Experiment 1
Buffer Overflow Attacks

The goal of this experiment is to investigate how to carry out a buffer overflow attack.

Case Study

The goal of this experiment is to investigate how to carry out a buffer overflow attack.

In computer security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations. This is a special case of the violation of memory safety.

Buffer overflows can be triggered by inputs that are designed to execute code or alter the way the program operates. This may result in erratic program behavior, including memory access errors, incorrect results, a crash, or a breach of system security. Thus, they are the basis of many software vulnerabilities and can be maliciously exploited.

Programming languages commonly associated with buffer overflows include C and C++, which provide no built-in protection against accessing or overwriting data in any part of memory and do not automatically check that data written to an array (the built-in buffer type) is within the boundaries of that array. Bounds checking can prevent buffer overflows.

Theory Background

Buffer is a contiguous block of computer memory that holds multiple instances of the same data type. Buffer overflow is a kind of attack that is executed by means of software but it results in a violation of memory safety.

A buffer overflow occurs when data written to a buffer also corrupts data values in memory addresses adjacent to the destination buffer due to insufficient bounds checking. This can occur when copying data from one buffer to another without first checking that the data fits within the destination buffer. Stack overflow, heap overflow and integer overflow are various kinds of buffer overflow attacks.

A simple example of the buffer overflow attack is shown below. A program has two data items which are adjacent in memory: an 8-byte-long string buffer, A, and a two-byte integer, B.

```
char           A[8] = "";
unsigned short B    = 1979;
```

Initially, A contains nothing but zero bytes, and B contains the number 1979.

variable name	A	B
value	[null string]	1979

Now, the program does not check the incoming data length to be stored at A. Suppose an attacker sends a 9-character long string that encodes to 10 bytes to store in A but A can take only 8 bytes. By failing to check the length of the string, it also overwrites the value of B as shown below.

variable name	A								B
value	'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	25856

Integer overflow is something that occurs when an arithmetic operation tries to create a numeric value that is too large in given (available) storage. Adding 1 to largest value is integer overflow. The result will be the largest negative. Arithmetic overflow occurs when any arithmetic calculation produces a result that is great than storage space. One way is to check overflow bit. Students will get into details of those and through executing code they will get an idea how exactly buffer overflow can be attacked and malicious code may be injected.

Experiment Set-up: Configuration

This experiment will require a computer with Linux system.

You can get access to Department of ECE's Linux server through SSH clients, such as "SSH Secure Shell Client" or "Putty".

Download and install SSH client on your PC, and configure as below.

The host name is "linux.ece.ufl.edu".

The user name is your gatorlink username as shown in Figure 1. You will be required to input your password for your gatorlink.

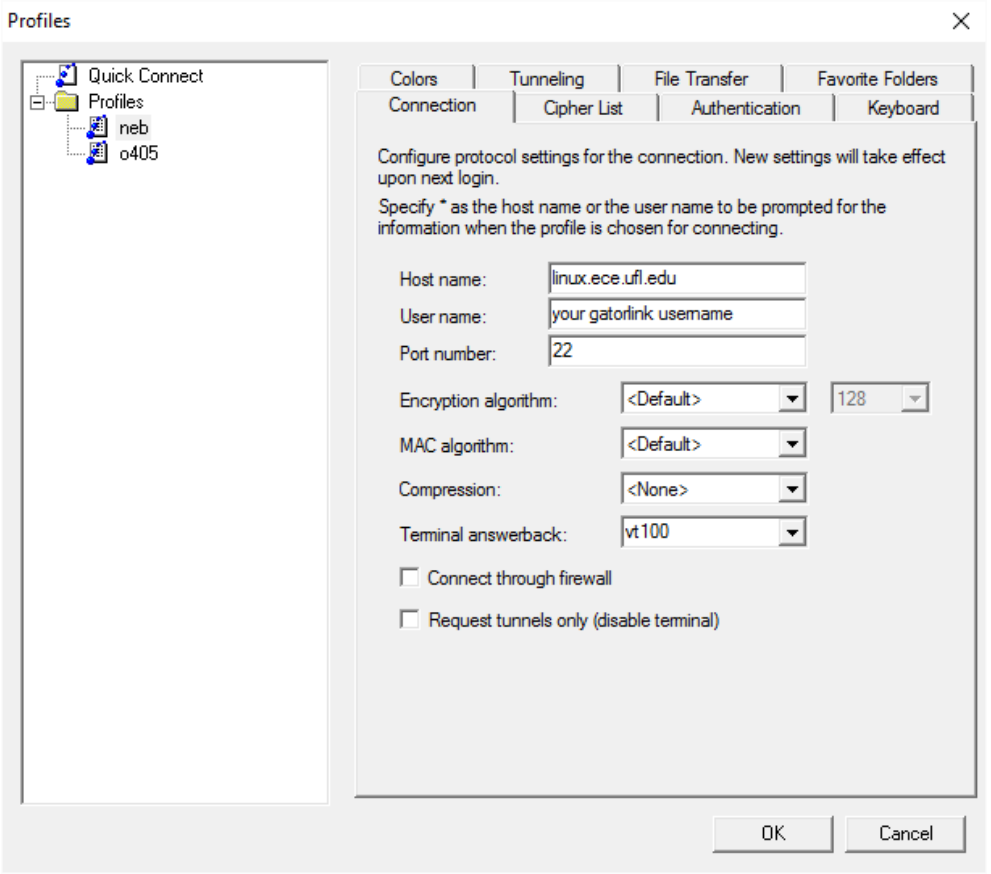

Figure 1 SSH configuration when using SSH Secure Shell Client.

Experiment Set-up: Instructions

Students will get into details of those and through executing code they will get an idea how exactly buffer overflow can be attacked and malicious code may be injected.

General Overview of what to do, simple examples

Students should code buffer overflow samples for heap overflow, integer overflow and stack overflow. The task given below will surely help you to understand how through input buffer will overflow and how it can be exploited. Also, execution path of running process may be changed by overwriting the return address with the address that points back to buffer where malicious code has been injected and that code will be executed.

Specific Challenge

All we demonstrate in this experiment is what one can do by overflowing buffer. If you overflow buffer even if you enter the wrong password still you may get access. This experiment will make things pretty clear. It is a perfect example of bad coding and how can one exploit it. Anyone who knows that program has buffer overflow can exploit it and that's perfectly demonstrated.

Steps for execution:

Step 1: Download ccode.c code and store it in a folder

Step 2: In Linux compile the code using: gcc -w -fno-stack-protector ccode.c -o ccode

Step 3: Run it: ./ccode

For more information on stack protector use references. Note: the key is "gainesville".

Measurement, Calculation, and Question

Try following inputs:

1) What is output when you enter correct key = gainesville
2) What is output when you enter a wrong key but characters will be like 10-12 for e.g.: ghgjhjhjkh
3) What is output when you enter long wrong key for e.g.: ghgjhkjhkjkhkhkjhkjlkjlkjkjlkjlkjlkjkljkjlkjk
4) Write your conclusions.
5) Why do all these happen?
6) How can you correct it?

C program which demonstrates how buffer will overflow and how malicious code will be injected. Follow steps and complete task.

Step 1: Unzip buffer_overflow folder

Step 2: In Linux compile the code using: gcc -w -fno-stack-protector bo_test.c -o bo_test

Step 3: Run it with any 5-letter argument or any number of letters less than 10. For eg: ./bo_test abcde

Step 4: you will get stack address of good code, malicious code, copy address of malicious code

Step 5: Copy that address in Perl script just write that address

Step 6: Run Perl code: Perl ./run.pl

Now buffer overflow attack has occurred and you can see output just take a screenshot and write conclusions. Various c codes are given. Run them and try to implement those as system call (– shutdown) was executed in previous one. Submit screenshots of current programs and suggest ways to improve it.

Optional Follow-up

Advanced Techniques (optional for students to do: Student will get extra credit if they work on it)

Use of Valgrind tool helps in preventing and detecting buffer overflow attack by proper memory auditing. Students who are interested can follow user manual and try it and submit their work.

Prevention Techniques

1) Proper and secure coding
2) Avoid using Library which are unsafe
3) Avoiding Buffer Overflows
    a. Compiler based run time bounds checking
    b. Library based runtime bounds checking
        i. Libsafe
        ii. Libverify
        iii. Libparanoia

☞ Lab Report Guidelines

1. In your report, give the results when you ran the code with different inputs.
2. Include screenshots for all the steps.
3. Give answers to all the questions.
4. Give suggestions for improving the program.

References and Further Reading

[1] http://www.outflux.net/blog/archives/2014/01/27/fstack-protector-strong/

[2] http://www.slideshare.net/aidanshribman/valgrind-29203055

[3] http://pages.cs.wisc.edu/~bart/537/valgrind.html

[4] http://www.freebsd.org/doc/en/books/developers-handbook/secure-bufferov.html

[5] http://www.outflux.net/blog/archives/2014/01/27/fstack-protector-strong/

[6] http://pages.cs.wisc.edu/~bart/537/valgrind.html

[7] http://www.cs.colostate.edu/~massey/Teaching/cs356/RestrictedAccess/Slides/356lecture21.pdf

[8] http://projects.webappsec.org/w/page/13246946/Integer%20Overflows