Network Security

Edoardo Righi

August 24, 2020

Contents

1 Security Properties . . . 2
1.1 Trust in information security . . . 3
1.2 Thompson's view example . . . 4
1.3 Computer Security vs Network Security . . . 4
1.4 Attack models in network setting: outright malicious attacker . . . 5
1.5 Attack models in network setting: honest-but-curious attacker . . . 5
2 Network aspects . . . 6
2.1 OSI Data link layer . . . 6
2.2 OSI Network layer . . . 7
2.3 OSI Transport Layer . . . 10
2.4 OSI Session/Presentation/Application Layer . . . 15
3 Security Protocols . . . 21
3.1 IPSec . . . 22
3.2 TLS . . . 27
4 Vulnerabilities . . . 34
4.1 Vulnerability grading . . . 38
5 Malware . . . 42
5.1 Keylogger . . . 42
5.2 Social Engineering . . . 42
5.3 Viruses . . . 43
5.4 Worms . . . 46
5.5 Botnets . . . 49
6 Web based attacks . . . 53
6.1 Malware propagation . . . 53
6.2 Malware delivery – mechanisms review . . . 54
6.3 Exploit kits . . . 57
6.4 Advanced Denial of Service attacks . . . 58
7 System hardening . . . 60
8 System hardening - Firewalls . . . 61
8.1 Static Packet Filtering . . . 62
8.2 Stateful Packet Filtering . . . 63
8.3 Proxy . . . 65
9 System hardening - IDS . . . 71
9.1 Intrusion Detection Approaches . . . 72
9.2 NIDS evasion [Siddharth 2005] . . . 77
9.3 Audit Records . . . 79
9.4 Security Information and Event Management (SIEM) . . . 80
9.5 Honeypots . . . 81
10 Privacy issues . . . 83
10.1 Browser cookies and tracking . . . 83
10.2 Attacks out of the browser: email . . . 86
10.3 Source Confidentiality . . . 87

1 Security Properties

> The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, the availability and confidentiality of information systems resources.
>
> NIST Computer Security Handbook

Computer security is about **preserving** security: you can only preserve something that you already have. Security technologies do not build security properties, they "merely" make sure that the security properties that are already there are maintained.
There are 2 main concepts to understand:

- What is it there to be preserved
- What is the "built-in" security that needs to be preserved

Everything that operates and is operated by a computer is information: all a computer system is about is information. At the bare minimum, any "piece of information" is only useful if:

- It can be reached
- It can be read
- It is correct

These can be seen as **"properties" of information**. Computer security is about preserving these properties:

- Availability: Assure that a piece of information can be reached when needed
- Confidentiality: Assure that a piece of info can be read only by those who are authorized to read it
- Integrity: Prevent unauthorised modification of information (prevent unauthorised writing). While data integrity, integrity synonymous for external consistency.

This is known as the **CIA triad**. They are the "core" security properties of a piece of information. On top of this, one can build additional properties:

- Accountability: the ability to know with certainty who/what operated on a piece of information
- Non-repudiation: the entity that acted on the information cannot "repudiate" his/her/its action
- Authentication:
  - User authentication: the ability to prove that a person is who she claims to be
  - Message authentication: assurance that the message has been sent by a party (IP address, server, computer, process, etc.)

The previously used terms for the actors on informations are who/what, entity, his/her/its. That is because humans are not the only "users" of information:

- The human user of the system avertedly or un-avertedly modifies information
- The automated user can modify the information (Avertedly? Un-avertedly?)

Surrogates and delegation

A system/software/module/thread can act on the whole system (or another system) on behalf of a human user. "Delegation" is the mechanism by which, for example, software threads are spawned (e.g. with the privileges of the entity that spawned it). While a human user may or may not know what he/she does, a "surrogate" does not. There is no notion of "avertedly" or "un-avertedly".

Core problem of computer security

Computer systems do not know what they are doing. They can only execute instructions (i.e. information) to operate over some other information. Systems can only be instructed to protect the security properties of that information by means of some mechanism:

- E.g. Confidentiality → crypto
- E.g. Integrity → secure hardware
- E.g. Availability → redundancy

1.1 Trust in information security

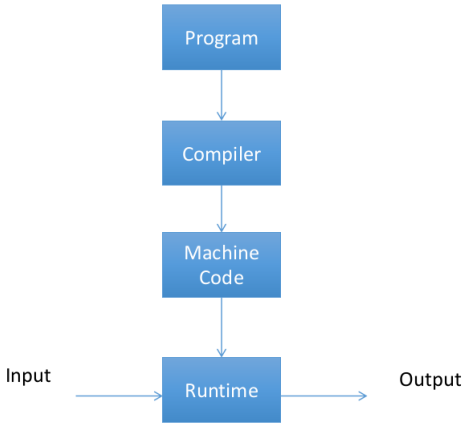
What software/system can one trust? Imagine an authentication mechanism:

- User inputs username;
- User inputs password;
- User presses enter;
- User has access to Desktop.

The user trusts the authentication mechanism, but what happened really? Did the authentication SW do the actual match? Will it only grant access if it matches your credentials? Did it send your credentials to a third party? Did it use your credentials to read and copy your data (e.g. as stored in an encrypted volume)?
Question: How do you increase your level of trust of the software? You look at the source code, but is this enough? What or who did really generate that software?

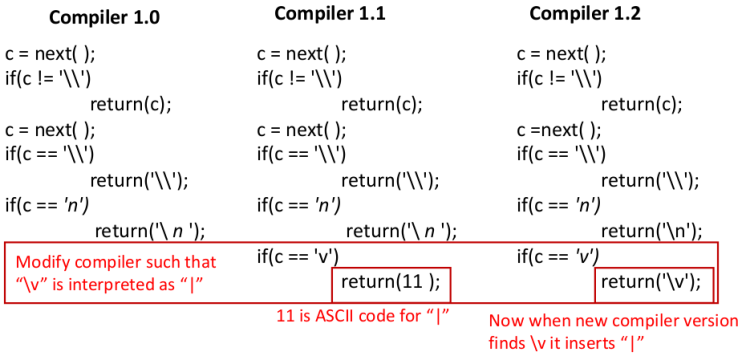
- The human being that wrote the sw source code?
- The compiler that compiled the sw source code?
- The human being that wrote the compiler that compiles the sw source code?
- The compiler that compiled the compiler that compiled the sw source code?
- Etc..

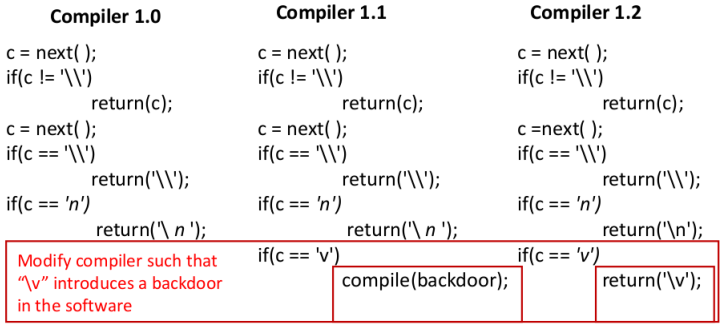
The point is: **who do you trust?**



1.2 Thompson's view example

A compiler is written in C and is compiled by a previous version of itself. It takes one generation to add a "backdoor" that will automatically be included in the software compiled with the next compiler.





The compiler can be modified in any way to include code that never appears in the SW source code. And depending on how many generations passed, it won't appear in the previous compiler versions source code either.

> You can't trust code that you did not totally create yourself (especially code from companies that employ people like me). No amount of source-level verification or scrutiny will protect you from using untrusted code.
>
> Thompson, Ken

1.3 Computer Security vs Network Security

A computer network is a general architecture that allows computer systems to share information remotely. Network security is based on the same exact idea of preserving the CIA properties of information.
Who can be trusted over the network? Can you trust your own system? Can you trust the communication channel? Can you trust the destination system?

1.3.1 Possible attacker actions

A general attacker might:

- Infiltrate the communication in between hops: impersonate the client, modify connections/routing, etc.
- Be/infiltrate one of the hops: act "legally" until end of service (after which it may act maliciously).

How do you know if any of this happened?

1.4 Attack models in network setting: outright malicious attacker

Typically the malicious attacker aims at reading or modifying the messages (in part or fully). That's a confidentiality, integrity, availability problem. In this contest, this attacker is typically called "man in the middle", or "man in the browser". Attacker can intercept and act upon a communication between client and server:

- Channel redirection
- Block communication entirely
- Spoofing the user's identity

Example: injection of malicious content

- Manipulation of server response: client's answer can also be modified by the attacker
- Connection Hijack: attacker injects him/herself in the communication and spoofs the victim's identity

1.5 Attack models in network setting: honest-but-curious attacker

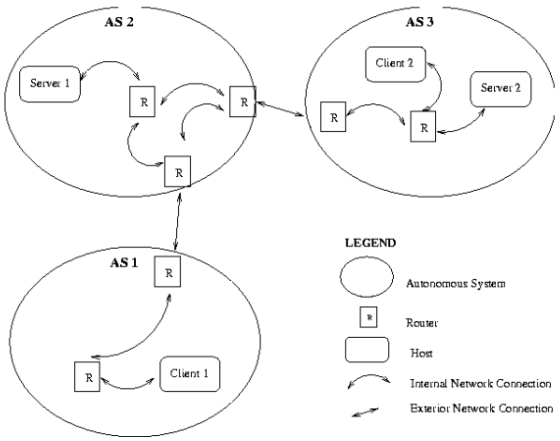
The goal of this attacker is to use the client's information after correctly handling the service. Typically resides at the service level (e.g. ISP) and typically implies confidentiality and possibly integrity threats.
Example: DB Server is the attacker. Provides agreed service correctly (e.g. answers queries with correct data). After the query is delivered to the client, the server uses the query's information to perform user profiling.

2 Network aspects

Internet is made of several logically separated networks: the **Autonomous Systems** (**AS**). Each AS autonomously manages communications within itself:

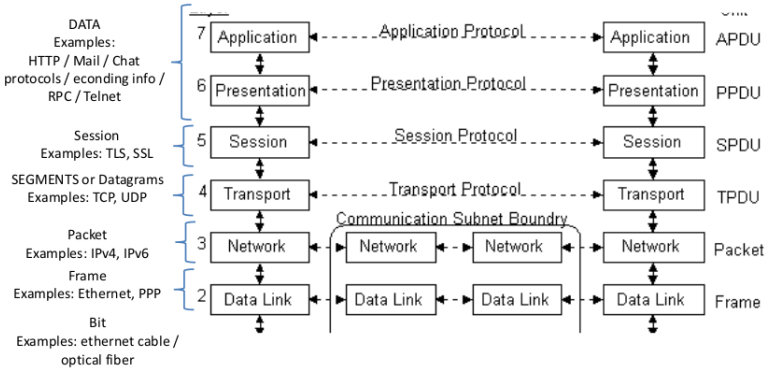
- Interior Gateway Protocols (IGP): route within each AS.
- E.g. two commonly used IGPs are the Routing Information Protocol (RIP) and the Open Shortest Path First (OSPF) protocol.

Each AS can communicate to other ASs: the Exterior Gateways Protocols (EGP) are the route between them (Border Gateway Protocol).



OSI model

Communication can be divided into **layers**:



2.1 OSI Data link layer

It is the lowest "logical" level, where data link interconnects physical interfaces. Each physical interface is identified by a MAC address:

- "Ethernet address"
- 48-bit Network interface identifiers
- Closest representation of final destination of a frame
- HEX notation: HH-HH-HH-HH-HH-HH (H = 16 = 2^4, so 4 bits for each H)
- Used to route packets in local networks

MAC addresses uniquely identify a network interface and are assigned by the producer according to the system command to list net interfaces standard IEEE 802. The first 24 bit are set by IEEE standard and identify the network interface producer.

2.2 OSI Network layer

Provides information on how to reach other systems (addressing functionalities). The IP operates at this layer: it is an high-level representation of a host's address and conveys information to route the datagram. IPv4 was defined in RFC 791. Most IP addresses are dynamically assigned by an authority (e.g. ISP's DHCP server), as opposed to MAC addresses that are fixed by the vendor. It is a "connectionless" protocol (stateless): there is no notion of "established connection" at this stage, it only provides the means necessary for a packet to reach its destination.

Stateless vs Stateful

A communication is made of a number of messagges. Communications start, develop, and end. Stateful protocols provide means to establish and close a connection (e.g. TCP). Stateless protocols do not have this notion (IP messages are stand-alone packets).

IP addresses

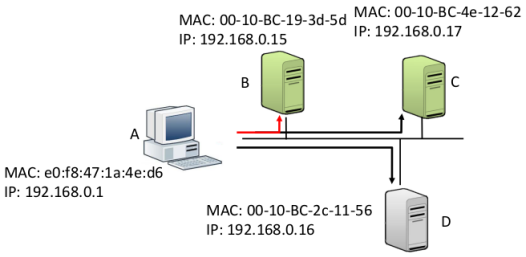
IP provides a structured way to abstract host addresses away from their physical properties. There exist two versions currently:

- IPv4: most common, currently used (32 bits)
- IPv6: early adoption, will be seen commonly in the future (128 bits)

Makes it possible to efficiently talk between systems in different ASs.

ARP protocol

The ARP (Address Resolution Protocol) allows systems to associate an IP address to a MAC address. It allows discovery through broadcast. ARP tables contain information to translate IP addresses into MAC addresses.



ARP tables A → B:

IP address	MAC address	... (e.g. TTL, interfaces..)
192.168.0.15	00-10-BC-19-3d-5d	...
192.168.0.17	00-10-BC-4e-12-62	...

ARP tables A → C:

IP address	MAC address	... (e.g. TTL, interfaces..)
192.168.0.15	00-10-BC-19-3d-5d	...
192.168.0.17	00-10-BC-4e-12-62	...

ARP tables A → D:

IP address	MAC address	... (e.g. TTL, interfaces..)
192.168.0.15	00-10-BC-19-3d-5d	...
192.168.0.17	00-10-BC-4e-12-62	...
???		

ARP query

All addresses in an ARP table are added by one of two mechanisms.

- ARP request-reply:
  - who is 192.168.0.16 tells 192.168.0.1
  - 192.168.0.16 is at 00-10-BC-2c-11-56
- Gratuitous ARP
  - 192.168.0.16 is at 00-10-BC-2c-11-56

The discovery process happens through queries to neighbor devices: broadcast message to the desired IP (e.g. L2 ethernet address FF-FF-FF-FF-FF-FF). The system with the requested IP replies back with its mac address.

ARP tables A → D:

IP address	MAC address	... (e.g. TTL, interfaces..)
192.168.0.15	00-10-BC-19-3d-5d	...
192.168.0.17	00-10-BC-4e-12-62	...
192.168.0.16	00-10-BC-2c-11-56	

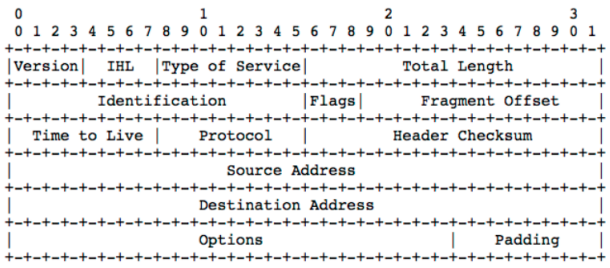
2.2.1 ARP poisoning

ARP answers or Gratuitous ARP frames do not require an (additional) answer/confirmation: it's a declarative protocol. Nodes are not authenticated, whomever can say I am x.x1.x2.x3, my mac address is hh.hh1.hh2.hh3.hh4.hh5.
C can tell to B "D is at [C mac address]" and to D "B is at [C mac address]". As a result every communication between B and D will pass by C.

ARP poisoning - limitations

It works only on local networks, where MAC addresses are actually meaningful. When communication is targeted to different network, IP addresses are used. Routers and DNSs have MAC addresses too. The poisoning works because systems are not authenticated; some implementations/third party tools can mitigate the problem (check for anomalies).

IP header



Subnets and CIDR

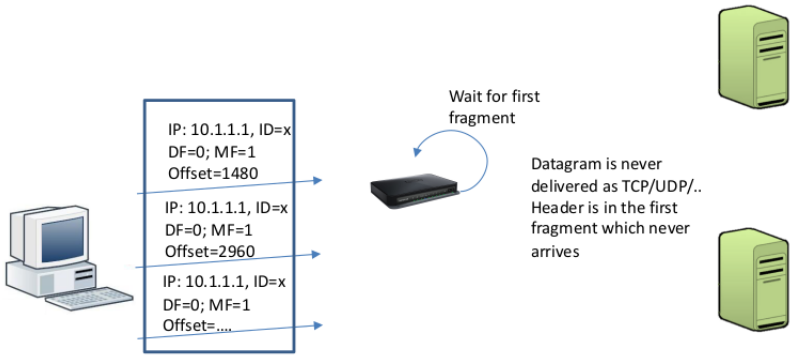
Subnets are logical divisions of IP addresses that make possible to split a network in multiple sub-networks. IP bits are divided in x **network bits**, y **subnet bits** and z **host bits**. Subnet mask indicates sections of IP addresses meant for network+subnet. 255.255.255.0 allocates 24 bits to network+subnet, 8 bits to hosts.
CIDR is a synthetic way to represent subnet masks, using classless Inter-Domain Routing. It indicates the number of bits covered by the mask:

$$192.168.10.1/24 = 192.168.10.1/255.255.255.0$$

IP Fragments

- Identification, 16 bit: unique identifier of the fragmented datagrams. All fragments have the same identification number.
- Flags, 3 bit
  - 0: Reserved, must be 0
  - DF: Don't fragment (0 = there may be fragments, 1 = don't fragment. If must be fragmented, drop datagram).
  - MF: More fragments (0 = last fragment, 1 = there are more fragments.
- Offset, 13 bits: offset of this datagram w.r.t first fragment with that ID.

2.2.2 Denial of service with IP fragments



Internet Control Message Protocol

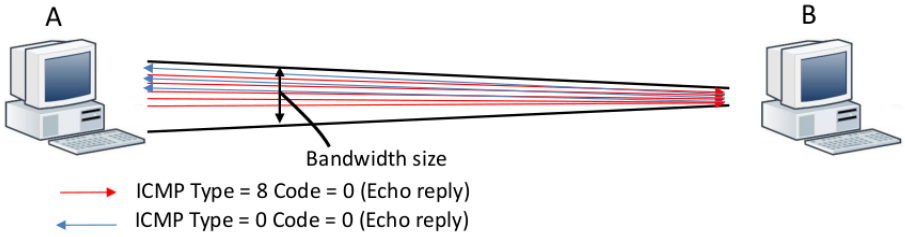
It was defined in RFC 792 and relies on IP. However, it is an integral part of the Internet Protocol. It is used by network devices to send error messages and information. All IP modules must have ICMP support.

2.2.3 Denial of Service

Denial of service (DoS) is a type of attack that aims at congesting or overpowering a system's capacity by generating requests the system will have to answer. It can affect the performance of the attacked system or its channels and lead to a system crash due to resource consumption. DoS can be operated locally or over the network.

A simple DoS (Ping Flood)

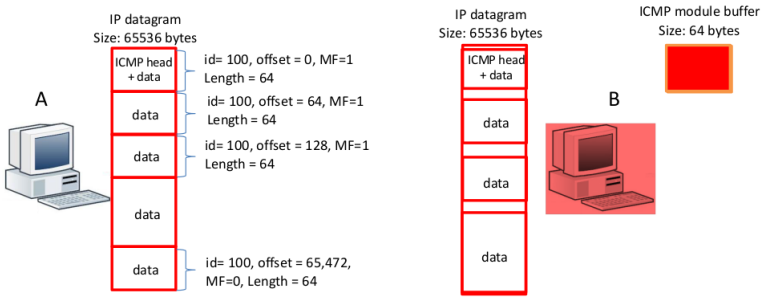
Network DoS attacks usually exploit protocol features.



A can exploit its wider bandwidth to flood B with ICMP echo requests. B's bandwidth gets (quickly, relatively to A's) exhausted with A's requests and B's replies; B can no longer operate on its network channel.

A more advanced DoS (Ping of Death)

ICMP packets are typically 64 bytes in size including IP headers and data IP datagram can extend up to 65,535 bytes. Data Length field is 16 bit. Early implementations of Internet modules were strictly implementing RFC directives, so not handling exceptions properly. The idea behind the Ping of Death is to generate large ICMP packet, and fragment them in 1024 IP packets of 64 Bytes. The destination receives regular packet, IP module compose fragments and ICMP module tries to read datagram bigger than assigned buffer size. The destination crashes because of "buffer overflow" (possible execution of code in memory).



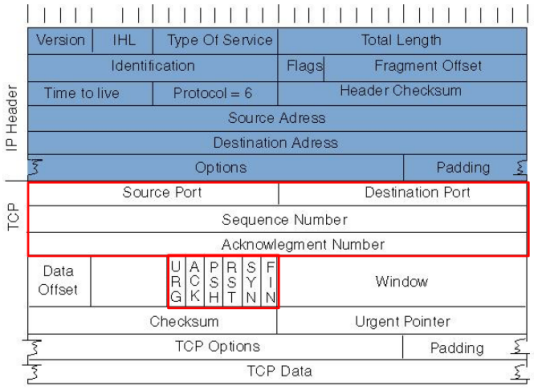
2.3 OSI Transport Layer

There are 2 protocols: TCP and UDP.

2.3.1 Transmission Control Protocol (TCP)

IP can only be used to send datagrams, "chunks" or "streams" of information, from sender IP to destination IP. TCP builds on top of IP the notion of "state": systems that communicate using the TCP protocol engage in a stateful communication. So, while IP delivers the data, TCP manages the data segments using checksums, re-delivery of unreceived or corrupt packets and more.

TCP/IP header



TCP is based on IP. Server and client that participate in a TCP connection open a "socket":

- SOURCEIP:SOURCEPORT
- DESTIP:DESTPORT

A connection between a client and a server is identified by the tuple

```
<SOURCEIP:SOURCEPORT, DESTIP:DESTPORT >
```

All TCP packets are directed toward a port. The common dest ports are port 22 (SSH), port 80 (HTTP), port 443 (HTTPS) and port 21 (FTP). Client usually generates source port randomly.

- LISTEN: service listening on port (open)
- CLOSE: no service listening on port (closed)

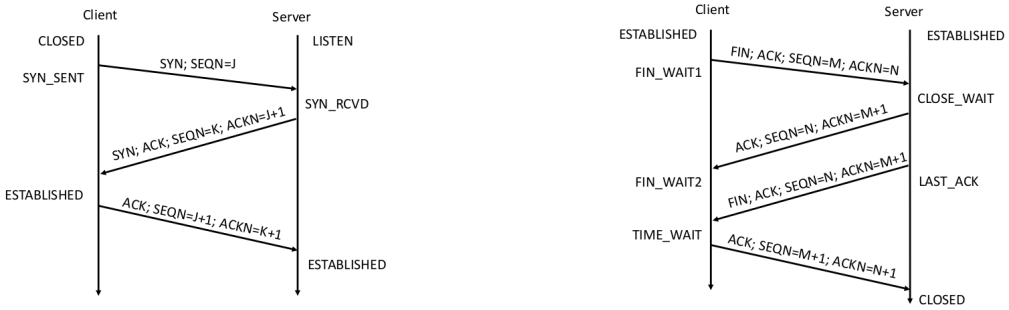
Some flags in the header are:

- SYN: initialize the TCP session, should be set to 1 only for first datagram by client and server;
- ACK: acknowledge the reception of the segment (associated with an ACK number)
- FIN: signals intention to close the connection (end of data)
- RST: connection is dropped (reset)

Other fields:

- Sequence number: 32 bit number generated by each end
  - communication start (SYN=1): `Client_seq = J / Server_seq = K`
  - during communciation: SEQN = "this is packet x"
- Acknowledgement number (32 bits) ACKN = "expecting x+1"

TCP 3-way handshake (SYN) and TCP 4-way handshake (FIN)



The first requires SYN only by one side, the second requires FIN by both.

Keeping track of TCP connections

The server receives a SYN request (`SYN_RCVD`), so it must keep track of this in order to establish a connection. Once it is ESTABLISHED, both ends set up a "Transmission Control Block" (TCB) to keep track of connection, which is a special data structure that stores information about connection (sockets, seq. numbers, pointers to buffer in memory). Then they allocate memory buffer to store data that will arrive. The TCB structure is freed from memory when connection reaches status CLOSED.

A packet with RST flag up does not receive an answer:

- CLOSED state: ANY packet with no RST receives a RST;
- LISTEN state: A packet with SYN flag up and no ACK opens a TCP session. Answer is SYN+ACK. A packet with only ACK receives a RST. Drop with no answer otherwise.

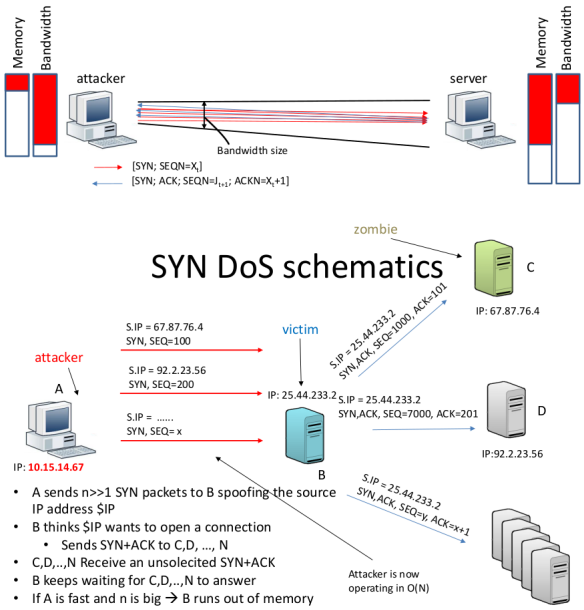
An unsolicited SYN+ACK gets a RST regardless of listening state.

2.3.2 SYN Denial of service attack

When the server receives a SYN J, it answers back with a SYN K and a ACK J+1. The server, at this moment, opens a new session in a separate thread, allocating resources (Transmission Control Block allocation). The server then waits for the ACK K+1 from client. A RST will be sent back after the amount of time set by the Maximum Segment Lifetime (MSL) is exceeded, which is set by default to 2 minutes. The same mechanism is used sender side: if the attacker controls the system, he may bypass it.

SYN Flood DoS, naïve solution

Server typically has more bandwidth available than single client. Client can drop all SYN ACKs (e.g. with a firewall) to not exhaust its own memory, but throughput necessarily slows down by O(2N): for each SYN, you get a SYN ACK, so the bandwidth quickly decays. Server's memory must be exhausted before the throughput becomes insufficient.



Denial of service limitations

In theory this attack should not work. Why? B should receive a RST by each zombie, which would free TCB and avoid DoS. An attacker can choose destination IPs that do not reply and firewalls may simply drop the packet with no RST. Some IPs may actually not be in use: in theory this will generate an ICMP packet (host not reachable) and close the connection (RFC 1122: A Destination Unreachable message that is received MUST be reported to the transport layer). SYN packets must arrive at very high rates. Other more sophisticated techniques exist:

- Distributed Denial of Service (nowadays more common)
- Coremelt DoS (attacking network links)

DoS Mitigation (pointers)

- Load balancing: distribute traffic loads evenly
- Rate limiter: deny traffic above a certain rate of SYN/sec
- Proof of work: require source to solve a crypto puzzle before allocating resources to connection (however it requires protocol support)

Network scans

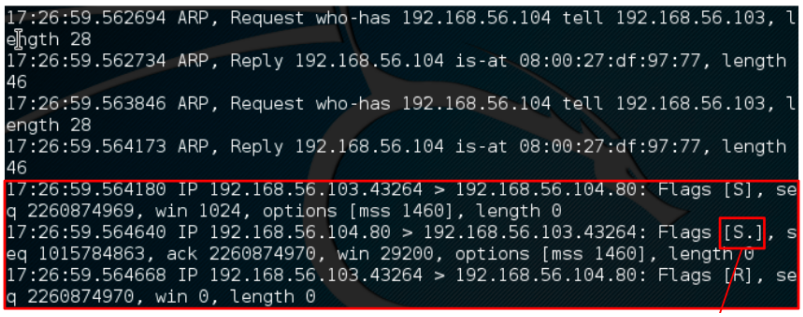
It's possible to exploit specifications of a network protocol (TCP, UDP,..) to learn something about a system or a network. For example build a list of services running on a remote system, infer a network's structure, build a list of zombie IPs that do not send RST back. There exists several types of scans and several popular tools to do one, like nmap.

SYN Scan

Attacker forges TCP packets (SYN=1). It is useful to measure whether remote system accepts incoming connections on port=x. Typically this corresponds to a specific service:

- SYN ACK from port 22: SSH is likely listening
- SYN ACK from port 80: HTTP server is likely listening
- RST: port x is closed on remote system

In a **half-open SYN scan**, after server's SYN ACK reply, the attacker sends RST and the 3-way handshake is never finished (example in the image).



Host fingerprinting

RFC 793 is the reference document for TCP stack implementation. However, not all specifications are always implemented as stated: different operating systems have their own independent implementation. It's possible to infer which operating system is on the other side on the basis of the received answers. This technique is called fingerprinting. An example of scan that allows for some level of fingerprinting is the **FIN/Xmas/Null scan**.

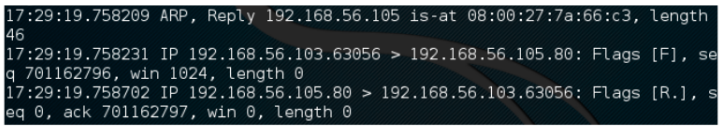
- FIN → flag FIN = 1
- Null → all flags = 0
- Xmas → FIN, URG, PSH = 1

From RFC:

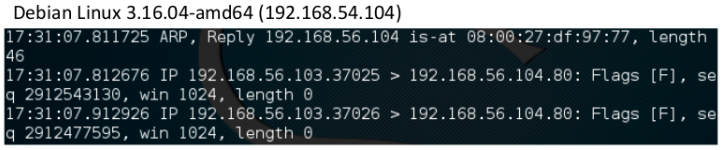
- Port is OPEN → DROP, no answer
- Port is CLOSED → DROP, RST

For example, Windows XP and HP/UX always reply with RST. Different hosts, different answers.

Windows XP 64bit sp0 (192.168.54.105)

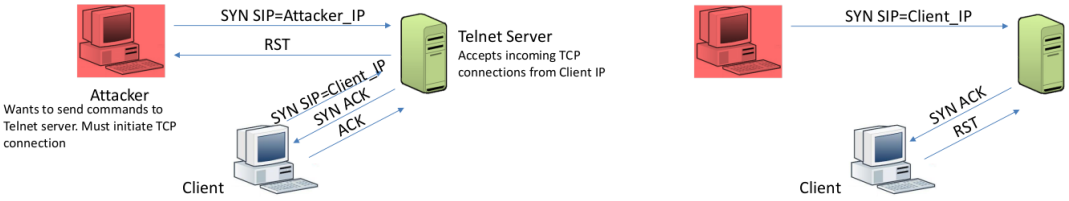


Debian Linux 3.16.04-amd64 (192.168.54.104)



2.3.3 TCP Session Hijacking

The goal is to to send commands to a server the attacker does not have access to: client is authorized (e.g. simple IP address authentication), the server must think that the attacker is the client but the attacker does not sit in between client and server.



How can he circumvent this? By pretending he is the client! A TCP segment between a client and a server is identified and validated by:

- Client IP: known (public)
- Destination IP: known (public)
- Port: known (public – if not standard, scan)
- Client SEQ number: known (attacker generates it)
- Server SEQ number: unknown (randomly generated by server and sent to `$CLIENT_IP`)

So it may be possible to predict the SEQ number. From RFC 793:

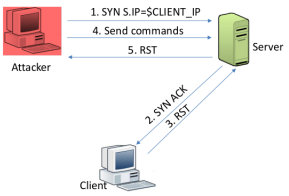
> When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds.

The **original BSD Unix** implementation was to increment by n units/second, and by n/2 units per new TCP connection. **Nowadays** implementations are (closer to) a random number generator.

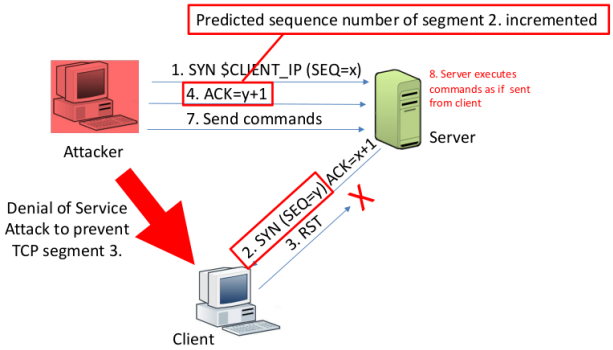
2.3.4 Mitnick attack

In order to impersonate the client, the attacker only needs to correctly guess the server's SEQ number ($1/2^{32}$ chances of getting it right). This is assuming perfect implementation of server's random number generator, in reality this may be much simpler ("TCP Sequence prediction").

The problem:



The solution (attacker anticipates Client's RST):



2.3.5 User Datagram Protocol (UDP)

Differently from TCP, UDP is a stateless protocol. It is faster to deliver data, while data integrity can be controlled at application level. It relies on reliability of underlying network link and does not guarantee delivery (no acknowledgment mechanism). UDP is used by some of the most important infrastructures of the Internet:

- DNS servers: to resolve internet domains
- NFS (Network File System): distributed FS
- SNMP (Simple Network Management Protocol): management of IP devices on a network
- DHCP (Dynamic Host Configuration Protocol): assign IP addresses to network devices
- Most real-time applications (real-time transactions, DBs, etc..)

UDP scans are interesting as many core services are running over UDP and listening to UDP ports. It can be used to discover (likely) open ports on the network:

- CLOSED = ICMP port unreachable
- OPEN = no answer

These scans are however prone to errors: ICMP packet can be filtered or dropped (firewalls/routers) and it is possible to configure a "stealth" system that does not reply to UDP requests to CLOSED ports.

2.4 OSI Session/Presentation/Application Layer

On top of IP, TCP, UDP, etc. there are a plethora of application-level protocols:

- FTP: file transfer
- SMTP/POP/IMAP: mail
- Telnet: remote access
- SSH: remote access
- HTTP: web
- DNS: infrastructure
- SSL/TLS: secure web

Pointless exercise to go through them all. Rather, we focus on some most important threats.

2.4.1 Domain Name Service

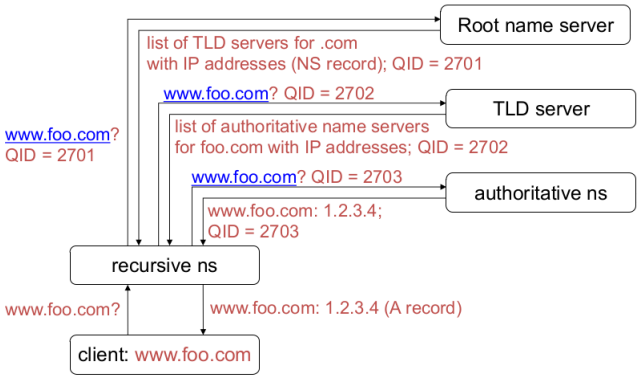
DNS is a hierarchical system for domain name resolving. It translates human-readable addressed (google.com) to (a set of) IP addresses the domain is reachable at. UDP is used for fast answers (port 53). Each transaction is identified by an ID (16 bits), the Query ID ("QID"). In the original DNS implementation there was an incremental QID. There are Several types of records. Of interest here:

- A (AAAA) $\rightarrow$ IPv4 (IPv6) of the requested domain (e.g. a.website.com A 65.61.198.201)
- NS $\rightarrow$ IP of the DNS server to ask (e.g. a.website.com NS ns.website.com), followed by an A answer for the DNS (ns.website.com A 2.2.2.2)

DNS hierarchy

Root DNSs are responsible for top level domain queries (e.g. .com NS ns.auth.net), **Authoritative** DNSs answer queries whose answer it already knows (does not ask to other DNSs) and **Recursive** DNSs forward queries to Authoritative DNSs. There are 13 root DNSs in the world.

Name resolution



Queries

- **recursive query**: DNS client requires DNS server respond with either the requested resource record, or an error message stating that the record or domain name does not exist.
- **iterative query**: contacted server replies with name of server to contact "I don't know this name, but ask this server XXX"

Cache & Time-to-live

Simplified description left out an important aspect. For reasons of performance optimisation: when name server receives an answer, it stores the answer in its cache. When receiving a request, name server first checks whether answer is already in its cache; if this is the case, the cached answer is given. The answer remains in cache until it expires, so the use of cache can be extensive; the time-to-live (TTL) of the answer is set by sender. Why is that? The server knows how often the website it is referring to usually changes. The security does not depend on the TTL: it all depends if the cached data is correct.

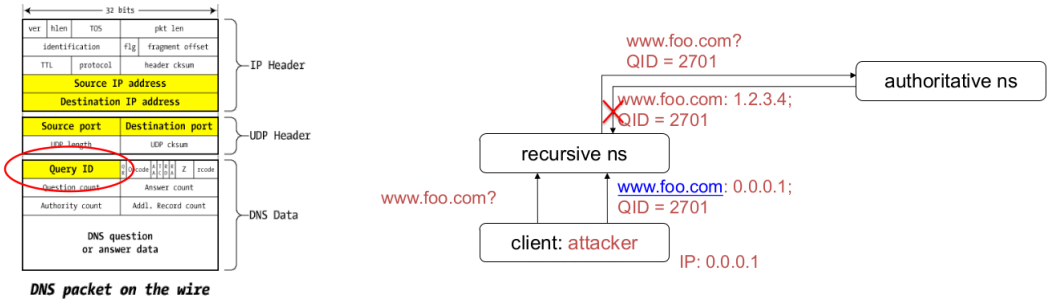
Light-weight Authentication

Anybody can pretend to be an authoritative name server for any zone. How does a recursive name server know that it has received a reply from an authoritative name server? Recursive name server includes a 16-bit query ID (QID) in its requests. Responding name server copies QID into its answer; applies it also to answer from authoritative name server. Recursive name server caches first answer for a given QID and host name; then discards this QID. Instead they drop answers that do not match an active QID.

If routing to and from root servers and TLD servers cannot be compromised, the attacker can only try to improve her chances of guessing a query ID. Some (earlier) versions of BIND used a counter to generate the QID.

2.4.2 Cache poisoning attack

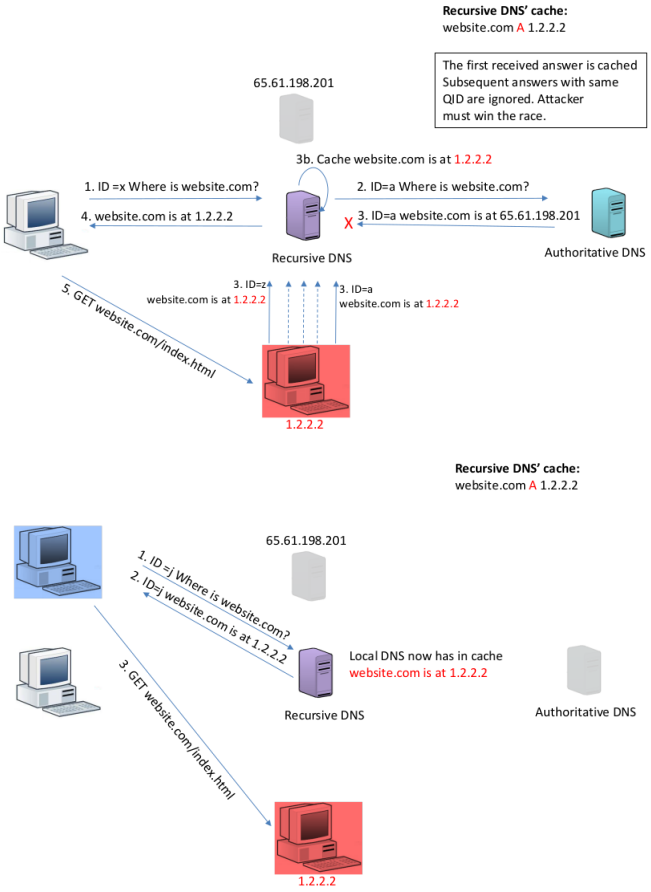
1. Ask recursive name server to resolve host name in attacker's domain.
2. Request to attacker's name server contains current QID.
3. Ask recursive name server to resolve host name you want to take over; send answer that includes next QID and maps host name to your chosen IP address.
4. If your answer arrives before the authoritative answer, your value will be cached; the correct answer is dropped.



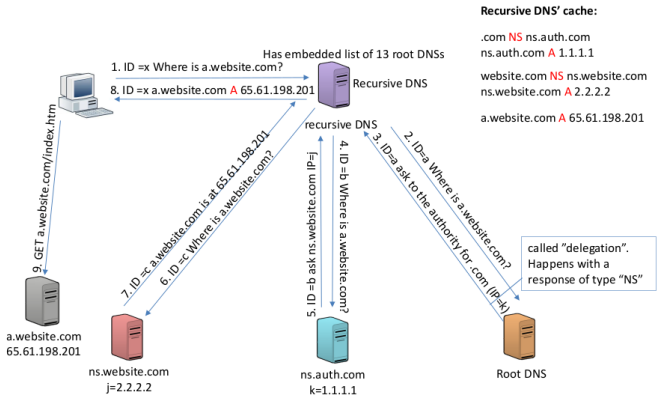
If you want to perform authentication without cryptography, **do not use predictable challenges**. More ways of improving the attack's chances:

- To account for other queries to the recursive name server concurrent to the attack, send answers with QIDs from a small window.
- To increase the chance that fake answer arrives before authoritative answer, slow down authoritative name server with a DoS attack.
- To prevent that a new query for the host name restores the correct binding, set a long time to live.

2.4.3 DNS cache poisoning

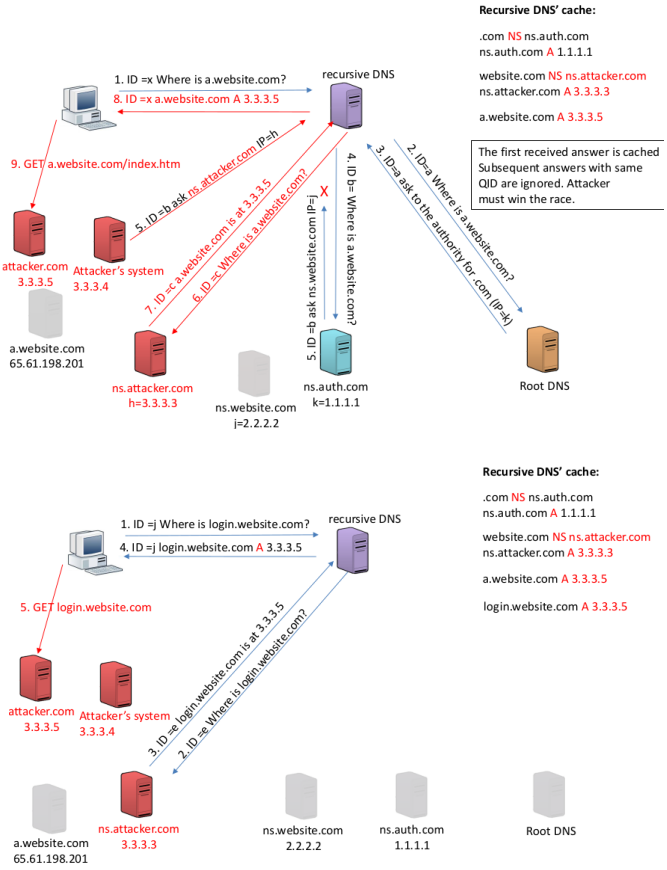


DNS full picture



2.4.4 Kaminsky vulnerability

The Kaminsky vulnerability can lead to a cache poisoning attack. The attacker rather than replacing an A record replaces an NS record: this way the attacker can get control over any (sub)domain.



Mitigation

Source of attack is low entropy with a 16 bit ID, the randomness is not enough to represent a significant margin. Moving ID size to 32 bits is not feasible, and it is not possible to change the protocol. The solution is to **randomize the source port** (16 bits) to increase entropy (in reality it is not possible to use all 16 bits for the source port because of reserved values). Any answer that does not match both source port and transaction ID will be dropped.

2.4.5 DNS amplification attack

It is a type of DoS attack which exploits certain type of DNS answers that are much bigger in size than the requests. The attack's throughput is much bigger than attacker's input. DNS works over UDP, so the source IP is easy to spoof.

2.4.6 DNS zone transfer

A zone is a domain for which a server is authoritative. "slave" servers can ask "authoritative" servers to copy their zone database (over TCP). An attacker pretends to be a slave server and dump the zone DB: it acquires knowledge of zone's infrastructure and can be used to facilitate further attacks (e.g. spoofing or more direct attacks).

2.4.7 DNSSec

It is the secure implementation of the DNS protocol. Implements DNS authentication on top of normal DNS exchange: digitally signed over a chain-of-trust starting from the root server, uses electronic certificates (Public-key crypto to authenticate by showing proof that you own a secret key). Protects data integrity (no confidentiality protection).

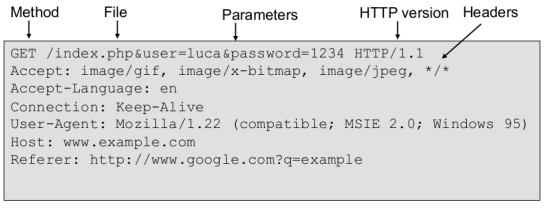
2.4.8 HTTP

Main protocol on which the www works. Based on the notion that client can either request or submit data to a server. There are two methods: GET (requests data from a specified resource) and POST (submits data to be processed to a specified resource). HTTP is stateless; statefulness is enabled with the use of cookies.

URLs

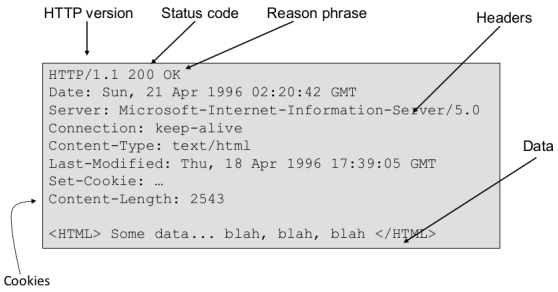
Global identifiers of network-retrievable documents. Special characters are encoded as hex (e.g., `%0A` = newline, `%20` = space, `%2B` = +).

GET request



POST request

HTTP response



Cookies

They are used to store state on user's machine, also for authentication.

2.4.9 HTTP session hijacking

Session ID is used by the webserver to authenticate client "victim". If the cookie is sent over in-the-clear, the attacker can read the session ID cookie and spoof the victim's identity (e.g. access to personal webpages/accounts).

The use of secure cookies provides confidentiality against network attacker (browser will only send cookie back over encrypted channels) but no integrity (still possible to rewrite secure cookies over HTTP).

2.4.10 Telnet

Protocol used in remote control services. It is implemented through a virtual terminal that connects to systems. Operates over TCP port 23. Remote client can issue commands to server: plain-text commands, typically without authentication or channel encryption. No need to go through possible attacks here, just use SSH instead.

2.4.11 Common issues

Most of the network attacks seen so far have at least one of two issues common among most network problems:

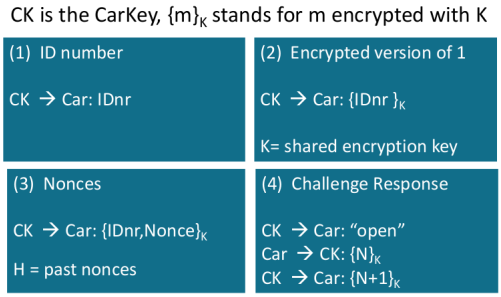
- Lack of authentication: the real sender/receiver of a packet/datagram cannot be authenticated (it is possible to spoof its identity);
- Communication channel is in the clear: a clever or well-positioned (in the network) attacker can read and potentially modify the information exchanged over the channel.

3 Security Protocols

The typical attacks to IPv4 are:

- Lack of confidentiality (stealing credentials)
- Lack of source authentication (spoofing, DoS)
- Source routing (spoofing and redirection)

Example of security protocols



Having an ID number to open the car is not safe because the message can just be copied and sent again; with the use of cryptography the problem does not change (can still just send again the message). With the nonces solution the Replay attack is avoided, however there still exists the possibility of doing a MITM attack.

"Secure Channels"

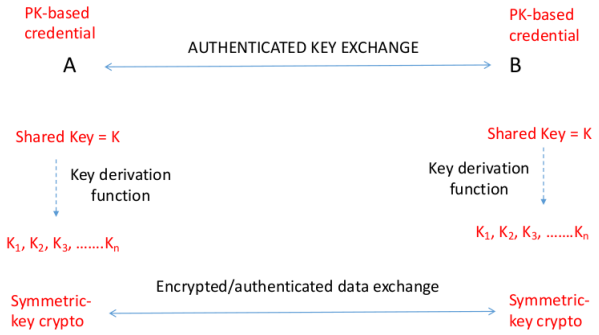
Protection is achieved by building a "secure channel" between two end points on an insecure network, typically offering:

- **Data origin authentication**;
- **Data integrity**;
- **Confidentiality**.

But usually not **non-repudiation** or **any services once data is received**.

Secure channel are usually built as follows:

- An authenticated key establishment protocol, during which one or both parties is authenticated and a fresh, shared secret is established.
- A key derivation phase, where MAC & bulk encryption keys are derived from shared secret.
- Then further traffic protected using derived keys: MAC gives data integrity mechanism and data origin authentication, encryption gives confidentiality.
- Optional: session re-use, fast re-keying, ...



Typical Cryptographic Primitives

- Symmetric encryption algorithms (for speed);
- MAC algorithms (usually built from hash functions, also fast);
- Asymmetric encryption and signature algorithms, Diffie-Hellman (for entity authentication and key exchange);
- (Keyed) pseudo-random functions (for key derivation);
- MAC-protected sequence numbers to prevent replay attacks;
- Nonces and timestamps for freshness in entity authentication exchanges.

IP Security Objectives

- Application level: transparent to applications and users (below transport layer);
- Host Level: provide security for individual hosts;
- Router Level: router or neighbor advertisements come from authorized routers, redirect message come from routers to which the initial packet was sent. A routing update is not forged.

3.1 IPSec

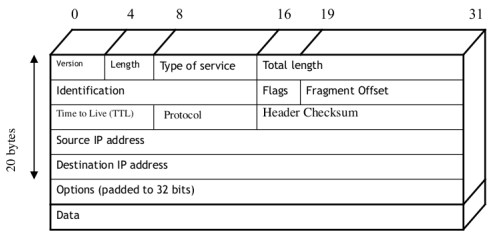
IPSec is a set of security protocols, a general framework that allows a pair of communicating entities (IP addresses!) to choose the appropriate crypto for the communication. Its service enables connectionless integrity, data origin authentication, rejection of replayed packets, confidentiality (encryption) and limited traffic flow confidentiality.

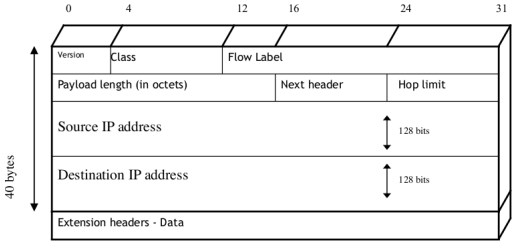
There are 2 basic mode of use:

- **"Transport" mode**: for IPsec-aware hosts as endpoints.
- **"Tunnel" mode**: for IPsec-unaware hosts, established by intermediate gateways or host OS.

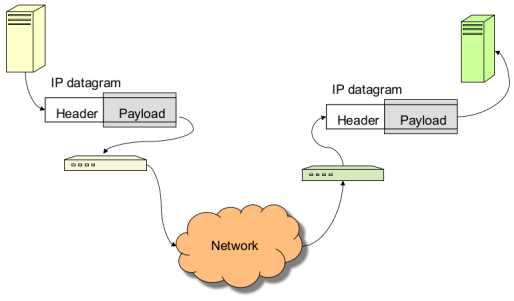
It provides authentication and/or confidentiality services for data (**AH** and **ESP** protocols) and a flexible set of key establishment methods (IKE, **IKEv2**).

IPv4 and IPv6 Header





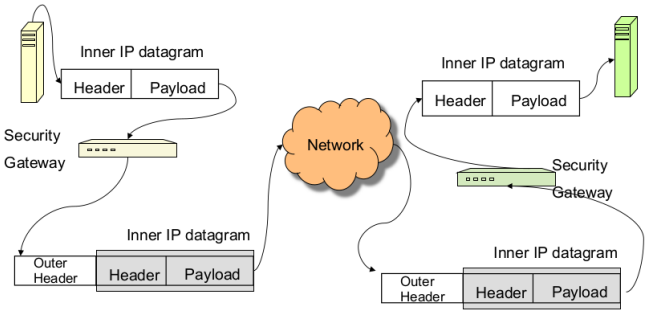
3.1.1 IPSec Transport Mode



Protection for upper-layer protocols.

- It covers IP datagram payload (and selected header fields); could be TCP packet, UDP, ICMP message, ...
- Host-to-host (end-to-end) security: IPsec processing is performed at the endpoints of secure channel. These endpoint hosts must be IPsec-aware.

3.1.2 IPsec Tunnel Mode



Protection for the entire IP datagram (Header + Payload).

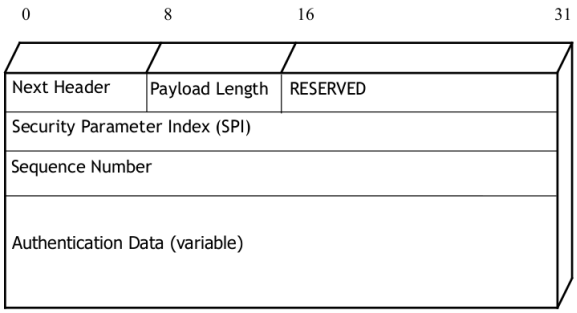
- The Entire datagram, plus the security fields, is treated as new payload of 'outer' IP datagram. The original 'inner' IP datagram encapsulated within 'outer' IP datagram.
- IPsec processing performed at security gateways on behalf of endpoint hosts.
  - Gateway could be perimeter firewall or router.
  - Gateway-to-gateway rather than end-to-end security.
  - Hosts do not need to be IPsec-aware.
- Inner IP datagram not visible to intermediate routers: even original source and destination addresses are encapsulated, and so 'hidden'.

Protocols

- AH: **Authentication Header** for authentication and integrity;
- ESP: **Encapsulating Security Payload** for confidentiality and authentication.

3.1.3 AH Protocol

- Provides data origin authentication and data integrity. AH authenticates whole payload and most of header.
- Prevents IP address spoofing. Source IP address is authenticated.
- Creates stateful channel (use of sequence numbers).
- Prevents replay of old datagrams (AH sequence number is authenticated).
- Uses MAC and secret key shared between endpoints.

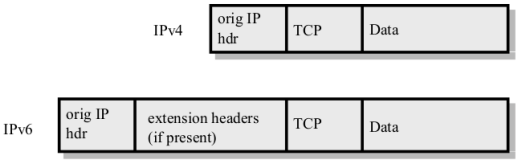


AH specifies a header added to IP datagrams. The fields of the header include:

- Payload length;

- SPI (Security Parameters Index): identifies which algorithms and keys are to be used for IPSec processing (more later);
- Sequence number;
- Authentication data (the MAC value): calculate over immutable IP header fields (so omit TTL) and payload or inner IP datagram.

Before applying AH



Transport mode and Tunnel mode AH authentication

3.1.4 ESP Protocol

Provides one or both:

- Confidentiality for payload/inner datagram; sequence number not protected by encryption.
- Authentication of payload/inner datagram; but not of any header fields (original header or outer header).

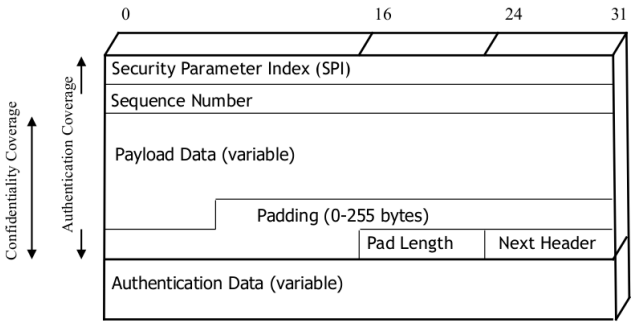
Traffic-flow confidentiality in tunnel mode. Uses symmetric encryption and MACs based on secret keys shared between endpoints.

ESP specifies a header and trailing fields to be added to IP datagrams. Header fields include:

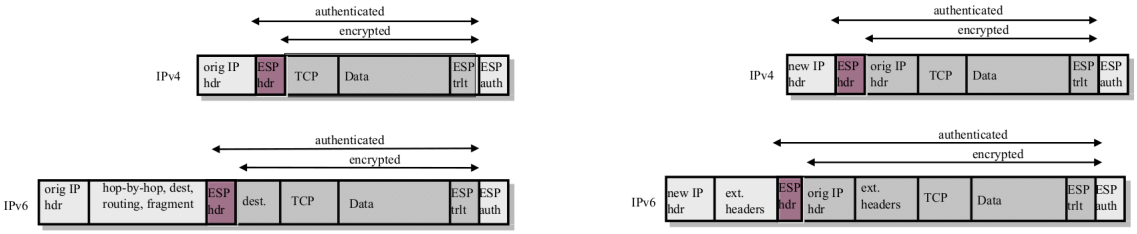
- SPI (Security Parameters Index): identifies which algorithms and keys are to be used for IPsec processing (more later).
- Sequence number.

Trailer fields include:

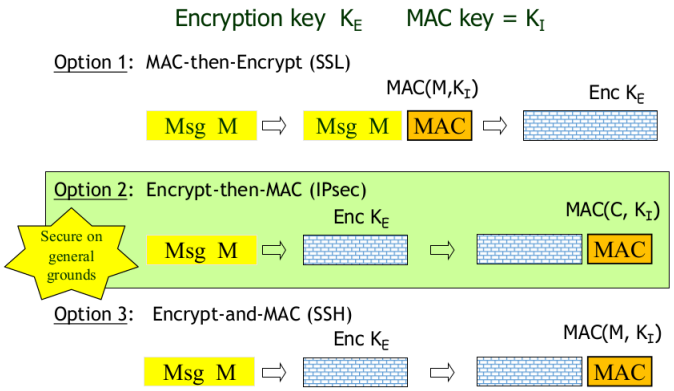
- Any padding needed for encryption algorithm (may also help disguise payload length).
- Padding length.
- Authentication data (if any) - the MAC value.



ESP Encryption and Authentication (Transport and Tunnel)



Combining MAC and ENC



IPSec uses 2 different protocols for encryption and MAC, so 2 different keys. Option 1 and 3 are secure but not in all configurations.

IPSec Key Management

IPSec is a heavy consumer of symmetric keys: one key for each SA is needed. Potentially, different SAs for every combination from:

{ESP,AH} x {tunnel,transport} x {sender, receiver} x {protocol} x {port}

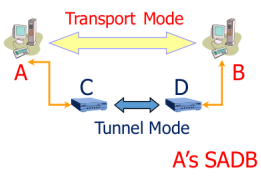
Where do these SAs and keys come from? Two sources:

- Manual keying: fine for small number of nodes and testing purposes, hopeless for reasonably sized networks of IPSec-aware hosts.
- IKE: **Internet Key Exchange**, RFC 2409: algorithms and parameters negotiation, protocols have many options and parameters (=complexity).

IKEv2 is the latest release, which addresses problems and complexities of IKE (i.e. DoS).
To exchange the keys over the network, IKE uses DH (Diffie-Hellman), applying over it signature and proof of knowledge to offer an authenticated DH (passing values with a signature).

SPD and SADB Example

Given two hosts, A and B, and two gateways, C and D: an end-to-end encrypted communication is on place. The communication between A and B (Transport Mode) is tunneled in Tunnel Mode. This way of operating generates and requires lots of keys, so A has two important databases: the **Security Associations DB** (**SADB**) and the **Security Policies DB** (**SPD**). These two are in every node that speaks IPSec.



A's SPD

From	To	Protocol	Port	Policy
A	B	Any	Any	AH[HMAC-MD5]

From	To	Protocol	SPI	SA Record
A	B	AH	12	HMAC-MD5 key

- A's SPD: when A communicates to B, no matter the protocol, the policy states that the traffic has to be authenticated using HMAC (with MD5 as block cipher);
- A's SADB: the communications from A to B using the AH protocol and the SPI 12 (which refers to the policy used in the SPD, so the HMAC-MD5) use the key in the SA Record.

This is not only host to host, it is applied also to routers.

From	To	Protocol	Port	Policy	Tunnel Dest
		Any	Any	ESP[3DES]	D

C's SPD

From	To	Protocol	SPI	SA Record
		ESP	14	3DES key

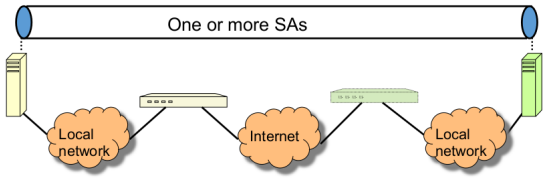
C's SADB

- C's SPD: always, no matter the source of destination and the protocol, the policy is to use ESP with 3DES, with as the tunnel destination D;
- C's SADB: protocol ESP with SPI 14 (3DES) uses the key in the SA Record.

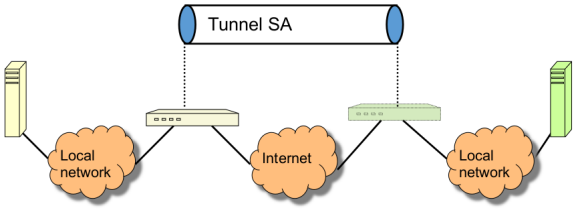
Required SA Combinations

Transport mode and Tunnel mode can be combined in different ways. Some typical combinations:

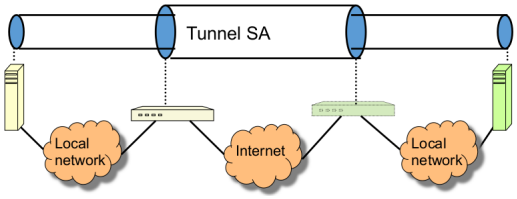
1. End-to-end application of IPsec between IPsec-aware hosts; one or more SAs, with one of the following combinations:
   - AH in transport;
   - ESP in transport;
   - AH followed by ESP, both transport;
   - Any of the above, tunnelled inside AH or ESP.



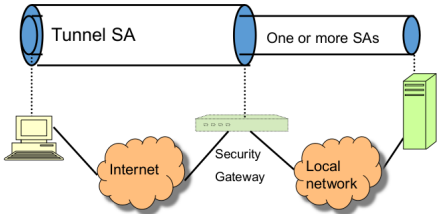
2. Gateway-to-gateway only (typical configuration between two routers):
   - No IPsec at hosts;
   - Simple Virtual Private Network (VPN);
   - Single tunnel SA supporting any of AH, ESP (conf only) or ESP (conf+auth).



3. A combination of 1 and 2 above:
   - Gateway-to-gateway tunnel as in 2 carrying host-to-host traffic as in 1;
   - Gives additional, flexible security on local networks (between gateways and hosts);
   - E.g., ESP in tunnel mode carrying AH in transport mode.



4. Remote host support:
   - Single gateway (typically firewall);
   - Remote host uses Internet to reach firewall, then gain access to server behind firewall;
   - Traffic protected in inner tunnel to server as in case 1 above;
   - Outer tunnel protects inner traffic over Internet.



Final Notes on IPSec

IPSec and firewalls have problems working together: authentication of source IP addresses in AH is the issue, some firewalls change these addresses on out-bound datagrams (NAT). IPSec support for ICMP is somewhat complicated. Managing IPSec policy and deployments is tricky: getting it wrong can mean losing connectivity, e.g. by making exchanges of routing updates unreadable and can mean loss of security. There are many, many IPSec options with a rather poor documentation.

HTTP

As explained in 2.4.8.

3.2 TLS

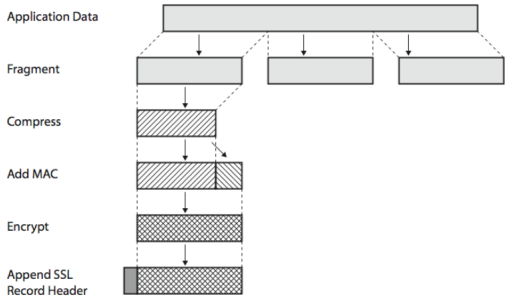
Protocol designed to secure HTTP, later became a protocol to secure any other application protocol. History:

- SSL 1.0: Internal Netscape design, early 1994;
- SSL 2.0: Published by Netscape, November 1994, several problems;
- SSL 3.0: Designed by Netscape and Paul Kocher, November 1996;
- TLS 1.0: Internet standard based on SSL 3.0, January 1999, not interoperable with SSL 3.0;
- TLS 1.1: RFC 4346 in April 2006, explicit IV and fixing the padding procedure;
- TLS 1.2: RFC 5246 in August 2008 and RFC 6176 in March 2011, adding cipher algorithms (e.g., AES). Prevents downgrading attack;
- TLS 1.3: RFC 8446 in August 2018, adding cipher algorithms (e.g., ECC). Removes obsolete and insecure features.

TLS is a set of protocol that tunnels HTTP (HTTPS, HTTP over TLS) and uses the reserved port 443. These protocols are **Record**, **Handshake**, **Change Cipher Spec** and **Alert**.

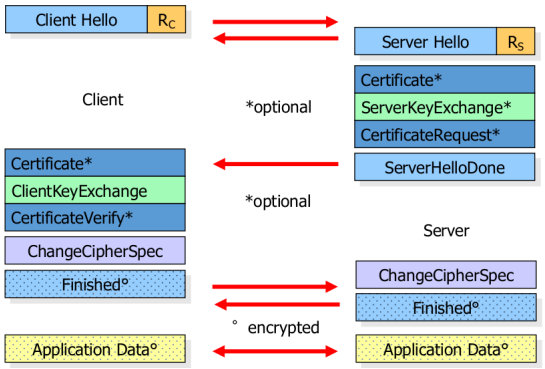
TLS Record Protocol Operation

Protocol that encrypts the traffic using symmetric key. The application data is fragmented into chunks, which are compressed. A MAC is added, and both the data and the MAC are encrypted and appended in the Record header.



TLS Handshake Protocol

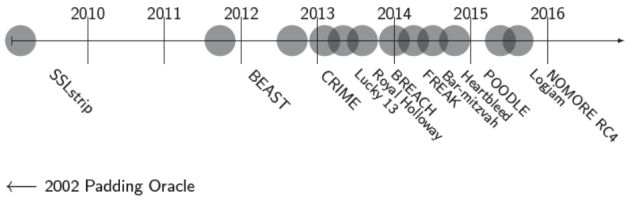
Initializes the connection, and uses the other two protocols: Change Cipher Spec to negotiate algorithm and key size (used also to resume a connection), Alert to deal with error messages.



Set of protocols

Cipher		Protocol Version					
Algorithm	Strength (bits)	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
AES GCM	256, 128	N/A	N/A	N/A	N/A	Secure	Secure
AES CCM		N/A	N/A	N/A	N/A	Secure	Secure
AES CBC		N/A	N/A	Depends	Secure	Secure	N/A
Camellia GCM	256, 128	N/A	N/A	N/A	N/A	Secure	Secure
Camellia CBC		N/A	N/A	Depends	Secure	Secure	N/A
ARIA GCM	256, 128	N/A	N/A	N/A	N/A	Secure	Secure
ARIA CBC		N/A	N/A	Depends	Secure	Secure	N/A
SEED CBC	128	N/A	N/A	Depends	Secure	Secure	N/A
3DES EDE CBC	112	Insecure	Insecure	Low/Dep.	Low	Low	N/A
GOST CNT	256	N/A	N/A	Secure	Secure	Secure	N/A
IDEA CBC	128	Insecure	Insecure	Depends	Secure	N/A	N/A
DES CBC	40	Insecure	Insecure	N/A	N/A	N/A	N/A
	56	Insecure	Insecure	Insecure	N/A	N/A	N/A
RC2 CBC	56	Insecure	Insecure	Insecure	N/A	N/A	N/A
ChaCha20-Poly1305	256	N/A	N/A	N/A	N/A	Secure	Secure
RC4	40	Insecure	Insecure	Insecure	N/A	N/A	N/A
	128	Insecure	Insecure	Insecure	Insecure	Insecure	N/A
NULL	–	N/A	Insecure	Insecure	Insecure	Insecure	Insecure

Attacks



TLS does not implement padding (adding data to the beginning, middle, or end of a message prior to encryption): this means that informations may be leaked from the size of the data.

A common requirement for an attack is that the attacker can sniff victim's network traffic. Listening on every possible switch is unfeasible, better at endpoints (browsers). Some attacks require to visit "evil.com", the victim clicks on a link or she surfs a non-HTTPS site (can be HTTPS if the attacker owns it).

3.2.1 Heartbleed (CVE-2014-0160)

Heartbleed is a security bug disclosed in April 2014 in the OpenSSL cryptographic library, which is a widely used for the implementation of the Transport Layer Security (TLS) protocol. Heartbleed may be exploited regardless of whether the party using a vulnerable OpenSSL instance for TLS is a server or a client. It results from improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension, thus the bug's name derives from "heartbeat". The vulnerability is classified as a buffer over-read, a situation where more data can be read than should be allowed.

OpenSSL is an open-source implementation of the SSL and TLS protocols. The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions. The OpenSSL project was founded in 1998 to provide a free set of encryption tools for the code used on the Internet. As of 2014 two thirds of all webservers use it.

The Heartbeat Extension provides a new protocol for TLS allowing the usage of keep-alive functionality without performing a renegotiation. TLS is based on reliable protocols, but there is no feature available to keep the connection alive without continuous data transfer. The Heartbeat Extension overcomes these limitations. The user can use the new HeartbeatRequest message, which has to be answered by the peer with a HeartbeartResponse immediately.
The Heartbeat protocol is a new protocol running on top of the Record Layer. The protocol itself consists of two message types: **HeartbeatRequest** and **HeartbeatResponse**. A HeartbeatRequest message can arrive almost at any time during the lifetime of a connection. There must not be more than one HeartbeatRequest message in flight at a time. A HeartbeatRequest message is considered to be in flight until the corresponding HeartbeatResponse message is received, or until the retransmit timer expires.

```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

The total length of a HeartbeatMessage must not exceed 2^{14} Bytes.

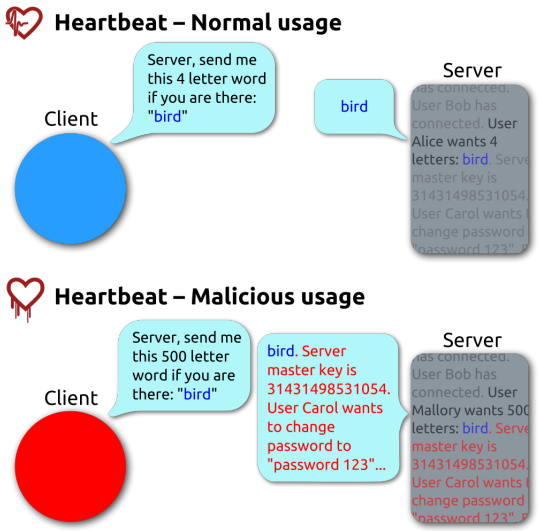
- `type`: The message type (either `heartbeat_request` or `heartbeat_response`);
- `payload_length`: The length of the payload;
- `payload`: The payload consists of arbitrary content;
- `Padding`: It can be any random content and the sender of this message must use at least 16 bytes of padding.

When a HeartbeatRequest message is received, the receiver must send a corresponding HeartbeatResponse message carrying an exact copy of the payload of the received HeartbeatRequest. If a received HeartbeatResponse message does not contain the expected payload, the message must be discarded. If it does contain the expected payload, the retransmission timer must be stopped.
As said earlier, this attack was made possible because of the improper input validation of the TLS Heartbeat extension. The vulnerability lies in the variable payload. Ideally the code must check the payload data length with the actual length of data sent in the Heartbeat request, but it isn't checking it. So if the payload is actually shorter the length specified in the request, the server may return more data in response than what it should ideally return. This is a case of Buffer Overflow.

An example of vulnerable code:

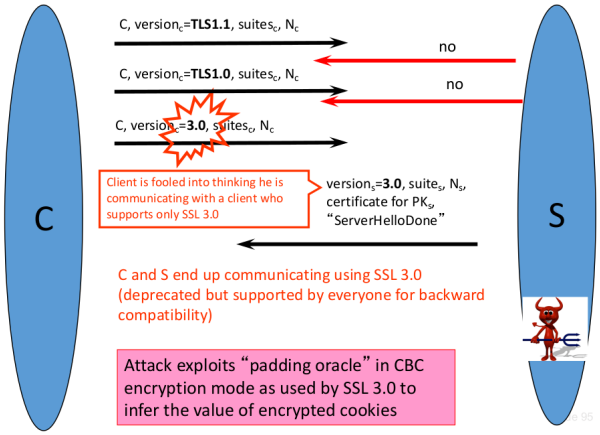
```
p = &s -> s3 -> rrec.data[0];
buffer = OPENSSL_malloc(1+2+payload+padding);
bp = buffer;
memcpy(bp, pl, payload);
```

The rrec contains all the incoming request data. The code reads the data. The first byte is to check if it's a Heartbeat protocol and then another 2 bytes determine the length of the Heartbeat payload. Ideally the length of the payload must be equal to the `payload_length` sent in the Heartbeat request. But as discussed, the code is not checking actual length sent in the Heartbeat request. So the code copies the amount of data requested by incoming requests to the outgoing server response and possibly more than requested in some cases.



This may leak valuable information to attackers, such as session IDs, tokens, keys, etc.

3.2.2 POODLE (CVE-2014-3566)



Padding adds those extra few bits which are necessary before encryption to make a meaningful block. Here we are interested to know the padding scheme PKCS#5 as it is used in the CBC (Cipher Block Chaining) Mode. In PKCS#5, the final block of plaintext is padded with N bytes of value N.
In Cryptography, an "oracle" is a system that performs cryptographic actions by taking in certain input. Hence a "padding oracle" is a type of system that takes in encrypted data from the user, decrypts it and verifies whether the padding is correct or not. Let us now try to understand how this attack is performed.

Example

Consider the below URL:

```
https://www.example.com/home.asp?
    UID=8A219A434525535FF324D4G56FC9534
```

Let us assume that some information is sent in this UID parameter (say username) in encrypted form using the CBC mode and PKCS #5 standard. So the application decrypts this value and returns the results based on that value. Three Scenarios are possible:

- Case 1) Valid cipher text - Valid and normal page;
- Case 2) Invalid cipher text [with improper padding] - Invalid page [such as 404 - Not Found];
- Case 3) Valid cipher text but invalid padding [error].

The MAC does not include the padding, so the server does not check the integrity of the padding but only that its length is correct. The server does not check if the padding has been modified.

- Case 1) Say you sent the value

  ```
  UID=8A219A434525535FF324D4G56FC95348
  ```

  and it decrypts to a valid user "Shreyas". Then the application would send a normal response.

- Case 2) Say you sent the value

  ```
  UID=998877PA434525535FF324D4G56FC95348
  ```

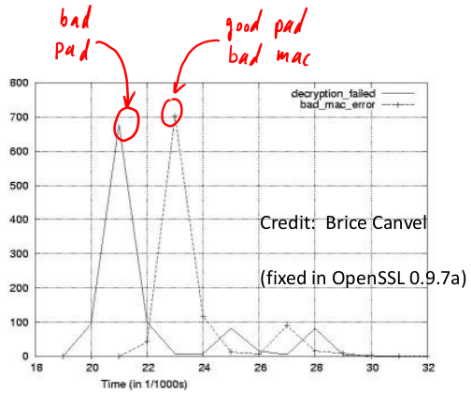
  and it decrypts to "aswjkaja" (invalid user). The application might respond back with a 404 message saying no such page exists.

- Case 3) Say you sent the value

  ```
  UID=66IXS7IA434525535FF324D4G56FC95348
  ```

  and it decrypts to "Ravi" (valid user) but with invalid padding. The application would return some exception.

Errors from case 2 and 3 were different, making possible to understand if decryption failed or if padding failed; so a first fix applied in TLS 1.0 was to make the Alert protocol send the same error in both cases. However that does not solve the problem: it was still possible to differentiate between the 2 studying the timing of the response (padding error answer in 21 milliseconds circa, encryption error in 23).



Padding Oracle

TODO

"Protocol Downgrading" Attack

Why do people release new versions of security protocols? Because the old version got broken! However new version must be backward-compatible, because not everybody upgrades right away. An attacker can fool someone into using the old, broken version and exploit known vulnerabilities or, similarly, fool victim into using weak crypto algorithms. Defense is hard: must authenticate version early. Many protocols had "version rollback" attacks (SSL, SSH, GSM).

A solution to the chosen ciphertext attack cannot just only to make alert message and time the same in both cases, but to always perform a MAC check as if zero-length padding is applied (RFCs). Encrypt-then-MAC avoids the problem because change is detected before decryption.

3.2.3 BEAST (CVE-2011-3389)

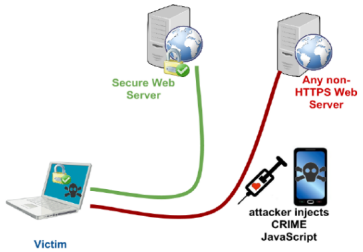
Another variant of attack to RC4 CBC implementation of SSL/TLS.

3.2.4 CRIME (CVE-2012-4929)

The CRIME attack exploits the compression algorithm of TLS. It is an acronym that stands for Compression Ratio Info-leak Made Easy. In the HTTP requests and responses there are some parts of the plaintext that are VERY predictable. Session cookies can be found on well-defined spots:

```
GET index.html HTTP/1.1
Host: thebankserver.com
(...)
Cookie: secret=7xc89f+94/wa
(...)
```

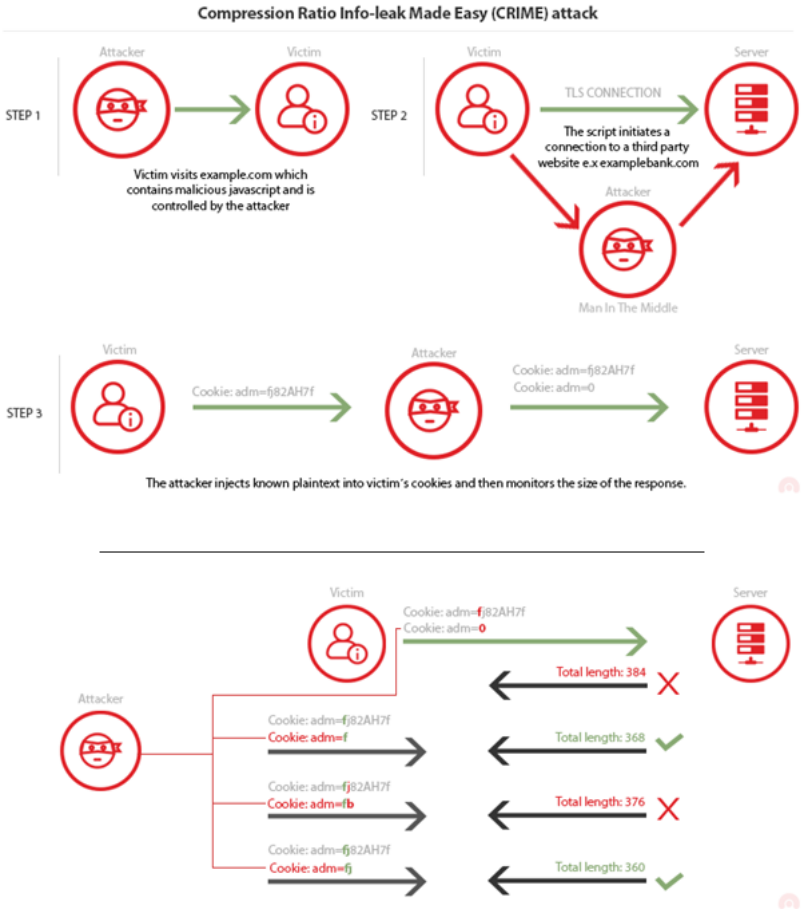
The attacker tries to send data from the client in the TLS context and see how this affects the ciphertext; it uses the server as an oracle, exploiting the response to "guess" the session cookie character by character. CRIME can also be done by injection, luring the victim to visit a non-HTTPS server or directly its website.



CRIME exploits the way in which compression algorithms work, so with the TLS algorithm which is DEFLATE.

The idea behind the attack is simple: an input, controlled by the attacker, is inserted and then a secret, and the total message is compressed. If there is no overlapping sequence between the input and the secret, the result will have a certain length; but if some pattern of the input is also present in the secret, after the compression the length will be different (usually smaller). By changing the input and measure the length it will be possible to guess the secret, because SSL/TLS doesn't hide request/response length.

```
len(compress(input + secret))
```



Algorithm:

1. Make a guess
2. Observe length of sent request
3. Correct guess is when length is different than usual

All of this has to happen on the clear, or on encrypted communication if webpage is owned by the attacker.

3.2.5 BREACH (CVE-2013-3587)

It is a variant of CRIME; it exploits the HTTP compression algorithm.

3.2.6 Conclusions

The TLS attacks use approaches that can work on other protocols:

- Exploiting vulnerability of the underlying cryptographic algorithm implementations
- Exploiting information leaked by compression algorithms
- Exploiting poorly written code (i.e., missing proper input validation)

4 Vulnerabilities

The previously seen attacks all exploited a vulnerability.

Software bug

A bug is a problem in the execution of the software that leads to unexpected behaviour: software crashes, wrong entries are displayed/stored in a backend database, execution loops infinitely.
The characteristics of a bug are:

- Replicability (if you can't repeat it it's not a bug);
- Classification: Logic/configuration/design/implementation;
- Fix priority (based on the damage they can inflict);
- It is not documented. If it is, it's a feature.

An example of a SW bug:

```
gets(password);
correct_pwd=lookup(username, database);
if (correct_pwd!=password)
    printf('Login failed');
    return err;
else{
    printf('login succeeded');
    exec(context);
}
return x;
```

The code does not behave as intended because of the missing parentheses at the first `if` clause. With an inverse order in the clauses, it would be a vulnerability: `exec(context)` is always executed.

Vulnerabilities

The definition from NIST SP 800-30 of a vulnerability is:

> A flaw or weakness in system security procedures, design, implementation, or internal controls that could be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system's security policy.

Vulnerabilities can be found at any level in an information system: **configuration**, **infrastructure**, **software**.

- **Configuration vulnerabilities**: software or system configuration does not correctly implement security policy (e.g. accept SSH root connections from any IP);
- **Infrastructure vulnerabilities**: design or implementation problems that directly or indirectly affect the security of a system (e.g. a sensitive database in a network's DMZ);
- **Software vulnerabilities**: design or implementation of a software module can be exploited to bypass security policy (e.g. authorisation mechanism can be bypassed).

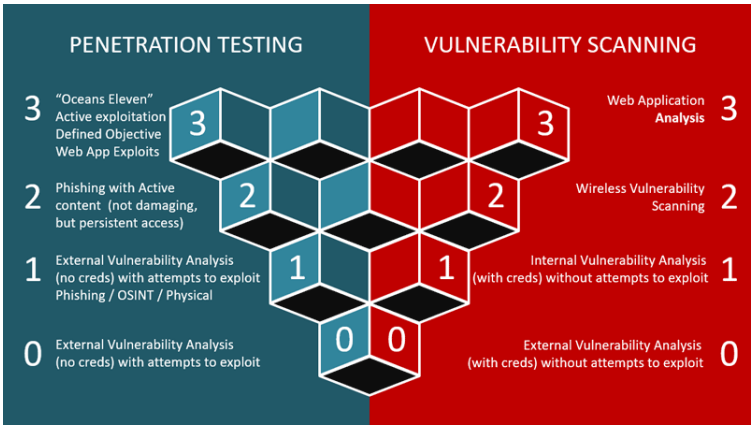
Thousands of vulnerabilities are discovered each year: some are publicly disclosed, others are not. The main publisher of discovered vulnerabilities is the NVD (National Vulnerability Database), managed by MITRE, a non-profit organisation (Massachusetts, U.S.A.) that supports, among others, activities from:

- Department of Homeland Security (DHS)
- Department of Defense (DoD)
- National Institute for Standards and Technology (NIST)

Moreover, it maintaines standard for vulnerability identification, the Common Vulnerabilities and Exposures (CVE). Each vulnerability has a CVE identifier.

Vulnerability discovery

There are activities that are performed periodically to discover vulnerabilities, like **vulnerability scanning** and **penetration testing**.



Moreover there are activities performed by individuals, like ethical hackers.
Vulnerabilities are widely different in nature: often implementation-dependent, may require deep understanding of SW module interaction and necessary in-depth knowledge of system design (e.g. kernel structure, memory allocation, ...). There are two main discovery techniques:

- Code lookups (you need source): manual/semi-automatic search in codebase for known patterns;
- Fuzzing: semi-automatic random input generation to try to crash a program;
- Bonus technique: "Google hacking" (look for known vulnerable functions in google, which returns vulnerable webpages).

Vulnerabilities can be found either internally or externally to a company:

- Internally: managed within the company (part of Q&A process), with a patch (fixing) prioritisation and communication to customers;
- Externally: found by an external security researcher, with disclosure to vendor (payment), patching prioritisation and disclosure to public.

Vulnerability handling

Internal process must accept information about new vulnerabilities (internal or external sources), verify vulnerability report and, if a vulnerability exists, develop a resolution and execute post-resolution activities. The standard ISO 30111 regulates all of this process, following different phases: verification, resolution and release.

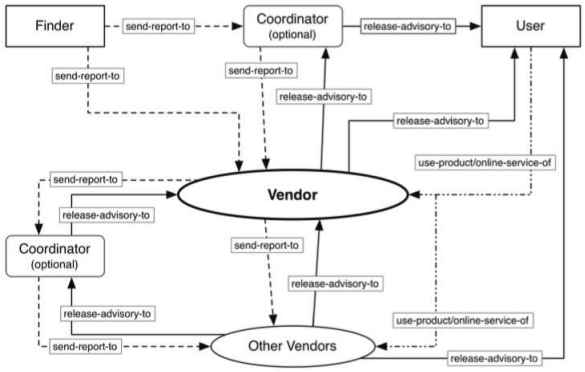
- Initial investigation (verification):
  1. The reported problem is a security vulnerability: must have repercussions over security policy;
  2. The vulnerability affects a supported version of the software the vendor maintains (e.g. not caused by 3 rd party modules). Else, exit process;
  3. The vulnerability is exploitable with currently known techniques. Else, exit process;
  4. Root cause analysis: underlying causes of vulnerability and look for similar problems in the code;
  5. Prioritisation: evaluate potential threat posed by the vulnerability.
- Resolution decision: vendor must decide how to resolve the vulnerability. There may be different decisions for different types of vulnerabilities:

1. Configuration vulnerabilities → advisory may be enough
2. Code vulnerabilities → patch
3. Critical vulnerabilities → release a mitigation before full patch

- Remediation development: every resolution must be tested before being delivered to clients to minimize negative impacts caused by software change.
- Release:
  - Web services → vendor deploys patch itself
  - Stand-alone product → patch release (see ISO 29147)
- Post-release: monitor situation (e.g. patch may not be always effective) and give support to the final client.

Vulnerability disclosure

Vulnerabilities are information sets and the vulnerability disclosure process is about information exchange (ISO 29147). The image shows how the **finder**, which is the individual that finds the vulnerability, and the **vendor** should behave according to this standard.



Basically the finder has to inform the vendor (directly or through a coordinator), which has to inform other vendors and its users.
There are also 2 standards for when vulnerabilities are exchanged, STIX and TAXII.

Confidentiality of vulnerability information

Vulnerability information is considered sensitive and confidential by vendors because an attack that poses a threat to end users may affect the vendor's reputation. So it is important to build secure communication channels to preserve confidentiality and integrity of information. Vulnerability advisories are typically published after patching: internal policies determine whether a vulnerability will be published or not (typically a function of vulnerability severity).

Security researcher that finds vulnerability may expect economic return and credit (to mention on curriculum). So how should one communicate a vulnerability to vendor?

- Say too little = vulnerability not reproducible and no money are obtained
- Say too much = vulnerability fully known, the vendor thanks for the info and gives no money

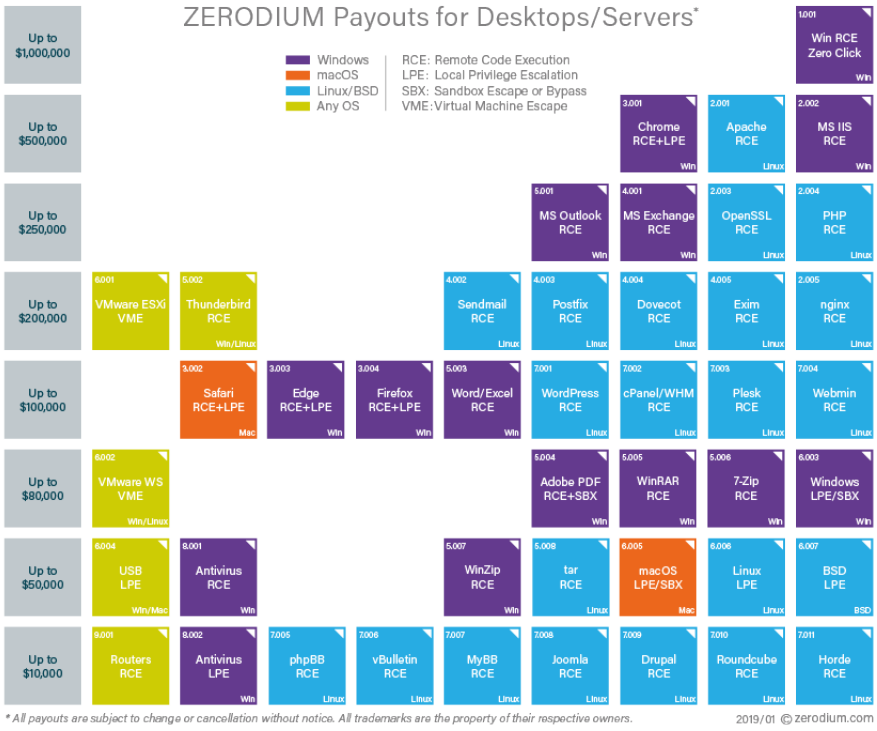
There has to be an agreement between sec researchers and vendors:

- Third party mediates (e.g. ZDI);
- Bug bounty programs (e.g. Microsoft, Google), Zerodium to know the quotation;
- Credit assured (e.g. Apple?)

Often involves development of Proof-of-Concept exploit that shows the vulnerability is exploitable.

Third party mediators

There are several mediators on the market, that act as proxy between security researcher and vendor. They communicate vulnerability to vendor, hold vulnerability information for a certain amount of time (typically 60-90 days) and when the hold period expires they disclose the vulnerability. It is a mechanism to push vendors to patch, so they won't be legally responsible for the exploits caused by it. Some of these mediators are Secunia, ZDI, SecurityFocus, Zerodium, ... If vulnerability is known before vendor releases patch = "zero day vulnerability" The Google Zero Day Project is a project to report vulnerabilities, discovering also those in competitors' software (e.g., meltdown and spectre). They aggressively release vulnerabilities informations after the deadline expires.



Vulnerability feeds

Vulnerabilities are disclosed by publication in the NVD and other vulnerability feeds (public and private). Private feeds release information earlier ("early advisories", like Secuina, SecurityFocus, ZDI). Public feeds typically release weekly or monthly updates (SANS@RISK).

The Open Web Application Security Project (OWASP), a worldwide non-profit charitable organization focused on improving the security of software, is a good resource for information security resources. It has compiled a top 10 of vulnerability threats, which is a good overview of most common vulnerability types with examples:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities
5. Broken Access Control
6. Security Misconfiguration
7. Cross Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

4.1 Vulnerability grading

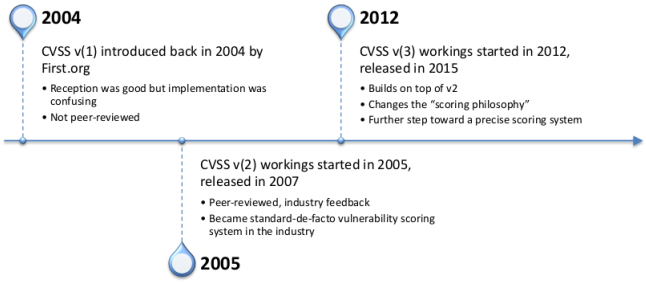
How severe are the security problems affecting a software configuration? Not all vulnerabilities are the same (XSS vs BoF vs SQLi vs Privilege escalation vs ...). Vulnerability counting can NOT be a measure of severity: even in a situation with plenty vulnerabilities it should be clear the general level of threat of the entire system. Moreover clients and users should be informed too: not all users are "security experts" (in many cases this score is read by people with just "IT knowledge").

A scoring system should be able to measure the security issue, but also communicate its criticality.

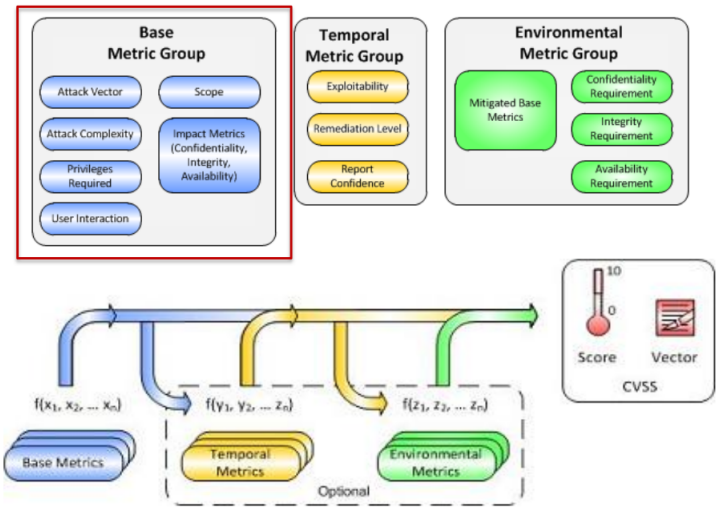
The Common Vulnerability Scoring System (CVSS)

CVSS is an open framework for communicating the characteristics and severity of software vulnerabilities. The goal is to have a shared system of metrics to analyze and measure vulnerabilities:

- Different users score the same vulnerability in the same way = **severity assessment**
- Different people "read" the same vulnerability and understand the same thing = **severity communication**



CVSS (v3) is based on three metric groups:



- **Exploitability metrics**: measured over the **vulnerable** component
  - Attack Vector
  - Attack Complexity
  - User Interaction
  - Privileges Required
- **Scope metric**: Security Authority of Vulnerable Component = Security Authority of Impacted Component?
- **Impact metrics** (Confidentiality, Integrity, Availability): measured over the **impacted** component (so it is affected by the threat).

Exploitability Metrics: Attack Vector

This metric reflects the context in which the vulnerability exploitation occurs. The more remote an attacker (or the attack) can be from the target, the greater the vulnerability score. Possible values:

1. Network: exploitation is bound to the network stack.
2. Adjacent Network: attacker needs to be in same subnet.
3. Local: attack is not bound to network stack, but rather to I/O on system. In some cases, the attacker may be logged in locally in order to exploit the vulnerability, otherwise, she may rely on User Interaction to execute a malicious file.
4. Physical: attacker must be physically operating over the vulnerable component.

Exploitability Metrics: Attack Complexity

This metric describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Possible values:

1. High: A successful attack depends on conditions outside the attacker's control. That is, a successful attack cannot be accomplished, but requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component before a successful attack can be expected.
2. Low: Specialized access conditions or extenuating circumstances do not exist. An attacker can expect repeatable exploit success against a vulnerable target.

For example, a successful attack (high complexity) may depend on an attacker overcoming any of the following conditions:

- The attacker must conduct target-specific reconnaissance. For example, on target configuration settings, sequence numbers, shared secrets, etc.
- The attacker must prepare the target environment to improve exploit reliability. For example, repeated exploitation to win a race condition, or overcoming advanced exploit mitigation techniques.
- The attacker injects herself into the logical network path between the target and the resource requested by the victim in order to read and/or modify network communications (e.g. man in the middle attack).

Exploitability Metrics: Privileges Required

This metric describes the level of privileges an attacker must possess before successfully exploiting the vulnerability. Possible values:

1. High: The attacker is authorized with (i.e. requires) privileges that provide significant (e.g. administrative) control over the vulnerable component that could affect component-wide settings and files.
2. Low: The attacker is authorized with (i.e. requires) privileges that provide basic user capabilities that could normally affect only settings and files owned by a user. Alternatively, an attacker with Low privileges may have the ability to cause an impact only to non-sensitive resources.
3. None: The attacker is unauthorized prior to attack, and therefore does not require any access to settings or files to carry out an attack.

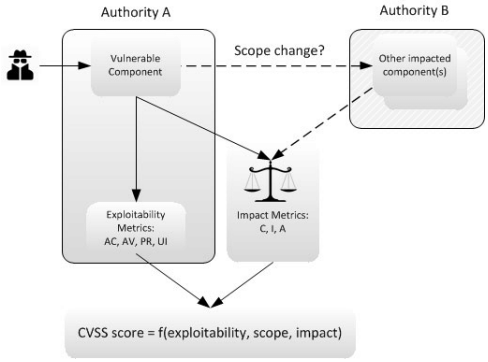
Exploitability Metrics: User Interaction

This metric captures the requirement for a user, other than the attacker, to participate in the successful compromise the vulnerable component. This metric determines whether the vulnerability can be exploited solely at the will of the attacker, or whether a separate user (or user-initiated process) must participate in some manner. Possible values:

1. Required: Successful exploitation of this vulnerability requires a user to take some action before the vulnerability can be exploited. For example, a successful exploit may only be possible during the installation of an application by a system administrator.
2. None: The vulnerable system can be exploited without any interaction from any user.

Scope

Scope refers to the collection of privileges defined by a security authority (e.g. an application, an operating system, or a sandbox environment) when granting access to computing resources (e.g. files, CPU, memory, etc). These privileges are assigned based on some method of identification and authorization. When the vulnerability of a software component governed by one security authority is able to affect resources governed by another security authority, a **Scope change** has occurred.



Possible values:

- Unchanged: An exploited vulnerability can only affect resources managed by the same authority. In this case the vulnerable component and the impacted component are the same.
- Changed: An exploited vulnerability can affect resources beyond the authorization privileges intended by the vulnerable component. In this case the vulnerable component and the impacted component are different.

Impact metrics

Measures the losses on

- Confidentiality = impact on confidentiality of data (property that information is not made available or disclosed to unauthorized individuals, entities, or processes);
- Integrity = impact on integrity of data (the "property of accuracy" of information)
- Availability = impact on availability of the component (is the "property of being accessible and usable upon demand by an unauthorized entity"?)

Each metric measures the losses suffered by the impacted component. Possible values:

1. High: total loss
2. Low: partial loss
3. None: no loss

Scoring Guide/Philosophy

When more than one assessment is possible, go with the more severe one (e.g. exploitation can happen through both local I/O and network stack, in that case go with network).

Scoring Exercise

MS Word Denial-of-Service attack (CVE-2013-6801): Microsoft Word 2003 SP2 and SP3 on Windows XP SP3 allows remote attackers to cause a denial of service (CPU consumption) via a malformed .doc file containing an embedded image, as demonstrated by word2003forkbomb.doc, related to a "fork bomb" issue.

Access Vector	**Local** (trigger of vulnerability by launching the file)
Access Complexity	**Low** (SPs probably installed by most users)
Privileges Required	**None**
User Interaction	**Required**
Scope	**Unchanged** (no other authentications required)
Confidentiality	**None**
Integrity	**None**
Availability	**High**

CISCO host crash (CVE-2011-0355): Cisco Nexus 1000V Virtual Ethernet Module (VEM) 4.0(4) SV1(1) through SV1(3b), as used in VMware ESX 4.0 and 4.1 and ESXi 4.0 and 4.1, does not properly handle dropped packets, which allows guest OS users to cause a denial of service (ESX or ESXi host OS crash) by sending an 802.1Q tagged packet over an access vEthernet port, aka Cisco Bug ID CSCtj17451.

Access Vector	**Adiacent** (virtual ethernet network)
Access Complexity	**Low** (virtual module)
Privileges Required	**None**
User Interaction	**None**
Scope	**Changed**(vulnerable component, the module, allows DoS by guest users)
Confidentiality	**None**
Integrity	**None**
Availability	**High**

CVE-2009-0927: Stack-based buffer overflow in Adobe Reader and Adobe Acrobat 9 before 9.1, 8 before 8.1.3 , and 7 before 7.1.1 allows remote attackers to execute arbitrary code via a crafted argument to the getIcon method of a Collab object, a different vulnerability than CVE-2009-0658.

Access Vector	
Access Complexity	
Privileges Required	
User Interaction	
Scope	
Confidentiality	
Integrity	
Availability	

Libvirt USB handling (CVE-2012-2693): libvirt, possibly before 0.9.12, does not properly assign USB devices to virtual machines when multiple devices have the same vendor and product ID, which might cause the wrong device to be associated with a guest and might allow local users to access unintended USB devices.

Access Vector	
Access Complexity	
Privileges Required	
User Interaction	
Scope	
Confidentiality	
Integrity	
Availability	

5 Malware

Malwares (or **malicious software**) are programs acting without the conscious or designed authorization of a user or system that may exploit system vulnerabilities. These programs need a host program to operate, they can't execute per se (e.g. viruses, logic bombs, and backdoors), can be independent self-contained programs (e.g. worms, bots) and can replicate or not. Malwares are a sophisticated threat to computer systems.

Taxonomy

- Virus = attaches, replicates and eventually hits;
- Worm = self-replicates;
- Trojan horse = allows taking control of a machine performing an action that looks legit;
- Keyloggers = sends typed info to attacker;
- Ransomware = attack data availability, asking for a ransom (a sum of money) to unlock the data. Has to be executed with root privileges;
- Rootkit = hook to libraries or system files;
- Bot-net = remote coordinated control of multiple machines.

Malware can assume characteristics of more than one type.

5.1 Keylogger

They are the most overlooked type of malware, because they are very effective and difficult to be detected. Modern OSs let users access directly the HW using the APIs, leaving open the door for a keylogger, who does not need to run in Kernel space (so with root privilege) but can attack remotely. A keylogger records the data inputted using a keyboard: in old OSs keys were sent directly to the foreground process, now they are published on a bus, where the process reads on, non-exclusively. This was introduced to enable the operation of shortcut managers, which need to be always listening to the user input.

```
int WINAPI WinMain(HINSTANCE inst, HINSTANCE hi, LPSTR cmd, int show) {
  HHOOK keyboardHook = SetWindowsHookEx(WH_KEYBOARD_LL, f, NULL, 0);
  MessageBox(NULL, L"Hook Activated!", L"Test", MB_OK);
  UnhookWindowsHookEx(keyboardHook);
  return 0;
}
```

The SetWindowsHook API has as parameters where to attach the hooking procedure (in this case the keyboard, could also be the mouse), the pointer to the hooking procedure function (f), the location of the function (NULL means that it is in the same program) and the thread associated to the keyboard (which can use it): in this last parameter, if the value is set to 0 it extends to all the threads, opening for the attack.

5.2 Social Engineering

Through Phishing or Spearphishing (more accurate/selective). An attack has to enforce a **problem recognition**, an **active involment** and a **constraint recognition**.

The ELM (Elaboration Likelihood Model) studies how humans change attitudes or make decisions they would not do without external stimuli.

There are 2 routes to persuasion:

- central route = force victim overthink
- peripheral route = force victim not to think

The principles of persuasion are:

- Reciprocity: expect to give something back in return to service
- Commitment: keep doing what you have been doing
- Social proof: do what the others do
- Similarity: trust people
- Scarcity: fear of losing something
- Authority: respect the law (rules)

5.3 Viruses

A virus is software that replicate and install itself without user consent. Copies can be installed into programs (modifying them to include a copy of the virus, so it executes secretly when host program is run), data files or boot sectors. The components of a virus include:

- infection mechanism - enables replication
- trigger - event that makes payload activate
- payload - what it does, malicious or benign

The virus can be prepended (beginning), postpended (end) or embedded into an infected program, which, when is invoked, executes the virus code. Virus payload may change size of executable, embedded layout may avoid this (system dependent): e.g. portable executables headers often have "empty" allocated memory words.

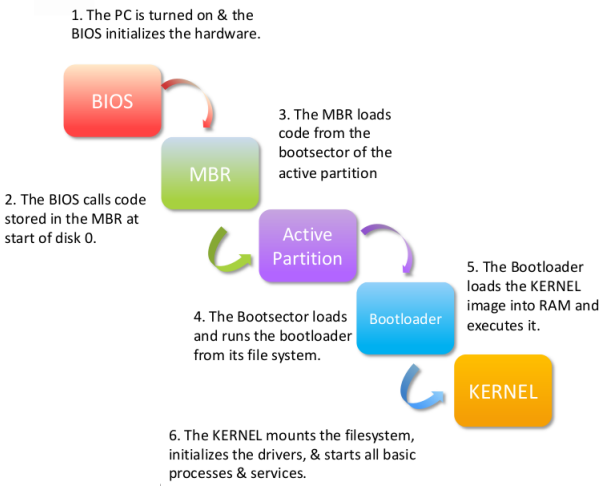
There are different types of viruses, categorized by infection target (boot sector, file infector, macro virus) or by concealment mechanism (encrypted virus, polymorphic virus, metamorphic virus). In the 80s the target of the virus were mainly executables, with the limitation that different virus needed to be developed for different OSs; then in the 90s became very common a new type of viruses that were platform independent, which instead of executables infected documents so they could easily spread. That was combined with the exploitability of macros in the office apps, where executable programs were embedded in the office documents (so whatever is the OS), not protected at that time; now they are recognized by many anti-virus programs so they evolved to email viruses, which exploit an auto-execution bug in email clients to infect the system.

5.3.1 I Love You (2000)

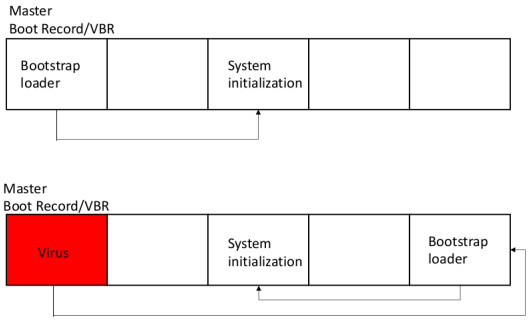
The user believes that the attachment is a txt file, but it's actually a VBS (Visual Basic Script): opening the attachment loads and executes the script. The impact caused by it is the disrupt of system files, and it also replicates by sending itself to the full contact list. Not relying on office but it still relies on an "interpreter" to execute, it is not native code.

Boot sector infections

What happens during the boot (old scheme, now it starts with UEFI)?



At boot time, the firmware checks for system components and tests them. The operating system is then copied from the hard drive to the RAM. The Master Boot Record (MBR) contains code that ultimately leads to loading OS in memory. The MBR, typically small in size, points to boot loader (in Volume Boot Record, VBR), forming a "chain loading" system. The boot loader actually loads the OS. The goal of the virus is to include itself in this chain, before the kernel so it can hide effectively.



In the second image is depicted an example in a windows version where it was possible to rewrite the MBR from the user space: the virus, placed in the MBR, was able to insert himself in the boot sequence, and the bootstrap loader was moved in a different location in the memory, executing the system initialization as usual: in this way it was undetectable.

Now UEFI and secure boot avoid these exploitations, with the Trusted Platform Module (TPM), that checks the integrity of all steps of the boot.

Rootkits

Rootkits can take control of the MBR, injecting into the kernel to defeat disk encryption (Stone Bootkit). They are a set of programs installed for admin access, subverting report mechanisms on processes, files, registry entries, etc. They may be:

- persistent or memory-based;
- kernel mode (hard to detect and remove);
- installed by user via trojan or intruder on system.

Virus countermeasures

Prevention is an ideal solution but difficult. It realistically needs: detection, identification and removal. If detect but impossible to identify or remove, the whole infected program must be discarded and replaced.

AV defenses

Virus and antivirus tech have both evolved: early viruses were just simple code, which were easily removed, and as they have become more complex, so have the countermeasures. There are 4 generations of AV technologies:

1. **signature scanners**: look for known traces of virus in memory;
2. **heuristics**: look for features common in malware traces/strands;
3. **identify actions**: behavioral fingerprint of the malware execution;
4. **machine learning**: classifiers trained to decide whether a file/program is acting maliciously.

Defense 1 - Signature scanners

Malware is analysed by a security firm, searching for a footprint of malware in memory. Every time malware is loaded into memory, a pre-fixed series of bits will appear in disk: this footprint is the "signature" of the malware. Recognition happens through matching those sequence of bytes with all signatures known to a security product. It is a purely "reactive" strategy, unknown malware does not yet have a signature. Detection can only happen after analysis.

Defense 1 - Heuristics

Partially addresses the polymorphism problem. Viruses may evolve to different strains of the same virus family: manual modifications, new malware versions or genetic algorithms. Different footprint but common characteristics so rather than having an exact match of the footprint in memory, detection happens by partial matching, searching for common characteristics of a virus strain.

Evolution 1 - Polymorphic viruses

Polymorphic is the first technique that posed a serious threat to Antiviruses. It uses encryption to obfuscate code, and the decryption module is modified at each infection so that all samples will have a different footprint in memory (fixed encryption per se would not be enough). A well-written polymorphic virus has no parts which remain identical between infections, so signature checking is useless and heuristics may work only if encryption-decryption pair does not vary enough.

Defense 2 - Generic Decryption

Each polymorphic virus will look different on disk, but at execution time code will always be the same. If detection happens when malware is executed, it's too late. Generic Decryption is basically Sandboxing: a potential virus is executed on an emulated environment, which has no actual access to system resources. The malware decrypts itself so signature checking will now work. Modern malware can prevent execution in emulated or virtual environment via analysis of the execution environment, and prevent analysis by researchers.

Evolution 2 - Metamorphic viruses

Metamorphic means that to avoid being detected by emulation, some viruses can rewrite themselves completely each time they are to infect new executables (differently from polymorphic also the logic changes). After execution on emulated environment, the signature won't match. A metamorphic engine is needed to enable virus, which has to be very large and complex (e.g., Win32/Simile virus consisted of over 14,000 lines of assembly code).

Defense 3 – Behavioural detection

Addresses issue with metamorphic malware and detection of previously unseen malware. It is based on a set of actions that the malware performs. The basic idea is that a malware behaves differently from legitimate software: system calls, interaction with drivers (e.g. I/O), system interrupts, etc. It is very hard to enumerate all possible actions (exponential time) and so be precise, so the analysis is restricted to closed patterns. It is also hard to correctly identify set of actions that characterise malware: the risk of false positives is higher than for heuristics and signatures (you need an hash collision for that).

Defense 4 – Using on ML

It is the natural extension of the previous approach. Machine learning is used to model the malware detection problem as a classification problem. Bayesian networks were the first of being employed with satisfactory results, neural networks are more popular nowadays. It is still hard to correctly identify set of features that characterise malware, so the risk of false positives remains. Another problem is the employment of Adversarial AI from the attacker point of view, which can study a malware that does not get classified as such.

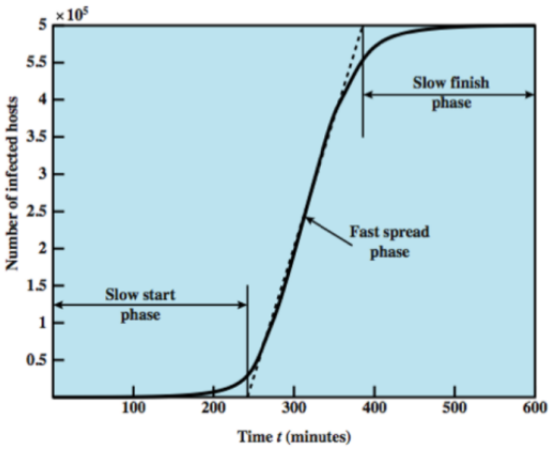
Defenses in practice

Defense is only effective when it prevents malware execution, once the system is infected, system can not be trusted anymore. Malware removal can not be trusted because malware can affect the integrity of system procedures too, intercept antivirus' calls to OS disk drivers to analyse stored malware (returns "null" or benign file) or disable antivirus itself (e.g. Conficker). Run analysis from a clean drive on uninitialized infected OS.

5.4 Worms

Worms are replicating program that propagate over net using email, remote exec, remote login. They exploit the remote payloads, typically using arbitrary code execution that causes buffer overflows. A worm has phases like a virus: dormant, propagation, triggering, execution. During the propagation phase, it searches for other systems, connects to one, copies self to it and runs; then it repeats this. It may disguise itself as a system process. It was implemented by Xerox Parc in Palo Alto in the 1980s.

The propagation model is a logistic model in finite systems. It starts off exponentially, then levels out (mainly because it becomes more difficult to find other machines not already infected).



5.4.1 Morris worm

Created in 1988 by Robert Morris, convicted under Computer Fraud and Abuse Act (3 years on probation, now CS professor at MIT). It exploited different vulnerabilities:

- Sendmail = could execute command via SMTP;
- Finger = BoF;
- weak passwords = dictionary attack.

There was no malicious payload, but propagation was too fast for the infrastructure to hold: a single computer could be infected multiple times, similarly to a "fork bomb" issue (malware needs testing too). It caused several million dollars in damage.

Historical internet worms

- Morris worm (1988): overflow in fingerd, 6,000 machines infected (10% of existing Internet);
- CodeRed (2001): overflow in MS-IIS server, 300,000 machines infected in 14 hours;
- SQL Slammer (2003): overflow in MS-SQL server, 75,000 machines infected in 10 minutes;
- Sasser (2004): overflow in Windows LSASS, around 500,000 machines infected.

5.4.2 The Welchia and Blaster worms

- Blaster: appeared in august 2003, affecting primarily Windows XP machines. It caused SYN D-DoS against *windowsupdate.com*. The vulnerability exploited a BoF in RPC (patch existed since May 2003). As a side effect it made RPC unstable and XP unusable.
- Welchia (anti-worm): removed the Blaster infection, patching the vulnerability. It used the same Microsoft RPC bug as Blaster and deleted itself after January 1, 2004. It turned out to be a bad approach to patch the victims because failures of it stopped some services (unwanted patching without any validation).

5.4.3 Slammer (Sapphire)

BoF in Microsoft's SQL server, patch released 6 months earlier. A single UDP packet sent to port 1434 infects the machine (binary fits in the packet). The worm overwrote RET to point to malware in buffer, then generated random generation of IP addresses to send copy of itself. It worked because IP space is populated, mostly by MS systems: the attacker did not care about false postives. 30k copies/second were sent via UDP, with an exponential growth, so fast it saturated the bandwidth of the whole internet in 10 minutes, in combination with routers failing and subsequent generation of route table updates traffic. 75k SQL servers were infected.

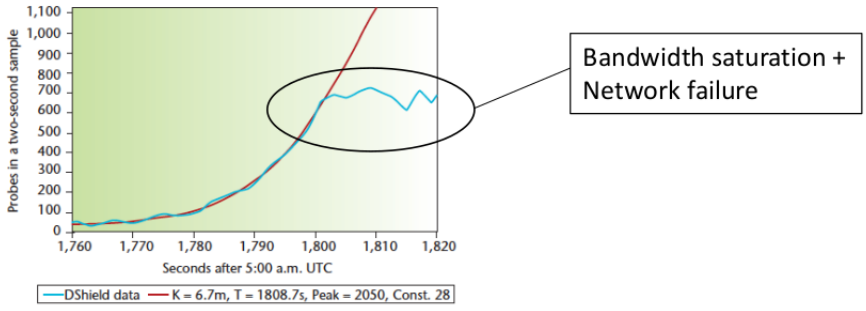


Figure 3. The early moments of the Distributed Intrusion Detection System (Dshield) data set, matched against the behavior of a random-scanning worm.

It killed several critical points of internet infrastructure (5 DNS root servers, all of South Korea's cell phone network, Bank of America ATMs): those were the one really afflicted by it since there was no malicious payload executed on infected systems. The infection followed a logistic model in finite systems, starting off exponentially, then levelling out also becuase of the same network failure it caused (no bandwidth available).

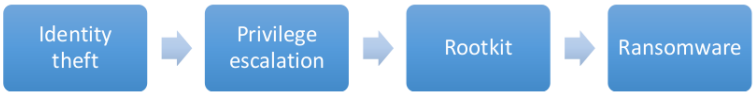
More recent malware

At first the use of malware was not a profitable activity: the attacks developed were mostly a type of "vandalism" also because the investment to create them was not important. Things have changed completely in the last years. Cyber crime has become an activity regulated by the rules of economy: there are investments in the production of malware aimed to gain money. This is reflected in the types of malware: the techniques seen in the previous malwares are still used but have been developed and are now much more sophisticated. Recent malware are **target attacks**, against generalized attacks typical of the previous decades. The targets are progressively more and more small businesses (from 18% in 2011 to 43% in 2015). The type of attacks seen later on is called Advanced Persistent Threat (ATP):

- Advanced: these attacks are more advanced because they mostly exploit not already known vulnerabilities (patching is now more efficient) but Zero-day vulnerabilities (not known yet);
- Persisent: while the previous malwares left the machine after doing the damage, these tend to remain hidden in the machine **logging information** for months or even years, to get as much as possible.

Most of APTs are also multi-threat and multi-stage: they are a bundle of exploits that include all the previously seen malwares and attack over different stages, like a chain.

These phases do not have to be performed all in this order, for example the first 3 of them may be repeated after the attacker gets into the system, so that it is possible to perform reconnaissance from inside the system and find other vulnerabilities to be exploited.
In the end it may be possible to execute a chained exploit:



Each vulnerability used is also rated without taking into consideration the following ones, so a vulnerability may end up underrated.

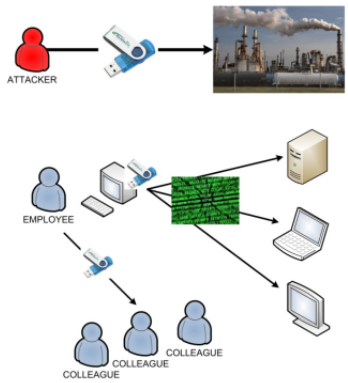
5.4.4 Stuxnet

Probably the prototype of sophisticated malware. It was a complex attack targeting not information systems but a cyberphisical system, a nuclear plant. An attack on a phisical system is a threat

not only to security but also to safety, as it could be one on the recent IoT systems. It is divided in 3 phases: phase 1 attacks the pure network information system (IT), phase 2 and 3 target the industrial control (OT).

- **PHASE 1**: an almost normal worm, amazingly smart, spreads, hides, updates itself (C&C, command and control, communication) and looks around to duplicate itself (only in an environment with equipment that makes possible to enter phase 2).
- **PHASE 2**: attacking the Siemens + PLC (Programming Logic Controller, which monitor the state of input devices) systems. It infects the SIEMENS System and it modifies the PLC programming.
- **PHASE 3**: sabotage. It checks for a specific factory environment (Iran): if it does not find it, it does nothing, but if it finds it, it changes the speed of the centrifuges.

Phase 1: the Window system



A very elaborate standard worm was spread inside the LAN using a compromised USB stick, which infected other USB sticks and exploited Print Spooler, Shared Folders, etc. 4 zero-days vulnerabilities were used:

- **LNK** (MS10-046): windows shell vulnerability to execute remotely code;
- **Print Spooler** (MS10-061): execute remotely code;
- **Server Service** (MS08-067): execute remotely code;
- **Privilege escalation**: obtain privileges.

It looked for Siemens Step 7 project folders, installed a rootkit that made it invisible and checked which antivirus was active then chose the target executable accordingly.

Phase 2: targeted attack

Looked for systems running WinCC and PCS 7 SCADA MANAGEMENT PROGRAMS, finding them even if disconnected (via USB sticks). It attacked them, exploiting:

- Hard-wired password (WinCC);
- Uploads as stored procedure;
- Siemens Project 7 folder vulnerabilities.

It hooked into specific Step 7 dll to communicate with the PLC, replacing the PLC Code and hiding using Rootkit techniques. It was the first PLC rootkit EVER.

Phase 3: the damage

Hit frequency converters controlling speed of motors, making them go faster than planned to burn the centrigue. It was designed to hit a specific plant.

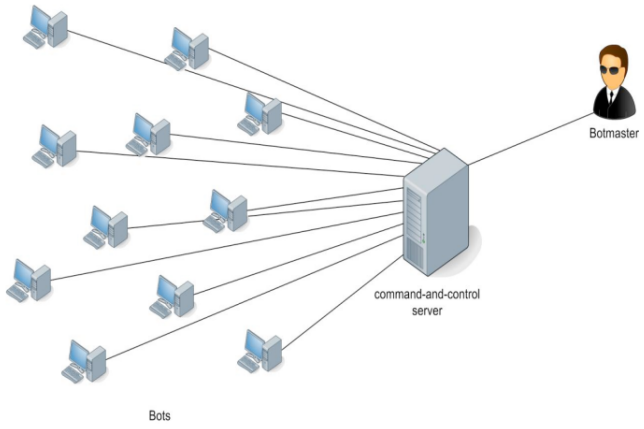
5.5 Botnets

Virtual Network of infected machines under the control of a "bot herder". These machines can perform any kind of action for the bot herder and are managed through a C&C server under the control of an attacker, which pushes configuration files and functionality updates. Bots must be able to communicate with the C&C (Command & Control) server in order to do something. There are 2 types of design of botnets: centralised and peer-to-peer.

Main usage of botnets:

- Performing distributed denial of service attacks (DDoS): same techniques as normal DoS attacks, but amplified by a factor equal to size of botnet;
- Spam: used to distribute spam emails, which can lead to further infections or subscription to services/goods;
- Computational power: use CPU/GPU time to find hash collisions, break ciphers, mine bitcoins;
- Steal sensitive information from the infected machine;
- Rental: bot herder can rent part of the bots to other criminals, to outsource computations, buy credit card numbers (CCNs), etc.

Botnets – centralised architecture

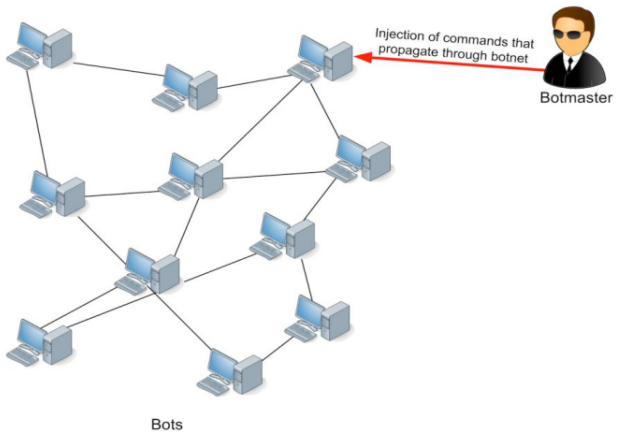


Bots communicate with the bot herder via

- IRC (Internet Relay Chat) server: first definition of "bot", served "human users" by providing automatised services. Essentially a program accepting commands in inputs and retrieving answers;
- HTTP (connects to a remote HTTP server): a bot may contact the herder fixed (set of) IP(s) or resolve the domain dynamically (using **fast-flux** or **domain-flux**).

C&C server is a single-point-of-failure: who controls the C&C controls the botnet.

Botnet – p2p architecture



More robust than centralised architecture. Commands are spread through the network and bots can act as both slaves and masters dynamically. When a new machine is infected, the bot joins the network with different approaches:

- Hard-coded list of peers are contacted upon infection, updates its neighboring peer list;
- Mixed p2p/centralised approach, centralised web cache with list of peers;
- Infected bot inherits peer list from infector.

Three types of p2p botnets [Silva 2012]

- **Parasite**: all bots are selected from vulnerable hosts within an existing P2P network. Number of vulnerable hosts in the existing P2P network limits the scale of a parasite botnet. It is not flexible and greatly reduces the number of potential bots under the botmaster's control.
- **Leeching**: members join an existing P2P network and depend on this P2P network for C&C communication. Bot candidates may be vulnerable hosts that were either inside or outside an existing P2P network.
- Bot-only: builds its own network in which all members are bots.

Centralised botnets - protecting C&C

Bots can not operate if they can not contact the C&C server. Centralised Botnet take downs happen by "sinkholing". Security researcher/firm takes control of C&C, and the C&C server needs to be protected:

- Change IP address frequently = **fast-flux**. Makes it hard for an attacker to take it down. One domain is mapped to several IP addresses.
- Change domain frequently = **domain-flux**. Each bot generates "valid domain names" periodically and resolves them.

Domain flux

Each bot uses a **Domain Generation Algorithm** (**DGA**) to generate a list of possible domains at a certain time ("rendezvous" domains). The list is generated independently by each bot. If a bot gets no answer from a generated domain, it simply switches over to the next in the list. Conficker A (a very famous botnet), for example was using *txkjngucnth.org* (`https://msrc-blog.microsoft.com/2009/02/12/conficker-domain-information/`). Sometimes botnets perform accidental DoS attacks against "colliding" domain names: DGA generates a domain that already exists and all bots try to contact that domain (it happened, e.g., jogli.com, praat.org).

5.5.1 Torpig [Stone-Gross 2009] (case study putting all together)

Torpig was a botnet active in 2009. It used Mebroot as a rootkit, which substitutes the Master Boot Record of the machine (used to perform actions at boot time). Mebroot is a hard to detect malware, executed in the context of *explorer.exe*, which operates directly on disk blocks (through disk drivers) and, upon reboot, downloads and activates malware. Torpig in this case encrypted the communcation with Mebroot server (the malware was stored locally, encrypted, so it was difficult to identify its behaviour). Mebroot provides functionalities to embed (malicious) modules to normal system boot.

The functionalities of Torpig were credential stealing, generation of phishing attacks for a set of pre-defined websites and injecting phishing content to webpages presented to user (typically a login page).

Sinkholing Torpig

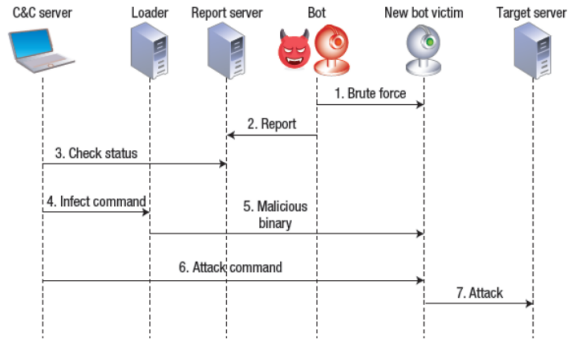
A team at University of California reverse engineered the DGA: they noticed that a set of domains that would be generated between 25th Jan and 15th Feb were not registered yet, so they registered the domains and replicated a "fake" C&C server. All it needed to do was to confirm itself as a valid server: Torpig uses HTTPS but accepts any certificate as valid. They started passively listening to whatever the bots were sending. On the 4th Feb the bot herder realised what was going on and a Mebroot update was pushed for Torpig, after only 10 days.

IPs changed very frequently (fast-flux) so the counting of unique IPs is not a good proxy for botnet size. Each bot has unique id + additional features. About 180.000 hosts (1.2M IP addresses).

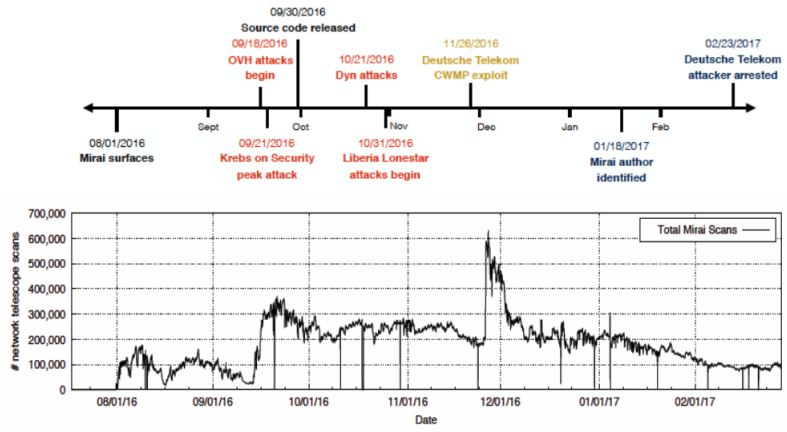
Torpig collected mailbox accounts, emails, form data, HTTP account, FTP account, POP account, SMTP account and Windows passwords.

5.5.2 Mirai botnet

They are botnet targeting IoT devices (cameras, DVR, routers, etc.). There is an high number of variants, some still active. It was first discovered Aug 2016. The source code has been published, resulting in a lot of people implementing it. Mirai was used to mount huge DDoS against several websites.



With Mirai it is possible to compromise IoT, like cameras, by trying bruteforce on Telnet, service still allowed by many devices that enables to login and execute remote commands; Telnet does not implement an encrypted channel and it is protected by a password, which is usually really weak. The malware tries a list of passwords and it has been very succesful, obtaining control of devices which report their IPs to a server. The C&C server can use these devices to mount a DDoS attack. An example of DDoS: after revealing the identity of two guys behind a DDoS service, the site Krebs on Security was subject to the most massive DDoS attack ever witnessed to that moment. Clocked at a staggering 620 gigabit per second, it was carried out by botnets consisting mainly of IoT devices, mostly security cameras and DVRs used in home or office settings.



After the release of the source code, variants appeared using other protocols (like HTTP).

6 Web based attacks

The increase of economic value in the use of internet (like in online banking) shifted the focus of cyber-crime towards it, too.

National States role

Each National State plays now a role in security since cyberspace has become a new defense dimension (Land, Sea, Air and *Information*).

> "a global domain within the information environment consisting of the interdependent network of information technology infrastructures and resident data, including the Internet, telecommunications networks, computer systems, and embedded processors and controllers"
>
> US DoD (Department of Defense) definition

6.1 Malware propagation

Internet Worms (=self-propagating malware) spread at very high speed and their payload could deliver any type of functionality to the attacker: faster propagation speed means a higher number of infected targets, higher number of infections means more bank accounts (not necessarly) and more bank accounts means higher ROI for the attacker.
Malware author operates in a competitive and adversarial environment:

- **Adversaries** are security researchers that reverse engineer their malware and security firms that build AV signatures for malware detection;
- **Competitors** are the many players in the malware development market. The market of infections has finite amount of resources and finite number of vulnerable systems (each system worth x $). Malware authors compete to access victim's valuable information.

Propagation vs operation

Strategy 1: High propagation rate

- PRO: several infections per unit of time
- CONS: the more samples of malware in the wild, the higher the chances to hand a sample to security researchers (more infections means faster detection)

Strategy 2: Low propagation rate

- PRO: higher stealthiness and fewer chances of infecting a system already infected by another malware
- CONS: fewer infections per unit of time

These conditions hold for all attackers. In the economic theory there is an "equilibrium point" whereby all competing players maximize their expectations in terms of return to investment.

Infection strategy example (intuition)

There are $K > 1$ attackers that compete to infect $N >> 1$ systems, collectively worth M. The average is M/N. Assume that all N systems have an antivirus: the survival time of malware K (L_k) is inversely proportional to the number N_K of systems infected by K (say $L_k = 1/N_K$).

- Strategy 1 = all attackers infect all systems. The average return for each attacker is M/K; L_k is $1/N_k = 1/N$ = lowest possible.
- Strategy 2 = all attackers infect N/K systems. The return for each attacker is $N/K * M/N = M/K$ which is the same as before; L_k is $1/N_k = 1/(N/K) > 1/N$: the mean lifetime of the K^{th} malware with S2 is higher than with S1. It is true for all values of K.

Self-replication vs controlled deployment

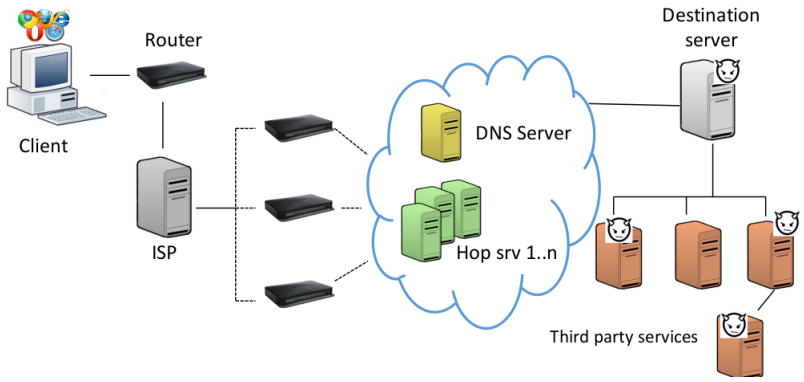
It is very hard to predict outcomes of fully-automated propagation mechanism (e.g. Morris worm was programmed to "contain" its propagation, replicating 1 time out of 7, but spread uncontrollably). That is why modern (post 2010) internet malware does not employ self-propagation mechanisms. Rather, malware distribution operates over standard request-reply network mechanisms (like botnets), since they have long lifetimes. They are usually distributed over the so-called Malware distribution networks, that control infections using different methods (it is like a market):

- Automated malware installs via software exploits (typically through the browser/third party plugins);
- Malware services that install malware (Mebroot);
- Pay-per-infection (do not care about methods).

There is an emergence of these markets for infection.

Malware Distribution networks

They enforce web attacks via several mechanisms. Servers on the web that "deliver" the malware to the final user may exploit compromised websites, content networks (e.g. advertisement) or redirects.



6.2 Malware delivery – mechanisms review

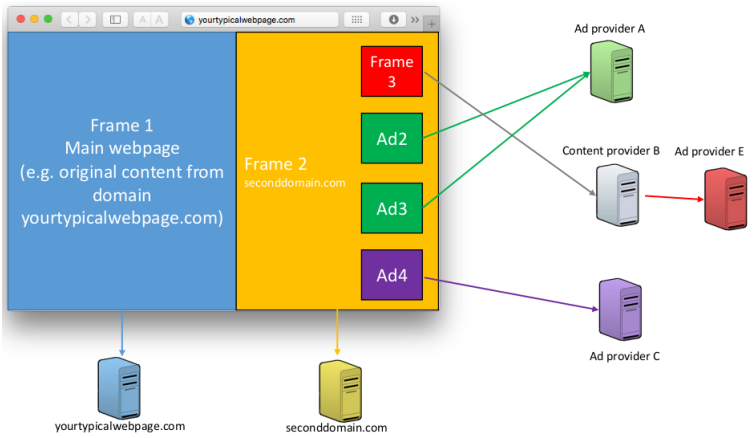
Malware infections happen through one or a combination of different channels:

- **Service infection**: buffer overflow of a vulnerable service listening on the network (RPC, Web servers, SQL servers, ...). Nowadays services are more difficult to reach: NAT and firewalls are implemented and incoming connections are controlled so that only services supposed to be listening on the network are reachable (e.g. SSH from internal network only, HTTP from everybody, so a SSH vulnerability can not be reached from outside).
- **Client infection**: buffer overflow against user's client (e.g. Browser, plugins), redirects of user's browser to compromised websites, social engineering (convince user in performing an action with mail or phishing websites) and others like password guessing, infected devices.

Client infections

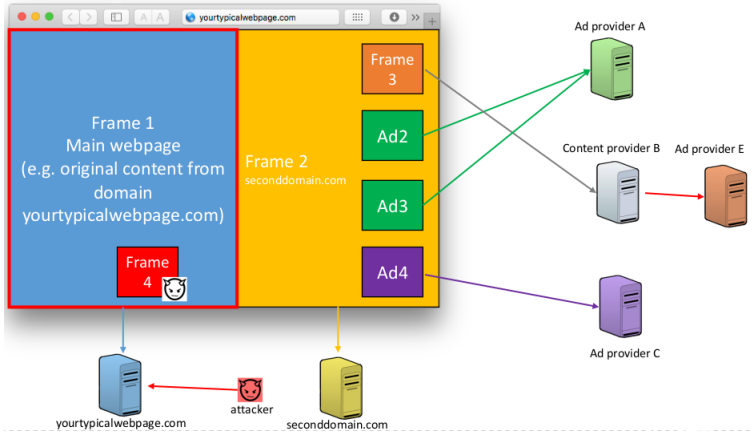
Browser-related content requests are by far the most common on the web. Client infections are typically driven by browser or other client activity (mail clients, chat clients, etc). There is a limited set of configurations, so a less uncertainty on vulnerability distribution: 4 browsers share the biggest fraction of users, with similar plugin configurations: Flash, Javascript, Adobe, Silverlight.

Webpage contents and operations



Same origin policy is enforced by browser so that content of FRAME 2(or 1) can not access content of FRAME 1(or 2), like stored cookies, loaded content and scripts. The browser will trust content from both frames and execute it in separate execution contexts; it requests and display content in one and executes scripts in another. An implicit trust-chain is established: browser trusts *yourtypicalwebsite.com*, *seconddomain.com*, Ad provider A,C and content provider B. Moreover content provider B trusts Ad provider E and browser implicitly trusts Ad provider E. However, trust is not-transitive: even if content provider B is trustworthy, entities trusted by B are not necessarily trustworthy too.

Sources of risk – domain compromisation



An attacker exploits a vulnerability on the domain's server (in the example, *yourtypicalwebpage.com*, could also be *seconddomain.com*), with BoF on the HTTP service or password attacks (e.g. against domain's administrative panel). The attacker inserts arbitrary content on webpage and the content is loaded by every user that requests the compromised webpage.

Sources of risk – content compromisation

A variant of the previous one is the compromisation of the content, like in a **XSS** attack.

Regardless of execution, XSS is based on the implicit notion of trust that exists between a browser and a server: the browser executes whatever the contacted website says, since the "Same-origin-policy" is respected (applied also to browser cookies, JS execution, etc). This vulnerability allows the attacker to inject content on a webpage and when the victim browser loads the webpage, it executes the injected content. The browser can not distinguish between legitimate and "malicious" instructions, they are all coming from a trusted source. In the **XSS Stored** (or **Persistent XSS**) variant, the content is stored on the remote server (e.g. a forum thread, a newsletter, a database)

and whenever a user retrieves a certain webpage, the malicious content is delivered to their browser.

Attacker exploits a vulnerability in some content manager present on the server (e.g. web forum, wiki engines, comment forms), with a similar vector to persistent XSS attacks, and injects unsanitised content onto webpage: typically javascript content that performs some actions (JS is Turing complete) like the redirection of the webpage towards malicious domain. Javascript is typically embedded in a `<script></script>` element:

- executed by the browser when the page is loaded

```
<script> alert("Javascript msg")</script>
```

- triggered by events

```
<a href src="seconddomain.com" onmouseover="alert("Javascript msg")">
```

- triggered by user actions

```
<a href src="Javascript: alert("Javascript msg");">Second domain.com</a>
```

Javascript can access elements of the DOM (Document Objected Model), like forms and links

```
document.cookie;
```

or of the BOM (Browser Objected Model), like window and location.

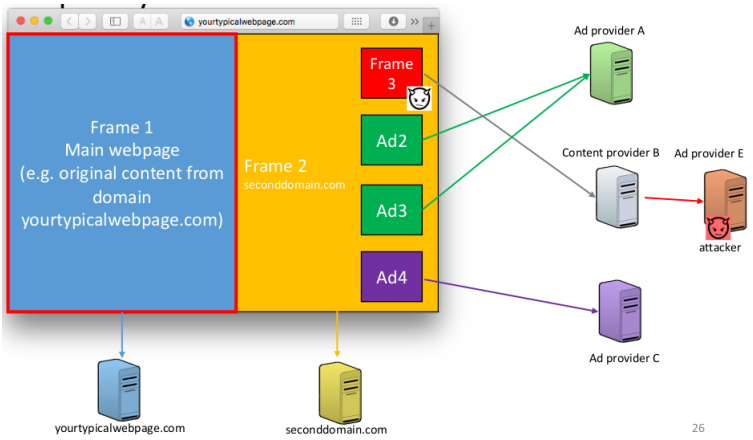
```
location.replace("thirddomain.com");
```

An example of content compromisation is the one found by Provos in 2006, on a website to create and publish customised online polls. It is an obfuscated javascript code.

```
<SCRIPT language=JavaScript>
function otqzyu(nemz)juyu="lo";sdfwe78="catio";
kjj="n.r";vj20=2;uyty="eplac";iuiuh8889="e";vbb25="('";
awq27="";sftfttft=4;fghdh="'ht";ji87gkol="tp:/";
polkiuu="/vi";jbhj89="deo";jhbhi87="zf";hgdxgf="re";
jkhuift="e.c";jygyhg="om'";dh4=eval(fghdh+ji87gkol+
polkiuu+jbhj89+jhbhi87+hgdxgf+jkhuift+jygyhg);je15="')"; if
(vj20+sftfttft==6) eval(juyu+sdfwe78+kjj+ uyty+
iuiuh8889+vbb25+awq27+dh4+je15);
otqzyu();//
</SCRIPT>

location.replace('http://videozfree.com')
```

Sources of risk – malicious third party content



Ad networks are a typical infection drive ("Malvertising"). Advert can deliver malicious javascript, social engineering attacks, exploit plugin vulnerabilities, etc. As an additional problem, it is hard to track evolution of third-party providers (advertisement, widgets, ...): they can be trustworthy at start of contract, may change behaviour later on, it is hard to know.

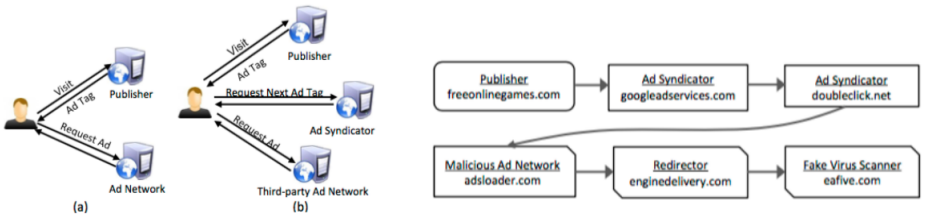
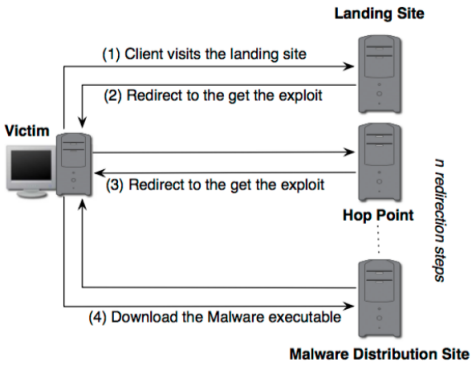


Figure 1: (a) Direct delivery (b) Ad syndication.

The delivery of Ads can be direct, but it is usually mediated by a Syndicator, an aggregator which chooses the right Ad to be displayed for every situation. As it can be seen in the real case scenario (second image), syndicators are also usually chained.

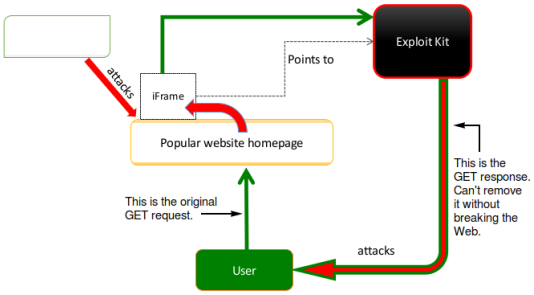
Drive-by downloads

It is a common infection mechanism employed by attackers: when contacted, the remote server delivers content that tries to exploit local vulnerabilities on the machine, typically buffer overflows against common browsers or browser plugins. If successful, shellcode calls home, downloads malware and executes it. Different redirections are made before reaching the Malware Distribution site to hide it and confuse forensics.



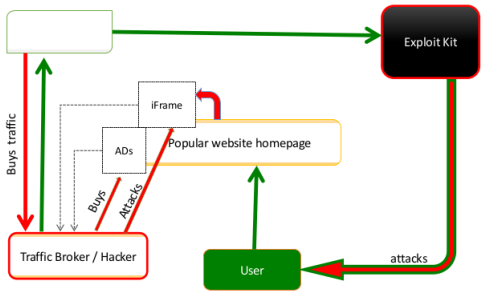
6.3 Exploit kits

Exploit kits are websites that serve vulnerability exploits and ultimately lead to malware. They can be reached through any of the mechanisms discussed so far: domain/content compromisation or third-party content. They typically feature < 10 exploits, but the trend is decreasing in time and now many exploit kits feature 3-4 exploits (why so few? Because they are valuable, there is no need to reveal more than the one necessary). Only the exploit for the detected configuration is employed. Kits are traded in the black markets.



Exploit kits only work if they receive victim traffic (direct links, ads, iframes, redirections, ...). Underground has services that trade connections ("Maladvertising", spam, iframes on legit websites). An attacker "buys" connections from specific users, with specific configurations: Javascript checks local configuration, sends to remote server which redirects to the exploit kit and when the user loads the webpage compromised, and if the characteristics match, traffic is redirected.

Traffic redirection



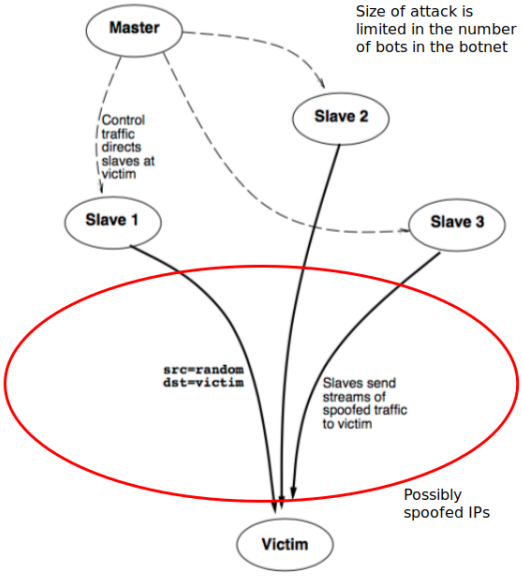
There are 2 types of traffic:

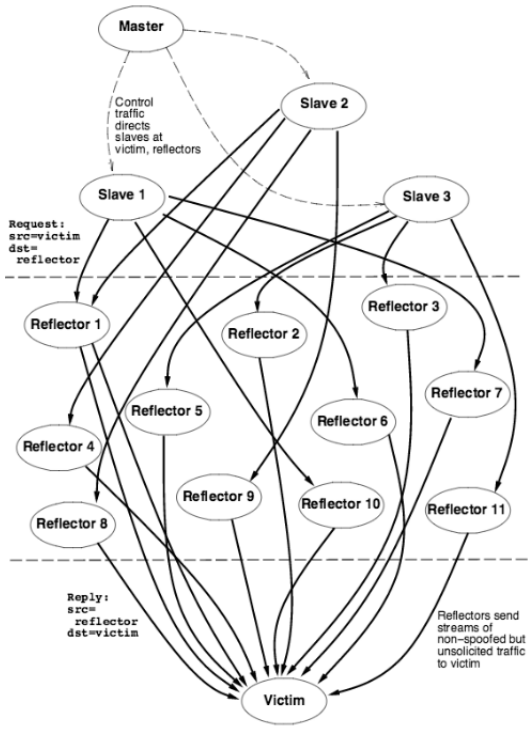
- Blind skymmed: when a visitor makes a click to open a video/photo/cam show but your landing page is opened instead. As a rule your page is opened in a new window and your page is the only page opened after a click.
- Popunder traffic: when a visitor makes a click anywhere on a page and your page is opened in a new window underneath current browser window. When a visitor closes the current window he will see your page opened underneath.

You can buy traffic from "traffic brokers". The user does not have to click on anything, it is an automatic redirect. High-quality traffic derives from selection of connection based on requested criteria like geographic source or installed software.

6.4 Advanced Denial of Service attacks

Basically every DDoS attack makes use of botnets. With standard DDoS attacks the attacker sends out orders to a big number of slaves which will then directly attack victim (traffic disguised as high quality). However most websites are able to support a good amount of connections.

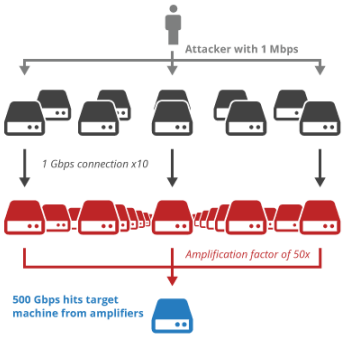




A more efficient way to implement a DDoS attack is the **Reflected DDoS** (Paxson 2001).
A Reflected DDoS uses "reflector" servers that receive, from the slaves of the Bot Herder, a connection request with as IP source the (spoofed) IP of the victim: these servers all reply to the victim.

Requests can be on any protocol (TCP, UDP, etc), as long as the victim is in LISTENING state.

There is also the **Amplification attack**, which aims to amplify the size of the response message sent to the victim, like in the previously seen DNS amplification attacks: a small spoofed request can generate a big reply (the spoofed machine is the victim of the attack). DNS configurations typically use UDP (preferred over TCP because the SYN ACK would be dropped by the server) only up to 512 bytes answers, generated by 64 bytes requests; if the size of the answer is greater than 512 bytes, it is switched to TCP (harder to spoof IP, foils attack). This means that the max amplification factor is $512/64 = 8x$.
Other protocols may allow for bigger ratios.



Network Time Protocol

It is a protocol that allows to have a time server to syncronize all the machines in the network, running on UDP, port 123. The NTP command *monlist*, intended for diagnostic purposes, returns addresses of the last (at most) 600 clients contacted by the NTP server. The size of an amplification attack using NTP is a lot bigger than the previously seen one. The NTP traffic rose in 3 orders of magnitude between January and March 2014, with several attacks in that period. The median amplification was $x4$ (25% of amplifiers up to $x15$), the max amplification up to $x1.000.000$ (likely misconfigured NTP servers, aka "mega-amplifiers"). The issue has now been largely resolved.

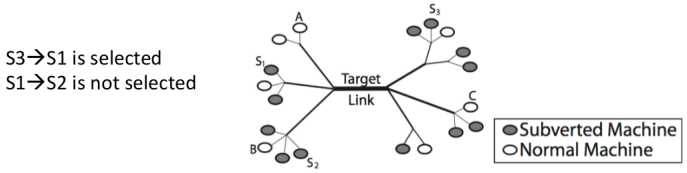
Mitigations

It is really difficult to mitigate this attack.

- **Source identification** (a bit useless): try to cut out from network hosts that generate DoS packets. IP spoofing however is a problem. It may be possible to trace back routing path, but it gets difficult with many sources (reflectors).
- **Capabilities**: rather than immediately granting resources to the initiator of a TCP communication, initiator has to ask, and the receiver grants a right to connect, or "capability". This capability is made of marks (unique hash values), set by routers on the path from sender to receiver, which have an expiration time. Routers check validity of marks upon response and, if valid, forward datagram: the receiver can deny capability if sender misbehaves. Otherwise, routers drop if capability is invalid (e.g. check will fail for answers to a spoofed IP). The limitations is that a Denial of Capability attack can still be performed: 5% of downstream bandwidth is dedicated to capability requests (e.g. 0.05 x 100Mbps), and can easily be saturated by a DDoS attack. New legitimate users that need a capability are cut out while there is no problem for clients that already obtained a capability before the start of DoS. It is hard to discern legitimate capability request traffic from non-legitimate: a sufficient low rate from each bot can flood the bandwidth.

Coremelt attack (theoretical)

It is a DDoS attack that overcomes obstacle posed by capabilities. Rather than attacking a victim system, it attacks a network link causing bandwidth saturation. The idea is that in a N bots botnet, there are N^2 possible connections. An attacker orders pairs of bots to send each other packets, which are wanted by both ends (valid capability). Bot pairs are defined so that communication passes through the target link (can be done with a traceroute). Effectiveness depends on bandwidth distribution between Systems and bot distribution in the network ASs.



7 System hardening

All systems (personal computers, servers, mainframes) have a default configuration. Usually it is the one given with a fresh installation of an operating system, which in some cases can be configured at installation time but it is still a limited access to full configuration settings (e.g. a linux distro typically allow to select packets but not all packet's functionalities). Default services are DHCP, RCP, NetBIOS, protocols like SSH or VNC, web servers, remote interfaces; default configuration satisfies the vast majority of user needs.
System hardening is the process by which a system's configuration is tuned to improve its security without compromising its functionality. The 100% secure system is one that is turned off. System hardening process takes into account:

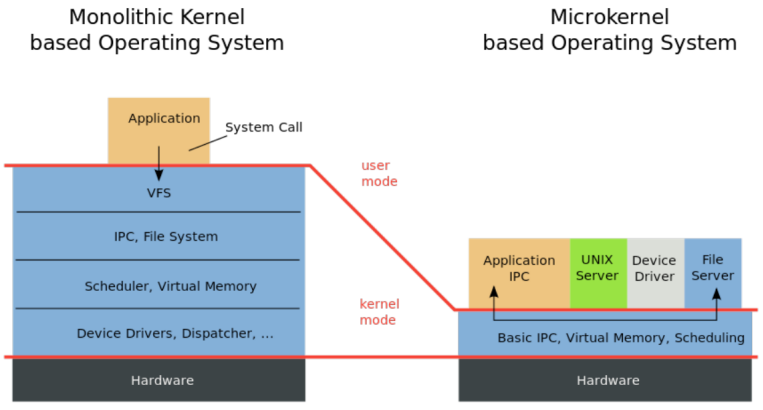
- System **functionality**: what is the role of that system? (Home computer, File server, Web server, general purpose server);
- System **security**: how can the security of the system be improved? (Minimise the attack surface of the system).

Attack surfaces

An attack surface is the set of system resources that are exposed to the attacker, e.g. weak passwords, software vulnerabilities, misconfigurations, services listening on the network or inaccurate access control. The golden rule of information security is the "**Need to know principle**": no user and no system component or process should be authorised or compiled to perform actions that are not strictly necessary for their normal operation (aka "If it's not there you can't brake it"). This principle can be applied at both system, users and processes. A **system** should be configured such that it does not embed or enable functionalities that are not needed for normal operation. A **user** should be authorised to only access and modify resources that are necessary for their normal operation; if a user is NOT authorised, they will NOT be able to accomplish their tasks.

OS design approaches

There can be different versions of an OS: for example some are more focused on security, but if the administrator is not able to select the correct values the overall security may end up being worse.



In the image, microkernel based OS is more secure (kernel contains only what is strictly necessary).

Minimal user privileges

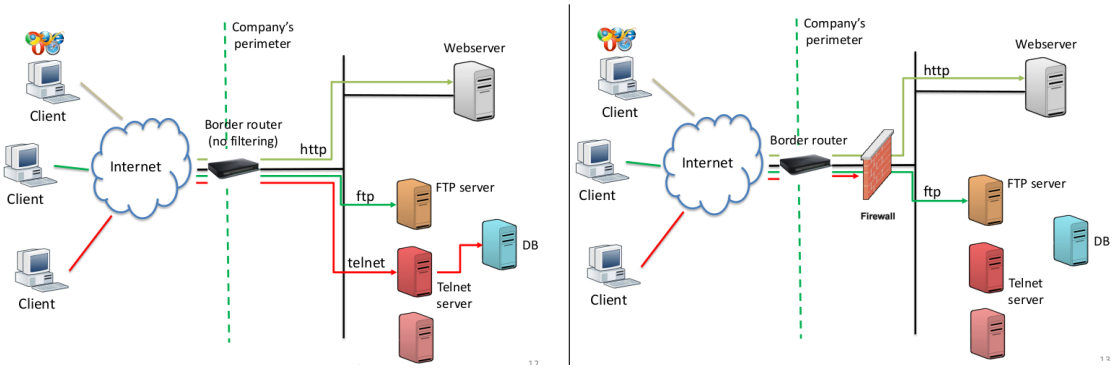
User should not be allowed to perform more actions on the system than necessary for their operation. A common policy requirement is to restrict the behavior of a user; to permit different users to do different things, we need a way to identify or distinguish between users:

- Identification mechanisms to indicate identity
- Authentication mechanisms to validate identity

8 System hardening - Firewalls

A system's minimal configuration may still have a higher attack surface than necessary (e.g. SSH is necessary for remote operation on server, however SSH logins may only be allowed from an internal IP address or other additional network measures to minimise attack surface). Firewalls are perimetral network components that filter incoming (outgoing) traffic from (to) the network, embedded in network devices or as a stand-alone software.

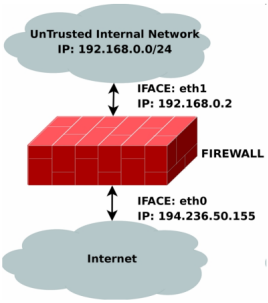
No perimetral defense vs Perimetral defense



Without the firewall every incoming connection can enter in the network, while with it some traffic (telnet in the example) can be stopped. A firewall can centralize security.

Networking with a firewall

Internal network can be treated as untrusted: **do not trust outgoing traffic** (Unintentional Insider Threat, UIT). Connections to remote servers can be regulated (e.g. remote storage services could be used to exfiltrate data from an organisation). Firewalls have at least two network interfaces: one facing the external network (or the router, depends on firewall placement w.r.t border router), one facing internally. More interfaces are possible if the firewall sits at the border with three or more networks.



Firewall Characteristics

The design goals of a firewall are that all traffic from inside or outside must pass through it (physically blocking all access to the local network except via the firewall): only authorized traffic (defined by the local security policy) will be allowed to pass. The firewall itself has to be immune to penetration (use of a trusted system with a secure operating system). There are 2 default policies:

- **Default deny**: All what is not explicitly allowed is denied.
- **Default permit**: All what is not explicitly denied is allowed.

In the case of default permit there is a **blacklist policy**, which is a list of what is blocked. Rules to remove/reduce services are specified when a problem is discovered, so users have more freedom on what they can do. It is suitable for small organizations or home systems. An example of permit policy is to deny incoming ftp traffic and allow everything else; another example could be to deny also incoming telnet traffic (in this case more suitable for open organizations like universities).

In the case of default deny there is a **whitelist policy**, which is a list of what is allowed. Rules to allow a service are added after a careful analysis, so it is more visible to users (users are restricted at what they can do). It is the preferred default policy for business and governmental organizations. An example of deny policy is to allow incoming http and deny everything else.

There are 3 types of firewall: **static packet filtering**, **stateful packet filtering** and **proxies** (application-level gateways and circuit-level gateways).

8.1 Static Packet Filtering

It is often implemented by a router. Applies a set of rules to each incoming IP packet to decide whether it should be forwarded or discarded. Header information is used for filtering (e.g., protocol number, source and destination IP, source and destination port numbers, etc.). It is tradionally **stateless**: each IP packet is examined isolated from what has happened in the past. Simple and fast, meaning a low demand on resources.
In packet filtering there exists access lists, which are the rules to write whitelists and blacklists.

Access lists

They are defined by the CISCO format. There are standard ACLs and extended ACLs.

- **Standard ACLs**

  ```
  access-list $number $action $src [wild card]
  ```

  number identifies the rule, *action* can be accept/deny, *src* is the source ip. *Wild card* is the inverse of subnet mask and says which part of the IP should be checked for and which ignored (e.g. `192.168.3.1 [0.0.255.255]` means that "0.0.3.1" is the subnet of interest).

- **Extended ACLs**

  ```
  access-list $number $action $type $src [wild card]
          $opt $dest [wild card] [log]
  ```

  Type is the IP, tcp, udp, etc, *opt* are the ports for TCP/UDP or type/code for ICMP, *log* writes in a log when the event is triggered.

It is possible to assign values to variables (e.g. `internal_net:=192.168.1.0/24`).

Packet Filtering

Example of (explicit) policies:

1. deny all incoming tcp connections to SSH;
2. allow outgoing TCP connections to SSH.

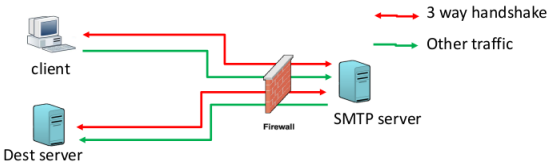
action	src	port	dest	dport	flags	comment
allow	192.168.2.0/24	*	*	22	S or ACK	Our outgoing traffic to remote ssh servers
allow	*	22	192.168.2.0/24	*	SACK	Their SYN ACK
allow	*	22	192.168.2.0/24	*	ACK	Rest of communication

action	src	port	dest	dport	flags	comment
deny	*	*	192.168.2.0/24	22	S	We do not allow remote connections to local SSH servers

Do we actually need this last rule? Yes, if default allow; no, if default deny. Notice that this is last in the list: the first rule that matches is used.

Note of caution

Some protocols are easy to implement, with clear distinction between client and server. Other protocols are not as straightforward. E.g. want to restrict SMTP operations, the SMTP server acts both as a server (receives mail) and as a client (forwards mail to next server). Firewall rules must match both cases.



Exercise: SMTP rules

Explicitly allow incoming SMTP traffic from `10.1.1.1` to SMTP-srv, allow all outgoing SMTP traffic.

action	src	port	dest	dport	flags	comment
allow	10.1.1.1	*	SMTP-srv	25	S xor A	Allow everything from trusted client
allow	SMTP-srv	25	10.1.1.1	*	S ACK	Allow server answer
allow	SMTP-srv	25	10.1.1.1	*	ACK	Allow rest of communication
allow	SMTP-srv	*	*	25	S xor A	Allow initiation of connection to remote SMTP
allow	*	25	SMTP-srv	*	SA	
allow	*	25	SMTP-srv	*	A	
deny	*	*	*	*	*	

Packet Filtering: Pros and cons

Pros:

- Transparent: it does not change the traffic flow or characteristics - either passes it through or doesn't;
- Simple: easy to implement rules to prevent IP spoofing (e.g. no outgoing traffic from non-private IP address space, control and log attempts to remotely connect to private services);
- Cheap.

Cons:

- It does not prevent application-specific attacks;
- Unsophisticated (protects against simple attacks);
- Calibrating rule set may be tricky;
- Limited logging.

8.2 Stateful Packet Filtering

The decision is taken by inspecting some state informations of packets: **Stateful Inspection** or **Dynamic Packet Filtering**. It maintains a history of previously seen packets to make better decisions about current and future packets: the connection state is maintained in a connection table. It defines rules to open state. It is possible to use existent state to control future packets (e.g. explicit rule for TCP SYN in LISTEN state, "NEW" connection in IPTABLES and subsequent packets can be filtered using the connection table, e.g. allow any packet for an ESTABLISHED

connection).

Stateful firewalls allow users to define states over stateless protocols (e.g., UDP traffic is stateless, so it uses `<sip,sport,dip,dport>` to correlate traffic). For these protocols there is no termination sequence (e.g. TCP's FIN 4-way handshake), typically they set a timeout wherein pseudo-state is defined. Traffic of stateless protocols depends on application, not on the protocol itself; it may be hard to manage, because it is application-specific.

Stateful firewall rule example

Possible states (iptables with conntrack):

- NEW: packet trying to open a not-yet existent connection;
- ESTABLISHED: incoming packet is relative to a connection already initiated;
- RELATED: packets that are stating a NEW connection but related existing one (needed by some applications – e.g. FTP);
- INVALID: none of the above, e.g. incoming packet with ACK but not belonging to ESTABLISHED connection (can you filter this with static filtering?).

Say you want to prevent ACK scans:

- Stateful rule:

```
iptables -A INPUT -i eth0 -m state -state INVALID - j DROP
```

- Static rule:

```
iptables -A INPUT -i eth0 -p tcp --tcp-flags ACK - j DROP
```

  However this rule will drop also the legit ACKs of TCP connections.

Another example

Example rule: allow all incoming traffic related to an existing connection

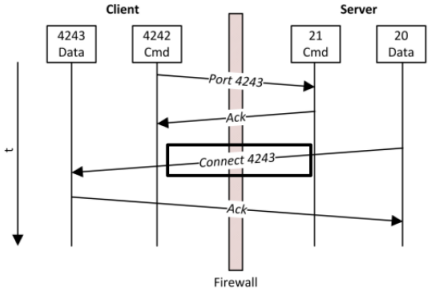
```
iptables -A INPUT -i eth0 -m state -state
    ESTABLISHED,RELATED -j ACCEPT
```

Mixed rules also possible (! for different, -P default policy):

```
iptables -A INPUT -i ! eth1 -j ACCEPT
iptables -A INPUT -m state -state
    ESTABLISHED,RELATED -j ACCEPT
iptables -P INPUT DROP
```

Application firewalls

Statefulness consider also application layer ("Deep packet inspection"; it can keep track of some packets and deny others). E.g. FTP PORT command:



- FTP commands are passed to port 21;
- In "Active mode" the server opens a connection with the client, and chooses dport (this happens with PORT command);
- Application firewall can detect PORT command and act on packet: simple stateless firewall can not manage this.

Stateful and app firewalls: pros and cons

Pros:

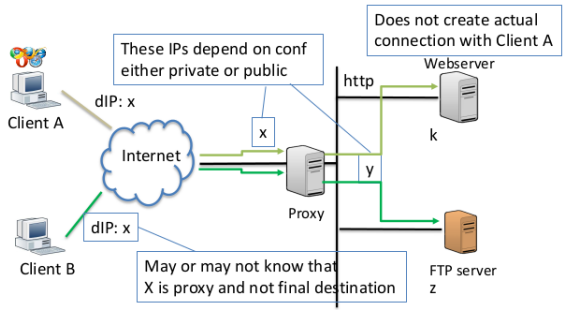
- Allow user to express more powerful rules;
- Policy definition is much simpler than with static packet filtering;
- Very diffused in all modern firewalls.

Cons:

- Severe impact on firewall performance;
- Deep packet inspection significantly slows down packet check;
- Application support may be very complicated (typically provided as "modules").

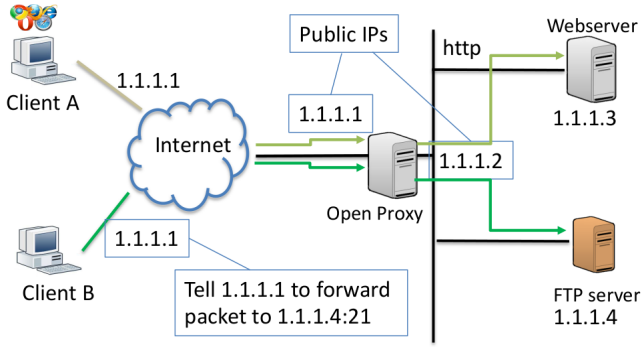
8.3 Proxy

A proxy is a network component that mediates network communications. It untangles the otherwise direct communication between client and server. Proxy acts both as a server (that receives remote connection) and as a client (that forwards the connection to its real destination).



Open proxy

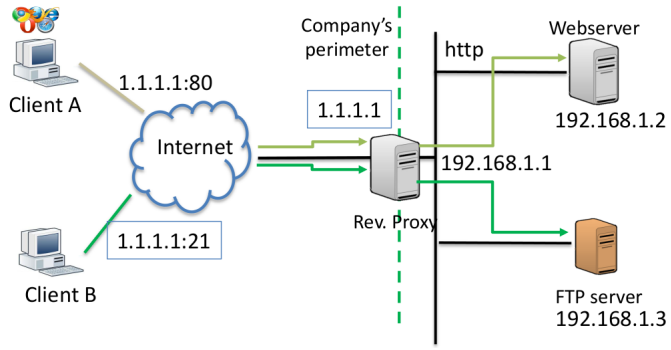
Proxy connects any client on the internet to any server on the internet: clients know th real destination of the packet. Servers can not normally know by whom was the packet originated.



Enables the user to achieve some level of anonymity on the network (*Anonymous proxies*), hiding IP, location and other information. Servers should not be able to collect source IP: some techniques exist to overcome this, like force the client to communicate its IP through third party services or plugins (e.g. flash). Trust issues come from the fact that all trust is put on the proxy service: this may or may not be sufficient depending on application. It is OK to bypass organisation's blacklist (e.g. block facebook.com), probably not trustworthy for more sensible Internet traffic (confidential exchange of information). It may be used as a malware distribution server, where the malicious proxy embeds malware in response packet.

Reverse Proxy

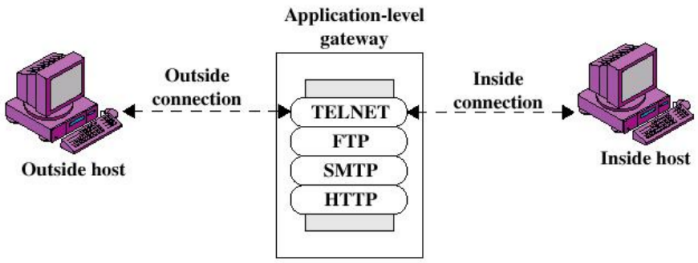
Mediates connection between Internet clients and servers on an internal network it protects. It can embed firewalling capabilities; may sit on border router. Client talks directly to the Proxy, which forwards to internal servers. Neither internal servers or clients know real origin/destination of packet.



It may hide properties of internal servers (IPs, non-custom service ports, versioning). If too aggressive, it may cause disservices (e.g. declares fake server version that breaks the protocol). May be used for load balancing: several internal replicas of a webserver, proxy automatically balances the load by forwarding client's connection to the most appropriate internal server (e.g. least busy server gets the connection). It may also be used to cache server's content, so the answer goes directly to requests for which a cache entry exists.

Application Level Proxy

Also called application proxy, acts as a relay of application-level traffic. All connections are mediated by the GW.



Pros (by not permitting application traffic directly to internal hosts):

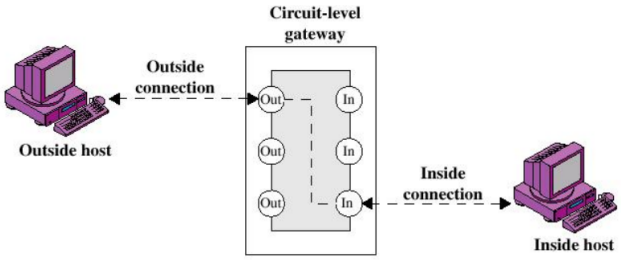
- Information hiding: names of internal systems are not known to outside systems;
- Can limit capabilities within an application;
- Robust authentication and logging: application traffic can be pre-authenticated before reaching host and can be logged;
- Cost effective: third-party software and hardware for authentication and logging only on gateway;
- Less-complex filtering rules for packet filtering routers, easier stateful firewall implementations;
- More secure.

Cons:

- Keeping up with new applications;
- May need to modify application client/protocols (custom implementation may be expensive).

Circuit-level Gateway

Also called circuit-level proxy. Usual, when there is a trust to internal users. No firewalling capabilities, it simply crosses client connection to inside host: the gateway typically relays TCP segments from one connection to the other without examining the content. Operates at L4 on OSI scale. Difficult to find, rarely used.



Firewall Basing

Packet filtering usually comes in a software module in a router or a LAN switch. For application level firewall there is **Bastion host**, a stand-alone machine running common OS (Unix, Windows). Then there are also **Host-based firewall** and **Personal firewall**.

Bastion Host

It is a system identified by the firewall administrator as a critical strong point in the network's security. The bastion host serves as a platform for an application-level or circuit-level gateway. It executes on a secure version of the OS (hardened system), it provides only essential services, may require additional user authentication before accessing proxy services; each proxy service may require also its own. Each proxy maintains detailed audit information, is independent and runs as a non-privileged separate user.

Firewall/Bastion Administration

- Access to management console can be made by dedicated clients using encryption, via SSH and https, possibly using also user authentication;
- Strategies of disaster recovery include switches capable of balancing/failover;
- Logging is ensured by the use of a remote syslog server (centralization of all logs);
- Security incidents have different severity levels, the policy determines which ones are significant: it keeps logs for legal analysis about the attacks and synchronization with a time server is important to know which came first.

Host-based Firewall

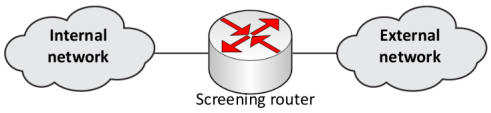
Software module used to secure an individual host, available in many operating systems. A common location for such firewalls is a server. Its advantages are the filtering rules that can be tailored to the host environment (specific rules for the servers), the protection is provided independent of topology (thus both internal nd external attacks must pass through the firewall) and in conjunction with stand-alone firewalls, the host-based firewall provides an additional layer of protection.

Personal firewall

Personal firewall controls the traffic between a personal computer or workstation on one side and the Internet or enterprise network on the other side. It is used in home environment and on corporate intranets, typically a software module on the personal computer, and is easy to configure. It is used to deny unauthorized remote access, plus detect and block worms and other malware.

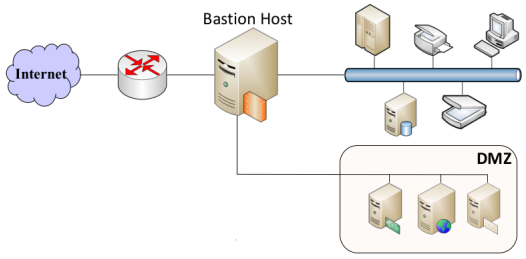
Firewall Topologies (from simpler to more complex)

- **Host-resident firewall**: personal firewall software and firewall software on servers;
- **Screening router** (packet filtering): single router between internal and external networks with stateless or full packet filtering, typical for small office/home office (SOHO) applications. Fast and cheap;

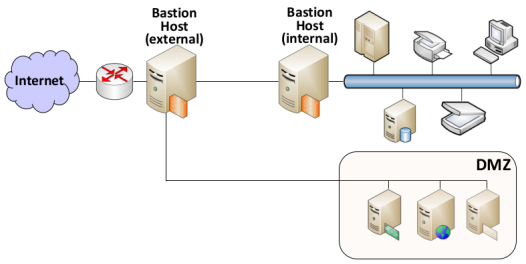


- **Single bastion inline**: when a configuration for the packet-filtering router is that only packets from and to the bastion host are allowed to pass through the router. The bastion host performs authentication and proxy functions. This configuration implements both packet-level and application-level filtering (allowing for flexibility in defining security policy). An intruder must generally penetrate two separate systems;

- **Single bastion T, with DMZ**: similar to Single Bastion Inline but has a third network interface to a DMZ (Demilitarized Zone) where externally visible servers are placed. Common topology for medium to large organizations;



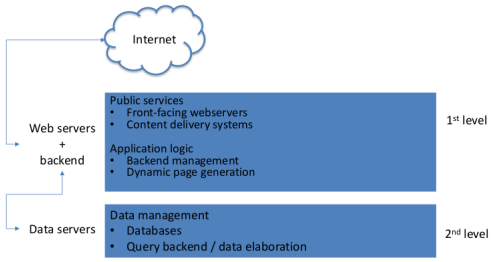
- **Double bastion T**: an extarnal bastion host is dedicated only to the DMZ, so compromising it does not give access to the internal bastion host, which defends the intranet. Extra firewalls to implement means, obviously, more time and money spent.



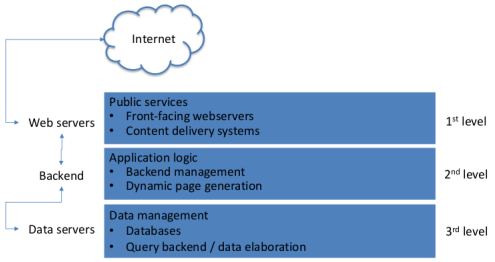
Advanced network topologies

Single and double bastion topologies are adequate only when mapped to a significant **separation of networks**. A good network separation allows for better management of firewall rules, higher control on incoming traffic, higher overall security and lower load on single appliances.

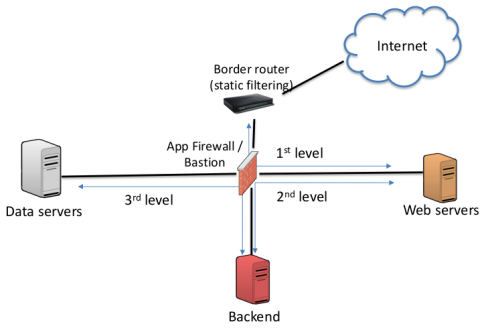
Typical multi-level network applications



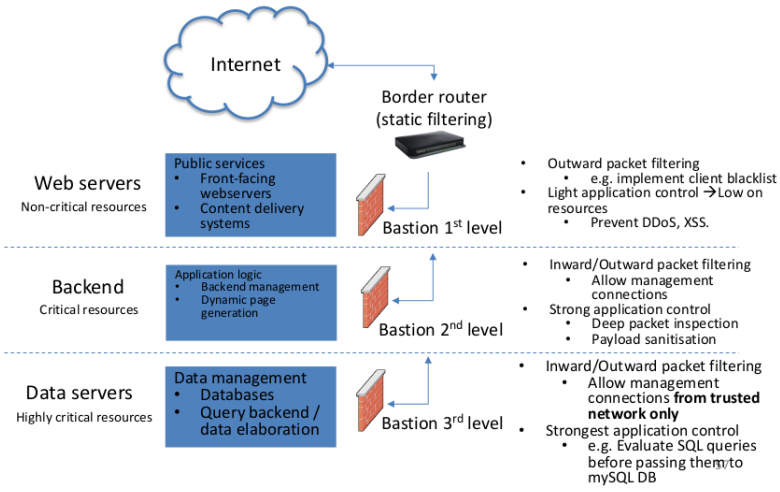
Separate network topology



A simple implementation of this separation:



Divide et impera - Cascade firewalls



Cascade firewalls are inter-dependent firewall policies. Each firewall must be configured considering functions needed at higher levels (e.g. firewall at level 1 must allow all packets eventually directed toward level 2 or 3). In complex networks this is unmanageable if network is not well configured.

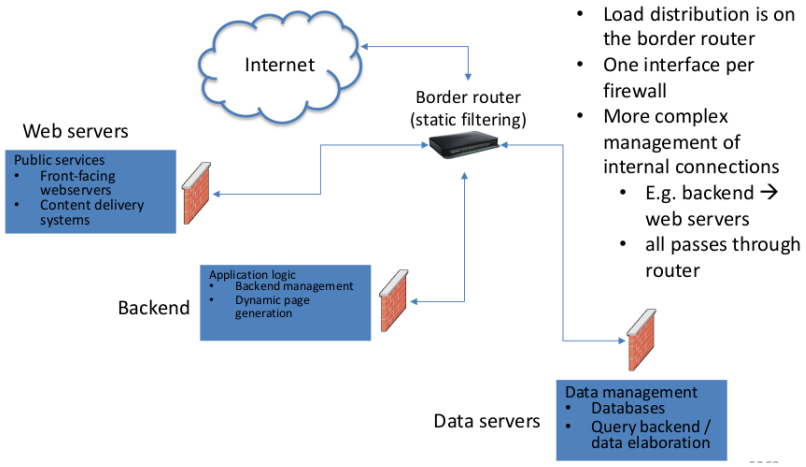
It requires a good mixture of NAT/PAT policies, firewall configurations, and good separation of services (e.g. hard to have effective NAT + firewalling for SSH services at both level 1 and level 3, so where should the packet go? Remember incoming packet will always have address of outward-facing NAT interface toward port 22).

Each layer should ideally be in a different subnet:

- Firewall @ Layer 1: 192.168.1.0/24
- Firewall @ Layer 2: 192.168.2.0/24, etc...
- F1 Accept all traffic that needs to be forwarded to F2

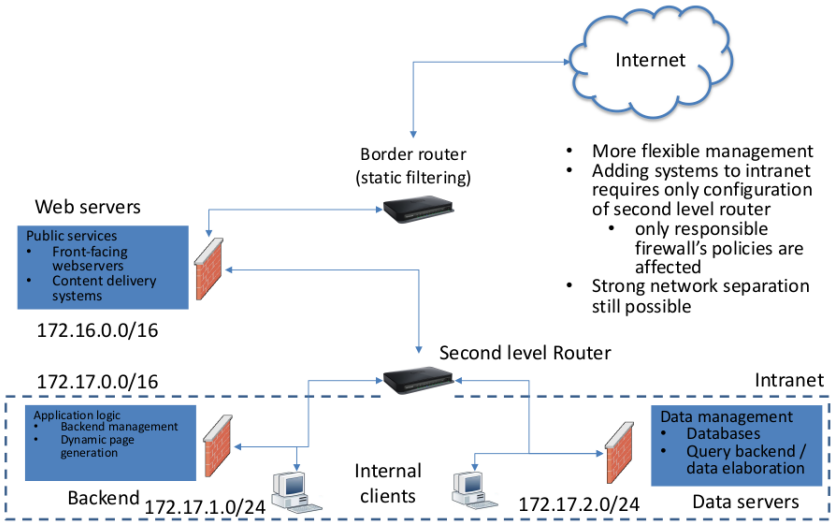
It has high design, management and maintenance costs: introducing a new service at any level requires testing all configuration at lower levels.

Divide et impera - Parallel firewalls



The firewalls run in parallel, with undependent policies.

Mixed architectures



Static filtering, bastion host of web servers and second level router are in cascade, while the following 2 firewalls are in parallel.

9 System hardening - IDS

Firewalls prevent unwanted access to network resources that should be isolated w.r.t. another network. An **Intrusion Detection System** (IDS), instead, monitors connections/activity on a host. The **Intrusion Prevention Systems** (IPS) can act over "malicious" behaviour.

- IDS = passive monitoring
- IPS = active monitoring

In reality functionalities are not entirely distinct (commercial lingo rather than actually different technology).

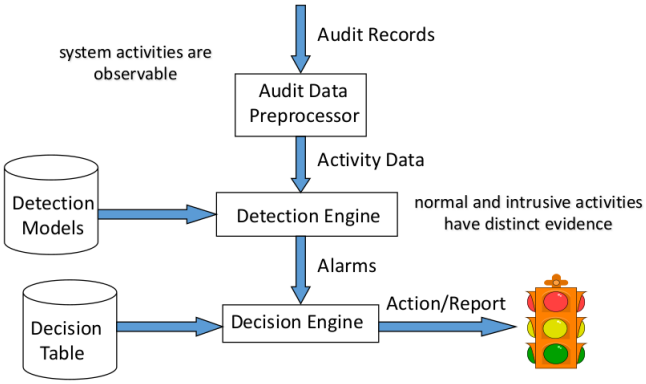
Intrusion is a set of actions, occurring on your system, aimed to compromise the security goals (namely integrity, confidentiality, or availability) of a computing and networking resource. An **intrusion detection** is the process of identifying and responding to intrusion activities.

An IDS can be classified in:

- **Host-based** IDS: monitor single host activity
- **Network-based** IDS: monitor network traffic

The **logical components** of IDSs are:

- sensors - collect data
- analyzers - determine if intrusion has occurred
- user interface - manage / direct / view IDS



IDSs **Requirements** include: to run continually; be fault tolerant; resist subversion; impose a reasonable overhead on system; configured according to system security policies; adapt to changes in systems and users; scale to monitor large numbers of systems; provide graceful degradation of service; allow dynamic reconfiguration.

The efficiency of an IDS system is measured on:

- **Accuracy**: the proper detection of attacks and the absence of false alarms;
- **Performance**: the rate at which traffic and audit events are processed. To keep up with traffic, it may not be able to put IDS at network entry point. Instead, place multiple IDSs downstream;
- **Fault tolerance**: resistance to attacks. Should be run on a single hardened host that supports only intrusion detection services;
- **Timeliness**: time elapsed between intrusion and detection.

9.1 Intrusion Detection Approaches

Intrusion detection works on the base of modeling the attack/intrusion. This modeling works on features (evidences extracted from audit data) which are pieced together in the analysis. Analyse includes **misuse detection** (aka signature-based), **anomaly detection** (aka behavioural-based) and **specification-based detection**.

Misuse detection

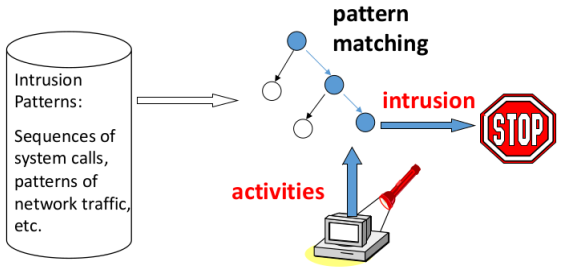
It is the equivalent of the "default allow" policies, so you "blacklist" patterns that are believed to be related to malicious activities; these patterns can be a chain of system calls (host based intrusion) or payloads in network protocols. It is signature-based, with very diffused detection technique and easy to deploy. It has a typical implementation for network-based IDSs. As all blacklisting approaches (signature-based), it can only detect patterns that are already known.

Signatures can be more advanced than a single pattern, it is necessary to observe events on the system and apply a set of rules to decide if there is an intruder. Possible approaches are:

- **rule-based anomaly detection**: analyze historical audit records for expected behavior, then match with current behavior;
- **rule-based penetration identification**: rules identify known penetrations/weaknesses, often by analyzing attack scripts from Internet, supplemented with rules from security experts.

Step	Command	Comment
1.	`%cp /bin/csh /usr/spool/mail/root`	assumes no root mail file
2.	`%chmod 4755 /usr/spool/mail/root`	make setuid file
3.	`%touch x`	create empty file
4.	`%mail root < x`	mail root empty file
5.	`%/usr/spool/mail/root`	execute setuid-to-root shell
6.	`root%`	

A single command/pattern cannot represent an attack, but a chain of commands spotted forming a signature can.



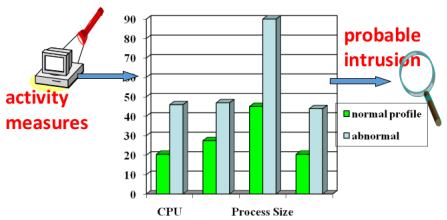
Example: if (traffic contains "`x90+de[^\r\n]{30}`") then "attack detected". The advantage is that it is mostly accurate (no false positives), but it can't detect new attacks.

Anomaly detection

Assumes the intruder behaviour differs from a legitimate profile (unrelated from what the attack does, so can block unknown threats). Building a legitimate profile may be an issue: it depends on data used for profiling (e.g. sampled vs whole dataset) and a profile can evolve, making the new "legitimate activity" looks suspicious. A lot of FP are to be expected, it should be more precise applied on simple systems.
It can be used both for **HIDS** (syscall, system file hashing, system states, ...) and **NIDS** (protocol analysis, similar to application proxy, using monitoring as opposed to filtering). There exists different approaches:

- **threshold detection**: checks excessive event occurrences over time. Alone is a crude and ineffective intruder detector. Both thresholds and time intervals must be determined;

- **profile based**: characterize past behavior of users/groups, then detect significant deviations. It is based on analysis of audit records: it gathers metrics (counter, guage, interval timer, resource utilization), analyzes (mean and standard deviation, multivariate, markov process, time series, operational model) and model as a classification problem (machine learning classifiers).



Defines a profile describing "normal" behavior, then detects deviations; thus can detect potential new attacks. A problem, however, is that it has relatively high false positive rates. Anomalies can just be new normal activities, and be caused by other element faults (e.g., router failure or misconfiguration, P2P misconfig). Which method will detect DDoS SYN flooding?

Specification-based detection

Basically a rigorous way of implementing anomaly detection (on protocols). Specifications that capture legitimate (not only previous seen) system behavior are manually developed: any deviation from it is an attack. Pros is that it can avoid false-positive since the specification and capture all legitimate behavior. However it is hard to develop a complete and detailed specification (may be feasible for simple and specific protocols), and it is error-prone. Approaches:

- state machine, extended finite state automata (EFSA)
- augment FSA with state variables

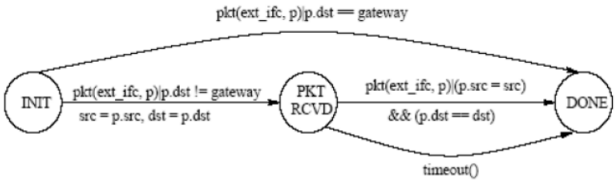
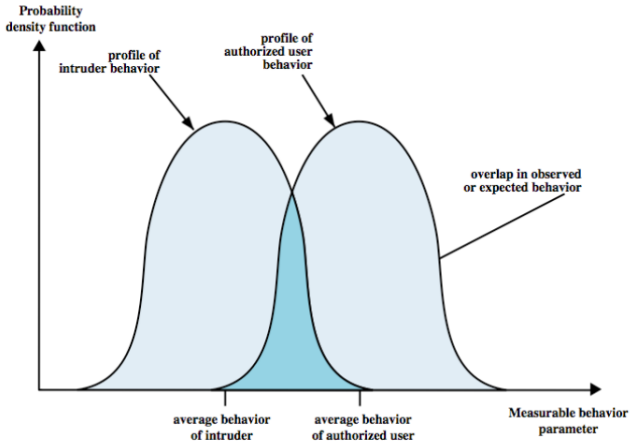


Figure 1: Simplified IP Protocol State Machine

IDS Principles

Anomaly detection leads to errors. Detectors assume intruder behavior differs from legitimate users: an overlap, as shown, is expected, observing deviations from past history. There are problems of false positives and false negatives, so a compromise is required.



Base-rate fallacy - can we have actually good detection rates?

As said, both anomaly and misuses detection necessarily lead to false positives and false negatives. A NIDS with 99% true positive rate and 99% true negative rate seems to have high-reliability alarms. So if an alarm fires up, you should worry, if no alarm fires up, all is good. However these statistics are incomplete because they can mislead about the total amount of cases. So the **base-rate fallacy** is used, which is a simple derivation from Bayes theorem, very well known by medics and doctors and still making its way through in InfoSec.

The base-rate fallacy [Axelsson 2000] states that tests with high true positives and negatives rates yield much "worse" results than expected by the average user. The Bayes theorem is:

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{\sum_{i=1}^{n} P(A_i) \cdot P(B|A_i)}$$

with the denominator being $P(B)$ expanded to all "n" cases for A that B comprises.

Base-rate fallacy example

Let's make the classic medical example where Attack = illness and IDS Alarm = medical test.

- A=event is patient is sick
- B=medical test says patient is sick
- $P(A|B)$ = patient is actually sick given that test said so (equivalent to "there is an actual attack given that NIDS fired alarm")
- set TP=99%; TN=99% so $P(B|A) = 0.99$
- Diseases are rare. Say 1/10.000 people have the illness, so $P(A) = 1/10.000$. Most of network traffic is legitimate.

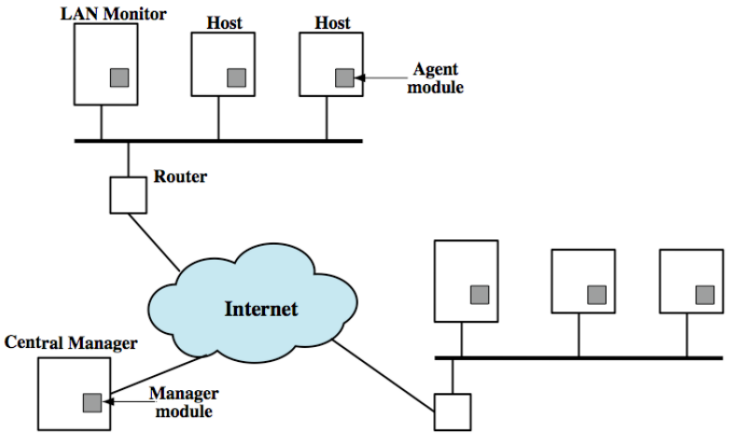
$$P(A|B) = \frac{1/10000 \cdot 0.99}{1/10000 \cdot 0.99 + (1 - 1/10000) \cdot 0.01} = 0.00980... \approx 1\%$$

There is only 1% chance that patient is sick when test says so!

Host-based IDSs

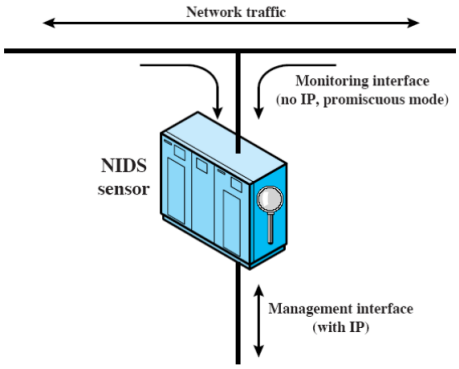
Host-based IDS (HIDS) use OS auditing and monitoring/analysis mechanisms to find malware, can execute full static and dynamic analysis of a program to monitor shell commands and system calls executed by user applications and system programs. It has the most comprehensive program info for detection, thus it is accurate. Problem is that only a local view of the attack is given: if the attacker takes over the machine, it can tamper with IDS binaries and modify audit logs.

Distributed Host-Based IDS



Network-Based IDS

Network-based IDS (NIDS) monitor traffic at selected points on a network in (near) real time to detect intrusion patterns. They may examine network, transport and/or application level protocol activity directed toward systems. A NIDS comprises a number of sensors, inline (possibly as part of other net device) and passive (monitors copy of traffic).



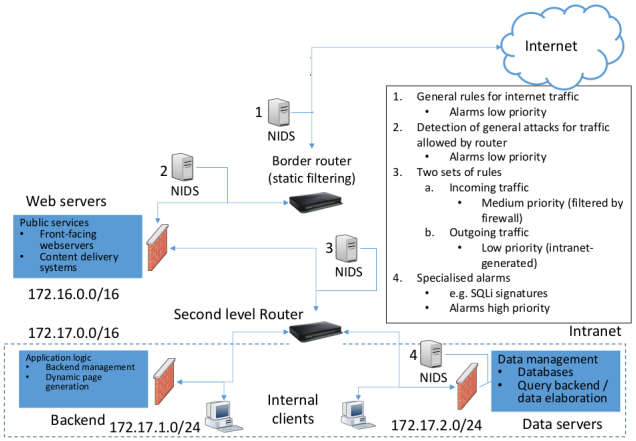
There are usually not just one but more sensors deployed at strategic locations (e.g., packet sniffing via tcpdump at routers). A sensor inspecting network traffic watches for violations of protocols and unusual connection patterns, and rarely look into the packet payload for malicious code. A big limitation is that it is not possible to execute the payload or do any code analysis; even DPI gives limited application-level semantic information. Huge amount of traffic is recorded and processed, so the performance can be easily impacted. NIDS may also be easily defeated by encryption, but this can be mitigated with encryption only at the gateway/prox.

Host-based vs. Network-based IDS

In any setting is more probable to have a combination of the two systems because it can happen that an attack can only be detected by host-based IDS but not network-based IDS. For example a dictionary attack can be detected just by HIDS, while the propagation of a worm just by NIDS.

Architectural aspects

- External NIDS: analysis of all set of incoming traffic. Only general signatures are possible, with high incidence of FP. All detected “attempted attacks” are logged, and "normal" Internet traffic may generate many alarms;
- Internal NIDS: analysis of traffic allowed by the firewall. More specific signatures are possible (e.g. based on services behind firewall, subnet characteristics, ...). It says nothing about attacks attempted but blocked by firewall (it does if joined with an external).



Example: SNORT signatures

SNORT is an open source intrusion detection tool, with a big active community that generates signatures for known attacks, so that it is possible to install already existent rules. A rule is a single line, the rule header is everything before parenthesis and the rule option is what's in the parenthesis. The syntax for a rule header is:

```
rule_action protocol src_add_range src_prt_range
    dir_operator dest_add_range dest_prt_range
```

An example (prevent phising):

```
alert tcp 192.168.1/24 1:1024 -> 124.17.8.1 80
```

- **rule actions**: alert, drop, log, ...
- **protocol**: tcp, udp, icmp, ...
- **direction**: -> and <>
- **src,dst port ranges**.

Another example:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
    (msg: "ICMP PING NMAP; dsize: 0; itype:8";)
```

The rule generates an alert for ICMP having empty payload, ICMP type 8, and arriving from the outside. This is part of an NMAP ping.

Another example:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139
    (msg: "DOS SBMdie attack":; flags: A+;
    content:"|57724c6568004577a|";)
```

The rule generates alert if a TCP packet from outside contains |57724c6568004577a| in payload and is headed to port 139 (netbios) for some internal host. This is part of a buffer overflow attack on a computer running Server Message Block Service.

Another example:

```
alert tcp $HOME_NET 2589 -> $EXTERNAL_NET any
    (msg:"MALWARE-BACKDOOR - Dagger_1.4.0";
    flow:to_client,established; content:"2|00 00 00 06
    00 00 00|Drives|24 00|"; depth:16;
    metadata:ruleset community; classtype:misc-
    activity; sid:105; rev:14;)
```

- dsize: payload size
- offset: XX; (start at byte XX in payload)
- depth: YY; (stop at byte YY in payload)

The baseline implementation of network IDS is of type misuse detection, because it is easier to implement and network traffic is hard to predict even on well-controlled environments. Example of a signature:

```
alert
    tcp $EXTERNAL_NET any -> $HOME_NET 139
    flow:to_server,established
    content:"|eb2f 5feb 4a5e 89fb 893e 89f2|"
    msg:"EXPLOIT x86 linux samba overflow"
    reference:bugtraq,1816 reference:cve,CVE-1999-0811
```

9.2 NIDS evasion [Siddharth 2005]

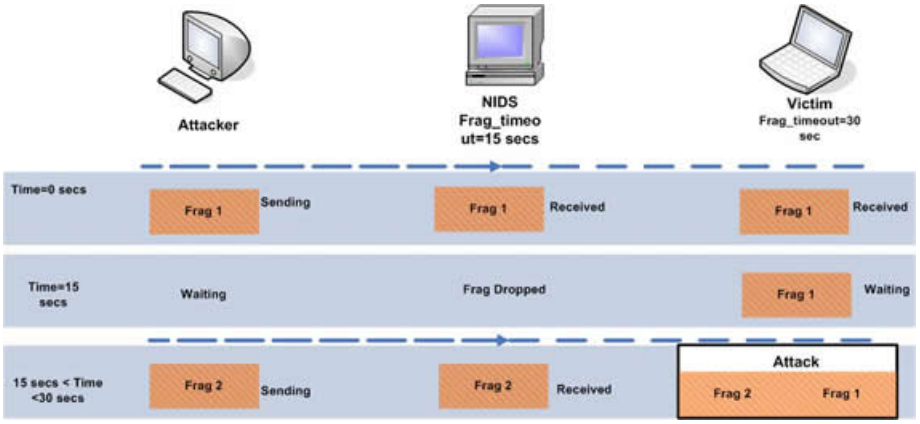
Intrusion detection tools are not perfect. In an article by Siddharth, the evasion technique is described. Signature-based evasion can be fairly trivial when understanding the technicality of the network and how it works in practice. It depends on the implementation of actual signature content: for example `"/bin/bash"` detects remote calls to bash, but does not detect the string `"/etc/../bin/bash"`, etc.
More advanced techniques are typically based on IP fragmentation. All these techniques have a common goal: a NIDS sees a different packet than the client. Look at these keeping in mind you may want to prevent the attacker from performing network mapping and OS fingerprinting.

Evasion technique – Reassembly time-out

This attack is possible because the time-out of the sensors (of the IDS) may be different from the time-out of the victim (host receiving the packet). The time-out is necessary to avoid the DoS attack after the fragmentation, when the receiver is waiting for all the fragments before rebuilding the message.

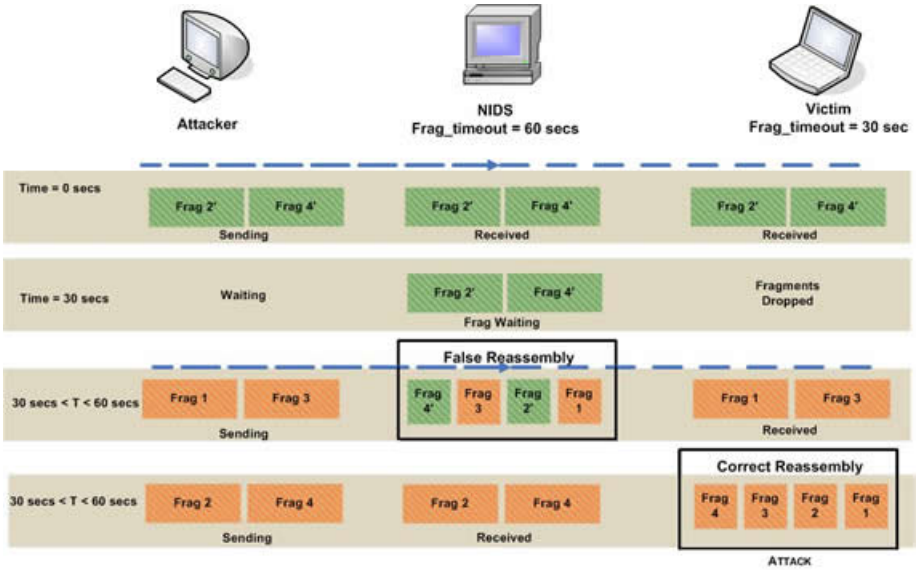
There are 2 cases of this attack.

- **Case 1: The IDS fragmentation reassembly timeout is less than fragmentation reassembly timeout of the Victim.**
  Suppose the IDS fragmentation reassembly timeout is 15 seconds and the system is monitoring some Linux hosts which have a default fragmentation reassembly timeout of 30 seconds. As shown below in the figure, after sending the first fragment the attacker can send the second fragment with a delay of 15 seconds but still within 30 seconds.



  Now, the victim reassembles the fragments whereas at the IDS the fragmentation reassembly timeout parameter kicks in and a timeout occurs. Remember, here the second fragment received by the IDS will be dropped as the IDS has already lost the first fragment, due to time out. Thus the victim will reassemble the fragments and will receive the attack whereas the IDS will not make any noise or generate alerts.

- **Case 2: The IDS fragmentation reassembly timeout is more than the fragmentation reassembly timeout of the operating system.**
  By default, Snort has a fragment reassembly timeout of 60 seconds. Compare that to Linux/FreeBSD where it is 30 seconds. This can be evaded as well. As shown below in the figure, consider that the attacker has fragmented the attack packet into four segments: 1,2,3,4.



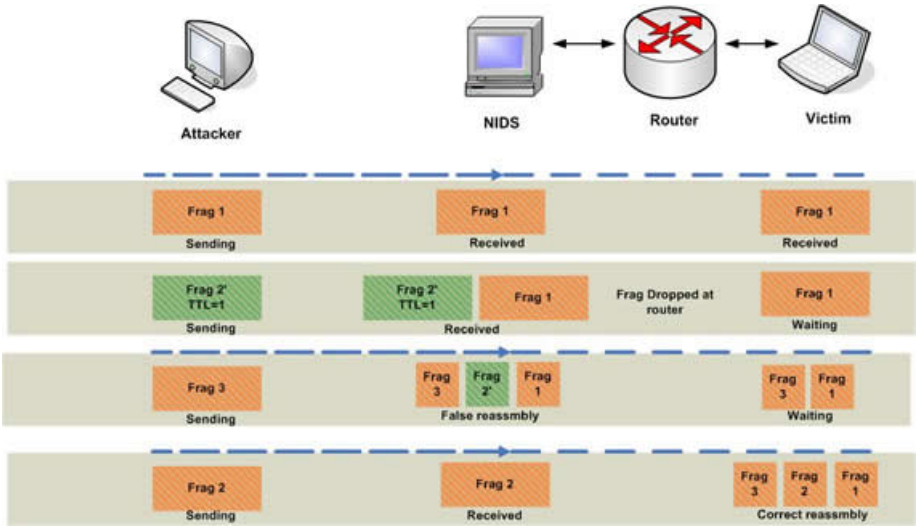
The attacker sends frag2 and frag4 with a false payload (referred as 2', 4'), which are received by both the victim and the IDS. She waits until the fragments' reassembly timeout occurs at the victim's end and it drops the initial fragments (30 seconds in this case).

The beauty of the attack is that the victim still has not received fragment 1 so it will quietly drop the fragments and no ICMP error message will be thrown by the victim. The attacker then sends packet (1, 3) with a legitimate payload. At this stage, the victim has only fragments (1, 3) whereas the IDS has fragments (1,2',3,4'). Remember that the 2,4 fragments sent by attacker have a false payload.

Since the IDS has all the 4 fragments it will do a TCP reassembly. Also, since fragments 2 and 4 have false payloads the net checksum computed will be invalid. So, the IDS will drop the packet. However, now the victim has only fragments 1,3. If the attacker now sends fragments 2, 4 again with valid payload, the IDS will have only these two fragments (2,4 with a valid payload as the previous fragments have been reassembled and dropped) whereas the victim will have all (1,3,2,4) fragments all with a valid payload, and it will do a reassembly and read the packet as an attack.

Evasion technique – Time-to-live

These attacks require the attacker to have a prior knowledge of the topology of the victim's network. This information can be obtained by using tools such as traceroute which give the information on the number of routers between the attacker and the victim. A TTL based attack is shown below.

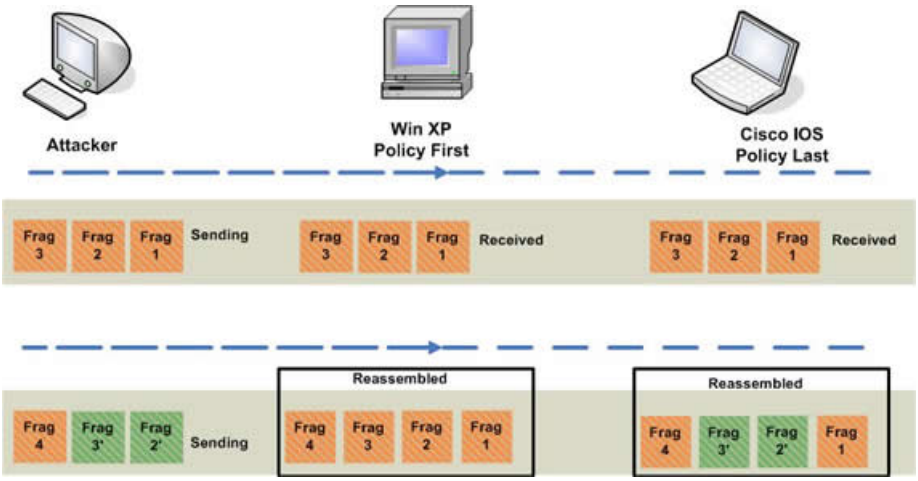


A router is present between the IDS and a victim - and the attacker is assumed to have this prior information. The attacker carries out the attack by breaking it into three fragments. She sends fragment 1 with a large TTL value and this is received by both the IDS and the victim. However, the second fragment (frag2') sent by the attacker has a TTL value of 1 and also has a false payload. This fragment is received by the IDS whereas the router (which is situated between the IDS and the victim) discards it as the TTL value is now reduced to zero.

The attacker then sends fragment 3 with a valid TTL. This makes the IDS perform a TCP-reassembly on fragments (1,2',3), whereas the victim still waits for the third fragment. The attacker finally sends the third fragment with a valid payload and the victim performs a reassembly on fragments (1, 2, 3) and gets the attack. At this stage, the IDS has only fragment 3 as it has already performed a reassembly and the stream has been flushed.

Evasion technique – Fragment replacement

Some operating systems replace fragments with newer ones, others keep old fragments.



The attacker carries out the attack by breaking it in 4 fragments. She sends fragment 1, fragment 2 and fragment 3 first, which both operating systems accept. Now the attacker sends fragment 2', fragment 3' and fragment 4. Here the payload of fragment 2' and fragment 3' is different from fragment 2 and fragment 3, respectively, but the fragment offset and the length of the fragment along with other fields in the IP-header remain the same. In such a scenario, an operating system doing a fragment reassembly based on policy First will do a reassembly on fragments 1,2,3,4 whereas the operating system doing a fragments reassembly based on policy Last will reassemble fragments 1,2',3',4.

9.3 Audit Records

Logs are a fundamental tool for intrusion detection. Two variants:

- **native audit records** (provided by O/S): always available but may not be optimum;
- **detection-specific audit records** (IDS specific): additional overhead but specific to IDS task, often log individual elementary actions (e.g. may contain fields for: subject, action, object, exception-condition, resource-usage, time-stamp).

9.4 Security Information and Event Management (SIEM)

SIEM is a system combining **Security Information Management** (SIM) and **Security Event Management** (SEM). SEM deals with real-time monitoring, correlation of events and threat intelligence, notifications and console views. SIM deals with long-term storage and analysis and reporting of log data.

SIEM Workflow

Defines methods of collecting data from sources.

- **Aggregation**: to gather data together as a whole in singular repository;

- **Normalization**: to create consistent records by type and format. Data coming from different sources have to go through this to avoid inefficiency in the analysis.

Links events to identify attacks.

- **Event based**: a single event identifies an attack;
- **Rule based**: if $X + Y + Z$ then do A, if X repeated 3 times within an hour, then do Y;
- **Anomaly based**: if the traffic on port X exceeds the standard deviation of historic traffic patterns, then there may be a problem.

Severity is usually scored in a scale that goes from Low to Critical.

Upon identifying a threat, notifications are sent to the security administrators (SOC, Security Operation Center). There may be an automated response: the majority of SIEM tools can execute external scripts to react on identified threats (change to FW rules, issue a Remedy ticket).

Collected log data is stored for future forensic investigations; this phase is not equivalent to **Log Management** solutions.

SIEM vs. LM

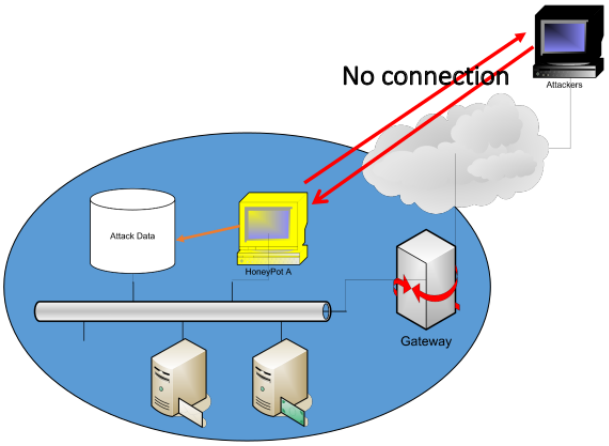
Functionality	SIEM	Log Management
Log collection	Security related logs	All logs
Log pre-processing	Parsing, normalization, categorization, and enrichment	Indexing, parsing, or none
Log retention	Retain parsed and normalized data	Retain raw log data
Reporting	Security focused reporting	Broad use reporting
Analysis	Correlation, threat scoring, event prioritization	Full text analysis, tagging
Alerting and notification	Advanced security focused reporting	Simple alerting on all logs
Other features	Incident management, analyst workflow, context analysis, etc.	High scalability of collection and storage

9.5 Honeypots

Honeypots are decoy systems filled with fabricated informations (that look real), instrumented with monitors/event loggers, that divert and hold attacker to collect activity info, without exposing production systems. Initially were single systems, more recently are/emulate entire networks.

Physical V.S. Virtual Honeypots

- **Physical**: real machines, with own IP Addresses, often high-interactive;
- **Virtual**: simulated by other machines that respond to the traffic sent to the honeypots and that may simulate a lot of (different) virtual honeypots at the same time. More limited than the physical because modern attacks can distinguish them more easily.



Honeypots cann be used for **production** and for **research**.

Production HPs: Protect the systems

Prevention is usually not the scope, keeping the bad guys out is not effective as a prevention mechanisms: Deception, Deterence, Decoys do NOT work against automated attacks (worms, auto-rooters, mass-rooters). Usually an honeypot is effective for **detection** (detecting the burglar when he breaks in) and **response** (can easily be pulled offline, has little to no data pollution).

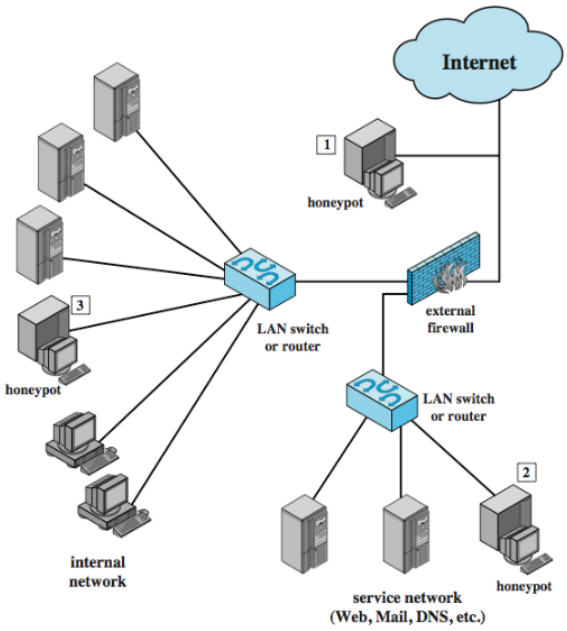
Research HPs: gathering information

Collect compact amounts of high value information, discover new Tools and Tactics, understand Motives, Behavior, and Organization, develop Analysis and Forensic Skills.

Honeypot deployment

An important aspect for honeypots is where they should be placed.

1. **Outside the external firewall**: it is useful for tracking attempts to connect to unused IP addresses with the scope of the network. However, being outside, it does not decrease the risk for the network and has also the disadvantage that it has little to no ability to trap internal attackers: it sees only external traffic, especially if the firewall filters traffic in both directions;



2. **Inside the DMZ**: the security administrator has to be sure that the other systems in the DMZ are secured against the activity generated by the honeypot. A disadvantage is that tipically the DMZ is not fully accessible and the firewall blocks traffic to the DMZ that attempts unneeded services. Removing those restrictions in the firewall might be dangerous;
3. **Inside the network**: it is possible to catch internal attacks (the most dangerous), detect misconfigured firewalls that forward unpermitted traffic. However, as seen previously, a path to reach the honeypot has to be left open for the attacker, which if it compromises and subverts the machines without someone realising it, really gets in the heart of the system.

10 Privacy issues

Recall: Outright malicious attacker

Typically the malicious attacker aims at reading or modifying the communication (in part or fully). In this contest, this attacker is typically called "man in the middle" (or "man in the browser"). The attacker can intercept and act upon a communication between client and server, using channel redirection, blocking communication entirely, using spoofing, etc. An example is the injection of malicious content, causing manipulation of server response (client's answer can also be modified by the attacker) or connection hijack (attacker injects him/herself in the communication and spoofs the victim's identity).

Recall: Honest-but-curious attacker

The goal of this attacker is to use the client's information after correctly handling the service. It typically resides at the service level (e.g. ISP, router) and typically implies confidentiality and possibly integrity loss. For example the DB Server is the attacker, providing agreed service correctly (e.g. answers queries with correct data): after the query is delivered to the client however, the server uses the query's information to perform user profiling.

10.1 Browser cookies and tracking

Cookies are set by the server during an HTTP answer, on SOP (Same Origin Policy) basis, and are used to set variable's values that are useful at the service level. For example, server sets cookie "ThemePreference", defining a value; at the next interaction, client will send "ThemePreference" to server with the obtained value. Attributes that can be defined at cookie level are:

- Name (of cookie) (User);
- Content (value of cookie) (mario);
- Host (name of the server that set the cookie) (mario.net) → remember the SOP, browser sends cookies only to the domain who created them;
- Path (server path onto which the cookie is valid) (/);
- Send for (all connections/ only encrypted);
- Expires (expiry date) (19 Giu 2020).

Different cookie types exists, based on the **attribute**:

- Temporary (session cookie): typically deleted at end of session (expires: NULL);
- Persistent: remain until expiry date (expires: `Tue, 19-Jun-2020`);
- Secure: set by a domain communicating over an HTTPS channel over SSL/TLS (secure transmission, harder to intercept).

Becuase of the SOP, the domain can be any domain-suffix of URL-hostname, except TLD. For example with as host `"login.unitn.it"`, allowed domains are `login.unitn.it` and `.unitn.com`, while disallowed domains are `user.unitn.it`, `unifi.it` and `.it`.
`login.unitn.it` can set cookies for all of `.unitn.it` but not for another site or TLD. The protocol (and port) must be also the same (in IE is slightly different), http is not https. The path can be set to anything.

There are also different cookie types based on the **setting**:

- Third parties: set by domains other than the one requested by the user, can be used to track them;
- Supercookies: like cookies, but associated to first-level domain names (e.g. `.com` or `.it`). `Malicious.it` can read supercookies set by `anotherdomain.it` ("same origin" policy).

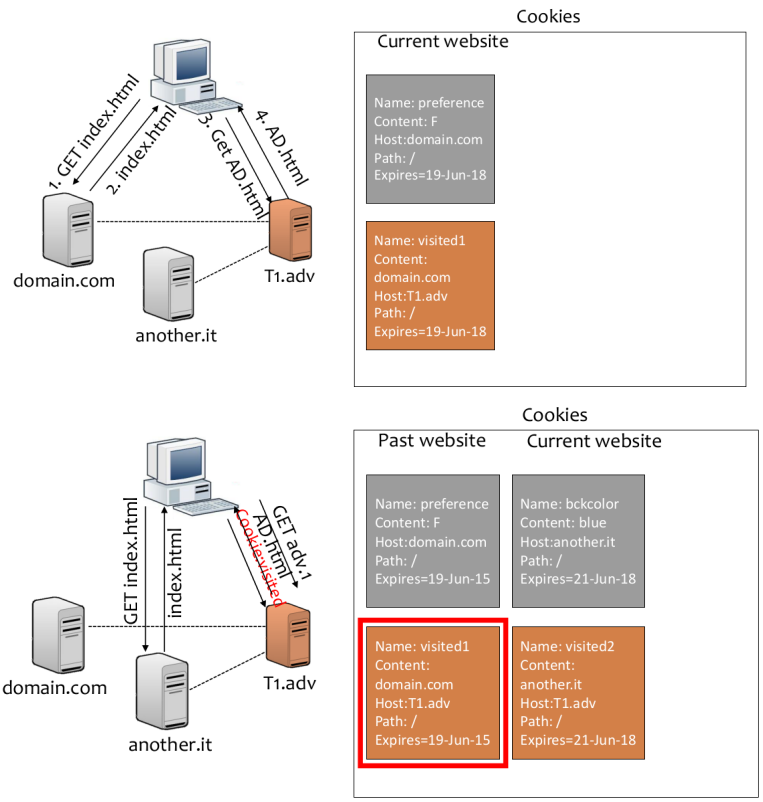
Third party cookies

Cookies can be set by domains called by the browser, which not necessarily correspond to the domain displayed in the address bar. E.g. Requests from `www.ilpost.it`:

304	GET	tween.js	advhd.banzaiadv.it	js	3.05 kB	8.81 kB
200	GET	codice_tabExpand_v4.js	advhd.banzaiadv.it	js	5.26 kB	15.45 kB
304	GET	video_native_post.js	advhd.banzaiadv.it	js	1.27 kB	3.29 kB
304	GET	style_300x100.css	advhd.banzaiadv.it	css	0.31 kB	0.58 kB
304	GET	player_video.css	advhd.banzaiadv.it	css	1.51 kB	5.73 kB
200	GET	style.css	advhd.banzaiadv.it	css	1.05 kB	4.31 kB
200	GET	blocco_classi.js	advhd.banzaiadv.it	js	3.34 kB	10.06 kB
200	GET	300x250.gif	advhd.banzaiadv.it	gif	54.19 kB	72.53 kB
304	GET	sdk.js	connect.facebook.net	js	52.28 kB	163.97 kB
200	GET	css?family=Open+Sans+Condensed:300	fonts.googleapis.com	css	0.43 kB	0.43 kB
200	GET	count-data.js?2=http://www.ilpost.it/20...	ilpostnews.disqus.com	js	0.39 kB	0.99 kB

Other domains can be contacted by the server on behalf of the client (e.g. third party services like facebook, or advertisers). These services can be requested by multiple, unrelated domains, which are managed by different organizations, collecting diverse data about the same user, and complying to different policies may use/embed the same third party service. This way third party services can track users over different domains.

10.1.1 Attacks: honest-but-curious attacker - tracking



Tracking: a persistent case

Almost anybody as a Facebook account. When you visit `www.facebook.com`, FB sets cookies on the browser. That's however now persistent behavior among majority of domains. In a web page with "share on FB" tools, even without owning a FB account, 3rd party cookies are set anyway when loading those page elements that are not on the requested domain, making tracking still possible. Another example is Captcha by Google, which is offered for free but also enables tracking over different domains.

Supercookies

Supercookies are not limited to a single domain, but rather to a first level domain; they can be stored in the cache. They are usually set by proprietary plugins (e.g. Flash, Silverlight), so the browser cookie deletion procedure does not affect them. Because of this, they are permanent (no expiration date), contain more info (<100KB vs <4KB of standard cookies) and are saved also when using "private browsing mode". Now Flash API permits deletion of supercookies from the browser interface.

Zombie cookies

Zombie cookies were generated by the Flash plugin, which was also recording them on a browser: if one expired and was deleted, the plugin would recreate them, allowing the tracking beyond the life of the cookie.

HSTS (HTTP as Strict as Transport Security)

It is a protocol that enforces the use of HTTPS, by storing a single bit. A researcher explained how to use it to breach the privacy and do tracking: by inserting in a single webpage 32 different links to 32 different websites, a string of 32 bit would be created, useful to identify the users.

Private browsing

Private browsing does not prevent user tracking or identification, it only disassociate past browsing history from future. Past browsing history and browser cookies can not be accessed by websites visited using private browsing. It is supported by all major browser now. Some type of supercookies can be passed by in between private sessions (not with Firefox).

Browser extensions

Browser extensions are basically third-party code that is executed by the browser. Trust issue is relevant: the browser will trust the code, but should you? Some extensions can help the user in preserving (or limiting violations to) his privacy online, like AdBlock (blocks ads and other tracking content), Ghostery (like AdBlock, but specialised in tracking, however it is closed source and it may be re-selling anonimized browsing data to advertisers), uBlock (open source, more memory efficient) and noScript (guerrilla version of the above, blocks all JS/scripts). Browser extensions allow the user to add new functionalities to the browser. They are typically written in JS and can access browser environment using APIs (i.e. software interfaces). Some APIs may allow the extension to access information outside of the private browsing env. Some extensions are clearly a security threat, e.g. Firefox' commandrun extension which can access all open browser windows: if private browsing does not close current session (e.g. till Firefox 20), the extension can reach over and link private and non-private sessions.

Plugins

Plugins pose a similar problem. They do not directly depend on the browser, they are third party applications that may or may not comply to the browser's (security) policies and do not follow the SOP. They can set cookies and supercookies, communicate system's IP address and have direct access to system functionalities (because of this, Chrome executes Flash in a sandbox).

Browser fingerprinting

Tracking typically happens using cookies. It is however possible to achieve reasonable tracking precision even for users with a "clean" browsing history. Browser Fingerprinting is a technique that can uniquely identify a browser over a set of rather stable metrics: User agent, header HTTP, screen resolution, plugin/fonts, codecs and supercookie settings (`https://panopticlick.eff.org`). Fingerprint's precision increases with the uniqueness of the user's configuration: the more you "personalize" your browser, the least common its configuration will be (e.g., disable 3rd party

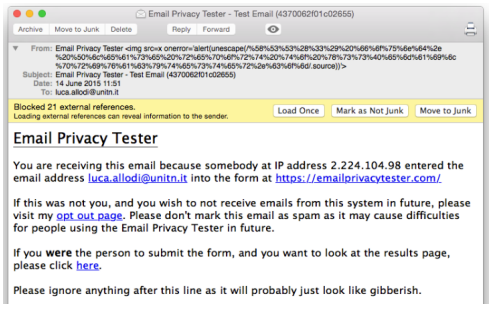
cookies, install Ghostery, install uBlock, kill plugins, install new system fonts, etc). However, being unique is not the same as being trackable. Fingerprint must be stable in time, or at least change in a somewhat predictable manner. Some implementations can predict a browser's fingerprint with good precision (65% detection, 99.1% true positives).

10.2 Attacks out of the browser: email

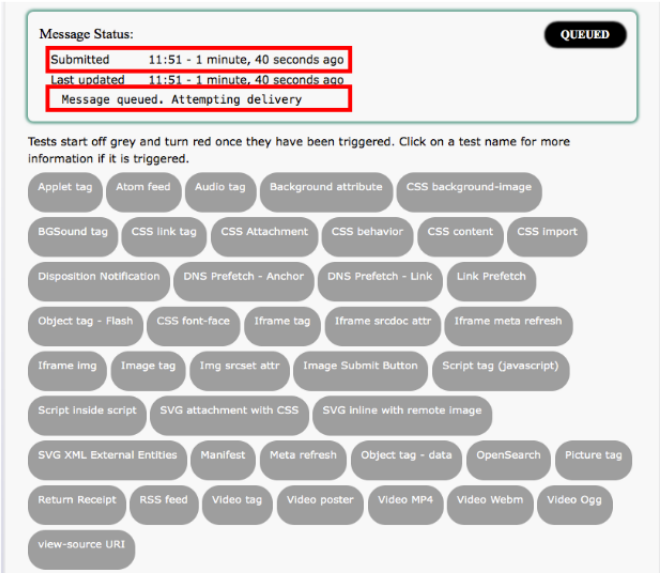
We already know that email is an attack vector for social engineering attacks such as phishing. There are however other, more technical attacks that allow the attacker to obtain private information from within the email client. Emails are basically webpages that can include a number of objects (video, picture, sound files, Javascript, VB script, CSS, iFrames). This can be exploited by the attacker to access to information about the user and/or deliver remote attacks to the email client. An example of type of information: this email address is valid, therefore I can send spam to it; the user appears to be Italian, and works/studies at the University of Trento; the user read this email on day X at time Y from the IP address Z.

Example

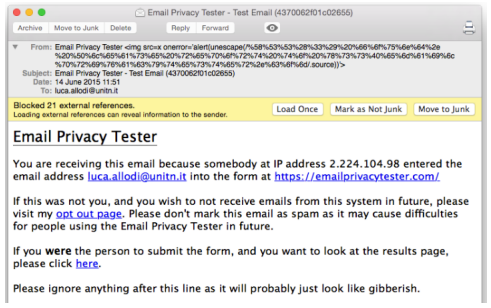
`https://emailprivacytester.com` (Mike Cardwell)



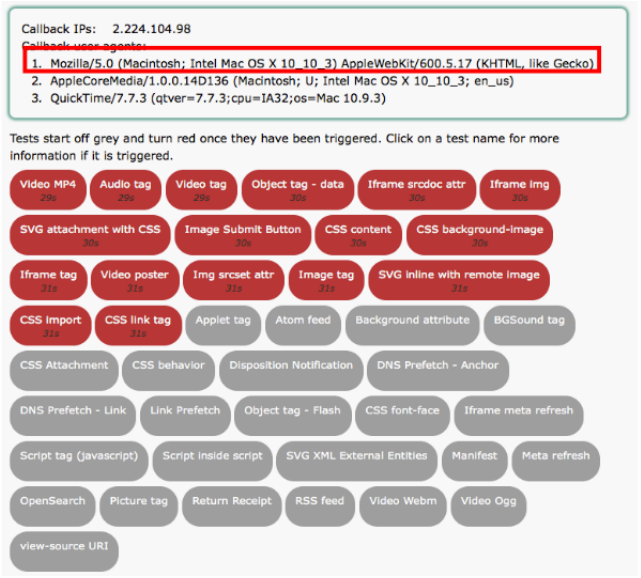
The tool collects some information obtained by the email, as shown in the image on the right.



After a second email sent:



The tool is now able to use all the collected information to understand behaviour and make the client unique.

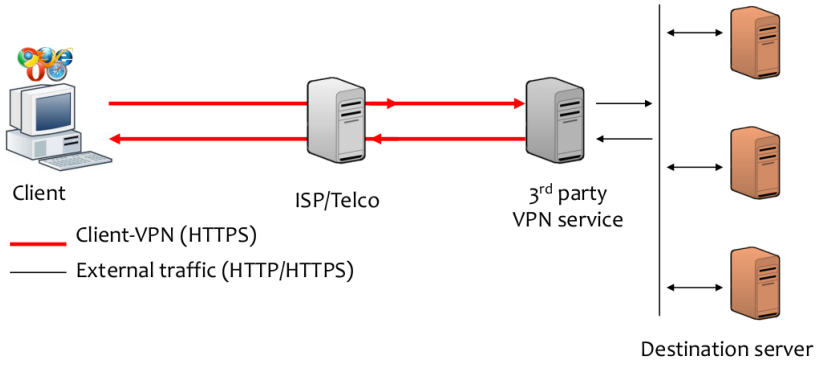


10.3 Source Confidentiality

The source of a connection is a type of identification, which one may want to hide. Different tools exist to do so.

VPN services / Secure proxies

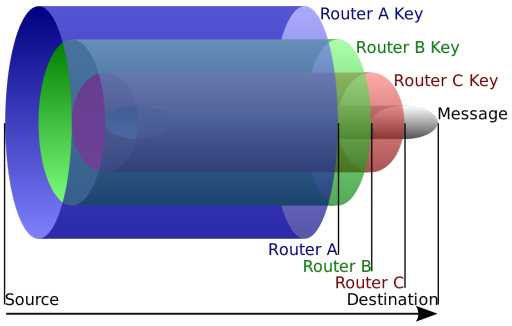
With a VPN (Virtual Private Network) service, users can decide to trust a proxy for his/her connection and send all traffic to it, since the ISP or the destination server may not be fully trusted. This way, the ISP sees only traffic toward VPN and does not know the final destination. Moreover the final destination does not know the real origin of a request.



In this case you have to trust the proxy, which may collect informations of all the sites you visit.

Onion routing

What to do if you can't trust a VPN server (or if it is blocked by the ISP)? Onion Routing is a protocol that puts multiple layers of encryption (as in an onion) around the protocol, and these layers are removed at subsequent hops. It wants to prevent a single party (a hop) to know who sent the message, to whom the message has been sent and what is the message.



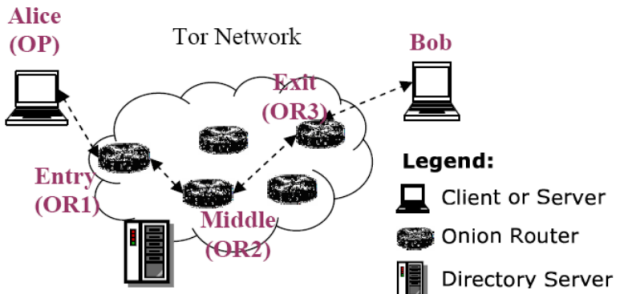
TOR

Tor is a virtual distributed network that allows the user to achieve high privacy levels thanks to Onion routing. It allows the user to connect to a certain service with intermediary infrastructural nodes knowing (e.g. ISP, proxy). Even the final destination never knows who really sent the request. It creates a virtual network with known nodes:

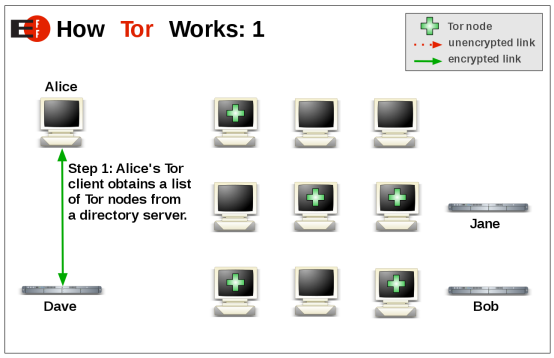
- Onion Routers (OR): route the traffic
- Onion Proxy (OP): creates the virtual circuit ($OR_1 \rightarrow OR_5 \rightarrow OR_2 \rightarrow OR_{\text{EXIT}}$) to route the traffic

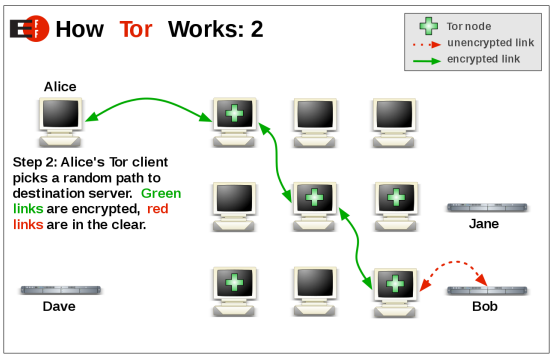
Traffic is sent through TLS.

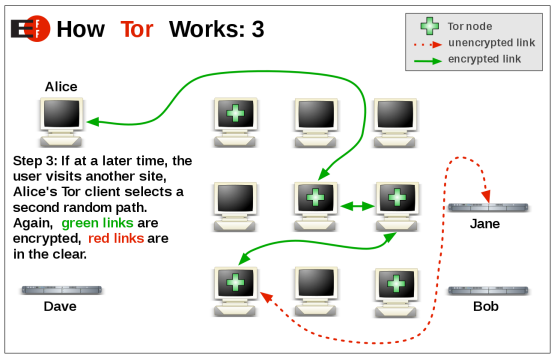
TOR: structure



TOR: in action

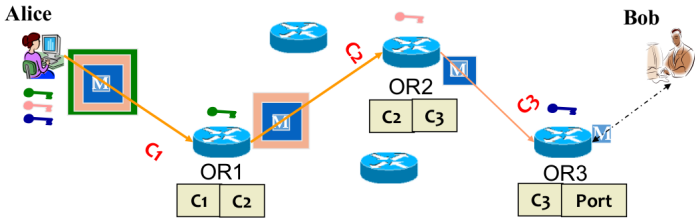






TOR: in detail

A closer look: Alice (OP) negotiate a key with every OR. Every OR only knows who is before and after it. OR3 for example knows that the message is for Bob but does not know Alice sent it.



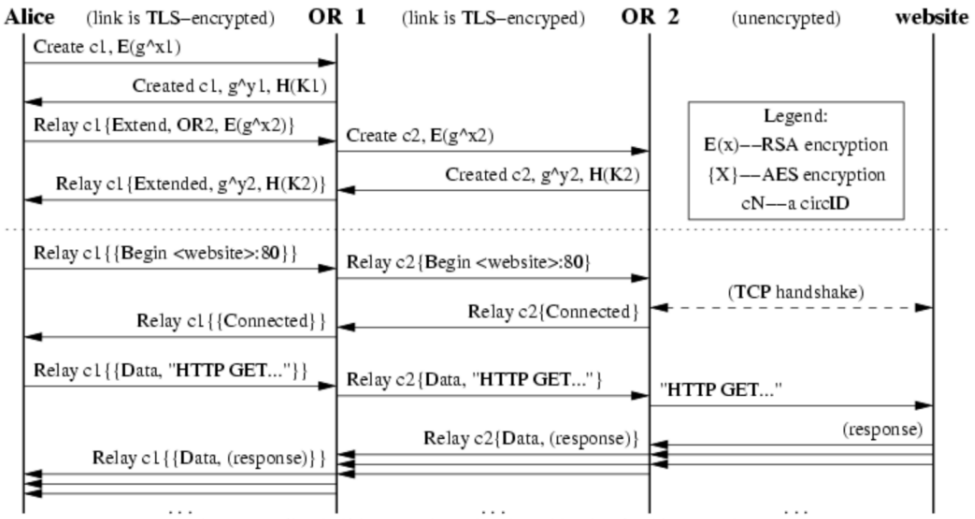


Figure 1: Alice builds a two-hop circuit and begins fetching a web page.

TOR: attacks

Exit node sees the original traffic: so if username and password are in the clear, we start all over again. **Timing-channel attacks** exist: it is possible to infer who's Alice by measuring how much time it passes between subsequent requests toward Bob.
Not all the traffic generated from the system necessarily passes through TOR: DNS requests (e.g. made by browser plugins) may reveal IP address or Javascript/browser extensions may reveal the IP too (apparently the FBI was able to find the owner of Silk Road, the infamous "darkweb" market, using this attack).