

Computer Security "Smashing the Stack" SPIM Homework
For CS2214 at Polytechnic University
By Michael Aiello
1/27/05

Introduction:

On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind. (Phrack 49).

A vulnerability that leads to the ability to overwrite the return address in a function instance on the stack has traditionally been the most commonly exploited "hole" in computer applications. It arises from a combination of insufficient input validation, improper memory handling, and indistinguishable data and instruction sections in memory. Several attempts have been made to mitigate these causes, however new stack smashing attacks still are discovered and exploited regularly across operating systems and platforms.

We will be using the SPIM simulator (a MIPS simulator) to learn about the stack and simulate "smashing" in order to execute arbitrary code.

The Assignment:

You will be provided with a template SPIM program that correctly calls a subroutine (blob), stores a parameter on to the stack, performs a trivial print operation, recalls the return address from the stack and returns to the main program.

Your job is to (1)create custom parameter for blob, and (2)adjust the way that input is stored on to the stack in such a way that the return address is overwritten with the address of a "hidden function" that is included at the end of the template. This will cause the instructions in the "hidden function" to be executed rather than returning to the main program.

(please note, this is **not** the standard method for using stack smashing to execute arbitrary code, in the real world, "shell code" machine instructions are included in the input parameter that are jumped to. The SPIM assembler prevents stack memory from being executed as instructions, so we have written a "hidden function" which is called. See Phrack 49.)

Rules:

1. There are sections marked as "MUST NOT CHANGE." You are not allowed to change code in these sections. (A diff script will be run against your homework, if anything in these sections is changed, we will know)
2. You won't get any credit if you just insert code to directly jump to the address of the hidden function or if you manipulate the return address register directly.

3. No copying homework from others, do this exercise yourself. (Another script will be run against the whole class that can detect this and it is smart enough to detect slightly different homework.)

Suggested Approach:

First, open up the template in SPIM and run it. It should happily respond with.

```
Hello World
HAH, my stack is stronger than your string.
```

Now, run the program again and step through it one instruction at a time(F10). Watch how the function is called and understand how stack space is allocated, and how the program is only supposed to copy the first 10 bytes of data from the first parameter.

Next, start building your smash_space data. This is the data that is passed into the blob function. It is up to you how to fill this memory; use loops, load it one byte at a time using several load immediates, etc.

Next, run the program again one instruction at a time and watch how the stack is built, use strategic data so you can see which part of your input is going where in the stack (important to correctly align the smashed return address when we overwrite it)

```
STACK
[0x7ffffefec] 0x11110000
[0x7ffffeff0] 0x22221111 0x33332222 0x004000dc 0x00000000
```

Now you have to figure out how to overwrite the return address, that’s the fun part. Here I have successfully set the return address register to dead1337.

```
General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 10010000 R9 (t1) = dddddddd R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000004 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 10010064 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
R5 (a1) = 1001000f R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7ffffef74
R6 (a2) = 00000000 R14 (t6) = 00000000 R22 (s6) = 00000000 R30 (s8) = 00000000
R7 (a3) = 00000000 R15 (t7) = 00000000 R23 (s7) = 00000000 R31 (ra) = dead1337
```

After you are able to successfully overwrite the return address, you should overwrite it with the address of the first instruction of hidden function. This will cause the program to “return” to our defined location (our hidden function) and execute the code there.

Once that is accomplished, run the program and the following should be output.

```
Hello World
You have smashed the stack! Go forth... y0U H4X0R
```

At this point, you’re done. If you have followed the rules, you will get full credit.