
Stack Protection Systems: (propolice, StackGuard, XP SP2)

Hiroaki Etoh

Tokyo Research Laboratory, IBM Japan

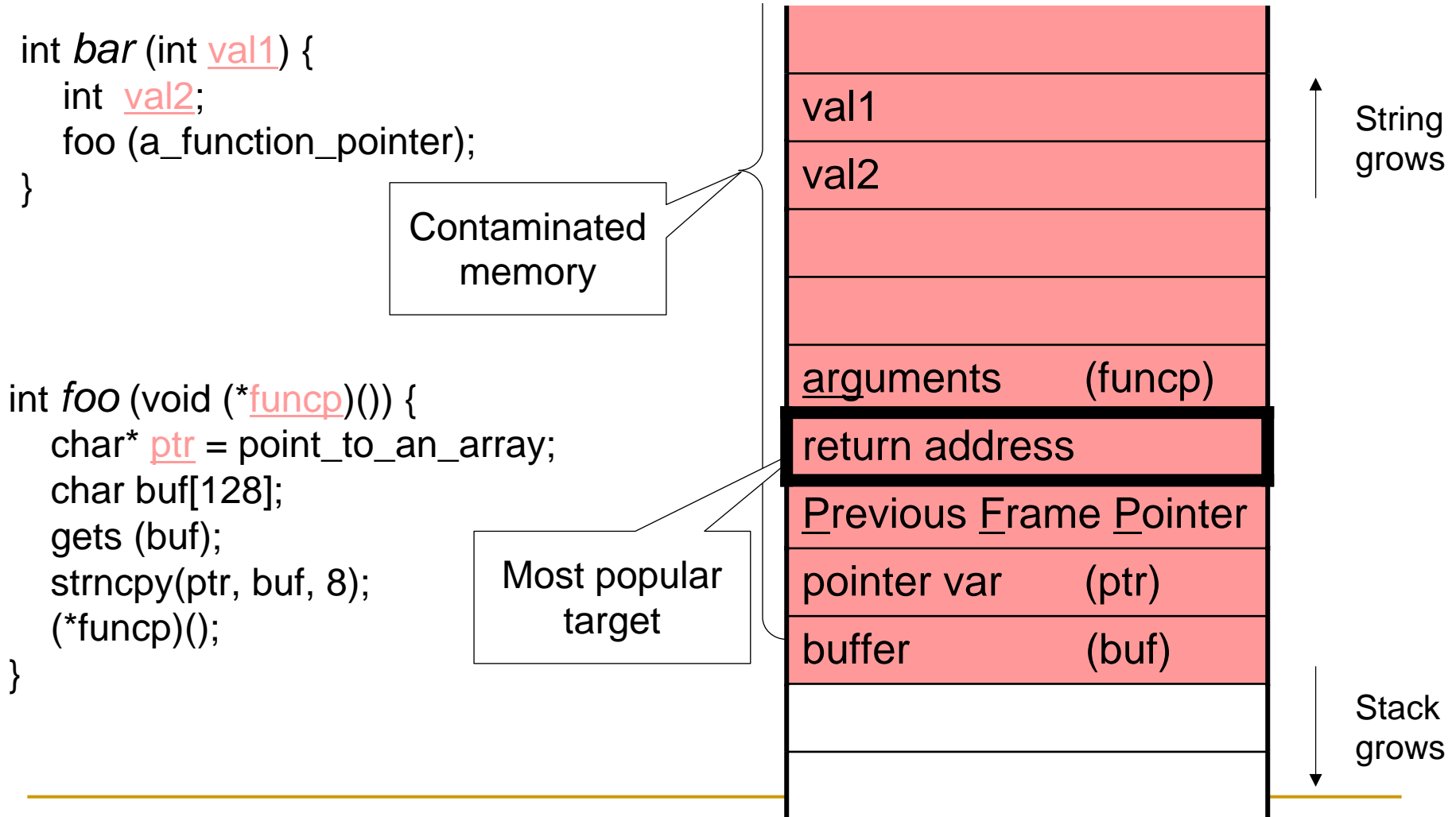
Contents

- Buffer overflow in stack
 - What is a stack smashing attack
 - Stack protector landscape
 - StackGuard
 - propolice
 - Windows XP SP2 (/Gs option)
 - Comparison
 - Summary
-

What is a buffer overflow in the stack

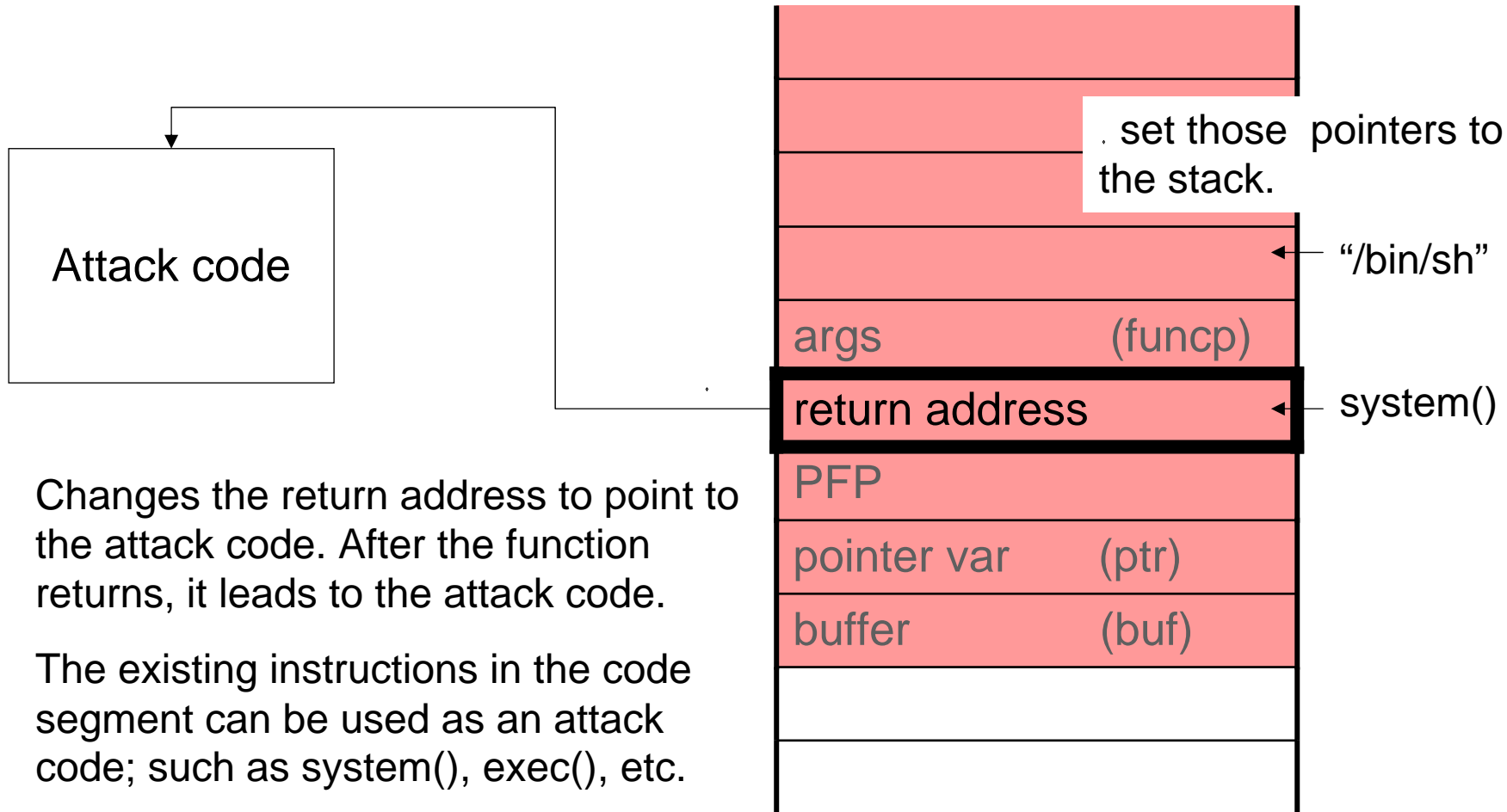
- A buffer overflow occurs when you try to put too many bits into an allocated buffer.
 - When this happens, the next contiguous chunk of memory is overwritten, such as
 - Return address
 - Function pointer
 - Previous frame pointer, etc.
 - Also an attack code is injected.
 - This can lead to a serious security problem.
-

Stack Layout and Contaminated Memory by the Attack --- when function foo is called by bar.



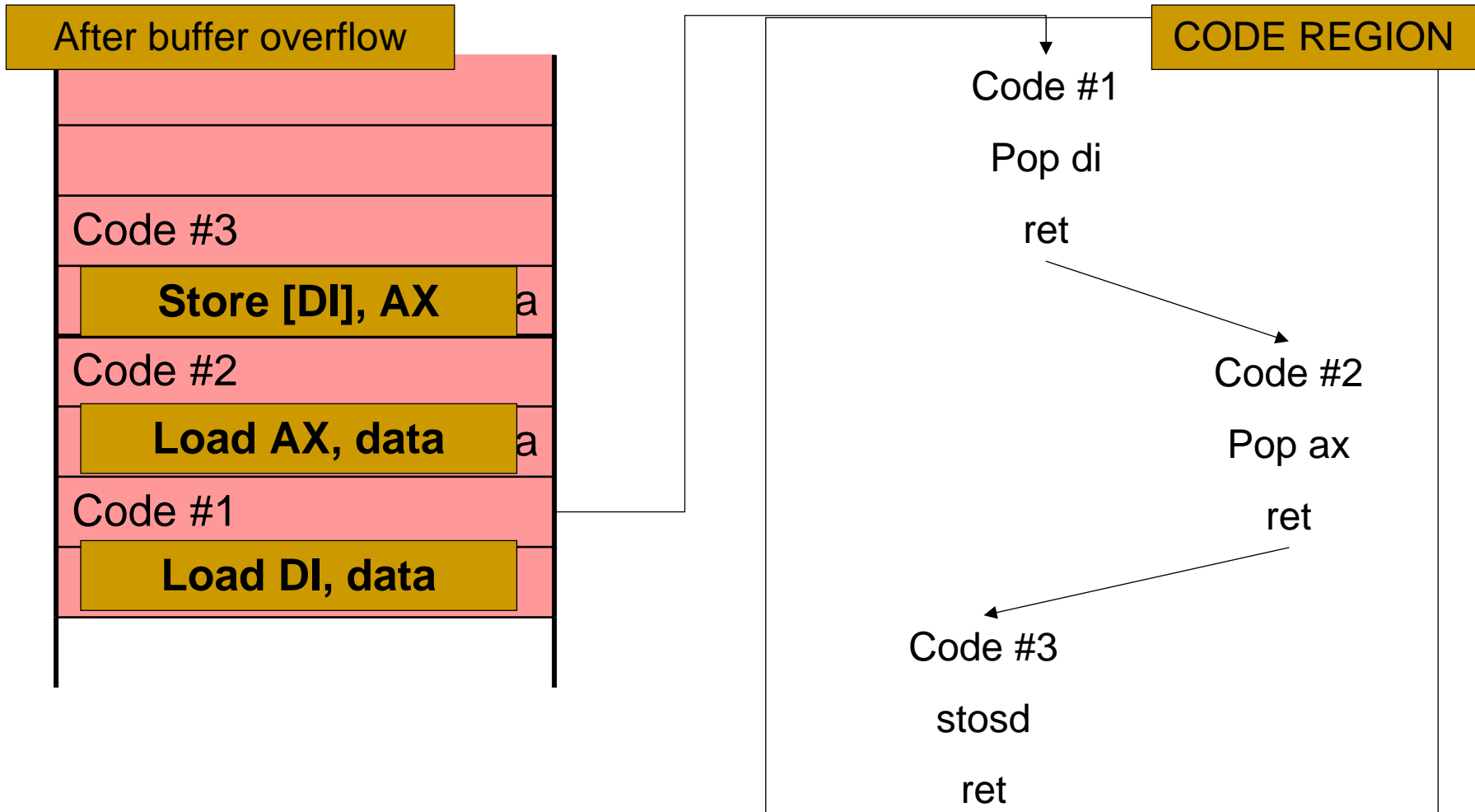
Attack Scenario #1

--- by changing the return address



Pseudo code execution on the stack, avoiding the non-executable stack method

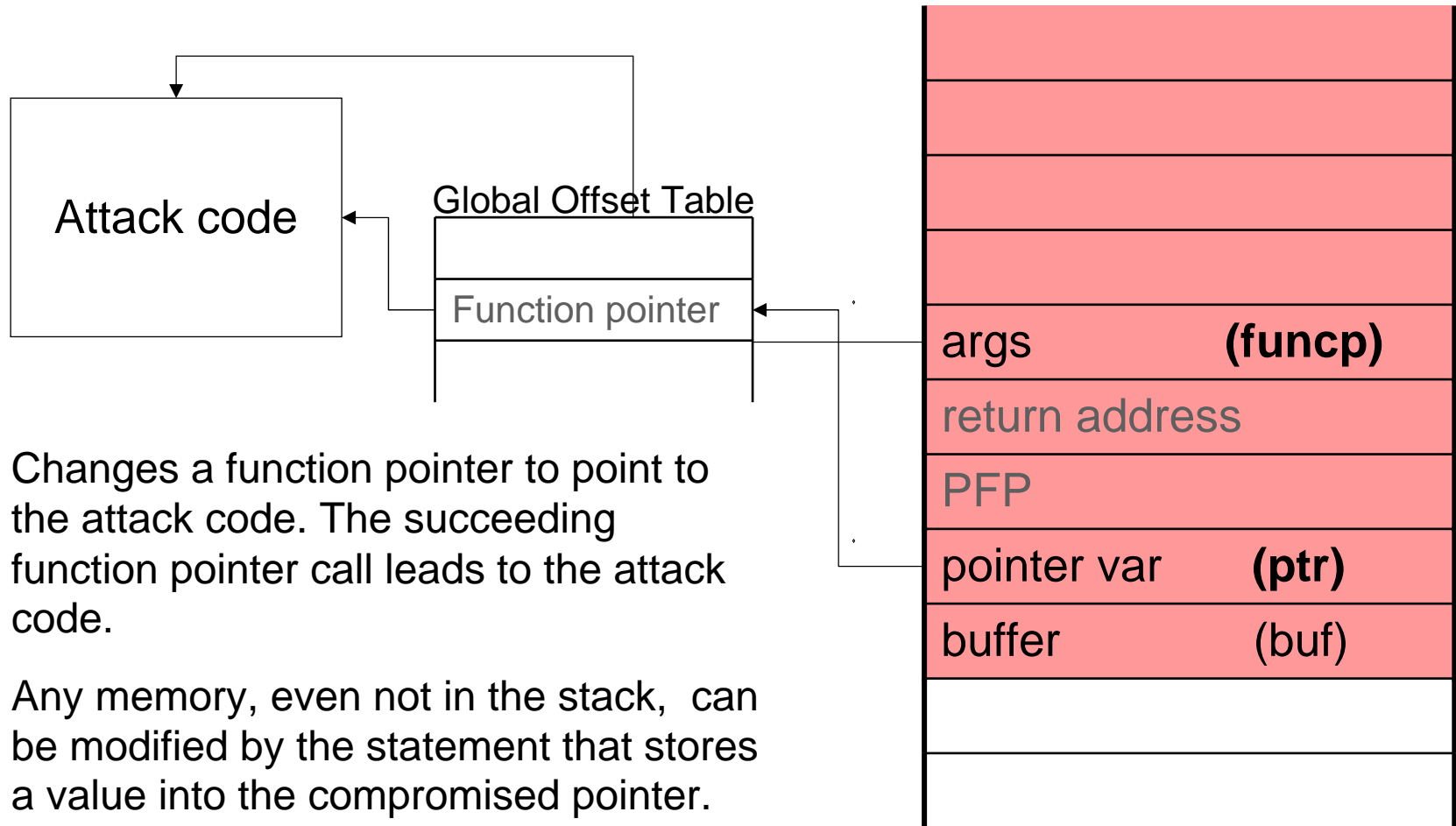
www.securiteam.com, "Avoiding Stackguard and Other Stack Protection - Proof of Concept Code"



Jumping through code fragments in the code region

Attack Scenario #2

--- by changing pointer variables



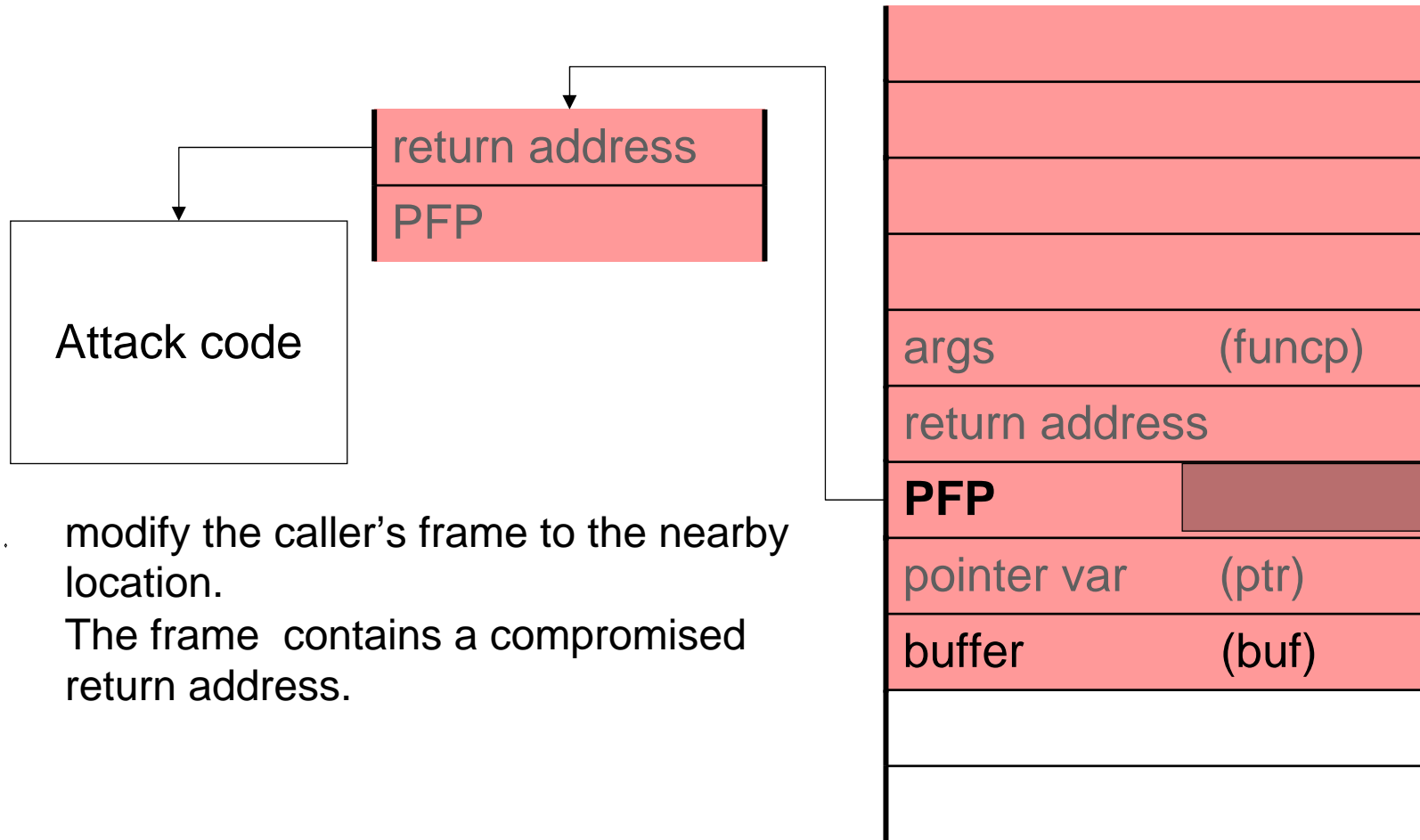
Changes a function pointer to point to the attack code. The succeeding function pointer call leads to the attack code.

Any memory, even not in the stack, can be modified by the statement that stores a value into the compromised pointer.

E.g. `strncpy(ptr, buf, 8);`
`*ptr = 0;`

Attack Scenario #3

--- by changing the previous frame pointer



Stack protector Landscape

- Compiler based protector
 - StackGuard, stack shield, proPolice, XP SP2 /Gs
- Runtime stack integrity checker
 - Libsafe
- Non-executable parts of the address space
 - Solar Designer's "non-exec stack patch", Exec Shield, OpenBSD's W^X, XP SP2 NX

✓ There is no single solution!!!

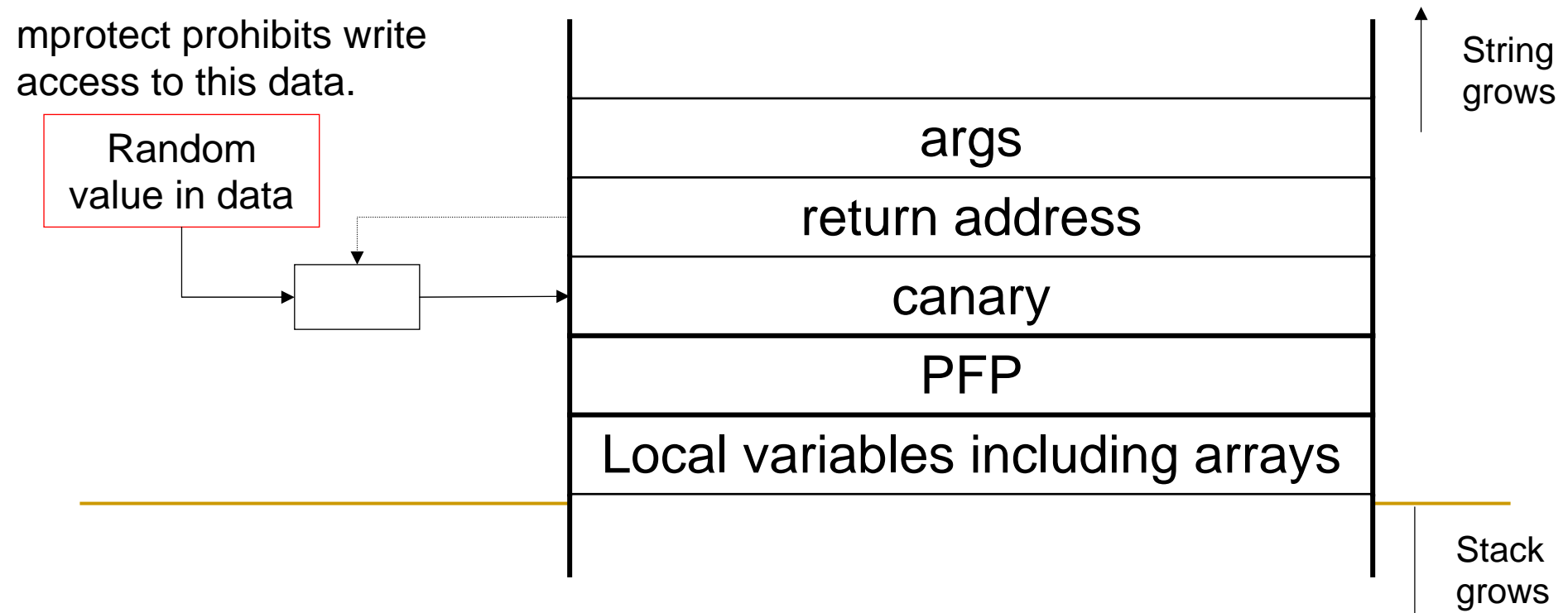
Stack Guard

- StackGuard places a “canary” word next to (prior) the return address on the stack.
 - Once the function is done, the protection instrument checks to make sure that the canary word is unmodified before jumping to the return address.
 - If the integrity of canary word is compromised, the program will terminate.

 - Vulnerability report
 - “BYPASSING STACKGUARD AND STACKSHIELD”, Phrack 56
 - “Four different tricks to bypass StackShield and StackGuard protection”
-

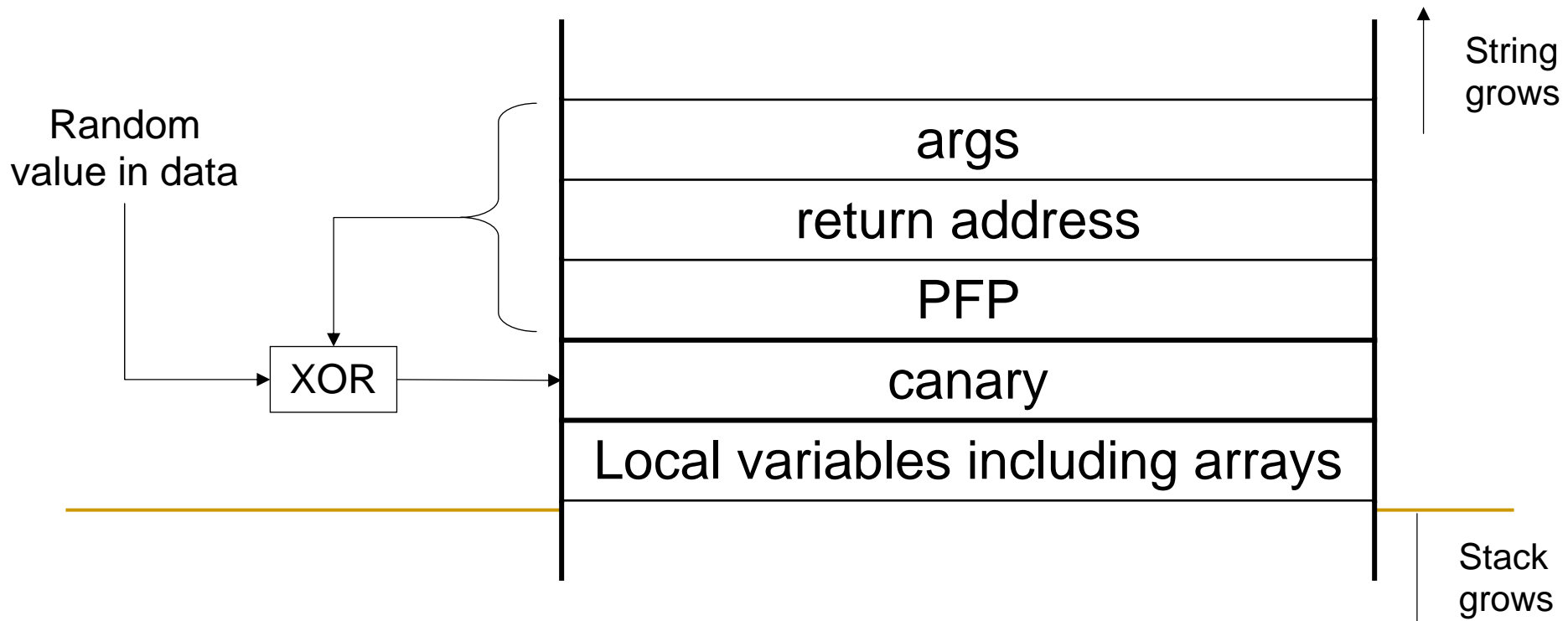
Stack Guard 2.1

- Canary value variations
 - ❑ Terminator canary 0x000aff0d
 - ❑ Random canary random
 - ❑ XOR canary: random ^ return address
- You choose a canary method when building the compiler.



Stack Guard under development

- Move the canary to eliminate the frame pointer problem
- Broad range of integrity check for return address, frame pointer, and local variables.

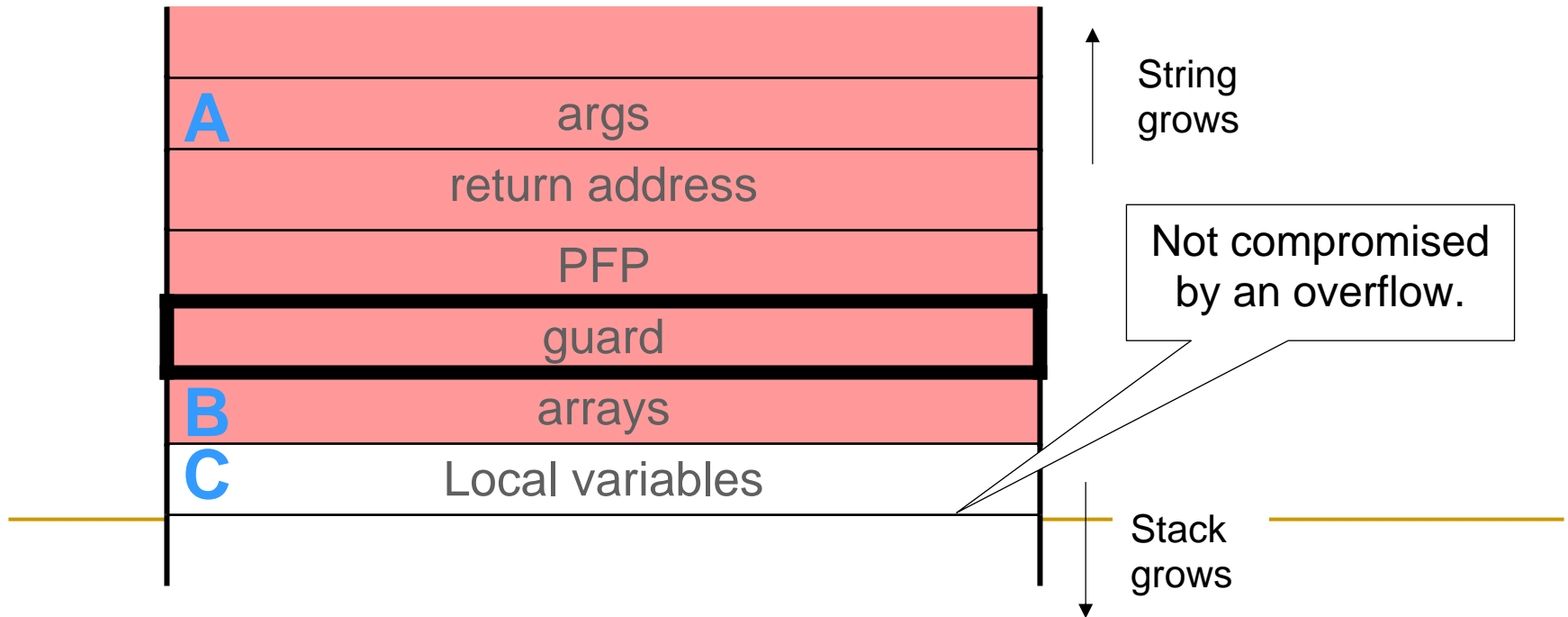


propolice: design goal

- Introduce “Safe Stack Usage Model”
 - This is a combination of an ideal stack layout and a way to check the stack integrity.
 - Transform a program to meet the ideal stack layout as much as possible.
 - A patch for GNU gcc compiler adds a compilation stage to transform the program.
-

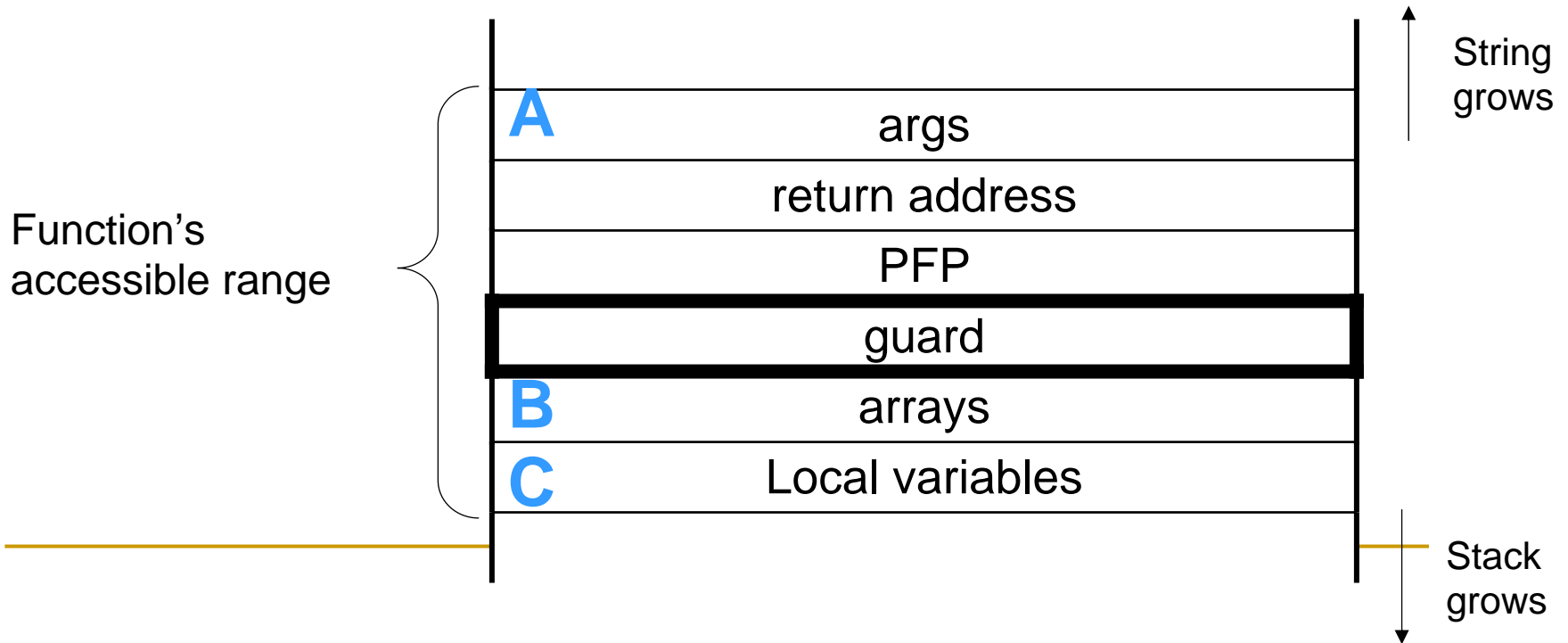
Safe Stack Usage Model

- Stack integrity check:
 - Assigns unpredictable value into the guard at the function prologue.
 - Confirms the integrity of the guard value at the function epilogue, or aborts the program execution.
- Ideal stack layout:
 - **A** doesn't have arrays nor pointer variables.
 - **B** has only arrays
 - **C** has no array, but has pointer variables.



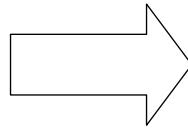
Why caller function is safe from a stack smashing attack.

- There are no pointer variables from args to guard, which is the function's accessible range. So any memory can't be compromised by a pointer attack.
- When a function successfully return to the caller function, it means that contiguous chunk of memory of caller function's stack is not compromised by buffer overflows.



Intuitive explanation: how to make a guard instrument between PFP and arrays.

```
foo () {  
    char *p;  
    char buf[128];  
    gets (buf);  
}
```



```
Int32    random_number;  
foo () {  
    volatile int32 guard;  
    char buf[128];  
    char *p;  
    guard = random_number;  
    gets (buf);  
    if (guard != random_number)  
        /* program halts */  
}
```

1. Insert guard instrument

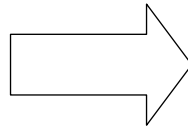
2. Relocate local variables

+ The optimizer may eliminate the second access for random_number.

- The buffer alloca allocated can not be relocate next to the guard.

Intuitive explanation: how to treat function arguments if any of them has a pointer type.

```
foo (int a, void (*fn)()) {  
    char buf[128];  
    gets (buf);  
    (*fn)();  
}
```



```
Int32    random_number;  
foo (int a, void (*fn)()) {  
    volatile int32 guard;  
    char buf[128];  
    (void *safefn)() = fn;  
    guard = random_number;  
    gets (buf);  
    (*safefn)();  
    if (guard != random_number)  
        /* program halts */  
}
```

1. Copy the pointer to a variable assigned from the region **C**.
In fact, it try to assign the register for that variable.
2. Rename the function call with the assigned variable.

propolice: stack protector options

- **-fstack-protector**
 - Stack protection instruments are generated only when the function has a byte array.
 - **-fstack-protector-all**
 - Always generate the guard instrument.
 - If a byte array is used, it is allocated next to the guard.
 - Otherwise, any array is allocated next to the guard.
-

propolice status

<http://www.research.ibm.com/trl/projects/security/ssp/>

- Actual usage
 - Laser5, trusted debian, openbsd, gentoo, etc
 - Supported architectures
 - ix86, powerpc, alpha, sparc, mips, vax, m68k, amd64
 - Gcc versions
 - gcc2.95.3 – gcc3.4.1
 - gcc HEAD cvs under development
-

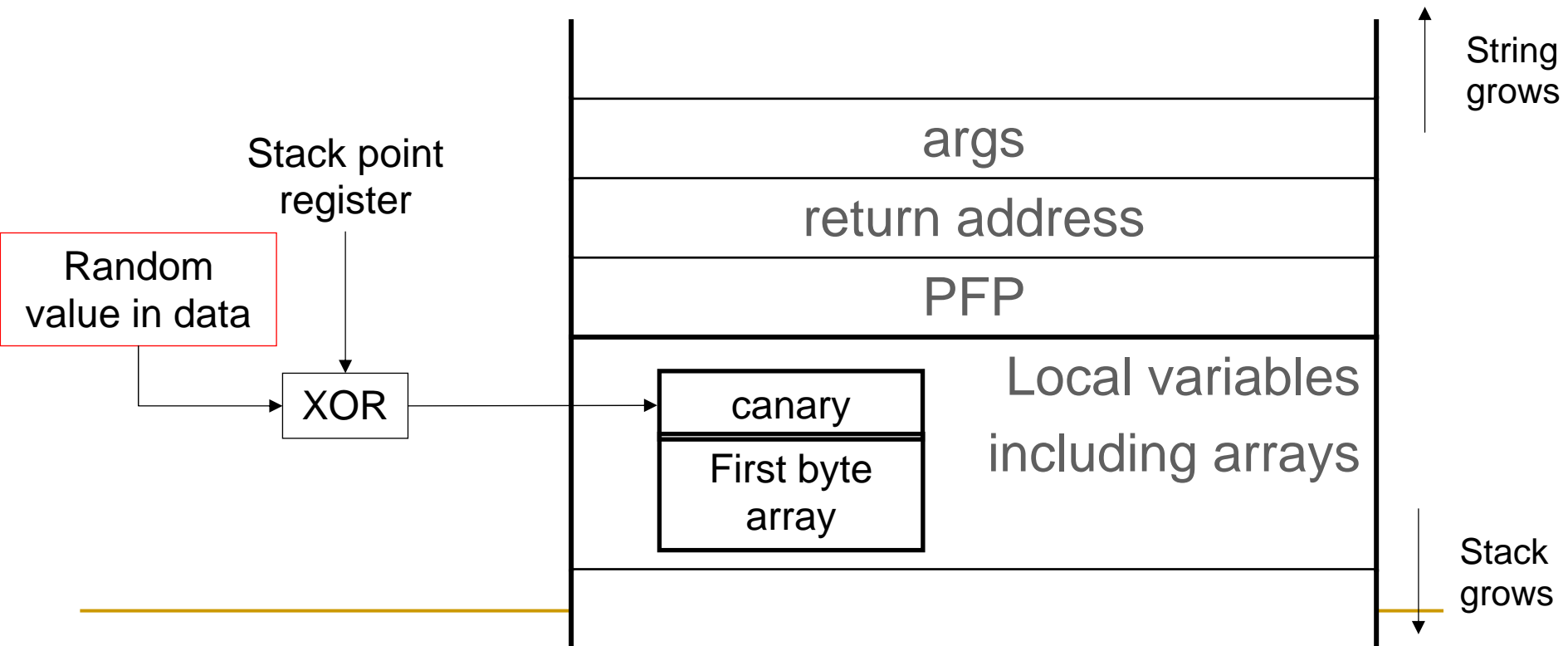
Microsoft XP SP2

--- Windows 2003 stack protection

- Non executable stack
 - Compiler /Gs option
 - Combination method of xor canary and propolice
 - Far from ideal stack layout
 - Vulnerability report
 - David Litchfield, “Defeating the stack based buffer overflow prevention mechanism of Microsoft Windows 2003 server”
-

How Gs option works

- Canary is inserted prior to the first occurrence of byte array allocated
- Local variables except arrays seems to be assigned alphabetical order in the stack.



Comparison of protection techniques

--- protection level

	StackGuard 2.1/3	MS /Gs	propolice	propolice stack-protector-all
Any buffer overflow	applicable	no	no	applicable
Return address	detect	detect	detect	detect
PFP	no/detect	detect	detect	detect
Pointers in local variable	no/detect	detect	protect	protect
Pointers in args	no/detect	detect	protect	protect
Function pointer	no	no	protect	protect
Modifications by pointer	no	no	protect	protect

detect: The modification is found at the end of the function.

protect: The modification can't be done.

Performance considerations

	SG/tc	SG/rc	SG/xc	SG/3	MS/Gs	propolice	propolice/ all
Protect all funcs	yes	yes	yes	yes	no	no	yes
Number of extra instructions executed at no overflow detection							
Mem load	1	3	5	5 -	3	2 - 3	2 - 3
Mem save	1	1	1	1	1	1	1
Other	2	2	4	4 -	4	2	2
Experimental benchmark (execution overhead: %)							
Ctag	-	3	-	-	-	1	-
Perl	-	8	-	-	-	4	-

The overhead percentages shown make it sufficient to enable this by default in all operating systems.

Summary

- Introduced stack overflow problem.
 - Explained the variety of stack smashing attacks.
 - Provided characteristics for StackGuard, propolice, and MS/Gs.
 - Compared each protection methods from various aspects.
-