

Secure Operating Systems (CYB610)

Master of Science in Cybersecurity

Lecture 4 OS Attacks and Countermeasures

Dr. Ibrahim Gomaa

*Faculty of Informatics
Midocean University*

Chapter

04

Topics

This chapter covers the following topics and concepts:

- **Types of Malware**
- **Relocation**
- **Base / Bounds Registers**
- **Buffer Overflow attack**
- **Cross Site Scripting (XSS) attack**
- **Input Validation**

Chapter

04

Goals

When you complete this chapter, you will be able to:

- **Define the difference between types of malware**
- **Explain the virus effects**
- **Determine the virus signatures**
- **Build a protected memory and address**
- **Explain what is the memory corruption defenses**
- **How to validating input syntax**



❖ Malware Attack:

- A generic term for software that has a malicious purpose
- Is software that is intentionally included or inserted in a system for a harmful purpose.

❖ Different forms of malicious software (malware) Intended to

- Cause distress to a user
- Damage files or systems
- Disrupt ordinary computer and network functions

❖ Examples

- Viruses, worms
- Logic bomb
- Trojan horses
- Spy-wares
- New ones: Spam/scam, Slammer, Nimda, e-payment frauds, etc...



Name	Description
Virus	Attaches itself to a program and propagates copies of itself to other programs
Worm	Program that propagates copies of itself to other computers
Logic bomb	Triggers action when condition occurs
Trojan horse	Program that contains unexpected additional functionality
Backdoor (trapdoor)	is a secret entry point into a program that allows someone that is aware of the backdoor to gain access without going through the usual security access procedures.
Auto-rooter	Malicious hacker tools used to break into new machines remotely
Kit (virus generator)	Set of tools for generating new viruses automatically
Spammer programs	Used to send large volumes of unwanted e-mail
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial of service (DoS) attack
Keyloggers	Captures keystrokes on a compromised system
Zombie	Program activated on an infected machine that is activated to launch attacks on other machines
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access



<i>Code Type</i>	<i>Characteristics</i>
Rabbit	Code that replicates itself without limit to exhaust resources
Time bomb	Code that triggers action when a predetermined time occurs
Dropper	Transfer agent code only to drop other malicious code, such as virus or Trojan horse
Hostile mobile code agent	Code communicated semi-autonomously by programs transmitted through the web
Script attack, JavaScript, Active code attack	Malicious code communicated in JavaScript, ActiveX, or another scripting language, downloaded as part of displaying a web page

A dropper is a Trojan horse that has been designed to install malware (such as viruses and backdoors) onto a computer. The malware within the dropper can be packaged to evade detection by antivirus software.

Hostile mobile code agent malware refers to malicious software designed to infiltrate mobile devices and carry out harmful actions



<i>Code Type</i>	<i>Characteristics</i>
RAT (Remote Access Trojan)	Trojan horse that, once planted, gives access from remote location
Bot	Semi-autonomous agent, under control of a (usually remote) controller or “herder”; not necessarily malicious
Browser hijacker	Code that changes browser settings, disallows access to certain sites, or redirects browser to others
Tool or toolkit	Program containing a set of tests for vulnerabilities; not dangerous itself, but each successful test identifies a vulnerable host that can be attacked
Scareware	Not code; false warning of malicious code attack



- **Infection mechanisms**

- First, the virus should search for and detect objects to infect
- Installation into the infectable object
 - Writing on the boot sector
 - Add some code to executable programs
 - Add some code to initialization/auto-executable programs

- **Trigger mechanism**

- Date
- Number of infections
- First use

- **Effects: It can be anything**

- A message
- Deleting files
- Formatting disk
- Overloading processor/memory
- Etc.



Year	Name	Characteristics
1982	Elk Cloner	First virus; targets Apple II computers
1985	Brain	First virus to attack IBM PC
1988	Morris worm	Allegedly accidental infection disabled large portion of the ARPANET, precursor to today's Internet
1989	Ghostballs	First multipartite (has more than one executable piece) virus
1990	Chameleon	First polymorphic (changes form to avoid detection) virus
1995	Concept	First virus spread via Microsoft Word document macro
1998	Back Orifice	Tool allows remote execution and monitoring of infected computer
1999	Melissa	Virus spreads through email address book
2000	IloveYou	Worm propagates by email containing malicious script. Retrieves victim's address book to expand infection. Estimated 50 million computers affected.
2000	Timofonica	First virus targeting mobile phones (through SMS text messaging)
2001	Code Red	Virus propagates from 1 st to 20 th of month, attacks whitehouse.gov web site from 20 th to 28 th , rests until end of month, and restarts at beginning of next month; resides only in memory, making it undetected by file-searching antivirus products



Year	Name	Characteristics
2001	Code Red II	Like Code Red, but also installing code to permit remote access to compromised machines
2001	Nimda	Exploits known vulnerabilities; reported to have spread through 2 million machines in a 24-hour period
2003	Slammer worm	Attacks SQL database servers; has unintended denial-of-service impact due to massive amount of traffic it generates
2003	SoBig worm	Propagates by sending itself to all email addresses it finds; can fake From: field; can retrieve stored passwords
2004	MyDoom worm	Mass-mailing worm with remote-access capability
2004	Bagle or Beagle worm	Gathers email addresses to be used for subsequent spam mailings; SoBig, MyDoom, and Bagle seemed to enter a war to determine who could capture the most email addresses
2008	Rustock.C	Spam bot and rootkit virus
2008	Conficker	Virus believed to have infected as many as 10 million machines; has gone through five major code versions
2010	Stuxnet	Worm attacks SCADA automated processing systems; zero-day attack
2011	Duqu	Believed to be variant on Stuxnet
2013	CryptoLocker	Ransomware Trojan that encrypts victim's data storage and demands a ransom for the decryption key



Virus Effect	How It Is Caused
Attach to executable program	⟨ Modify file directory ⟨ Write to executable program file
Attach to data or control file	⟨ Modify directory ⟨ Rewrite data ⟨ Append to data ⟨ Append data to self
Remain in memory	⟨ Intercept interrupt by modifying interrupt handler address table ⟨ Load self in non-transient memory area
Infect disks	⟨ Intercept interrupt ⟨ Intercept operating system call (to format disk, for example) ⟨ Modify system file ⟨ Modify ordinary executable program
Conceal self	⟨ Intercept system calls that would reveal self and falsify result ⟨ Classify self as "hidden" file
Spread infection	⟨ Infect boot sector ⟨ Infect systems program ⟨ Infect ordinary program ⟨ Infect data ordinary program reads to control its execution
Prevent deactivation	⟨ Activate before deactivating program and block deactivation
	⟨ Store copy to reinfect after deactivation



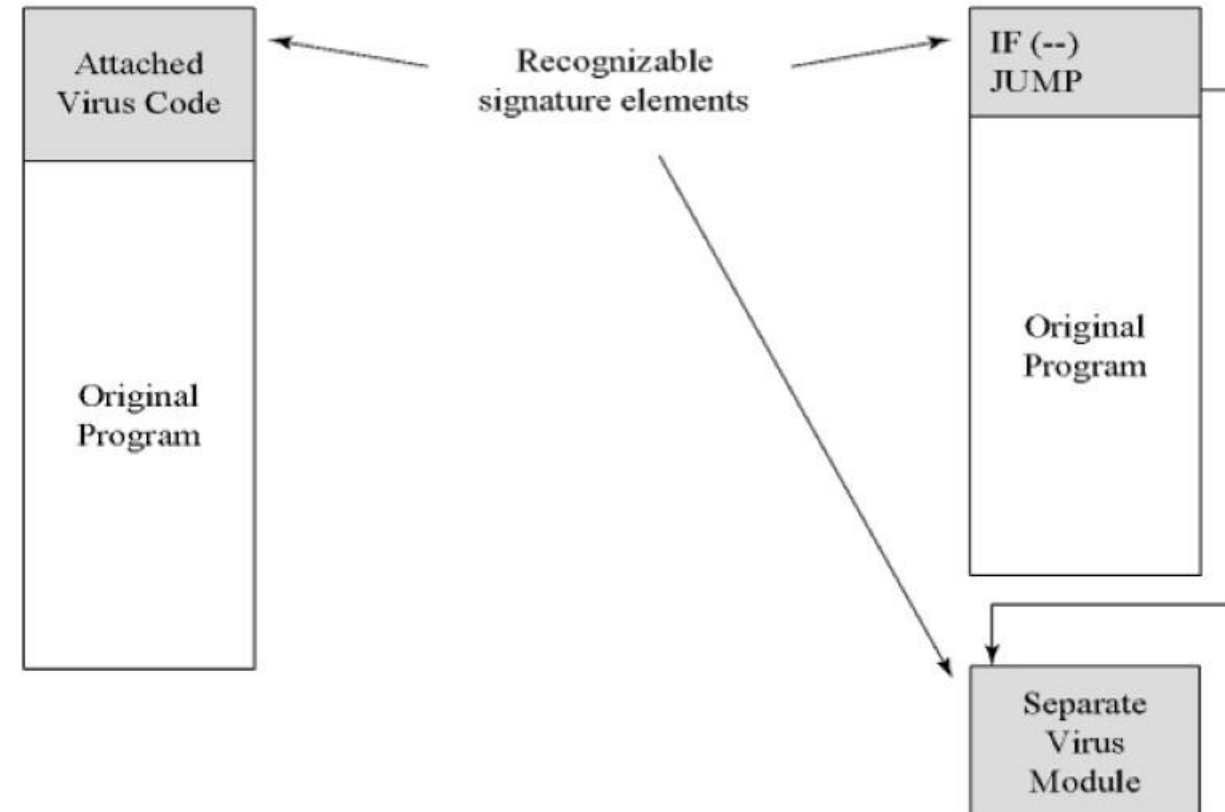
- Use software acquired from reliable sources
- Test software in an isolated environment
- Only open attachments when you know them to be safe
- Treat every website as potentially harmful
- Create and maintain backups



- Virus scanners look for **signs** of malicious code infection using **signatures** in program files and memory
- Traditional virus scanners have trouble keeping up with new malware—detect about 45% of infections
- Detection mechanisms:
 - Known string patterns in files or memory
 - Execution patterns
 - Storage patterns



- Alternatively known as a virus definition, a virus signature is the fingerprint of a virus. It is a set of unique data, or bits of code, that allows it to be identified. Antivirus software uses a virus signature to find a virus in a computer file system, allowing to detect, quarantine, and remove the virus. In the antivirus software, the virus signature is called a definition file or DAT file.





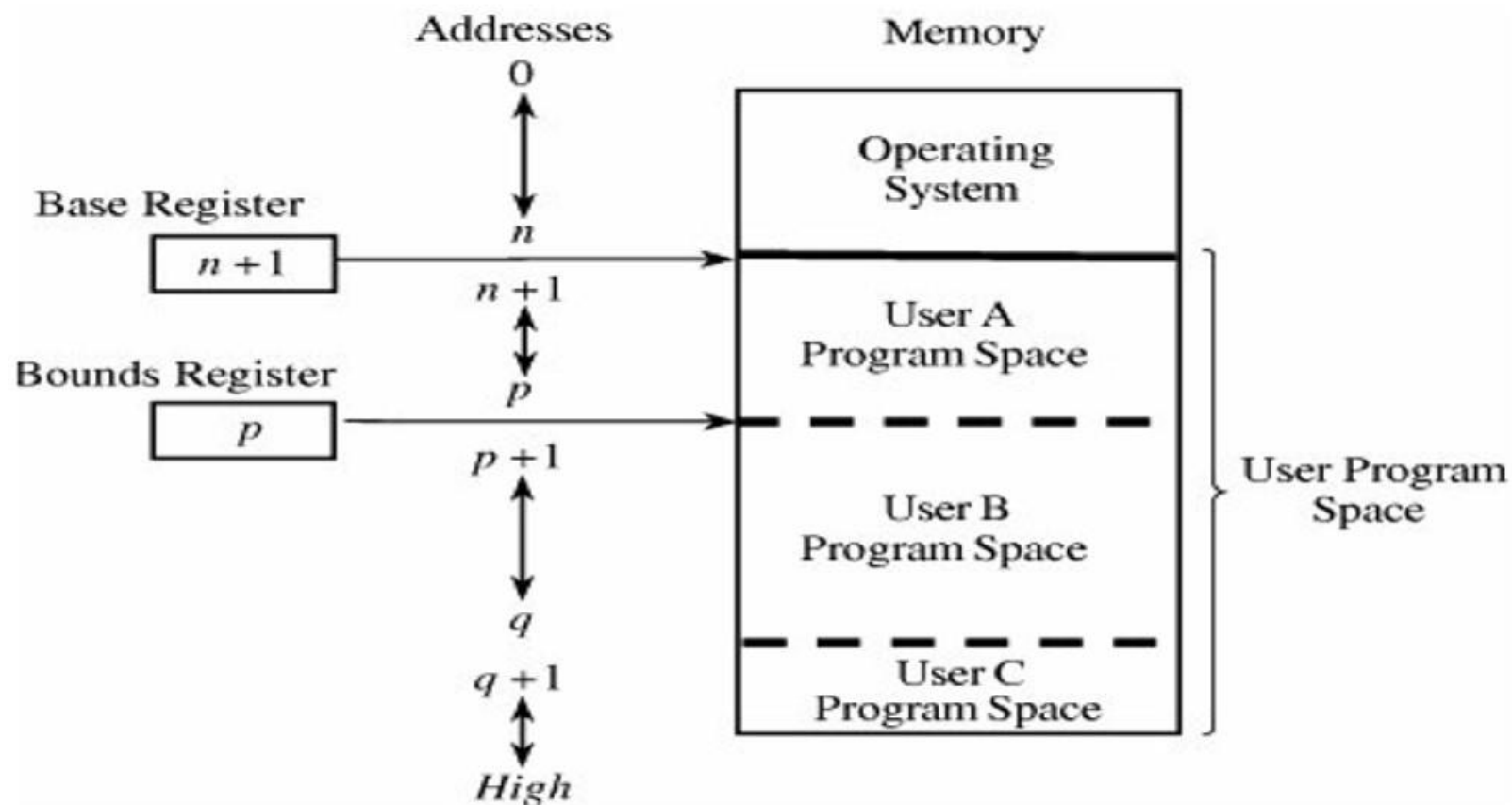
The most obvious problem of multiprogramming is preventing one program from affecting the data and programs in the memory space of other users.

- Fence
- Relocation
- Base/Bounds Registers
- Tagged Architecture
- Segmentation
- Paging
- Combined Paging with Segmentation



- Is the process of taking a program written as if it began at address **0** and changing all addresses to reflect the actual address at which the program is located in memory.
- By adding a constant relocation factor to each address of the program.
- The **relocation factor** is the starting address of the memory assigned for the program.

Relocation, in the context of memory address protection, is the process of adjusting the addresses within a program or data so that it can be loaded and executed at a different location in memory than where it was initially designed to run.





- By previous methods with two or more users, none can know in advance where a program will be loaded for execution.
- The **relocation register** solves the problem by providing a **base or starting address**.
- All addresses inside a program are offsets from that base address.
- A variable **fence register** is generally known as a **base register**.

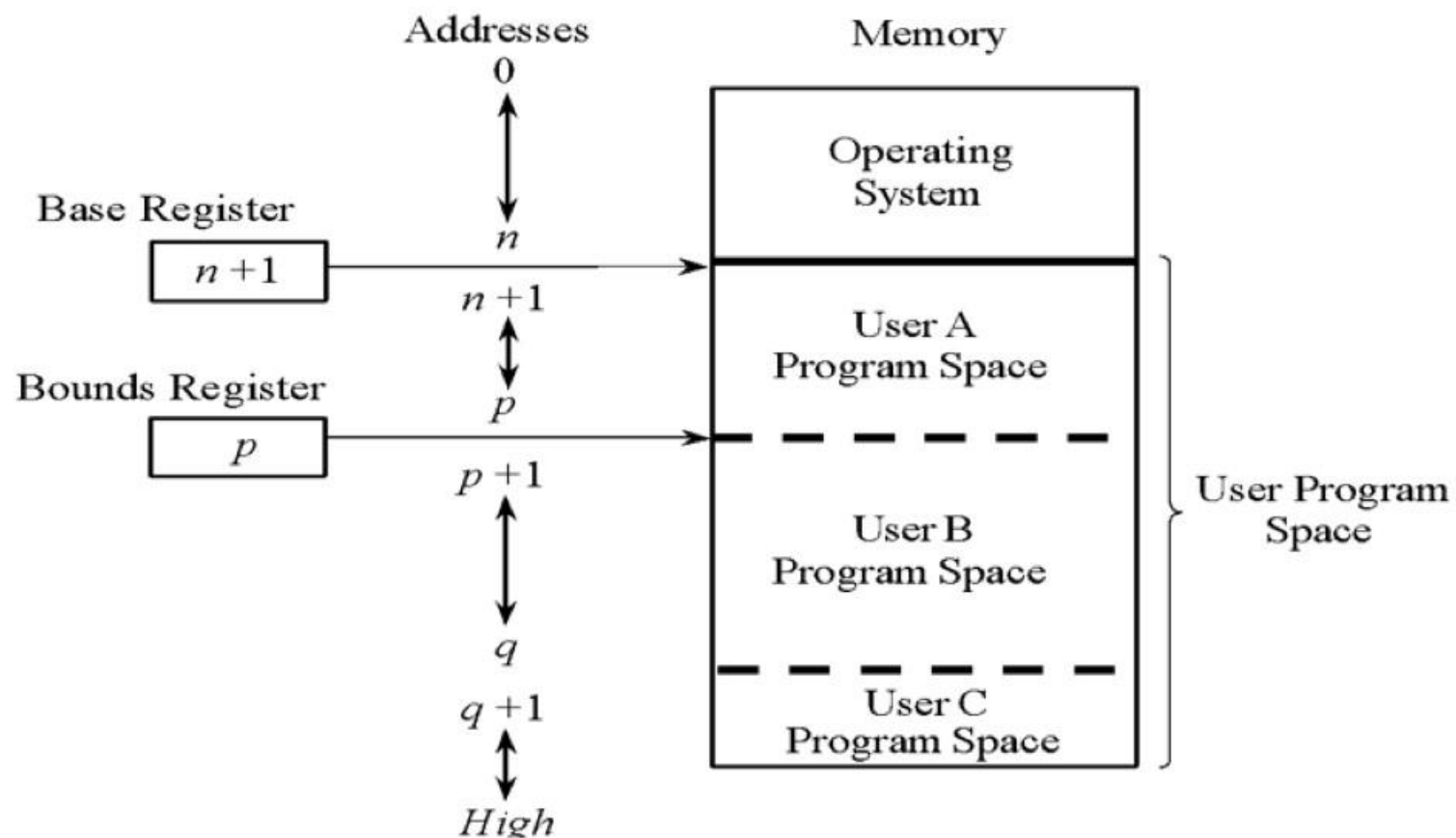


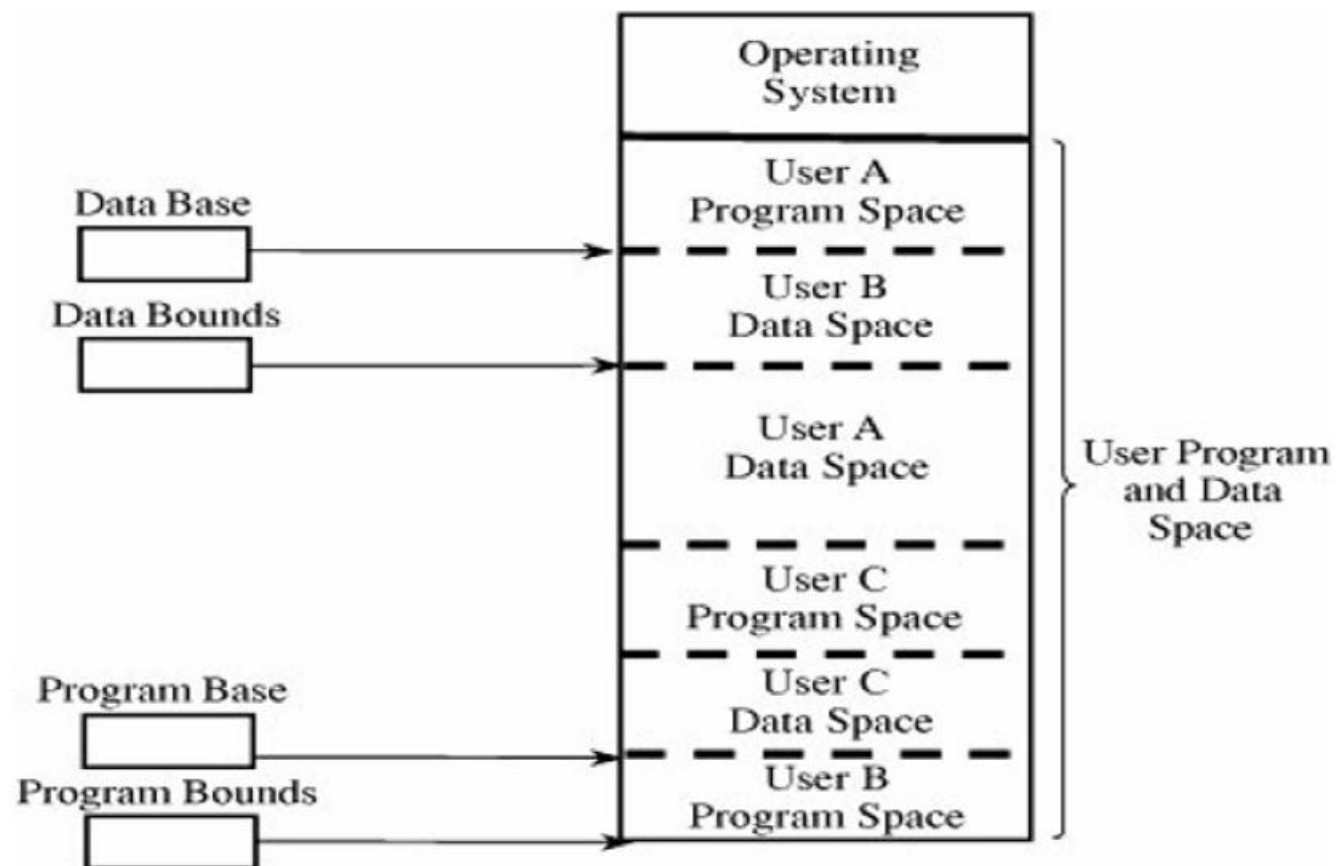
- **Context Switch**

- When execution changes from one user's program to another's, the operating system must change the contents of the base and bounds registers to reflect the true address space for that user.
- This change is part of the general preparation, called a **context switch**.
- Erroneous addresses inside a user's address space can still affect that program because the base/bounds checking guarantees only that each address is inside the user's address space.



- **For example**, a user error might occur when a subscript is out of range or an undefined variable generates an address reference within the user's space but, unfortunately, inside the executable instructions of the user's program.
- We can solve this **overwriting problem** by using another **pair of base/bounds registers**, one for the **instructions (code)** of the program and a second for the **data space**.







Tagged architecture enhances memory address protection by associating extra bits (tags) with each memory word, acting as metadata that defines the data type or function.

These tags are checked at runtime by the hardware to enforce security policies, preventing unauthorized access and thus enhancing security. This contrasts with software-based approaches which can be more vulnerable to attacks

As shown in the figure, one memory location may be protected as execute-only (for example, the object code of instructions), whereas another is protected for fetch-only (for example, read) data access, and another accessible for modification (for example, write). In this way, two adjacent locations can have different access rights.

Tag	Memory Word
R	0001
RW	0137
R	0099
X	
X	
X	
X	
X	
X	
R	4091
RW	0002

Code: R = Read-only RW = Read/Write
X = Execute-only



- Overwrite the **program counter** stored in the stack
- Overwrite **part of the code** in low memory, substituting new instructions
- Overwrite the **program counter and data** in the stack so that the program counter points to the stack



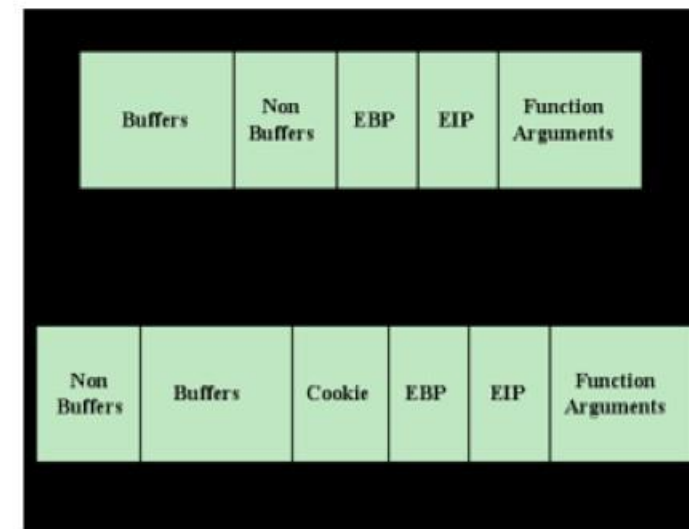
- **Overwrite:**
 - Another piece of your program's data
 - An instruction in your program
 - Data or code belonging to another program
 - Data or code belonging to the operating system
- Overwriting a **program's** instructions gives attackers that **program's execution privileges**
- Overwriting **operating system** instructions gives attackers the **operating system's execution privileges**



- Staying within bounds
 - Check lengths before writing
 - Confirm that array subscripts are within limits
 - Double-check boundary condition code for off-by-one errors
 - Limit input to the number of acceptable characters
 - Limit programs' privileges to reduce potential harm
- Many languages have overflow protections
- Code analyzers can identify many overflow vulnerabilities

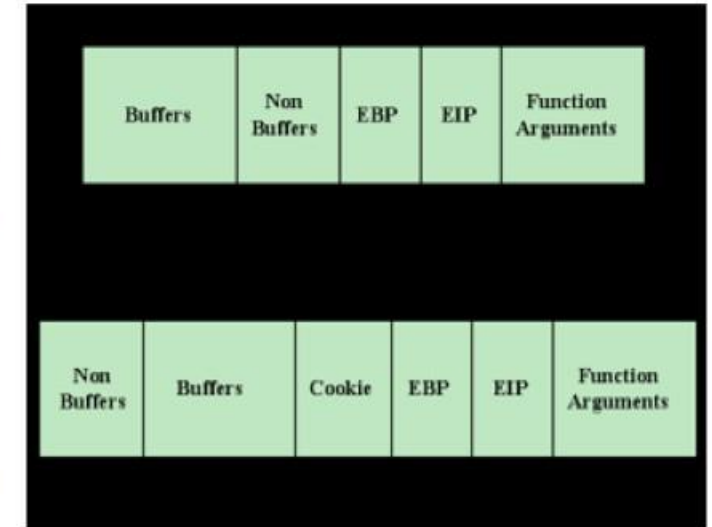


- **Normally a function's stack looks like the Figure**
- Note the stacks EBP (extended base pointer) and EIP (extended instruction pointer). When the function returns, it must continue execution at the next instruction after the instruction that called this function.
- The CPU does this by taking the values off stack & populating the EBP & EIP registers.
- If the attacker can overflow the buffer on the stack, they can overrun the data used to populate the EBP & EIP registers with values & change the application's execution flow.





- A cookie has been inserted between stack data and function return address. This random value is checked when the function exits, and if the cookie is corrupted, the application is halted.
- Buffers on the stack are placed in higher memory than non buffers, such as function pointers, and scalar values to make it harder for some attacks to succeed.
- By **switching the order around**, the attacker must take advantage of a buffer underrun, which is rarer, to successfully corrupt the function pointer. There are variants of the buffer overrun that will still corrupt a function pointer, such as corrupting a stack frame in higher memory.





- The 2010 Australian Signals Directorate (ASD) lists the “Top 35 Mitigation Strategies”.
- **Note:** implementing just the top **four** of these strategies would have prevented at least 85% of the targeted cyber intrusions investigated by ASD
- **The top four strategies for prevention are:**
 - ✓ White-list approved applications
 - ✓ Patch third-party applications and operating system vulnerabilities
 - ✓ Restrict administrative privileges
 - ✓ Create a defense-in-depth system



- Possible for a system to be compromised during the installation process before it can install the latest patches
- Building and deploying a system should be a planned process designed to counter this threat
- **Process must:**
 - Assess risks and plan the system deployment
 - Secure the underlying OS and then the key applications
 - Ensure any critical content is secured
 - Ensure appropriate network protection mechanisms are used
 - Ensure appropriate processes used to maintain security



- Flaws relating to **invalid handling of input data**, specifically when program input data can accidentally or deliberately influence the flow of execution of the program
 - This script contains a critical vulnerability.
 - The value of the user is passed directly to the finger program as a parameter.
 - If the identifier of a legitimate user is supplied, for example, lpb, then the output will be the information on that user.
 - The attacker is able to run any program on the system with the privileges of the Web server.

Most often occur in scripting languages

- Encourage reuse of other programs and system utilities where possible to save coding effort

```
1 #!/usr/bin/perl
2 # finger.cgi - finger CGI script using Perl's CGI module
3
4 use CGI;
5 use CGI::Carp qw(fatalsToBrowser);
6 $q = new CGI; # create query object
7
8 # display HTML header
9 print $q->header;
10 $q->start_html('Finger User');
11 $q->h1('Finger User');
12 print "<pre>";
13
14 # get name of user and display their finger details
15 $user = $q->param("user");
16 print "/usr/bin/finger -sh $user";
17
18 # display HTML footer
19 print "</pre>";
20 print $q->end_html;
```

```
<html><head><title>Finger User</title></head><body>
<h1>Finger User</h1>
<form method=post action="finger.cgi">
<b>Username to finger</b>: <input type=text name=user value="">
<p><input type=submit value="Finger User">
</form></body></html>
```

```
Finger User
Login Name TTY Idle Login Time Where
lpb Lavie Brown p0 Sat 15:24 ppp4l.grapevine

Finger User
attack success
-XXXX-XX-X 1 lpb staff 537 Oct 21 16:19 finger.cgi
-XX-X--X-- 1 lpb staff 231 Oct 21 16:14 finger.html
```

```
14 # get name of user and display their finger details
15 $user = $q->param("user");
16 die "The specified user contains illegal characters!"
17 unless ($user =~ /\w+/);
18 print "/usr/bin/finger -sh $user";
```



- Another broad class of vulnerabilities concerns inputs provided to a program by one user that is subsequently output to another user. Such attacks are known as cross-site scripting (XSS) attacks because they are most commonly seen in scripted Web applications. This vulnerability involves the inclusion of script code in the HTML content of a Web page displayed by a user's browser
- Cross-site scripting attacks exploit this assumption and attempt to **bypass the browser's security checks** to gain elevated access privileges to sensitive data belonging to another site. These data can include page contents, session cookies, and a variety of other objects.
- The attacker **includes the malicious script content in data supplied to a site**. If this content is subsequently displayed to other users without sufficient checking, they will execute the script assuming it is trusted to access any data associated with that site.
- Consider the widespread use of guestbook allowing comments, which are subsequently viewed by other users. Unless the contents of these comments are checked and any dangerous code removed, the attack is possible.



Attacks where input provided by one user is subsequently output to another user

Commonly seen in scripted Web applications

- Vulnerability involves the inclusion of script code in the HTML content
- Script code may need to access data associated with other pages
- Browsers impose security checks and restrict data access to pages originating from the same site

Exploit assumption that all content from one site is equally trusted and hence is permitted to interact with other content from the site

XSS reflection vulnerability

- Attacker includes the malicious script content in data supplied to a site



- Given that the programmer cannot control the content of input data, it is necessary to ensure that such data conform with any assumptions made about the data before subsequent use. If the data are textual, these assumptions may be that the data contain only printable characters, have certain HTML markup, are the name of a person, a user-id, an e-mail address, a filename, and/or a URL.
- Alternatively, the data might represent an integer or other numeric value. A program using such input should confirm that it meets these assumptions. An important principle is that input data should be compared against what is wanted, accepting only valid input. The alternative is to compare the input data with known dangerous values.
- The problem with this approach is that new problems and methods of bypassing existing checks continue to be discovered. By trying to block known dangerous input data, an attacker using a new encoding may succeed. By only accepting known safe data, the program is more likely to remain secure.
- This type of comparison is commonly done using regular expressions. It may be explicitly coded by the programmer or may be implicitly included in a supplied input processing routine. A regular expression is a pattern composed of a sequence of characters that describe allowable input variants. An appropriate reference should be consulted for the language in use.
- If the input data fail the comparison, they could be rejected. In this case a suitable error message should be sent to the source of the input to allow it to be corrected and reentered. Alternatively, the data may be altered to conform.



It is necessary to ensure that data conform with any assumptions made about the data before subsequent use



Input data should be compared against what is wanted



Alternative is to compare the input data with known dangerous values



By only accepting known safe data the program is more likely to remain secure



- The issue of multiple, alternative encodings of the input data could occur **because the data are encoded in HTML** or some other structured encoding that allows **multiple representations of characters**.
- Growing requirement to support users around the globe, **The Unicode character** set is now widely used for this purpose. It is the native character set used in the Java language, for example. It is also the native character set used by operating systems such as Windows XP and later.
- A Unicode character can be encoded to accommodate many programs, databases, and other computer and communications applications. **Any specific character is supposed to have a unique encoding**.
- Consider the consequences of multiple encodings when validating input. There is a class of attacks that attempt to supply an absolute pathname for a file to a script that expects only a simple local filename. The common check to prevent this is to ensure that the supplied filename does not start with "/" and does not contain any "../" parent directory references.
- If this check only assumes the correct, encoding of slash, then an attacker using one of the longer encodings could avoid this check.



May have multiple means of encoding text

Growing requirement to support users around the globe and to interact with them using their own languages

Unicode used for internationalization

- Many Unicode decoders accept any valid equivalent sequence

Canonicalization

- Transforming input data into a single, standard, minimal representation
- Once this is done the input data can be compared with a single representation of acceptable input values



Programs can be vulnerable to PATH variable manipulation

- Must reset to “safe” values

If dynamically linked may be vulnerable to manipulation of LD_LIBRARY_PATH

- Used to locate suitable dynamic library
- Must either statically link privileged programs or prevent use of this variable

Use of Least Privilege

Privilege escalation

- Exploit of flaws may give attacker greater privileges

Least privilege

- Run programs with least privilege needed to complete their function

Determine appropriate user and group privileges required

- Decide whether to grant extra user or just group privileges

Ensure that privileged program can modify only those files and directories necessary

Programs with root/
administrator privileges
are a major target of
attackers

- They provide highest levels of system access and control
- Are needed to manage access to protected system resources

Often privilege is only
needed at start

- Can then run as normal user

Good design partitions
complex programs in
smaller modules with
needed privileges

- Provides a greater degree of isolation between the components
- Reduces the consequences of a security breach in one component
- Easier to test and verify



- Many programs use temporary files. Often in common, shared system area
- Must be unique, not accessed by others
- Commonly create name using process ID
 - Unique, but predictable
 - Attacker might guess and attempt to create own file between program checking and creating
- Secure temporary file creation/use requires use of random names

Programs may use functionality and services of other programs

- **Security vulnerabilities can result unless care is taken with this interaction**
 - Such issues are of particular concern when the program being used did not adequately identify all the security concerns that might arise
 - Occurs with the current trend of providing Web interfaces to programs
 - Burden falls on the newer programs to identify and manage any security issues that may arise

Issue of data confidentiality/integrity

Detection and handling of exceptions and errors generated by interaction is also important from a security perspective



- Types of Malware
- Relocation Base/Bounds Registers and Tagged Architecture
- The difference between types of Malware
- The Virus Effects
- The Virus Signatures
- Protected Memory and Address
- Buffer Overflows
- Cross Site Scripting (XSS) Attacks
- The memory corruption defenses
- Validating input syntax



Thank You