Buffer overflow - system solutions.

1. Stack Guard.

```c
void getusername(int fd) {
    char username[1024];
    read(fd, username, 2048);
    if (canary != CANARY_VALUE) abort();
    return;
}
```

- Let the buffer-overflow occurs, and if the return address is corrupted then abort.

- Attackers must guess canary value (prob success = $2^{-32}$).

Good
- Easy
- Backwards-compatible
- Overhead 10-100%

Bad
- Can't heal all vulnerabilities.
  - Format-string bugs.
  - Data corruption attacks
  - Other function pointers
  - Heap corruption attacks

2. Pointer Guard.

Suppose each word had bit indicating whether it's a pointer or not.

```c
void getusername(int fd) {
    char username[1024];
    read(fd, username, 2048);
    return;
}
```
Instead, point Guard encrypts pointers in memory.

\[
\begin{array}{c|c}
\text{fd} & -\\
\text{net addr} \oplus M & \rightarrow \text{attacker overwrite net_addr} \\
\text{username} & \text{if return to attacker \_net_addr} \oplus M
\end{array}
\]

Good

overhead < 20%

Bad

overhead:

very backwards-incompatible

'brute-force?'

in some cases

data corrupm attacks

fails unpredictably

(Example) void getuser(int fd)

  char *p;
  char *username[2];
  p = malloc();
  read(fd, username, 2048);
  *p = 0
  return;

  \[
  \rightarrow \text{call malloc} \\
  \rightarrow \text{xor } \%r8, \%r8, \%r3 \\
  \rightarrow \text{store } \%sp +1024, \%r6 \\
  \rightarrow \text{call read} \\
  \rightarrow \text{load } \%r6, \%sp+1024 \\
  \rightarrow \text{xor } \%r6, \%r6, \%r6 \\
  \rightarrow \text{store } \%r6, 0.
\]
Address Space Randomization
- Force Attacker to guess addresses.

- simplest: move entire sections
  - at most 20 bit of entropy/section
  - and some attackers only need to guess one.

More Randomization.
- randomize each subsection
  - location of each function
  - stack frame padding
  - insert padding between local
  - reorder args (and pad?)
  - order of struct field? X.