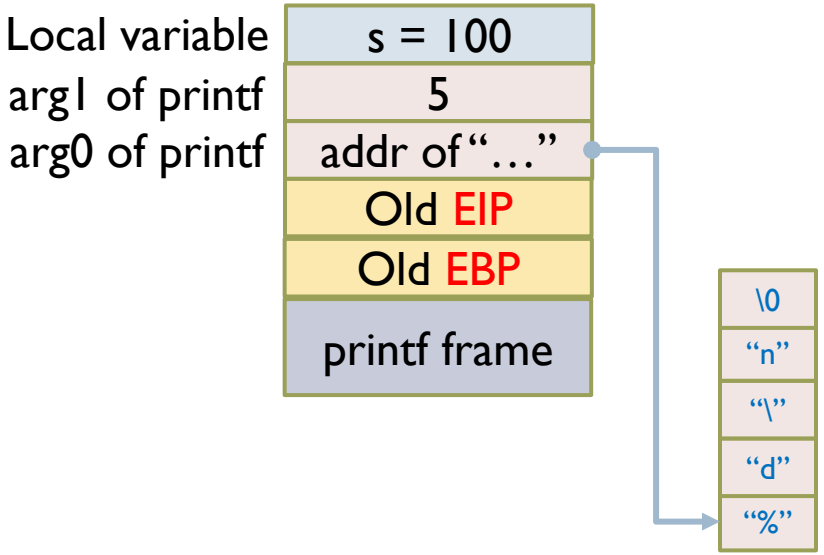
Attack 1: Leak Information from Stack

Correct usage of **printf**

- Two arguments are pushed into the stack as function parameter

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    int s = 100;
    printf("%d\n", 5);
    return 0;
}
```



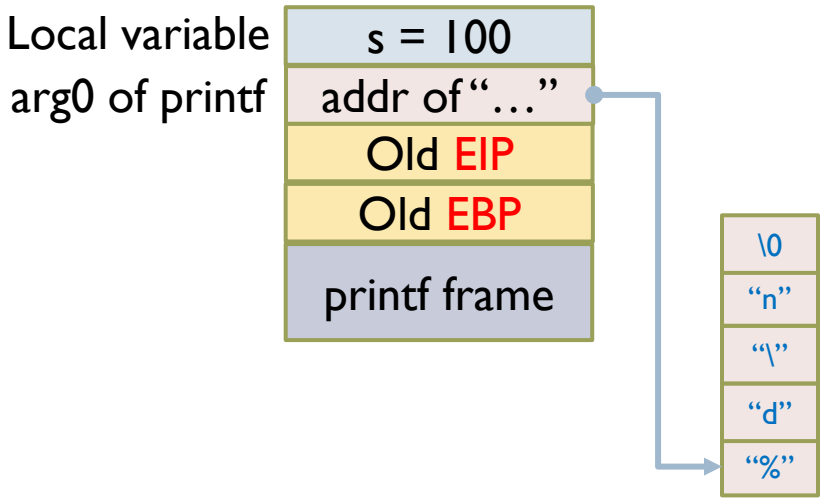
Attack 1: Leak Information from Stack

Incorrect usage of `printf`

- The stack does not realize an argument is missing, and will retrieve the local variable as the argument to print out.
- Data that do not belong to the user are thus leaked to the attacker.
- The attacker can print out any types of data, including integer (**%d**), floating point (**%f**), string (**%s**), address (**%p**)…

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    int s = 100;
    printf("%d\n");
    return 0;
}
```



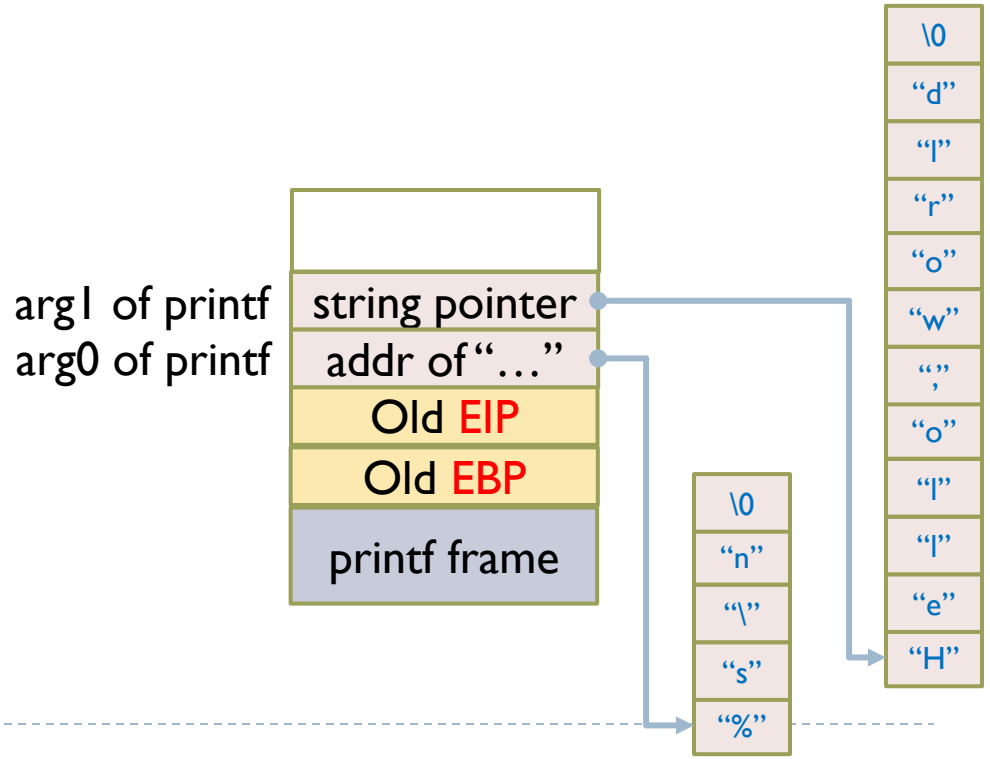
Attack 2: Crash the Program

Correct usage of **printf**

- For format specifier **%s**, a pointer of a string is pushed into the stack as the corresponding function parameter

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    printf("%s\n", "hello, world");
    return 0;
}
```



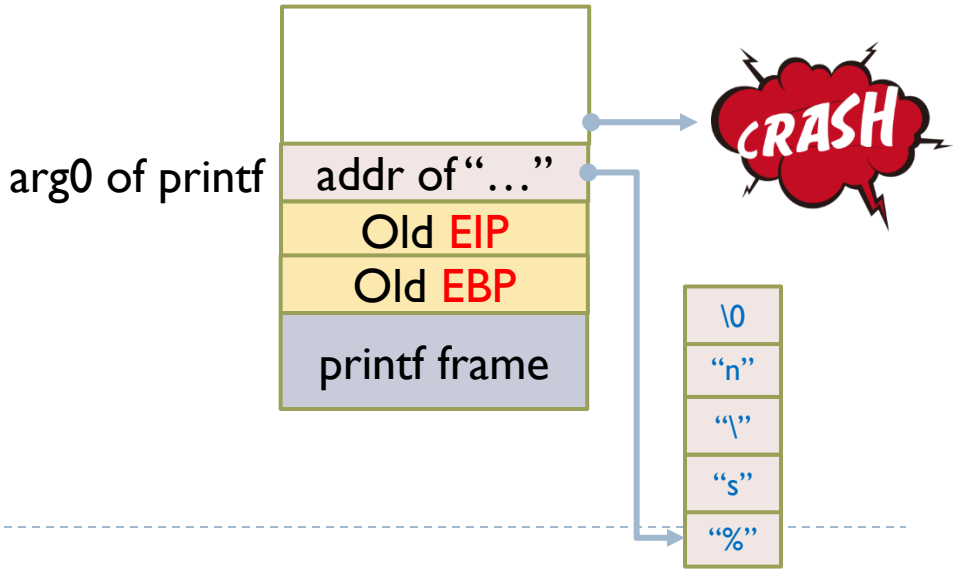
Attack 2: Crash the Program

Incorrect usage of `printf`

- The stack does not realize an argument is missing, and will retrieve other stack values as addresses, and access data there.
- This address can be invalidated, and then the program will crash
  - No physical address is assigned to this address
  - Protected memory, e.g., kernel.
- Can include more `%s` to increase the crash probability

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    printf("%s\n");
    return 0;
}
```



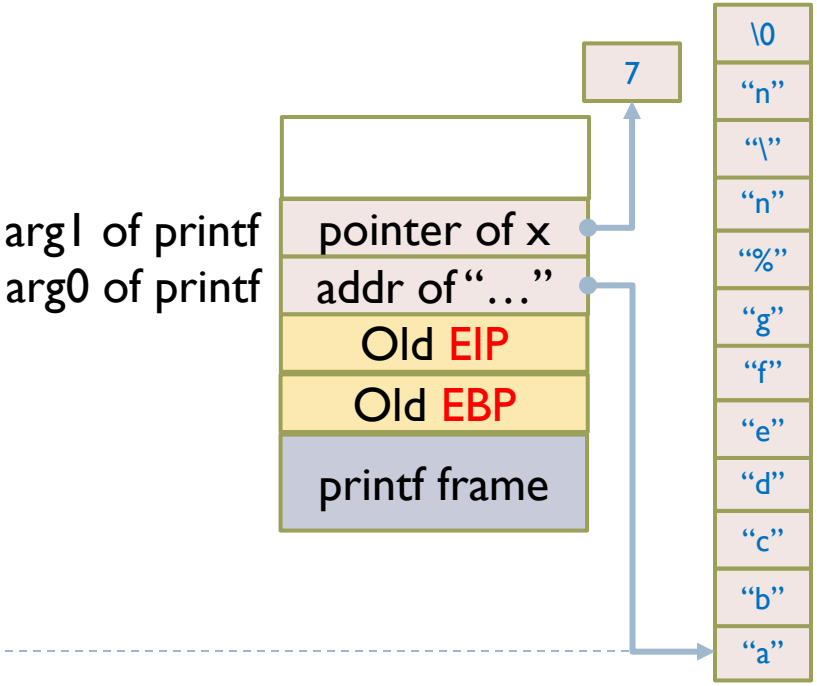
Attack 3: Modify the Memory

Correct usage of `printf`

- For format specifier **%n**, a pointer of a signed integer is pushed into the stack as the corresponding function parameter.
- Store the number of characters written so far into that integer

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    int *x = (int *)malloc(sizeof(int));
    printf("abcdefg%n\n",x);
    return 0;
}
```



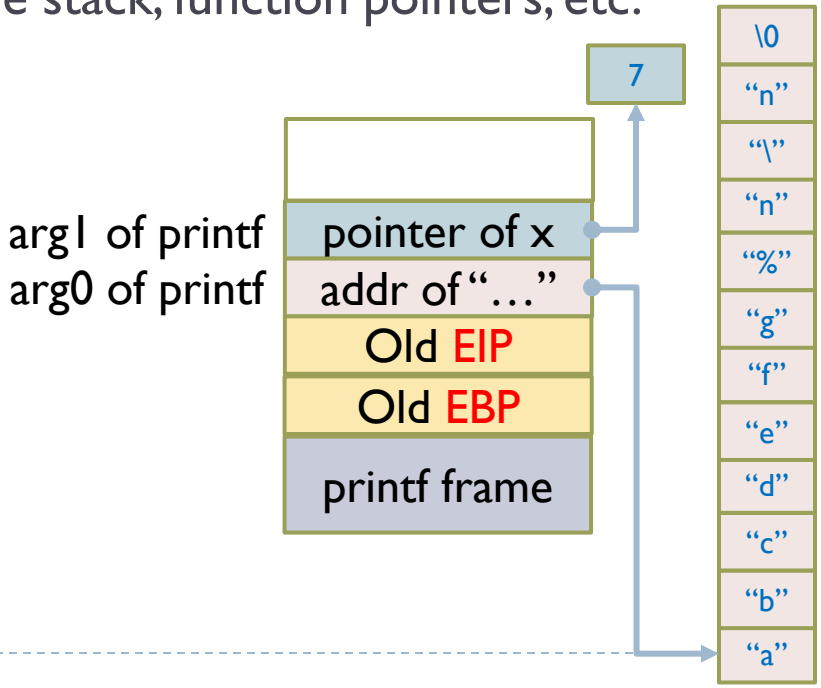
Attack 3: Modify the Memory

Incorrect usage of **printf**

- The stack does not realize an argument is missing, and will retrieve the data from the stack and write 7 into this address.
- Attacker can achieve the following goal:
  - Overwrite important program flags that control access privileges
  - Overwrite return addresses on the stack, function pointers, etc.

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    int *x = (int *)malloc(sizeof(int));
    printf("abcdefg%n\n");
    return 0;
}
```



More Similar Vulnerable Functions

Functions	Descriptions
`printf`	prints to the 'stdout' stream
`fprintf`	prints to a FILE stream
`sprintf`	prints into a string
`snprintf`	prints into a string with length checking
`vprintf`	prints to 'stdout' from a va_arg structure
`vfprintf`	print to a FILE stream from a va_arg structure
`vsprintf`	prints to a string from a va_arg structure
`vsnprintf`	prints to a string with length checking from a va_arg structure
`syslog`	output to the syslog facility
`err`	output error information
`warn`	output warning information
`verr`	output error information with a va_arg structure
`vwarn`	output warning information with a va_arg structure
……	

History of Format String Vulnerability

Originally noted as a software bug (1989)

- By the fuzz testing work at the University of Wisconsin

Such bugs can be exploited as an attack vector (September 1999)

- `snprintf` can accept user-generated data without a format string, making privilege escalation was possible

Security community became aware of its danger (June 2000)

Since then, a lot of format string vulnerabilities have been discovered in different applications.

Application	*Found by*	*Impact*	*years*
wu-ftpd 2.*	security.is	remote root	> 6
Linux rpc.statd	security.is	remote root	> 4
IRIX telnetd	LSD	remote root	> 8
Qualcomm Popper 2.53	security.is	remote user	> 3
Apache + PHP3	security.is	remote user	> 2
NLS / locale	CORE SDI	local root	?
screen	Jouko Pynnönen	local root	> 5
BSD chpass	TESO	local root	?
OpenBSD fstat	ktwo	local root	?