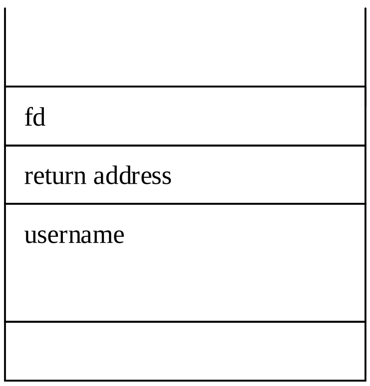
Note: March 6, 2007

Buffer Overflow: System Solution

Stack Guard:

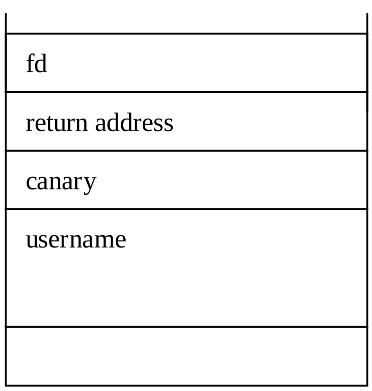
Example code with buffer overflow/

```
void getUser(int fd){
        char username[1024];
        read(fd, username, 2048);
        …
        return };
```



Stack guard detects the return address and add a canary:

```
⇨   void getUser(int fd){
        int canary = CANARY_VALUE;
        char username[1024];
        read(fd, username, 2048);
        if (canary != CANARY_VALUE)
                abort();
        …
    }
```



Thus, attackers must guess canary value

(Prob. success = 2^{-32})

Good:

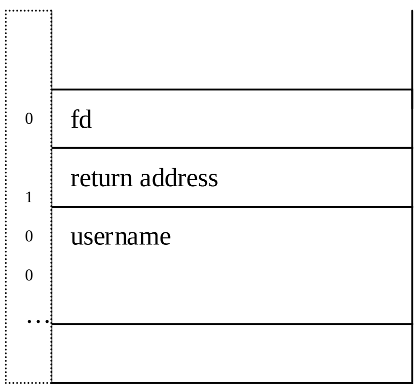
- ⇨ easy
- ⇨ backward compatible
- ⇨ overhead 10-100%

Bad:

- ⇨ can't head all vulnerabilities, including
  - format string bugs
  - data corruption attacks
  - other function pointers
  - heap corruption attacks

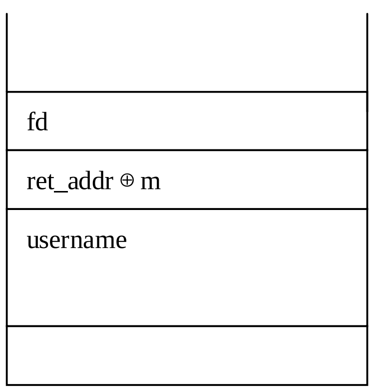
Point Guard:

Suppose each word has a bit indicating whether it's a pointer or not,



When buffer overflow occurs, then return address's extra bit will be 0, which will result in error when loaded to PC b/c it's no longer a pointer.

Point guard takes a different approach; instead, it encrypts pointers in memory,



- ⇨ At startup, pick random mask m
- ⇨ when buffer overflow occurs, username = attackers' code ret_addr ⊕ m = attacker's ret_addr ⊕ m
- ⇨ will return to attacker's ret_addr ⊕ m which is a random location

Good: overhead < 20%

Bad:

- ⇨ overhead?
- ⇨ very backward incompatible

- ⇨ brute force? in some cases
- ⇨ no defense against data corruption
- ⇨ fails unpredictably

Point Guard Example:

```
void getUser(int fd){
        char *p;
        char username[1024];
        p=malloc(..);
        read(fd, username, 2048);
        p=0;
        return;
}
```

```
Call malloc
Xor %r0, %r0, %r31
Store %sp + 1024 %r0
Call read
Load %r0, %sp+1024
Xor %r0, %r0, %r31
Store %r0, 0
```

Address-Space Randomization:

- force attacker to guess address

stack
X
mmap
X
heap
bbs
data
text
X

- Simple: move entire sections
- At most 20-bits of entropy/section
- And some attacks only need to guess one

More randomization:

- randomize each subsection
- randomize location of each section
- randomize return address: stack frame padding
- randomize reorder local variables
- insert padding between locals
- reorder args (and pads)
- reorder of struct field? X