Politecnico di Milano

Computer Security

Short Summary

Authors:
FabC07

July 12, 2018

Contents

1 Fundamentals of cryptography

1.1 One-time-pad

- XOR of a message m and a random key k of the same size of m: $len(k) = len(m)$
- The key is pre-shared and consumed while writing. It can never be re-used again
- Very inconvenient and used only in special cases

1.2 Symmetric encryption

- Use key K to encrypt plaintext in ciphertext. Use same key K to decrypt ciphertext in plaintext
- Two important concepts:
  - Substitution: replacing each byte with another (e.g. Caesar cipher)
  - Transposition: swapping the values of given bits (e.g. matrix)

1.3 Asymmetric encryption

- a cipher that uses two keys:
  - What is encrypted with key1 can be decrypted only with key2 (and not with key1 itself), and viceversa
  - The keys cannot be retrieved from each other
- Diffie-Hellman Exchange algorithm:
  - based on factorization problem
  - Pick p prime, a primitive root of p, public (both Alice and Bob know)
  - Pick a number X in $[1, \ldots, (n-1)]$ (this is secret and is the private key of Alice $X_A$and Bob X_B)
  - Obtain the public key as $Y_A = \left(a^{X_A}\right) mod\,(p)$ (for Alice) and $Y_B = \left(a^{X_B}\right) mod\,(p)$(for Bob) and exchange them
  - Compute the common secret as $K_A = \left(Y_B^{X_A}\right) mod\,(p) = K_B = \left(Y_A^{X_B}\right) mod\,(p)$
- RSA algorithm:
  - based on factorization problem
  - Pick p and q two large primes

  - Computing n = p * q is easy, but given n it is painfully slow to get p and q
  - Different problem than mod-log (D-H)
  - At least 1024-bit key (typical choice are 2048-bit or 4096-bit)

1.4 Important notes

- Security of a cryptosystem is based on the robustness of the algorithm
- No algorithm, save the one-time-pad, is invulnerable
- An algorithm is broken if there is at least one attack faster than brute forcing
- There is no way to prove robustness of a cipher, save by trying to break it
- Secret algorithms are insecure. Security is transparency
- Key comparison:
  - You can compare symmetric algorithms based on the key (e.g., CAST-128 bit "weaker" than AES-256)
  - You cannot compare asymmetric algorithms based on key length
  - More importantly, never compare asymmetric vs. symmetric key lengths!

2 Hash function

2.1 Definition

A function H( ) that maps arbitrary-length input x on fixed-length output, h. Possibility of collisions because codomain "smaller" than domain. An hash function should be:

- Preimage attack resistant (by knowing h it should hard to find an x s.t. $H(x) = h$)
- Second preimage attack resistant (by knowing x and $H(x)$ it should be hard to find y s.t. $H(x) = H(y)$)
- Collision resistant (it should be hard to find $\{x, y\}$ s.t. $H(x) = H(y)$)

Notice that random collision can happen in $2^{\frac{n}{2}}$ cases due to the birthday paradox

3 Authentication

3.1 Three factors of authentication

- Something that the entity knows (to know) e.g.: password, PIN, secret handshake
- Something that the entity has (to have) e.g.: Door key, smart card, token
- Something that the entity is (to be) e.g.: Face, voice, fingerprints

Multi-factor authentication uses two or three factors

3.2 "To know" factor

- Advantages:
  - Low cost
  - Ease of deployment
  - Low technical barrier
- Disadvantages:
  - Stolen/snooped (intercepted)
  - Guessed
  - Cracked (enumerated)
- Countermeasures (costs):
  - Change/expire frequently
  - Are long and have a rich character set
  - Are not related to the user

3.3 "To have" factor

- Advantages:
  - Human factor (less likely to hand out a key)
  - Relatively low cost
  - Good level of security
- Disadvantages:
  - Hard to deploy
  - Can be lost or stolen
- Countermeasures:
  - Use with second factor

3.3.1 Secure Password Exchange and Storage

How to minimize the risk that secrets get stolen?

- Use mutual authentication if possible
- Use a challenge-response scheme (Server sends a challenge (random data) to the client. The client concatenates the challenge with the secret and other random data, and hashes the resulting string, which is sent back to the server along with the other random data in clear. The server has all the "ingredients" to perform the same operation and verify that the 2 hashes match)
- Use random data to avoid replay attacks
- Never store passwords in clear
- Never disclose secrets in password-recovery schemes

3.4 "To be" factor

- Advantages:
  - High level of security
  - Requires no extra hardware to carry around
- Disadvantages:
  - Hard to deploy
  - Non-deterministic matching
  - Invasive measurement
  - Can be cloned
  - Bio-characteristics change
  - Privacy sensitivity
  - Users with disabilities
- Countermeasures
  - Re-measure often
  - Secure the process

4 Access control

Can be divided in

- Discretionary Access Control (DAC)

- Mandatory Access Control (MAC)
- Role-Based Access Control (RBAC)

Difference between DAC and MAC is who assigns privileges, RBAC is sort of a "hybrid"

4.1 Discretionary Access Control

Resource owner discretionarily decides its access privileges (if User1 creates a file, User1 can assign to User2 the privilege of reading it). All the OSs mostly used are DAC (e.g. MacOS, Linux, Windows)

4.2 Mandatory Access Control

Basic idea: do not let owners assign privileges, which are set by a security administrator

4.2.1 Bell-LaPadula model

1. Rule 1: no read up (a subject s at a given secrecy level cannot read an object o at a higher secrecy level)
2. Rule 2: no write down (a subject s at a given secrecy level cannot write an object o at a lower secrecy level)

5 Buffer overflow

It happens when a function `foo`() allocates a buffer e.g. `char buf`[8] and `buf`is filled without size checking. Many typical C function does this: `strcopy`, `strcat`, `fgets`, `gets`, `sprintf`, `scanf`

5.1 Registers

- EBP: base pointer register
- ESP: stack pointer register (points to the top of the stack)
- EIP: istruction pointer register

5.2 Calling convenction

- Before being executed a function `foo`(), the CPU need to save the current EIP so it is saved on the stack. This operation is perfomed with the the `call` istruction
- After the `jmp` istruction the function `foo`() (function prologue):
  - save the current stack base address onto the stack: `push %ebp`

  - the new base of the stack is the old top of the stack sub: `mov   %esp, %ebp`
  - allocate 0x4 bytes (32 bits integer) for foo()'s local variables: `sub   0x4, %esp`

- At the end of the execution of the function `foo()` (function epilogue), the function:
  - current base is the new top of the stack: `mov   %ebp, %esp`
  - restore the saved EBP to registry: `pop   %ebp`
  - pop the saved EIP and jump there: `ret`

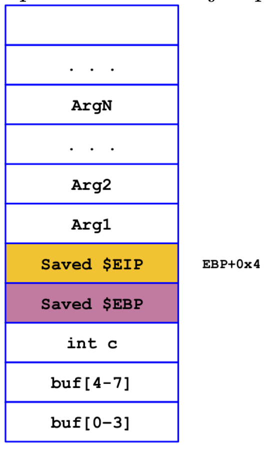
Lets suppose to have this code:

```
int foo(int a, int b){
        int c = 14;
        char buf[8];
        gets(buf);
        //security bug -> vulnerability
        c = (a + b) * c;
        return c;
}
```

What is happened if a user insert a string longer then 8 char?

5.3 Stack smashing

If an user insert a string 20 char long, he overvrite the saved EIP and can manipulate where to jump after the `ret` instruction



5.4 Defending against Buffer Overflow

- Defenses at source code level: finding and removing the vulnerabilities
- Defenses at compiler level: making vulnerabilities non exploitable

- Defenses at operating system level: to thwart, or at very least make more difficult

5.4.1 Defenses at Source Code Level

- Programmer errors cause buffer overflows. Solution is education of developers
- Using safe(r) libraries: Standard Library: `strncpy`, `strncat`, etc. (with length parameter)

5.4.2 Compiler Level Defenses

- Randomized reordering of stack variables
- Embedding stack protection mechanisms at compile time
  - Verifying, during the epilogue, that the frame has not been tampered with
  - Usually a canary is inserted between local variables and control values (saved EIP/EBP)
  - When the function returns, the canary is checked and if tampering is detected the program is killed

5.4.3 Canaries

- Terminator canaries: made with terminator characters (typically \0) which cannot be copied by string-copy functions and therefore cannot be overwritten
- Random canaries: random sequence of bytes, chosen when the program is run
- Random XOR canaries: same as above, but canaries XORed with part of the structure that we want to protect

5.4.4 OS Level Defenses

- Non-executable stack (Return-to-libc or ROP attack still usable): this protection requires the OS and the hardware to cooperate: the NX (nonexecutable) bit should be supported by the processor, and the OS needs to set this bit for the pages which are used as stack
- Address layout randomization (ASLR)

6 Format string

6.1 Reading the stack

Consider this snippet of code:

```
int main (int argc, char* argv[]) {
        printf(argv[1]);
        return 0;
}
```

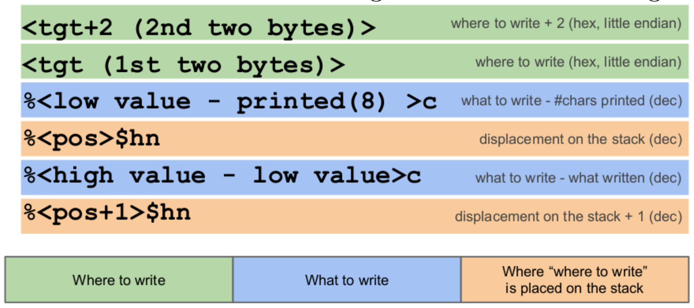
If an user inser as `argv`[1] one or more placeholders (e.g. %`x`), he can read some stack areas above (higher addresses) the areas allocated for the funcion `printf`. If an user put something with a known ASCII coding (e.g. "AAAA") following by a certain number of %`x` in his format string, the user will be able to reading the string itself (e.g. he will see "41414141" somewhere)

6.2 Writing the stack

With placeholder %`n` an user can write, in the address pointed to by the argument, the number of chars printed so far (e.g. `printf`("`hello`%`n`",&`i`); `i` becomes equal to 5). So an user can control the flow execution by:

1. Putting, on the stack, the address (addr) of the memory cell (target) to modify
2. Use %x to go find it on the stack
3. Use %n to write an arbitrary number in the cell pointed to by addr, which is target

so the structure of the format string should be the following:



<tgt+2 (2nd two bytes)>
where to write + 2 (hex, little endian)
<tgt (1st two bytes)>
where to write (hex, little endian)
%<low value - printed(8) >c
what to write - #chars printed (dec)
%<pos>$hn
displacement on the stack (dec)
%<high value - low value>c
what to write - what written (dec)
%<pos+1>$hn
displacement on the stack + 1 (dec)
Where to write
What to write
Where "where to write" is placed on the stack

7 Web app security

The golden rule of web application security is that the client is never trustworthy, so we need to filter and check carefully anything that is sent to us

7.1 XSS (cross-site scripting)

Cross site scripting (XSS) is a vulnerability by means of which client-side code can be injected in a page. For example if in a blog site an user insert a comment like this:

```
<SCRIPT>
        alert('JavaScript Executed');
</SCRIPT>
```

without adeguate filtering policy, popup would appear on next visitors' screen

7.1.1 Stored XSS (AKA Persistent)

The attacker input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc

7.1.2 Reflected XSS (AKA Non-Persistent)

User input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request without permanently storing the user provided data

7.1.3 DOM Based XSS

The user provided data never even leave the browser so the attack payload is executed as a result of modifying the DOM "environment" in the victim's browser

7.1.4 Escaping policy

Blacklisting or whitelisting are not the right policies for handle this vulnerability, instead if we adopt a filtering policy. for instance we can swap > with its HTML safe equivalent `&gt;` and < with `&lt;`

7.2 SQL injection

Consider this function:

```
public void onLogon(Field txtUser, Field txtPassword) {
        SqlCommand cmd = new SqlCommand(String.Format(
                ``SELECT *
                FROM Users
                WHERE username='{0}' AND password='{1}';``,
                txtUser.Text, txtPassword.Text));
        SqlDataReader reader = cmd.ExecuteReader();
if(reader.HasRows())
        IssueAuthenticationTicket();
else
          RedirectToErrorPage();
}
```

it is clearly vulnerable to SQL injection because doesn't filter the user input before exectue the query. In fact if an user insert as username something like "`someUser';--`" the query wold be:

```
SELECT *
FROM Users
WHERE username='someUser';-- ' AND password='';
```

and all is after the two dashes is commented out, so an attacker can log in as "`someUser`" without knowing his password. The solution of this vulnerability is to filter the user input before execute the SQL query

7.2.1 Blind SQL injection

- Some SQL queries, such as the login query we saw, do not display returned values
- Rather, they do, or do not do, stuff based on the return value
- We cannot use them to directly display data, but we can play with their behavior to infer data

7.3 CSRS (cross-site request forgery)

Forces an user to execute unwanted actions (state-changing action) on a web application in which he is currently authenticated (e.g., with cookies). To mitigate this attack is to use a random session token:

- Associated to user's session (unique)
- Regenerated at each request (e.g., for form involving sensitive operations)
- Sent to the server and then compared against the stored token
- Server-side operation allowed only if it matches

7.4 Other vulnerabilities

7.4.1 URL parameter tampering

When in the URL there is something like "`someId = 1297`" by changing this id it is possible to access to data that the user 1297 is not allow to see

7.4.2 Path traversal

If the name of a specifc file given by the user is not adequately filtered, an attacker can insert a series of ../ (this attack is also called "dot dot slash") in order to reach the root and access whatever he want (e.g. ../../../../../../`etc/passwd`)

8 Network protocol attack

8.1 Taxonomy

- Denial of Service (against availability): service unavailable to legitimate users
- Sniffing (against confidentiality): abusive reading of network packets
- Spoofing (against integrity and authenticity): forging network packets

8.2 Killer packets

8.2.1 PoD (ping of death)

Pathological ICMP echo request that exploit a memory error (buffer overflow) in the protocol implementation. Machines can be crashed by sending IP packets that exceed the maximum legal length (65535 octets). This vulnerability can be solved by upgrading operating system to a version that is not vulnerable or by using a firewall that drops the package with a payload too large

8.2.2 Teardrop

Exploit vulnerabilities in the TCP reassembly. Fragmented packets with overlapping offsets. While reassembling, kernel can hang/crash

8.2.3 Land Attack

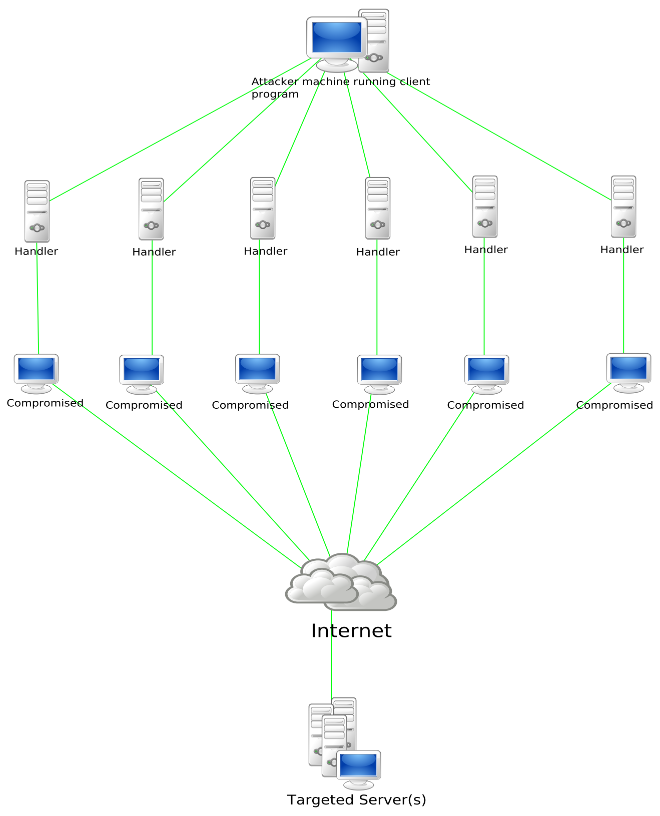
A long time ago, in Windows 95, a packet with:

- src IP == dst IP
- SYN flag set

could loop and lock up a TCP/IP stack

8.3 DDoS (distributed denial-of-service)

A distributed denial-of-service (DDoS) is a large-scale DoS attack where the perpetrator uses more than one unique IP address, often thousands of them. Since the incoming traffic flooding the victim originates from many different sources, it is impossible to stop the attack simply by using ingress filtering. It also makes it very difficult to distinguish legitimate user traffic from attack traffic when spread across so many points of origin

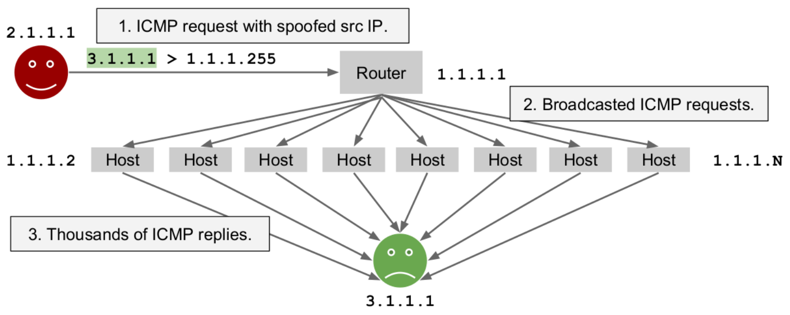


8.3.1 SYN flood attack

Attacker generates a high volume of SYN requests with spoofed source address. Many half-open TCP/IP connections fill the queue. SYN requests from legitimate clients dropped. SYN-cookies avoid this attack

8.3.2 Smurf

The Smurf attack is an attack in which large numbers of Internet Control Message Protocol (ICMP) packets (ping) with the intended victim's spoofed source IP are broadcast to a computer network using an IP broadcast address. Most devices on a network will, by default, respond to this by sending a reply to the source IP address. If the number of machines on the network that receive and respond to these packets is very large, the victim's computer will be flooded with traffic


2.1.1.1
1. ICMP request with spoofed src IP.
3.1.1.1 > 1.1.1.255
Router
1.1.1.1
2. Broadcasted ICMP requests.
1.1.1.2
Host
1.1.1.N
3. Thousands of ICMP replies.
3.1.1.1

8.4 Network sniffing

Normally, a network interface card (NIC) intercepts and passes to the OS only the packets directed to that host's IP. In promiscuous mode the NIC passess to the OS any packet read. Mitigation for sniffing can be to use switched networks instead hub-based networks (hubs broadcast traffic to every host, Switches selectively relay traffic to the wire corresponding to the correct NIC)

8.5 ARP spoofing

The ARP basically maps 32-bits IPv4 addresses to 48-bits hardware or MAC addresses. The main vulnerability about ARP are:

- First come, first trusted so an attacker can forge replies easily: lack of authentication
- Each host caches the replies
- Switches use CAM tables to know (i.e., cache) which MAC addresses are on which ports. It is possible using some tools to fills the CAM table in seconds by generating a lot of spoofed packets. When CAM table is full switches cannot cache ARP replies and must forward everything to every port (like a hub does)

8.6 IP spoofing

The IP source address is not authenticated. Changing it in UDP or ICMP packets is easy. However, the attacker will not see the answers, because they will be sent to the spoofed host (blind spoofing). But if the attacker is on the same network, he can sniff the rest, or use ARP spoofing

8.7 TCP session hijacking

- TCP uses sequence numbers for reordering and acknowledging packets
- A semi-random initial SEQuence number (ISN) is chosen

- If a blind spoofer can predict the ISN, he can blindly complete the 3-way handshake without seeing the answers
- Else if the attacker (C) can sniff the packets of an already active TCP session:
  - C follows the conversation of A and B recording the sequence numbers
  - C somehow disrupts B's connection (e.g. SYN Flood): B sees only a "random" disruption of service
  - C takes over the dialogue with A by spoofing B address and starting with a correct ISN.

8.8 MITM (man in the middle)

An attack where an attacker can impersonate the server with respect to the client and vice-versa:

- physical or logical MITM
- full or half-duplex (blind)

8.9 DNS poisoning

DNS is on UDP transport, sothe messages are not authenticated. When a non-authoritative DNS server receives a request to resolve a domain name:

- if it cached the answer, it answers
- If no answer in cache:
  - Recursion: resolves the name on behalf of the client
  - Iterative: gives the authoritative DNS address

If an attacker makes a recursive query to the victim DNS server:

- The victim DNS serverwill contact the authoritative server
- The attacker spoofs the answer impersonating the authoritative DNS server and the server will trust it

8.10 DHCP poisoning

DHCP is not authenticated. The attacker can intercept requests, be the first to answer, and client will believe that answer. With a single (spoofed) DHCP response, the attacker can set IP address, DNS addresses, default gateway of the victim client

8.11 ICMP redirect

Tells an host that a better route exists for a given destination, and gives the gateway for that route. The attacker can forge an ICMP redirect packet to elect his computer as the gateway. The attacker needs to intercept a packet in the "original" connection in order to forge the reply

9 Firewalls

A firewall is a network access control system that verifies all the packets flowing through it. Its main functions are usually:

- IP packet filtering
- Network address translation (NAT)

Firewalls can be devided depending on their packet inspection capability:

- Network layer firewalls
  - Packet filters (network)
  - Stateful packet filters (network-transport)
- Application layer firewalls
  - Circuit level firewalls (transport-application)
  - Application proxies (application)

9.1 Packet filters

Regardless of the specific syntax, every network packet filter allows to express the following concept: if (packet matches certain condition) do this e.g.:

- Block
- Allow
- Log
- Other actions

Packet filters are stateless: cannot track TCP connections

9.2 Stateful (or dynamic) packet filters

Include network packet filters, plus:

- Keep track of the TCP state machine
- We can track connections without adding a response rule

Session-handling is fundamental for NAT

9.3 Circuit firewalls (legacy)

Client connects to a specific TCP port on the firewall, which then connects to the address and port of the desired server (not transparent). Essentially, a TCP-level proxy

9.4 Application proxies

Same as circuit firewalls, but at application layer. Inspect, validate, manipulate protocol application data (e.g., rewrite HTTP frames). Almost never transparent to clients

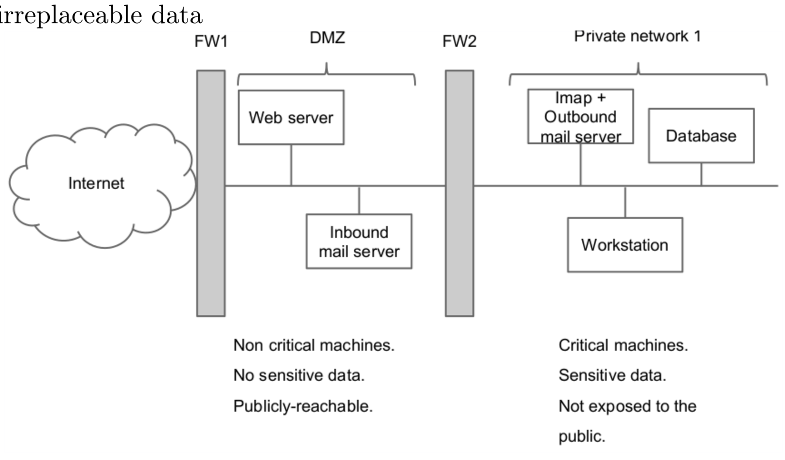
10 Secure network architecture

10.1 Dual- or Multi-zone architectures

In most cases, the perimeter defense works on the assumption that what is "good" is inside, and what's outside should be kept outside if possible. There are two counterexamples:

- Access to resources from remote (i.e., to a web server, to FTP, mail transfer)
- Access from remote users to the corporate network

Problem: if we mix externally accessible servers with internal clients, we lower the security of the internal network. Solution: we allow external access to the accessible servers, but not to the internal network. In practice, we create a semi-public zone called DMZ (demilitarized zone). The DMZ will host public servers (web, FTP, public DNS server, intake SMTP). On the DMZ no critical or irreplaceable data

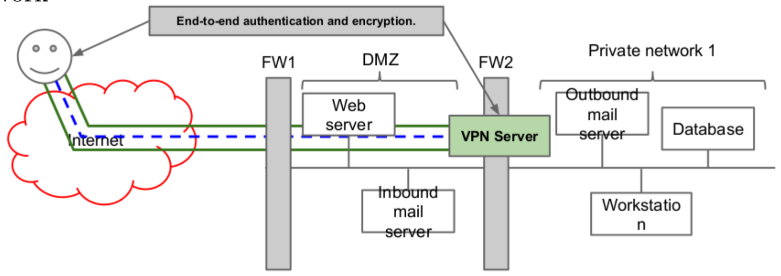


10.2 VPN (virtual private network)

If a user need to:

- Work "as if" they were in the office, accessing resources on the private zone
- Connect remote sites without using dedicated lines
- Ensure CIA (confidentiality, integrity, availability) to data transmitted over a public network (i.e., the Internet)

The solution is to use a VPN, an encrypted overlay connection over a (public) network



- Full tunnelling:
  - Every packet goes through the tunnel
  - Traffic multiplication, could be inefficient
  - Single point of control and application of all security policies as if the client were in the corporate network
- Split tunnelling
  - Traffic to the corporate network: in VPN
  - traffic to the Internet: directly to ISP
  - More efficient, less control
  - Just similar to the case of the PC connected via 3G modem to the Internet

11 Network Security

- Problems of remoteness
  - Trust factor between parties
  - Use of sensitive data
  - Atomicity of transaction
- Internet protocol problems

  - Authentication
  - Confidentiality
  - Transparence and critical mass problem

Two possible solutions:

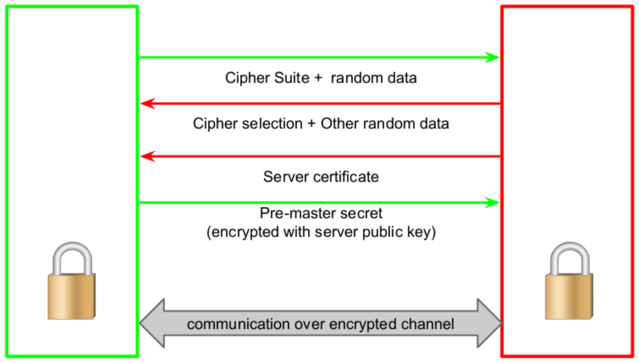
- HTTP over SSL (Secure Socket Layer), or HTTPS
  - Communication confidentiality and integrity
  - Mutual authentication
  - No guarantees on data usage
  - No strict authentication of client (in practice)
- SET (Secure Electronic Transaction)
  - Guarantees on data usage and transaction security enforcement
  - Missing critical mass support

11.1 SSL

SSL enforces:

- Confidentiality and integrity of the communications
- Server authentication
- Client authentication (optionally)

It uses both symmetric and asymmetric cryptography for performance reasons

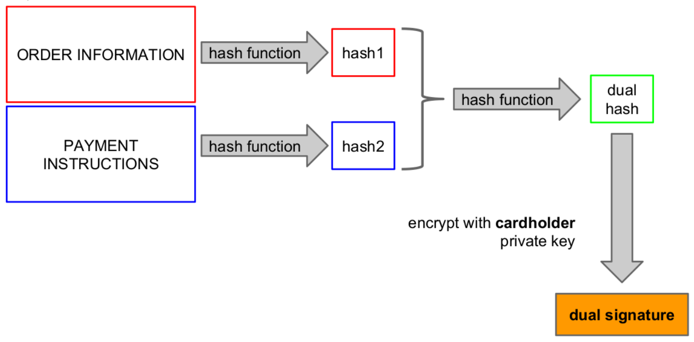

Cipher Suite + random data
Cipher selection + Other random data
Server certificate
Pre-master secret
(encrypted with server public key)
communication over encrypted channel

- PROs
  - Protects transmissions
  - Ensures authentication
  - design to be resistant to MITM

- CONs
  - No protection before or after transmission
  - Relies on PKI
  - Not foolproof

11.2 SET

It guarantees that the minimal amount of information about a commercial transaction is disclosed to the parties: the merchant is only informed about the successful payment (without knowing any payment details, e.g., credit card number), whereas the payment gateway is only informed about the payment details (without knowing the purchased goods). It is created by hashing the payment information (PI) and order information (OI) individually, concatenate the hashes so obtained, hash the result and encrypt it with the buyer's private key. The resulting encrypted hash is the dual signature, which is concatenated with the OI (resp. PI) and the hash of the PI (resp. OI), and encrypted with the public key of the merchant (resp. payment gateway). Unfortunately, the dual signature as specified in SET cannot be used in the real world because it requires certificate-based authentication of all the involved parties. While this is generally not a problem for merchants and payment gateway, it is definitely a problem for the buyers, thus not creating a critical mass to allow SET to be deployed.



A possible intermediate solution based on HTTPS to implement the "equivalent" of the dual signature can be: instead of processing the payment directly by collecting the payment information from the buyer, the merchant creates a unique token for the transaction (which substitutes the hash of the OI) and redirects the buyer to the payment gateway, which processes the payment and releases a response token based on the unique token above (which substitutes the hash of the PI). Then, the buyer goes back to the merchant, which can verify that the hash of the PI is related to the hash of the OI

12 Malware

Malware is code that is intentionally written to violate a security policy. Three main categories of malware:

- Viruses: self-propagate by infecting other files, usually executables (but also documents with macros, boot loader code). They are not programs (i.e., not executables)
- Worms: programs that self-propagate, even remotely, often by exploiting host vulnerabilities, or by social engineering (e.g., mail worms)
- Trojan horses: apparently benign program that hide a malicious functionality and allow remote control

12.1 Botnets

A botnet is a network that consists of several malicious bots that are controlled by a commander, commonly known as botmaster (botherder). The main threats caused by the botnets are:

- For the infected host:
  - information harvesting
  - Identity data Financial data
  - Private data E-mail address books
  - Any other type of data that may be present on the host of the victim
- For the rest of the Internet
  - Spamming
  - DDoS
  - Propagation (network or email worm)
  - Support infrastructure for illegal internet activity (the botnet itself, phishing sites, drive-by-download sites)

12.2 Antivirus and anti-malware

- Basic strategy is signature-based detection: database of byte-level or instruction-level signatures that match malware
- Implemented through
  - Heuristics algorithm (check for signs of infection) e.g.:
    - Code execution starts in last section
    - Incorrect header size in PE header
    - Suspicious code section name

        - Patched import address table
    - Behavioral Detection:
        - Detect signs (behavior) of known malware
        - Dtect “common behaviors” of malware

12.2.1 Virus and Worm Stealth Techniques

- Entry Point Obfuscation
    - Multicavity viruses
    - Virus hijacks control later (after program is launched)
        - Overwrite import table addresses (e.g., libraries)
        - Overwrite function call instructions
- Polymorphism
    - Change layout (shape) with each infection
    - Payload is encrypted (~ packing)
        - Using different key for each infection
        - Makes static string analysis practically impossible
        - Of course, AV could detect encryption routine
- Metamorphism
    - create different “versions” of code that look different but have the same semantics (i.e., do the same)
- Dormant period
    - During which no malicious behavior is exhibited
- Event-triggered payload
- Encryption / Packing
    - Similar to polymorphism but more advanced techniques are available in more complex malware
- Rootkit techniques
    - Anti-virtualization techniques

12.2.2 Rootkit

A rootkit is a malware that infects the machine at user or kernel level. The goal of a rootkit is to give the attacker persistence access to the machine and, optionally, hide the attacker's actions (including the presence of the rootkit itself) by modifying how the (operating) system behaves (e.g., by hiding processes or files). Rootkits are hard to detect, especially if they are planted in the operating system kernel, because they alter its behavior to make detection difficult or impossible. Usually, a reasonable first approach is to analyze the computer's disk on a different computer, and try to spot any differences (e.g. any files or processes that show up on the other computer and not on the supposedly infected one). This is called cross-layer examination.