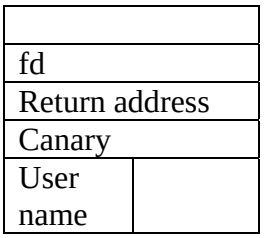
Buffer Overflows-System Solutions:
Stack Guard:



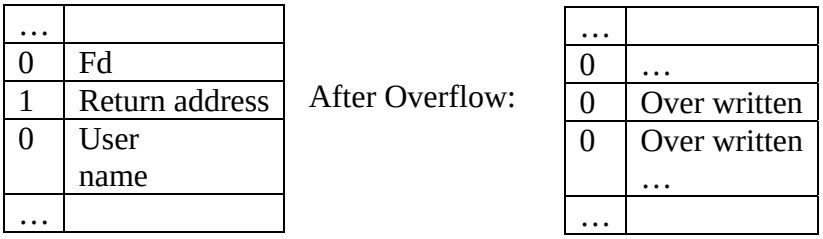

```
void getUser(int fd)
{
char username[1024];
read(fd,username,2048);
if(canary!=CANARY_VALUE)  abort();
return;
}
```

Attacker must guess canary value to succeed. Probability of success= 2^{-32}

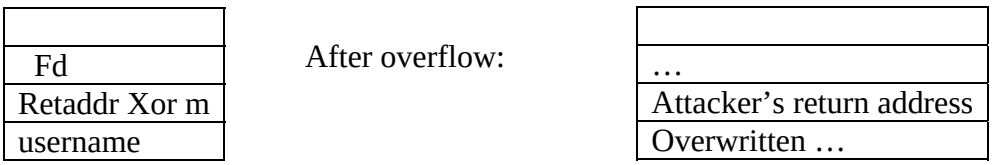
Good	Bad
-very easy	-can't heal all vulnerabilities, eg: format string bugs
-backwards compatible	-data corruption attacks
-overhead 10-100%	-other function pointers
	-heap corruption

Point Guard:
Suppose each word had a bit indicating whether it is a pointer or not:

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



When overflow occurs, the extra bit indicating pointer variables gets over written too.
Instead point guard encrypts pointers in memory. (eg. retaddr X-OR mask m)



So when the return address is overwritten by the attacker's return address, system will return to (attacker_ret Xor m) which is wrong and the system may crash.

Good	Bad
Not much overhead (<20%)	-very backwards incompatible (difficult to link point guarded and non point guarded code) -brute forcible in some cases -data corruption attacks -fails unpredictably

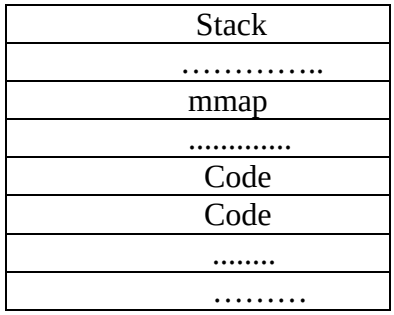
Point guard example:

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

Would be written as:

```
call malloc;
Xor  %rØ, %rØ,  %r31(<- mask)
store %sp+1024,  %rØ
call read
load %rØ
Xor  %rØ, %rØ,  %r31
store %rØ,0
```

Address Space Randomization:
-Force attacker to guess addresses.
-Simplest: move entire sections
-At most 20 bits of entropy per section
-And some attacks need to guess only one



More Randomization:
-randomize each subsection
-location of each function
-stack frame padding
-reorder locals
-insert padding between locals
-reorder arguments(and pad?)
-order of struct fields? NO