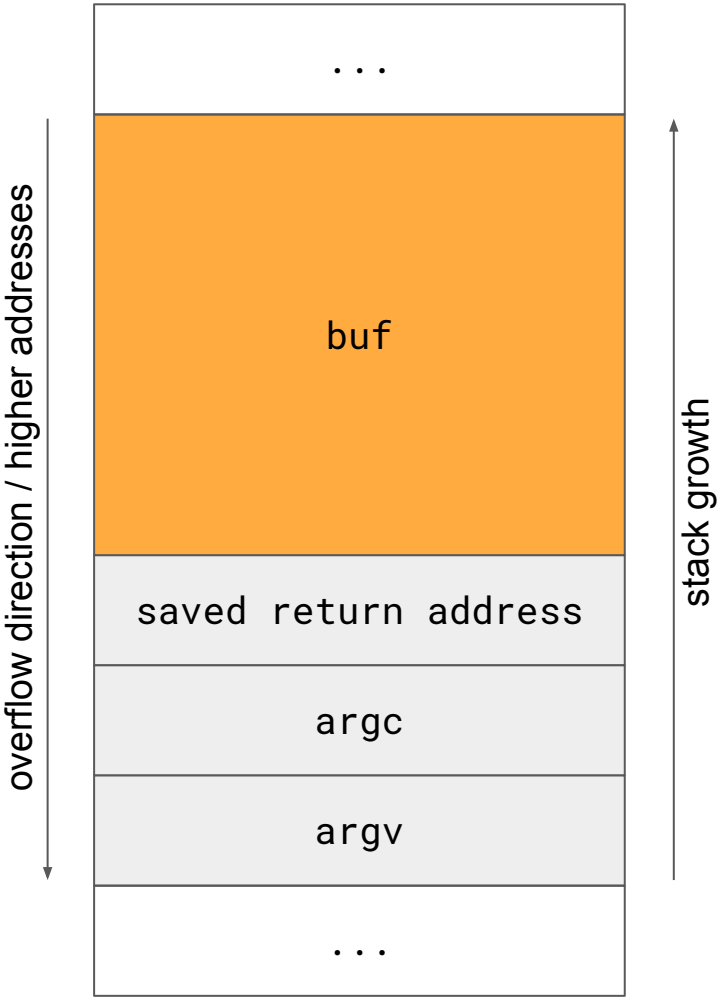
Smashing the Stack Protector for Fun and Profit

1996: Smashing The Stack for Fun and Profit

```
int main(int argc, char **argv)
{
    char buf[0x10];
    gets(buf);
    return 0;
}
```

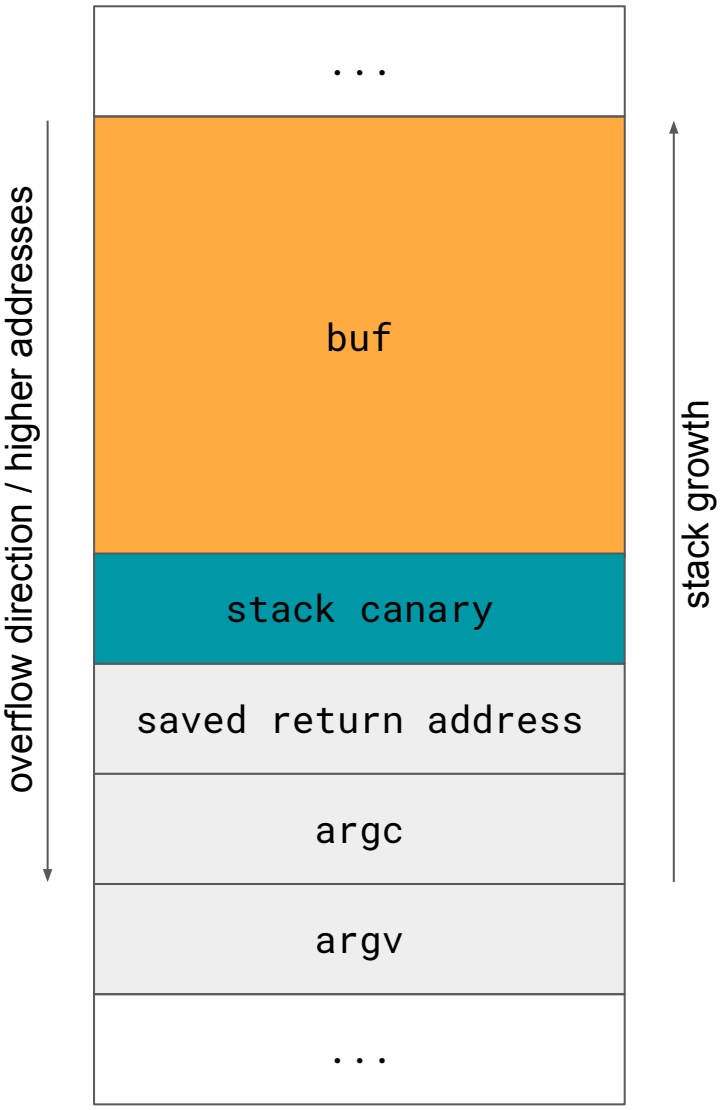


1998: StackGuard

Function Prologue:
Place canary on stack

```
int main(int argc, char **argv)
{
    char buf[0x10];
    gets(buf);
    return 0;
}
```

Function Epilogue:
Check canary integrity



Our Work: (Systematic) Evaluation of Implementations

- Integrity check is implemented as a comparison with some reference value

- Central question:

Where is the reference value stored?

Background: Memory Types

- Stack based variable - local variable

```
int a(void) {
    char loc[0x10];
```

- Thread Local Storage - storage specific to one thread

```
char __thread tls[0x10];
```

- Static - global variable

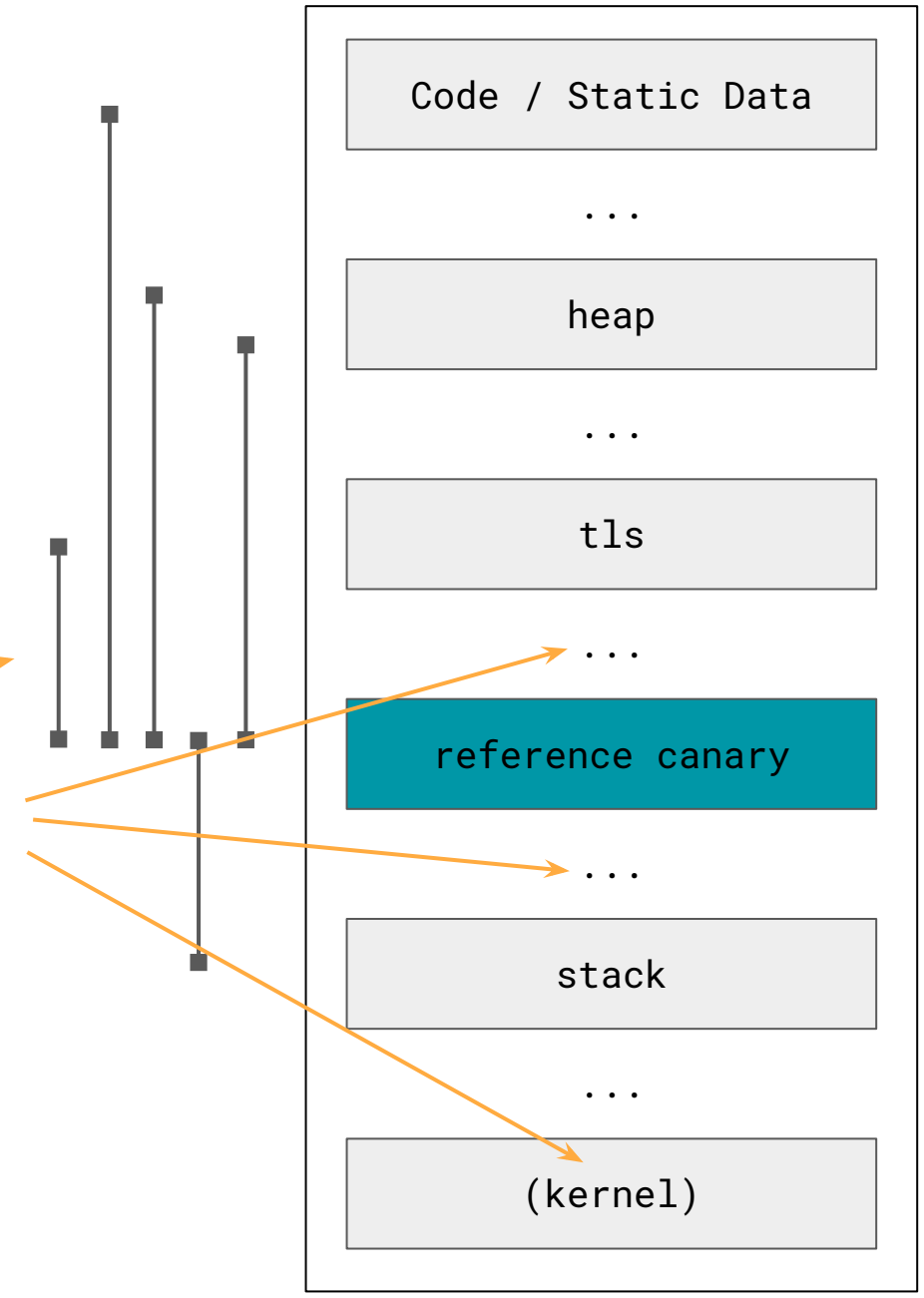
```
static char sta[0x10];
```

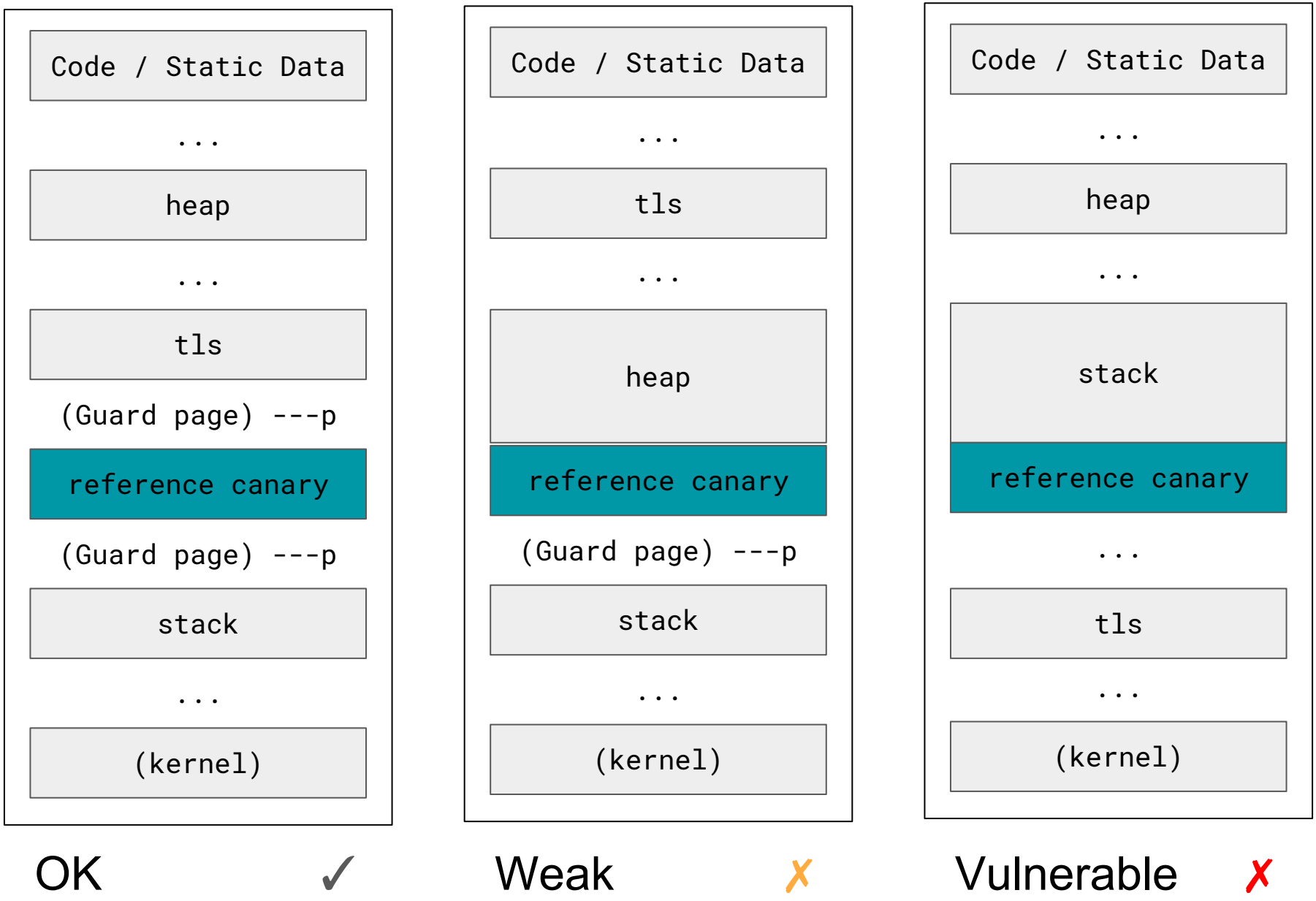
- Dynamic - allocated with `malloc`

```
char *dyn = malloc(0x10);
```

Sample Address Space Layout

- Measurement:
  - Distances
  - Gaps / Permissions
- Classification:
  - OK ✓
  - Weak ✗
  - Vulnerable ✗




Code / Static Data
...
heap
...
tls
(Guard page) ---p
reference canary
(Guard page) ---p
stack
...
(kernel)
OK
✓
Code / Static Data
...
tls
...
heap
reference canary
(Guard page) ---p
stack
...
(kernel)
Weak
✗
Code / Static Data
...
heap
...
stack
reference canary
...
tls
...
(kernel)
Vulnerable
✗

	OS	Arch.	C Standard Library	LOC		TLS		STA		DYN	
				main	*sub*	*main*	*sub*	*main*	*sub*	*main*	*sub*
1	Android 7.0	ARM	`Bionic`	✓	✓	✓	✓	✓	✓	✓	✓
2	Android 7.0	x86_64	`Bionic`	✓	✗	✓	✓	✓	✓	✓	✓
3	macOS 10.12.1	x86_64	`libSystem.dylib`	✓	✓	✓	✓	✓	✓	✓	✓
4	FreeBSD 11.00	x86_64	`libc.so.7`	✓	✓	✓	✓	✗	✗	✓	✓
5	OpenBSD 6.0	x86_64	`libc.so.88.0`	✓	✓	-	-	✓	✓	✓	✓
6	Windows 10	x86	`msvcr1400.dll`	✓	✓	✓	✓	✗	✗	✓	✓
7	Windows 10	x86_64	`msvcr1400.dll`	✓	✓	✓	✓	✗	✗	✓	✓
8	Windows 7	x86	`msvcr1400.dll`	✓	✓	✓	✓	✗	✗	✓	✓
9	Windows 7	x86_64	`msvcr1400.dll`	✓	✓	✓	✓	✗	✗	✓	✓
10	Arch Linux	x86_64	`libc-2.24.so`	✓	✗	✗	✗	✓	✓	✓	✓
11	Debian Jessie	x86	`libc-2.19.so`	✓	✗	✗	✗	✓	✓	✓	✓
12	Debian Jessie	x86	`musl libc 1.1.5`	✓	✗	✗	✗	✓	✓	✗	✓
13	Debian Jessie	ARM	`libc-2.19.so`	✓	✓	✓	✓	✗	✗	✓	✓
14	Debian Jessie	PowerPC	`libc-2.19.so`	✓	✗	✗	✗	✓	✓	✓	✓
15	Debian Jessie	s390x	`libc-2.19.so`	✓	✗	✗	✗	✓	✓	✓	✓
16	Debian Stretch	x86_64	`diet libc 0.33`	✗	✗	✗	✗	✓	✓	✓	✗
17	Ubuntu 14.04 LTS	x86_64	`EGLIBC 2.15`	✓	✗	✗	✗	✓	✓	✓	✓

Conclusion:
libcs with Thread Local Storage & Threading
Are completely broken!

```
=========================================================================

ASIACCS'17 Review #386A

-------------------------------------------------------------------------

Paper #386: CookieCrumbl0r: Smashing the Stack Protector for Fun and Profit

-------------------------------------------------------------------------

Overall merit: 2. Weak reject

Reviewer expertise: 4. Expert
```

(...)

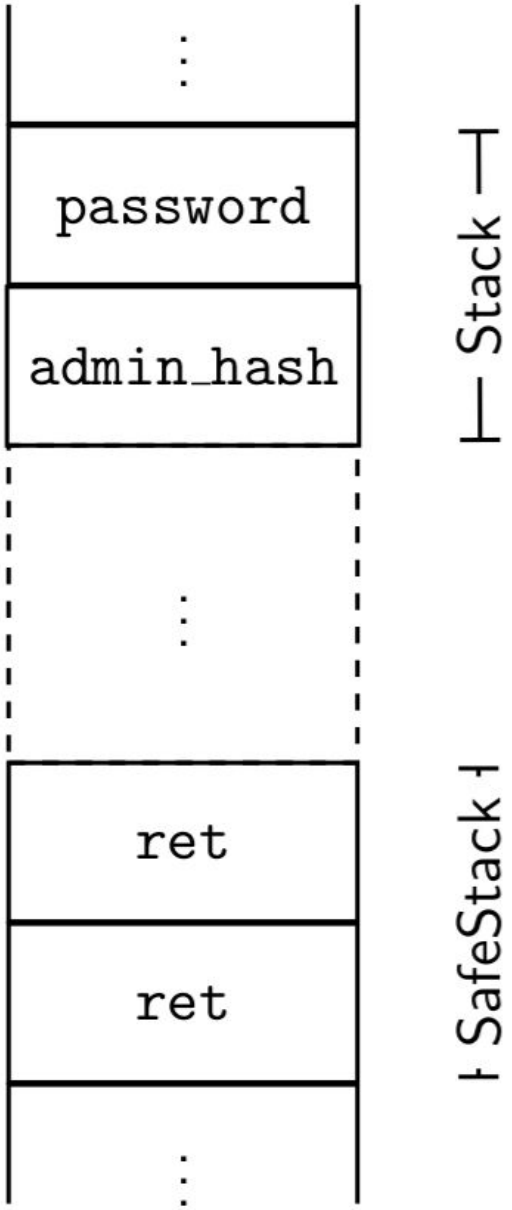
```
===== Weaknesses of paper =====

With the rise of CFI mechanisms that will protect the backward edge through
some form of stack integrity, defenses that rely on stack cookies are on
their way out, therefore this paper has low novelty and impact.
```

⇒ Fallacy!

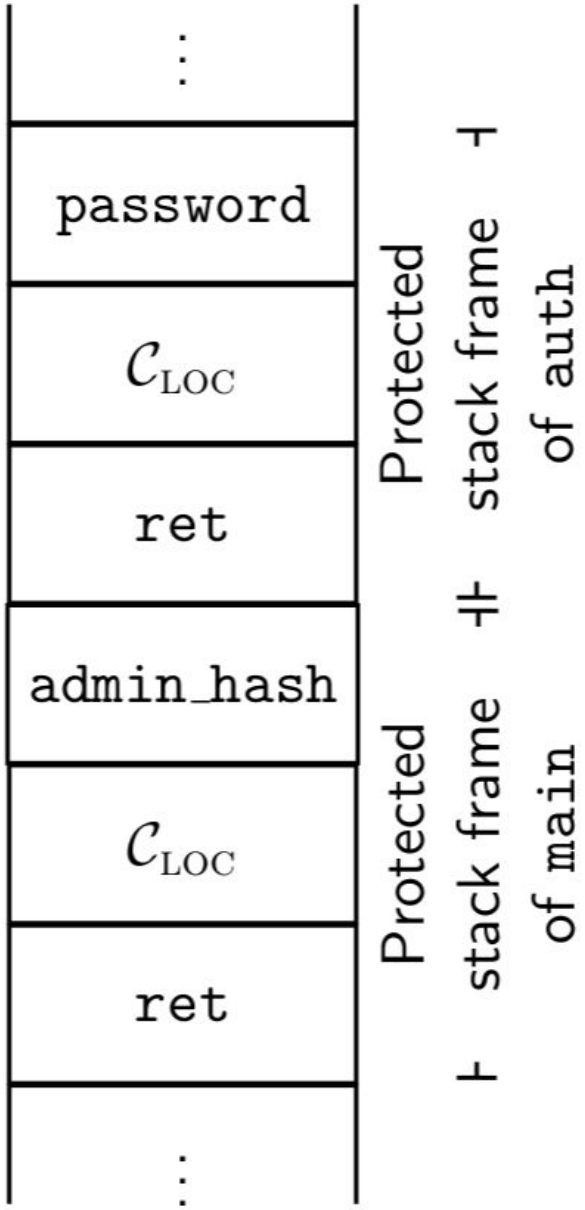
Why we need stack canaries, even with CFI:

```
int auth(char * valid) {
    char password[32];
    gets(password);
    return strcmp(valid, crypt(password, valid)) == 0;
}
void main(void) {
    char admin_hash[] = "$6$..."; // long hash value
    if (auth(admin_hash)) {
        puts("Welcome to the Admin Area");
    }
}
```

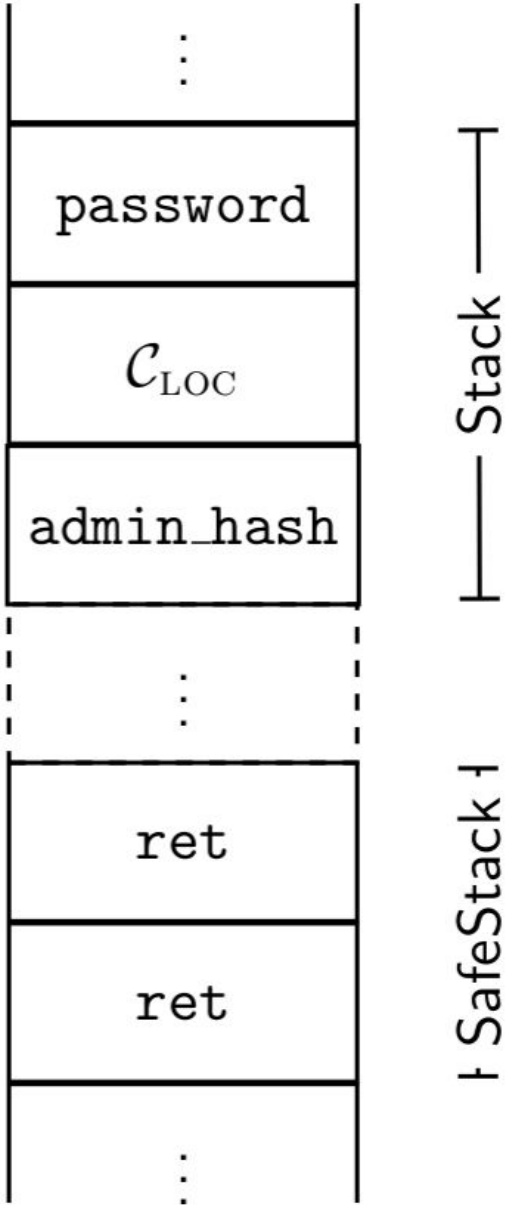
SafeStack:



Canaries:



SafeStack + Canaries:



Conclusion:
Stack Canaries are still a strong protection mechanism and should be used together with newer techniques like CFI.