



Transparent Run-Time Defense Against Stack Smashing Attacks

Timothy Tsai, Navjot Singh
Avaya Labs
Avaya Inc.

<http://www.bell-labs.com/org/11356/libsafe.html>

In the news...

Study says "buffer overflow" is most common security bug

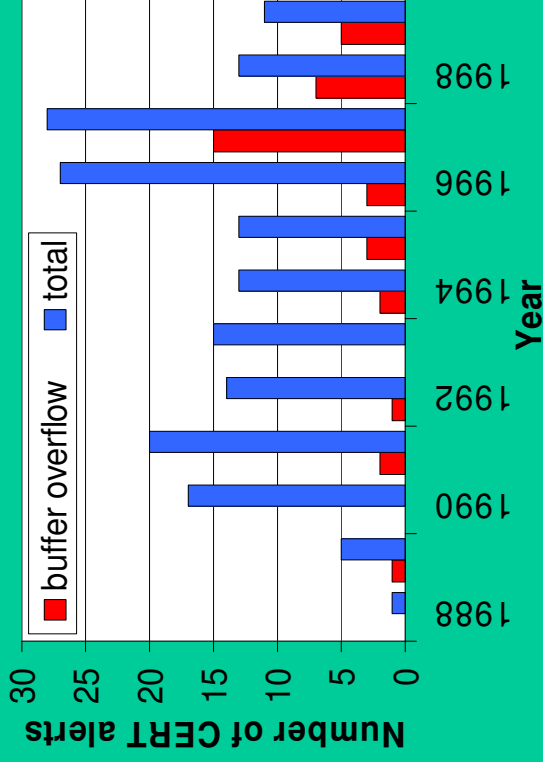
By [Paul Festa](#)

Staff Writer, CNET News.com

November 23, 1999, 2:05 PM EDT

Quick: What's the common security bug of the decade?

It's not the Y2K bug, according to a new study from the science and security analysis firm CERT. The most common weakness known as the Y2K bug, which threatened computers unable to distinguish between the two-digit shorthand, this year has been replaced by computers to attacks by computer hackers who can use the bug to compromise a computer. [...]



Blocking Buffer Overflow Attacks

An easily avoided attack takes advantage of programming errors.

by Rik Farrow, [magazine.com](#)

1999 Black Hat security conference announced that there was something happening in security. Not just anything was happening, just that it was the same old stuff—stuff that has not been fixed years ago.

The real story...

- **Buffer overflows are:**
 - **Common programming errors**
 - **Caused by the lack of bounds checking**
 - **“The most common form of security vulnerability for the past 10 years.” — OGI**
- **“Crackers” leverage buffer overflows to accomplish two dependent goals:**
 - ☆ **Inject attack code**
 - ✂ **Alter the control flow to execute the attack code**

Sample buffer overflow attack

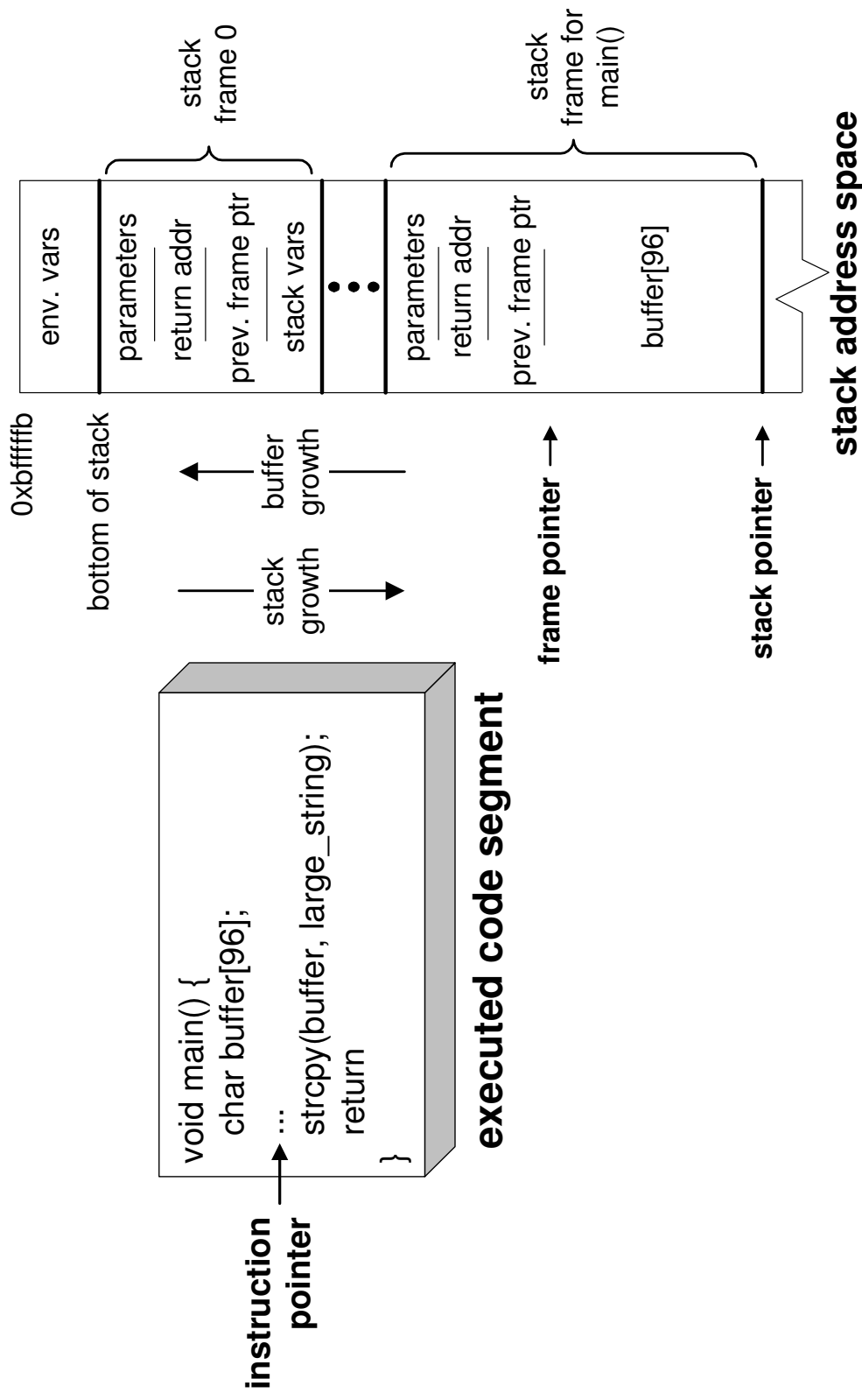
```
#include <stdio.h>
#include <string.h>
char shellcode[] = "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

char large_string[128];
int i;
long *long_ptr;

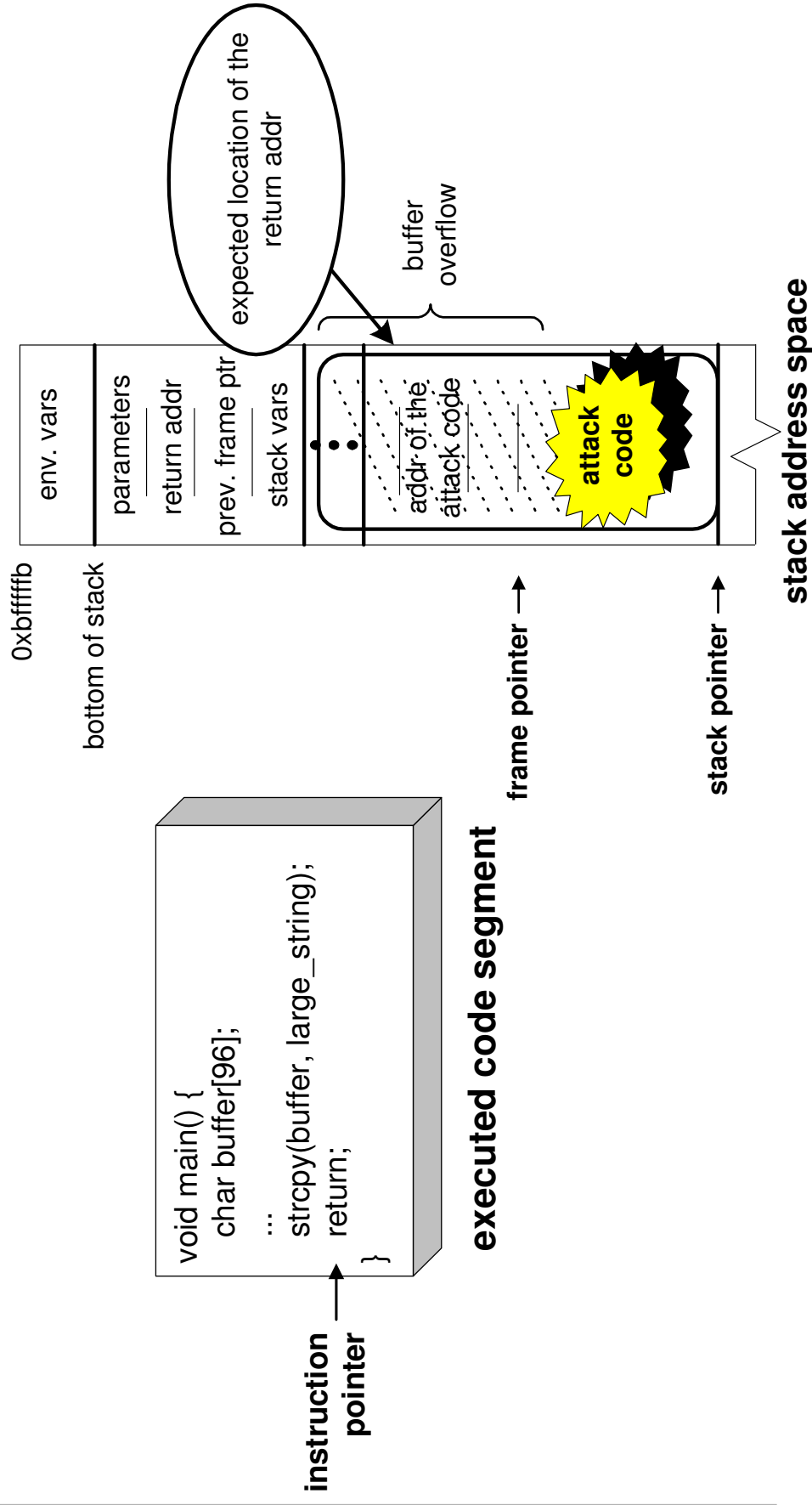
int main() {
    char buffer[96];

    long_ptr = (long *)large_string;
    for (i=0; i<32; i++)
        *(long_ptr+i) = (int)buffer;
    for (i=0; i<strlen(shellcode); i++)
        large_string[i] = shellcode[i];
    strcpy(buffer, large_string);
    return 0;
}
```

A buffer overflow exploit (1 of 3)



A buffer overflow exploit (2 of 3)

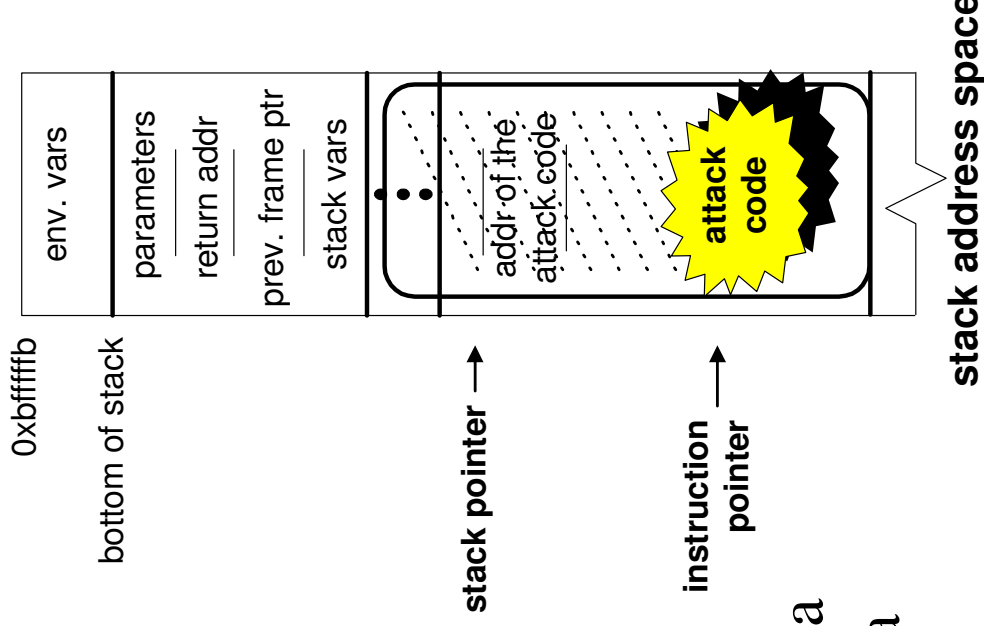


```
void main() {  
    char buffer[96];  
    ...  
    strcpy(buffer, large_string);  
    return;  
}
```

instruction pointer

executed code segment

A buffer overflow exploit (3 of 3)



```
void main() {  
    char buffer[96];  
    ...  
    strcpy(buffer, large_string);  
    return;  
}
```

executed code segment

Buffer overflow targeted at a process stack is referred to a “stack smashing attack”

Our contribution...

- Two complementary solutions to protect against stack smashing attacks
- Unlike previous work
 - transparent: easy to install and to use
 - does not require access to source code
 - efficient efficiency allows use for ALL programs
- ✉ **Interception:** “safe” execution of “unsafe” functions (libsafe)
- ✉ **Binary re-write:** ensures return addresses are valid before use (libverify)

Interception technique

- **Key observations**
 - **Large majority of buffer overflows are caused by misuse of “unsafe” functions, e.g., strcpy() and fscanf()**
 - **At run-time, it is possible to estimate a safe upper bound on the size of stack buffers**
- **Implemented as a software library called libsafe**

Libsafe features

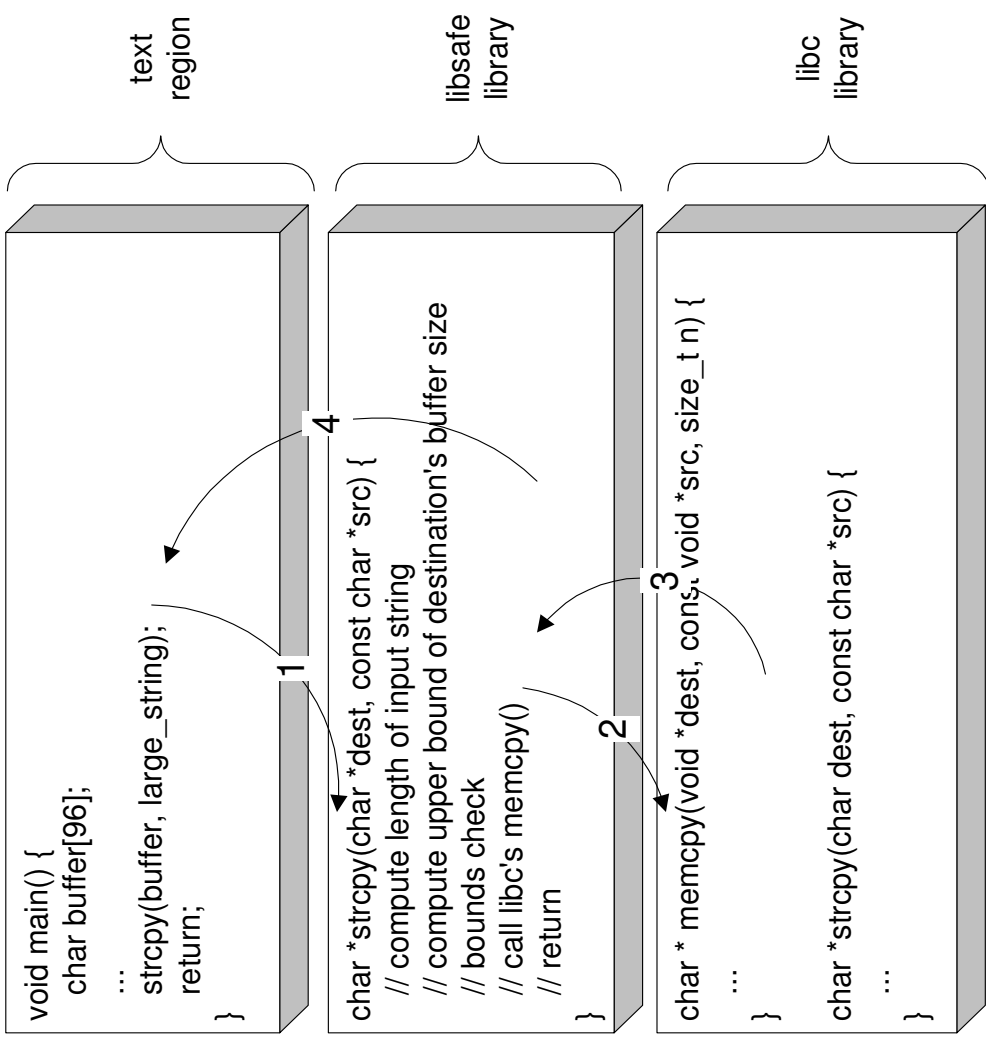
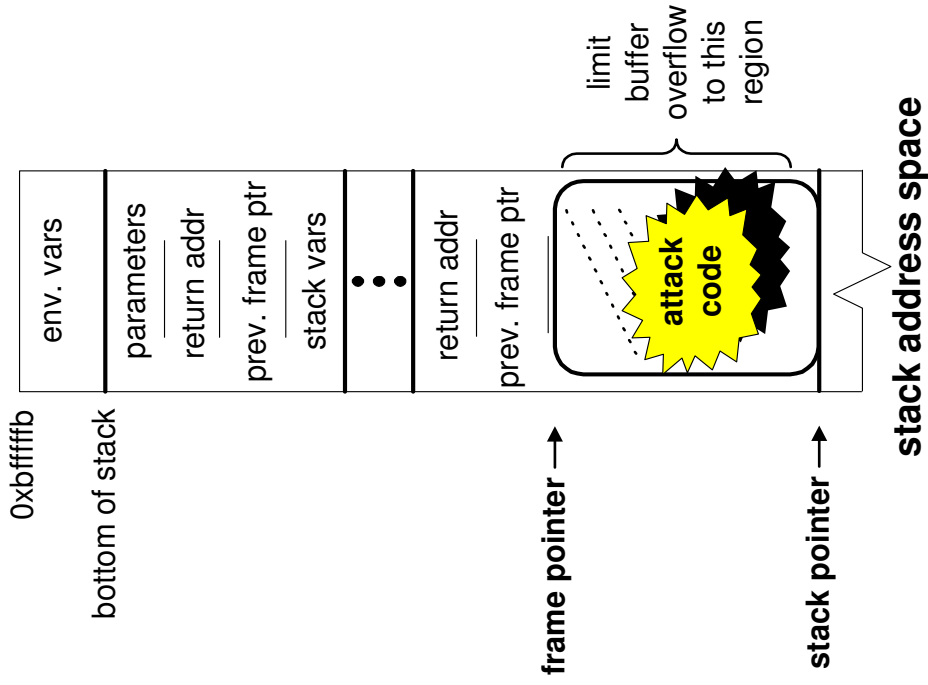
- **Dynamically loadable library**
 - Intercepts “unsafe” function calls.
 - Estimates a safe upper bound on buffers.
 - Executes the function in a “safe” fashion: contains buffer overflows to a safe region
 - Technique similar to zlib
- **Installation: one entry in /etc/ld.so.preload**
- **Guarantees that**
 - Correct programs will execute correctly (except for libc5, -fomit-frame-pointer)
 - Return addresses are protected from misuse of “unsafe” functions

Partial list of unsafe functions

Function prototypes

```
strcat(char *dest, const char *src)
strcpy(char *dest, const char *src)
getwd(char *buf)
gets(char *s)
fscanf(FILE *stream, const char *format, ...)
scanf(const char *format, ...)
realpath(char *path, char resolved_path[])
sprintf(char *str, const char *format, ...)
```

Libsafe



Libsafe containment of buffer overflow

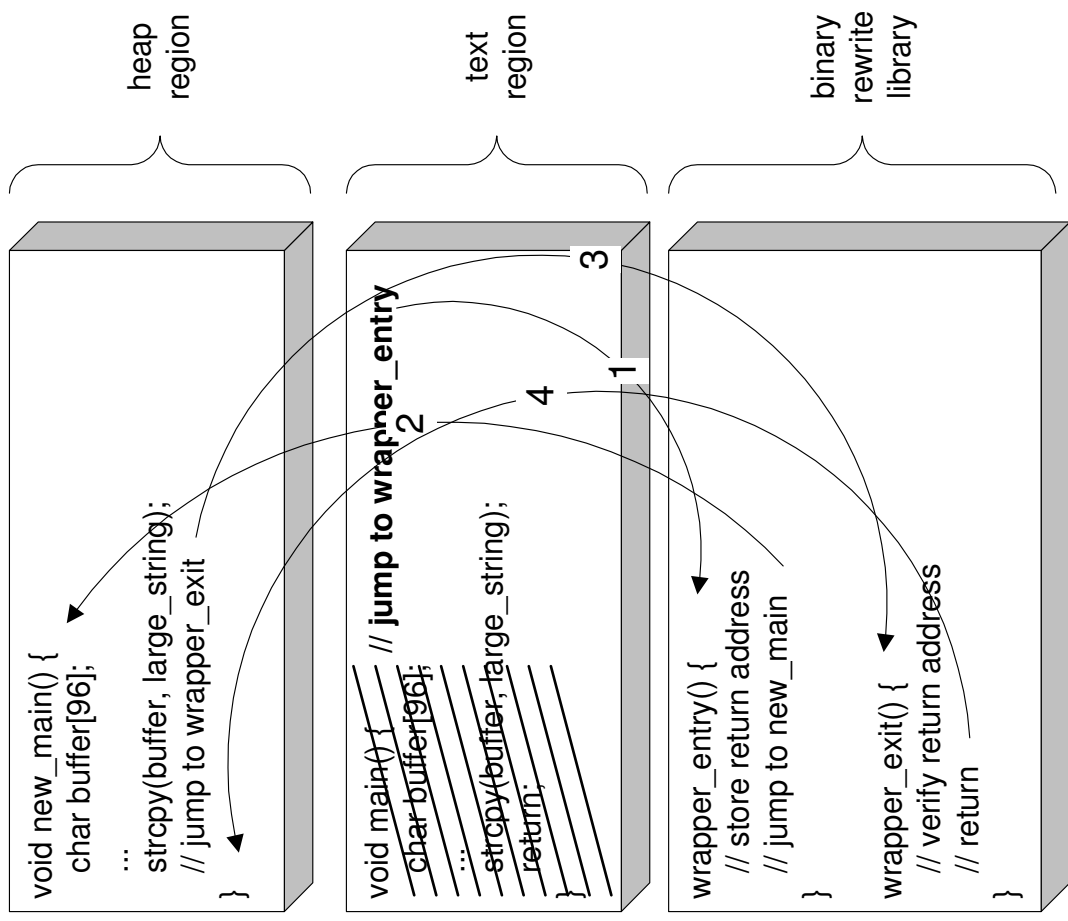
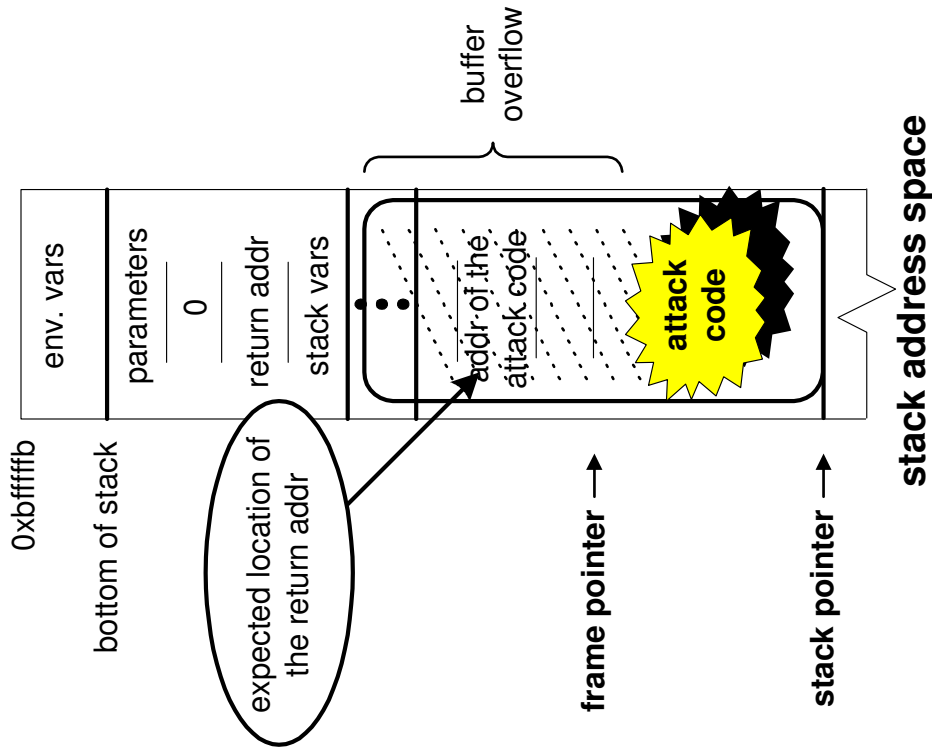
Programs with known buffer overflows

Security attacks gain root access to machines running any one of these programs:

Program name	Version	Description
xlockmore	3.1	Locks an X Window display
amd	6.0	Auto remote file system mount
imapd	3.6	IMAP mail server
elm	2.5PL8	ELM email agent
SuperProbe	2.11	Probe and identify video hardware

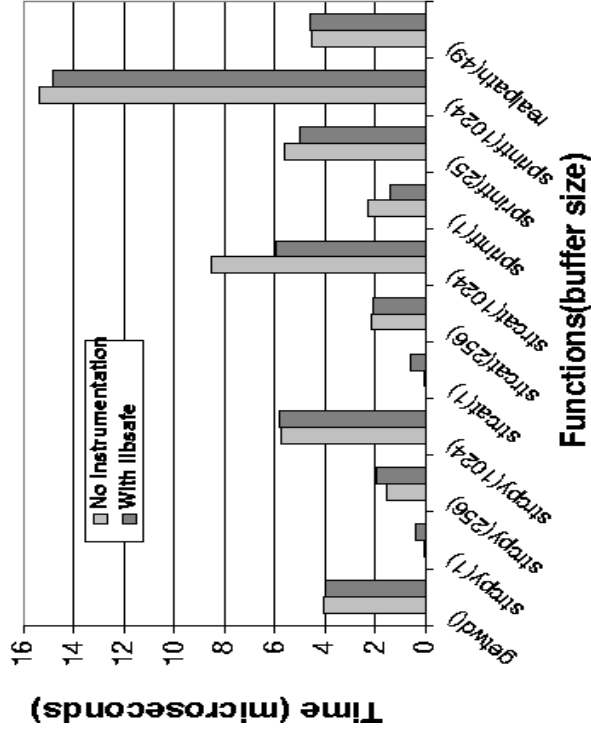
Libsafe demonstrated its ability to detect and terminate such security attacks.

Libverify

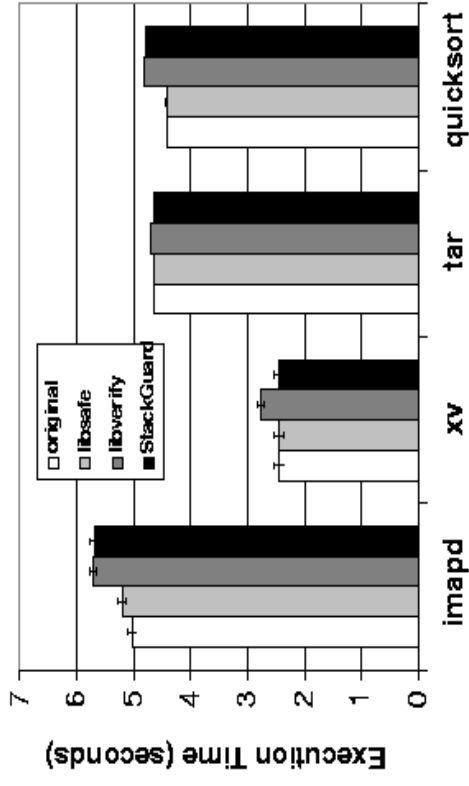


Performance

Micro benchmarks



Application benchmarks



Performance results are very encouraging: negligible overhead.

Where to get libsane

- <http://www.bell-labs.com/org/11356/libsane.html>
- Linux distributions:

– Red Hat



– Debian



– TurboLinux



– Mandrake



– Slackware



– Yggdrasil



Related work

- **Non-executable stacks (Openwall Project)**
- **Stack check in custom libc (Snarskii)**
- **Compile-time checks (Lint, Evans, Jones, Wagner, ITS4)**
- **Dynamic checks (StackGuard, MemGuard, StackShield)**
- **Sandboxing (Janus)**

Special cases

- **Libc incompatibilities**
- **Absence of frame pointer code**
 - **Disable libsafe checking if**
 - **main() does not contain frame pointer code**
 - **all stack frame pointers are not ...**
 - **word aligned**
 - **monotonically increasing**
 - **between addr and stack_start**
 - **last frame pointer is at known position and points to 0x0**