Insecurity of StackGuard

Attacker can obtain the canary's value, which will be used to overwrite the canary in the stack without changing the value.

- Format string vulnerability allows the attacker to print out values in the stack (%x).
- The attacker can use brute-force technique to guess the canary.

Attacker can overwrite the return address in the stack without touching the canary.

- Format string vulnerability allows the attacker to write to any location in memory, not need to be consecutive with the buffer (%n).
- Heap overflows do not overwrite a stack canary.

Shadow Stack

Keep a copy of the stack in memory

- On function call: push the return address (EIP) to the shadow stack.
- On function return: check that top of the shadow stack is equal to the return address (EIP) on the stack.
- If there is difference, then attack happens and the program will be terminated.

Shadow stack requires the support of hardware

- Intel CET (Control-flow Enforcement Technology):
  - New register SSP: Shadow Stack Pointer
  - Shadow stack pages marked by a new "shadow stack" attribute:
  - only "call" and "ret" can read/write these pages

StackShield

A GNU C compiler extension that protects the return address.

Separate control (return address) from the data.

- On function call: copies away the return address (EIP) to a non-overflowable area.
- On function return: the return address is restored.
- Even if the return address on the stack is altered, it has no effect since the original return address will be copied back before the returned address is used to jump back.

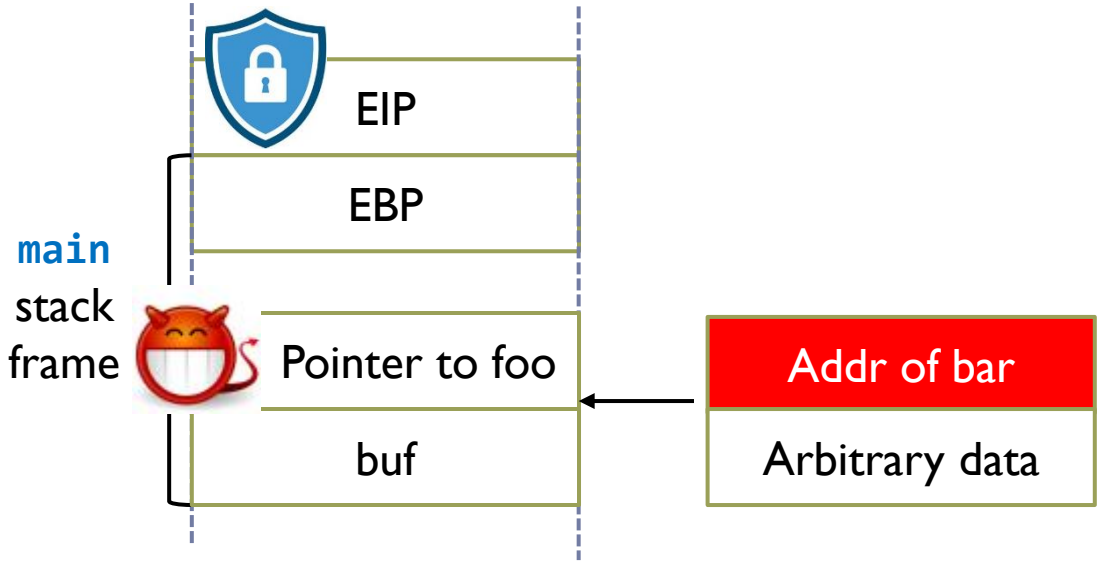
Common Limitations of StackGuard, Shadow Stack, and StackShield

Only protect the return address, but not other important pointers

Hijacking a function pointer

- Even if the attacker cannot overwrite the return address due to the canary, he can overwrite a function pointer.

```
void foo () {...}
void bar () {...}
int main() {
    void (*f) () = &foo;
    char buf [16];
    gets(buf);
    f();
}
```



PointGuard

A compiler-based approach to protect the function pointers from being overwritten

- Encrypt all points while stored in memory, and decrypt them when loaded into CPU registers for use.

Steps

- A secret key is randomly generated for each program when it is launched.
- Pointer encryption: when loading a pointer to memory, encrypt it with the key (typically XOR).
- Pointer decryption: before a pointer is used by CPU, decrypt it with the key (XOR). This ensures the pointer is in its original, unencrypted form only during actual use within CPU, minimizing the window of vulnerability.
- Without knowing the correct key, the attacker cannot overwrite stack data with the encrypted form of malicious function address.