated Input

1. What is the program intended to do (use the man pages if need be to explain *how* it works)?

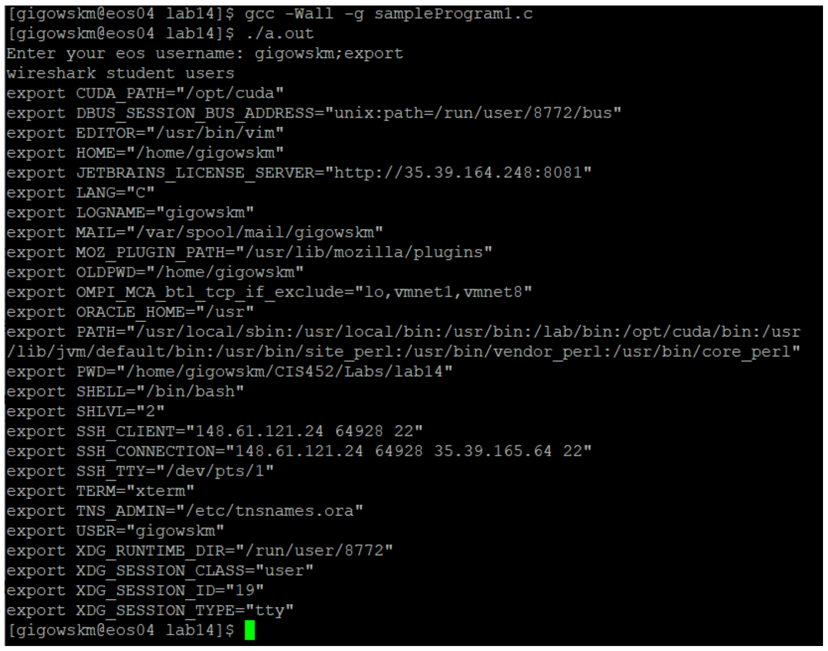
This program attempts to leverage the current user's eos username to run a system command using their account. It first starts by initializing two char arrays - one that will hold the username and one for holding the command. The program prints to the screen to ask the current user for their eos username, then stores the input in one of the char arrays. The program then calls sprintf(), passing in the empty command char array, the string "groups %s", and the username that was given from input; sprinf() takes the second argument string and places the third argument in place of %s, then stores the new string into the command char array buffer (argument one). It then passes the command string into a system() function call, which executes the given command argument - "groups gigowskm," in our case. This command then prints the names of the primary, and any supplementary groups, for the given username, which are shown in our screen snapshot below.



```
[gigowskm@eos04 lab14]$ ./a.out
Enter your eos username: gigowskm
wireshark student users
[gigowskm@eos04 lab14]$
```

2. Describe what happened, and *why* it occurred.

In running this program using the additional ";export" string, we are effectively adding a second command to what system() executes; in this case, we are telling it to run the "export" command after the "groups [username]" command, which generates and displays all exported variables for that user account. In the screenshot below you can see the list of all exported variables.



```
[gigowskm@eos04 lab14]$ gcc -Wall -g sampleProgram1.c
[gigowskm@eos04 lab14]$ ./a.out
Enter your eos username: gigowskm;export
wireshark student users
export CUDA_PATH="/opt/cuda"
export DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/8772/bus"
export EDITOR="/usr/bin/vim"
export HOME="/home/gigowskm"
export JETBRAINS_LICENSE_SERVER="http://35.39.164.248:8081"
export LANG="C"
export LOGNAME="gigowskm"
export MAIL="/var/spool/mail/gigowskm"
export MOZ_PLUGIN_PATH="/usr/lib/mozilla/plugins"
export OLDPWD="/home/gigowskm"
export OMPI_MCA_btl_tcp_if_exclude="lo,vmnet1,vmnet8"
export ORACLE_HOME="/usr"
export PATH="/usr/local/sbin:/usr/local/bin:/usr/bin:/lab/bin:/opt/cuda/bin:/usr
/lib/jvm/default/bin:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl"
export PWD="/home/gigowskm/CIS452/Labs/lab14"
export SHELL="/bin/bash"
export SHLVL="2"
export SSH_CLIENT="148.61.121.24 64928 22"
export SSH_CONNECTION="148.61.121.24 64928 35.39.165.64 22"
export SSH_TTY="/dev/pts/1"
export TERM="xterm"
export TNS_ADMIN="/etc/tnsnames.ora"
export USER="gigowskm"
export XDG_RUNTIME_DIR="/run/user/8772"
export XDG_SESSION_CLASS="user"
export XDG_SESSION_ID="19"
export XDG_SESSION_TYPE="tty"
[gigowskm@eos04 lab14]$
```

In running the program using the additional ";set" string, we are again adding a second command to what system() executes, this time telling it to run the "set" command after the original "groups [username]" command. With no given arguments, the "set" command displays the names and values of all shell variables and functions. The list of these variables and functions that displayed for us on the command line after running the program with ";set" was too long to add a screenshot.

3. What is some potentially dangerous information that an attacker could learn about the system with this approach? Give a *specific* example of information that might be useful in a targeted attack?

By typing ";set" or ";export" it allows us to find information about the current system. We can find what versions of bash the system is using, and if we can find what version(s) they are using then we can tell if they are running outdated versions of bash. Now we can research possible security vulnerabilities to that version of bash. Not to mention that after you type ';' you have complete access to the terminal and when you have access to the terminal without an account your system is compromised.

Buffer Overflows

4. What warning does the compiler give? What is the program intended to do (use the man pages if need be to explain *how* it works)?

The compiler gives the following warnings:

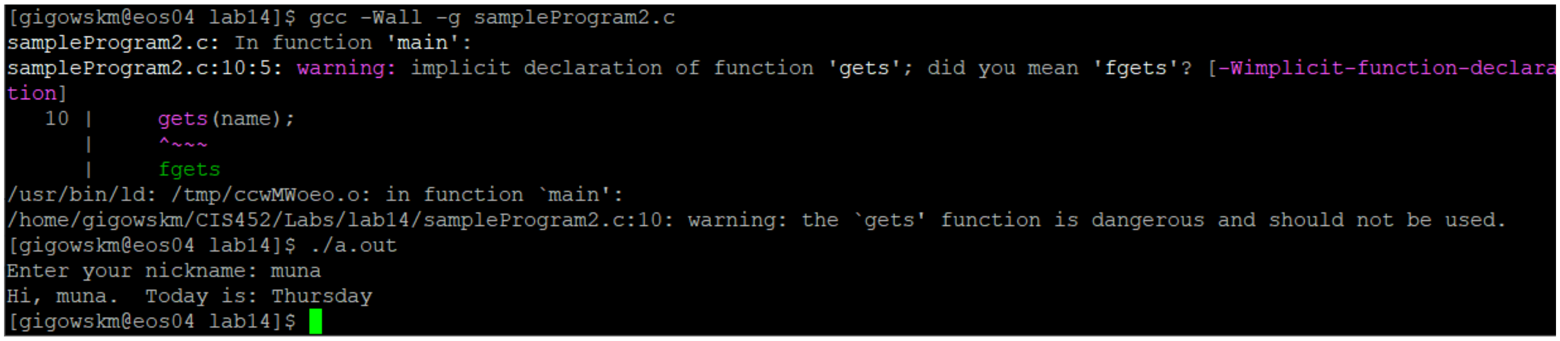
warning: implicit declaration of function 'gets'; did you mean 'fgets'?

warning: the `gets' function is dangerous and should not be used.

This program aims to simply take in the user's nickname, then print out "Hello, [nickname]" back out to them, along with the current day of the week. It does this by first declaring two char arrays - one called "name" to hold the nickname input later, and one named "command" that is instantiated to a string command to get the current day of the week. The program then asks the user for their nickname and places the input into the "name" char array using gets(). It then uses the given nickname to print out a greeting to the user before flushing the output buffer, calling system() with the command string argument that gets the day of the week, and displaying it to the user after the greeting message before finally returning.

The program's purpose is innocent enough, but its method of execution is what makes it dangerous. Since the program uses gets() instead of fgets(), there is an opportunity here for a hacker to enter a nickname that exceeds the declared buffer size (32 bytes, in this case), filling up the additional space with any malicious content that they choose.

PROGRAM RUN OUTPUT:

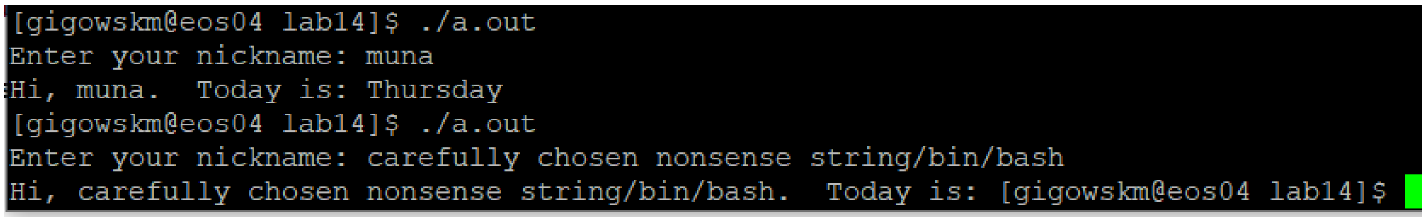


```
[gigowskm@eos04 lab14]$ gcc -Wall -g sampleProgram2.c
sampleProgram2.c: In function 'main':
sampleProgram2.c:10:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declara
tion]
   10 |     gets(name);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/ccwMWoeo.o: in function `main':
/home/gigowskm/CIS452/Labs/lab14/sampleProgram2.c:10: warning: the `gets' function is dangerous and should not be used.
[gigowskm@eos04 lab14]$ ./a.out
Enter your nickname: muna
Hi, muna.  Today is: Thursday
[gigowskm@eos04 lab14]$
```

5. Describe what happened, and *why* it occurred.

What is happening here is that we are entering a nickname that exceeds the declared buffer size of 32 bytes; the section of the nickname string “carefully chosen nonsense string” is exactly 32 bytes, so anything that is entered beyond that is considered overflow. In this case, the overflow is the section of the nickname string “/bin/bash,” which opened up a new shell window. This occurred because, when the nickname string experienced the buffer overflow, that overflowed portion of the string “leaked” into the space that the char array for the command was supposed to have reserved, effectively replacing the data that was originally there. Now, instead of calling system() with the command to get the current day, system() was getting called with the argument “/bin/bash,” causing the system to open a new shell.

PROGRAM OUTPUT USING "carefully chosen nonsense string/bin/bash":



```
[gigowskm@eos04 lab14]$ ./a.out
Enter your nickname: muna
Hi, muna.  Today is: Thursday
[gigowskm@eos04 lab14]$ ./a.out
Enter your nickname: carefully chosen nonsense string/bin/bash
Hi, carefully chosen nonsense string/bin/bash.  Today is: [gigowskm@eos04 lab14]$
```

6. Suppose a program like this was running on a kiosk, providing simple information to users in a friendly manner. Describe how an attacker could use this approach to execute commands on the kiosk system, even if they had no login access.

You can escape the program and execute commands on the terminal. To verify this I added an infinite while loop in the program and I was able to escape it start running commands from the shell. Since I can access the shell from this program without an account this completely compromises the system.