StackGuard
Protecting Against Buffer Overflows
Part I – Detailed Overview

By Konstantin Rozinov

Polytechnic University

February 19, 2003

Outline Of Presentation

- What is a buffer?
- What is a buffer overflow?
- How does a buffer overflow work?
- What is StackGuard?
- How does StackGuard work?
- Live Demo
- Limitations and Defeats of StackGuard.
- Conclusions
- Links

What Is A Buffer?

- Buffers: a contiguous place in memory where bits can be put (i.e. Arrays).
  - Heap-allocated: Using malloc, calloc, realloc to allocate buffers during runtime via pointers.
  - Stack-allocated: Used to store function (local) variables, parameters, and almost everything else (stack frames).
  - Global variables and static variables are held in the data segment and the BSS (Block Started by Symbol).

What Is A Buffer Overflow?

- 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, with the extra bits.
- As you can guess, this can lead to a serious security problem.

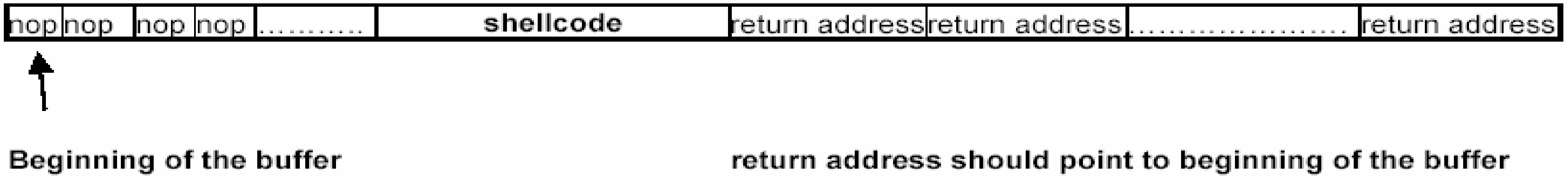
How Does A Buffer Overflow Work?

1. Overflow a buffer within a function.
2. Overwrite the return address of the function with another address (where your attack code starts).
3. The attack code is usually inside the original buffer (hopefully it’s big enough).
4. Execute the code, get root.

How Does A Buffer Overflow Work?

The egg contains the shellcode, NOPs, and return addresses, as can be seen below. The shellcode is a simple program, compiled (gcc), and disassemble with gdb:

"\xeb\x1d\x5e\x29\xc0\x88\x46\x07\x89\x46\x0c\x89\x76\x08\xb0\x0b\x87\xf3\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\x29\xc0\x40\xcd\x80\xe8\xde\xff\xff\xff/bin/sh"



What Is StackGuard?

- A small patch for the GNU gcc compiler, which enhances the way gcc generates the code for setting up and tearing down functions (activation records – stack frames). Specifically, these functions are the `function_prolog()` and `function_epilog()` routines.
- Stack-smashing attacks exploit buffer overflows within the stack. StackGuard detects and defeats stack smashing attacks by protecting the return address on the stack from being altered.

How Does StackGuard Work?

- StackGuard places a “canary” word next to (prior) the return address on the stack.
- Once the function is done, the new tear down code from gcc first checks to make sure that the canary word is unmodified and intact before jumping to the return address.
- If the integrity of canary word is compromised, the program will terminate.

How Does StackGuard Work?

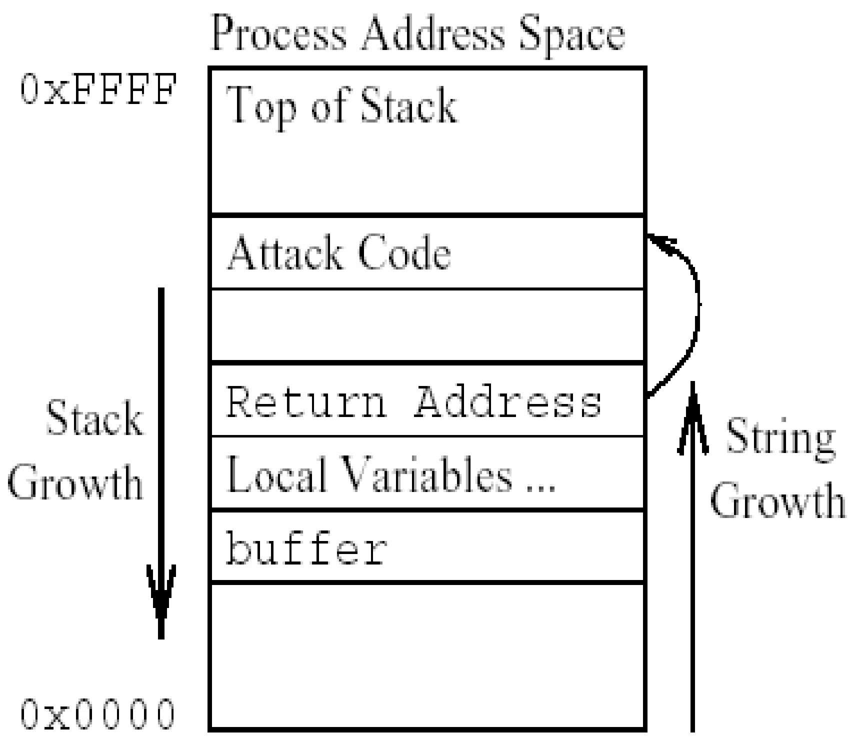


Figure 1: Stack Smashing Buffer Overflow Attack

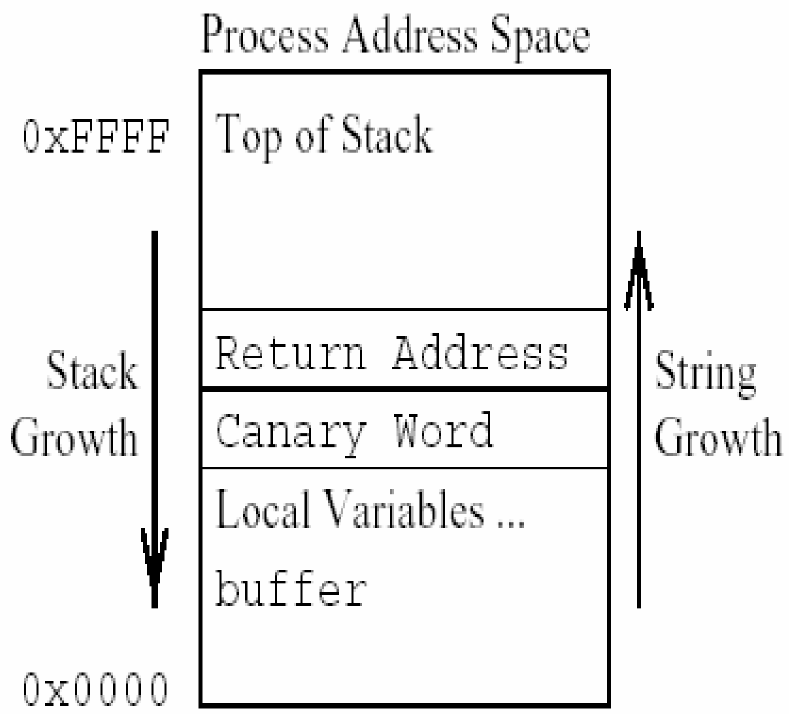


Figure 2: Canary Word Next to Return Address

How Does StackGuard Work?

- To prevent the forging of the canary from within the overflow string (the egg), StackGuard can do two things:
  - Use a Terminator Canary
  - Use a Random Canary
  - Use a Null Canary (not used anymore)

How Does StackGuard Work?

- Use a Terminator Canary: comprised of common termination symbols for C standard string library functions - \0, CR, LF, and -1 (EOF). The egg cannot contain these symbols and thus cannot be copied into the memory space. The one used in StackGuard 2.01 is 0x000aff0d.
- 0x00 (null) will stop strcpy(), 0x0a will stop gets(), etc. return address will not be overwritten.

How Does StackGuard Work?

- Use a Random Canary: a 32-bit random number chosen at program runtime. The attacker cannot learn the canary value prior to program start by searching the executable image. The random value is taken from /dev/urandom if available, and created by hashing the time of day if /dev/urandom is not supported.

How Does StackGuard Work?

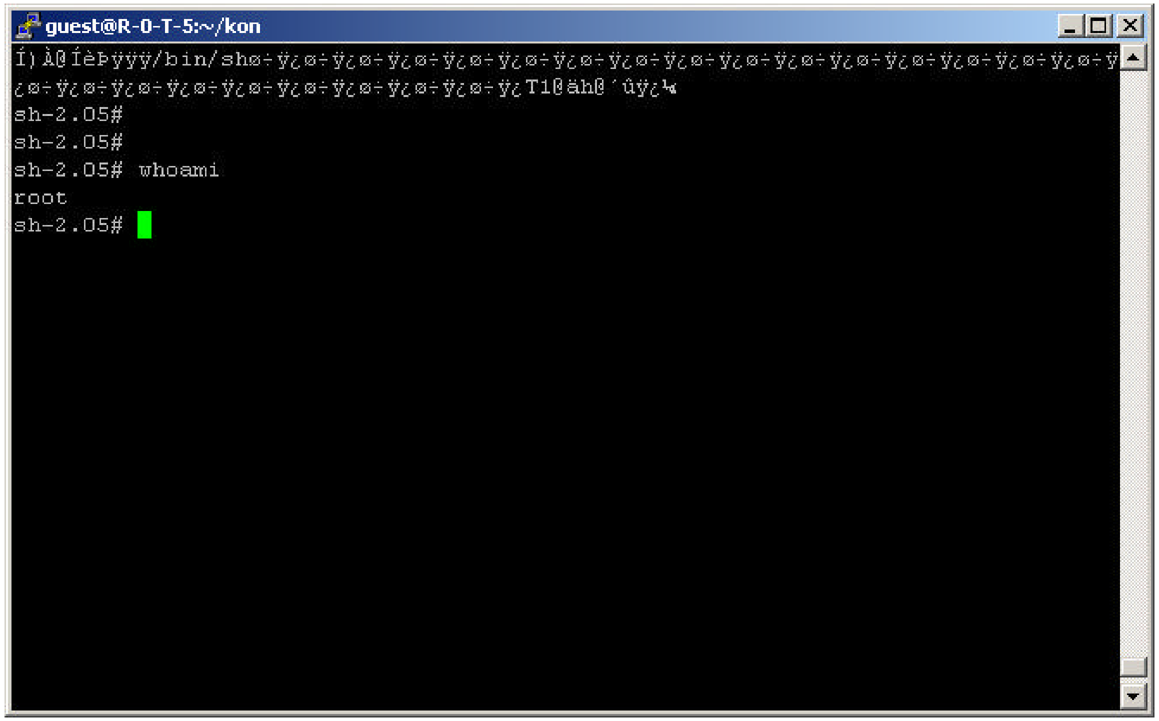
- Use a Null Canary the canary word is "null", i.e. 0x00000000. Since most string operations that are exploited by stack smashing attacks terminate on null, the attacker cannot easily spoof a series of nulls into the middle of the string. The Null Canary has been superseded by the Terminator Canary.

Live Demo

- Compile rootecho (CS392 – hw#5) without StackGuard.
  - Should be able to get root shell.
- Compile rootecho with StackGuard.
  - The program should terminate and not result in a root shell.

Live Demo

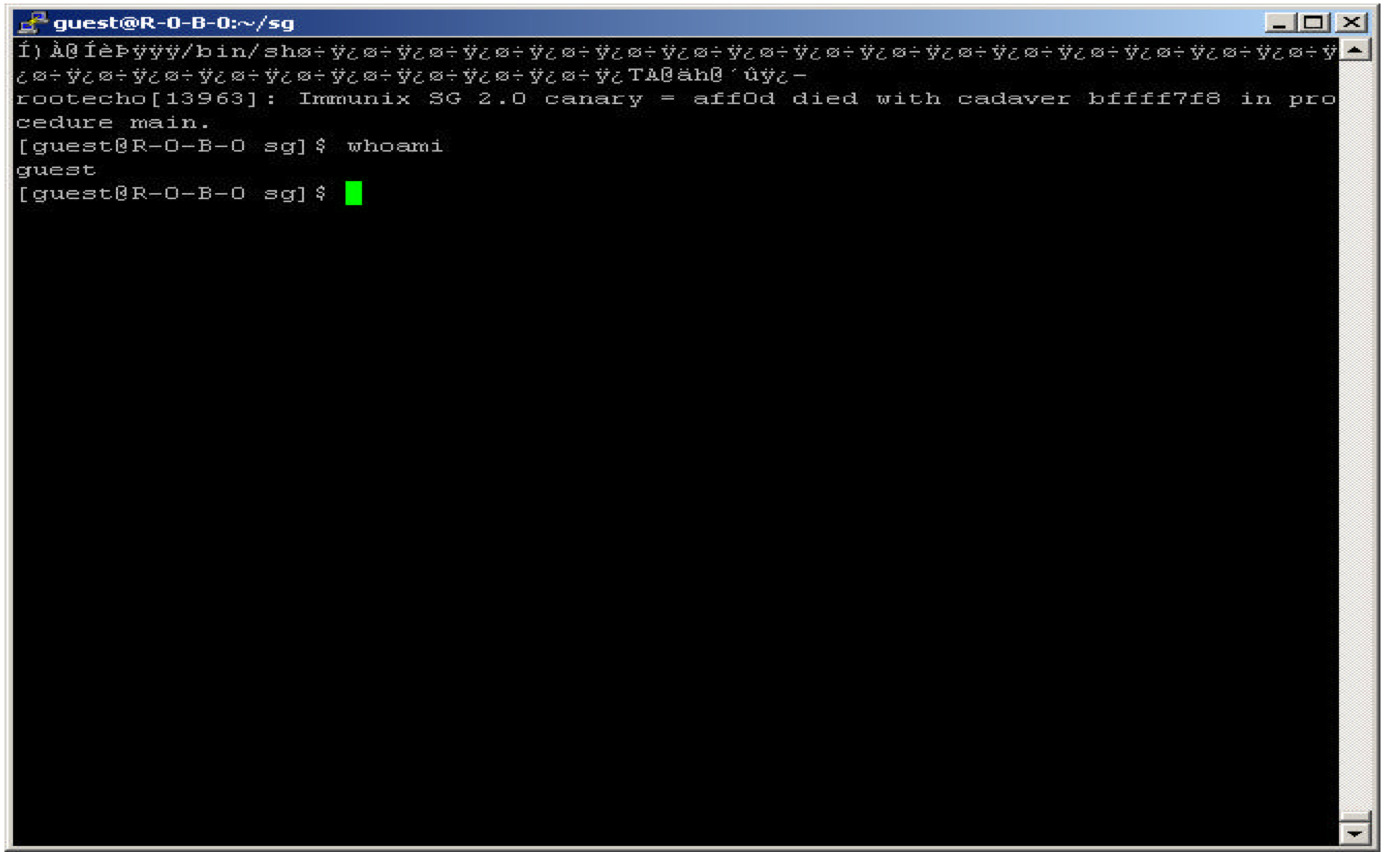
- Without StackGuard:



```
Í)À@ÍèÞÿÿÿ/bin/shø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ
¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿T1@äh@´ûÿ¿¼
sh-2.05#
sh-2.05#
sh-2.05# whoami
root
sh-2.05#
```

Live Demo

- With StackGuard:



```
Í)À@ÍèÞÿÿÿ/bin/shø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ
¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿ø÷ÿ¿TA@äh@´ûÿ¿–
rootecho[13963]: Immunix SG 2.0 canary = aff0d died with cadaver bffff7f8 in pro
cedure main.
[guest@R-0-B-0 sg]$ whoami
guest
[guest@R-0-B-0 sg]$
```

Limitations and Defeats of StackGuard

- Function parameters are not protected.
- Frame pointers can be altered.
- Local variables can be controlled.

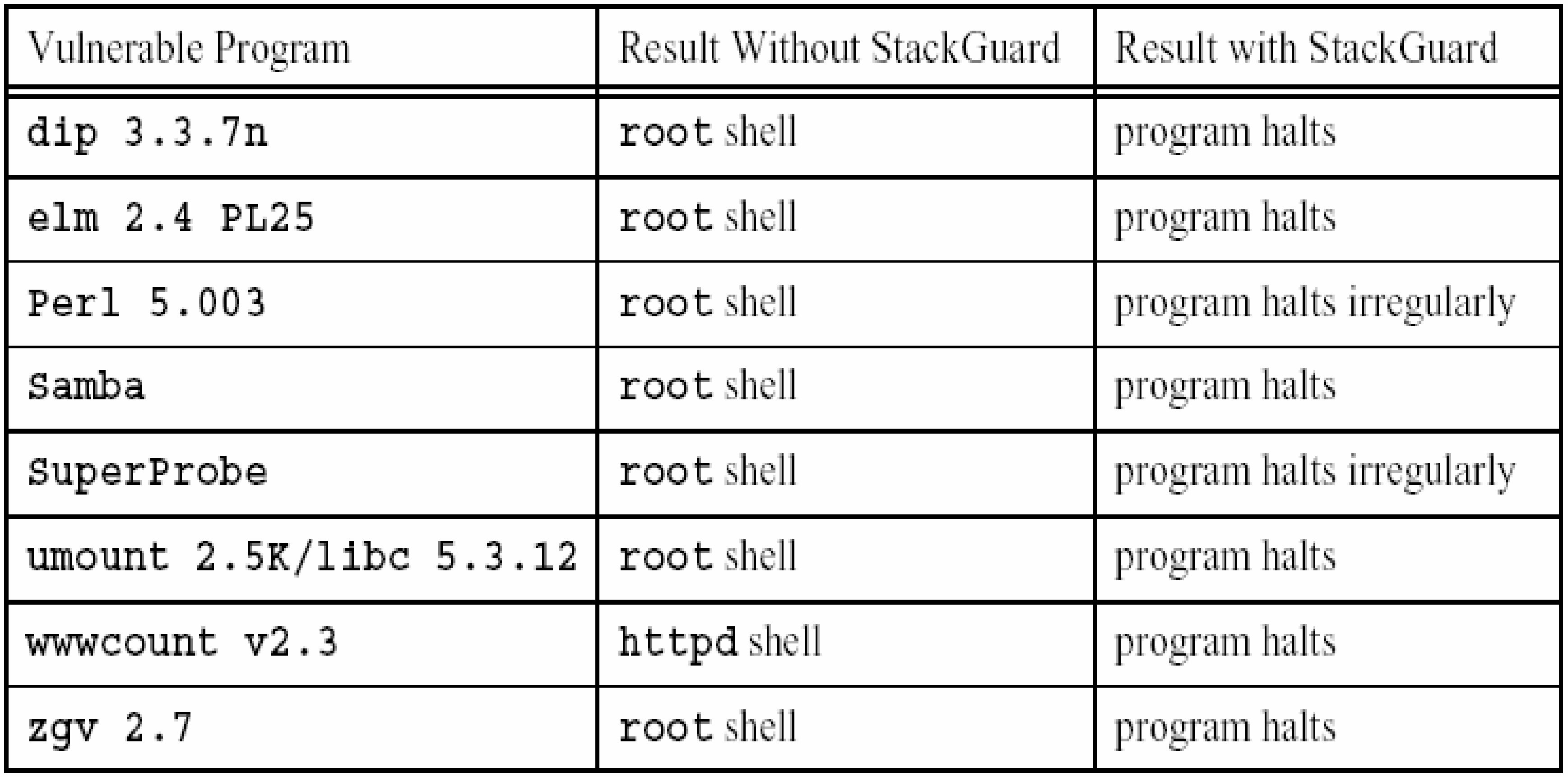
- Solution: Illegal frame pointer alterations can be stopped by placing the canary prior to it instead of after it. (StackGuard 3.0 will do this). However, local variables and function parameters will still be vulnerable.

Conclusions

- StackGuard has been shown to be effective against stack smashing attacks, while preserving virtually all of system compatibility and performance.
- The creators have built an entire Redhat Linux distribution protected with StackGuard and it is functional and being used on production machines.

Conclusions – Penetration Resistance

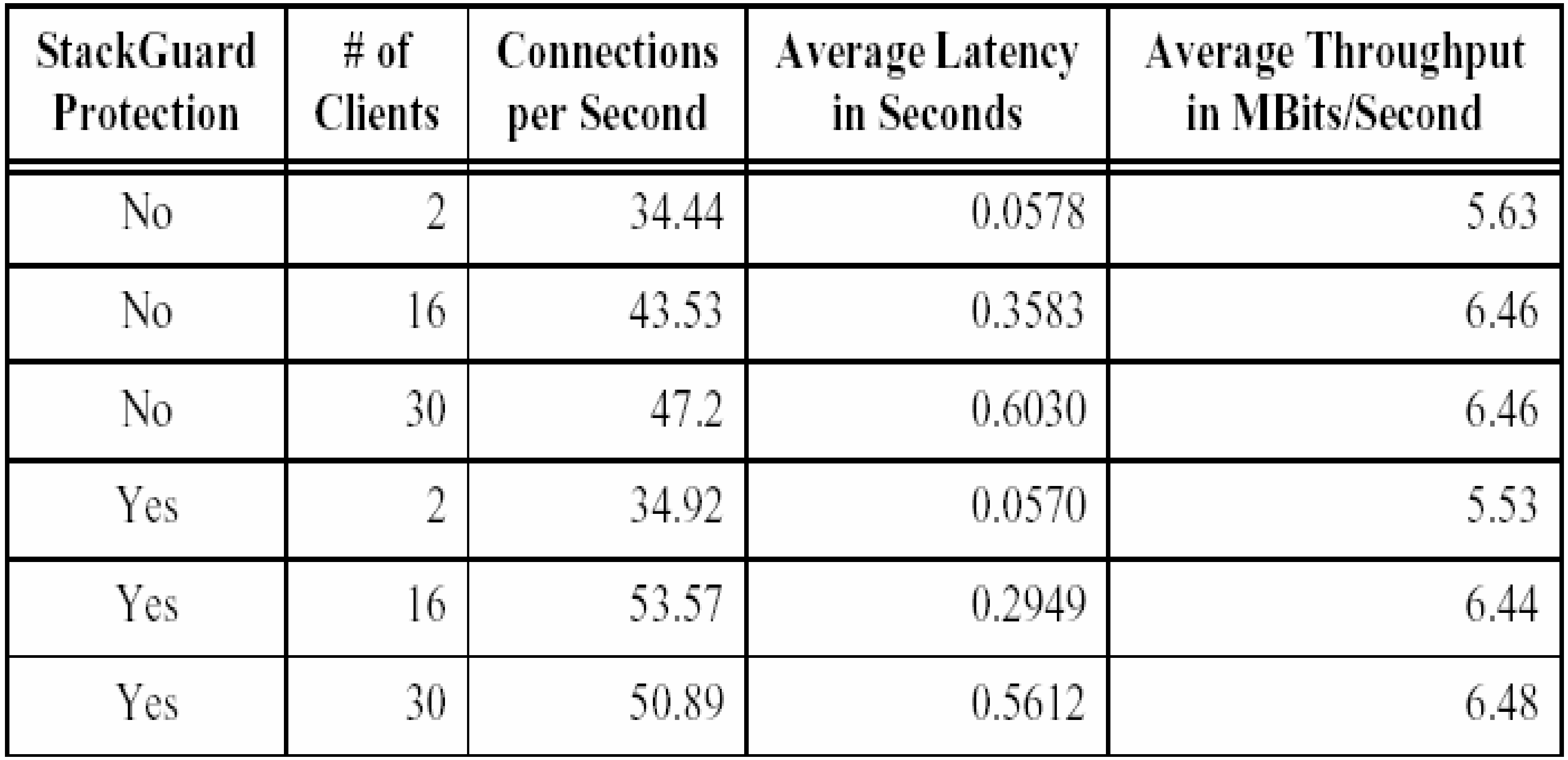
Table 1: StackGuard Penetration Resistance



Vulnerable Program	Result Without StackGuard	Result with StackGuard
`dip 3.3.7n`	`root` shell	program halts
`elm 2.4 PL25`	`root` shell	program halts
`Perl 5.003`	`root` shell	program halts irregularly
`Samba`	`root` shell	program halts
`SuperProbe`	`root` shell	program halts irregularly
`umount 2.5K/libc 5.3.12`	`root` shell	program halts
`wwwcount v2.3`	`httpd` shell	program halts
`zgv 2.7`	`root` shell	program halts

Conclusions – Performance

Table 2: Apache Web Server Performance With and Without StackGuard Protection



StackGuard Protection	# of Clients	Connections per Second	Average Latency in Seconds	Average Throughput in MBits/Second
No	2	34.44	0.0578	5.63
No	16	43.53	0.3583	6.46
No	30	47.2	0.6030	6.46
Yes	2	34.92	0.0570	5.53
Yes	16	53.57	0.2949	6.44
Yes	30	50.89	0.5612	6.48

Links

- http://www.immunix.org/stackguard.html
  - Many good links here about StackGuard
- http://isis.poly.edu/courses/cs392-f2002/labs/lab5.pdf
  - How to create and implement buffer overflows
- http://ouah.sysdoor.net/gerastackguard.pdf
  - How to bypass the StackGuard protection mechanism