



# AVAYA

## Buffer Overflow Vulnerabilities and Solutions

Timothy Tsai  
Avaya Labs Research  
ICC 2002  
April 29, 2002

**AVAYA**  
*labs*



# Overview

- **How significant are buffer overflows?**
  - **Context of overall security picture**
- **What are buffer overflows?**
- **What are the possible solutions?**

# The Twenty Most Critical Internet Security Vulnerabilities

Version 2.503 April 8, 2002; Copyright 2001-2002, The SANS Institute

## Top Vulnerabilities That Affect All Systems (G)

- G1 - Default installs of operating systems and applications ✓
- G2 - Accounts with No Passwords or Weak Passwords
- G3 - Non-existent or Incomplete Backups
- G4 - Large number of open ports
- G5 - Not filtering packets for correct incoming and outgoing addresses
- G6 - Non-existent or incomplete logging
- G7 - Vulnerable CGI Programs

## Top Vulnerabilities to Windows Systems (W)

- W1 - Unicode Vulnerability (Web Server Folder Traversal)
- W2 - ISAPI Extension Buffer Overflows ✓
- W3 - IIS RDS exploit (Microsoft Remote Data Services)
- W4 - NETBIOS - unprotected Windows networking shares
- W5 - Information leakage via null session connections
- W6 - Weak hashing in SAM (LM hash)

## Top Vulnerabilities To Unix Systems (U)

- U1 - Buffer Overflows in RPC Services ✓
  - U2 - Sendmail Vulnerabilities ✓
  - U3 - Bind Weaknesses ✓
  - U4 - R Commands
  - U5 - LPD (remote print protocol daemon) ✓
  - U6 - sadmind and mountd ✓
  - U7 - Default SNMP Strings
- ✓ = buffer overflow vulnerability

# What Are the Problems?

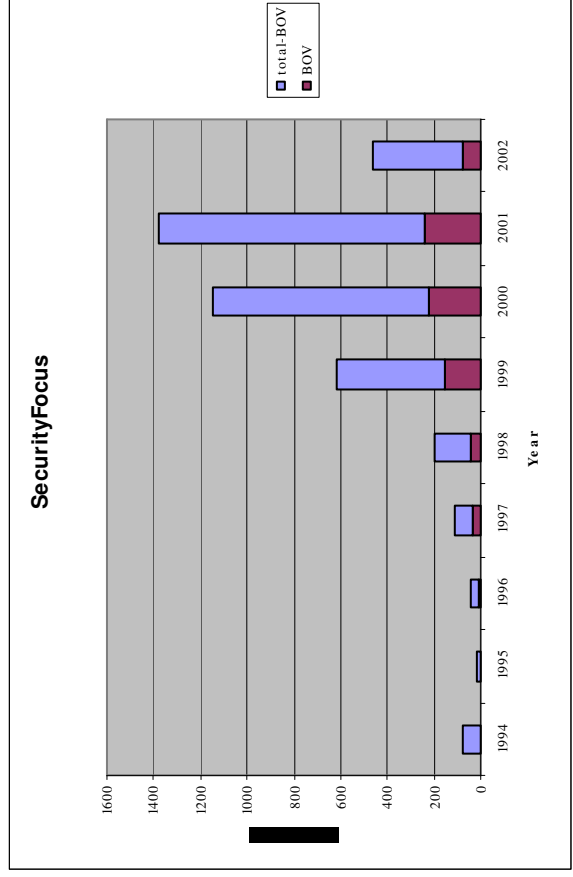
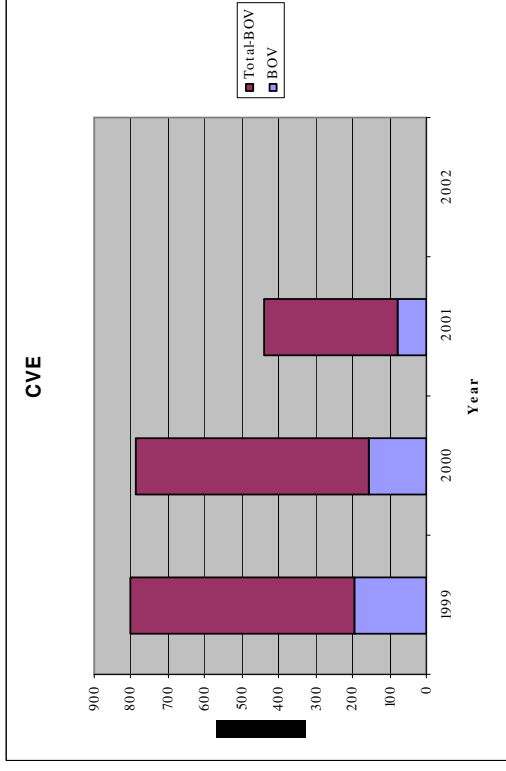
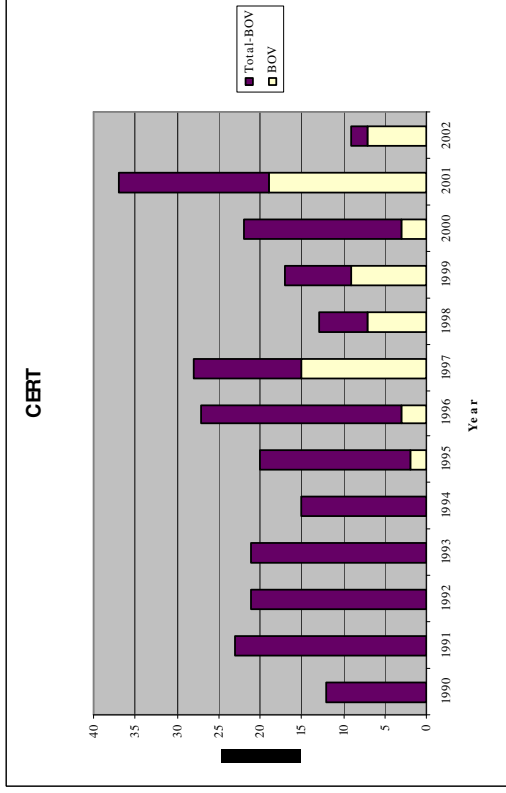
- Buggy software design and development
  - Programming for security is not taught
  - Good software engineering processes are not universal
  - Legacy code
- System administration
  - Too many machines to administer
  - Too many platforms and applications to support
  - Too many updates and patches to apply
  - Administrators may be handcuffed (e.g., not allowed to block mail attachments)
  - Not all administrators are up-to-date (e.g., desktop and home users)
    - Too many unnecessary services
    - Inadequate policies and procedures
- New technologies
  - The Internet, wireless, converged communications, commercial web sites
  - Vulnerabilities are quickly and widely published
  - Scripts and tools are downloadable (giving rise to "script kiddies")
  - Large pool of victims
  - Remote attacks are easy to perpetrate (cheap access, high-speed connections)

# Solutions

- Education
- Deployment of known technologies (firewalls, VPN, encryption)
- Better processes (design and code reviews, testing)
- Tools
  - Software development tools (testing, code scanning)
  - Software update tools (patches, virus updates)
  - Security audit tools (passwords, ports, services)
  - Run-time instrumentation

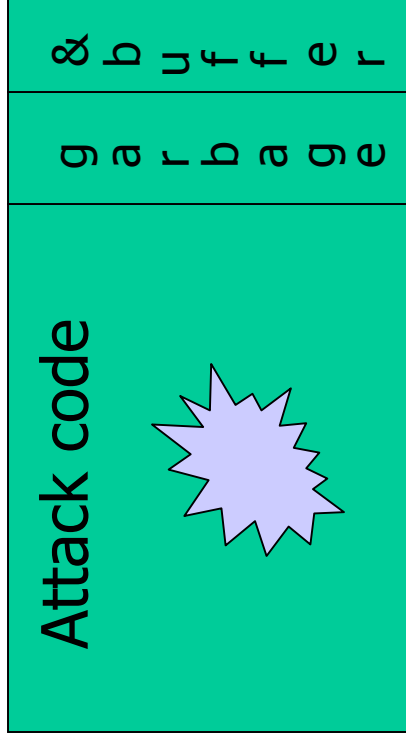


# Significance of Buffer Overflows



# What is a Buffer Overflow?

Buffer (80 bytes) fp ra



```
void foo(char * input_string)
{ char buffer[80];
  strcpy(buffer, input_string);
  return;
} /* input_string =
   attack code+garbage+&buffer
   total length >= 88 bytes */
```

# Consequences of Buffer Overflows

## Common attack techniques

- Find vulnerable programs
  - Target root processes
  - “Fuzz” testing
  - Use source code and debugger to examine stack contents
- Write code
  - Connect to port
  - Send data, including shellcode

## Consequences

- Root shell from remote attack site
  - Compromise other machines (passwords)
  - Stage DDoS, man-in-the-middle attacks
  - Monitor LAN traffic
- Denial of service



# Buffer Overflow Solutions

- **Install patches**
  - Use checklists and web sites
  - Use update and audit tools
- **Better programming**
  - Use safe functions (snprintf instead of sprintf)
  - Use safe languages
  - Conduct code reviews
  - Use code scanners (ITS4, Splint)
  - Better testing
- **Run-time instrumentation**
  - Kernel patches (Openwall, PaX)
  - Compiler solutions (StackGuard, ProPolice, StackShield)
  - Library solutions (Snarskii, Libsafe, BOWall)

# Kernel Patches

- Mechanism
  - Make parts of memory (stack, heap, data) non-executable
  - Allow overflow to occur, but generate memory exception when executing attack code
- Pros
  - Automatically applied to all processes
  - Can catch some overflows in heap and data space
- Cons
  - Requires kernel recompilation
  - Doesn't catch return-into-libc exploits

# Compiler Solutions

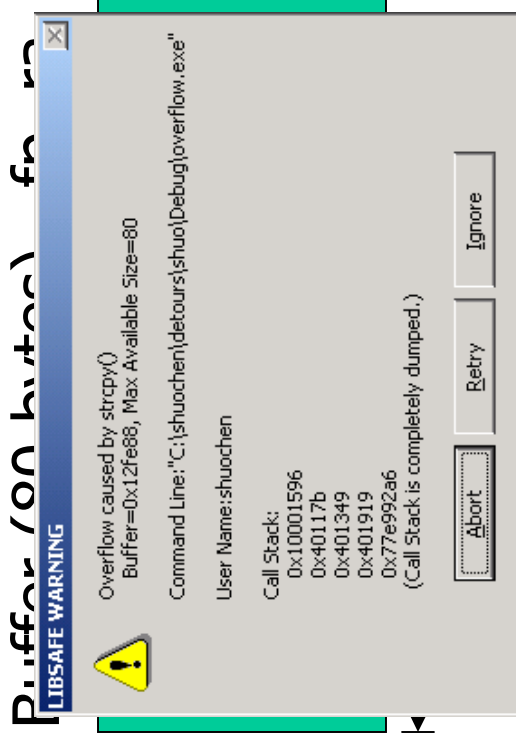
- Mechanism
  - Instrument the compiler to insert additional run-time instrumentation
  - Allow overflow to occur, but verify return address before returning from function to prevent execution of attack code
- Pros
  - Can be selectively applied to applications to minimize overhead
- Cons
  - Requires recompilation of applications (or installation of pre-instrumented applications)
  - Doesn't catch small overflows



# Library Solutions

- **Mechanism**
  - Use shared or static library to intercept calls to vulnerable functions
  - Perform bounds checking on function arguments to prevent overflow from occurring
- **Pros**
  - Easy to install
- **Cons**
  - Doesn't catch overflows via non-intercepted functions or code
  - Doesn't catch small overflows

# Example of Libsafe Protection



```
void foo(char * input_string)
{ char buffer[80];
  strcpy(buffer,input_string);
  return;
} /*len(input_string)=88 bytes*/

char * libsafeStrcpy(
  char *dest,
  const char * src)
{ if (src is longer than max_size)
  report the event;
  else
  return strcpy(dest,src);
}
```



# Recommendations

- **Turn off all unnecessary services (web, mail, print, SNMP)**
  - **Use a firewall**
  - **Use intrusion detection tools**
- **Get latest versions of software, especially services**
  - **Consider automatic update tools**
- **Run security audit tools**
- **Consider run-time instrumentation**