Information Security Assignment

NAME	Ameya S. Daddikar
COLLEGE I.D.	161070015
YEAR & PROGRAM	Final Year Btech. Computer Engineering

Q1. Explain covert channels and their types.

A covert channel is any method of communication that is used to illicitly transfer information, thus breaking the security policy of the system. Any shared resources can potentially be used as a covert channel.

According to the Department of Defense's (DoD) Trusted Computer System Evaluation Criteria (TSEC), "A covert channel is any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy."

To illustrate the problem more fully, let us use an analogy. Alice and Bob have been incarcerated and placed in two separate jail cells. They want to coordinate an escape plan. However, they have a small problem. All messages that they send to each other must first be read by the warden before being passed on. In order to be able to coordinate their plans while at the same time keeping them hidden from the warden, they communicate with each other in code. Each word with an even number of letters is read as a 1. Each word with an odd number of letters is read as a 0. For example, if Bob sent a message to Alice asking "Hey, what are you up to," Alice would interpret is as "010011." The warden, in this case, has been used as a covert channel.

There are many threats that modern network security must take into account. From brute force password attacks to port scanning, the issues, which system engineers and administrators must worry about, increase at a faster than normal pace.

Types of Covert Channels:

1. Storage Channels

   Covert storage channels are methods of communication that "include all vehicles that would allow the direct or indirect writing of a storage location by one process and the direct or indirect reading of it by another".

   In other words, one process writes to a shared resource, while another process reads from it. Storage channels can be used between processes within a single computer or between multiple computers across a network.

A good example of a storage channel is a printing queue. The process with higher security privileges, the sending process, either fills up the printer queue to signal a 1 or leaves it as it is to signal a 0. The process with lower security privileges, the receiving process, polls the printer queue to see whether or not it is full and determines the value accordingly.

2. Timing Channels

   Covert timing channels are methods of communication that “include all vehicles that would allow one process to signal information to another process by modulating its own use of system resources in such a way that the change in response time observed by the second process would provide information”.

   It is essentially any method that uses a clock or measurement of time to signal the value being sent over channel. Similarly to storage channels, timing channels can exist both in a single-computer setting and a network setting. However, they are less practical in a network setting.

   An example of a timing channel can be found in a movable head I/O device, such as a hard disk. One process with higher security privileges, the sending process, has access to the entire device while another process with lower security privileges, the receiving process, only has access to a small portion of the device. Requests to the device are processed serially. To signal a 1, the sending process makes a read request far away from the section that the receiving process has access to. To signal a 0, it does nothing. The receiving process makes a read request within its own section and uses the time it takes for the head to travel to the section and finish the read request to determine the value accordingly.

Q2. What are stored procedures?

A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs.

Stored procedures can access or modify data in a database, but it is not tied to a specific database or object, which offers a number of advantages.

Advantages of Stored Procedures

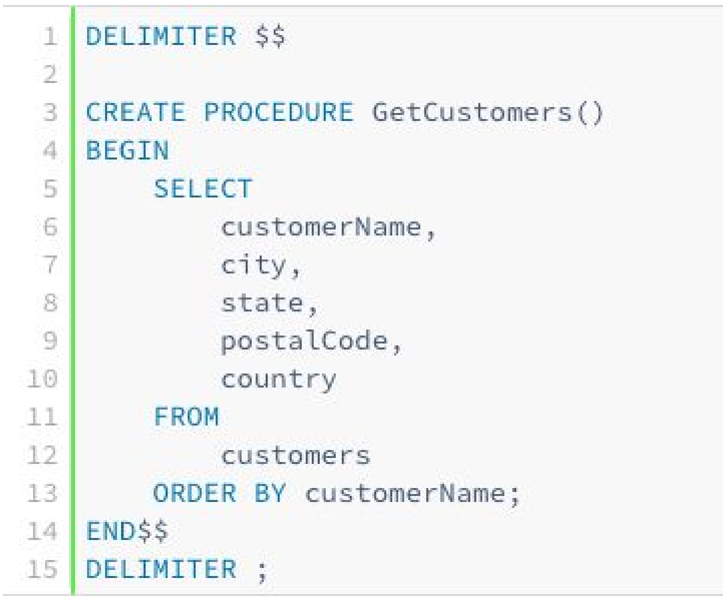
1. A stored procedure provides an important layer of security between the user interface and the database. It supports security through data access controls because end users may enter or change data, but do not write procedures.
2. A stored procedure preserves data integrity because information is entered in a consistent manner. It improves productivity because statements in a stored procedure only must be written once.

3. Stored procedures offer advantages over embedding queries in a graphical user interface (GUI). Since stored procedures are modular, it is easier to troubleshoot when a problem arises in an application.

4. Stored procedures are also tunable, which eliminates the need to modify the GUI source code to improve its performance. It's easier to code stored procedures than to build a query through a GUI.

5. Use of stored procedures can reduce network traffic between clients and servers, because the commands are executed as a single batch of code. This means only the call to execute the procedure is sent over a network, instead of every single line of code being sent individually.

Given below, is the syntax for creating stored procedures in a MySQL server.



```
DELIMITER $$

CREATE PROCEDURE GetCustomers()
BEGIN
    SELECT
        customerName,
        city,
        state,
        postalCode,
        country
    FROM
        customers
    ORDER BY customerName;
END$$
DELIMITER ;
```

Q3. Differentiate between stateful firewall and circuit level gateway.

STATEFUL FIREWALL	CIRCUIT LEVEL GATEWAY
Stateful inspection firewalls judge according to information from multiple packets.	A circuit-level gateway firewall connects two separate subnetworks as if they were one contiguous unit.
A stateful inspection firewall maintains state information from one packet to another in the input stream.	One use of the circuit-level gateway firewall is to setup a virtual private network.
Attackers may try to attack multiple packets by forcing some of the packets to have very short lengths so the firewalls may not detect them. However, a stateful inspection firewall would track the sequence of packets and conditions and thwart any such attack.	Say N/W ‘X’ and N/W ‘Y’ want to communicate securely. The circuit gateway at X routes all traffic to Y via an encryption unit. When traffic returns, the firewall at X routes all traffic from Y through the encryption unit (for decryption).

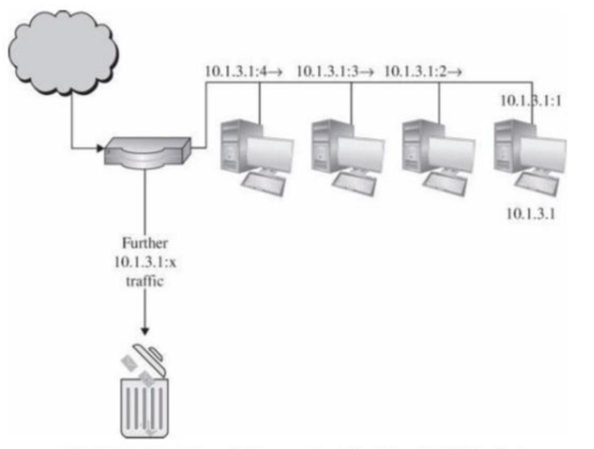


FIGURE 6-55 Stateful Inspection Blocking Multiple Probes

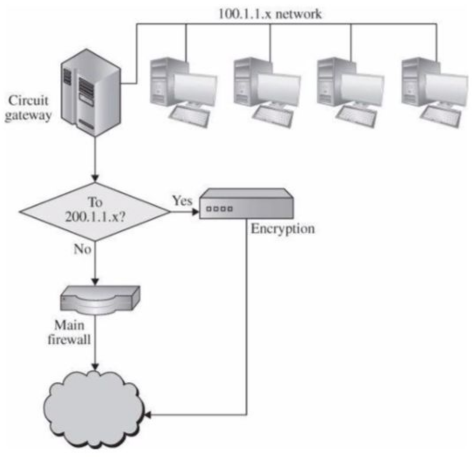


FIGURE 6-57 Circuit-Level Gateway

Q4. Explain the mitigation of reflection attack.

Implement an Anti-CSRF Token

An Anti-CSRF token is a type of CSRF protection. It is a random string that is only known by the user's browser and the web application. The anti-CSRF token is usually stored inside a session variable. It is typically on a page inside a hidden form field which is sent together with the request.

If the value of the session variable and the hidden form field match, the web application accepts the request. If they do not match, then the request is dropped; since in this case, the attacker does not know the exact value of the hidden form field that is hidden for the request to be accepted, he cannot launch a CSRF attack.

Setting the HttpOnly attribute of the session cookie

When the HttpOnly attribute is set to true, then it cannot be accessed by a client-side script. This can help mitigate cross-site scripting threats that result in stolen cookies that may contain sensitive information identifying the user to the site, such as the ASP.NET session ID or forms authentication ticket, and can be replayed by the attacker in order to masquerade as the user or obtain sensitive information. Although this attribute does help mitigate the risk of XSS attacks, it has no impact towards a CSRF attack.

Use the Same-Site Flag in cookies

The Same-Site Flag in cookies is a relatively new method that is being used to prevent CSRF attacks and improve web application security. In the above scenario, we saw that https://attacker.com/ could send a POST request to https://example.com/ together with a session cookie. This session cookie is unique for every user and the web application uses it to distinguish different users from each other, and to determine if you are logged in.

If the session cookie is marked as Same-Site cookie, it is only sent along with requests that originate from the same domain. Therefore when https://example.com/index.php wants to make a POST request to https://example.com/postcomment.php it is allowed. However, https://attackers.com/ cannot make POST requests to https://example.com/postcomment.php since the session cookie is not sent along with the request, because it originates from a different domain.

Other Methods

- Request users to re-authenticate when performing important actions (e.g. authorising a payment to another bank account)
- Logging off web applications when not in use
- Securing usernames and passwords
- Not allowing browsers to remember passwords
- Avoiding simultaneously browsing while logged into an application.

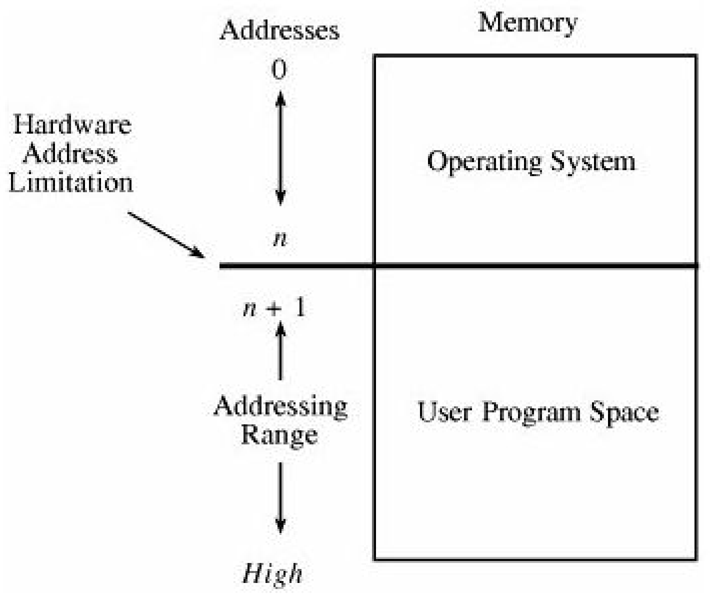
Q5. Elaborate on different memory and address protection mechanisms.

The most obvious problem of multiprogramming is preventing one program from affecting the data and programs in the memory space of other users. Fortunately, protection can be built into the hardware mechanisms that control efficient use of memory, so solid protection can be provided at essentially no additional cost.

Fence

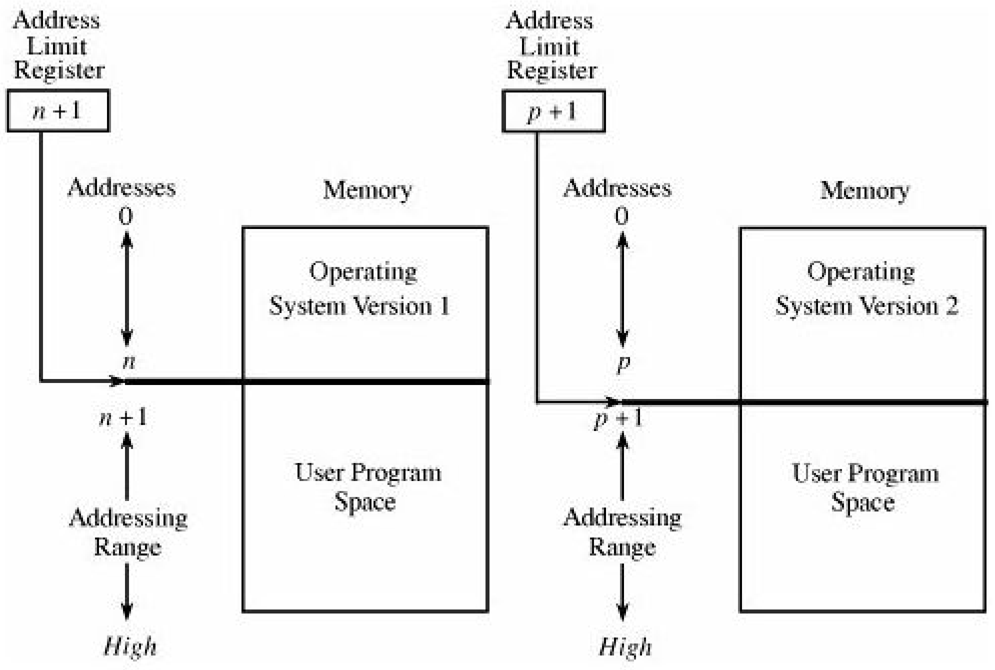
The simplest form of memory protection was introduced in single-user operating systems to prevent a faulty user program from destroying part of the resident portion of the operating system. As its name implies, a fence is a method to confine users to one side of a boundary.

In one implementation, the fence was a predefined memory address, enabling the operating system to reside on one side and the user to stay on the other. An example of this situation is shown in the figure below.



Unfortunately, this kind of implementation was very restrictive because a predefined amount of space was always reserved for the operating system, whether it was needed or not. If less than the predefined space was required, the excess space was wasted. Conversely, if the operating system needed more space, it could not grow beyond the fence boundary.

Another implementation used a hardware register, often called a fence register, containing the address of the end of the operating system. In contrast to a fixed fence, in this scheme the location of the fence could be changed. Each time a user program generated an address for data modification, the address was automatically compared with the fence address. If the address was greater than the fence address (that is, in the user area), the instruction was executed; if it was less than the fence address (that is, in the operating system area), an error condition was raised.



A fence register protects only in one direction. In other words, an operating system can be protected from a single user, but the fence cannot protect one user from another user. Similarly, a user cannot identify certain areas of the program as inviolable (such as the code of the program itself or a read-only data area).

Relocation

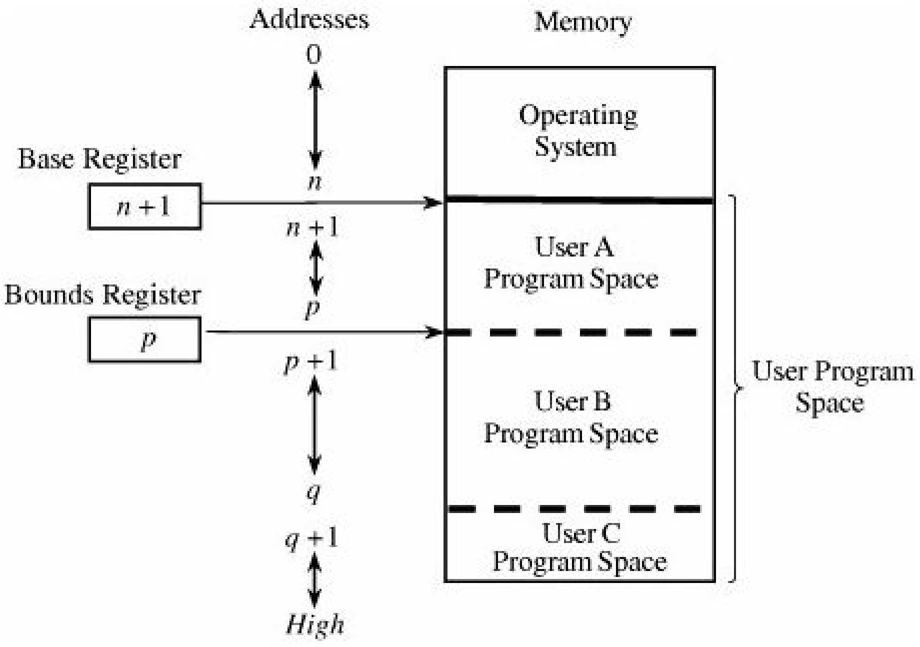
Relocation is the process of taking a program written as if it began at address 0 and changing all addresses to reflect the actual address at which the program is located in memory. In many instances, this effort merely entails adding a constant relocation factor to each address of the program. That is, the relocation factor is the starting address of the memory assigned for the program.

Conveniently, the fence register can be used in this situation to provide an important extra benefit: The fence register can be a hardware relocation device. The contents of the fence register are added to each program address. This action both relocates the address and guarantees that no one can access a location lower than the fence address. Special instructions can be added for the few times when a program legitimately intends to access the location of the operating system.

Base/Bounds Registers

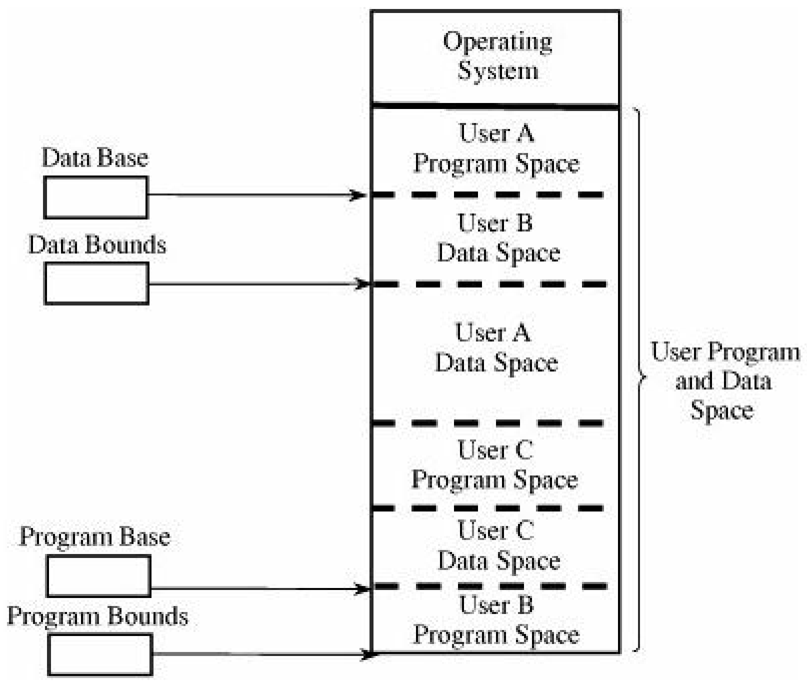
A major advantage of an operating system with fence registers is the ability to relocate; this characteristic is especially important in a multiuser environment. With two or more users, none can know in advance where a program will be loaded for execution. The relocation register solves the problem by 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.

Fence registers provide a lower bound (a starting address) but not an upper one. An upper bound can be useful in knowing how much space is allotted and in checking for overflows into "forbidden" areas. To overcome this difficulty, a second register is often added, as shown below. The second register, called a bounds register, is an upper address limit, in the same way that a base or fence register is a lower address limit. Each program address is forced to be above the base address because the contents of the base register are added to the address; each address is also checked to ensure that it is below the bounds address. In this way, a program's addresses are neatly confined to the space between the base and bounds registers.



With a pair of base/bounds registers, a user is perfectly protected from outside users, or, more correctly, outside users are protected from errors in any other user's program. 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.

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. Then, only instruction fetches (instructions to be executed) are relocated and checked with the first register pair, and only data accesses (operands of instructions) are relocated and checked with the second register pair.

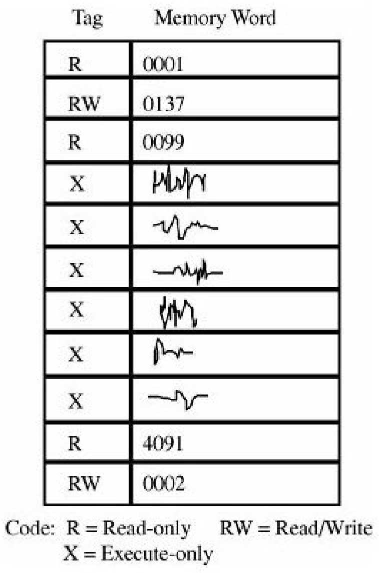


Although two pairs of registers do not prevent all program errors, they limit the effect of data-manipulating instructions to the data space. The pairs of registers offer another more important advantage: the ability to split a program into two pieces that can be relocated separately.

Tagged Architecture

Another problem with using base/bounds registers for protection or relocation is their contiguous nature. Each pair of registers confines accesses to a consecutive range of addresses. A compiler or loader can easily rearrange a program so that all code sections are adjacent and all data sections are adjacent.

An alternative is tagged architecture, in which every word of machine memory has one or more extra bits to identify the access rights to that word. These access bits can be set only by privileged (operating system) instructions. The bits are tested every time an instruction accesses that location.



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

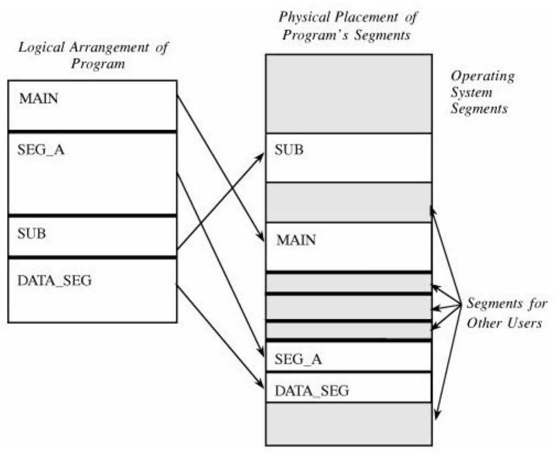
Segmentation

Segmentation involves the simple notion of dividing a program into separate pieces. Each piece has a logical unity, exhibiting a relationship among all of its code or data values. For example, a segment may be the code of a single procedure, the data of an array, or the collection of all local data values used by a particular module.

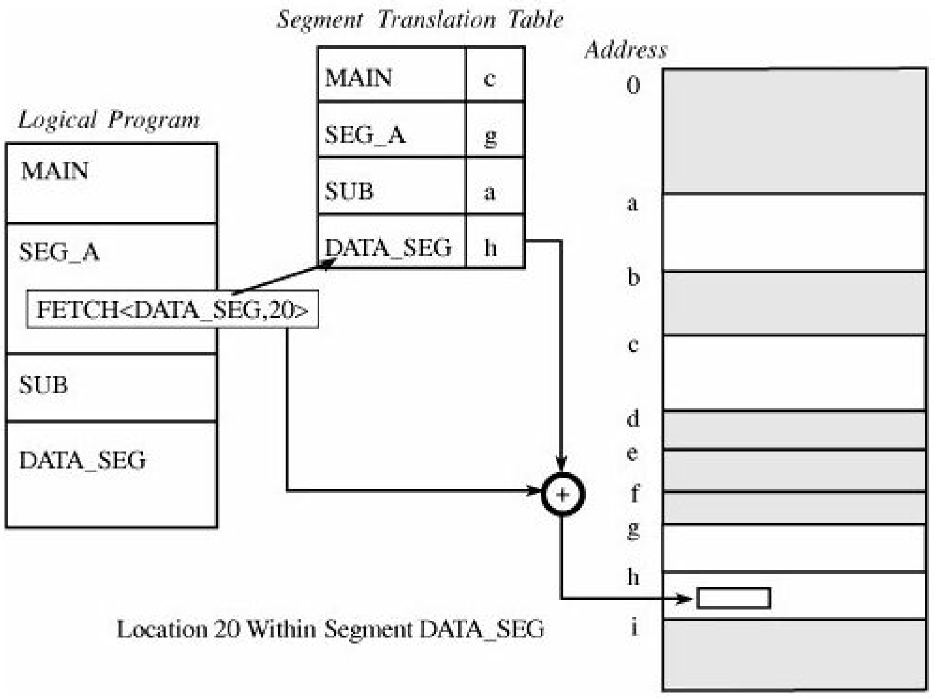
Segmentation was developed as a feasible means to produce the effect of the equivalent of an unbounded number of base/bounds registers. In other words, segmentation allows a program to be divided into many pieces having different access rights.

Each segment has a unique name. A code or data item within a segment is addressed as the pair <name, offset>, where name is the name of the segment containing the data item and offset is its location within the segment (that is, its distance from the start of the segment).

Logically, the programmer pictures a program as a long collection of segments. Segments can be separately relocated, allowing any segment to be placed in any available memory locations. The relationship between a logical segment and its true memory position is shown in figure below.


Logical Arrangement of Program
MAIN
SEG_A
SUB
DATA_SEG
Physical Placement of Program's Segments
Operating System Segments
SUB
MAIN
SEG_A
DATA_SEG
Segments for Other Users

The operating system must maintain a table of segment names and their true addresses in memory. When a program generates an address of the form <name, offset>, the operating system looks up name in the segment directory and determines its real beginning memory address. To that address the operating system adds offset, giving the true memory address of the code or data item.



Segmentation offers these security benefits:

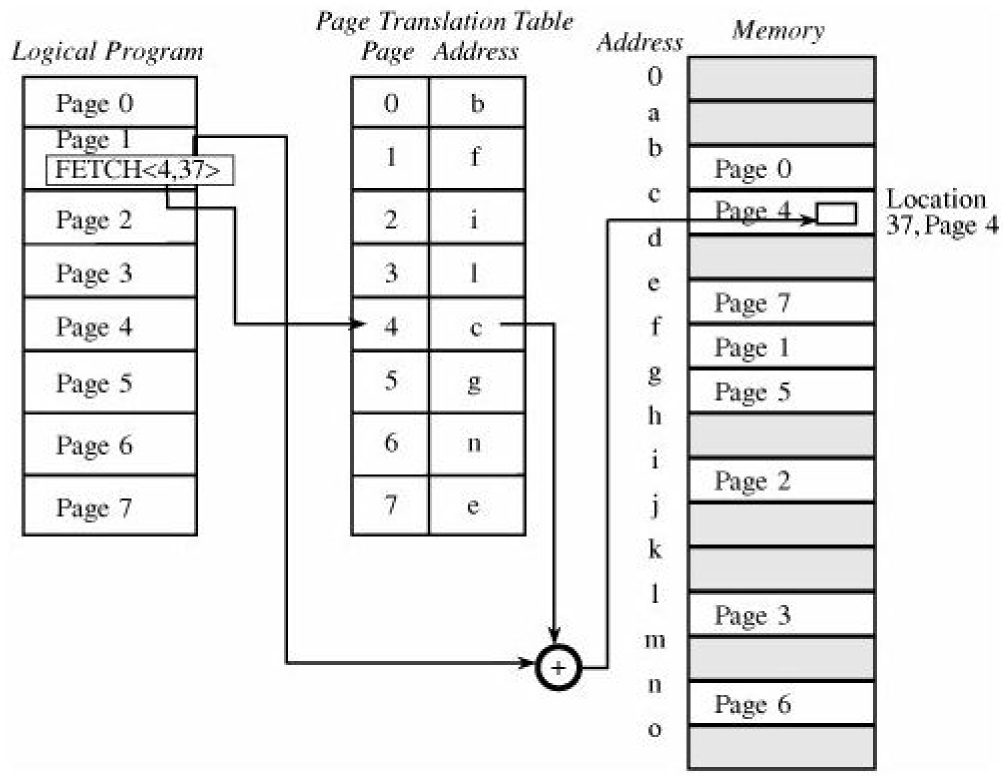
- Each address reference is checked for protection.
- Many different classes of data items can be assigned different levels of protection.
- Two or more users can share access to a segment, with potentially different access rights.
- A user cannot generate an address or access to an unpermitted segment.

Paging

One alternative to segmentation is paging. The program is divided into equal-sized pieces called pages, and memory is divided into equal-sized units called page frames. (For implementation reasons, the page size is usually chosen to be a power of two between 512 and 4096 bytes.) As with segmentation, each address in a paging scheme is a two-part object, this time consisting of <page, offset>.

Each address is again translated by a process similar to that of segmentation: The operating system maintains a table of user page numbers and their true addresses in memory.

The page portion of every <page, offset> reference is converted to a page frame address by a table lookup; the offset portion is added to the page frame address to produce the real memory address of the object referred to as <page, offset>. This process is illustrated in the figure below.



Unlike segmentation, all pages in the paging approach are of the same fixed size, so fragmentation is not a problem. Each page can fit in any available page in memory, and thus there is no problem of addressing beyond the end of a page. The binary form of a <page, offset> address is designed so that the offset values fill a range of bits in the address. Therefore, an offset beyond the end of a particular page results in a carry into the page portion of the address, which changes the address.

Q6. Explain Onion Routing/ Garlic Routing.

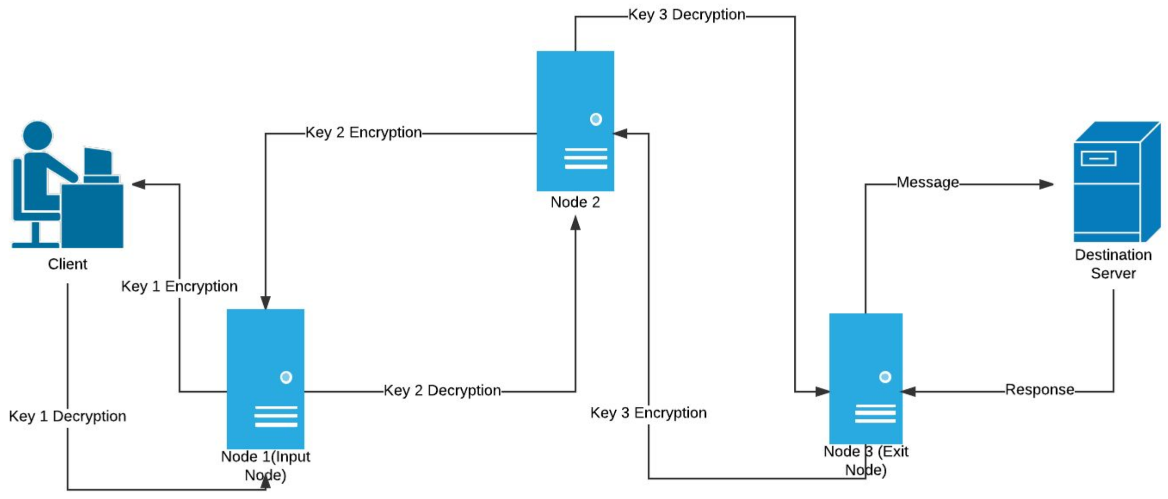
Onion Routing

The Onion Routing program is made up of projects researching, designing, building, and analyzing anonymous communications systems. The focus is on practical systems for low-latency Internet-based connections that resist traffic analysis, eavesdropping, and other attacks both by outsiders (e.g. Internet routers) and insiders (Onion Routing servers themselves). Onion Routing prevents the transport medium from knowing who is communicating with whom -- the network knows only that communication is taking place. In addition, the content of the communication is hidden from eavesdroppers up to the point where the traffic leaves the OR network.

Paul Syverson and colleagues introduced the concept of onion of onion routing. That model uses a collection of forwarding hosts, each of whom knows only from where the communication was received and where to send it next.

Thus, to send untraceable data from *A* to *B*. A picks some number of forwarding hosts, call them *X*, *Y* and *Z*. *A* begins by encrypting the communication under B's public key. A then appends a header from *Z* to *B*, and encrypts the result under *Z's* public key. A then puts a header on that communication from *X* to *Y* and *Y* encrypts that under *X's* public key. Finally, *A* puts on a header to send the package to *X*.

Upon receiving the package, X decrypts it and finds instructions to forward the inner package to *Y*. *Y* then decrypts it and finds instructions to forward the inner package to *Z*. *Z* then decrypts it and finds instructions to forward the inner package to B. The package is deconstructed like *peeling the layers of an onion*, which is why this technique is called onion routing.

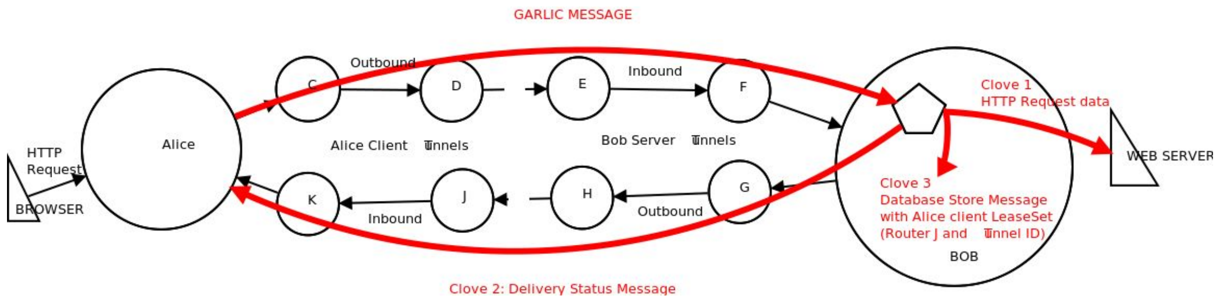


Garlic Routing

"Garlic routing" was first coined by *Michael J. Freedman* in Roger Dingledine's Free Haven Master's thesis Section 8.1.1 (June 2000), as derived from Onion Routing.

Michael Freedman defined "garlic routing" as an extension to onion routing, in which multiple messages are bundled together. He called each message a "bulb". All the messages, each with its own delivery instructions, are exposed at the endpoint. This allows the efficient bundling of an onion routing "reply block" with the original message.

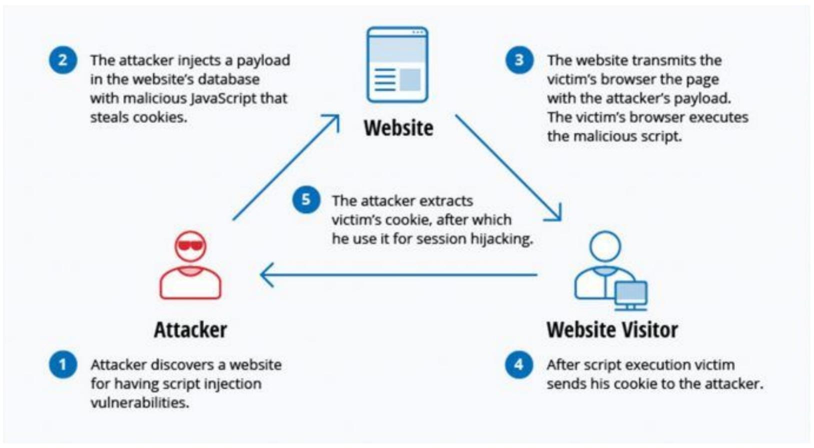
A significant difference from the method described by Freedman is that the path is unidirectional - there is no "turning point" as seen in onion routing or mixmaster reply blocks, which greatly simplifies the algorithm and allows for a more flexible and reliable delivery.



Q7. Explain the XSS & CSRF attack.

XSS Attack

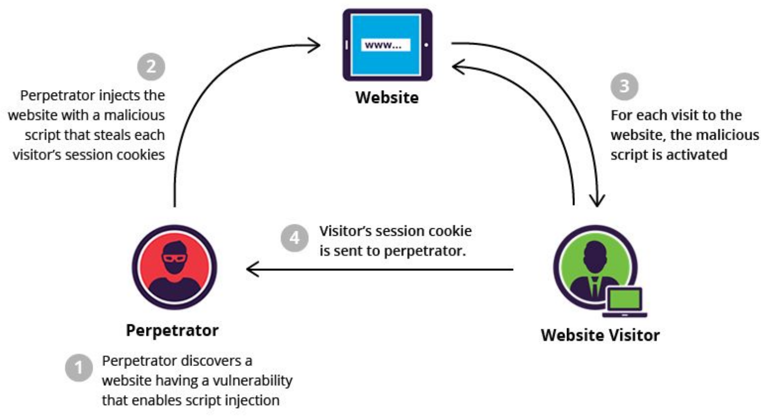
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.


2 The attacker injects a payload in the website's database with malicious JavaScript that steals cookies.
Website
3 The website transmits the victim's browser the page with the attacker's payload. The victim's browser executes the malicious script.
5 The attacker extracts victim's cookie, after which he use it for session hijacking.
Attacker
Website Visitor
1 Attacker discovers a website for having script injection vulnerabilities.
4 After script execution victim sends his cookie to the attacker.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

CSRF Attack

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.


www...
Website
2 Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3 For each visit to the website, the malicious script is activated
4 Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1 Perpetrator discovers a website having a vulnerability that enables script injection

With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

Q8. Discuss the threat-vulnerability control paradigm with the help of an example.

The goal of computer security is protecting valuable assets. To study different ways of protection, we use a framework that describes how assets may be harmed and how to counter or mitigate that harm.

Vulnerability

A vulnerability is a weakness in the system, for example, in procedures, design, or implementation, that might be exploited to cause loss or harm. For instance, a particu- lar system may be vulnerable to unauthorized data manipulation because the system does not verify a user’s identity before allowing data access.

Threat

A threat to a computing system is a set of circumstances that has the potential to cause loss or harm. There are many threats to a computer system, including human-initiated and computer-initiated ones. We have all experienced the results of inadvertent human errors, hardware design flaws, and software failures. But natural disasters are threats, too; they can bring a system down when the computer room is flooded or the data cen- ter collapses from an earthquake, for example.

The three aspects, availability, integrity, and confidentiality, make your computer valuable to you. But viewed from another perspective, they are three possible ways to make it less valuable, that is, to cause you harm. If someone steals your computer, scrambles data on your disk, or looks at your private data files, the value of your com- puter has been diminished or your computer use has been harmed. These characteris- tics are both basic security properties and the objects of security threats.

We can define these three properties as follows.

- **availability**: the ability of a system to ensure that an asset can be used by any authorized parties
- **integrity**: the ability of a system to ensure that an asset is modified only by authorized parties
- **confidentiality**: the ability of a system to ensure that an asset is viewed only by authorized parties

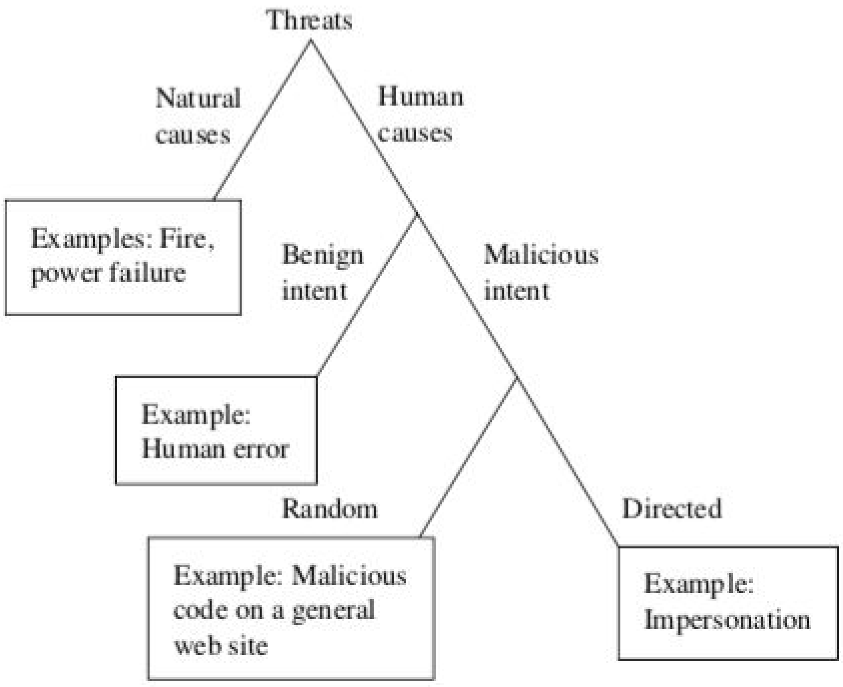


FIGURE 1-8 Kinds of Threats

One way to analyze harm is to consider the cause or source. We call a potential cause of harm a threat. Different kinds of threats are shown in the figure above. Harm can be caused by either nonhuman events or humans.

Harm

The negative consequence of an actualized threat is harm; we protect ourselves against threats in order to reduce or eliminate harm.

The value of many assets can change over time, so the degree of harm (and there- fore the severity of a threat) can change, too. With unlimited time, money, and capabil- ity, we might try to protect against all kinds of harm. But because our resources are limited, we must prioritize our protection, safeguarding only against serious threats and the ones we can control. Choosing the threats we try to mitigate involves a process called risk management, and it includes weighing the seriousness of a threat against our ability to protect.

Control

A control or countermeasure is a means to counter threats. Harm occurs when a threat is realized against a vulnerability. To protect against harm, then, we can neutral- ize the threat, close the vulnerability, or both. The possibility for harm to occur is called risk. We can deal with harm in several ways:

- **prevent** it, by blocking the attack or closing the vulnerability
- **deter** it, by making the attack harder but not impossible
- **deflect** it, by making another target more attractive (or this one less so)
- **mitigate** it, by making its impact less severe
- **detect** it, either as it happens or some time after the fact
- **recover** from its effects

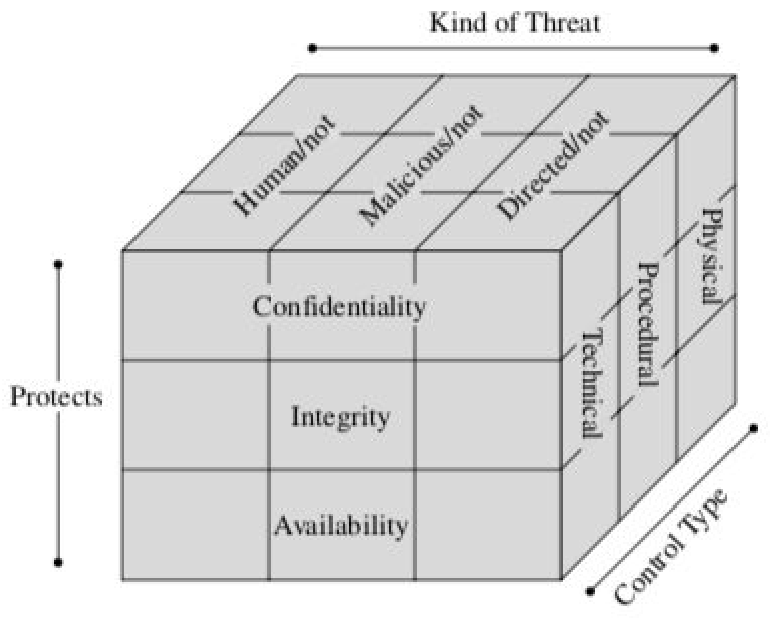


FIGURE 1-12 Types of Countermeasures

As shown in the figure above, you can think in terms of the property to be protected and the kind of threat when you are choosing appropriate types of countermeasures. None of these classes is necessarily better than or preferable to the others; they work in dif- ferent ways with different kinds of results. And it can be effective to use overlapping controls or defense in depth: more than one control or more than one class of control to achieve protection.

Example:: Online Payment Service

An online payment system is a way of making transactions or paying for goods and services through an electronic medium, without the use of checks or cash. It's also called an electronic payment system or e-payment system.

Threats

Following are some of the major threats to the system described above:

- Service not reachable/ available
- Invalid transactions performed between customers.
- Customer account hijacking.
- Invasion of privacy of customers.
- Breach of security protocols

Potential Attacks:

- Denial of Service (DoS) attack.
- Cross Site Request Forgery (CSRF) attack.
- Man In The Middle (MITM) attack.
- Replay Attack

To realize these sets of threats, the vendor should be equipped with the following tools:

- Intrusion Prevention System
- Application Firewall
- System Status Monitoring Tool

Harm

Following harm can be potentially inflicted on the system:

- Customers losing money and control to their account.
- Damages inflicted by downtime induced onto mission critical servers.
- Partial/ total loss of control over the system.

Any attack on the system is extremely damaging for the reputation of a real time online payment service, and may result in loss of trust or mass boycotting of that service by its customers.

Vulnerabilities

Following design/ implementation vulnerabilities can be exploited to perform above mentioned attacks:

- Untested APIs/ Interfaces.
- Incorrectly configured firewalls and/or credentials.
- Data synchronisation challenges across multiple distant servers.

Any of these vulnerabilities, if not mitigated before hand, can lead to attack on the system.

Control

Following measures should be taken to mitigate the vulnerabilities:

- Implement a tested and proven SDLC process that performs all the necessary checks before deploying updates/ changes to the system in the production environment.
- Perform regular in-house and/or out-sourced pen-testing of the service.
- Properly configure network and system credentials, users, configurations, etc.
- Place proper alert mechanisms and alert response teams to mitigate risks immediately.

These controls are partial in the sense that there is still a chance that the system might be exposed to an attack, however, with apt and immediate response, the vendor has a chance to stop the attack at its very beginning and have an upper hand in such attacks.