Computer Security

A series of notes on the "Computer Security" course as taught by Stefano Zanero during the second semester of the academic year 2018-2019 at Politecnico di Milano.

Ass-Savers

- ***Buffer-overflow***
  - Ogni volta che pushi nella stack vai a occupare indirizzi più bassi (zone alte nel disegno)
  - Quando hai una struct pushi i datatype dal basso verso l'alto, e.g. se ho una struct che dichiara un *char*, un *int* e un *array* $\to$ pusho prima l'array, poi l'int e alla fine il $char$
  - All'interno di una funzione pusho le variabile in ordine di "spazio occupato", e.g. se all'interno di una funzione dichiaro prima un int (4 byte), poi un $char$ (1 byte), un array di 5 $char$ (5 byte) e un array di 2 int (8 byte) $\to$ pusho prima il $char$, poi l'int, poi l'array di $char$ e poi l'array di int
- ***Firewalls***

  Quando penso a una regola mi devo chiedere sempre : "Chi vuole iniziare la comunicazione??" chiamiamolo X

  la regola sarà nella direzione $X\to${destinatario}

Definitions

- ***ASLR***
  it's a security mechanism used for avoiding exploitation of Buffer Overflow vulnerabilities.
  It means Address Space Layout Randomization.
  It consists in repositioning the stack, among other things, at each execution at random; impossible to guess return addresses correctly
- ***Canary***
  it's security mechanism used for avoiding exploitation of Buffer Overflow vulnerabilities.
  It works by placing a small integer, the value of which is randomly chosen at program start, in memory just before the stack return pointer.
- ***Cross-Site-Scripting***
  - ***Reflected***

    User input is directly returned by the web application in a response (e.g. error message,search result) which includes some or all of the input provided by the user in the request, *without being stored* and made safe to render in the web browser.

    ```
    <?php
    $var = $HTTP['varible_name']; // retrieve content from request
    echo $var //print variable in the response
    ?>

    malicious crafted URL : http://example.com/?variable_name=
    <script>alert('XSS')</script>
    ```

  - ***Stored***

The attacker(malicious) input is stored on the target server in a database (e.g. comment field) and then is retrieved by a victim from the web application.

- ***DOM based***

  User provided input never leaves the victim's browser: malicious payload is executed by client side script to modify/update the DOM "environment" (e.g. dynamic pages, forms)

```
...
<script>
    document.write("<b>Current URL</b> : " + document.baseURI);
</script>
...
malicious crafted URL : http://example.com/test.html#
<script>alert('XSS')</script>
```

- ***Cross-Site Request Forgery (CSRF)*** $5/2/2018$

  Forces an user to execute a wanted action (state-changing action) on a web application in which it is authenticated (e.g. with cookies)

  Key concept: malicious requests (e.g. crafted links) are routed to the vulnerable web application through the victim's browser: web sites can not distinguish if the requests coming from authenticated users have been originated by an explicit user interaction or not.

  `To solve this problem, include a CSRF token with every legitimate request, and check that cookie['csrftoken'] == param['csrftoken'].The CSRF stems from the fact that the web application authenticates the user only with "ambient credentials" (cookies, which are sent automatically with every request) also to perform a state-changing action (buy a book in this case); this way, whatsonbook is not able to authenticate whether the request was voluntarily initiated by the user or not. A solution is to require for every state-changing action a further secret token (e.g., automatically inserted as a hidden field in the bank's form, ...) and checking its validity before performing the operation. This way, to buy a book, it is necessary to read the content of a request to whatsonbook's servers, and not just to blindly perform a request. As the same origin policy does not allow a website to read the response of an arbitrary request to a different origin, the attackers would not be able to read the secret CSRF`

  if you still have doubts check here

- ***What is a Rainbow Table?***
  It is a precomputed table for reversing hash functions, usually for passwords that are stored hashed.
- ***What is a salt?***
  a salt is random data that is used as an additional input to a one-way function that "hashes" data, a password or passphrase. Salts are used to safeguard passwords in storage (it prevents the usage of rainbow tables).
- ***What is a prepared statement?***

  A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

  Basically it's just a template.

  Prepared statements basically work like this:

    - Prepare : An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
    - The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
    - Execute : At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values
- ***ARP spoofing / poisoning***

  https://www.ilsoftware.it/articoli.asp?tag=ARP-cos-e-e-cosa-sono-gli-attacchi-poisoning_18690
- ***Rootkit***

  A rootkit is a program or, more often, a collection of software tools that gives a threat actor remote access to and control over a computer or other system.

  Different types:

    - User-Land:

      Easer to build, but often incomplete

      Easier to detect (cross layer examination, use of non-trojaned tools)
    - Kernel-Space:

      More difficult to build, but can hide artifacts completely.

      Can only be detected via post-mortem analysis.

      You can get rid of it by reinstalling the OS or substating the hard-drive.
    - BIOS

      Devi bruciare il computer.
- ***SQL Injection***

  There must be a data flow from a user-controlled HTTP variable (e.g., parameter, cookie, or other header fields) to a SQL query, without appropriate filtering and validation. If this happens, the SQL structure of the query can be modified.
- ***Blind SQL Injection***

  In a blind injection the data retrieved by the modified SQL query is not displayed back to the attacker. This can still be exploited: changes can be blindly executed in the database, or by using side-effects of queries the attacker can guess the answers.
- ***Virus***

  self-propagate by infecting other files, usually executables (but also documents with macros, boot loader code). They are not programs (i.e., not executables).
- ***Worm***

  programs that self-propagate, even remotely, often by exploiting host vulnerabilities, or by social engineering (e.g., mail worms).
- ***Trojan***

  apparently benign program that hide a malicious functionality and allow remote control
- ***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

  `Encryption is the most common method to hide code. With encryption, the main body of the code (also called its payload) is encrypted and will appear meaningless. For the code to function as before, a decryption function is added to the code. When the code is *executed* this function reads the payload and decrypts it before executing it in turn.`

  `Encryption alone is not polymorphism. To gain polymorphic behavior, the encryptor/decryptor pair are mutated with each copy of the code. This allows different versions of some code which all function the same.`

- ***Metamorphism***

  create different "versions" of code that look different but have the same semantics (i.e., do the same)

- ***What is, in general, the goal of a DoS attack? What are the differences between a DoS attack and a distributed DDoS attack? Is, given enough resources, a DDoS attack always feasible? Why?***

  A DoS attack is an attempt to overload an online service (website) with traffic. The goal is to disrupt the website or network in order to stop legitimate users from accessing the service.

  The DoS attack is usually launched from a single machine, as opposed to a DDoS attack which is launched from multiple machines.

  DDoS is generally always feasible, given enough resources (i.e., the attacker can just rent a botnet for a few hours).

  Per gli attacchi Dos: `UDP->BRUTTO, TCP->BELLO`

  `*Move the protocol to TCP instead of UDP as a transport protocol. Due to the three-way handshake, TCP is immune to the amplification issue.*`

- ***What is an antivirus? Which are the strategies through which it detects malware? State their names and give a brief explanation of how they work.***

  Antivirus software is a program or set of programs that are designed to prevent, search for, detect, and remove software viruses, and other malicious software like worms, trojans, adware, and more.

  Strategies:

    - *Ex-Post* workflow
        - suspicious executable reported by "someone"
        - automatically analyzed
        - manually analyzed
        - antivirus signature developed
    - *Static Analysis*
        - Parse the executable code
        - PROs and CONs
            - +Code coverage, dormant code
            - -Obfuscation(metamorphism, encryption,packing)
    - *Dynamic Analysis*
        - Observe the runtime behavior of the executable
        - PROs and CONs

    - -Code coverage, dormant code
    - +Obfuscation(metamorphism, encryption,packing)
- ***What are the techniques used by malware to evade detection? State their names and give a brief explanation of how they work.***

  Metamorphism and Polymorphism.

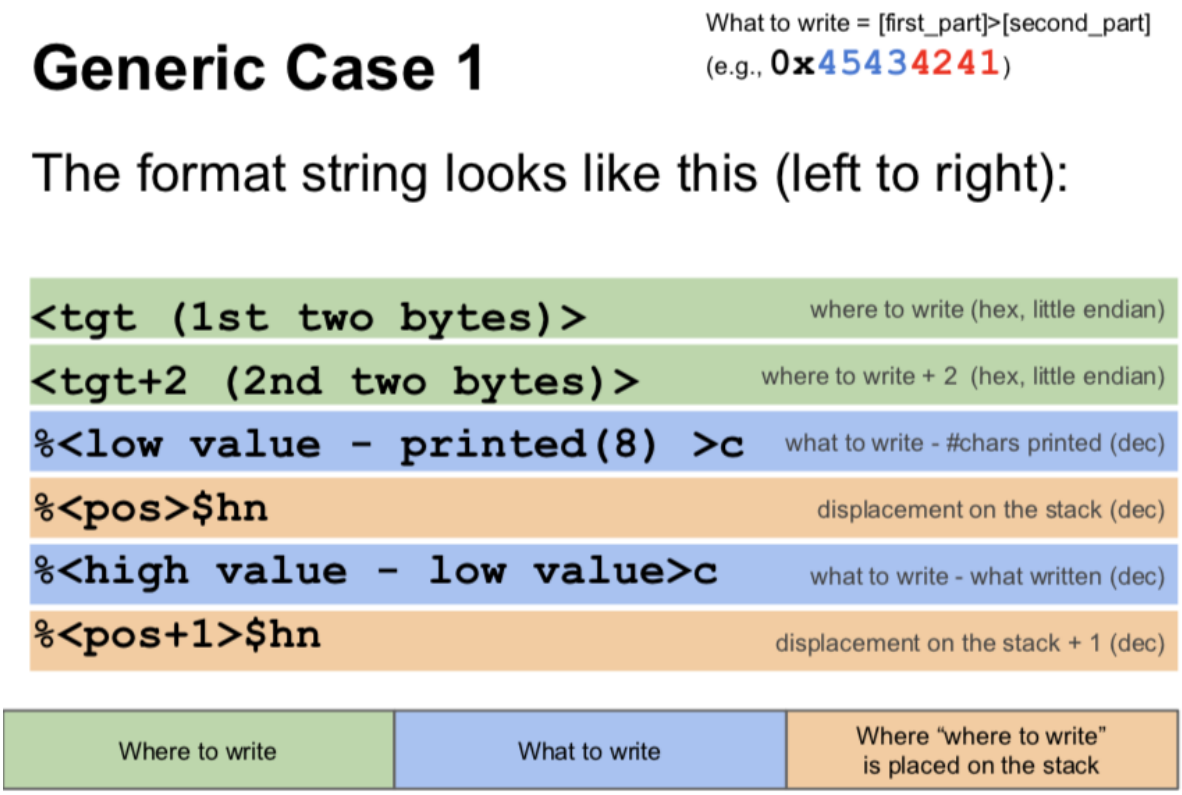
- ***Consider mobile malware developed for the Android platform. Can an antivirus for Android be as effective as an antivirus developed for a legacy operating system (e.g., Windows, or Linux) in the task of detecting a malware already running in the system? Why?***

  No, because Mobile security model (single user, multi app) is different from traditional security models (multi user). In the Mobile security model AVs are apps. Apps cannot interfere with each other.

  How can AVs check for malicious apps if they cannot access the entire system?

    - run check upon installation
    - list of package names of malicious apps
    - list of MD5s of malicious apps
    - limit scan to the SD card (world readable)

Format strings



AAA! A QUANTO PARE SE `FIRST_PART = SECOND_PART` CI TROVIAMO NEL `GENERIC CASE 2` (TDE 01/07/19)

Generic Case 2

What to write = [first_part]<[second_part]
(e.g., 0x42414543)
SWAP Required

The format string looks like this (left to right):

`<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

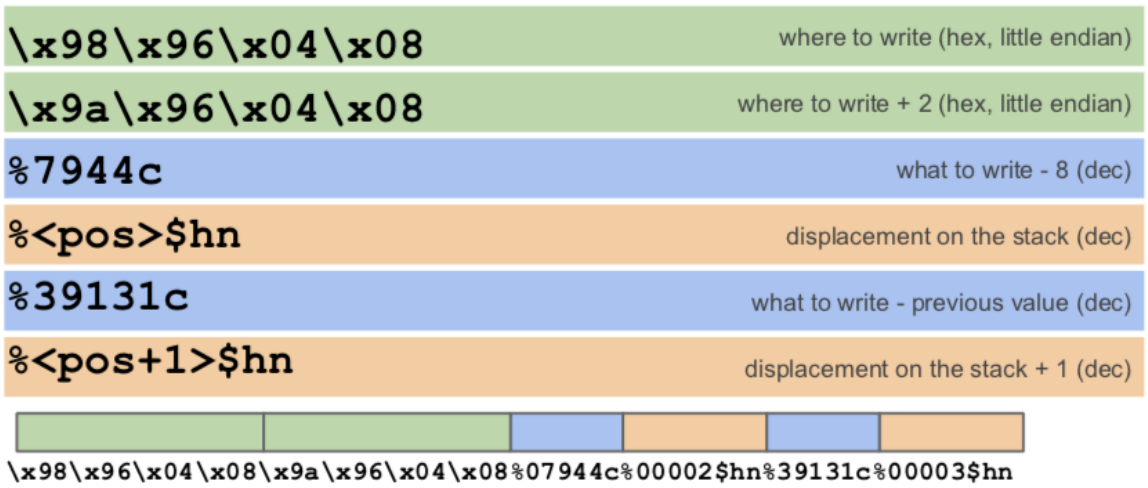
Example:

Let's write `0xb7eb1f10` to `0x08049698`

`0xb7eb = 47083 > 7952 = 0x1f10 ~> 7952 must be written 1st`

`\x98\x96\x04\x08`	where to write (hex, little endian)
`\x9a\x96\x04\x08`	where to write + 2 (hex, little endian)
`%(7952-8)c`	what to write - 8 (dec)
`%<pos>$hn`	displacement on the stack (dec)
`%(47083-7952)c`	what to write - previous value (dec)
`%<pos+1>$hn`	displacement on the stack + 1 (dec)

Where to write	What to write	Where "where to write" is placed on the stack



HowToCrackEx3

- ***Kind of attacks:***
  - $\color{red}{SMURF\ attack}$ `(DDOS)`:

    An attacker spoof an IP address (this can be done through HTTP, not HTTPS).

    The attacker send an ICMP request with the spoofed IP address, then the router broadcast the request and any host present in the domain generates an ICMP reply with the victim's address as destination. $\to$ This results in a *distributed denial of service (DDOS)*

    `an ICMP request is a protocol able to notify errors and failures without executing any correction`

    How to prevent this shit?

    - Configure individual hosts and routers to not respond to ICMP requests or broadcasts
    - Configure routers to not forward packets directed to broadcast addresses.
  - $\color{red}{Ping\ of\ Death}$ `(DOS)`:

    A ping of death is a type of attack on a computer system that involves sending a malformed ping to a computer.

    The maximum size of an IP packet may be as large as 65,535 bytes. Like other large but well-formed packets, a ping of death is fragmented into groups of 8 octets before transmission. However, when the target computer reassembles the malformed packet, a buffer overflow can occur, causing a system crash and potentially allowing the injection of malicious code.

    How to prevent this shit?

    - In general can be mitigated adding controls during the process of reassembly.

    - Upgrade the operating system to a non-vulnerable version.
    - Use a firewall that drops such anomalous packets.
- $\color{red}{ARP\ spoofing\ (or\ poisoning)}$ `(e.g. Man in the Middle)`

  `The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address`

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

  ARP request :

  $\text{"where is 192.168.0.1 ?"}$

  ARP reply :

  $\text{"192.168.0.1 is at b4:e9:b0:c9:81:03"}$

  First come, first trusted !

  An attacker can forge replies easily in UDP or ICMP packets...TCP/IP BOH

  How to prevent this shit?

    - Application level : HTTPS (trusted certificate)
    - Network level : use 802.1x (authenticates clients connected)

  How to detect ARP spoofing?

    - Router level : Notice multiple ARP responses with different MAC addresses (to prevent this attack it is possible to block responses with the IP of the sensible target (e.g. the router) but a different MAC address w.r.t. the real ones)
    - Gateway level : Notice ARP response with IP of the gateway
- $\color{red}{DHCP\ poisoning}$ `(e.g. Man in the Middle, Traffic interception, Traffic redirection)`

  `DHCP (Dynamic Host Configuration Protocol) is a protocol used to provide quick, automatic, and central management for the distribution of IP addresses within a network. DHCP is also used to configure the proper subnet mask, default gateway, and DNS server information on the device.`

  DHCP does not support authentication, so every client *must* blindly believe any DHCP offer that it receives, thus an attacker can race and win against the real DHCP server. In this way the attacker can intercept the request, be the first to answer, and craft a DHCP response setting the *IP address*, the *DNS address* and the *default gateway* of the victim client.

- $\color{red}{TCP/IP}$

  TCP/IP is the main protocol which provides reliable ordered and error-checked delivery of data. It uses sequence numbers for reordering packets, in particular a *semi-random* INITIAL SEQUENCE NUMBER (ISN) is chosen.

  An attacker able to guess the ISN can perform the free-way handshake (SYN $\to$ SYN+ACK $\to$ ACK) without the need to pose as a Man In The Middle.

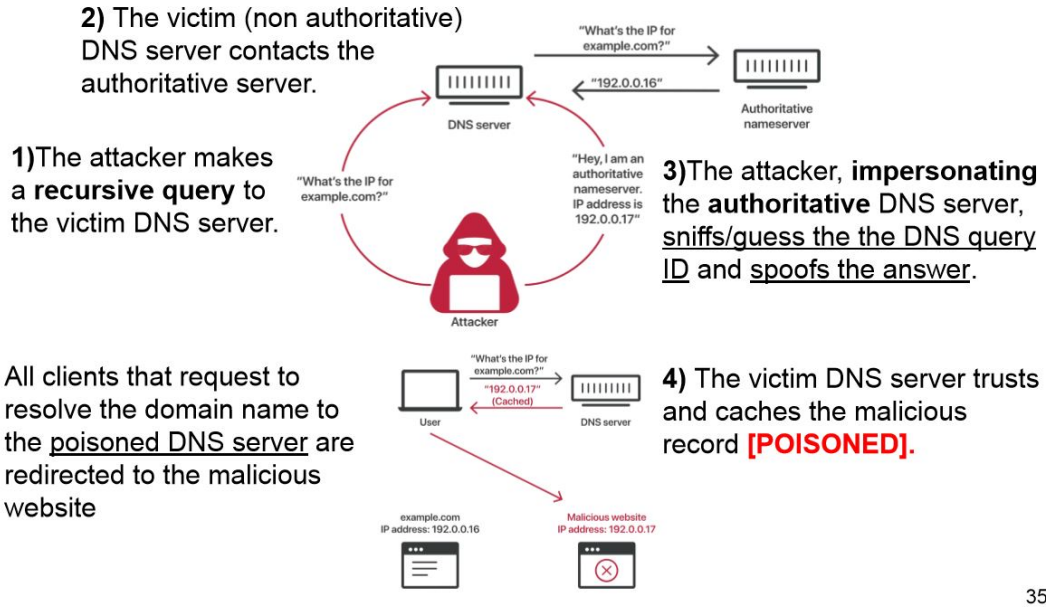
  In order to reduce the time available to the attacker to guess the ISN it is possible to use TCP-syn cookies.

- $\color{red}{DNS\ cache\ poisoning}$

DNS translates domain names to the numerical IP addresses. It is based on UDP and messages are *not authenticated*.

In this kind of attack the IP address is *spoofed* FOR SURE (the attacker want to save his ass I think) ,the MAC address *could* be spoofed.

The purpose of this attack is to divert any traffic directed to an URL to another server, likely an attacker-controlled machine. The actual purpose could be to manipulate the information on the URL or to capture personal information (e.g. credential, cookies credit cards etc...)



35

How to prevent this shit?

- If you are `the administrator of a website e.g www.polimi.it`, use HTTPS (assuming that the attacker is not able to perform DNS spoofing for the network that the certification authority uses to connect to www.polimi.it in order to verify the domain for issuing the certificate. if the attacker can do this, they can obtain a valid certificate for www.polimi.it and mount the same attack)
- If you are `the administrator of the DNS server used by the network e.g. 10.79.3.0/24`

  If we assume that the attacker can't sniff the traffic to\from the DNS server (thus sniffing the query ID): use a random query ID and increase the space of query IDs to prevent guessing the ID in a reasonable time. In general: the server could detect duplicate responses with different A entries and raise an alarm, or detect response with different query IDs and raise an alarm. Notice that if, upon attack detection, the DNS server returns an error, this will transform the attack to a denial of service attack.
- If you are `the network administrator of the network e.g. 10.79.3.0/24`,

  Given that the DNS server is outside the network, the network administrator could reject known spoofed packets (all packets coming from inside the network with an IP address that doesn't belong to the network). This mitigation is effective ONLY for attackers physically present on the network 10.79.3.0/24, and can't do anything for spoofed packets that come from outside the administrator-controlled network (e.g., from the Internet).