**Module 12 – Operating Systems Security**

**Learning objectives:**

1) Describe the basic operating systems concepts such as users, permission, file system security

2) Explain buffer overflow vulnerabilities, exploitation and prevention

3) Describe intrusion detection system and differentiate between signature and anomaly-based IDS

4) Explain firewall rule tables and packet filtering

Contents

[**1. System security** 2](#_Toc98230878)

[**a) Root access** 2](#_Toc98230879)

[**b) User groups** 2](#_Toc98230880)

[**c) Permissions** 2](#_Toc98230881)

[**d) Passwords** 2](#_Toc98230882)

[**e) Patching** 3](#_Toc98230883)

[**f) File system security** 4](#_Toc98230884)

[**2. Buffer overflow attacks and defense mechanisms** 6](#_Toc98230885)

[**3. Intrusion detection systems** 9](#_Toc98230886)

[**4. Firewall** 11](#_Toc98230887)

[**References** 13](#_Toc98230888)

# **1. System security**

A secure operating system environment is achieved not only by the design concepts of these operating systems, but also through vigilant user and administrative practices. Some of the inherent features present in widely used operating systems (OS) are shown below using Linux/Unix as example.

## **a) Root access**

Most Unix and Unix-like systems have an account or group which enables a user to exact complete control over the system, often known as a [root](https://en.wikipedia.org/wiki/Superuser) account. If access to this account is gained by an unwanted user, this results in a complete breach of the system. A root account however is necessary for administrative purposes, and for the above security reasons the root account is *seldom* used for day to day purposes (the [sudo](https://en.wikipedia.org/wiki/Sudo" \o "Sudo) program is more commonly used), so usage of the root account can be more closely monitored.

## **b) User groups**

Users under Unix style operating systems often belong to managed groups with specific access permissions. This enables users to be grouped by the level of access they have to this system. Many Unix implementations add an additional layer of security by requiring that a user be a member of the [*wheel*](https://en.wikipedia.org/wiki/Wheel_(computing)) [user privileges group](https://en.wikipedia.org/wiki/Group_identifier_(Unix)) in order to access the su command.

## **c) Permissions**

A core security feature in OS is the file system permission. All files in a typical Unix files system have permissions set enabling different access to a file.

Permissions on a file are commonly set using the [chmod](https://en.wikipedia.org/wiki/Chmod" \o "Chmod) command and seen through the [ls](https://en.wikipedia.org/wiki/Ls) command. For example:

-r-xr-xr-x 1 root wheel 745720 Sep 8 2002 /bin/sh

[Unix permissions](https://en.wikipedia.org/wiki/Unix_permissions) permit different users access to a file. Different *user groups* have different permissions on a file.

More advanced Unix filesystems include the *Access Control List* concept which allows permissions to be granted to multiple users or groups. An *Access Control List* may be used to grant permission to additional individual users or groups. For example:

/pvr [u::rwx,g::r-x,o::r-x/u::rwx,u:sue:rwx,g::r-x,m::rwx,o::r-x]

In this example, which is from the [chacl](https://en.wikipedia.org/w/index.php?title=Chacl&action=edit&redlink=1" \o "Chacl (page does not exist)) command on the [Linux](https://en.wikipedia.org/wiki/Linux) operating system, the user **sue** is granted *write* permission to the */pvr* directory.

## **d) Passwords**

Selecting a strong [password](https://en.wikipedia.org/wiki/Password) and guarding it properly are probably the most important things a user can do to improve Unix security. In Unix systems, the essential information about users is stored under the file [/etc/passwd](https://en.wikipedia.org/wiki/Passwd_(file))This file keeps track of the users registered in the system and their main definitions. Passwords, or more correctly, the hash of the password, can also be stored in the same place. The entries in /etc/passwd occupy exactly one line each, and have the following form:

nickname:password\_hash:UserID:GroupID:Complete\_Name:home\_dir:shell\_bin

An example would be:

xfze:$$1zuW2nX3sslp3qJm9MYDdglEApAc36r/:1000:100:Daniel Ernesto Ortiz Costa:/home/xfze:/bin/bash

Since all users must have read access to the /etc/passwd file to do many common tasks (ls -l /home will use /etc/passwd to map UIDs to login names for example), anyone could also read the password hashes of other users. To solve this problem, the file [/etc/shadow](https://en.wikipedia.org/wiki/Password_shadowing) was created to store the password hashes, with only [root](https://en.wikipedia.org/wiki/Root) having read access. Under password shadowing, the 2nd field ([password hash](https://en.wikipedia.org/wiki/Key_derivation_function)) is replaced by an 'x' which tells the system to retrieve the corresponding user's password via the /etc/shadow file.

The /etc/shadow file often only contains values for the first two fields:

xfze:$$1zuW2nX3sslp3qJm9MYDdglEApAc36r/:::::

The remaining fields in the /etc/shadow file include:

1. The minimum number of days between password changes
2. The maximum number of days until the password must be changed
3. The number of days of warning given before the password must be changed
4. The number of days after the password must be changed when the account becomes unusable
5. The date (expressed as the number of days since [January 1st, 1970](https://en.wikipedia.org/wiki/Unix_epoch)) when the account is expired

These fields may be used to improve Unix security by enforcing a password security policy.

## **e) Patching**

Operating systems, like all software, may contain bugs in need of fixing or may be enhanced with the addition of new features. Patching the operating system in a secure manner requires that the software come from a trustworthy source and not have been altered since it was packaged. Common methods for verifying that operating system patches have not been altered include the use of [cryptographic hash](https://en.wikipedia.org/wiki/Cryptographic_hash), such as an [MD5](https://en.wikipedia.org/wiki/MD5) based checksum, or the use of read-only media.

From a security standpoint, the specific packaging method, such as the [RPM Package Manager](https://en.wikipedia.org/wiki/RPM_Package_Manager) format originally from [Red Hat Linux](https://en.wikipedia.org/wiki/Red_Hat_Linux) is not as important as the use of features which ensure the integrity of the patch itself.

Unnecessary system software should not be installed or configured on a system. Software which is no longer required should be removed completely, if possible.

* Identify what services are running
  + netstat -na
  + lsof
  + nmap
  + sockstat -4 ([FreeBSD](https://en.wikipedia.org/wiki/FreeBSD))

The commands [inetd](https://en.wikipedia.org/wiki/Inetd" \o "Inetd) and [xinetd](https://en.wikipedia.org/wiki/Xinetd" \o "Xinetd) act as super-servers for a variety of network protocols such as [rlogin](https://en.wikipedia.org/wiki/Rlogin), [telnet](https://en.wikipedia.org/wiki/Telnet) and [ftp](https://en.wikipedia.org/wiki/Ftp).

Turning off unnecessary services

* using *update-rc.d* on [Debian](https://en.wikipedia.org/wiki/Debian)
* using *chkconfig* on [Red Hat Linux](https://en.wikipedia.org/wiki/Red_Hat_Linux)
* using */etc/rc.conf* and */usr/local/etc/rc.d* on FreeBSD (mention /etc/rc.local)
* using *rc-update* on [Gentoo Linux](https://en.wikipedia.org/wiki/Gentoo_Linux)

This approach is usually called *proactive security*. There are some operating systems which are *secure by default*. Amongst others, the free BSD flavours ([FreeBSD](https://en.wikipedia.org/wiki/FreeBSD), [NetBSD](https://en.wikipedia.org/wiki/NetBSD), and [OpenBSD](https://en.wikipedia.org/wiki/OpenBSD)) are proactively secure. For example, the output of netstat on a NetBSD 3.0 workstation clearly outlines this technique:

**$** netstat -a

Active Internet connections (including servers)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 localhost.smtp \*.\* LISTEN

tcp 0 0 \*.ssh \*.\* LISTEN

Active Internet6 connections (including servers)

Proto Recv-Q Send-Q Local Address Foreign Address (state)

tcp6 0 0 localhost.smtp \*.\* LISTEN

tcp6 0 0 \*.ssh \*.\* LISTEN

Active UNIX domain sockets

Address Type Recv-Q Send-Q Inode Conn Refs Nextref Addr

c0d10d80 dgram 0 0 0 c0cd8680 0 c0cb7000 -> /var/run/log

c0cb7000 dgram 0 0 0 c0cd8680 0 0 -> /var/run/log

c0cd8680 dgram 0 0 cb9639e8 0 c0d10d80 0 /var/run/log

## **f) File system security**

File system security within [UNIX](https://en.wikipedia.org/wiki/UNIX) and [Unix-like](https://en.wikipedia.org/wiki/Unix-like) systems is based on 9 permission bits, set user and group ID bits, and the [sticky bit](https://en.wikipedia.org/wiki/Sticky_bit), for a total of 12 bits. These permissions apply almost equally to all filesystem objects such as files, directories and devices.

The 9 permission bits are divided into three groups of three bits each. The first group describes the permissions of the file owner, the second group describes the permissions of the [GID](https://en.wikipedia.org/wiki/Group_identifier) assigned to the file, which by default is the group associated with the file owner or the directory containing the file when it is [*set-GID*](https://en.wikipedia.org/wiki/Setgid), and the third group describes the permissions associated with any process which does not have the same user ID as the file. Each group of three bits contains a bit indicating the read, write or execute access is granted. In the case of directories, execute access is interpreted as the permission to perform a filename lookup within the directory.

We can use **ls -l** command to display various information related to file permission as follows −

$ls -l /home/amrood

-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10 myfile

drwxr-xr--- 1 amrood users 1024 Nov 2 00:10 mydir

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

* The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
* The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.
* The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

**Read:** Grants the capability to read, i.e., view the contents of the file.

**Write:** Grants the capability to modify, or remove the content of the file.

**Execute:** User with execute permissions can run a file as a program.

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

**Read**: Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

**Write**: Access means that the user can add or delete files from the directory.

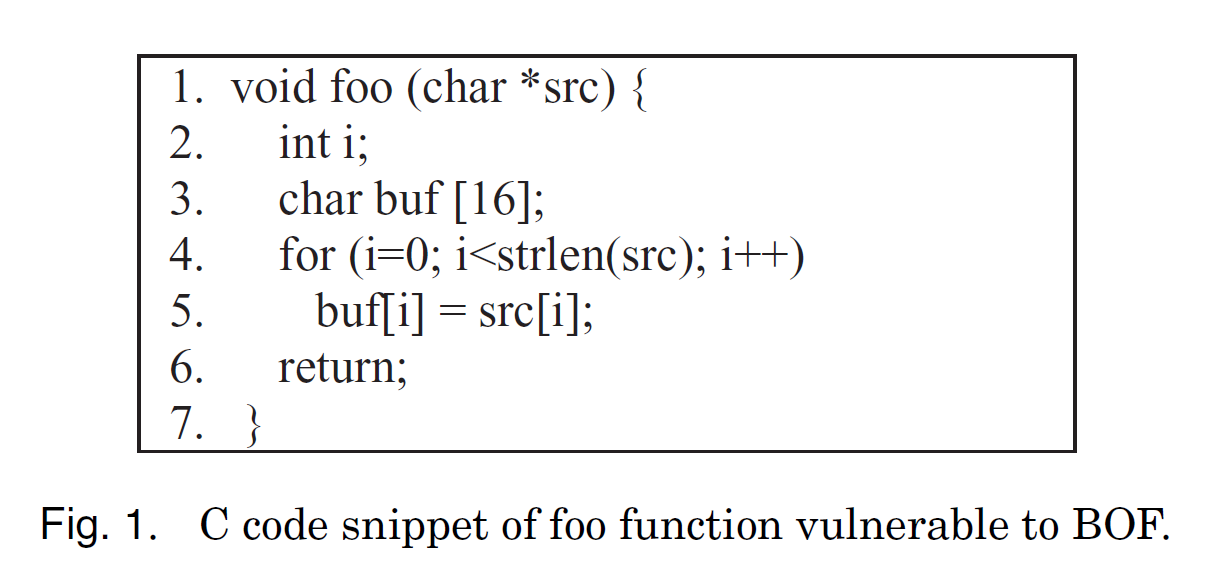
**Execute**: Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

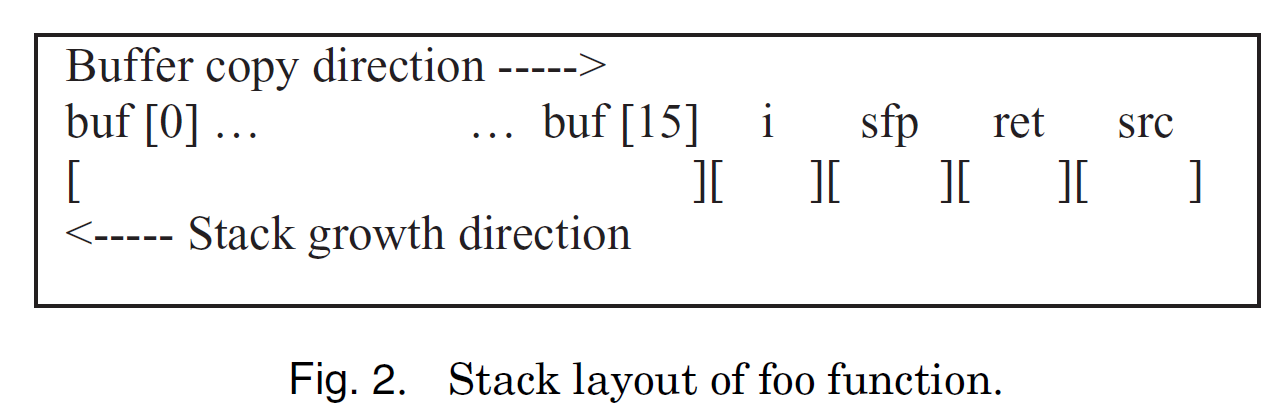
To change the file or the directory permissions, we use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

# **2. Buffer overflow attacks and defense mechanisms**

Operating systems and their services can be vulnerable to buffer overflow (BOF) attacks, which allow reading or writing data to program buffers exceeding allocated sizes and overwriting contents of memory locations. BOF might be present in programs having unsafe library function calls (e.g., ANSI C), lack of null characters at the end of buffers, buffer accesses through pointers and aliases, logic errors (off by one), and insufficient checks before accessing buffers.

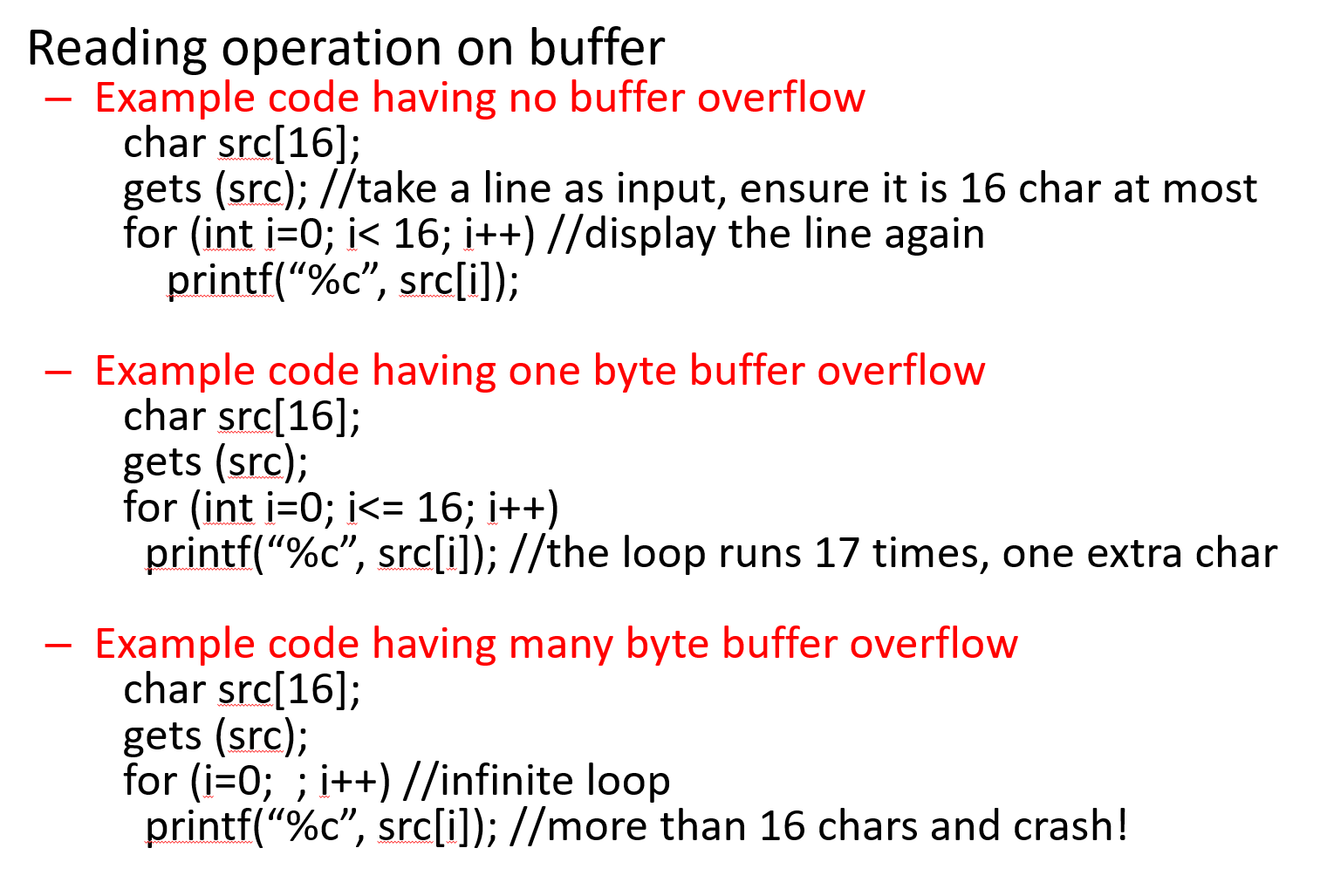


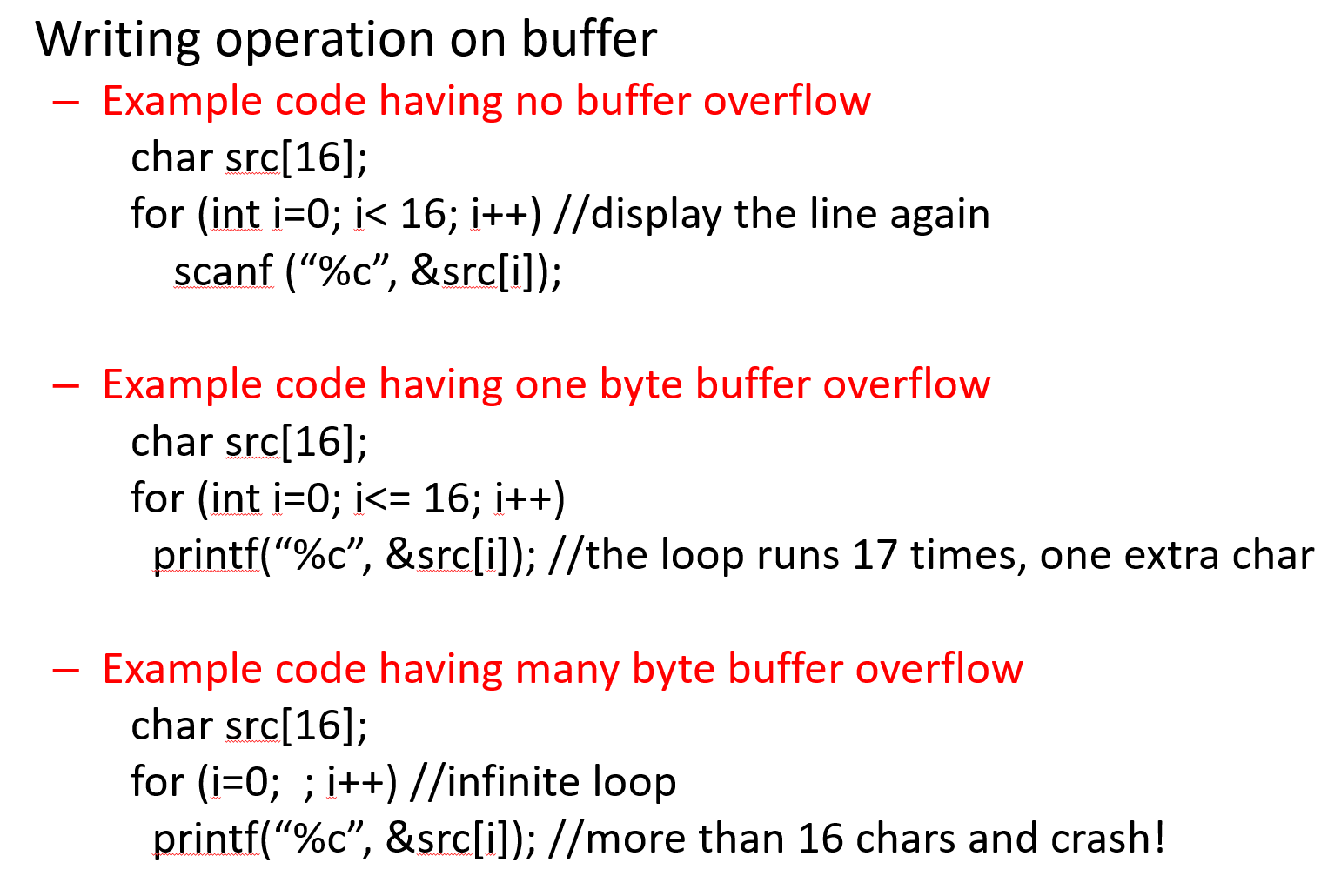
We provide an example of C code snippet (the function *foo*) in Figure 1 which is vulnerable to stack-based BOF. The buffer declared at line 3 (*buf* ) is located in the stack region and has 16 bytes of memory for reading and writing operations. The valid location of this buffer is between *buf[0]* and *buf[15]*. Lines 4 and 5 copy the *src* buffer into *buf* using a for-loop. However, the code is vulnerable as there is no checking on destination buffer length. As a result, the loop allows copying more than the capacity of *buf*. To understand the effect of an overflow, we consider a snapshot of the stack frame of the function *foo* shown in Figure 2.



The stack stores the argument of *foo* (*src*), the return address (*ret*), the saved frame pointer (*sfp*), and declared variables (*i* and *buf* ), followed by the loop counter (i). Note that the direction of stack growth and buffer copy is opposite to each other. We assume that both *ret* and *sfp* occupy four bytes, whereas *i* occupies two bytes. Storing an input of 17 bytes length (pointed by *src*) in the *buf* results in one byte overflow. This corrupts the neighboring variable *i* and leads to further unexpected behaviors by the program. However, an attacker usually corrupts the return address to execute injected code through buffer. This is known as a “return address clobbering” or “direct code injection” attack. An attacker injects a payload which can modify the content of *ret* to point to the beginning address of the buffer (i.e., the address of *buf[0]*). In this case, the *src* buffer should be 26 bytes (16 bytes for the *buf*, two bytes for *i*, four bytes for both *sfp* and *ret*) to overwrite *ret*. The injected code can contain a shell code (e.g., “/bin/sh”) to launch a remote shell with the root privilege. Note that several conditions need to be met to perform a successful attack such as allowing code execution from a stack segment, presence of no null character in the injected payload, and allowing an input of sufficient length to reach the location of *ret*

**BOF vulnerability Pattern examples while reading and write:**





**Unsafe library function calls:** BOF occurs due to no bound checking or loose bound checking on source and destination buffer, some examples are below.

* strcpy (dest, src) //copy src buffer into dest buffer
* strncpy (dest, src, n); //copy at most n bytes from src to dest buffer
* memcpy (dest, src, n) //copy at most n bytes from src to dest

Though srtcpy is highly vulnerable, strncpy is less vulnerable to BOF. Here, value of n is crucial, in the worst case n can be arbitrary large number

* + char dest[10], src [10];
  + strncpy (dest, src, 20); //n is too large than the capacity of dest buffer

One can look at the exhaustive list of function calls from <http://www.acm.uiuc.edu/webmonkeys/book/c_guide/>

Particularly, give attention to all library functions from string.h and stdlib.h

**BOF Prevention with Secure coding practice**: Developers are encouraged to use low risk function all, and avoid high risk function calls related to buffer operations. Below we show color coded ANSI library calls for high, moderate and low risk calls.



**BOF prevention with Runtime checking: Examples include Stackguard and Pointguard**

Stack canaries work by modifying every function's prologue and epilogue regions to place and check a value on the stack respectively. If a stack buffer is overwritten during a memory copy operation, the error is noticed before execution returns from the copy function. An exception is raised, which is passed back up the exception handler hierarchy until it finally hits the OS's default exception handler.

Pointguard protects function pointers with canaries. IBM developed Propolice by re-ordering stack element to reduce potential overflow.

StackGuard was introduced as a set of patches to the Intel x86 backend of GCC 2.7. IBM developed GCC patches for stack-smashing protection, known as ProPolice. RedHat engineers re-implemented ProPolice stack-smashing protection for inclusion in GCC 4.1 and introduced the -fstack-protector and the -fstack-protector-all flag

Address space layout randomization is another existing approach used in Microsoft Operating system. When a program runs, the location of buffer and other sensitive variables are randomly located in the memory space, so attackers are not able to run arbitrary code with buffer overflow.

For Linux-based system, non-executable memory is being used to prevent arbitrary code injection from buffer overflow attack.

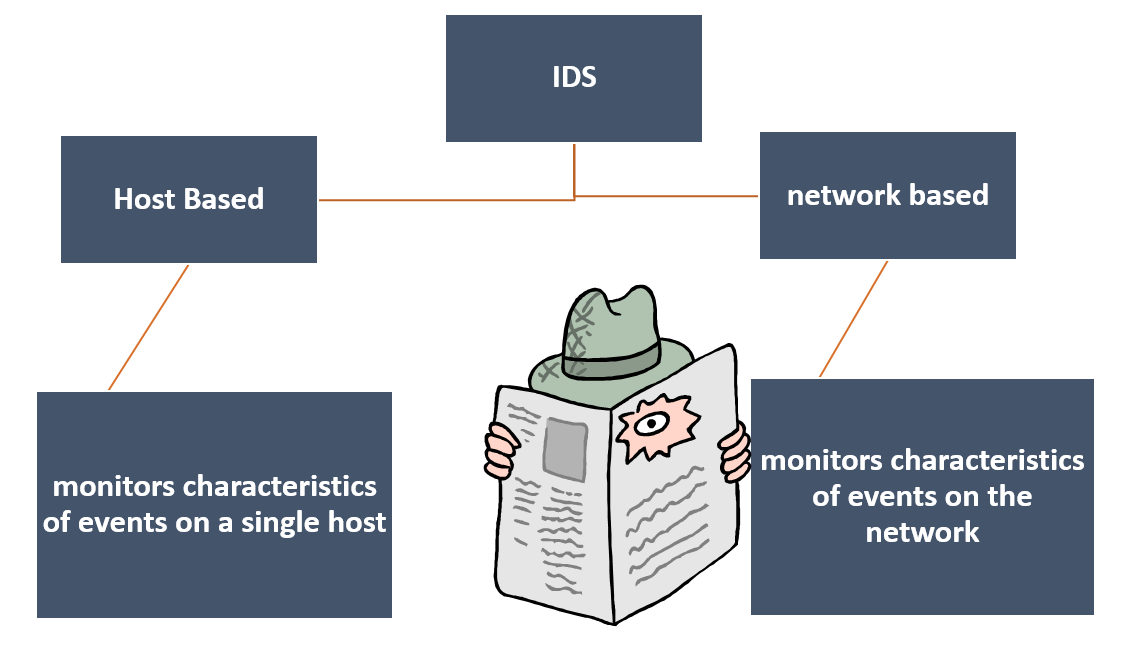
Code analysis or scanner tool can be used to detect buffer overflow in the source code. Two examples below:

* **Flawfinder:** It can check bof presence, <http://www.dwheeler.com/flawfinder>
* **Splint:** It can check bof automatically, <http://www.splint.org/manual/manual.html>

# **3. Intrusion detection systems**

Intrusion Detection is a special software or service that monitors and analyzes system events for the purpose of finding, and providing real-time or near-real- time warning of, attempts to access system resources in an unauthorized manner. Intrusion detection systems (IDSs) can be classified as follows:

* **Host-based IDS:** Monitors the characteristics of a single host and the events occurring within that host for suspicious activity.
* **Network-based IDS:** Monitors network traffic for particular network segments or devices and analyzes network, transport, and application protocols to identify suspicious activity.



An IDS comprises 2 major logical components: Sensors, analyzers

**Sensors:** Sensors are responsible for collecting data. The input for a sensor may be any part of a system that could contain evidence of an intrusion. Types of input to a sensor include network packets, log files, and system call traces. Sensors collect and forward this information to the analyzer.

Types of input data is collected from various sources including network packets, log files, and system call traces.

Examples of data collection for Host-based IDS

* **Identifying Hosts:** An IDS can create a list of hosts on the organization’s network arranged by IP address or MAC address
* **Identifying Operating Systems:** An IDS can identify the OSs and OS versions used by the hosts
* **Identifying Applications:** An IDS can identify the application versions in use by keeping track of which ports are used and monitoring certain characteristics of application communications

Examples of data collection for Network-based IDS

* Timestamp (usually date and time)
* Session ID (typically a unique number assigned to each TCP connection or like groups of packets for connectionless protocols)
* Source and destination IP addresses
* Source and destination TCP or UDP ports
* Number of bytes transmitted over the connection
* State-related information (e.g., authenticated username)

**Analyzers:** Analyzers receive input from one or more sensors or from other analyzers. The analyzer is responsible for determining if an intrusion has occurred. The output of this component is an indication that an intrusion has occurred. The output may include evidence supporting the conclusion that an intrusion occurred. The analyzer may provide guidance about what actions to take as a result of the intrusion.

Analyzers receive input from one or more sensors and responsible for determining if an intrusion has occurred. The output of this component is an indication that an intrusion has occurred

Two types of analysis for detecting intrusion: Signature-based and anomaly-based

**Signature-based:** A network IDS signature is a pattern that we want to look for in traffic. Some examples are below.

* A TCP packet with the SYN and FIN flags set (a violation of RFC 793, you cannot start a connection and finish the connection at the same packet)
* Packet with an illegal TCP flag combination. For example known bad flag combinations (X-mas attack)
* TCP packet with an acknowledgement value set to a non-zero number, but the acknowledgment flag is not set

**For more examples of signatures please see the document at**

* [**http://www.symantec.com/connect/articles/network-intrusion-detection-signatures-part-one**](http://www.symantec.com/connect/articles/network-intrusion-detection-signatures-part-one)
* [**http://www.symantec.com/connect/articles/network-intrusion-detection-signatures-part-two**](http://www.symantec.com/connect/articles/network-intrusion-detection-signatures-part-two)

Signatures are expressed in terms of rules: An example of rule (Snort tool)

**alert tcp any any -> 192.168.1.0 111 (content:"|00 01 86 a5|"; msg: "mountd ";)**

This above rule describes an alert generated when Snort matches a network packet with all of the following attributes:

* + TCP packet
  + Source from any IP address on any port
  + Destined to the IP address 192.168.1.0 on port 111.

The word(s) before the colons in the rule options section are called option keywords. The rule is triggered if the content of packet includes the hexa-decimal numbers sequence or mountd command.

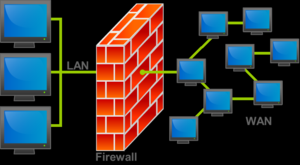
**Anomaly-based IDS:** Anomaly detection starts with profiling normal behavior of traffic or data. An alarm is generated if there is a deviation of traffic data from the normal pattern. Anomaly-based attack detection examples are below.

* TCP SYN attack: connection attempt from a fixed IP address having SYN flag set and coming at a faster rate than normal (e.g., more than 60 connection requests per minute)
* Excessive number of login attempt to a service (SSH login)
* Login attempt beyond normal work hours (e.g., 1AM)
* Excessive bandwidth usage (could mean that one or most host is infected with worms)

# **4. Firewall**

The firewall is an important complement to host-based security services such as intrusion detection systems. Typically, a firewall is inserted between the premises network and the Internet to establish a controlled link and to erect an outer security wall or perimeter. The aim of this perimeter is to protect the premises network from Internet-based attacks and to provide a single choke point where security and auditing can be imposed.

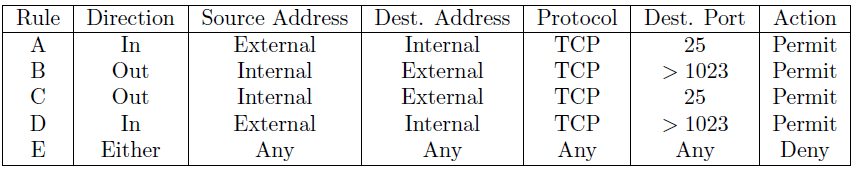
The firewall provides an additional layer of defense, insulating the internal systems from external networks. This follows the classic military doctrine of "defense in depth," which is just as applicable to IT security.

****

Packet filters act by inspecting the "packets" which transfer between computers on the Internet. If a packet matches with the packet filter's set of rules, the packet filter will drop (silently discard) the packet, or reject it (discard it, and send "error responses" to the source). The first Firewall was developed in 1988 by Digital Equipment Corporation (DEC).

**Firewall: packet filter rule:**

One advantage of a packet filtering firewall is its simplicity. IT examines each packet independently based on a rule table.



Rules A and B allow inbound SMTP connections (incoming email).

Rules C and D allow outbound SMTP connections (outgoing email).

Rule E is the default rule that applies if all else fails.

Firewall examines the rules in order, i.e., it starts at the top until it ends a rule that matches the packet, then it takes the action specified

Some weaknesses of packet filter firewalls:

* Because packet filter firewalls do not examine upper-layer data, they cannot prevent attacks that employ application-specific vulnerabilities or functions. For example, a packet filter firewall cannot block specific application commands; if a packet filter firewall allows a given application, all functions available within that application will be permitted.
* Because of the limited information available to the firewall, the logging functionality present in packet filter firewalls is limited. Packet filter logs normally contain the same information used to make access control decisions (source address, destination address, and traffic type).
* Most packet filter firewalls do not support advanced user authentication schemes. Once again, this limitation is mostly due to the lack of upper-layer functionality by the firewall.

# **References**

[1] ONE, A. 1996. Smashing the stack for fun and profit. *Phrack Mag. 7,* 49. <http://insecure.org/stf/smashstack.html>

[2] ERLINGSSON,U. 2007. Low-Level software security: Attacks and defenses. Tech. rep. MSR-TR-07-153, Microsoft Research.

[3] <https://www.tutorialspoint.com/unix/unix-file-permission.htm>

[4] <https://en.wikipedia.org/wiki/Unix_security>