**CHAPTER – 1 : INTRODUCTION**

**Cyber Security**

Cyber security comprises technologies, processes and controls that are designed to protect systems, networks and data from cyber-attacks.

Effective cyber security reduces the risk of cyber-attacks, and protects organizations and individuals from the unauthorized exploitation of systems, networks and technologies.

### What are the consequences of a cyber-attack?

Cyber-attacks can disrupt and cause considerable financial and reputational damage to even the most resilient organization.

If you suffer a cyber-attack, you stand to lose assets, reputation and business, and potentially face regulatory fines and litigation – as well as the costs of remediation.

### The cyber threats organizations face!

Although larger organizations tend to have a realistic appreciation of the cyber threats they face, many small to medium-sized enterprises are unclear about the ways in which they’re vulnerable, and as many as 45% mistakenly think they’re not a viable target.

In fact, all Internet-facing organizations are at risk of attack. And it’s not a question of if you’ll be attacked, but when you’ll be attacked. The majority of cyber-attacks are automated and indiscriminate, exploiting known vulnerabilities rather than targeting specific organizations. Your organization could be being breached right now and you might not even be aware.

**1.1: Types of Cyber Security Attacks**

## **MALWARE**

If you've ever seen an antivirus alert pop up on your screen, or if you've mistakenly clicked a malicious email attachment, then you've had a close call with malware. Attackers love to use malware to gain a foothold in users' computers—and, consequently, the offices they work in—because it can be so effective.

## **Phishing**

In a phishing attack, an attacker may send you an email that appears to be from someone you trust, like your boss or a company you do business with. The email will seem legitimate, and it will have some urgency to it (e.g. fraudulent activity has been detected on your account). In the email, there will be an attachment to open or a link to click. Upon opening the malicious attachment, you’ll thereby install malware in your computer. If you click the link, it may send   
  
  
you to a legitimate-looking website that asks for you to log in to access an important file—except the website is actually a trap used to capture your credentials when you try to log in.

In order to combat phishing attempts, understanding the importance of verifying email senders and attachments/links is essential.

**SQL INJECTION ATTACK**

SQL (pronounced “sequel”) stands for structured query language; it’s a programming language used to communicate with databases. Many of the servers that store critical data for websites and services use SQL to manage the data in their databases. A SQL injection attack specifically targets this kind of server, using malicious code to get the server to divulge information it normally wouldn’t. This is especially problematic if the server stores private customer information from the website, such as credit card numbers, usernames and passwords (credentials), or other personally identifiable information, which are tempting and lucrative targets for an attacker.

**DENIAL OF SERVICE (DOS)**

Imagine you're sitting in traffic on a one-lane country road, with cars backed up as far as the eye can see. Normally this road never sees more than a car or two, but a county fair and a major sporting event have ended around the same time, and this road is the only way for visitors to leave town. The road can't handle the massive amount of traffic, and as a result it gets so backed up that pretty much no one can leave.

That's essentially what happens to a website during a [denial-of-service](https://www.us-cert.gov/ncas/tips/ST04-015) (DoS) attack. If you flood a website with more traffic than it was built to handle, you'll overload the website's server and it'll be nigh-impossible for the website to serve up its content to visitors who are trying to access it.

**SESSION HIJACKING AND MAN-IN-THE-MIDDLE ATTACKS**

When you're on the internet, your computer has a lot of small back-and-forth transactions with servers around the world letting them know who you are and requesting specific websites or services. In return, if everything goes as it should, the web servers should respond to your request by giving you the information you're accessing. This process, or session, happens whether you are simply browsing or when you are logging into a website with your username and password.

number of methods an attacker can use to steal the session ID, such as a cross-site scripting attack used to hijack session IDs.

**CHAPTER – 2 : PYTHON PROGRAMMING**

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source softwareand has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

**2.1: SOCKET PROGRAMMING**

**2.1.1: What is Sockets?**

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

|  |  |
| --- | --- |
| Sr.No. | Term & Description |
| 1 | Domain  The family of protocols that is used as the transport mechanism. These values are constants such as AF\_INET, PF\_INET, PF\_UNIX, PF\_X25, and so on. |
| 2 | type  The type of communications between the two endpoints, typically SOCK\_STREAM for connection-oriented protocols and SOCK\_DGRAM for connectionless protocols. |
| 3 | protocol  Typically zero, this may be used to identify a variant of a protocol within a domain and type. |
| 4 | hostname  The identifier of a network interface −  A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation  A string "<broadcast>", which specifies an INADDR\_BROADCAST address.  A zero-length string, which specifies INADDR\_ANY, or  An Integer, interpreted as a binary address in host byte order. |
| 5 | Port Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service. |

**Server Socket Methods:**

|  |  |
| --- | --- |
| Sr.No. | Method & Description |
| 1 | s.bind()  This method binds address (hostname, port number pair) to socket. |
| 2 | s.listen()  This method sets up and start TCP listener. |
| 3 | s.accept()  This passively accept TCP client connection, waiting until connection arrives (blocking). |

**Client Socket Methods:**

|  |  |
| --- | --- |
| Sr.No. | Method & Description |
| 1 | s.connect()  This method actively initiates TCP server connection. |

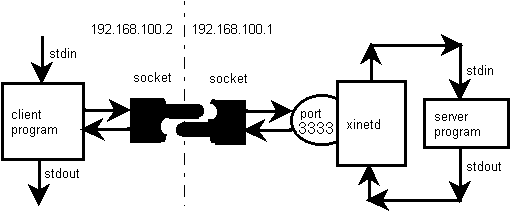
**General Socket Methods:**

|  |  |
| --- | --- |
| Sr.No. | Method & Description |
| 1 | s.recv()  This method receives TCP message |
| 2 | s.send()  This method transmits TCP message |
| 3 | s.recvfrom()  This method receives UDP message |
| 4 | s.sendto()  This method transmits UDP message |
| 5 | s.close()  This method closes socket |
| 6 | socket.gethostname()  Returns the hostname. |

**2.1.2: Python Internet modules:**

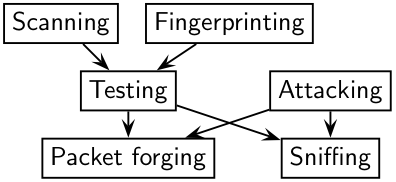
A list of some important modules in Python Network/Internet programming.

|  |  |  |  |
| --- | --- | --- | --- |
| Protocol | Common function | Port No | Python module |
| HTTP | Web pages | 80 | httplib, urllib, xmlrpclib |
| NNTP | Usenet news | 119 | nntplib |
| FTP | File transfers | 20 | ftplib, urllib |
| SMTP | Sending email | 25 | smtplib |
| POP3 | Fetching email | 110 | poplib |
| IMAP4 | Fetching email | 143 | imaplib |
| Telnet | Command lines | 23 | telnetlib |
| Gopher | Document transfers | 70 | gopherlib, urllib |

**  
  
  
2.2: SCAPY**

Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks.

In other words, Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark).

[](https://scapy.readthedocs.io/en/latest/_images/testing-taxonomy.png)

Scapy also performs very well on a lot of other specific tasks that most other tools can’t handle, like sending invalid frames, injecting your own 802.11 frames, combining techniques (VLAN hopping+ARP cache poisoning, VOIP decoding on WEP encrypted channel, …), etc.

The idea is simple. Scapy mainly does two things: sending packets and receiving answers. You define a set of packets, it sends them, receives answers, matches requests with answers and returns a list of packet couples (request, answer) and a list of unmatched packets. This has the big advantage over tools like Nmap or hping that an answer is not reduced to (open/closed/filtered), but is the whole packet.

On top of this can be build more high level functions, for example, one that does traceroutes and give as a result only the start TTL of the request and the source IP of the answer. One that pings a whole network and gives the list of machines answering. One that does a portscan and returns a LaTeX report.

**What makes Scapy so special?**

First, with most other networking tools, you won’t build something the author did not imagine. These tools have been built for a specific goal and can’t deviate much from it. For example, an ARP cache poisoning program won’t let you use double 802.1q encapsulation. Or try to find a program that can send, say, an ICMP packet with padding (I said padding, not payload, see?). In fact, each time you have a new need, you have to build a new tool.

Second, they usually confuse decoding and interpreting. Machines are good at decoding and can help human beings with that. Interpretation is reserved for human beings. Some programs try to mimic this behavior. For instance they say “this port is open” instead of “I received a SYN-ACK”. Sometimes they are right. Sometimes not. It’s easier for beginners, but when you know what you’re doing, you keep on trying to deduce what really happened from the program’s interpretation to make your own, which is hard because you lost a big amount of information. And you often end up using tcpdump -xX to decode and interpret what the tool missed.

Third, even programs which only decode do not give you all the information they received. The network’s vision they give you is the one their author thought was sufficient. But it is not complete, and you have a bias. For instance, do you know a tool that reports the Ethernet padding?

Scapy tries to overcome those problems. It enables you to build exactly the packets you want. Even if I think stacking a 802.1q layer on top of TCP has no sense, it may have some for somebody else working on some product I don’t know. Scapy has a flexible model that tries to avoid such arbitrary limits. You’re free to put any value you want in any field you want and stack them like you want. You’re an adult after all.

In fact, it’s like building a new tool each time, but instead of dealing with a hundred line C program, you only write 2 lines of Scapy.

After a probe (scan, traceroute, etc.) Scapy always gives you the full decoded packets from the probe, before any interpretation. That means that you can probe once and interpret many times, ask for a traceroute and look at the padding for instance.

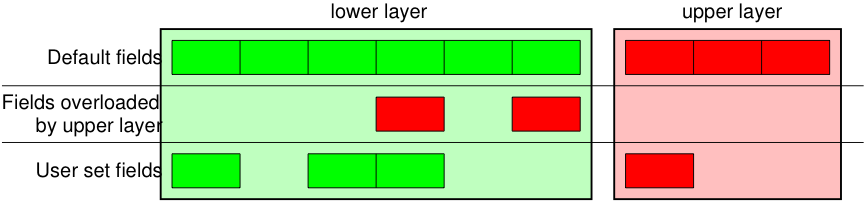
**2.2.1: Fast packet design:**

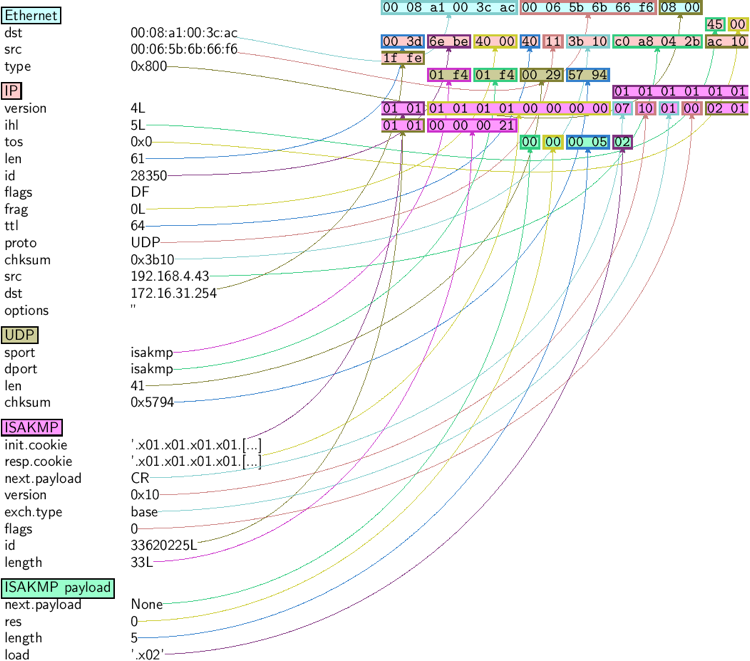
Other tools stick to the **program-that-you-run-from-a-shell** paradigm. The result is an awful syntax to describe a packet. For these tools, the solution adopted uses a higher but less powerful description, in the form of scenarios imagined by the tool’s author. As an example, only the IP address must be given to a port scanner to trigger the **port scanning** scenario. Even if the scenario is tweaked a bit, you still are stuck to a port scan.

Scapy’s paradigm is to propose a Domain Specific Language (DSL) that enables a powerful and fast description of any kind of packet. Using the Python syntax and a Python interpreter as the DSL syntax and interpreter has many advantages: there is no need to write a separate interpreter, users don’t need to learn yet another language and they benefit from a complete, concise and very powerful language.

Scapy enables the user to describe a packet or set of packets as layers that are stacked one upon another. Fields of each layer have useful default values that can be overloaded. Scapy does not oblige the user to use predetermined methods or templates. This alleviates the requirement of writing a new tool each time a different scenario is required. In C, it may take an average of 60 lines to describe a packet. With Scapy, the packets to be sent may be described in only a single line with another line to print the result. 90% of the network probing tools can be rewritten in 2 lines of Scapy.

Network discovery is blackbox testing. When probing a network, many stimuli are sent while only a few of them are answered. If the right stimuli are chosen, the desired information may be obtained by the responses or the lack of responses. Unlike many tools, Scapy gives all the information, i.e. all the stimuli sent and all the responses received. Examination of this data will give the user the desired information. When the dataset is small, the user can just dig for it. In other cases, the interpretation of the data will depend on the point of view taken. Most tools choose the viewpoint and discard all the data not related to that point of view. Because Scapy gives the complete raw data, that data may be used many times allowing the viewpoint to evolve during analysis. For example, a TCP port scan may be probed and the data visualized as the result of the port scan. The data could then also be visualized with respect to the TTL of response packet. A new probe need not be initiated to adjust the viewpoint of the data.

[](https://scapy.readthedocs.io/en/latest/_images/fieldsmanagement.png)



**Commands:**

| Command | Effect |
| --- | --- |
| raw(pkt) | assemble the packet |
| hexdump(pkt) | have a hexadecimal dump |
| ls(pkt) | have the list of fields values |
| pkt.summary() | for a one-line summary |
| pkt.show() | for a developed view of the packet |
| pkt.show2() | same as show but on the assembled packet (checksum is calculated, for instance) |
| pkt.sprintf() | fills a format string with fields values of the packet |
| pkt.decode\_payload\_as() | changes the way the payload is decoded |
| pkt.psdump() | draws a PostScript diagram with explained dissection |
| pkt.pdfdump() | draws a PDF with explained dissection |
| pkt.command() | return a Scapy command that can generate the packet |

Some operations (like building the string from a packet) can’t work on a set of packets. In these cases, if you forgot to unroll your set of packets, only the first element of the list you forgot to generate will be used to assemble the packet.

| Command | Effect |
| --- | --- |
| summary() | displays a list of summaries of each packet |
| nsummary() | same as previous, with the packet number |
| conversations() | displays a graph of conversations |
| show() | displays the preferred representation (usually nsummary()) |
| filter() | returns a packet list filtered with a lambda function |
| hexdump() | returns a hexdump of all packets |
| hexraw() | returns a hexdump of the Raw layer of all packets |
| padding() | returns a hexdump of packets with padding |
| nzpadding() | returns a hexdump of packets with non-zero padding |
| plot() | plots a lambda function applied to the packet list |
| make table() | displays a table according to a lambda function |

**2.2.2: Sending packets:**

Now that we know how to manipulate packets. Let’s see how to send them. The send() function will send packets at layer 3. That is to say, it will handle routing and layer 2 for you. The sendp() function will work at layer 2. It’s up to you to choose the right interface and the right link layer protocol. send() and sendp() will also return sent packet list if return\_packets=True is passed as parameter.

**2.2.3: Send and receive packets (sr):**

Now, let’s try to do some fun things. The sr() function is for sending packets and receiving answers. The function returns a couple of packet and answers, and the unanswered packets. The function sr1() is a variant that only returns one packet that answered the packet (or the packet set) sent. The packets must be layer 3 packets (IP, ARP, etc.). The function srp() do the same for layer 2 packets (Ethernet, 802.3, etc.). If there is, no response a None value will be assigned instead when the timeout is reached.

**CHAPTER – 3 : INFORMATION GATHERING (FOOT-PRINTING)**

**Footprinting** is an ethical hacking process of gathering information about the target and its environment.

This is a pre-attack stage and maximum efforts are deployed to ensure that the operations conducted are executed under stealth and target can’t trace back you. Footprinting is a first and the important step because after this a penetration tester knows how the hacker sees this network.

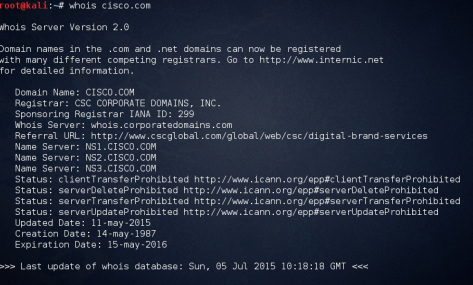
Good information gathering can make the difference between a successful pentest and one that has failed to provide maximum benefit to the client.

**It includes:**

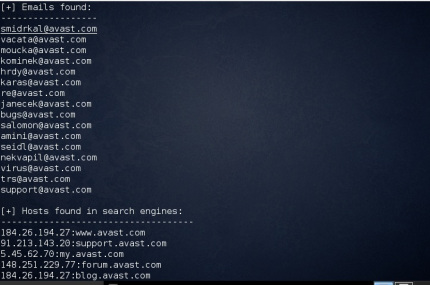
* Registration details of the website, contact details.
* Email harvesting,
* Finding out the target IP address and determine network range
* Identify active machine, DNS record, subdomains.
* Operating system fingerprinting.
* Finding login pages, sensitive directory
* Find out any known vulnerability for that particular version.

**3.1: WHOIS Database Lookup:**

WHOIS allows us to access information about the target including Registration Detail, IP address, contact information containing the address, Email ID, phone number. It also displays domain owner and domain registrar.



**3.2: Email Harvesting:**  
The **theharvester** tool available in Kali-Linux is an e-mail accounts, username, and hostname/ subdomains gathering tool.



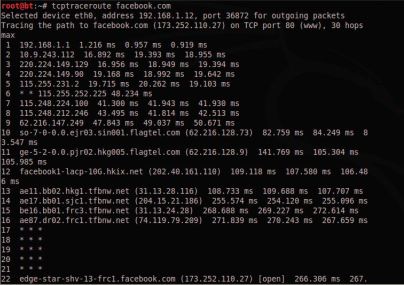
Email harvesting can be used by hackers to carry out a phishing campaign against an entire organization. This is one aspect of how emails can be misused. Computer users, who are often unaware of phishing attacks can fall victim and end up losing confidential information to the hackers.

**3.3: Search Engines Hacking:**

Marking a search query against your target in search engines (Google, Yahoo & Bing etc.) can also reveal great amount of information if used properly. Google Advance search or Google Hacking can help to locate more detailed information like company policies, employee’s details & online hidden pages etc. Google Hacking Database is a database of queries that identify sensitive information.

**3.4: Traceroute:**  
  
Traceroute is using UDP or ICMP ECHO to send out the packet with a Time To Live (TTL)  of one, and incrementing it until reaching the target, the tcptraceroute is using TCP SYN to send out the packet to the target.

tcptraceroute will receive a SYN/ACK packet if the port is open, and it will receive a RST packet if the port is closed.

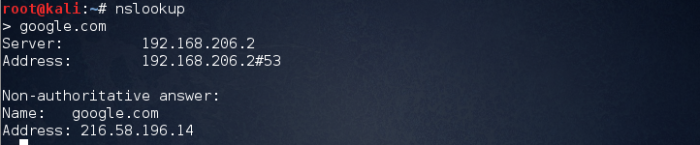


After route number 17, we are no longer able to get the route information. Usually this is because our traceroute is being blocked by a filtering device.

**3.5: DNS Reconnaissance:**

We can interact with a DNS server using various DNS clients such as host, nslookup, dig,etc.

nslookup is a computer program used in Windows and Unix to query Domain Name System(DNS) servers to find DNS details, including IP addresses of a particular computer, MX records for a domain and the NS servers of a domain. The name nslookup means “name server lookup”.



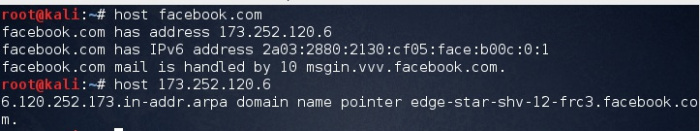
The above image explains that we connected to local server and asked to resolve a record for us. The server responded with the IP address of the victim.

– A – Points to host  IP address  
– MX – Points to domain mail server.  
– NS- Points to host name server  
– CNAME-Canonical naming allowing aliases to host.  
– SOA- Indicate authority for domain.  
– SRV-Service Record.  
– PTR-Maps IP address to hostname.  
– RP-Responsible Person.  
– INFO- Host Information.

* **Forward lookup bruteforce**
* **Reverse lookup bruteforce**
* **Verifying SPF Record**
* **Zone transfers**

**A. Forward lookup bruteforce**

The main idea behind this technique is to guess correct valid server names of organization. We can try this using the host command. The output gave us an IP address of the server.



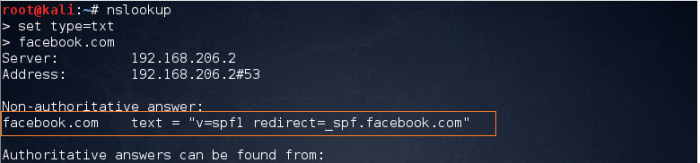
**B. Reverse lookup bruteforce**

This is a technique which is reverse to forward lookup bruteforce, in this case victim’s IP address is known and we need to find the server names and other information pertaining to the organization.

https://secur1tyadvisory.files.wordpress.com/2015/07/rv1.jpg?w=700

**C. Verifying SPF Record**

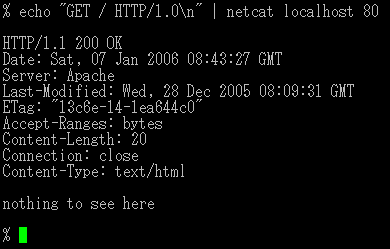
An SPF record is a TXT record that is part of a domain’s DNS zone file. The TXT record specifies a list of authorized host names/IP addresses that mail can originate from for a given domain name.



The purpose of an SPF record is to prevent spammers from sending messages with forged From addresses at your domain.

**3.6: NETCAT**

Netcat (often abbreviated to nc) is a computer networking utility for reading from and writing to network connections using TCP or UDP. Netcat is designed to be a dependable back-end that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and investigation tool, since it can produce almost any kind of connection its user could need and has a number of built-in capabilities.



**Opening a raw connection to port 25 (like SMTP)**

nc mail.server.net 25

**Performing an HTTP request**

printf "GET /hypertext/WWW/TheProject.html HTTP/1.0\r\nHost: info.cern.ch\r\n\r\n" | nc info.cern.ch 80

The full response (including HTTP headers) will be dumped to standard output.

**Setting up a one-shot webserver on port 8080 to present the content of a file**

{ printf 'HTTP/1.0 200 OK\r\nContent-Length: %d\r\n\r\n' "$(wc -c < some.file)"; cat some.file; } | nc -l 8080

The file can then be accessed via a web browser under http://servername:8080/. Netcat only serves the file once to the first client that connects and then exits; it also provides the content length for browsers that expect it. (This should work fine in a LAN, but may potentially fail with any kind of firewall between.). In some versions of netcat like netcat-traditional in Debian 8.6, you need to specify -p before the port number.

**Checking whether UDP ports (-u) 80–90 are open on 192.168.0.1 using zero mode I/O (-z)**

nc -vzu 192.168.0.1 80-90

**Note that UDP tests will always show as "open". The -z argument is useless.**

Test whether UDP port is open: simple UDP server and client

This test is useful, if you have shell access to the server that should be tested, but you do not know whether there is a firewall blocking a specific UDP port on the server.

**On the listening host, i.e. on the server whose port needs to be checked, do the following:**

nc -l -u -p 4172

**On the sending host, do the following – note that servname is the hostname of the listening host:**

nc -u Security\_Server\_IPaddress 4172

If text typed on the sending host (type something and hit enter) is displayed also on the listening host, then the UDP port 7000 is open. If it is not open, you will get an error such as "Connection refused".

There is a caveat. On some machines, IPv6 may be the default IP version to use by netcat. Thus, the host specified by the hostname is contacted using IPv6, and the user might not know about this. Ports may appear closed in the test, even though they would be open when using IPv4. This can be difficult to notice and may cause the false impression that the port is blocked, while it is actually open. You can force the use of IPv4 by using adding -4 to the options of the nc commands.

Pipe via UDP (-u) with a wait time (-w) of 1 second to "loggerhost" on port 514

echo '<0>message' | nc -w 1 -u loggerhost 514

**3.6.1: Port scanning:**

**An uncommon use of netcat is port scanning. Netcat is not considered the best tool for this job, but it can be sufficient (a more advanced tool is nmap)**

nc -v -n -z -w 1 192.168.1.2 1-1000

**The -n parameter here prevents DNS lookup, -z makes nc not receive any data from the server, and -w 1 makes the connection timeout after 1 second of inactivity.**

Proxying

**Another useful behaviour is using netcat as a proxy. Both ports and hosts can be redirected. Look at this example:**

nc -l 12345 | nc www.google.com 80

Port 12345 represents the request.

This starts a nc server on port 12345 and all the connections get redirected to google.com:80. If a web browser makes a request to nc, the request will be sent to google but the response will not be sent to the web browser. That is because pipes are unidirectional. This can be worked around with a named pipe to redirect the input and output.

mkfifo backpipe

nc -l 12345 0<backpipe | nc www.google.com 80 1>backpipe

**The -c option may also be used with the ncat implementation:**

ncat -l 12345 -c 'nc www.google.com 80'

**Using a named pipe is a more reliable method because using -c option provides only a one-shot proxy.**

Another useful feature is to proxy SSL connections. This way, the traffic can not be viewed in wire sniffing applications such as wireshark. This can be accomplished on UNIXes by utilizing mkfifo, netcat, and openssl.

mkfifo tmp

mkfifo tmp2

nc -l 8080 -k > tmp < tmp2 &

while true; do

openssl s\_client -connect www.google.com:443 -quiet < tmp > tmp2

done

**3.6.2: Making any process a server:**

netcat can be used to make any process a network server. It can listen on a port and pipe the input it receives to that process.

**The -e option spawns the executable with its input and output redirected via network socket.**

For example, it is possible to expose a bourne shell process to remote computers.

To do so, on a computer A with IP address 192.168.1.2, run this command:

$ nc -l -p 1234 -e /bin/sh

**Then, from any other computer on the same network, one could run this nc command:**

$ nc 192.168.1.2 1234

$ ls -las

total 4288

4 drwxr-xr-x 15 imsovain users 4096 2009-02-17 07:47 .

4 drwxr-xr-x 4 imsovain users 4096 2009-01-18 21:22 ..

8 -rw------- 1 imsovain users 8192 2009-02-16 19:30 .bash\_history

4 -rw-r--r-- 1 imsovain users 220 2009-01-18 21:04 .bash\_logout

...

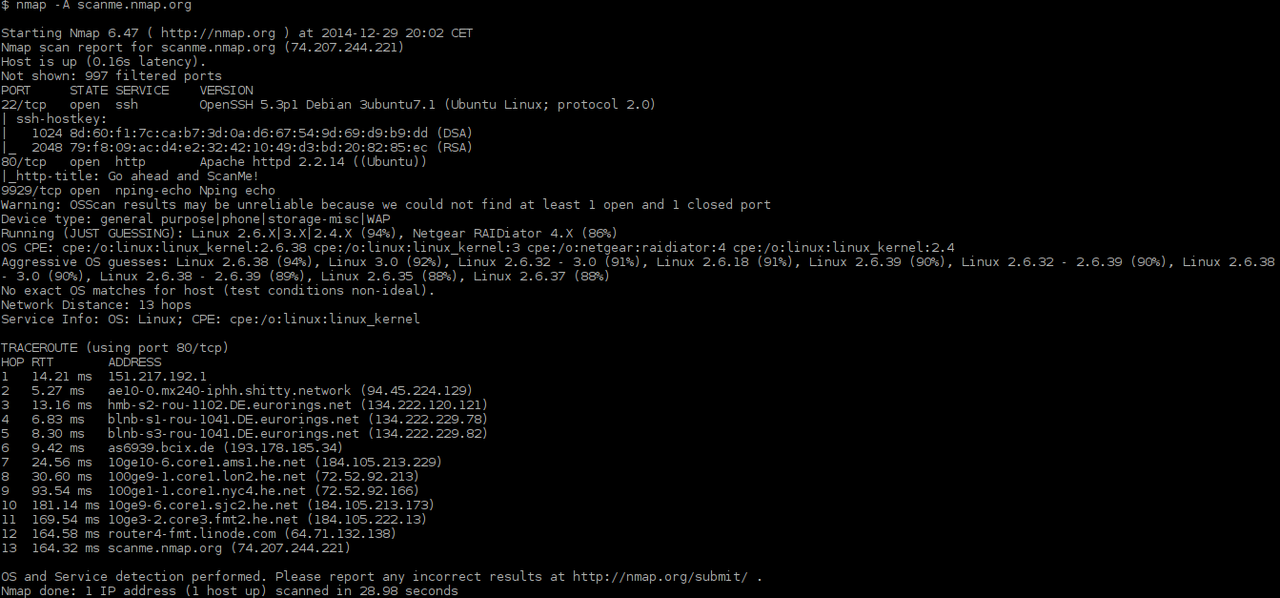
In this way, the -e option can be used to create a rudimentary backdoor. Some administrators perceive this as a risk and thus do not allow netcat on a computer.

**3.7: NMAP**

Nmap (Network Mapper) is a security scanner, originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich), used to discover hosts and services on a computer network, thus building a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host(s) and then analyzes the responses.

The software provides a number of features for probing computer networks, including host discovery and service and operating-system detection. These features are extensible by scripts that provide more advanced service detection, vulnerability detection, and other features. Nmap can adapt to network conditions including latency and congestion during a scan. The Nmap user community continues to develop and refine the tool.

Nmap started as a Linux-only utility, but porting to Windows, Solaris, HP-UX, BSD variants (including macOS), AmigaOS, and IRIX have followed. Linux is the most popular platform, followed closely by Windows.



**3.7.1: Features:**

**Nmap features include:**

* Host discovery – Identifying hosts on a network. For example, listing the hosts that respond to TCP and/or ICMP requests or have a particular port open.
* Port scanning – Enumerating the open ports on target hosts.
* Version detection – Interrogating network services on remote devices to determine application name and version number.
* OS detection – Determining the operating system and hardware characteristics of network devices.
* Scriptable interaction with the target – using Nmap Scripting Engine (NSE) and Lua programming language.

**Typical uses of Nmap:**

* Auditing the security of a device or firewall by identifying the network connections which can be made to, or through it.Identifying open ports on a target host in preparation for auditing.
* Network inventory, network mapping, maintenance and asset management.
* Auditing the security of a network by identifying new servers.
* Generating traffic to hosts on a network, response analysis and response time measurement.
* Finding and exploiting vulnerabilities in a network.

**Sample output :**

$ nmap -A scanme.nmap.org

Starting Nmap 6.47 ( https://nmap.org ) at 2014-12-29 20:02 CET

Nmap scan report for scanme.nmap.org (74.207.244.221)

Host is up (0.16s latency).

Not shown: 997 filtered ports

PORT STATE SERVICE VERSION

22/tcp open ssh OpenSSH 5.3p1 Debian 3ubuntu7.1 (Ubuntu Linux; protocol 2.0)

| ssh-hostkey:

| 1024 8d:60:f1:7c:ca:b7:3d:0a:d6:67:54:9d:69:d9:b9:dd (DSA)

|\_ 2048 79:f8:09:ac:d4:e2:32:42:10:49:d3:bd:20:82:85:ec (RSA)

80/tcp open http Apache httpd 2.2.14 ((Ubuntu))

|\_http-title: Go ahead and ScanMe!

9929/tcp open nping-echo Nping echo

Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port

Device type: general purpose|phone|storage-misc|WAP

Running (JUST GUESSING): Linux 2.6.X|3.X|2.4.X (94%), Netgear RAIDiator 4.X (86%)

OS CPE: cpe:/o:linux:linux\_kernel:2.6.38 cpe:/o:linux:linux\_kernel:3 cpe:/o:netgear:raidiator:4 cpe:/o:linux:linux\_kernel:2.4

Aggressive OS guesses: Linux 2.6.38 (94%), Linux 3.0 (92%), Linux 2.6.32 - 3.0 (91%), Linux 2.6.18 (91%), Linux 2.6.39 (90%), Linux 2.6.32 - 2.6.39 (90%), Linux 2.6.38 - 3.0 (90%), Linux 2.6.38 - 2.6.39 (89%), Linux 2.6.35 (88%), Linux 2.6.37 (88%)

No exact OS matches for host (test conditions non-ideal).

Network Distance: 13 hops

Service Info: OS: Linux; CPE: cpe:/o:linux:linux\_kernel

**3.8: TRACEROUTE (using port 80/tcp):**

HOP RTT ADDRESS

1 14.21 ms 151.217.192.1

2 5.27 ms ae10-0.mx240-iphh.shitty.network (94.45.224.129)

3 13.16 ms hmb-s2-rou-1102.DE.eurorings.net (134.222.120.121)

4 6.83 ms blnb-s1-rou-1041.DE.eurorings.net (134.222.229.78)

5 8.30 ms blnb-s3-rou-1041.DE.eurorings.net (134.222.229.82)

6 9.42 ms as6939.bcix.de (193.178.185.34)

7 24.56 ms 10ge10-6.core1.ams1.he.net (184.105.213.229)

8 30.60 ms 100ge9-1.core1.lon2.he.net (72.52.92.213)

9 93.54 ms 100ge1-1.core1.nyc4.he.net (72.52.92.166)

10 181.14 ms 10ge9-6.core1.sjc2.he.net (184.105.213.173)

11 169.54 ms 10ge3-2.core3.fmt2.he.net (184.105.222.13)

12 164.58 ms router4-fmt.linode.com (64.71.132.138)

13 164.32 ms scanme.nmap.org (74.207.244.221)

**CHAPTER – 4 : BIND/ REVERSE SHELL**

**4.1: What are shells?**

Shell can simply be described as a piece of code or program which can be used to gain code or command execution on a device (like servers, mobile phones, etc.).

**Types of shells**

* **Reverse shell**
* **Bind shell**

**4.1.1: Reverse shell:**

A reverse shell is a type of shell in which the target machine communicates back to the attacking machine. The attacking machine has a listener port on which it receives the connection, which by using, code or command execution is achieved.



Figure 1: Reverse TCP shell

**4.1.2: Bind shell:**

Bind shell is a type of shell in which the target machine opens up a communication port or a listener on the victim machine and waits for an incoming connection. The attacker then connects to the victim machine’s listener which then leads to code or command execution on the server.

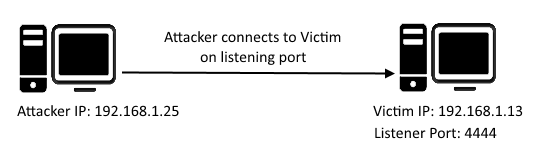
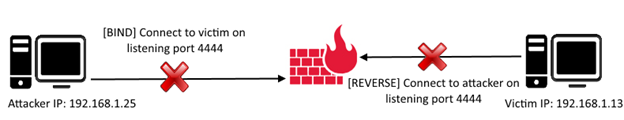


Figure 2: Bind TCP shell

There are a number of popular shell files. To name a few: Reverse TCP Meterpreter, C99 PHP web shell, JSP web shell, Netcat, etc. One thing which is common between all these shells is that they all communicate over a TCP protocol.

Imagine a scenario in which communication to and from the server is protected and filtered by a firewall and does not allow TCP shell communication to take place on any listening port (both reverse and bind TCP connection).



But many environments allow ping requests to be sent and received. Ping requests work on the ICMP protocol.

ICMP stands for Internet Control Message Protocol; it is used by network devices’ query and error messages. ICMP differs from the widely used TCP and UDP protocols because ICMP is not used for transferring data between network devices.

When a device wants to test connectivity to another device, it uses the PING tool (ICMP communication) to send an ECHO REQUEST and waits for an ECHO RESPONSE. Images below show the PING echo request-response communication taking place between two network devices.

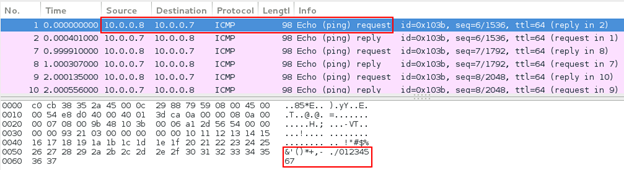


Figure 4: Ping echo request

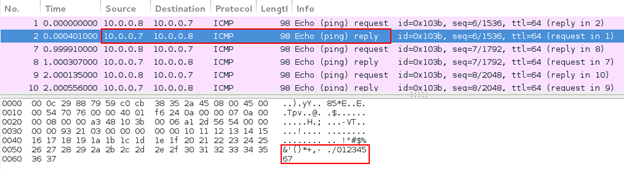


Figure 5: Ping echo response

Looking at the ping echo request and response, we can see that the ping echo request ICMP packet sent by network device A (10.0.0.7) contains 48 bytes of data. Network device B (10.0.0.8) replies with a ping echo response with the same 48 bytes of data. See the image below:

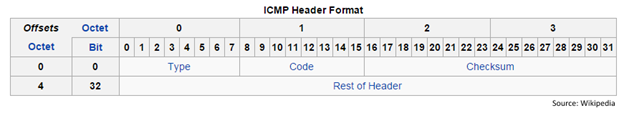
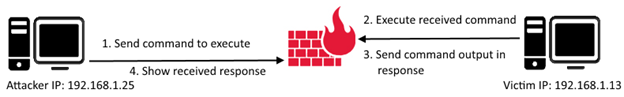


Figure 6: ICMP header format

As you can see, the packet does not contain source and destination port numbers like TCP and UDP header formats. Hence, echo request-response communication is taking place between the network devices, but not over specific port(s).

The above discussion laid down little idea that ICMP communication can be used to contact between two devices using a custom agent running on victim and attacking devices.

The client ICMP agent listens for ICMP packets from a specific host and uses the data in the packet for command execution.

The server ICMP Agent sends ICMP packets to connect to the victim running a custom ICMP agent and sends it commands to execute.   
  
  


**CHAPTER – 5 : DoS (Denial of Service) ATTACK**

In computing, a denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled.

In a distributed denial-of-service attack (DDoS attack), the incoming traffic flooding the victim originates from many different sources. This effectively makes it impossible to stop the attack simply by blocking a single source.

A DoS or DDoS attack is analogous to a group of people crowding the entry door of a shop, making it hard for legitimate customers to enter, disrupting trade.

Criminal perpetrators of DoS attacks often target sites or services hosted on high-profile web servers such as banks or credit card payment gateways. Revenge, blackmailand activism can motivate these attacks.

**Types:**

Denial-of-service attacks are characterized by an explicit attempt by attackers to prevent legitimate use of a service. There are two general forms of DoS attacks: those that crash services and those that flood services. The most serious attacks are distributed.

**4.1: Distributed DoS:**

A distributed denial-of-service (DDoS) is a DoS attack where the perpetrator uses more than one unique IP address, often thousands of them. Since the incoming traffic flooding the victim originates from many different sources, it is impossible to stop the attack simply by using ingress filtering. It also makes it very difficult to distinguish legitimate user traffic from attack traffic when spread across so many points of origin. As an alternative or augmentation of a DDoS, attacks may involve forging of IP sender addresses (IP address spoofing) further complicating identifying and defeating the attack.

The scale of DDoS attacks has continued to rise over recent years, by 2016 exceeding a terabit per second.

**5.2: Application layer attacks:**

An application layer DDoS attack (sometimes referred to as layer 7 DDoS attack) is a form of DDoS attack where attackers target the application layer of the OSI model.The attack over-exercises specific functions or features of a website with the intention to disable those functions or features. This application-layer attack is different from an entire network attack, and is often used against financial institutions to distract IT and security personnel from security breaches. As of 2013, application layer DDoS attacks represent 20% of all DDoS attacks. According to research by the company Akamai, there have been "51 percent more application layer attacks" from Q4 2013 to Q4 2014 and "16 percent more" from Q3 2014 over Q4 2014. In November 2017; Junade Ali, a Computer Scientist at Cloudflare noted that whilst network-level attacks continue to be of high capacity, they are occurring less frequently.

**5.3: OSI model:**

The Open Systems Interconnection (OSI) model (ISO/IEC 7498-1) is a conceptual model that characterizes and standardizes the internal functions of a communication system by partitioning it into abstraction layers. The model is a product of the Open Systems Interconnection project at the International Organization for Standardization (ISO). The model groups similar communication functions into one of seven logical layers. A layer serves the layer above it and is served by the layer below it. For example, a layer that provides error-free communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that make up the contents of that path. Two instances at one layer are connected by a horizontal connection on that layer.

Main article: Application layer

In the OSI model, the definition of its application layer is narrower in scope than is often implemented. The OSI model defines the application layer as being the user interface. The OSI application layer is responsible for displaying data and images to the user in a human-recognizable format and to interface with the presentation layer below it. In an implementation, the application and presentation layers are frequently combined.

**5.4: Method of attack:**

An application layer DDoS attack is done mainly for specific targeted purposes, including disrupting transactions and access to databases. It requires fewer resources than network layer attacks but often accompanies them. An attack is disguised to look like legitimate traffic, except it targets specific application packets. The attack on the application layer can disrupt services such as the retrieval of information or search function as well as web browser function, email services and photo applications. To be deemed a distributed denial of service attack, more than around 3–5 nodes on different networks should be used; using fewer nodes qualifies as a DoS attack but not a DDoS attack.

**5.5: Advanced persistent DoS:**

An advanced persistent DoS (APDoS) is more likely to be perpetrated by an advanced persistent threat (APT): attackers who are well-resourced, exceptionally skilled and have access to substantial commercial grade computer resources and capacity. APDoS attacks represent a clear and emerging threat needing specialised monitoring and incident response services and the defensive capabilities of specialised DDoS mitigation service providers.

This type of attack involves massive network layer DDoS attacks through to focused application layer (HTTP) floods, followed by repeated (at varying intervals) SQLi and XSS attacks. Typically, the perpetrators can simultaneously use from 2 to 5 attack vectors involving up to several tens of millions of requests per second, often accompanied by large SYN floods that can not only attack the victim but also any service provider implementing any sort of managed DDoS mitigation capability. These attacks can persist for several weeks. The longest continuous period noted so far lasted 38 days. This attack involved approximately 50+ petabits (50,000+ terabits) of malicious traffic.

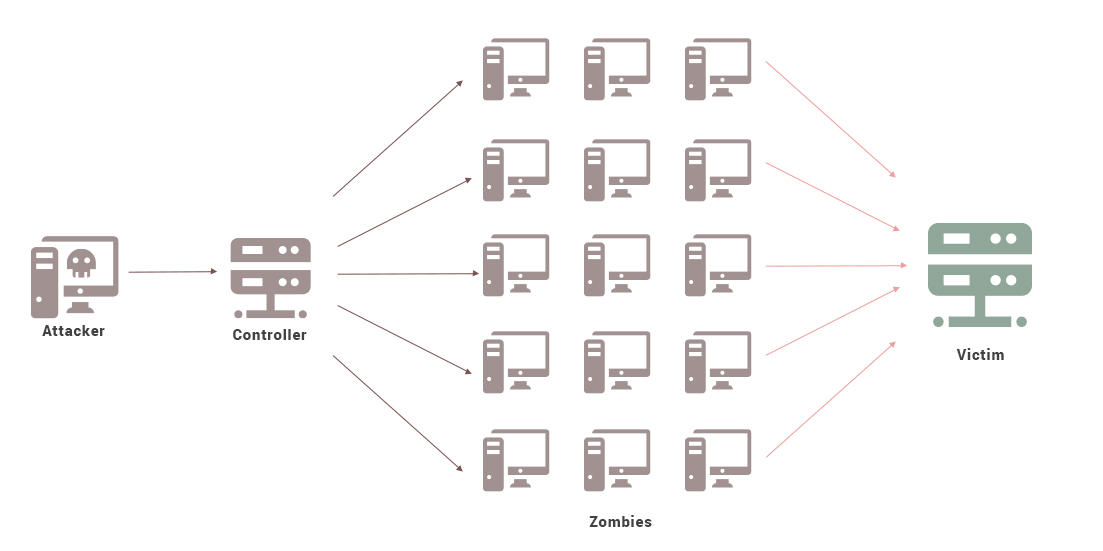
Attackers in this scenario may tactically switch between several targets to create a diversion to evade defensive DDoS countermeasures but all the while eventually concentrating the main thrust of the attack onto a single victim. In this scenario, attackers with continuous access to several very powerful network resources are capable of sustaining a prolonged campaign generating enormous levels of un-amplified DDoS traffic.

**5.6: APDoS attacks are characterised by:**

* Advanced reconnaissance (pre-attack OSINT and extensive decoyed scanning crafted to evade detection over long periods)   
  Tactical execution (attack with both primary and secondary victims but focus is on primary)
* Explicit motivation (a calculated end game/goal target)
* Large computing capacity (access to substantial computer power and network bandwidth)
* Simultaneous multi-threaded OSI layer attacks (sophisticated tools operating at layers 3 through 7)
* Persistence over extended periods (combining all the above into a concerted, well managed attack across a range of targets).

**5.7: Denial-of-service as a service:**

Some vendors provide so-called "booter" or "stresser" services, which have simple web-based front ends, and accept payment over the web. Marketed and promoted as stress-testing tools, they can be used to perform unauthorized denial-of-service attacks, and allow technically unsophisticated attackers access to sophisticated attack tools without the need for the attacker to understand their use. Usually powered by a botnet, the traffic produced by a consumer stresser can range anywhere from 5-50 Gbit/s, which can, in most cases, deny the average home user internet access.

****

**CHAPTER – 6 : ARP CACHE POISONING**

In computer networking, ARP spoofing, ARP cache poisoning, or ARP poison routing, is a technique by which an attacker sends (spoofed) Address Resolution Protocol (ARP) messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead.

ARP spoofing may allow an attacker to intercept data frames on a network, modify the traffic, or stop all traffic. Often the attack is used as an opening for other attacks, such as denial of service, man in the middle, or session hijacking attacks.

The attack can only be used on networks that use ARP, and is confined to and requires the attacker to gain direct access to the local network segment to be attacked.

**6.1: ARP vulnerabilities:**

* The Address Resolution Protocol (ARP) is a widely used communications protocol for resolving Internet layer addresses into link layer addresses.
* When an Internet Protocol (IP) datagram is sent from one host to another in a local area network, the destination IP address must be resolved to a MAC address for transmission via the data link layer. When another host's IP address is known, and its MAC address is needed, a broadcast packet is sent out on the local network. This packet is known as an ARP request. The destination machine with the IP in the ARP request then responds with an ARP reply that contains the MAC address for that IP.
* ARP is a stateless protocol. Network hosts will automatically cache any ARP replies they receive, regardless of whether network hosts requested them. Even ARP entries that have not yet expired will be overwritten when a new ARP reply packet is received. There is no method in the ARP protocol by which a host can authenticate the peer from which the packet originated. This behavior is the vulnerability that allows ARP spoofing to occur.
* Anatomy of an ARP spoofing attack
* The basic principle behind ARP spoofing is to exploit the lack of authentication in the ARP protocol by sending spoofed ARP messages onto the LAN. ARP spoofing attacks can be run from a compromised host on the LAN, or from an attacker's machine that is connected directly to the target LAN.
* Generally, the goal of the attack is to associate the attacker's host MAC address with the IP address of a target host, so that any traffic meant for the target host will be sent to the attacker's host. The attacker may choose to inspect the packets (spying), while forwarding the traffic to the actual default destination to avoid discovery, modify the data before forwarding it (man-in-the-middle attack), or launch a denial-of-service attack by causing some or all of the packets on the network to be dropped.

**6.2: Defences:**

**6.2.1: Static ARP entries:**

The simplest form of certification is the use of static, read-only entries for critical services in the ARP cache of a host. This prevents only simple attacks and does not scale on a large network, since the mapping has to be set for each pair of machines resulting in n2-n ARP entries that have to be configured when n machines are present: On every machine there must be an ARP entry for every

other machine on the network, which are n-1 ARP entries on every of the n machines.

IP address-to-MAC address mappings in the local ARP cache may be statically entered so that hosts ignore all ARP reply packets. While static entries provide some security against spoofing if the operating system handles them correctly, they result in maintenance efforts as address mappings of all systems in the network have to be distributed.

**6.2.2: ARP spoofing detection and prevention software:**

Software that detects ARP spoofing generally relies on some form of certification or cross-checking of ARP responses. Uncertified ARP responses are then blocked. These techniques may be integrated with the DHCP server so that both dynamic and static IP addresses are certified. This capability may be implemented in individual hosts or may be integrated into Ethernet switches or other network equipment. The existence of multiple IP addresses associated with a single MAC address may indicate an ARP spoof attack, although there are legitimate uses of such a configuration. In a more passive approach a device listens for ARP replies on a network, and sends a notification via email when an ARP entry changes.

AntiARP also provides Windows-based spoofing prevention at the kernel level. ArpStar is a Linux module for kernel 2.6 and Linksys routers that drops invalid packets that violate mapping, and contains an option to repoison/heal.

**6.2.3: OS security:**

Operating systems react differently. Linux ignores unsolicited replies, but, on the other hand, uses responses to requests from other machines to update its cache. Solaris accepts updates on entries only after a timeout. In Microsoft Windows, the behaviour of the ARP cache can be configured through several registry entries under HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters, ArpCacheLife, ArpCacheMinReferenceLife, ArpUseEtherSNAP, ArpTRSingleRoute, ArpAlwaysSourceRoute, ArpRetryCount.

**6.2.4: Legitimate usage:**

The techniques that are used in ARP spoofing can also be used to implement redundancy of network services. For example, some software allows a backup server to issue a gratuitous ARP request in order to take over for a defective server and transparently offer redundancy.  There are two companies known to-date that have tried to commercialize products centered around this strategy, Disney Circle and CUJO. The latter has recently run into significant problems with its ARP-spoofing strategy in consumer's homes; they have now completely removed that capability and replaced it with a DHCP-based strategy.

ARP spoofing is often used by developers to debug IP traffic between two hosts when a switch is in use: if host A and host B are communicating through an Ethernet switch, their traffic would normally be invisible to a third monitoring host M. The developer configures A to have M's MAC address for B, and B to have M's MAC address for A; and also configures M to forward packets. M can now monitor the traffic, exactly as in a man-in-the-middle attack.

**6.2.5: Spoofing:**

**Some of the tools that can be used to carry out ARP spoofing attacks:**

Arpspoof (part of the DSniff suite of tools)

Arpoison

Subterfuge

Ettercap

Seringe

ARP-FILLUP -V0.1

arp-sk -v0.0.15

ARPOc -v1.13

arpalert -v0.3.2

arping -v2.04

arpmitm -v0.2

arpoison -v0.5  
ArpSpyX -v1.1

ArpToXin -v 1.0

Cain And Abel -v 4.3

cSploit -v 1.6.2

SwitchSniffer

APE – ARP Poisoning Engine

Simsang

zANTI -v2

NetSec Framework -v1

Minary

**6.3: ARP Cache Poisoning Concept:**

ARP protocol was designed to be simple and efficient but a major flaw in the protocol is lack of authentication. No authentication was added to its implementation and as a result, there is no way to authenticate the IP to MAC address mapping in the ARP reply. Further, the host does not even check whether it sent an ARP request for which it is receiving ARP reply message.

In a layman’s language, if computer ‘A’ has sent and ARP request and it gets an ARP reply, then ARP protocol by no means can check whether the information or the IP to MAC mapping in the ARP reply is correct or not. Also, even if a host did not send an ARP request and gets an ARP reply, then also it trusts the information in reply and updates its ARP cache. This is known as ARP cache poisoning.

So you can see that its easy to exploit this weakness of ARP protocol. An evil hacker can craft a valid ARP reply in which any IP is mapped to any MAC address of the hackers choice and can send this message to the complete network. All the devices on network will accept this message and will update their ARP table with new Information and this way the hacker can gain control of the to and fro communication from any host in network.

**6.4: ARP Cache Poisoning Consequences:**

After a hacker sees a possibility of ARP cache poisoning, the attacker can use various attack techniques to harm or to gain control of the victims machine.  
  
Let’s discuss some of them here:

**A) Denial of service**

A hacker can send an ARP reply mapping an IP address on network with a wrong or non-existent MAC address. For example, a fake ARP reply mapping the network’s router IP with a non-existent MAC will bring down the connectivity of the whole network with the outer world as now any packet sent to IP of router will be sent to a machine with a MAC address that does not exist.

**B) Man in Middle**

As the name suggest, the hacker can make his machine sit right in between of the communication between your system and any other system on network. This way the hacker can sniff all the traffic to and from both the machines.

To achieve this suppose your machine is host ‘A’ and your network router is host ‘B’. ‘A’ has IP-A and MAC-A, while ‘B’ has IP-B and MAC-B as IP address and MAC address respectively. Now, the hacker sends an ARP reply to the router mapping your IP (IP-A) with his machine’s MAC address and another ARP reply to your machine mapping routers IP with his machine’s MAC address. Now any message sent by your machine to router or from router to your machine will reach the hacker’s machine. The hacker can now switch on the ‘IP forwarding’ feature on his machine which lets the hacker’s machine to forward all the traffic to and fro to your machine and router. This way the hacker’s machine sits right in the middle and can sniff or block the traffic.

**C) MAC Flooding**

For switches on network, MAC flooding is an ARP cache poising technique that is used. Many network switches when overloaded can start acting like a hub and start broadcasting all the network traffic to all the hosts connected to network. So a hacker can flood a switch with fake ARP replies and can make the switch to start behaving like a hub. In this role, the switch does not enable its ‘port security’ feature due to which it broadcast all the network traffic and taking advantage of this, the hacker can packet sniff the network.

ARP Cache Poisoning Mitigation Techniques

Poisoning ARP cache remotely is bit difficult as it requires either physical access to the network or control of one of the machines in the network. Since its not always easy so ARP attacks are not frequently heard. Anyways, taking precautions is better than taking medicines. Network administrators should take care that these type of attacks do not take place. Here are a few mitigation points:

For small networks, static ARP entries can be maintained. Static means unchanging, so as the name suggests these entries cannot be changed and thus any tries by hackers to change the mapping fails. This is good for small networks but not for big networks as mapping for every new device added to network needs to be done manually.

For a large network, the port security features of network switches can be explored. Some features when turned on force the switch to allow only one MAC address for each physical port on switch. This feature makes sure that machines cannot change their MAC address and cannot map more than one MAC to their machine hence preventing attacks like ‘man in middle’.

In general, some monitoring tool like ARPwatch can be deployed to get alerts when some malicious ARP activity takes place on your network.

To conclude, in this article, we studied the basics of ARP protocol, its loopholes, how these loopholes can be exploited and how they can be mitigated.

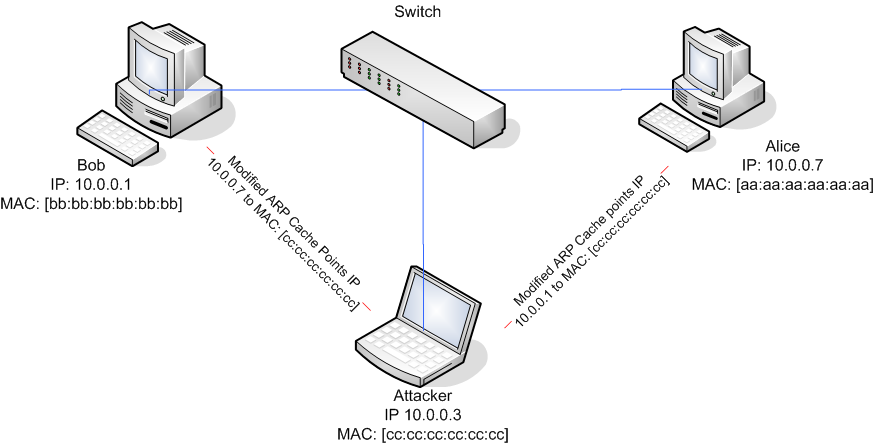
**6.5: ARP Cache Poisoning with Scapy:**

Okay, so now things are getting interesting with the ARP poison attack. I have always read about the subject but ever had the chance to experiment with it. It is a type of attack in which a malicious actor sends falsified **ARP**(Address Resolution Protocol) messages over a local area network. This results in the linking of an attacker’s MAC address with the IP address of a legitimate computer or server on the network. We can fool our target into thinking that it is sending traffic to its intended destination. The ARP table can be spoofed and we can insert ourselves in the middle.

Each machine maintains an ARP table, which is a mapping of MAC addresses to IP Addresses. For a machine to send data to another IP address, it would have to lookup this table to determine the MAC address associated to that IP Address.   
  
**Here is the ARP table on my machine:**

arp -a

(10.0.0.1) at 0:8:a2:c:9e:8 on en5 ifscope [ethernet]  
? (10.0.0.3) at 0:c:29:d4:aa:95 on en5 ifscope [ethernet]  
? (10.0.0.20) at 0:e:58:52:c5:8 on en5 ifscope [ethernet]  
? (10.0.0.24) at 0:e:58:98:10:c8 on en5 ifscope [ethernet]  
? (10.0.0.25) at 0:e:58:73:44:fe on en5 ifscope [ethernet]  
? (10.0.0.26) at 0:e:58:53:1:4a on en5 ifscope [ethernet]  
? (10.0.0.28) at 0:e:58:53:c8:38 on en5 ifscope [ethernet]  
? (10.0.0.36) at c8:69:cd:64:14:f7 on en5 ifscope [ethernet]  
? (10.0.0.244) at 0:17:88:41:f0:10 on en5 ifscope [ethernet]  
? (224.0.0.251) at 1:0:5e:0:0:fb on en5 ifscope permanent [ethernet]

****

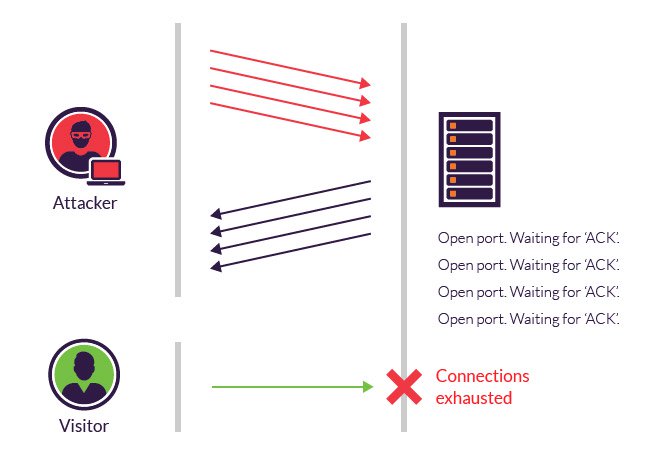
**CHAPTER – 7 : TCP SYN FLOOD**

TCP SYN flood (a.k.a. SYN flood) is a type of Distributed Denial of Service (DDoS) attack that exploits part of the normal TCP three-way handshake to consume resources on the targeted server and render it unresponsive.

Essentially, with SYN flood DDoS, the offender sends TCP connection requests faster than the targeted machine can process them, causing network saturation.

**7.1: ATTACK DESCRIPTION:**

* When a client and server establish a normal TCP “three-way handshake,” the exchange looks like this:
* Client requests connection by sending SYN (synchronize) message to the server.
* Server acknowledges by sending SYN-ACK (synchronize-acknowledge) message back to the client.
* Client responds with an ACK (acknowledge) message, and the connection is established.
* In a SYN flood attack, the attacker sends repeated SYN packets to every port on the targeted server, often using a fake IP address. The server, unaware of the attack, receives multiple, apparently legitimate requests to establish communication. It responds to each attempt with a SYN-ACK packet from each open port.
* The malicious client either does not send the expected ACK, or—if the IP address is spoofed—never receives the SYN-ACK in the first place. Either way, the server under attack will wait for acknowledgement of its SYN-ACK packet for some time.



During this time, the server cannot close down the connection by sending an RST packet, and the connection stays open. Before the connection can time out, another SYN packet will arrive. This leaves an increasingly large number of connections half-open – and indeed SYN flood attacks are also referred to as “half-open” attacks. Eventually, as the server’s connection overflow tables fill, service to legitimate clients will be denied, and the server may even malfunction or crash.

While the "classic" SYN flood described above tries to exhaust network ports, SYN packets can also be used in DDoS attacks that try to clog your pipes with fake packets to achieve network saturation. The type of packet is not important. Still, SYN packets are often used because they are the least likely to be rejected by default.  
  
 **7.2: METHODS OF MITIGATION:**

While modern operating systems are better equipped to manage resources, which makes it more difficult to overflow connection tables, servers are still vulnerable to SYN flood attacks.

**There are a number of common techniques to mitigate SYN flood attacks, including:**

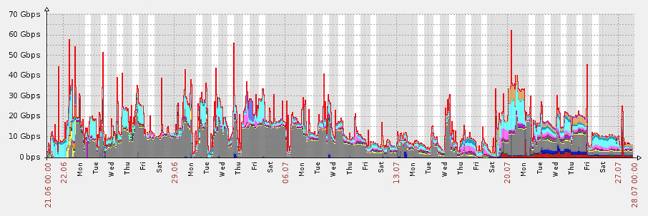
**Micro blocks**—administrators can allocate a micro-record (as few as 16 bytes) in the server memory for each incoming SYN request instead of a complete connection object.

**SYN cookies**—using cryptographic hashing, the server sends its SYN-ACK response with a sequence number (seqno) that is constructed from the client IP address, port number, and possibly other unique identifying information. When the client responds, this hash is included in the ACK packet. The server verifies the ACK, and only then allocates memory for the connection.

**RST cookies**—for the first request from a given client, the server intentionally sends an invalid SYN-ACK. This should result in the client generating an RST packet, which tells the server something is wrong. If this is received, the server knows the request is legitimate, logs the client, and accepts subsequent incoming connections from it.

**Stack tweaking**—administrators can tweak TCP stacks to mitigate the effect of SYN floods. This can either involve reducing the timeout until a stack frees memory allocated to a connection, or selectively dropping incoming connections.

Obviously, all of the above mentioned methods rely on the target network's ability to handle large-scale volumetric DDoS attacks, with traffic volumes measured in tens of Gigabits (and even hundreds of Gigabits) per second.

  
  
  
Incapsula mitigates a 38 day-long SYN flood and DNS flood multi-vector DDoS attack.

Incapsula DDoS protection leverages Anycast technology to balance the incoming DDoS requests across its global network of high-powered scrubbing centers. With the combined capacity of its global network, Incapsula can cost-effectively exceed attacker resources, rendering the DDoS attack ineffective. The service is build to scale on demand, offering ample resources to deal with even the largest of volumetric DDoS attacks.

To assure business continuity, Incapsula filtering algorithm continuously analyzes incoming SYN requests, using SYN cookies to selectively allocate resources to legitimate visitors. This enables transparent DDoS mitigation, wtih no downtime, latency of any other business disruptions.  
  
**7.3: What is a SYN flood attack?**  
The SYN flood attack works by the attacker opening multiple "half made" connections and not responding to any SYN\_ACK packets.   
In order to understand the SYN flood attack it is vital to understand the TCP 3-way handshake first.

**7.3.1: TCP handshake:**

When a client wants to talk to a server over TCP, the client initiates what is called the 3-way handshake. It begins with the client sending a SYN packet to the server, the server receives the packet and responds with a SYN\_ACK indicating to the client that it received the initial SYN packet. When the client receives the SYN\_ACK it will reply with an ACK packet which now establishes a connection between the client and server and they can begin exchanging data. A visual representation can be seen below.

**7.3.2: The SYN flood attack:**

We now know that clients and servers establishes a connection by completing a handshake with each other, what happens if you do not complete the handshake? By sending multiple SYN packets to the victim and not responding with an ACK message to the victim's SYN\_ACK message, the victim will have a multiple "half" open connections causing the victim's connection table to potentially overflow.

**7.3.3: SYN spoofing:**

An attacker could also spoof the source address of the initial SYN packets causing the victim to send SYN\_ACK messages to another host, which will respond with a RSTpacket indicating to drop the connection causing the SYN attack to fail. In order for the spoofing to work the attacker needs to select source addresses where there exists no hosts that can respond.

**7.4: Protection:**

**There are different ways to limit the effects of SYN flood/spoof attacks, some of are:**

**6.4.1: Remove random connections:**

Start deleting random "half made" connections, this can however delete legitimate connections.

Reduce time in SYN\_RECEIVED state

When you send SYN packet to a server the connection will be placed in a SYN\_RECVstate and the server will respond with a SYN\_ACK packet. The server will resend SYN\_ACKa few times until an ACK is received, during this time the connection is still placed in a SYN\_RECV state. This means the server needs to keep track of thousands of connections which can overflow the server's connection table. By reducing the SYN\_ACK retries the server will close connections placed in a SYN\_RECV state earlier which can be very helpful protecting against SYN flood attacks.

In Linux you can edit **/etc/sysctl.conf** and change net.ipv4.tcp\_synack\_retries from the default value 5 to something lower depending on fast you want your system to close connections in a SYN\_RECV state. The default value 5 leaves connections in SYN\_RECV state open for 3 minutes and a value of 3 leaves connections open for roughly 45 seconds. After you have modified the file you can make your changes permanent by running sysctl –p /etc/sysctl.conf and verify that is has been changed by:

root@ajax:~# cat /proc/sys/net/ipv4/tcp\_synack\_retries

**7.4.2: SYN cookies:**

Instead of the server keeping track of states for each connection which allocates memory, we can use SYN cookies instead. When a SYN is received a hash is computed based on meta information. The receiver (server) sends a SYN\_ACK with the hash and does not allocate any memory yet, only the hash is stored. The hash consists of the the following:

Initial Sequence Number (ISN) = hash(source\_ip, source\_port, destination\_ip, destination\_port, client's ISN, secret)

The sender must send an ACK with this hash so that the receiver can compare with the stored hash, if success than allocate memory and data structures.

Enabling SYN cookies in linux is very easy. Edit the file **/etc/sysctl.conf** and make the following modification:

net.ipv4.tcp\_syncookies = 1

Save and run sysctl -p to make the change permanent.

**7.4.3: Detection:**

Using an **Intrusion Detection System** such as Snort, it's possible to detect SYN flood attacks. Since I am running Snort in my network, I decided to create a snort rule to detect when running my SYN flooder program. Below you will find the rule:

alert tcp any any -> 192.168.1.0/24 any ( msg:"Possible SYN flood"; classtype:attempted-dos; sid:1999999; flags:S; flow: stateless; detection\_filter: track by\_dst, count 50, seconds 10;)

This rule will alert every SYN packet during one sampling period of 10 seconds, after the first 50 SYN packets. However this rule may not apply for all network environments. Depending on your needs you may need to increase/decrease the count or amount of seconds.

**7.5: Conclusion:**

Protecting against DoS attacks, particular SYN flood attacks with spoofing can be difficult. But there are some security measures that can be taken which will hopefully reduce the effects of a DoS attack. Using SYN cookies, IDS/IPS or simply reducing time spent in SYN\_RECV state are all possible methods to stop or reduce DoS attacks.

**CHAPTER – 8 : AUDITING WEB APPLICATIONS USING OWASP**

The Open Web Application Security Project (OWASP), an online community, produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.

**8.1: Here are the OWASP Top Ten:  
  
Invalidated input:** Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.

Broken access control: Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.

Broken authentication and session management: Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.

**Cross site scripting (XSS) flaws:** The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.

**Buffer overflows:** Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.

**Injection flaws:** Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.

**Improper error handling:** Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.

**Insecure storage:** Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.

**Denial of service (DoS):** Attackers can consume Web application resources to a point where other legitimate users can no longer access or use the application. Attackers can also lock users out of their accounts or even cause the entire application to fail.

**Insecure configuration management:**Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.

OWASP (Open Web Application Security Project) is an organization that provides unbiased and practical, cost-effective information about computer and Internet applications. Project members include a variety of security experts from around the world who share their knowledge of vulnerabilities, threats, attacks and countermeasures.

**8.2: DVWA**

**Damn Vulnerable Web App (DVWA)** is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

**8.2.1: Cross-site Scripting (XSS) Attack:**

Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of invalidated or uuencoded user input within the output it generates.

By leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim’s browser.

While XSS can be taken advantage of within VBScript, ActiveX and Flash (although now considered legacy or even obsolete), unquestionably, the most widely abused is JavaScript – primarily because JavaScript is fundamental to most browsing experiences.

**How Cross-site Scripting works?**

In order to run malicious JavaScript code in a victim’s browser, an attacker must first find a way to inject a payload into a web page that the victim visits. Of course, an attacker could use social engineering techniques to convince a user to visit a vulnerable page with an injected JavaScript payload.

In order for an XSS attack to take place the vulnerable website needs to directly include user input in its pages. An attacker can then insert a string that will be used within the web page and treated as code by the victim’s browser.

When the page loads in the victim’s browser, the attacker’s malicious script will execute, most often without the user realizing or being able to prevent such an attack.

**Important Note** — An XSS vulnerability can only exist if the payload (malicious script) that the attacker inserts ultimately get parsed (as HTML in this case) in the victim’s browser

**What’s the worst an attacker can do with JavaScript?**

The consequences of what an attacker can do with the ability to execute JavaScript on a web page may not immediately stand out, especially since browsers run JavaScript in a very tightly controlled environment and that JavaScript has limited access to the user’s operating system and the user’s files.

However, when considering that JavaScript has access to the following, it’s easier to understand how creative attackers can get with JavaScript.

Malicious JavaScript has access to all the same objects the rest of the web page has, including access to cookies. Cookies are often used to store session tokens, if an attacker can obtain a user’s session cookie, they can impersonate that user.

JavaScript can read and make arbitrary modifications to the browser’s DOM (within the page that JavaScript is running).

JavaScript can use XMLHttpRequest to send HTTP requests with arbitrary content to arbitrary destinations.

JavaScript in modern browsers can leverage HTML5 APIs such as accessing a user’s geolocation, webcam, microphone and even the specific files from the user’s file system. While most of these APIs require user opt-in, XSS in conjunction with some clever social engineering can bring an attacker a long way.

The above, in combination with social engineering, allow attackers to pull off advanced attacks including cookie theft, keylogging, phishing and identity theft. Critically, XSS vulnerabilities provide the perfect ground for attackers to escalate attacks to more serious ones.

**“Isn’t Cross-site scripting the user’s problem?”**

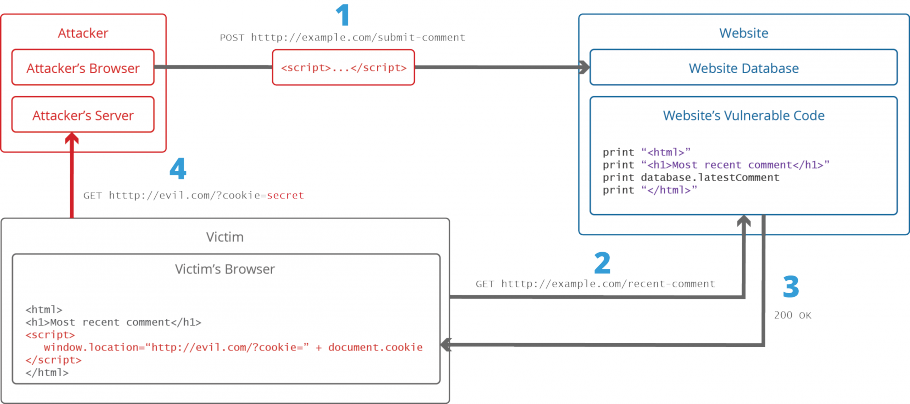
If an attacker can abuse a XSS vulnerability on a web page to execute arbitrary JavaScript in a visitor’s browser, the security of that website or web application and its users has been compromised — XSS is not the user’s problem, like any other security vulnerability, if it’s affecting your users, it will affect you.

**8.2.2: The anatomy of a Cross-site Scripting attack:**

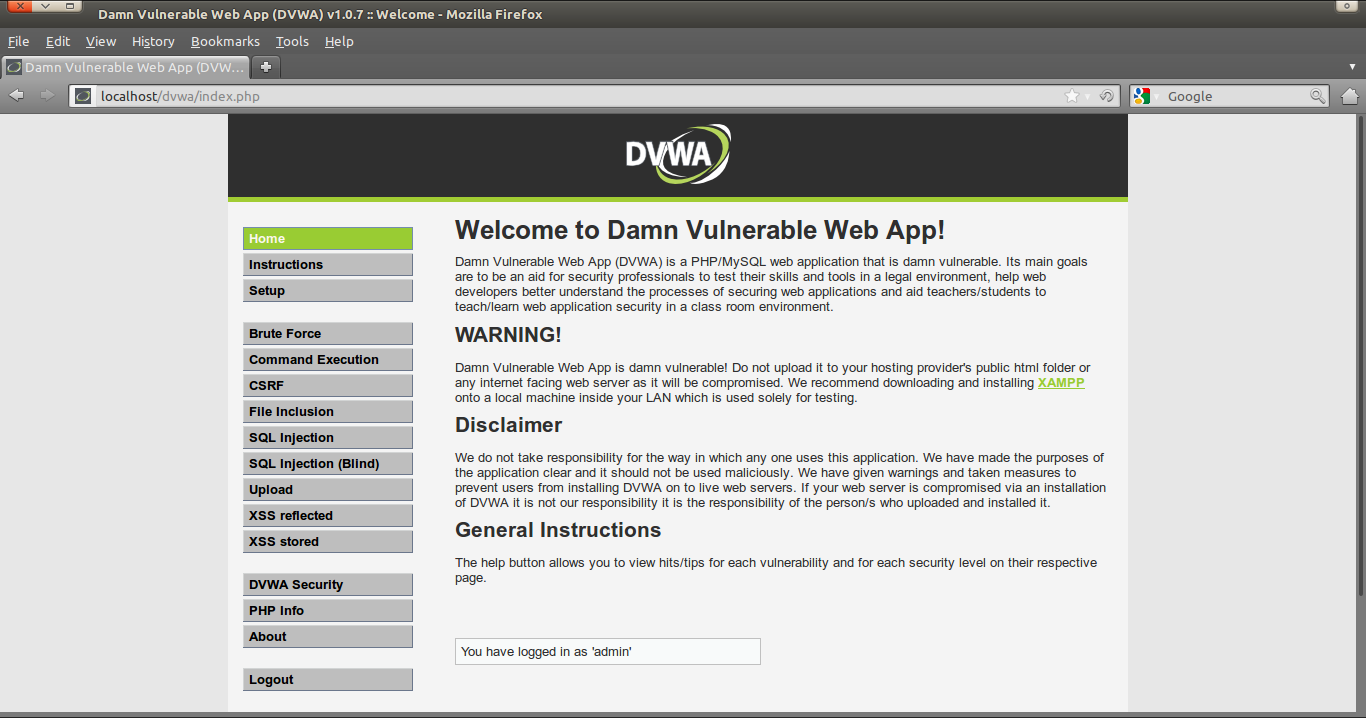
An XSS attack needs three actors — **the website, the victim** and **the attacker.**

In the example below, it shall be assumed that the attacker’s goal is to impersonate the victim by stealing the victim’s cookie. Sending the cookie to a server the attacker controls can be achieved in a variety of ways,

The figure below illustrates a step-by-step walkthrough of a simple XSS attack.



* The attacker injects a payload in the website’s database by submitting a vulnerable form with some malicious JavaScript
* The victim requests the web page from the website
* The website serves the victim’s browser the page with the attacker’s payload as part of the HTML body.
* The victim’s browser will execute the malicious script inside the HTML body. In this case it would send the victim’s cookie to the attacker’s server. The attacker now simply needs to extract the victim’s cookie when the HTTP request arrives to the server, after which the attacker can use the victim’s stolen cookie for impersonation.



**CHAPTER – 9 : BUFFER OVERFLOW**

In information security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.

Buffers are areas of memory set aside to hold data, often while moving it from one section of a program to another, or between programs. Buffer overflows can often be triggered by malformed inputs; if one assumes all inputs will be smaller than a certain size and the buffer is created to be that size, then an anomalous transaction that produces more data could cause it to write past the end of the buffer. If this overwrites adjacent data or executable code, this may result in erratic program behavior, including memory access errors, incorrect results, and crashes.

Exploiting the behavior of a buffer overflow is a well-known security exploit. On many systems, the memory layout of a program, or the system as a whole, is well defined. By sending in data designed to cause a buffer overflow, it is possible to write into areas known to hold executable code, and replace it with malicious code. Buffers are widespread in operating system (OS) code, so it is possible to make attacks that perform privilege escalation and gain unlimited access to the computer's resources. The famed Morris worm in 1988 used this as one of its attack techniques.

Programming languages commonly associated with buffer overflows include C and C++, which provide no built-in protection against accessing or overwriting data in any part of memory and do not automatically check that data written to an array (the built-in buffer type) is within the boundaries of that array. Bounds checking can prevent buffer overflows, but requires additional code and processing time. Modern operating systems use a variety of techniques to combat malicious buffer overflows, notably by randomizing the layout of memory, or deliberately leaving space between buffers and looking for actions that write into those areas ("canaries").

**9.1: Exploitation**:

The techniques to exploit a buffer overflow vulnerability vary by architecture, by operating system and by memory region. For example, exploitation on the heap (used for dynamically allocated memory), differs markedly from exploitation on the call stack.

**9.1.1: Stack-based exploitation:**

A technically inclined user may exploit stack-based buffer overflows to manipulate the program to their advantage in one of several ways:

* By overwriting a local variable that is located near the vulnerable buffer on the stack, in order to change the behavior of the program.
* By overwriting the return address in a stack frame. Once the function returns, execution will resume at the return address as specified by the attacker - usually a user-input filled buffer.
* By overwriting a function pointer or exception handler, which is subsequently executed.
* By overwriting a local variable (or pointer) of a different stack frame, which will be used by the function which owns that frame later.

If the address of the user-supplied data used to effect the stack buffer overflow is unpredictable, exploiting a stack buffer overflow to cause remote code execution becomes much more difficult. One technique that can be used to exploit such a buffer overflow is called "trampolining". In that technique, an attacker will find a pointer to the vulnerable stack buffer, and compute the location of their shellcode relative to that pointer. Then, they will use the overwrite to jump to an instruction already in memory which will make a second jump, this time relative to the pointer; that second jump will branch execution into the shellcode. Suitable instructions are often present in large code. The Metasploit Project, for example, maintains a database of suitable opcodes, though it lists only those found in the Windows operating system.

**9.1.2: Heap-based exploitation:**

A buffer overflow occurring in the heap data area is referred to as a heap overflow and is exploitable in a manner different from that of stack-based overflows. Memory on the heap is dynamically allocated by the application at run-time and typically contains program data. Exploitation is performed by corrupting this data in specific ways to cause the application to overwrite internal structures such as linked list pointers. The canonical heap overflow technique overwrites dynamic memory allocation linkage (such as malloc meta data) and uses the resulting pointer exchange to overwrite a program function pointer.

Microsoft's GDI+ vulnerability in handling JPEGs is an example of the danger a heap overflow can present.

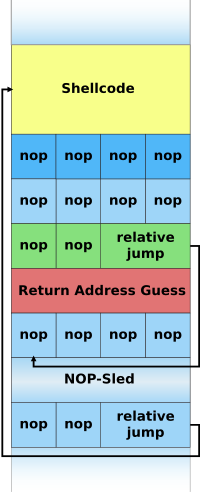
**9.2: Barriers to exploitation:**

Manipulation of the buffer, which occurs before it is read or executed, may lead to the failure of an exploitation attempt. These manipulations can mitigate the threat of exploitation, but may not make it impossible. Manipulations could include conversion to upper or lower case, removal of metacharacters and filtering out of non-alphanumeric strings. However, techniques exist to bypass these filters and manipulations; alphanumeric code, polymorphic code, self-modifying code and return-to-libc attacks. The same methods can be used to avoid detection by intrusion detection systems. In some cases, including where code is converted into Unicode, the threat of the vulnerability has been misrepresented by the disclosers as only Denial of Service when in fact the remote execution of arbitrary code is possible.

**9.3: Practicalities of exploitation:**

In real-world exploits there are a variety of challenges which need to be overcome for exploits to operate reliably. These factors include null bytes in addresses, variability in the location of shellcode, differences between environments and various counter-measures in operation.

**9.3.1: NOP sled technique:**

[](https://en.wikipedia.org/wiki/File:Nopsled.svg)

**Illustration of a NOP-sled payload on the stack:**

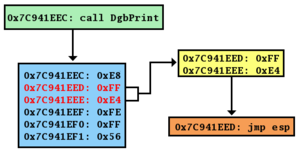
A NOP-sled is the oldest and most widely known technique for successfully exploiting a stack buffer overflow. It solves the problem of finding the exact address of the buffer by effectively increasing the size of the target area. To do this, much larger sections of the stack are corrupted with the no-op machine instruction. At the end of the attacker-supplied data, after the no-op instructions, the attacker places an instruction to perform a relative jump to the top of the buffer where the shellcode is located. This collection of no-ops is referred to as the "NOP-sled" because if the return address is overwritten with any address within the no-op region of the buffer, the execution will "slide" down the no-ops until it is redirected to the actual malicious code by the jump at the end. This technique requires the attacker to guess where on the stack the NOP-sled is instead of the comparatively small shellcode.

Because of the popularity of this technique, many vendors of intrusion prevention systems will search for this pattern of no-op machine instructions in an attempt to detect shellcode in use. It is important to note that a NOP-sled does not necessarily contain only traditional no-op machine instructions; any instruction that does not corrupt the machine state to a point where the shellcode will not run can be used in place of the hardware assisted no-op. As a result, it has become common practice for exploit writers to compose the no-op sled with randomly chosen instructions which will have no real effect on the shellcode execution.

While this method greatly improves the chances that an attack will be successful, it is not without problems. Exploits using this technique still must rely on some amount of luck that they will guess offsets on the stack that are within the NOP-sled region. An incorrect guess will usually result in the target program crashing and could alert the system administrator to the attacker's activities. Another problem is that the NOP-sled requires a much larger amount of memory in which to hold a NOP-sled large enough to be of any use. This can be a problem when the allocated size of the affected buffer is too small and the current depth of the stack is shallow (i.e. there is not much space from the end of the current stack frame to the start of the stack). Despite its problems, the NOP-sled is often the only method that will work for a given platform, environment, or situation; as such it is still an important technique.

**9.3.2: The jump to address stored in a register technique:**

The "jump to register" technique allows for reliable exploitation of stack buffer overflows without the need for extra room for a NOP-sled and without having to guess stack offsets. The strategy is to overwrite the return pointer with something that will cause the program to jump to a known pointer stored within a register which points to the controlled buffer and thus the shellcode. For example, if register A contains a pointer to the start of a buffer then any jump or call taking that register as an operand can be used to gain control of the flow of execution.

[](https://en.wikipedia.org/wiki/File:JumpToEsp.png)

An instruction from ntdll.dll to call the DbgPrint() routine contains the i386 machine opcode for jmp esp.

In practice a program may not intentionally contain instructions to jump to a particular register. The traditional solution is to find an unintentional instance of a suitable opcode at a fixed location somewhere within the program memory. In figure E on the left you can see an example of such an unintentional instance of the i386 jmp espinstruction. The opcode for this instruction is FF E4. This two-byte sequence can be found at a one-byte offset from the start of the instruction call DbgPrint at address 0x7C941EED. If an attacker overwrites the program return address with this address the program will first jump to 0x7C941EED, interpret the opcode FF E4 as the jmp esp instruction, and will then jump to the top of the stack and execute the attacker's code.

When this technique is possible the severity of the vulnerability increases considerably. This is because exploitation will work reliably enough to automate an attack with a virtual guarantee of success when it is run. For this reason, this is the technique most commonly used in Internet worms that exploit stack buffer overflow vulnerabilities.

This method also allows shellcode to be placed after the overwritten return address on the Windows platform. Since executables are mostly based at address 0x00400000 and x86 is a Little Endian architecture, the last byte of the return address must be a null, which terminates the buffer copy and nothing is written beyond that. This limits the size of the shellcode to the size of the buffer, which may be overly restrictive. DLLs are located in high memory (above 0x01000000) and so have addresses containing no null bytes, so this method can remove null bytes (or other disallowed characters) from the overwritten return address. Used in this way, the method is often referred to as "DLL Trampolining".

**9.4: Protective countermeasures**:

Various techniques have been used to detect or prevent buffer overflows, with various tradeoffs. The most reliable way to avoid or prevent buffer overflows is to use automatic protection at the language level. This sort of protection, however, cannot be applied to legacy code, and often technical, business,   
  
or cultural constraints call for a vulnerable language. The following sections describe the choices and implementations available.

**9.4.1: Choice of programming language:**

Assembly and C/C++ are popular programming languages that are vulnerable to buffer overflow, in part because they allow direct access to memory and are not strongly typed. C provides no built-in protection against accessing or overwriting data in any part of memory; more specifically, it does not check that data written to a buffer is within the boundaries of that buffer. The standard C++ libraries provide many ways of safely buffering data, and C++'s Standard Template Library (STL) provides containers that can optionally perform bounds checking if the programmer explicitly calls for checks while accessing data. For example, a vector's member function at() performs a bounds check and throws an out\_of\_range exception if the bounds check fails. However, C++ behaves just like C if the bounds check is not explicitly called. Techniques to avoid buffer overflows also exist for C.

**9.4.2: Use of safe libraries:**

The problem of buffer overflows is common in the C and C++ languages because they expose low level representational details of buffers as containers for data types. Buffer overflows must thus be avoided by maintaining a high degree of correctness in code which performs buffer management. It has also long been recommended to avoid standard library functions which are not bounds checked, such as gets, scanf and strcpy. The Morris worm exploited a gets call in fingerd.

Well-written and tested abstract data type libraries which centralize and automatically perform buffer management, including bounds checking, can reduce the occurrence and impact of buffer overflows. The two main building-block data types in these languages in which buffer overflows commonly occur are strings and arrays; thus, libraries preventing buffer overflows in these data types can provide the vast majority of the necessary coverage. Still, failure to use these safe libraries correctly can result in buffer overflows and other vulnerabilities; and naturally, any bug in the library itself is a potential vulnerability. "Safe" library implementations include "The Better String Library", Vstr and Erwin. The OpenBSD operating system's C library provides the strlcpy and strlcat functions, but these are more limited than full safe library implementations.

However, the efficacy of these functions for the purpose of reducing buffer overflows is disputable; it requires programmer intervention on a per function call basis that is equivalent to intervention that could make the analogous older standard library functions buffer overflow safe.

**9.4.3: Buffer overflow protection:**

Buffer overflow protection is used to detect the most common buffer overflows by checking that the stack has not been altered when a function returns. If it has been altered, the program exits with a segmentation fault. Three such systems are Libsafe, and the StackGuard and ProPolice gcc patches.

Microsoft's implementation of Data Execution Prevention (DEP) mode explicitly protects the pointer to the Structured Exception Handler (SEH) from being overwritten.

Stronger stack protection is possible by splitting the stack in two: one for data and one for function returns. This split is present in the Forth language, though it was not a security-based design decision. Regardless, this is not a complete solution to buffer overflows, as sensitive data other than the return address may still be overwritten.

**9.4.4: Pointer protection:**

Buffer overflows work by manipulating pointers (including stored addresses). PointGuard was proposed as a compiler-extension to prevent attackers from being able to reliably manipulate pointers and addresses. The approach works by having the compiler add code to automatically XOR-encode pointers before and after they are used. Because the attacker (theoretically) does not know what value will be used to encode/decode the pointer, he cannot predict what it will point to if he overwrites it with a new value. PointGuard was never released, but Microsoft implemented a similar approach beginning in Windows XP SP2 and Windows Server 2003 SP1. Rather than implement pointer protection as an automatic feature, Microsoft added an API routine that can be called at the discretion of the programmer. This allows for better performance (because it is not used all of the time), but places the burden on the programmer to know when it is necessary.

Because XOR is linear, an attacker may be able to manipulate an encoded pointer by overwriting only the lower bytes of an address. This can allow an attack to succeed if the attacker is able to attempt the exploit multiple times or is able to complete an attack by causing a pointer to point to one of several locations (such as any location within a NOP sled). Microsoft added a random rotation to their encoding scheme to address this weakness to partial overwrites.

**CHAPTER – 10 : RANSOMWARE ATTACK**

Ransomware is a type of malicious software from cryptovirology that threatens to publish the victim's data or perpetually block access to it unless a ransom is paid. While some simple ransomware may lock the system in a way which is not difficult for a knowledgeable person to reverse, more advanced malware uses a technique called cryptoviral extortion, in which it encrypts the victim's files, making them inaccessible, and demands a ransom payment to decrypt them. In a properly implemented cryptoviral extortion attack, recovering the files without the decryption key is an intractable problem – and difficult to trace digital currencies such as Ukash and cryptocurrency are used for the ransoms, making tracing and prosecuting the perpetrators difficult.

Ransomware attacks are typically carried out using a Trojan that is disguised as a legitimate file that the user is tricked into downloading or opening when it arrives as an email attachment. However, one high-profile example, the "WannaCry worm", traveled automatically between computers without user interaction.

**10.1: Operation**:

The concept of file encrypting ransomware was invented and implemented by Young and Yung at Columbia University and was presented at the 1996 IEEE Security & Privacy conference. It is called cryptoviral extortion and it was inspired by the fictional facehugger in the movie Alien. Cryptoviral extortion is the following three-round protocol carried out between the attacker and the victim.

**[attacker→victim]** The attacker generates a key pair and places the corresponding public key in the malware. The malware is released.

**[victim→attacker]** To carry out the cryptoviral extortion attack, the malware generates a random symmetric key and encrypts the victim's data with it. It uses the public key in the malware to encrypt the symmetric key. This is known as hybrid encryption and it results in a small asymmetric ciphertext as well as the symmetric ciphertext of the victim's data. It zeroizes the symmetric key and the original plaintext data to prevent recovery. It puts up a message to the user that includes the asymmetric ciphertext and how to pay the ransom. The victim sends the asymmetric ciphertext and e-money to the attacker.

**[attacker→victim]** The attacker receives the payment, deciphers the asymmetric ciphertext with the attacker's private key, and sends the symmetric key to the victim. The victim deciphers the encrypted data with the needed symmetric key thereby completing the cryptovirology attack.

The symmetric key is randomly generated and will not assist other victims. At no point is the attacker's private key exposed to victims and the victim need only send a very small ciphertext (the encrypted symmetric-cipher key) to the attacker.

Ransomware attacks are typically carried out using a Trojan, entering a system through, for example, a downloaded file or a vulnerability in a network service. The program then runs a payload, which locks the system in some fashion, or claims to lock the system but does not (e.g., a scareware program). Payloads may display a fake warning purportedly by an entity such as a law enforcement agency, falsely claiming that the system has been used for illegal activities, contains content such   
  
  
as pornography and "pirated" media.

Some payloads consist simply of an application designed to lock or restrict the system until payment is made, typically by setting the Windows Shell to itself, or even modifying the master boot record and/or partition table to prevent the operating system from booting until it is repaired.The most sophisticated payloads encrypt files, with many using strong encryption to encrypt the victim's files in such a way that only the malware author has the needed decryption key.

**10.2: File system defences against ransomware**:

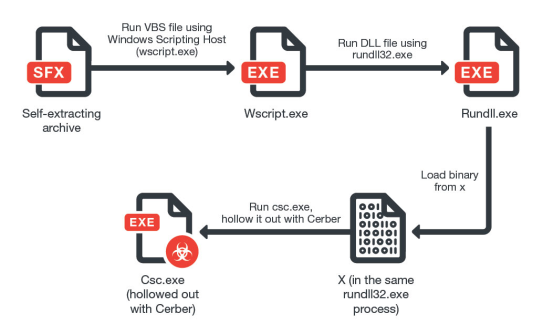
A number of file systems keep snapshots of the data they hold, which can be used to recover the contents of files from a time prior to the ransomware attack in the event the ransomware doesn't disable it.

On Windows, the Volume shadow copy (VSS) is often used to store backups of data; ransomware often targets these snapshots to prevent recovery and therefore it is often advisable to disable user access to the user tool VSSadmin.exe to reduce the risk that ransomware can disable or delete past copies.

File servers running ZFS are almost universally immune to ransomware, because ZFS is capable of snapshotting even a large file system many times an hour, and these snapshots are immutable (read only) and easily rolled back or files recovered in the event of data corruption. In general, only an administrator can delete (but cannot modify) snapshots.

**10.3: File decryption and recovery:**

There are a number of tools intended specifically to decrypt files locked by ransomware, although successful recovery may not be possible. If the same encryption key is used for all files, decryption tools use files for which there are both uncorrupted backups and encrypted copies (a known-plaintext attack in the jargon of cryptanalysis); recovery of the key, if it is possible, may take several days. Free ransomware decryption tools can help decrypt files encrypted by the following forms of ransomware: AES\_NI, Alcatraz Locker, Apocalypse, BadBlock, Bart, BTCWare, Crypt888, CryptoMix, CrySiS, EncrypTile, FindZip, Globe, Hidden Tear, Jigsaw, LambdaLocker, Legion, NoobCrypt, Stampado, SZFLocker, TeslaCrypt, XData.

In addition, old copies of files may exist on the disk, which have been previously deleted. In some cases these deleted versions may still be recoverable using software designed for that purpose.  
  
 

**CHAPTER – 11 : AES ENCRYPTION ALGORITHM**

**11.1: Description of the cipher:**

AES is based on a design principle known as a substitution–permutation network, a combination of both substitution and permutation, and is fast in both software and hardware.Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification per se is specified with block and key sizes that may be any multiple of 32 bits, with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4 × 4 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a particular finite field.

{\displaystyle {\begin{bmatrix}b\_{0}&b\_{4}&b\_{8}&b\_{12}\\b\_{1}&b\_{5}&b\_{9}&b\_{13}\\b\_{2}&b\_{6}&b\_{10}&b\_{14}\\b\_{3}&b\_{7}&b\_{11}&b\_{15}\end{bmatrix}}}

The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of cycles of repetition are as follows:

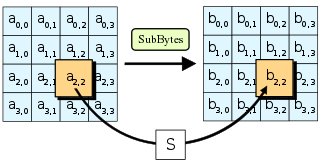
**10 cycles of repetition for 128-bit keys.**

**12 cycles of repetition for 192-bit keys.**

**14 cycles of repetition for 256-bit keys.**

Each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

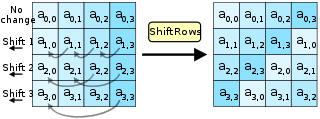
**The SubBytes step:**

[](https://en.wikipedia.org/wiki/File:AES-SubBytes.svg)

In the SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S; bij = S(aij).

In the SubBytes step, each byte {\displaystyle a\_{i,j}}in the state matrix is replaced with a SubByte {\displaystyle S(a\_{i,j})}using an 8-bit substitution box, the Rijndael S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF(28), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), also any opposite fixed points. While performing the decryption, the InvSubBytes step (the inverse of SubBytes) is used, which requires first taking the inverse of the affine transformation and then finding the multiplicative inverse.

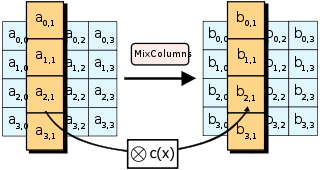
**The ShiftRows step:**

[](https://en.wikipedia.org/wiki/File:AES-ShiftRows.svg)

In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row.

The ShiftRows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For blocks of sizes 128 bits and 192 bits, the shifting pattern is the same. Row {\displaystyle n}is shifted left circular by {\displaystyle n-1}bytes. In this way, each column of the output state of the ShiftRows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets). For a 256-bit block, the first row is unchanged and the shifting for the second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively—this change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks. The importance of this step is to avoid the columns being encrypted independently, in which case AES degenerates into four independent block ciphers.

**The MixColumns step:**

[](https://en.wikipedia.org/wiki/File:AES-MixColumns.svg)

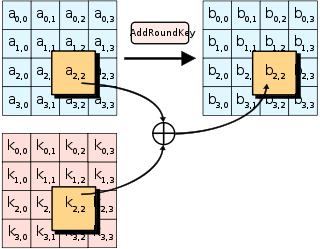
In the MixColumns step, each column of the state is multiplied with a fixed polynomial{\displaystyle c(x)}.

In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides diffusion in the cipher.

During this operation, each column is transformed using a fixed matrix (matrix left-multiplied by column gives new value of column in the state):

{\displaystyle {\begin{bmatrix}2&3&1&1\\1&2&3&1\\1&1&2&3\\3&1&1&2\end{bmatrix}}}Matrix multiplication is composed of multiplication and addition of the entries. If processed bit by bit, then, after shifting, a conditional XORwith 1B16 should be performed if the shifted value is larger than FF16 (overflow must be corrected by subtraction of generating polynomial). In more general sense, each column is treated as a polynomial over {\displaystyle \operatorname {GF} (2^{8})}and is then multiplied modulo {\displaystyle {01}\_{16}\cdot z^{4}+{01}\_{16}}with a fixed polynomial {\displaystyle c(z)={03}\_{16}\cdot z^{3}+{01}\_{16}\cdot z^{2}+{01}\_{16}\cdot z+{02}\_{16}}. The coefficients are displayed in their hexadecimal equivalent of the binary representation of bit polynomials from {\displaystyle \operatorname {GF} (2)[x]}. The MixColumns step can also be viewed as a multiplication by the shown   
  
particular MDS matrix in the finite field {\displaystyle \operatorname {GF} (2^{8})}. This process is described further in the article Rijndael MixColumns.

**The AddRoundKey step**:

[](https://en.wikipedia.org/wiki/File:AES-AddRoundKey.svg)

In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XORoperation (⊕).

In the AddRoundKey step, the subkey is combined with the state. For each round, a subkey is derived from the main keyusing Rijndael's key schedule; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by combining the SubBytes and ShiftRows steps with the MixColumns step by transforming them into a sequence of table lookups. This requires four 256-entry 32-bit tables (together occupying 4096 bytes). A round can then be performed with 16 table lookup operations and 12 32-bit exclusive-or operations, followed by four 32-bit exclusive-or operations in the AddRoundKey step. Alternatively, the table lookup operation can be performed with a single 256-entry 32-bit table (occupying 1024 bytes) followed by circular rotation operations.

Using a byte-oriented approach, it is possible to combine the SubBytes, ShiftRows, and MixColumns steps into a single round operation.

**11.2: Initialization vector (IV):**

An initialization vector (IV) or starting variable (SV)is a block of bits that is used by several modes to randomize the encryption and hence to produce distinct ciphertexts even if the same plaintext is encrypted multiple times, without the need for a slower re-keying process.

An initialization vector has different security requirements than a key, so the IV usually does not need to be secret. However, in most cases, it is important that an initialization vector is never reused under the same key. For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages. For OFB and CTR, reusing an IV completely destroys security. This can be seen because both modes effectively create a bitstream that is XORed with the plaintext, and this bitstream is dependent on the password and IV only. Reusing a bitstream destroys security. In CBC mode, the IV must, in addition, be unpredictable at   
  
encryption time; in particular, the (previously) common practice of re-using the last ciphertext block of a message as the IV for the next message is insecure (for example, this method was used by SSL 2.0). If an attacker knows the IV (or the previous block of ciphertext) before he specifies the next plaintext, he can check his guess about plaintext of some block that was encrypted with the same key before (this is known as the TLS CBC IV attack).

**11.3: Padding:**

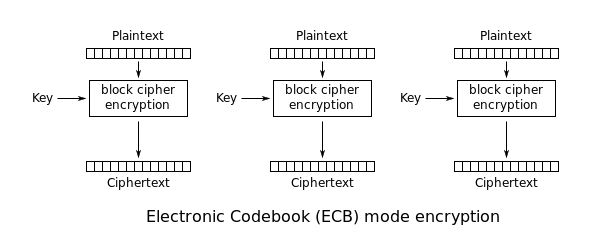
A block cipher works on units of a fixed size (known as a block size), but messages come in a variety of lengths. So some modes (namely **ECB** and **CBC**) require that the final block be padded before encryption.   
  
Several padding schemes exist. The simplest is to add null bytes to the plaintext to bring its length up to a multiple of the block size, but care must be taken that the original length of the plaintext can be recovered; this is trivial, for example, if the plaintext is a C style string which contains no null bytes except at the end. Slightly more complex is the original DES method, which is to add a single one bit, followed by enough zero bits to fill out the block; if the message ends on a block boundary, a whole padding block will be added. Most sophisticated are CBC-specific schemes such as ciphertext stealing or residual block termination, which do not cause any extra ciphertext, at the expense of some additional complexity. Schneier and Ferguson suggest two possibilities, both simple: append a byte with value 128 (hex 80), followed by as many zero bytes as needed to fill the last block, or pad the last block with n bytes all with value n.

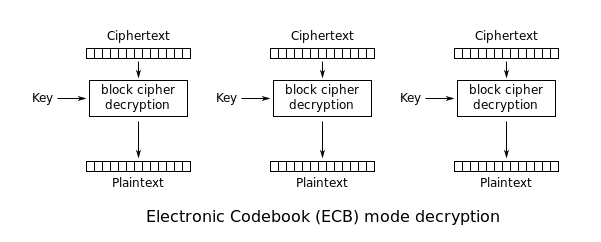
**CFB**, **OFB** and **CTR** modes do not require any special measures to handle messages whose lengths are not multiples of the block size, since the modes work by XORing the plaintext with the output of the block cipher. The last partial block of plaintext is XORed with the first few bytes of the last keystream block, producing a final ciphertext block that is the same size as the final partial plaintext block. This characteristic of stream ciphers makes them suitable for applications that require the encrypted ciphertext data to be the same size as the original plaintext data, and for applications that transmit data in streaming form where it is inconvenient to add padding bytes.

**11.4: BLOCK CIPHER MODES OF OPERATION:**

**11.4.1: Electronic Codebook (ECB):**

The simplest of the encryption modes is the Electronic Codebook (ECB) mode (named after conventional physical codebooks).The message is divided into blocks, and each block is encrypted separately.

[](https://en.wikipedia.org/wiki/File:ECB_encryption.svg)

[](https://en.wikipedia.org/wiki/File:ECB_decryption.svg)

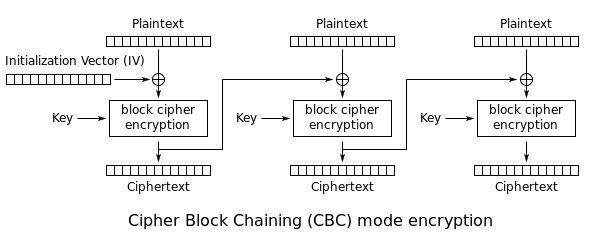
The disadvantage of this method is a lack of diffusion. Because ECB encrypts identical plaintext blocks into identical ciphertext blocks, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.

A striking example of the degree to which ECB can leave plaintext data patterns in the ciphertext can be seen when ECB mode is used to encrypt a bitmap image which uses large areas of uniform color. While the color of each individual pixel is encrypted, the overall image may still be discerned as the pattern of identically colored pixels in the original remains in the encrypted version.

ECB mode can also make protocols without integrity protection even more susceptible to replay attacks, since each block gets decrypted in exactly the same way.

**11.4.2: Cipher Block Chaining (CBC):**

Ehrsam, Meyer, Smith and Tuchman invented the Cipher Block Chaining (CBC) mode of operation in 1976. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.

[](https://en.wikipedia.org/wiki/File:CBC_encryption.svg)

[](https://en.wikipedia.org/wiki/File:CBC_decryption.svg)

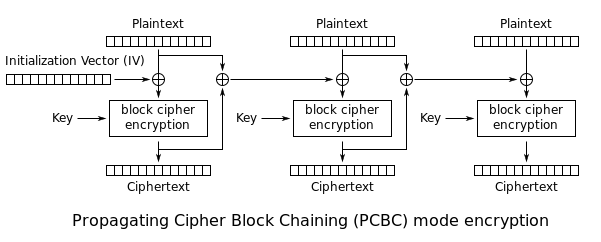
CBC has been the most commonly used mode of operation. Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size. One way to handle this last issue is through the method known as ciphertext stealing. Note that a one-bit change in a plaintext or initialization vector (IV) affects all following ciphertext blocks.

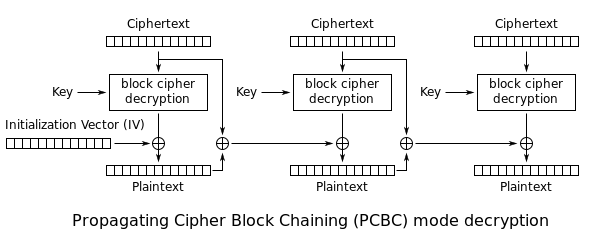
Decrypting with the incorrect IV causes the first block of plaintext to be corrupt but subsequent plaintext blocks will be correct. This is because each block is XORed with the ciphertext of the previous block, not the plaintext, so one does not need to decrypt the previous block before using it as the IV for the decryption of the current one. This means that a plaintext block can be recovered from two adjacent blocks of ciphertext. As a consequence, decryption can be parallelized. Note that a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext, but the rest of the blocks remain intact. This peculiarity is exploited in different padding oracle attacks, such as POODLE.

Explicit Initialization Vectors takes advantage of this property by prepending a single random block to the plaintext. Encryption is done as normal, except the IV does not need to be communicated to the decryption routine. Whatever IV decryption uses, only the random block is "corrupted". It can be safely discarded and the rest of the decryption is the original plaintext.

**11.4.3: Propagating Cipher Block Chaining (PCBC):**

The Propagating Cipher Block Chaining or plaintext cipher-block chaining mode was designed to cause small changes in the ciphertext to propagate indefinitely when decrypting, as well as when encrypting. In PCBC mode, each block of plaintext is XORed with both the previous plaintext block and the previous ciphertext block before being encrypted. As with CBC mode, an initialization vector is used in the first block.

[](https://en.wikipedia.org/wiki/File:PCBC_encryption.svg)

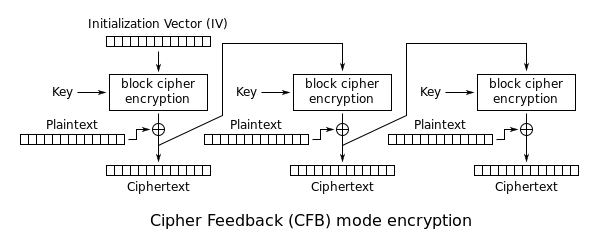
[](https://en.wikipedia.org/wiki/File:PCBC_decryption.svg)

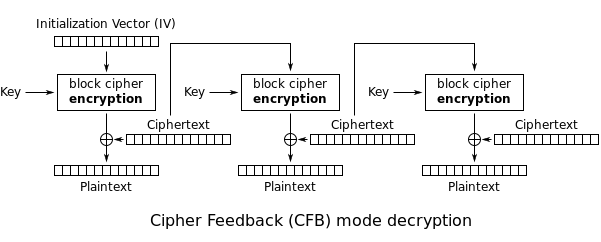
**Encryption and decryption algorithms are as follows:**

PCBC is used in Kerberos v4 and WASTE, most notably, but otherwise is not common. On a message encrypted in PCBC mode, if two adjacent ciphertext blocks are exchanged, this does not affect the decryption of subsequent blocks. For this reason, PCBC is not used in Kerberos v5.

**11.4.4: Cipher Feedback (CFB):**

The Cipher Feedback (CFB) mode, a close relative of CBC, makes a block cipher into a self-synchronizing stream cipher. Operation is very similar; in particular, CFB decryption is almost identical to CBC encryption performed in reverse:

[](https://en.wikipedia.org/wiki/File:CFB_encryption.svg)

[](https://en.wikipedia.org/wiki/File:CFB_decryption.svg)

By definition of self-synchronising cipher, if part of the ciphertext is lost (e.g. due to transmission errors), then the receiver will lose only some part of the original message (garbled content), and should be able to continue to correctly decrypt the rest of the blocks after processing some amount of input data. This simplest way of using CFB described above is not self-synchronizing. Only if a whole blocksize of ciphertext is lost CFB will synchronize, but losing only a single byte or bit will permanently throw off decryption. To be able to synchronize after the loss of only a single byte or bit, a single byte or bit must be encrypted at a time. CFB can be used this way when combined with a shift register as the input for the block cipher.

To use CFB to make a self-synchronizing stream cipher that will synchronize for any multiple of x bits lost, start by initializing a shift register the size of the block size with the initialization vector. This is encrypted with the block cipher, and the highest x bits of the result are XOR'ed with x bits of the plaintext to produce x bits of ciphertext. These x bits of output are shifted into the shift register, and the process (starting with encrypting the shift register with the block cipher) repeats for the next x bits of plaintext. Decryption is similar, start with the initialization vector, encrypt, and XOR the high bits of the result with x bits of the ciphertext to produce x bits of plaintext, then shift the x bits of the ciphertext into the shift register and encrypt again. This way of proceeding is known as CFB-8 or   
  
  
CFB-1 (according to the size of the shifting).

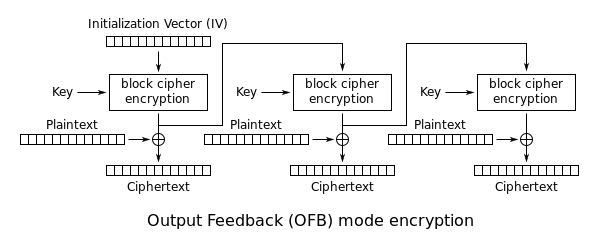
If x bits are lost from the ciphertext, the cipher will output incorrect plaintext until the shift register once again equals a state it held while encrypting, at which point the cipher has resynchronized. This will result in n/s blocksize of output being garbled with n the blocksize and s the amount of bits shifted.

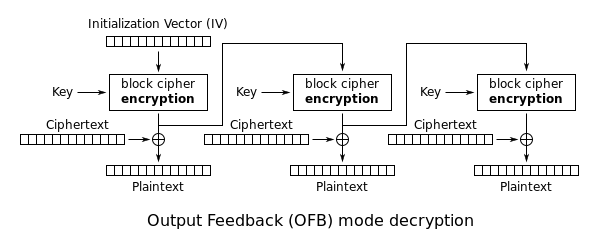
Like CBC mode, changes in the plaintext propagate forever in the ciphertext, and encryption cannot be parallelized. Also like CBC, decryption can be parallelized. When decrypting, a one-bit change in the ciphertext affects two plaintext blocks: a one-bit change in the corresponding plaintext block, and complete corruption of the following plaintext block. Later plaintext blocks are decrypted normally.

CFB shares two advantages over CBC mode with the stream cipher modes OFB and CTR: the block cipher is only ever used in the encrypting direction, and the message does not need to be padded to a multiple of the cipher block size (though ciphertext stealing can also be used to make padding unnecessary).

**11.4.5: Output Feedback (OFB):**

The Output Feedback (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Just as with other stream ciphers, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error correcting codes to function normally even when applied before encryption.  
  
  
Because of the symmetry of the XOR operation, encryption and decryption are exactly the same:

[](https://en.wikipedia.org/wiki/File:OFB_encryption.svg)

[](https://en.wikipedia.org/wiki/File:OFB_decryption.svg)

Each output feedback block cipher operation depends on all previous ones, and so cannot be performed in parallel. However, because the plaintext or ciphertext is only used for the final XOR, the block cipher operations may be performed in advance, allowing the final step to be performed in parallel once the plaintext or ciphertext is available.

It is possible to obtain an OFB mode keystream by using CBC mode with a constant string of zeroes as input. This can be useful, because it allows the usage of fast hardware implementations of CBC mode for OFB mode encryption.

Using OFB mode with a partial block as feedback like CFB mode reduces the average cycle length by a factor of {\displaystyle 2^{32}} or more. A mathematical model proposed by Davies and Parkin and substantiated by experimental results showed that only with full feedback an average cycle length near to the obtainable maximum can be achieved. For this reason, support for truncated feedback was removed from the specification of OFB.

**11.4.6: Counter (CTR):**

CTR mode (CM) is also known as integer counter mode (ICM) and segmented integer counter (SIC) mode

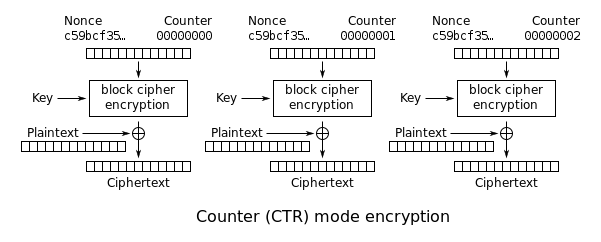
Like OFB, Counter mode turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a "counter". The counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual increment-by-one counter is the simplest and most popular. The usage of a simple deterministic input function used to be controversial; critics argued that "deliberately exposing a cryptosystem to a known systematic input represents an unnecessary risk." However, today CTR mode is widely accepted and any problems are considered a weakness of the underlying block cipher, which is expected to be secure regardless of systemic bias in its input. Along with CBC, CTR mode is one of two block cipher modes recommended by Niels Ferguson and Bruce Schneier.

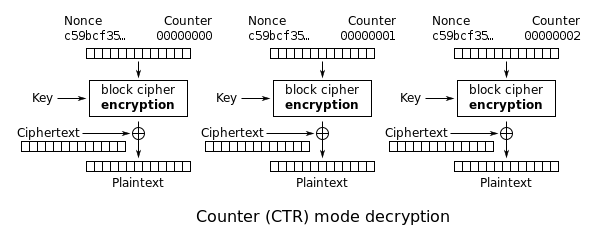
**CTR mode was introduced by Whitfield Diffie and Martin Hellman in 1979.**

CTR mode has similar characteristics to OFB, but also allows a random access property during decryption. CTR mode is well suited to operate on a multi-processor machine where blocks can be encrypted in parallel. Furthermore, it does not suffer from the short-cycle problem that can affect OFB.

If the IV/nonce is random, then they can be combined together with the counter using any lossless operation (concatenation, addition, or XOR) to produce the actual unique counter block for encryption.   
  
In case of a non-random nonce (such as a packet counter), the nonce and counter should be concatenated (e.g., storing the nonce in the upper 64 bits and the counter in the lower 64 bits of a 128-bit counter block). Simply adding or XORing the nonce and counter into a single value would break the security under a chosen-plaintext attack in many cases, since the attacker may be able to manipulate the entire IV–counter pair to cause a collision. Once an attacker controls the IV–counter pair and plaintext, XOR of the ciphertext with the known plaintext would yield a value that, when XORed with the ciphertext of the other block sharing the same IV–counter pair, would decrypt that block.

Note that the nonce in this diagram is equivalent to the initialization vector (IV) in the other diagrams. However, if the offset/location information is corrupt, it will be impossible to partially recover such data due to the dependence on byte offset.

[](https://en.wikipedia.org/wiki/File:CTR_encryption_2.svg)

[](https://en.wikipedia.org/wiki/File:CTR_decryption_2.svg)