# Stack Smashing Introduction

http://tinyurl.com/9uh458s

The programs and techniques presented in this guide are found in "Hacking: The Art of Exploitation" by Jon Erickson. I highly recommend this book if you would like to expand on this introduction to the topic.

**Useful GDB Debugger Commands**

```
gdb <program name>                    * start up debugger from the command line

(gdb) list 1                          * list source code, hit <enter> to
continue listing
(gdb) break 10                  * set break point at line 10
(gdb) run <parameters>          * start running program, enter parameters
(gdb) step                      * run the next line of code and pause
(gdb) print $esp                * display values of registers or variables
(gdb) x /32xw <address>         * examine a block of memory in hex
(gdb) set {int}<addr> = <val>   * change the value of a memory location
(gdb) disassemble main          * see the assembly code (includes memory
address!)
```

**How to input shellcode on the command line using perl**

```
# Use Perl to generate strings (with repeated patterns)
perl -e 'print "AAAAAAAA"'
perl -e 'print "AAAA" . "BBBB"'
perl -e 'print "AAAA" x 4'
perl -e 'print "\xf0\xd0\xff\xff"'                    * enter address
xffffd0f0
perl -e 'print "\xf0\xd0\xff\xff"x20'                       * repeat address
20 times
perl -e 'print "\x90" x 200'                          * NOP repeated 200 times

# Enter pw argument on command line using Perl - surround with left tick above
tab key
./auth_overflow3 `perl -e 'print "this is the place"'`
```

**Compiler Flags**

Compiler options must be used to turn off buffer overflow defenses
(see contents of the file compiler_flags)

Use the following form of compiler options to compile example c programs
gcc -g -m32 -fno-stack-protector -z execstack -o <executable file>  <source file>

Here is how you compile the auth_overflow3.c program to auth_overflow3 executable
gcc -g -m32 -fno-stack-protector -z execstack -o auth_overflow3 auth_overflow3.c

**Beginners Tutorial**

Download sample tar file at http://faculty.cs.byu.edu/~seamons/sample.tar
tar -xvf sample.tar * command to untar the files on Linux

**Part 1**

**Goal: Gain access without a valid password, understand the runtime stack**

1) Study the auth_overflow1.c program and see the valid passwords
2) Try different passwords to see if you can cause unexpected behavior
   - gain access without a valid password
   - gain access without a valid password, then program crashes
   - program crashes without gaining access
3) Use the debugger and examine the stack to understand why the errors occur
4) Be able to explain how the stack works.
   - ebp, esp, return address, local variables
5) Changing the code as shown in auth_overflow2.c doesn't fix the problem. This compiler places local variables on the stack in the same order.

**Part 2**

**Goal: Use the debugger to obtain access by overwriting the return address**

1) Study auth_overflow3.c, compile it
2) This program no longer provides access for an invalid password
3) Use the debugger to force the program to grant access without a valid pw
   - overwrite the return address on the stack
   - Use "disassemble main" to determine new return address

**Part 3**

**Goal: Gain access using only the command line**

1) Now cause the program to grant access without using the debugger.
2) Input data on the command line only so that access is granted.
   - Overwrite return address on the stack
   - Program will crash after access is granted. That is ok. Damage is done.

**Part 4 - Inject shellcode on the stack and execute it**

1) Compile and run shellcode5.c to verify it works on your platform.
   The binary shellcode is also included in shellcode5.bin.
   This is the binary data you must place on the stack.
   You can add `cat shellcode5.bin` on command line to insert this in the pw parameter
2) Now input the shellcode and cause it to be executed.
   You may need to use a NOP sled.
   Use the debugger to estimate the address used to overwrite the return address.
   Use Perl to help you enter the binary data as a string.

**Part 5 - Use environment variable to inject shellcode on the stack**

1) Environment variables are also included on the stack.
   - Find environment variables on the stack in the debugger
     x /s $esp  (then hit return 30-50 times watching for env)
     or find the address of the command line parameter
   - Input shellcode in an environment variable
     for example, for in my /bin/tcsh shell I can say
         setenv SHELLCODE `cat shellcode5.bin`
     figure out how to do this in your shell
   - Run the program in the debugger and find shellcode on the stack
   - Force the shellcode to execute
2) Once you can do it in the debugger, now try it from only the command line
3) You may need to add a NOP sled to the environment variable (use Perl)

**Recommended Reading**

The original article that brought stack smashing into the mainstream:
*Smashing The Stack For Fun And Profit*
http://insecure.org/stf/smashstack.html