

# Memory, Part 3: Smashing the Stack Example

Alex Kizer edited this page on Feb 15 · 2 revisions

Each thread uses a stack memory. The stack 'grows downwards' - if a function calls another function, then the stack is extended to smaller memory addresses. Stack memory includes non-static automatic (temporary) variables, parameter values and the return address. If a buffer is too small some data (e.g. input values from the user), then there is a real possibility that other stack variables and even the return address will be overwritten. The precise layout of the stack's contents and order of the automatic variables is architecture and compiler dependent. However with a little investigative work we can learn how to deliberately smash the stack for a particular architecture.

The example below demonstrates how the return address is stored on the stack. For a particular 32 bit architecture [Live Linux Machine](#), we determine that the return address is stored at an address two pointers (8 bytes) above the address of the automatic variable. The code deliberately changes the stack value so that when the input function returns, rather than continuing on inside the main method, it jumps to the exploit function instead.

```
// Overwrites the return address on the following machine:
// http://angrave.github.io/sys/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void breakout() {
    puts("Welcome. Have a shell...");
    system("/bin/sh");
}

void input() {
    void *p;
    printf("Address of stack variable: %p\n", &p);
    printf("Something that looks like a return address on stack: %p\n", *((&p)+2));
    // Let's change it to point to the start of our sneaky function.
    *((&p)+2) = breakout;
}

int main() {
    printf("main() code starts at %p\n",main);

    input();
    while (1) {
        puts("Hello");
        sleep(1);
    }

    return 0;
}
```

Edit

New Page

▼ Pages 51

[Home](#)

[#Example Markdown](#)

[#Informal Glossary](#)

[#Piazza: When And How to Ask For Help](#)

[C Programming, Part 1: Introduction](#)

[C Programming, Part 2: Text Input And Output](#)

[C Programming, Part 3: Common Gotchas](#)

[C Programming, Part 4: Debugging](#)

[Deadlock, Part 1: Resource Allocation Graph](#)

[Deadlock, Part 2: Deadlock Conditions](#)

[File System, Part 1: Introduction](#)

[File System, Part 2: Files are inodes \(everything else is just data...\)](#)


[File System, Part 3: Permissions](#)

[File System, Part 4: Working with directories](#)

[File System, Part 5: Virtual file systems](#)

Show 36 more pages...

Clone this wiki locally

 Clone in Desktop

[Commons License](#). If you are not the copyright holder, please give proper attribution and credit to existing content and ensure that you have license to include the materials.

