# Address Sanitizer

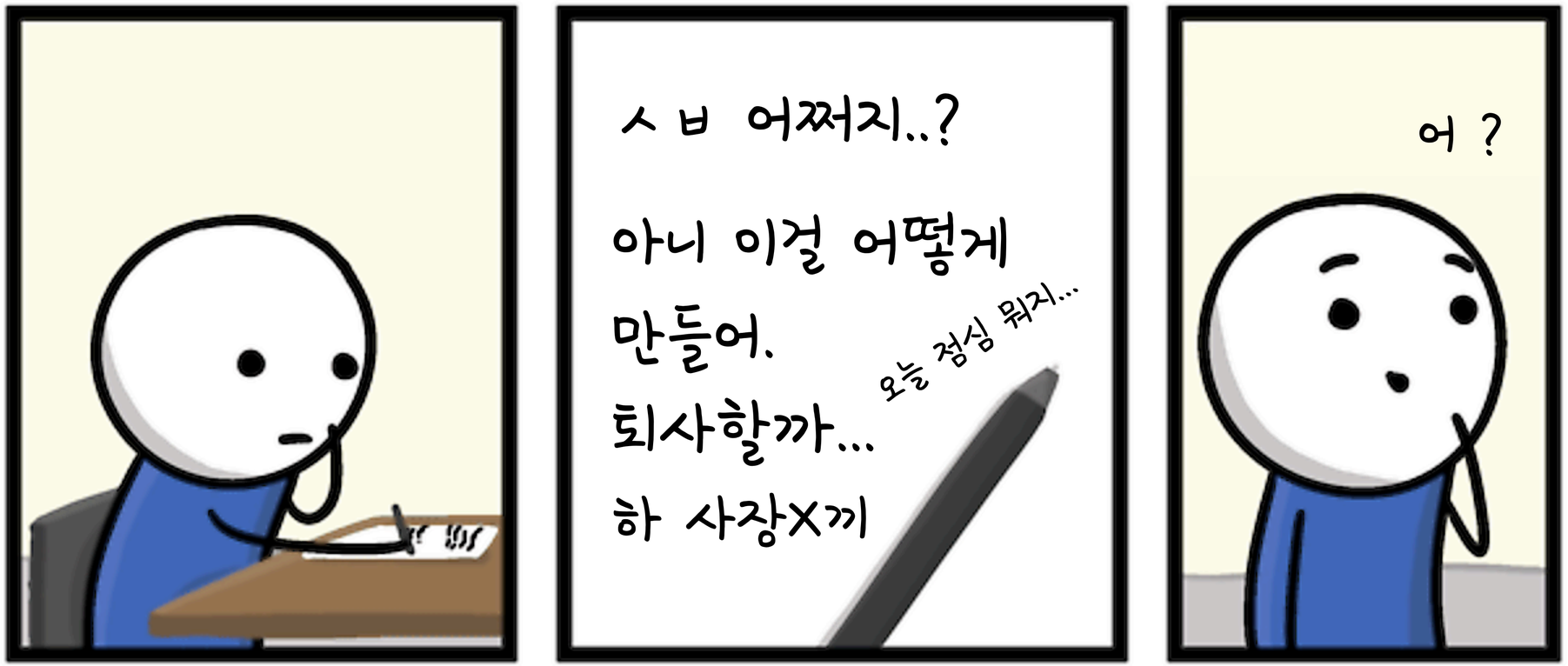
willwayy

2018.07.26

# Memory Bug

- Stealing information - Heartbleed
- Remote code exection - Shellshock, glibc
- Privilege escalation - Shellshock

# How ?



Google made it!

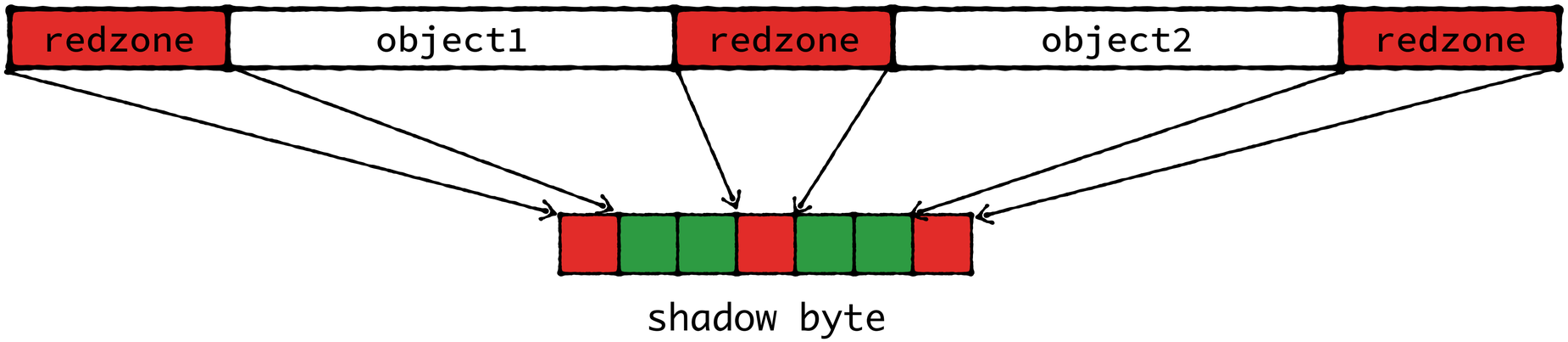
# Address Sanitizer

# *Address Sanitizer?*

- Address Sanitizer is a vulnerability detection tool provided by Google.
- It is the best defense against dynamic memory error vulnerability.
    - It detects various memory errors.
        - Use After Free (dangling pointer dereference)
        - Heap / Stack / Global buffer overflow
        - Memory leak, etc...

# How it works?

- To verify the validity of memory access, a special space called redzone is inserted between memory objects
    - To manage these redzones, we maintain a data structure called a shadow byte.
- Check the validity of the corresponding memory address each time the command accessing the memory is executed.



# ASAN env,useage

- Clang (version 3.1 or later) and GCC (version 4.8 or later)

```
clang -o test test.c -fsanitize=address
gcc -o test test.c -fsanitize=address
```

  - For Android, it will be implemented in the latest version of clang-3.5 or later.

# Other Sanitizer

- Leak Sanitizer
  - Heap leak detection during execution.
- Thread Sanitizer
  - Detect data races during execution.
- Memory Sanitizer
  - Detects references to uninitialized memory during execution.
- Undefined Behavior Sanitizer (UBSan)
  - Detects undefined behavior during execution.

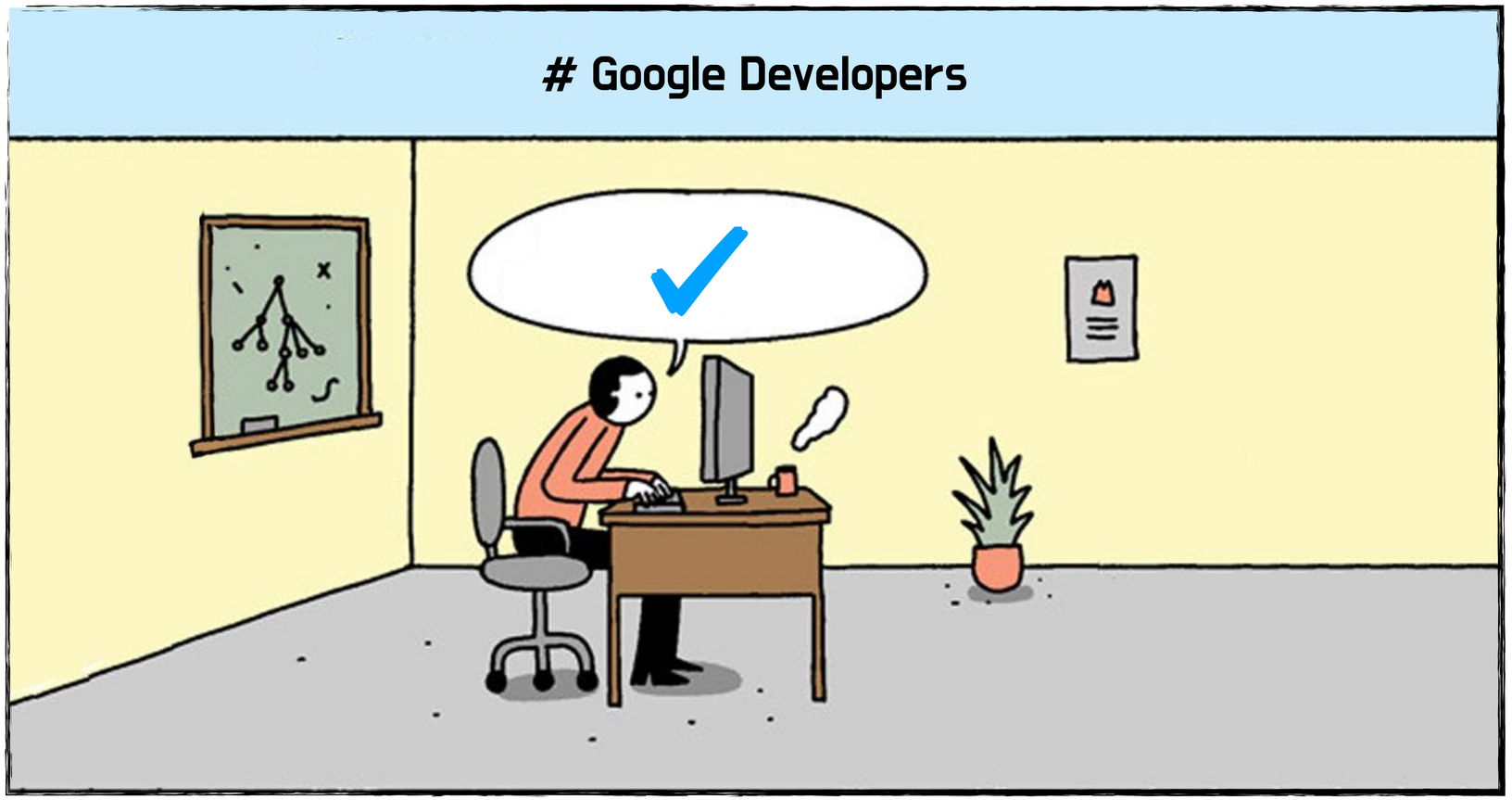
# ASAN example

```
int main() {
  char *a = malloc(1);
  a[1] = 1;
}
```

# ASAN example

```
int main(int argc, char *argv[]) {
  int size = atoi(argv[1]);
  char *a = malloc(size);
  a[size] = 1;
}
```

# ASAN example



# ASAN example

```
int main(int argc, char *argv[]) {
  int size = atoi(argv[1]);
  char *a = malloc(size);
  a[size] = 1;
}
```

```
a[size] = 1;
```

# ASAN example

```c
int main(int argc, char *argv[]) {
  int size = atoi(argv[1]);
  char *a = malloc(size);
  a[size] = 1;
}
```

```c
char *tmp = a + size;

if(SHADOW[tmp / 8]) check_slowpath(tmp);

a[size] = 1;
```

# ASAN example

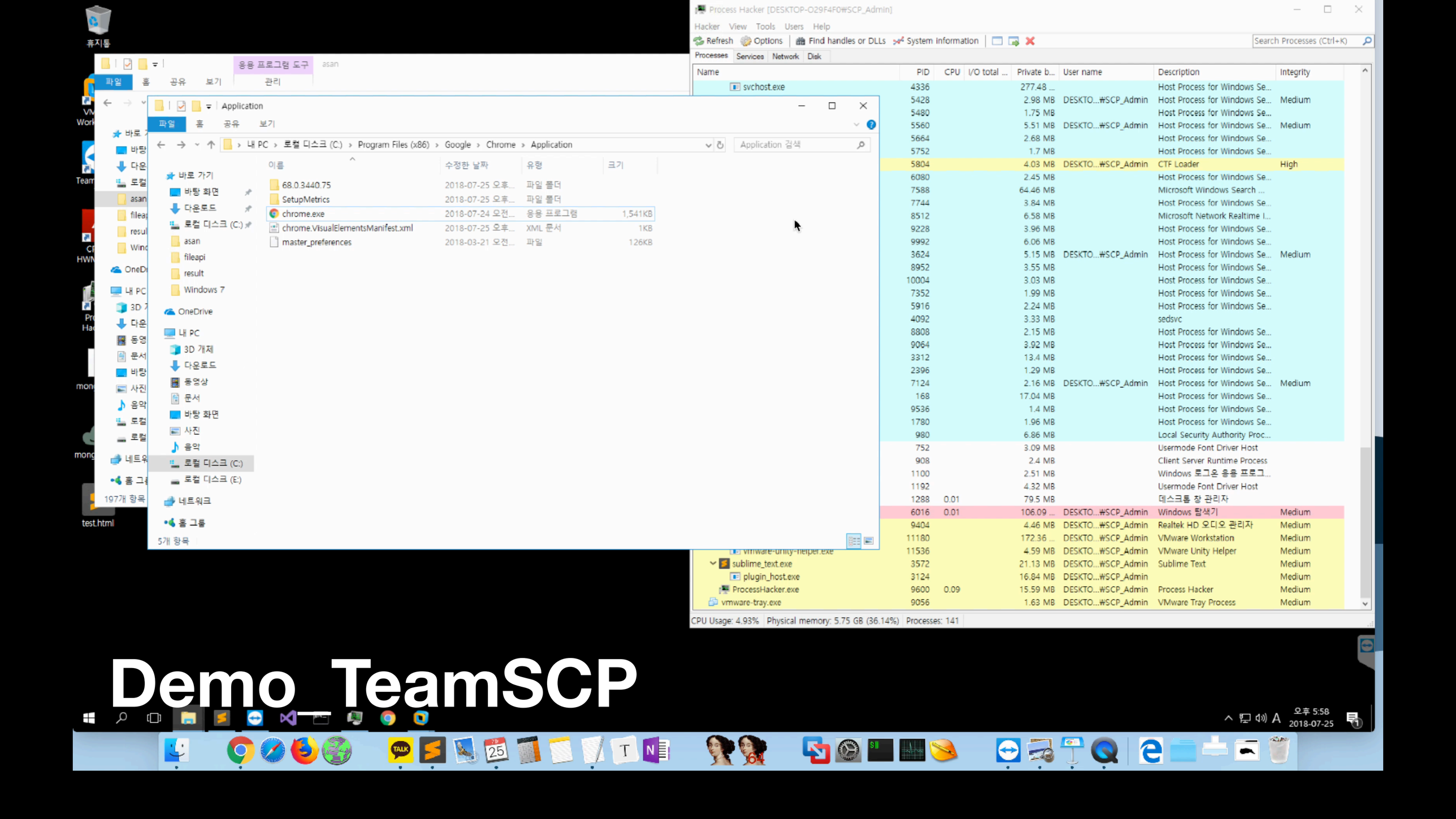
```
=================================================================
==96233==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60200000eff1 at pc 0x000000400873 bp 0x7ffc054f3070 sp 0x7ffc054f3060
WRITE of size 1 at 0x60200000eff1 thread T0
    #0 0x400872 in main /tmp/asan.c:4
    #1 0x7f159be3182f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #2 0x400708 in _start (/tmp/asan+0x400708)

0x60200000eff1 is located 0 bytes to the right of 1-byte region [0x60200000eff0,0x60200000eff1)
allocated by thread T0 here:
    #0 0x7f159c273602 in malloc (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x98602)
    #2 0x40082c in main /tmp/asan.c:3
    #2 0x7f159be3182f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)

SUMMARY: AddressSanitizer: heap-buffer-overflow /tmp/asan.c:4 main
Shadow bytes around the buggy address:
  0x0c047fff9da0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9db0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9dc0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9dd0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9de0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c047fff9df0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa[01]fa
  0x0c047fff9e00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9e10: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9e20: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9e30: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9e40: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
```

# ASAN

- Address Sanitizer is mainly used for Fuzzing




# ***Video***


Process Hacker [DESKTOP-O29F4F0#SCP_Admin]
Application
Demo_TeamSCP

# *QnA*