

# *Security Presentation*

Dave Rohret – Cyber-terrorism

Shane Wright – Tool Development

Jason Cochetti – Malicious Logic

# *The Problem: Cyber-terrorism*

- interception of information (man-in-the-middle)
- Website defacing
- credit card and data thefts
- corporate intrusions
- denial of service (DOS) attacks

# Current Status of S/W Security

## ◆ Feb 2002 Security Connections:

- ◆ 2001 Most Active Year for Computer Related Crimes

- ◆ Security Threats are Expected to Proliferate in 2002

  - ◆ Wireless, Smart Phones and PDAs

- ◆ CERT Coordination Center recorded a 300% Increase for Successful Intrusions in 2002

## ◆ Survey Conducted by CERT

- ◆ 71% of Public VERY Concerned about Internet and Computer Security

- ◆ 74% are worried about their personal information on the Internet being Compromised

- ◆ 78% are concerned about their Government Held Data being Misused

## ◆ Report on Cyber Crime

- ◆ Calls U.S. Business on Internet Infrastructure “America’s Achilles Heel”

- ◆ Estimates Number of Hackers with Credible Skills to Reach 20 Million

# Current Status of S/W Security

- ◆ Newsbytes, Jan 22, 2002
  - 27% of U.S. & Canadian Banks Breached
  - 18% of Medical/Healthcare and Telecommunications Industry Breached
  - 12% of On-Line Corporate Databases Breached
- ◆ Security Focus ARIS Report, 22 Feb 2002
  - Attacks Worldwide:
    - Today: 1,265,528
    - 7 Days: 13,695,294
    - To Date for 2002: 599,229,333
  - Attack Types Seen:
    - Today: 635
    - 7 Days: 1,260
    - To Date for 2002: 9,777
  - Attacking IPs:
    - Today: 36,450
    - 7 Days: 188,993
    - To Date for 2002: 3,122,060

**“An area of concern in corporate security is the occurrence of exploits in databases, which house a corporations crown jewels, and their lack of protection”, Joe McKinderick, Evans Data.**

# Level of Attacker Expertise

- ◆ Level 1: Beginning or Intermediate Knowledge of Target OS
  - Uses Exploit Scripts Found on the Internet
- ◆ Level 2: Intermediate Knowledge of Target OS
  - Able to Use Software that Must First be Compiled
  - Able to Make Minor Changes to Scripts and Source Code
- ◆ Level 3: Advanced Knowledge of Target OS
  - Able to Compile Source Code Even if Additional Libraries are Required
  - Able to Make Significant Changes to Scripts and Source Code
  - Generally Capable of Hiding Evidence
  - Has Resources to Test Exploits Before Releasing them in the Wild
- ◆ Level 4: Advanced Knowledge of Multiple OSs
  - Authors Exploits
  - Adept in Hiding Evidence
  - Able to Change Compromised Systems Configurations
  - Have Extensive Access to Resources to Test and Gather pre-exploit Information

# Tool Development

## Large Scale Development

- IDS/Firewalls
- Scanners

Examples: Nessus, ISS

## Tools Based on the Unix Tradition

- Do one thing and do it well
- Flexible and Robust

Examples: Nmap, Netcat

# Categorization

1. Host/Network Enumeration
2. Exploits
3. Rootkits---Trojans
4. Investigative

# Exploits

- **Input Validation**
  - Improper parsing of user supplied input. Found often in web applications.
- **Heap Overflows**
  - Typically allows execution of arbitrary instructions
- **Format String**
  - Ability to overwrite arbitrary areas in memory.
- **Buffer Overflows**
  - Typically allows execution of arbitrary instructions.

# Buffer Overflow

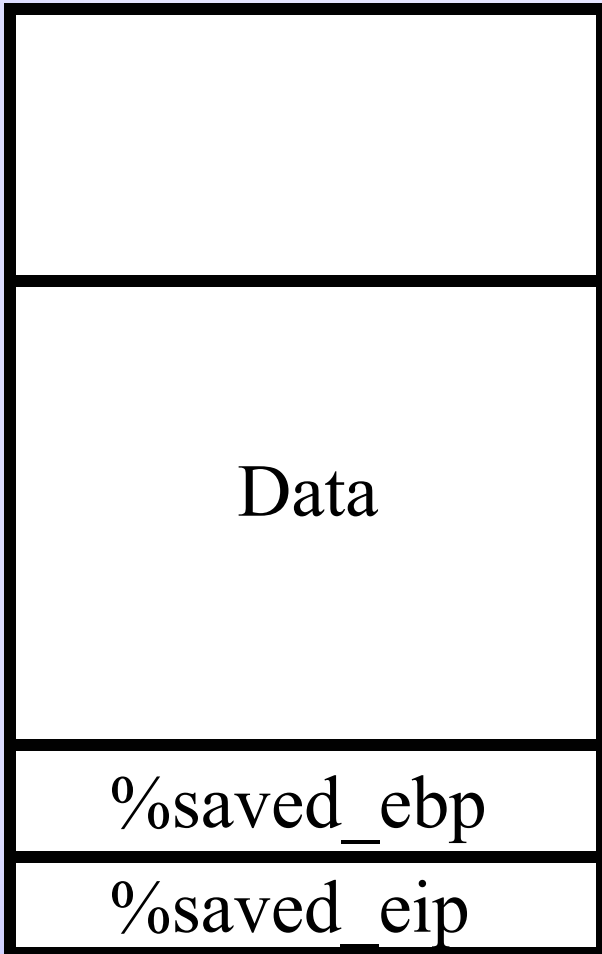
## The Poisoned NUL Byte

Olaf Kirch Bugtraq 1998

Demonstrated that an off by one error in the libc implementation of `realpath()` could be used to execute arbitrary instructions.

# Procedure Prolog

0x00000000



%esp

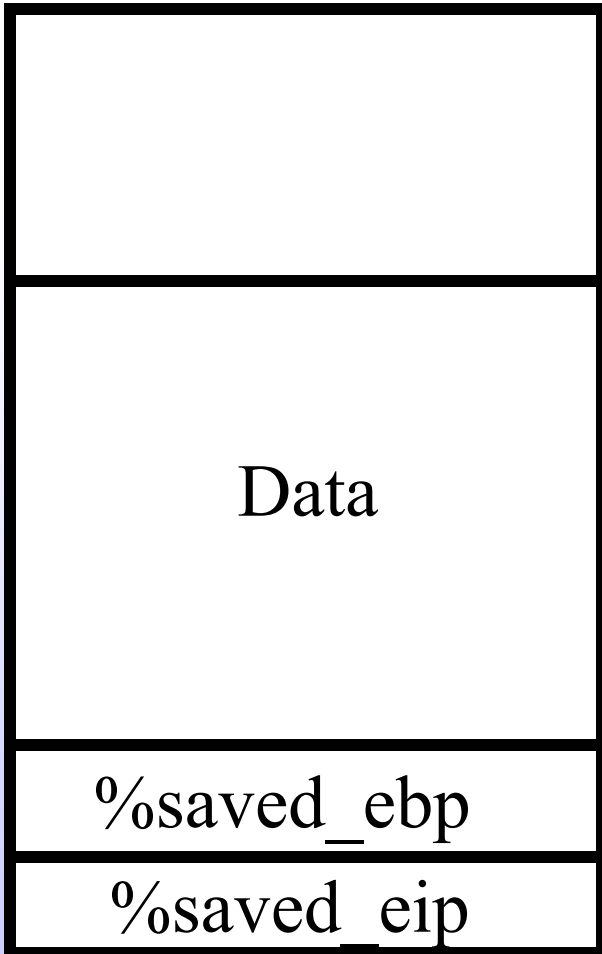
```
pushl %ebp  
movl %esp,%ebp  
subl $0xFF,%esp
```

%ebp = %old\_esp

0xffffffff

# Procedure Epilog

0x00000000



```
movl %ebp,%esp  
popl %ebp  
ret
```

$\%esp = \%ebp$

0xffffffff

# Off By One

0x00000000

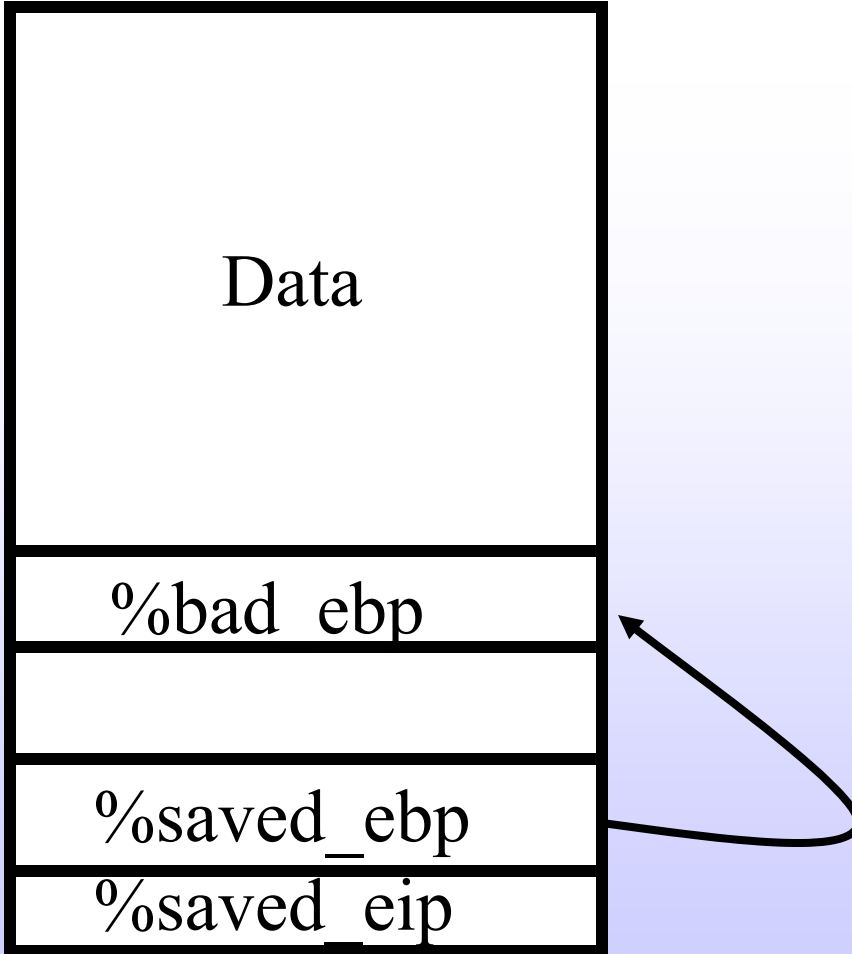
Data

%bad ebp

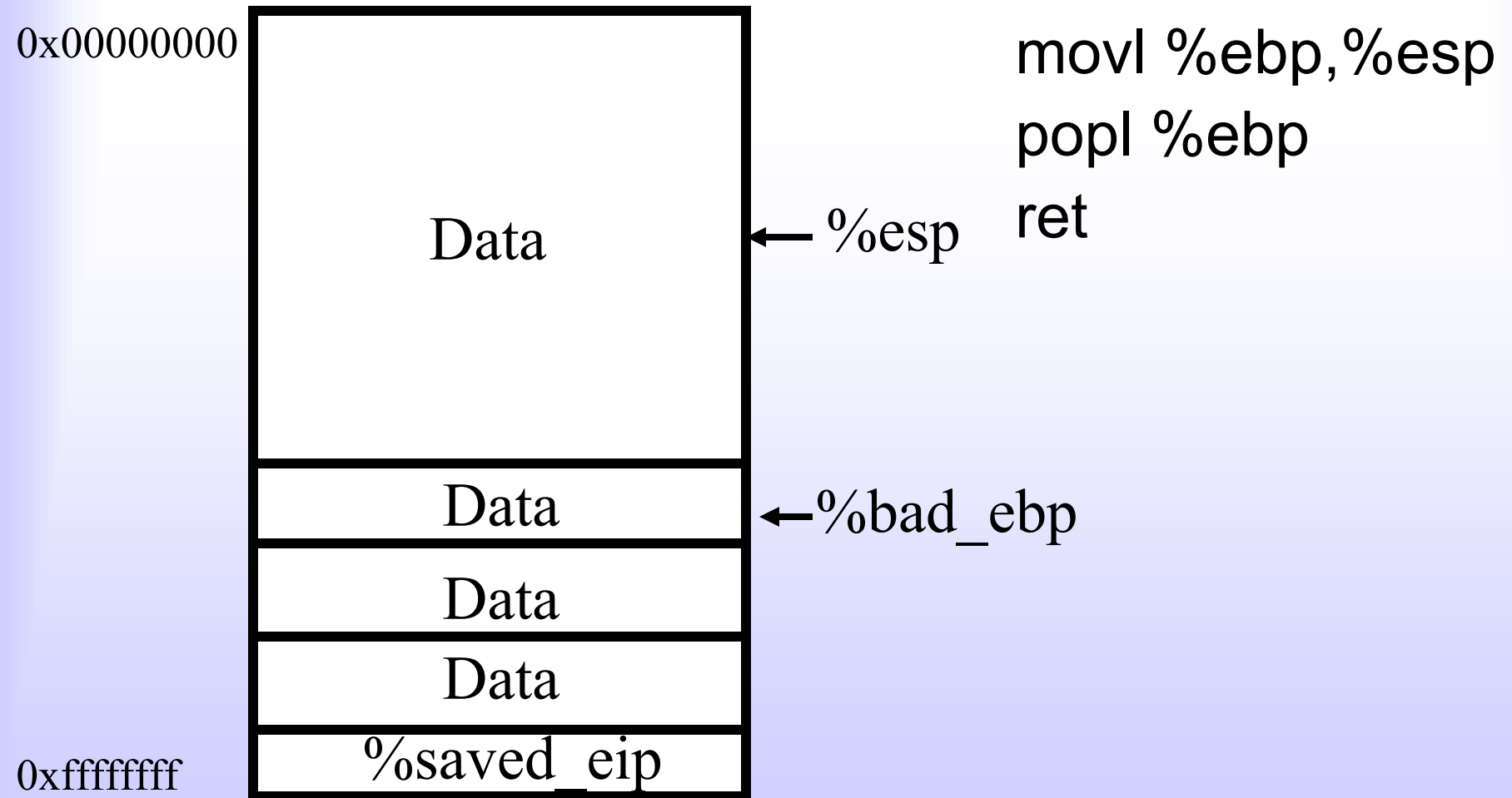
%saved\_ebp

0xffffffff

%saved\_eip

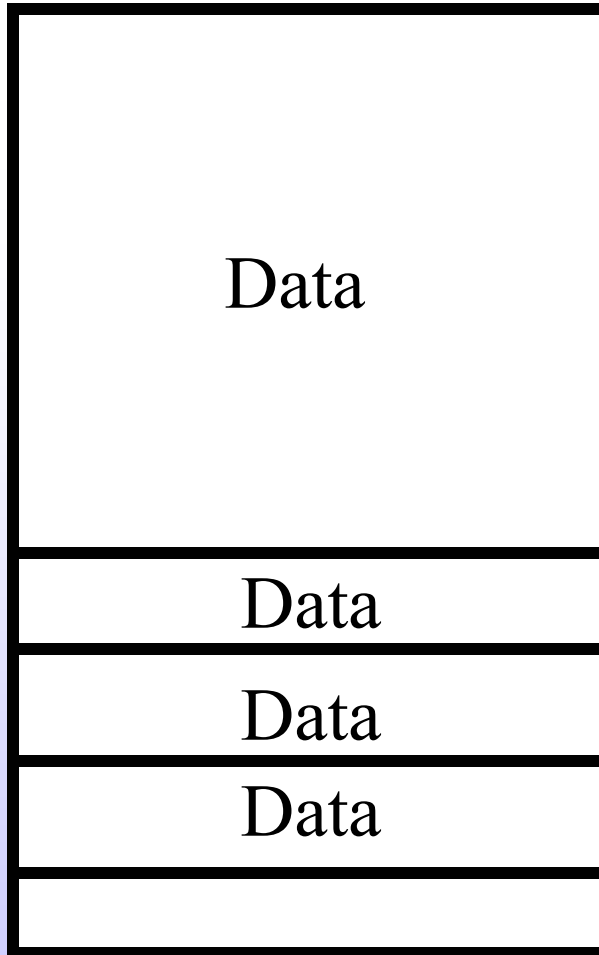


# Off By One



# Off By One

0x00000000



```
movl %ebp,%esp  
popl %ebp  
ret
```

```
%bad_esp → %esp  
pop arbitrary data  
ret arbitrary data
```

# What Is In The Data Area?

Shell Code.

Cpu Instructions Written In Assembler For Specific Platforms.

NOP Sled

# Future Of Exploitation

- IDS Evasion
- Polymorphism
- Cohesive Environments

# Conclusions

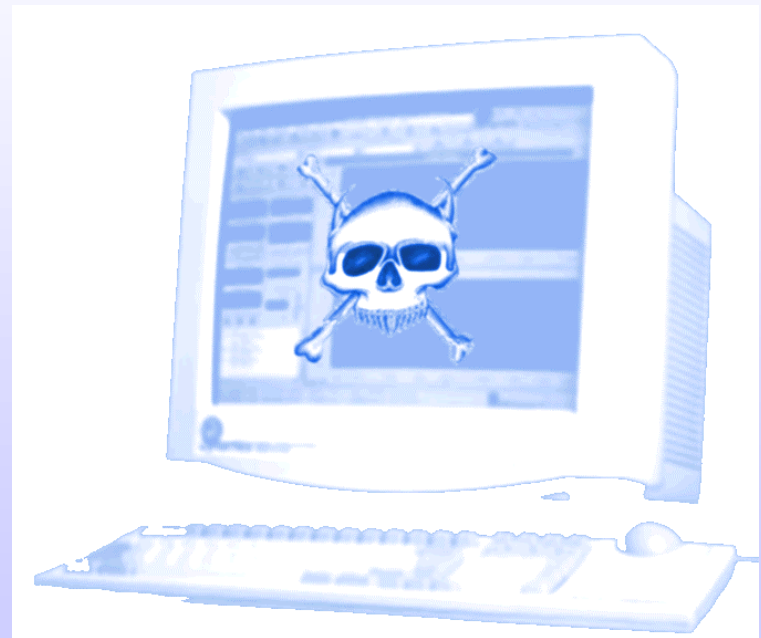
# Malicious Logic

Defined:

A program implemented in hardware, firmware, or software, and whose purpose is to perform some unauthorized or harmful action.

Examples of malicious logic are

- ◆ Logic bombs
- ◆ Trojans
- ◆ Viruses
- ◆ Worms



# Types Malicious Logic

**Logic bomb:** Code that is triggered by some event or condition that will cause damage or corruptions of a particular system.



Omega  
Engineering  
Consultants

**Trojan:** A program containing hidden code that executes in the background and/or silently in addition to running the normal program.



# Types Malicious Logic

**Virus:** A program or code that replicates, that is infects another program, boot sector, partition sector or document that supports macros by inserting itself or attaching itself to that medium. Most viruses just replicate, a lot also do damage.

When Linux.Jac.8759 is executed, it starts by checking all files that are in the same directory as the one from which the virus was executed. If it finds executable files that have write permission, it attempts to infect them.

**Worm:** A program that makes copies of itself, for example from one disk drive to another, or by copying itself using email or some other transport mechanism. It may do damage and compromise the security of the computer. It may arrive in the form of a joke program or software of some sort.

The CodeRed Worm uses a known buffer overflow vulnerability to replicate itself on Microsoft Windows NT 4.0 and Windows 2000 that run IIS 4.0 and 5.0 Web servers.

# Working with Malicious Logic

Two approaches:

## **Detecting**

- ◆ Code received from exploits needs to be free of malicious logic.
- ◆ 3rd party software.

## **Insertion / Development**

- ◆ Extra features needed to be added to current software.
- ◆ Hidden information
- ◆ Logging
- ◆ Reporting

# Approaches for Detecting

## Preliminary Scan of Code

1. Look for functions that can be misused.
2. Identify each line of the code.
3. Look for usual code.
4. Look for tagging.
5. Comment unknown.

## Preliminary Scan of Binaries

1. Size of the Binary
2. Checksums / MD5sum
3. Running `'strings'`
4. De-compiler



# Approaches for Detecting

## Shellcode Evaluation

1. Take shellcode and covert it back to assemble level language.
2. Verify against known (safe) shellcode.
3. Attempt to reproduce the shellcode.

```
char shellcode[]=
    "\x20\xbf\xff\xff" /* bn,a <shellcode-4> */
    "\x20\xbf\xff\xff" /* bn,a <shellcode> */
    "\x7f\xff\xff\xff" /* call <shellcode+4> */
    "\x90\x03\xe0\x20" /* add %o7,32,%o0 */
    "\x92\x02\x20\x10" /* add %o0,16,%o1 */
    "\xc0\x22\x20\x08" /* st %g0,[%o0+8] */
    "\xd0\x22\x20\x10" /* st %o0,[%o0+16] */
    "\xc0\x22\x20\x14" /* st %g0,[%o0+20] */
    "\x82\x10\x20\x0b" /* mov 0x0b,%g1 */
    "\x91\xd0\x20\x08" /* ta 8 */
    "/bin/ksh";
```

# Approaches for Detecting

## Running the program / code (local).

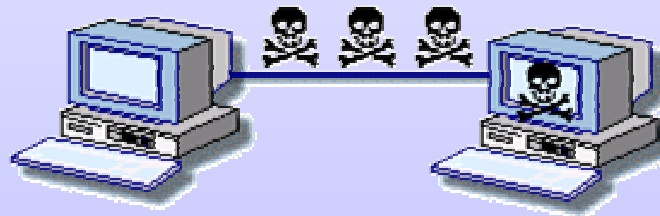
1. Using 'strace' to look at system calls.
2. Watching the process and/or processes on the machine.
3. Attaching gdb to the running process.



# Approaches for Detecting

## Running the program / code (remote concerns).

1. Set up a network scanner to watch packets on the network before, during, and after running an exploit.
  - a. Are the connections legitimate?
  - b. Unknown packets being sent (bad IP address not on your network).
  - c. Connection to the outside network.
2. Simulating a remote machine to receive the information.
3. Attach a debugger to the targets machine process and step through the exploit/code as it is being launched from the attacking machine.



# Approaches for Detecting

## Malicious Residue

1. Does the exploit leave behind a backdoor?
  - a. Scanning your machine for open ports. That weren't open before.
  
2. Can you verify that the integrity of remote and the attacking machine?
  - a. Have any files been touched/written to recently that should have been?
  - b. New files?
  - c. Checking you system with a rookit detector.
  - d. Usual Occurrences

# Conclusion

- ◆ Malicious logic
- ◆ Examples: Logic bombs, Trojans, Viruses, Worms
- ◆ Detection
  - Preliminary Scan
  - Shellcode examination
  - Running the program
    - Looking for local events
    - Watching the network
  - Residue