# Week 6

FIREWALLS AND INTRUSION PREVENTION SYSTEMS

BUFFER OVERFLOW

# Chapter 9

Firewalls and Intrusion Prevention Systems

## The Need for Firewalls

Internet connectivity is essential

- However it creates a threat

Effective means of protecting LANs

Inserted between the premises network and the Internet to establish a controlled link

- Can be a single computer system or a set of two or more systems working together

Used as a perimeter defense

- Single choke point to impose security and auditing
- Insulates the internal systems from external networks

## Firewall Characteristics

**Design goals**

All traffic from inside to outside, and vice versa, must pass through the firewall

Only authorized traffic as defined by the local security policy will be allowed to pass

The firewall itself is immune to penetration

## Firewall Access Policy

A critical component in the planning and implementation of a firewall is specifying a suitable access policy

- This lists the types of traffic authorized to pass through the firewall
- Includes address ranges, protocols, applications and content types

This policy should be developed from the organization’s information security risk assessment and policy

Should be developed from a broad specification of which traffic types the organization needs to support

- Then refined to detail the filter elements which can then be implemented within an appropriate firewall topology

# Firewall Filter Characteristics

Characteristics that a firewall access policy could use to filter traffic include

**IP address and protocol values**

- This type of filtering is used by packet filter and stateful inspection firewalls
- Typically used to limit access to specific services

**Application protocol**

- This type of filtering is used by an application-level gateway that relays and monitors the exchange of information for specific application protocols

**User identity**

- Typically for inside users who identify themselves using some form of secure authentication technology

**Network activity**

- Controls access based on considerations such as the time or request, rate of requests, or other activity patterns

## Try an External Firewall Scan

https://www.grc.com/x/ne.dll?bh0bkyd2

What are your results?

# Firewall Capabilities and Limits

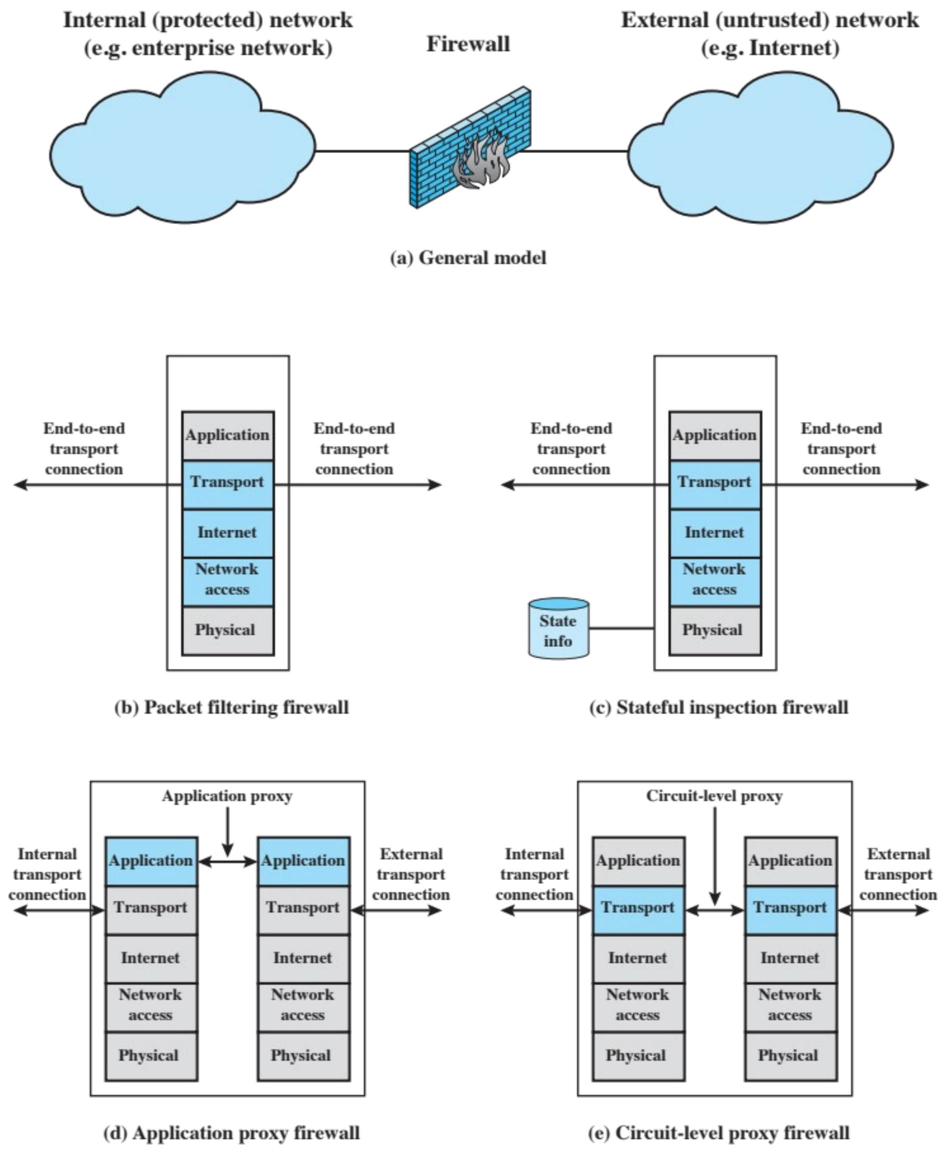
## Capabilities:

- Defines a single choke point
- Provides a location for monitoring security events
- Convenient platform for several Internet functions that are not security related
- Can serve as the platform for IPSec

## Limitations:

- Cannot protect against attacks bypassing firewall
- May not protect fully against internal threats
- Improperly secured wireless LAN can be accessed from outside the organization
- Laptop, PDA, or portable storage device may be infected outside the corporate network then used internally

# Types of Firewalls



Internal (protected) network (e.g. enterprise network)
Firewall
External (untrusted) network (e.g. Internet)
(a) General model
End-to-end transport connection
Application
Transport
Internet
Network access
Physical
End-to-end transport connection
(b) Packet filtering firewall
State info
(c) Stateful inspection firewall
Application proxy
Internal transport connection
External transport connection
(d) Application proxy firewall
Circuit-level proxy
(e) Circuit-level proxy firewall

Figure 9.1 Types of Firewalls

# Packet Filtering Firewall

Applies rules to each incoming and outgoing IP packet

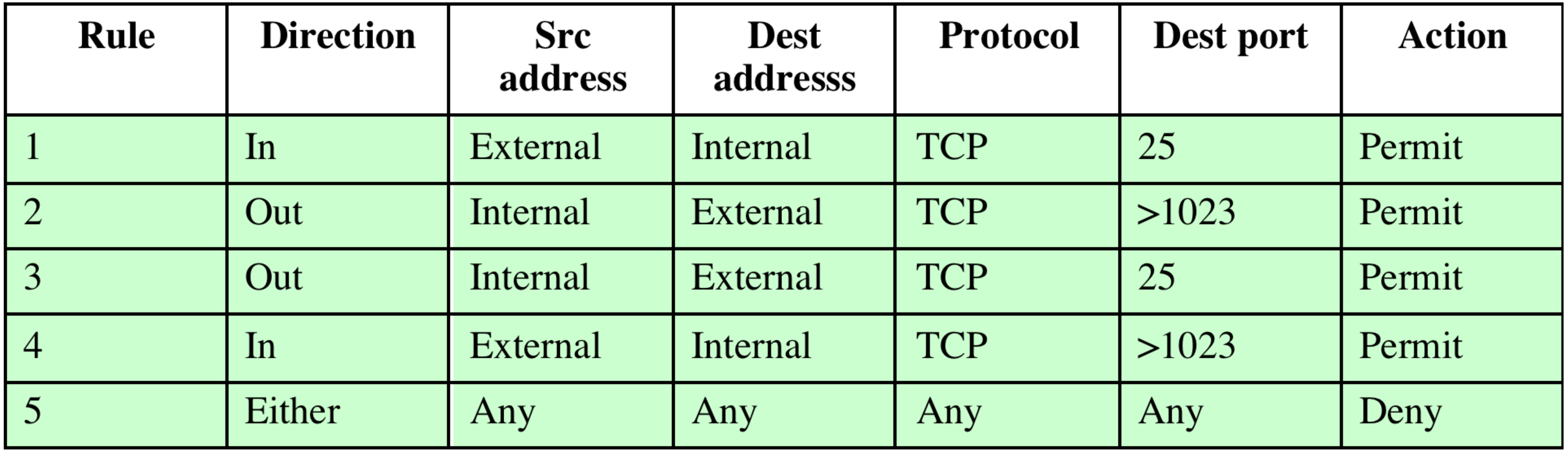
- Typically a list of rules based on matches in the IP or TCP header
- Forwards or discards the packet based on rules match

Two default policies:

- Discard - prohibit unless expressly permitted
  - More conservative, controlled, visible to users
- Forward - permit unless expressly prohibited
  - Easier to manage and use but less secure

Filtering rules are based on information contained in a network packet

- Source IP address
- Destination IP address
- Source and destination transport-level address
- IP protocol field
- Interface



| Rule | Direction | Src address | Dest addresss | Protocol | Dest port | Action |
|---|---|---|---|---|---|---|
| 1 | In | External | Internal | TCP | 25 | Permit |
| 2 | Out | Internal | External | TCP | >1023 | Permit |
| 3 | Out | Internal | External | TCP | 25 | Permit |
| 4 | In | External | Internal | TCP | >1023 | Permit |
| 5 | Either | Any | Any | Any | Any | Deny |

## Packet Filtering Example

## Packet Filtering Tradeoffs

Advantages

- Simplicity
- Typically transparent to users and are very fast

Weaknesses

- Cannot prevent attacks that employ application specific vulnerabilities or functions
- Limited logging functionality
- Do not support advanced user authentication
- Vulnerable to attacks on TCP/IP protocol bugs
- Improper configuration can lead to breaches

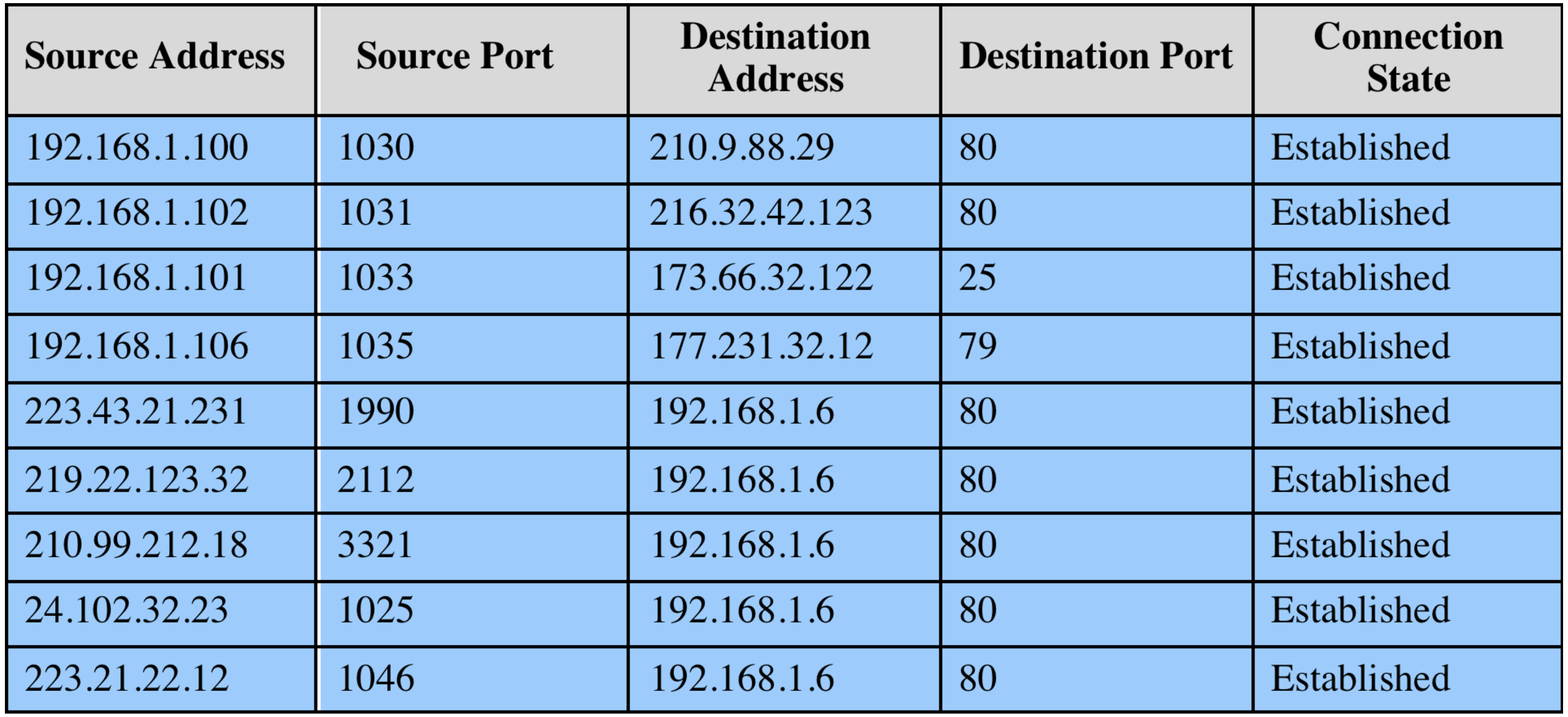
## Stateful Inspection Firewall

**Tightens rules for TCP traffic by creating a directory of outbound TCP connections**

- There is an entry for each currently established connection
- Packet filter allows incoming traffic to high numbered ports only for those packets that fit the profile of one of the entries in this directory

**Reviews packet information but also records information about TCP connections**

- Keeps track of TCP sequence numbers to prevent attacks that depend on the sequence number
- Inspects data for protocols like FTP, IM and SIPS commands



| Source Address | Source Port | Destination Address | Destination Port | Connection State |
|---|---|---|---|---|
| 192.168.1.100 | 1030 | 210.9.88.29 | 80 | Established |
| 192.168.1.102 | 1031 | 216.32.42.123 | 80 | Established |
| 192.168.1.101 | 1033 | 173.66.32.122 | 25 | Established |
| 192.168.1.106 | 1035 | 177.231.32.12 | 79 | Established |
| 223.43.21.231 | 1990 | 192.168.1.6 | 80 | Established |
| 219.22.123.32 | 2112 | 192.168.1.6 | 80 | Established |
| 210.99.212.18 | 3321 | 192.168.1.6 | 80 | Established |
| 24.102.32.23 | 1025 | 192.168.1.6 | 80 | Established |
| 223.21.22.12 | 1046 | 192.168.1.6 | 80 | Established |

# Stateful Connection Firewall Example

## Application Level Gateway

Also called an application proxy

Acts as a relay of application-level traffic
- User contacts gateway using a TCP/IP application
- User is authenticated
- Gateway contacts application on remote host and relays TCP segments between server and user

Must have proxy code for each application
- May restrict application features supported

Tend to be more secure than packet filters

Disadvantage is the additional processing overhead on each connection

## Circuit-level Gateway

### Circuit level proxy

- Sets up two TCP connections, one between itself and a TCP user on an inner host and one on an outside host
- Relays TCP segments from one connection to the other without examining contents
- Security function consists of determining which connections will be allowed

### Typically used when inside users are trusted

- May use application-level gateway inbound and circuit-level gateway outbound
- Lower overheads

## Host-based Firewalls

Used to secure an individual host

Available in operating systems or can be provided as an add-on package

Filter and restrict packet flows

Common location is a server

**Advantages**:

- Filtering rules can be tailored to the host environment
- Protection is provided independent of topology
- Provides an additional layer of protection

## Personal Firewall

Controls traffic between a personal computer or workstation and the Internet or enterprise network

For both home or corporate use

Typically is a software module on a personal computer

Can be housed in a router that connects all of the home computers to a DSL, cable modem, or other Internet interface

Typically much less complex than server-based or stand-alone firewalls

Primary role is to deny unauthorized remote access

May also monitor outgoing traffic to detect and block worms and malware activity

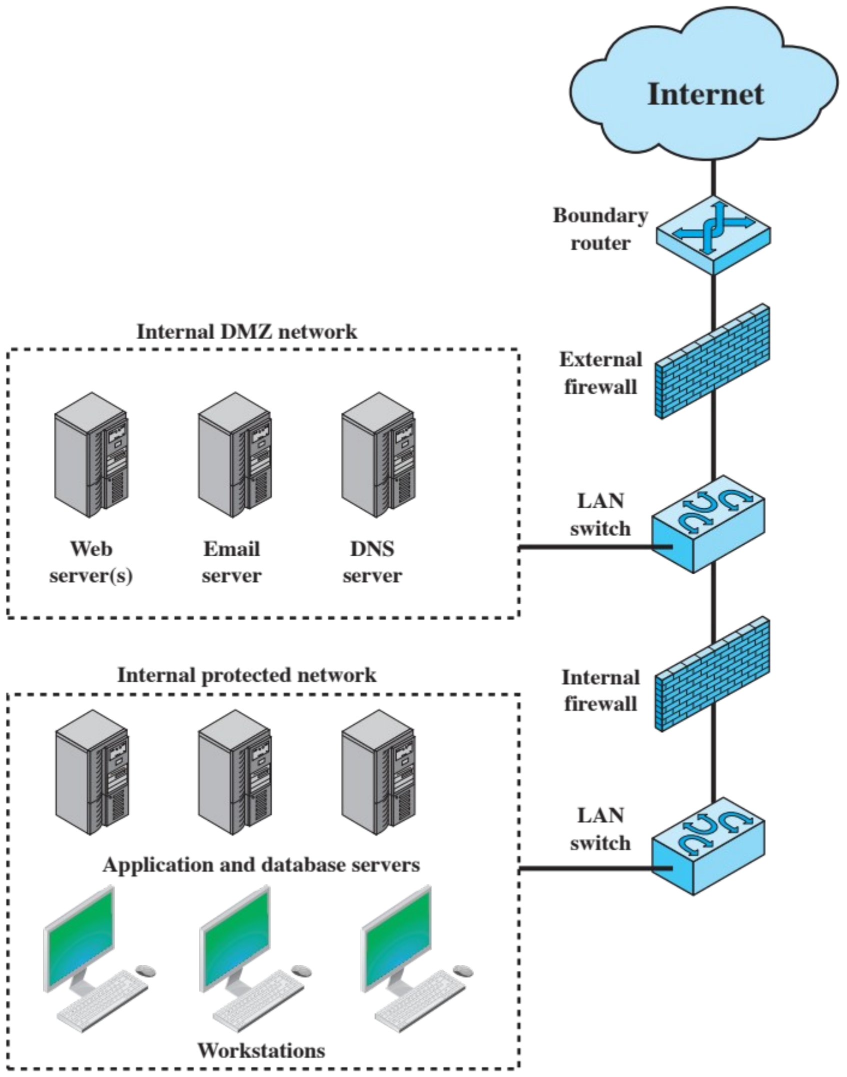


Figure 9.2 Example Firewall Configuration

## Firewall Configuration

# VPN Security Scenario

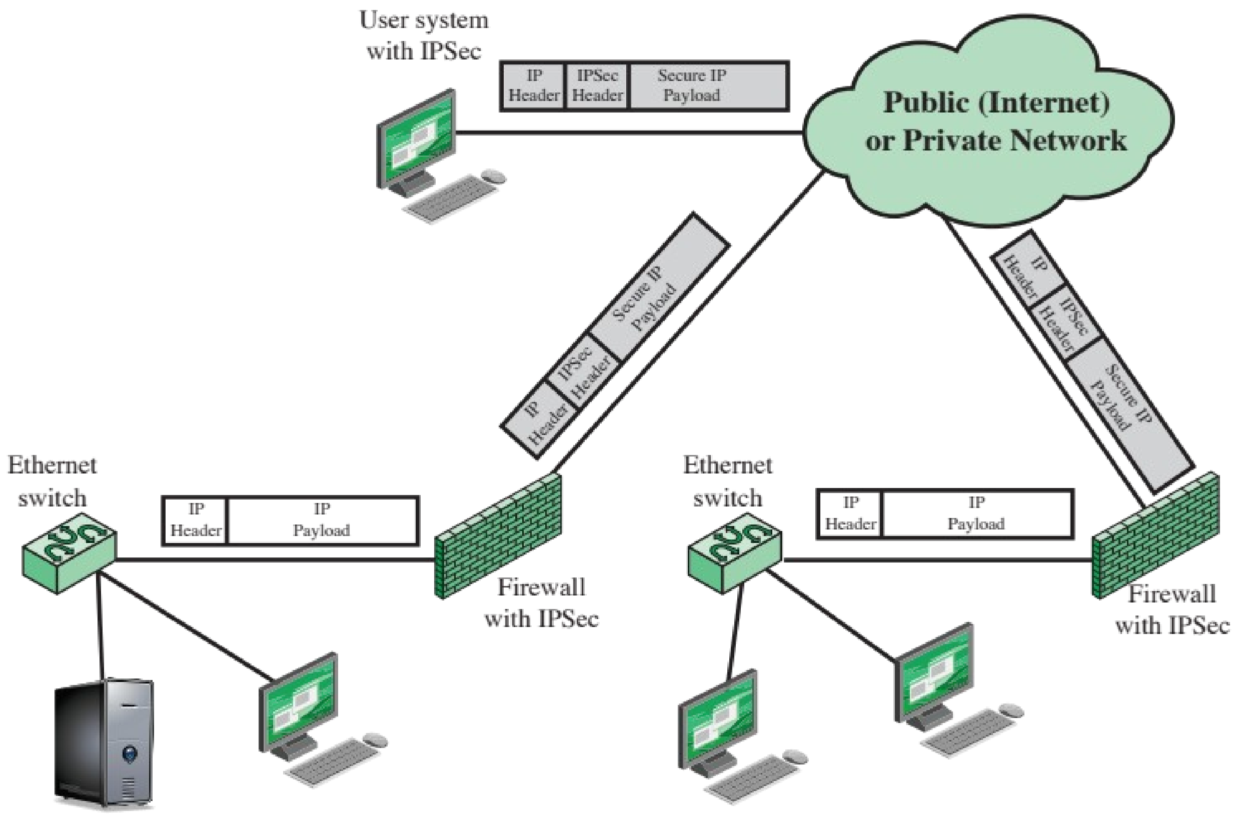


Figure 9.3 A VPN Security Scenario

# Distributed Firewall Configuration

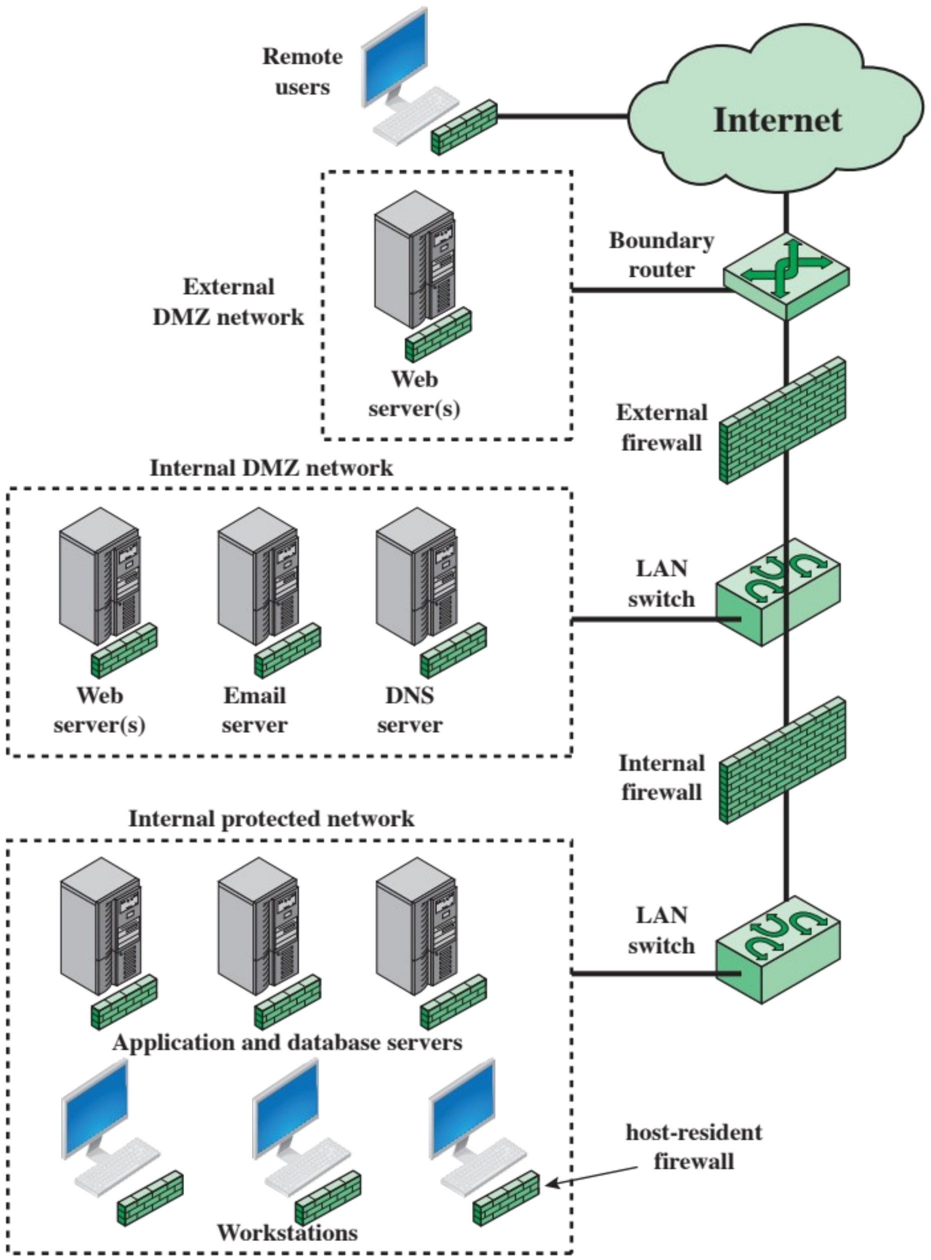


Figure 9.4 Example Distributed Firewall Configuration

## Intrusion Prevention System (IPS)

Also known as Intrusion Detection and Prevention System (IDPS)

Is an extension of an IDS that includes the capability to attempt to block or prevent detected malicious activity

Can be host-based, network-based, or distributed/hybrid

Can use anomaly detection to identify behavior that is not that of legitimate users, or signature/heuristic detection to identify known malicious behavior can block traffic as a firewall does, but makes use of the types of algorithms developed for IDSs to determine when to do so

## Host-based IPS (HIPS)

Can make use of either signature/heuristic or anomaly detection techniques to identify attacks

- Signature: focus is on the specific content of application network traffic, or of sequences of system calls, looking for patterns that have been identified as malicious
- Anomaly: IPS is looking for behavior patterns that indicate malware

Examples of the types of malicious behavior addressed by a HIPS include:

- Modification of system resources
- Privilege-escalation exploits
- Buffer-overflow exploits
- Access to e-mail contact list
- Directory traversal

## HIPS (cont.)

Capability can be tailored to the specific platform

A set of general purpose tools may be used for a desktop or server system

Some packages are designed to protect specific types of servers, such as Web servers and database servers

- In this case the HIPS looks for particular application attacks

Can use a sandbox approach

- Sandboxes are especially suited to mobile code such as Java applets and scripting languages
- HIPS quarantines such code in an isolated system area then runs the code and monitors its behavior

Areas for which a HIPS typically offers desktop protection:

- System calls
- File system access
- System registry settings
- Host input/output

## Role of HIPS

Many industry observers see the enterprise endpoint, including desktop and laptop systems, as now the main target for hackers and criminals

- Thus security vendors are focusing more on developing endpoint security products
- Traditionally, endpoint security has been provided by a collection of distinct products, such as antivirus, antispyware, antispam, and personal firewalls

Approach is an effort to provide an integrated, single-product suite of functions

- Advantages of the integrated HIPS approach are that the various tools work closely together, threat prevention is more comprehensive, and management is easier

A prudent approach is to use HIPS as one element in a defense-in-depth strategy that involves network-level devices, such as either firewalls or network-based IPSs

# Network-based IPS

Inline NIDS with the authority to modify or discard packets and tear down TCP connections

Makes use of signature/heuristic detection and anomaly detection

May provide flow data protection

- Requires that the application payload in a sequence of packets be reassembled

Methods used to identify malicious packets:

- Pattern matching
- Stateful matching
- Protocol anomaly
- Traffic anomaly
- Statistical anomaly

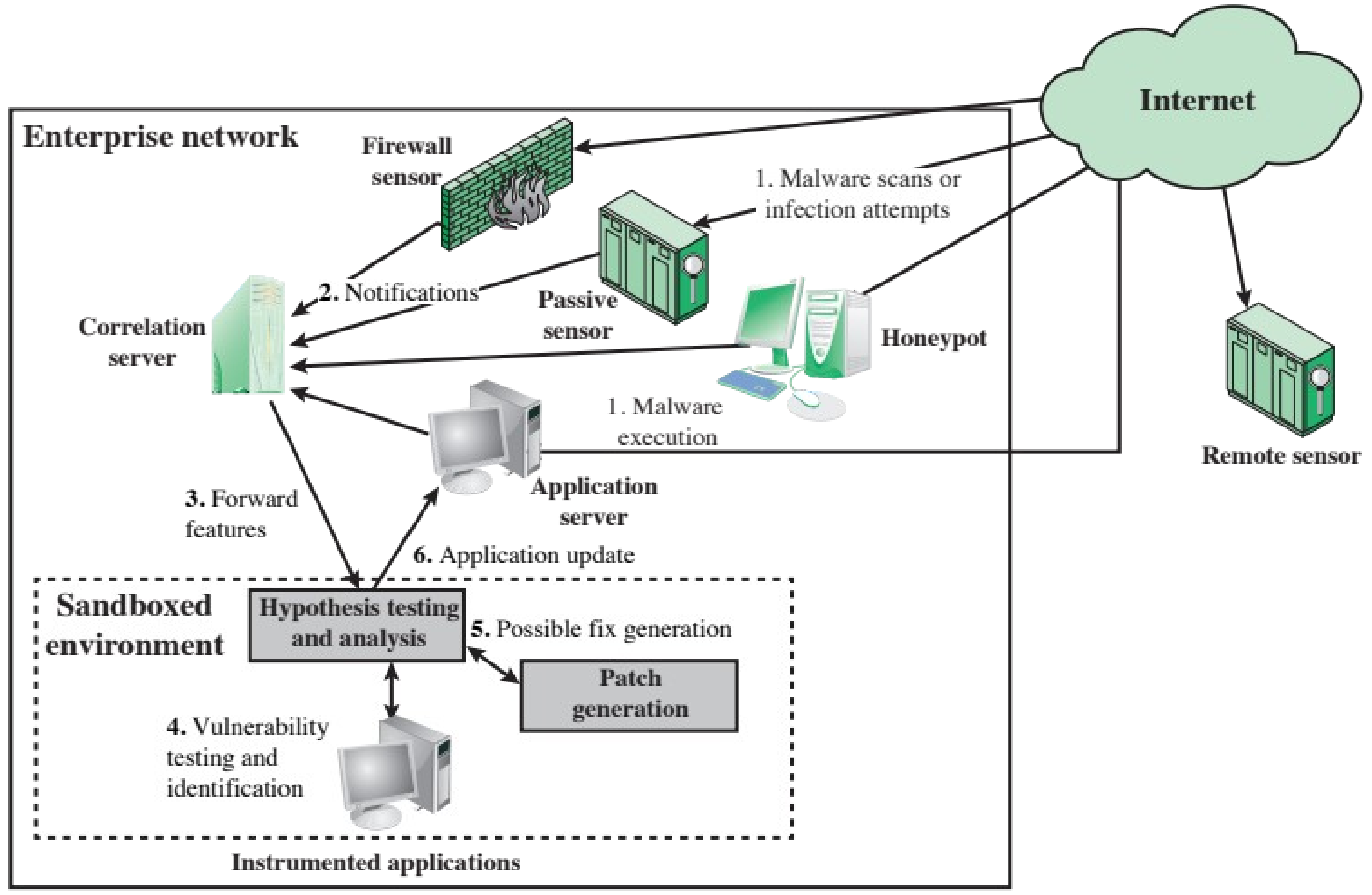


**Figure 9.5 Placement of Worm Monitors**

# Placement of Malware Monitors

# Threat Management Appliance

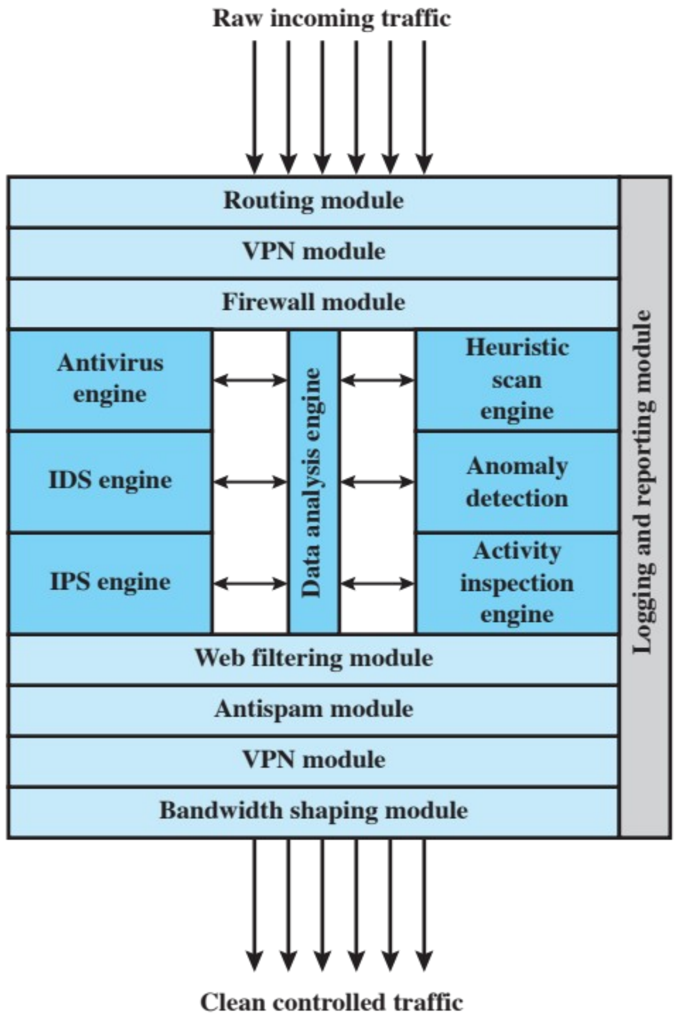
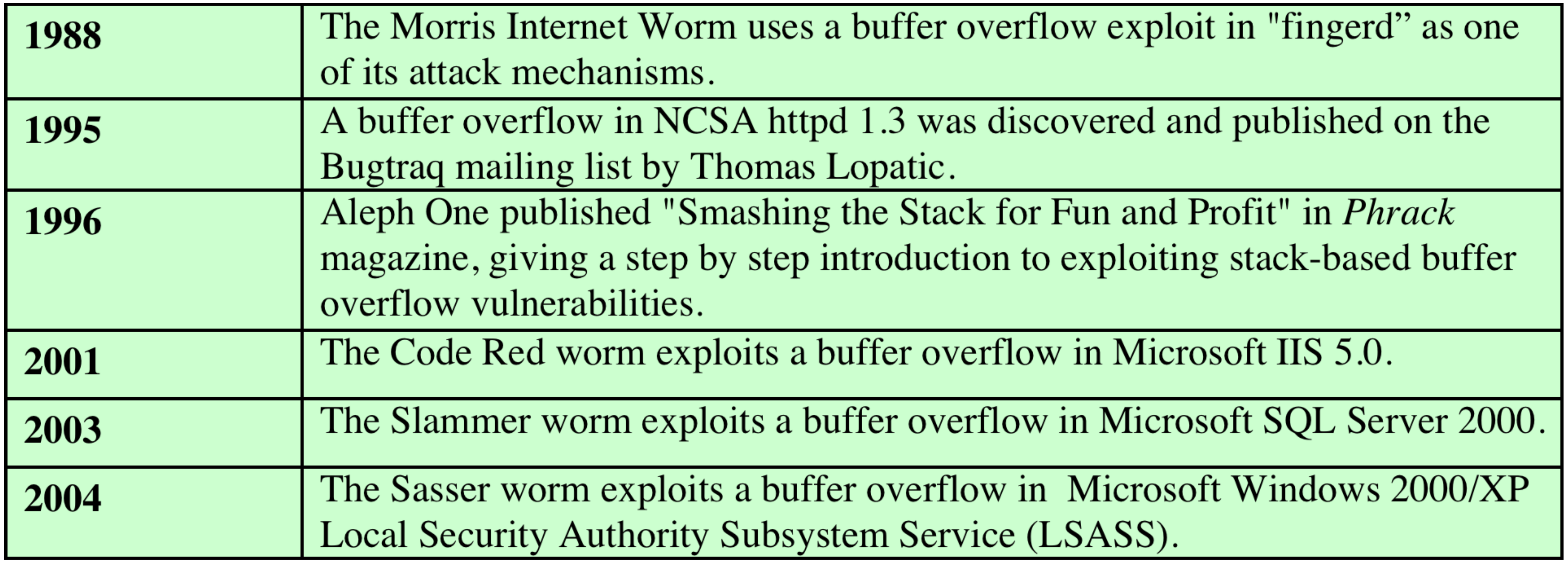


**Figure 9.6** **Unified Threat Management Appliance (based on [JAME06])**

# Chapter 10

# Buffer Overflow

## Brief History of Famous Buffer Overflow Attacks



| | |
|---|---|
| **1988** | The Morris Internet Worm uses a buffer overflow exploit in "fingerd" as one of its attack mechanisms. |
| **1995** | A buffer overflow in NCSA httpd 1.3 was discovered and published on the Bugtraq mailing list by Thomas Lopatic. |
| **1996** | Aleph One published "Smashing the Stack for Fun and Profit" in *Phrack* magazine, giving a step by step introduction to exploiting stack-based buffer overflow vulnerabilities. |
| **2001** | The Code Red worm exploits a buffer overflow in Microsoft IIS 5.0. |
| **2003** | The Slammer worm exploits a buffer overflow in Microsoft SQL Server 2000. |
| **2004** | The Sasser worm exploits a buffer overflow in Microsoft Windows 2000/XP Local Security Authority Subsystem Service (LSASS). |

# Buffer Overflow

A very common attack mechanism

- First widely used by the Morris Worm in 1988

Prevention techniques known

Still of major concern

- Legacy of buggy code in widely deployed operating systems and applications
- Continued careless programming practices by programmers

A buffer overflow, also known as a buffer overrun, is defined in the NIST *Glossary of Key Information Security Terms* as follows:

“A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system.”

## Buffer Overflow Basics

A buffer overflow, also known as a buffer overrun, is defined in the NIST *Glossary of Key Information Security Terms* as follows:

“A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system.”

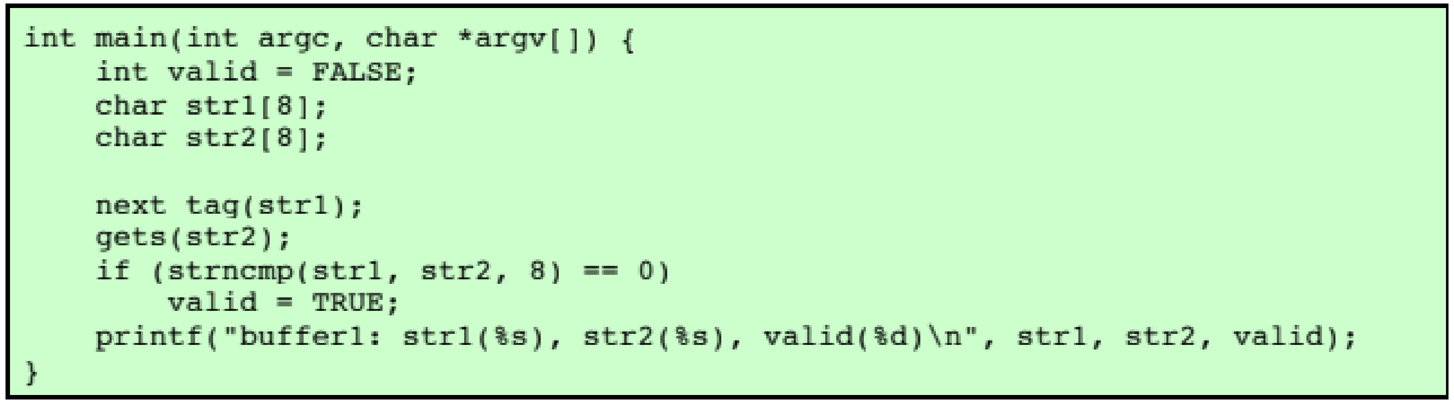
**Consequences:**

- **Corruption of program data**
- **Unexpected transfer of control**
- **Memory access violations**
- **Execution of code chosen by attacker**

## Buffer Overflow Basics

https://www.youtube.com/watch?v=SOoJcrR4Ijo

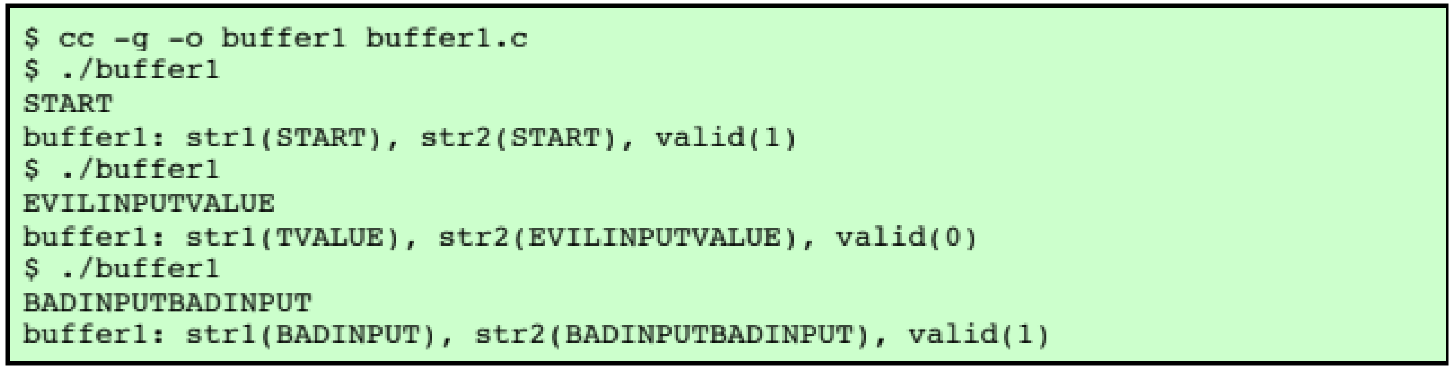
# Basic Code Example



```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

**(a) Basic buffer overflow C code**



```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

**(b) Basic buffer overflow example runs**

**Figure 10.1 Basic Buffer Overflow Example**

## Buffer Overflow Attacks

To exploit a buffer overflow an attacker needs:

- To identify a buffer overflow vulnerability in some program that can be triggered using externally sourced data under the attacker's control
- To understand how that buffer is stored in memory and determine potential for corruption

Identifying vulnerable programs can be done by:

- Inspection of program source
- Tracing the execution of programs as they process oversized input
- Using tools such as fuzzing to automatically identify potentially vulnerable programs

# Programming Language History

At the machine level data manipulated by machine instructions executed by the computer processor are stored in either the processor's registers or in memory

Assembly language programmer is responsible for the correct interpretation of any saved data value

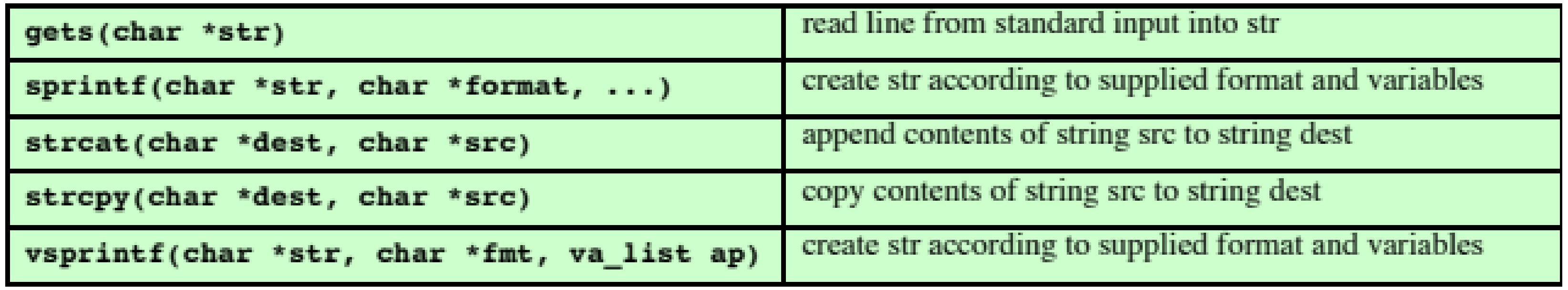
**Modern high-level languages have a strong notion of type and valid operations**

- **Not vulnerable to buffer overflows**
- **Does incur overhead, some limits on use**

**C and related languages have high-level control structures, but allow direct access to memory**

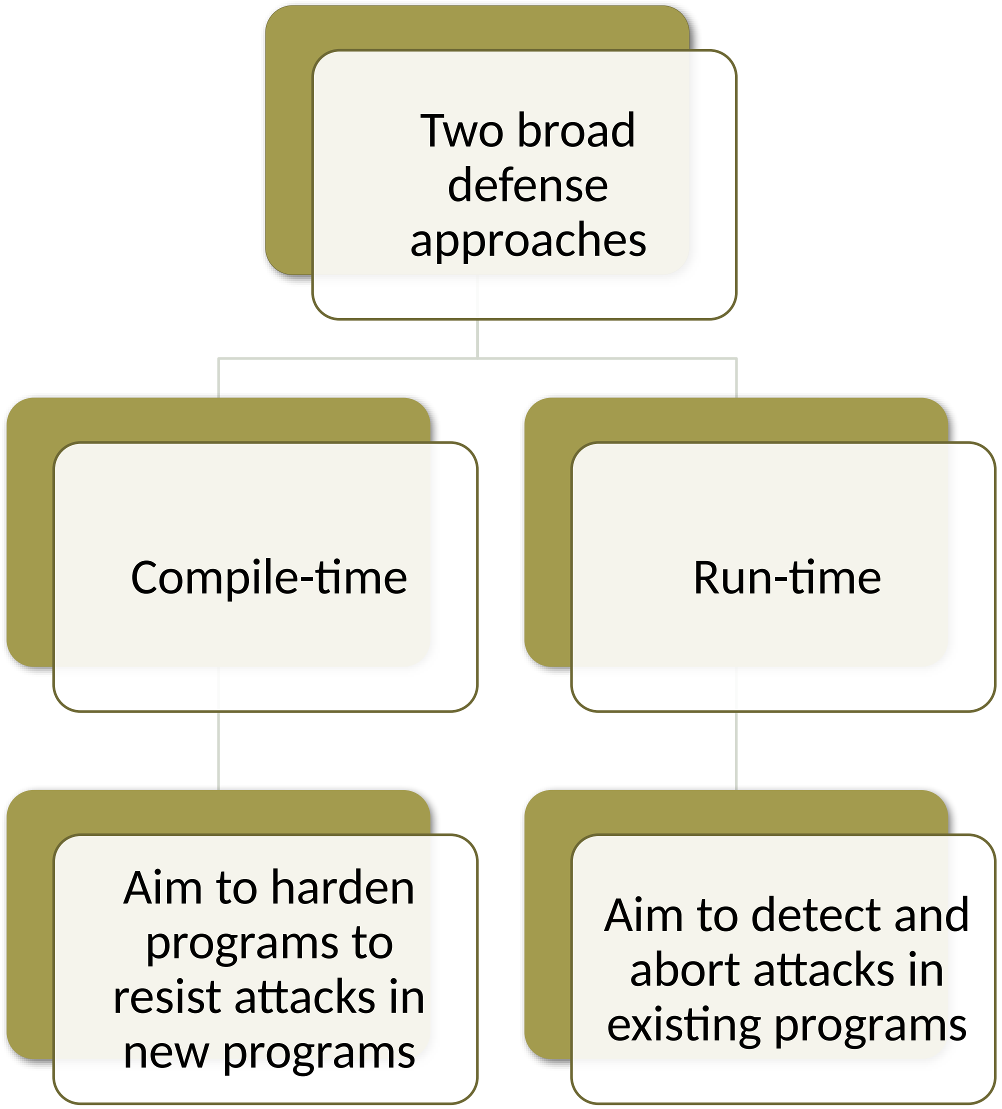
- **Hence are vulnerable to buffer overflow**
- **Have a large legacy of widely used, unsafe, and hence vulnerable code**

## Example of Common Unsafe C STD Lib Routines



| | |
|---|---|
| `gets(char *str)` | read line from standard input into str |
| `sprintf(char *str, char *format, ...)` | create str according to supplied format and variables |
| `strcat(char *dest, char *src)` | append contents of string src to string dest |
| `strcpy(char *dest, char *src)` | copy contents of string src to string dest |
| `vsprintf(char *str, char *fmt, va_list ap)` | create str according to supplied format and variables |

# Buffer Overflow Defenses


Two broad defense approaches
Compile-time
Run-time
Aim to harden programs to resist attacks in new programs
Aim to detect and abort attacks in existing programs

## Compile-Time Defenses: Programming Language

Use a modern high-level language

- Not vulnerable to buffer overflow attacks
- Compiler enforces range checks and permissible operations on variables

**Disadvantages**

- Additional code must be executed at run time to impose checks
- Flexibility and safety comes at a cost in resource use
- Distance from the underlying machine language and architecture means that access to some instructions and hardware resources is lost
- Limits their usefulness in writing code, such as device drivers, that must interact with such resources

## Compile-Time Defenses: Language Extensions and Safe Libraries

Handling dynamically allocated memory is more problematic because the size information is not available at compile time

- Requires an extension and the use of library routines
  - Programs and libraries need to be recompiled
  - Likely to have problems with third-party applications
  - Concern with C is use of unsafe standard library routines
    - One approach has been to replace these with safer variants
      - Libsafe is an example
      - Library is implemented as a dynamic library arranged to load before the existing standard libraries

## Runtime Defenses: Executable Address Space Protection

**Use virtual memory support to make some regions of memory non-executable**

- **Requires support from memory management unit (MMU)**
- **Long existed on SPARC / Solaris systems**
- **Recent on x86 Linux/Unix/Windows systems**

**Issues**

- **Support for executable stack code**
- **Special provisions are needed**

## Runtime Defenses: Address Space Randomization

Manipulate location of key data structures

- Stack, heap, global data
- Using random shift for each process
- Large address range on modern systems means wasting some has negligible impact

Randomize location of heap buffers

Random location of standard library functions

# Heap Overflow

Attack buffer located in heap

- Typically located above program code
- Memory is requested by programs to use in dynamic data structures (such as linked lists of records)

No return address

- Hence no easy transfer of control
- May have function pointers can exploit
- Or manipulate management data structures

**Defenses**

- Making the heap non-executable
- Randomizing the allocation of memory on the heap

# Tools

## Parasoft Insure++

- Corrupted stack/heap
- Memory leaks
- Dangling pointers
- Out of bounds / overflow errors

https://www.parasoft.com/products/parasoft-insure/c-c-memory-debugging/