



Cyber Security Course

VERSION 2
النسخة الثانية

دورة الأمن السيبراني

Course Instructor : Eng. Mohammed Kher Alkhawaldeh

Twitter : @Storm_0x05

Website : jorshield.com

E-Mail : mkk@jorshield.com

PreSecurity :

Intro About Offensive Security
Intro About Defensive Security

1. Operating Systems

2. Web Application Technologies

3. Networking Basics

4. Programming languages.

5. Cryptography

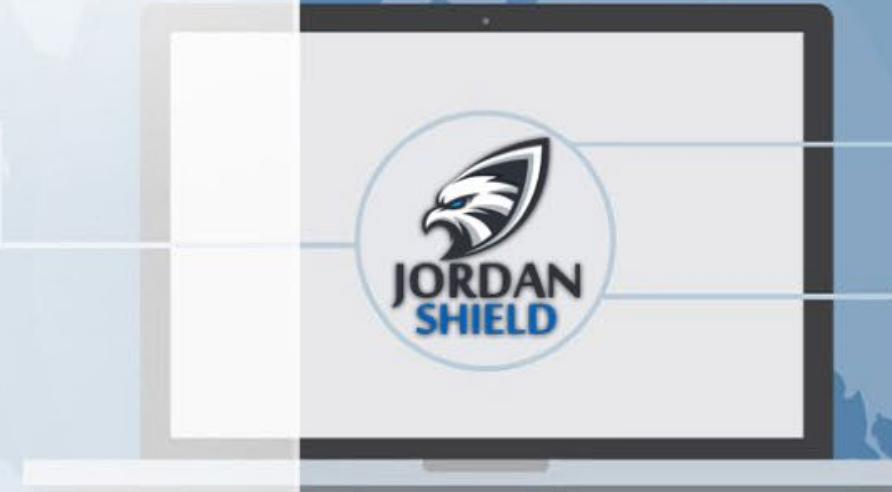
TryHackMe Tracks

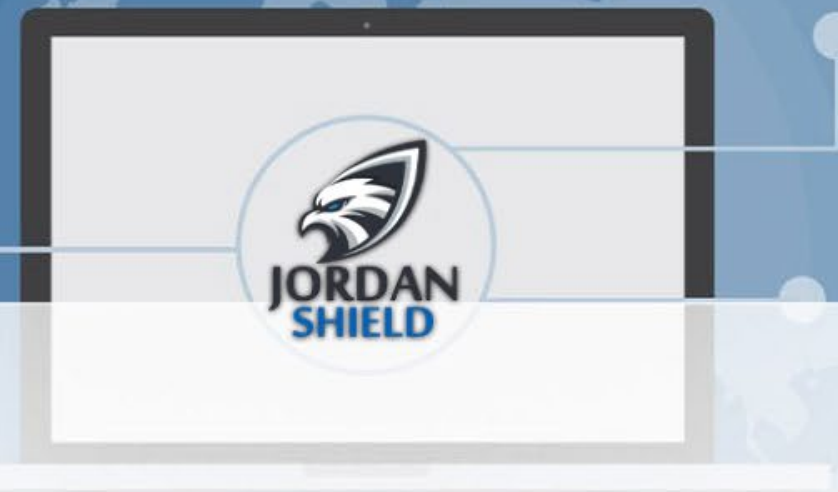
CTF Tracks

HackTheBox Tracks

PortSwigger Tracks

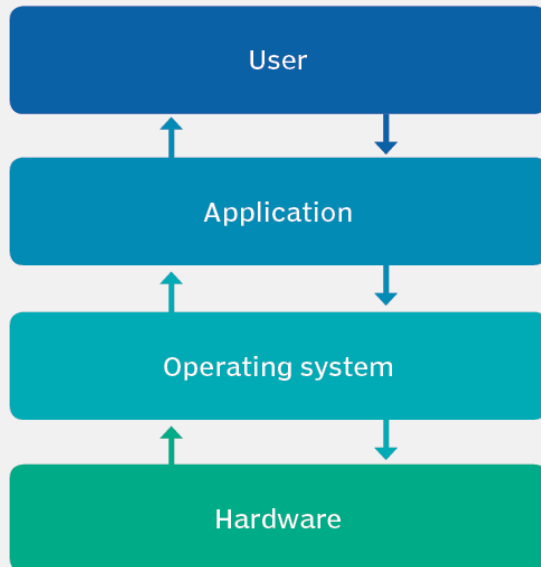
Etc...





Operating System:

Operating system placement



©2019 TECHTARGET. ALL RIGHTS RESERVED

operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all of the other application programs in a computer. The application programs make use of the operating system by making requests for services through a defined application program interface (API). In addition, users can interact directly with the operating system through a user interface, such as a command-line interface (CLI) or a graphical UI (GUI)

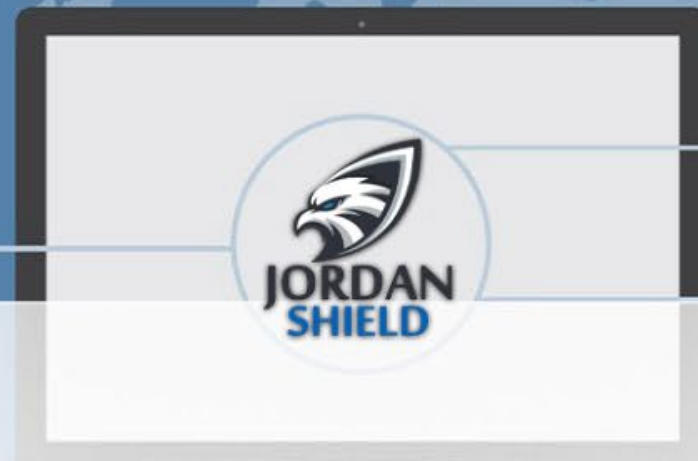


What is Virtual Machine (VM) ?

A virtual machine (VM) is defined as a computer system emulation, where VM software replaces physical computing infrastructure/hardware with software to provide an environment for deploying applications and performing other app-related tasks.

How do virtual machines work?

Virtualization allows for creating a software-based computer with dedicated amounts of memory, storage, and CPU from the host computer. This process is managed by hypervisor software. As needed, the hypervisor moves resources from the host to the guest. It also schedules operations in VMs to avoid conflicts and interference when using resources.



Benefits of using virtual machines

Cost savings

Energy savings

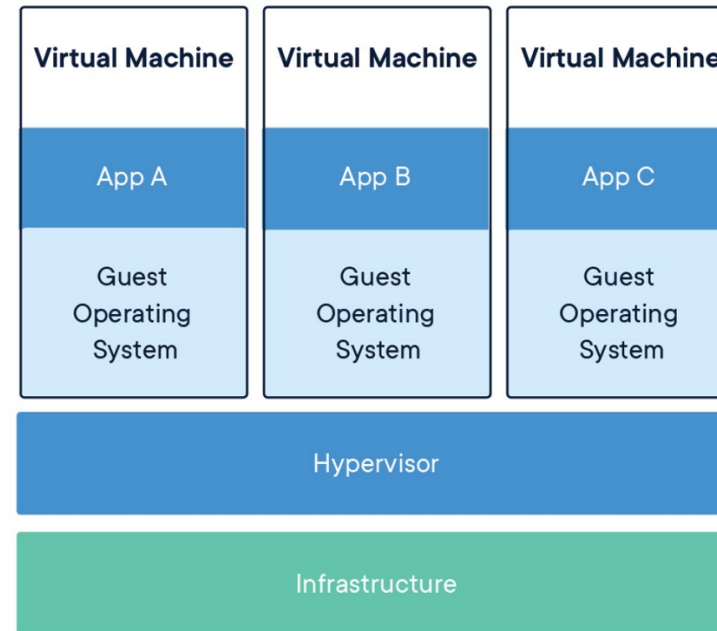
The isolated environment provided by virtual machines

Easy to backup and clone

Save space

Flexibility

Workplace mobility and customization





What is the best OS for security engineers ?

1) Kali Linux

Kali Linux is a Security Distribution of Linux specifically designed for digital forensics and penetration testing. It is one of the best hacking OS which has over 600 preinstalled penetration-testing applications (cyber-attack performs against computer vulnerability). This OS can be run on Windows as well as Mac OS.

2) Parrot OS

Parrot OS is a platform for hacking. It has an easy to use editor for software development. This platform enables you to surf the web privately and securely. Hackers can use Parrot OS to perform vulnerability assessment, penetration testing, computer forensics, and more.





What is the best OS for security engineers ?

3) BackBox

BackBox is an Ubuntu based open-source Operating System that offers a penetration test and security assessment facility. This system also provides a network analysis toolkit for security in the IT environment. It contains a toolkit that is needed for ethical hacking.

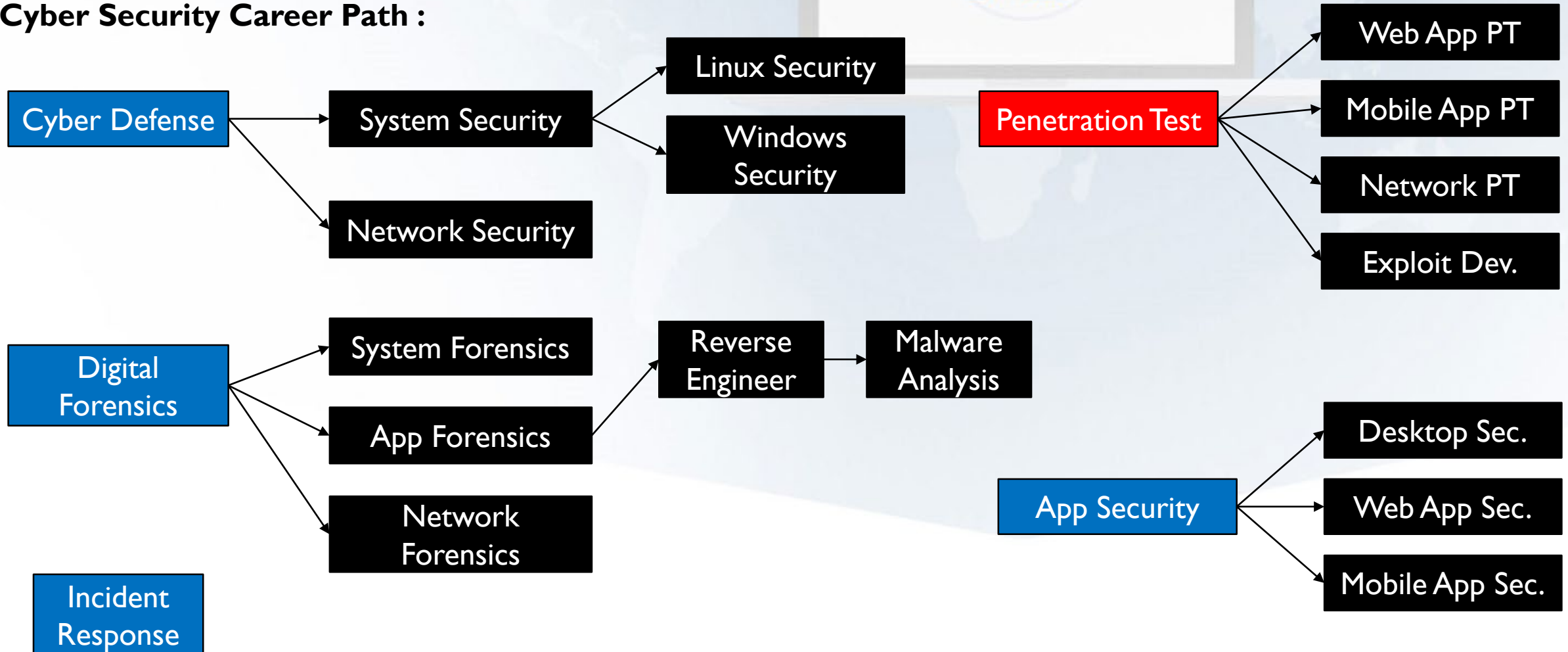
4) BlackArch

BlackArch is a free Linux based platform that contains a wide range of tools for automation, mobile, networking, etc. It is an open-source Operating System that is specially created for security researchers and penetration testers. You can install it individually or in a group.





Cyber Security Career Path :





HTTP Protocol Basics



3.2 HTTP Protocol Basics

- + **How does this support my pentesting career?**
 - The ability to exploit web applications and find vulnerabilities in web servers and services
 - Web applications technology is used market-wide also by desktop or mobile applications



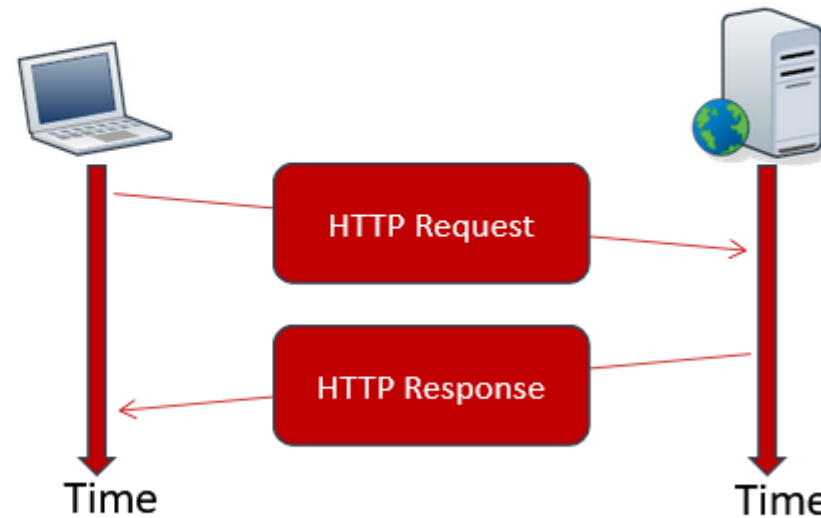
3.2 HTTP Protocol Basics

- + **Hypertext Transfer Protocol (HTTP)** is the most used application protocol on the Internet. It is the client-server protocol used to transfer web pages and web application data.
- + In HTTP, the client, usually a web browser, connects to a web server such as **MS IIS** or **Apache HTTP Server**. HTTP is also used under the hood by many mobile and modern applications.



3.2 HTTP Protocol Basics

- + During an HTTP communication, the client and the server exchange **messages**.
- + The client sends **requests** to the server and gets back **responses**.



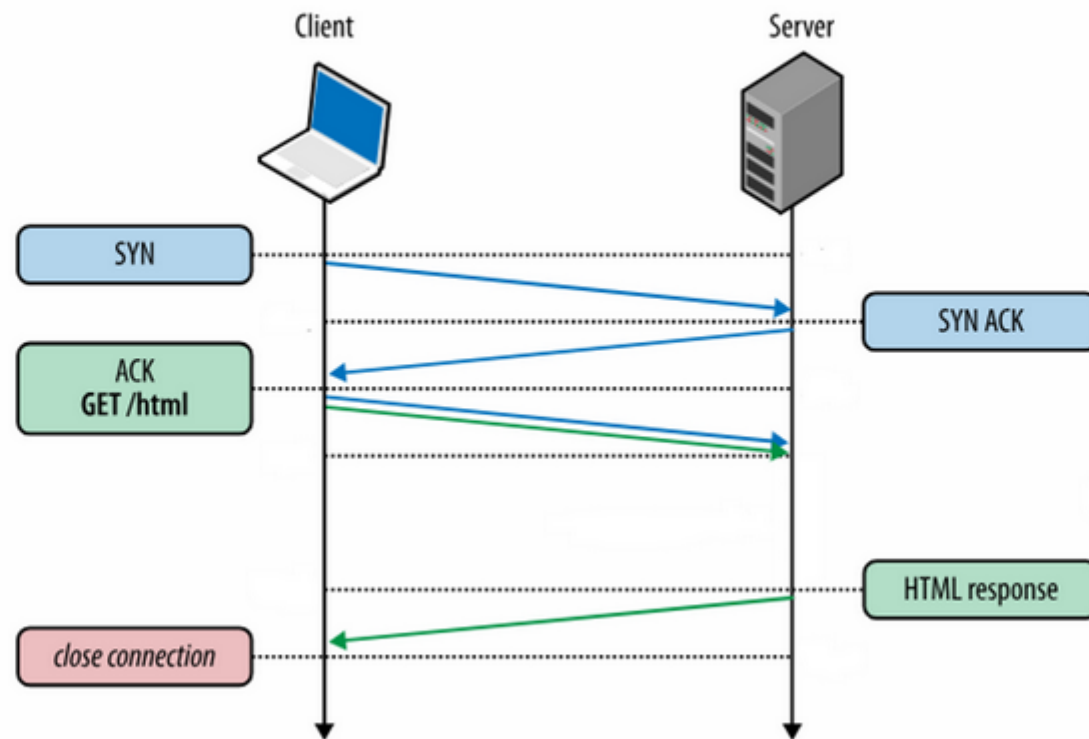


3.2 HTTP Protocol Basics

- + HTTP works on top of TCP protocol.
- + That means, first a TCP connection is established, and then the client sends its request, and waits for the answer. The server processes the request and sends back its answer, providing a status code and appropriate data.



3.2 HTTP Protocol Basics





3.2 HTTP Protocol Basics

- + The format of an HTTP message is:

Headers\r\n

\r\n

Message Body\r\n



3.2 HTTP Protocol Basics

- + To end lines in HTTP, you have to use the `\r` (carriage return) and the `\n` (newline) characters.
- + The header contains a request followed by some header fields. Every header field has the following format:

`Header-name: header value`



3.2.1 HTTP Requests

+ The following is an HTTP request example.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + This request has an empty body, as there is nothing after the two empty lines following the headers.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

+ This is the **HTTP verb** of the request.



+ The HTTP verb, or request method, states the type of the request.

```
GET / HTTP/1.1
```

```
Host: www.elearnsecurity.com
```

```
User-Agent: Mozilla/5.0 (X11; Linux  
x86_64; rv:31.0) Gecko/20100101  
Firefox/31.0 Iceweasel/31.2.0
```

```
Accept: text/html
```

```
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
```

```
Connection: keep-alive
```



3.2.1 HTTP Requests

- + GET is used when opening web resources.



- + If you open a browser and type `www.elearnsecurity.com` in the address bar, your browser will send this very same request to the server.

```
GET / HTTP/1.1
```

```
Host: www.elearnsecurity.com
```

```
User-Agent: Mozilla/5.0 (X11; Linux  
x86_64; rv:31.0) Gecko/20100101  
Firefox/31.0 Iceweasel/31.2.0
```

```
Accept: text/html
```

```
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
```

```
Connection: keep-alive
```




3.2.1 HTTP Requests

- + After the HTTP VERB you can see the **path (/)** and the **protocol version (HTTP 1.1)**.



- + The path tells the server which resource the browser is asking for. The protocol version tells the server how to communicate with the browser.

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + There are many HTTP methods, like:
 - + PUT
 - + TRACE
 - + HEAD
 - + POST
- + These are only a few but know that there are many more out there.



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + The **Host** header field specifies the Internet hostname and port number of the resource being requested.
- + A web server can host multiple websites. This header field tells the server which site the client is asking for.



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + The host value is obtained from the URI of the resource.



- + In this case:
`www.elearnsecurity.com`

```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + **User-Agent** tells the server what client software is issuing the request.
- + A client could be:
 - + **Firefox**
 - + **Internet Explorer**
 - + **Safari**
 - + **Opera**
 - + **Chrome**
 - + **A mobile app...**



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + It also reveals to the server the operating system version.



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```




3.2.1 HTTP Requests

- + The browser sends the Accept header field to specify which document type it is expecting in the response.



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + Similarly, with **Accept-Language**, the browser can ask for a specific (human) language in the response.



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + **Accept-Encoding** works similarly to **Accept** but restricts the content encoding, not the content itself.
- + In this case, the browser accepts two types of compression:
 - + gzip
 - + deflate



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.1 HTTP Requests

- + The **Connection** header field allows the sender to specify options that are desired for that particular connection.
- + Future communications with the server will reuse the current connection.



```
GET / HTTP/1.1
Host: www.elearnsecurity.com
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:31.0) Gecko/20100101
Firefox/31.0 Iceweasel/31.2.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



3.2.2 HTTP Responses

- + When the server receives a request, it processes it and then sends an **HTTP response** to the client. The response has its own header format.

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

< PAGE CONTENT > ...



3.2.2 HTTP Responses

- + As you can see, this response has a message body (< PAGE CONTENT >). The header and message body are separated by two empty lines (\r\n\r\n).

```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

```
< PAGE CONTENT > ...
```




3.2.2 HTTP Responses

- + The first line of a Response message is the **Status-Line**, which consists of the **protocol version** (HTTP 1.1) followed by a numeric **status code** (200) and its relative **textual meaning** (OK).



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

```
< PAGE CONTENT > ...
...
```



3.2.2 HTTP Responses

The more common status codes are:

- **200 OK:** the resource is found.
- **301 Moved Permanently:** the requested resource has been assigned a new permanent URI.
- **302 Found:** the resource is temporarily under another URI.



HTTP/1.1 200 OK

Date: Wed, 19 Nov 2014 10:06:45 GMT

Cache-Control: private, max-age=0

Content-Type: text/html;
charset=UTF-8

Content-Encoding: gzip

Server: Apache/2.2.15 (CentOS)

Content-Length: 99043

< PAGE CONTENT > ...

...



3.2.2 HTTP Responses

- **403 Forbidden:** the client does not have enough privileges, and the server refuses to fulfill the request.
- **404 Not Found:** the server cannot find a resource matching the request.
- **500 Internal Server Error:** the server does not support the functionality required to fulfill the request.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

```
< PAGE CONTENT > ...
...
```



3.2.2 HTTP Responses

- + **Date** represents the date and time at which the message was originated.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

```
< PAGE CONTENT > ...
...
```



3.2.2 HTTP Responses

- + With the **Cache-Control** header, the server informs the client about cached content.
- + Using cached content saves bandwidth, as it prevents the client from re-requesting unmodified content.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

```
< PAGE CONTENT > ...
...
```



3.2.2 HTTP Responses

- + **Content-Type** lets the client know how to interpret the body of the message.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

< PAGE CONTENT > ...

...



3.2.2 HTTP Responses

- + **Content-Encoding** extends Content-Type.
- + In this case, the message body is compressed with gzip.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

```
< PAGE CONTENT > ...
...
```




3.2.2 HTTP Responses

- + The **Server** header field simply contains the header of the server that generated the content.
- + This (optional) field is very useful during a pentest to identify the software running on a server.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

< PAGE CONTENT > ...

...



3.2.2 HTTP Responses

- + **Content-Length**
indicates the length,
in bytes, of the
message body.



```
HTTP/1.1 200 OK
Date: Wed, 19 Nov 2014 10:06:45 GMT
Cache-Control: private, max-age=0
Content-Type: text/html;
charset=UTF-8
Content-Encoding: gzip
Server: Apache/2.2.15 (CentOS)
Content-Length: 99043
```

```
< PAGE CONTENT > ...
```

```
...
```



3.2.3 HTTPS

- + Now that you know how HTTP works, let's see how to **protect it!**
- + HTTP content, as in every clear-text protocol, can be easily intercepted or mangled by an attacker on the path. Moreover, HTTP does not provide strong authentication between the parties.



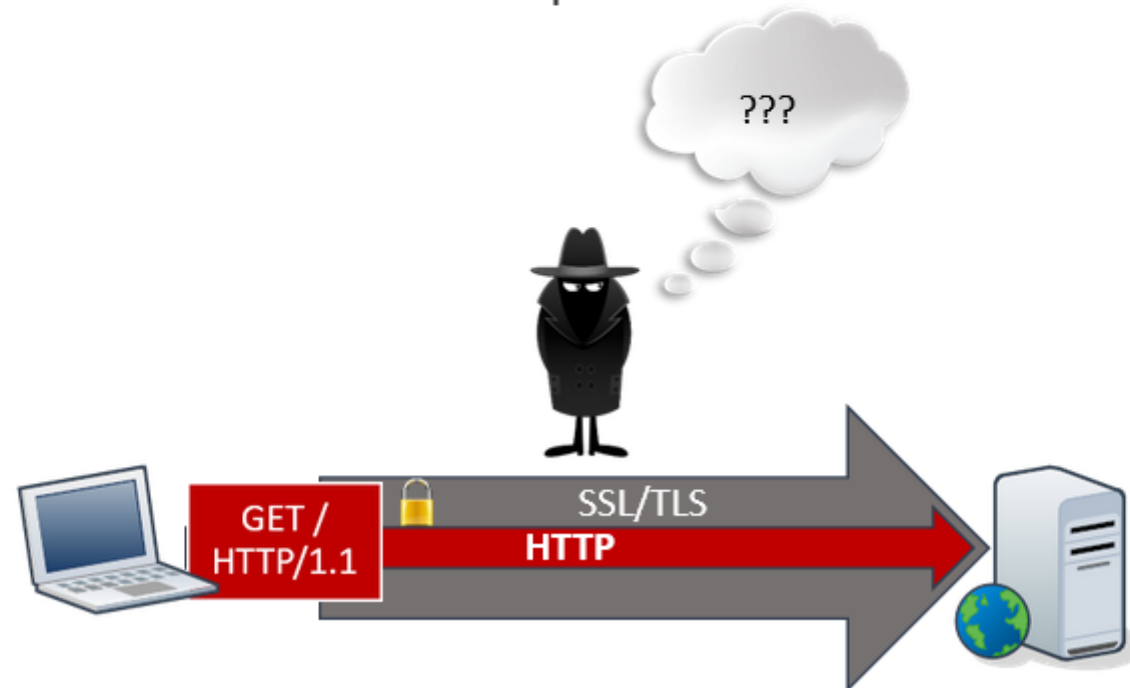
3.2.3 HTTPS

- + In the following slides, you will see how to **protect HTTP** using an **encryption layer**.
- + **HTTP Secure** (HTTPS), or HTTP over SSL/TLS, is a method to run HTTP which is a clear-text protocol over SSL/TLS, a cryptographic protocol.



3.2.3 HTTPS

- + This layering technique provides confidentiality, integrity protection and authentication to the HTTP protocol.





3.2.3 HTTPS

- + In other words, when using HTTPS:
 - An attacker on the path cannot sniff the application layer communication.
 - An attacker on the path cannot alter the application layer data.
 - The client can tell the real identity of the server and, sometimes, vice-versa.



3.2.3 HTTPS

- + HTTPS offers encryption, which means that a network adjacent user is able to sniff the traffic, but he will not know:
 - HTTP Request headers, body, target domain
 - HTTP Response headers, body
- + On the other hand, when inspecting HTTPS, one cannot know what domain is contacted and what data is exchanged.



3.2.3 HTTPS

- + A network adjacent user might recognize:
 - Target IP address
 - Target port
 - DNS or similar protocols may disclose which domain user tries to resolve



3.2.3 HTTPS

- + **HTTPS does not protect against web application flaws!** All the attacks against an application happen regardless of SSL/TLS.
- + The extra encryption layer just protects data exchanged between the client and the server. It does not protect from an attack against the application itself.



3.2.3 HTTPS

- + Attacks such as XSS and SQL injections will still work.
- + Understanding how HTTP and web applications work is fundamental to mount stealthy and effective attacks!



References

- + [HTTP Status Code Reference](#): a document referencing HTTP status codes and their meaning.
- + [SSL/TLS Strong Encryption: An Introduction](#): Apache's introduction on protecting HTTP by means of SSL/TLS.
- + [HTTP Overview, History, Versions and Standards](#): History and evolution of HTTP.



HTTP Cookies



3.3 HTTP Cookies

- + HTTP is a **stateless** protocol; this means that websites cannot keep the state of a visit across different HTTP requests.
- + In other words, every HTTP request is **completely unrelated** to the ones preceding and following it.



3.3 HTTP Cookies

- + To overcome this limitation, sessions and **cookies** were invented in 1994.
- + **Netscape**, a leading company at that time, invented cookies **to make HTTP stateful**.



3.3 HTTP Cookies

- + **How does this support my pentesting career?**
 - Cookies are foundation of authorization of many applications
 - Often exploits rely on stealing cookies



3.3 HTTP Cookies

- + Cookies are not rocket science. They are just textual information installed by a website into the "cookie jar" of the web browser.
- + The **cookie jar** is the storage space where a web browser stores the cookies.





3.3.1 Cookies Format

A server can set a cookie via the **Set-Cookie** HTTP header field in a response message.

A cookie contains the following attributes:

- + The actual content
- + An expiration date
- + A path
- + The domain
- + Optional flags:
 - + Http only flag
 - + Secure flag

HTTP/1.1 200 OK

Date: Wed, 19 Nov 2014 10:06:45 GMT

Cache-Control: private, max-age=0

Content-Type: text/html;
charset=UTF-8

Content-Encoding: gzip

Server: Apache/2.2.15 (CentOS)

Set-Cookie: ID=Value; expires=Thu,
21-May-2015 15:25:20 GMT; path=/;

domain=.example.site; HttpOnly

Content-Length: 99043



3.3.1 Cookies Format

Cookie content	{ ID=Value;
Expiration date	{ expires =Thu, 21-May-2015 15:25:20 GMT;
Path	{ path =/example/path;
Domain	{ domain =.example.site;
Flag-setting attributes	{ HttpOnly ; Secure



3.3.2 Cookies Handling

- + Browsers use domain, path, expires and flags attributes to choose whether or not to send a cookie in a request.
- + Cookies are sent only to the **valid domain/path** when they are **not expired** and according to their **flags**.



3.3.3 Cookie Domain

- + The **domain** field and the **path** field set the **scope** of the cookie. The browser sends the cookie only if the request is for the right domain.
- + When a web server installs a cookie, it sets the domain field, e.g., `elearnsecurity.com`. Then, the browser will use the cookie for every request sent to that domain and **all its subdomains**.



3.3.3 Cookie Domain

EXAMPLE

+ When a cookie has the domain attribute set to:

- `domain=elearnsecurity.com`

or

- `domain=.elearnsecurity.com`

+ The browser will send the cookie to:

- `www.elearnsecurity.com`
- `whatever.subdomain.elearnsecurity.com`
- `elearnsecurity.com`

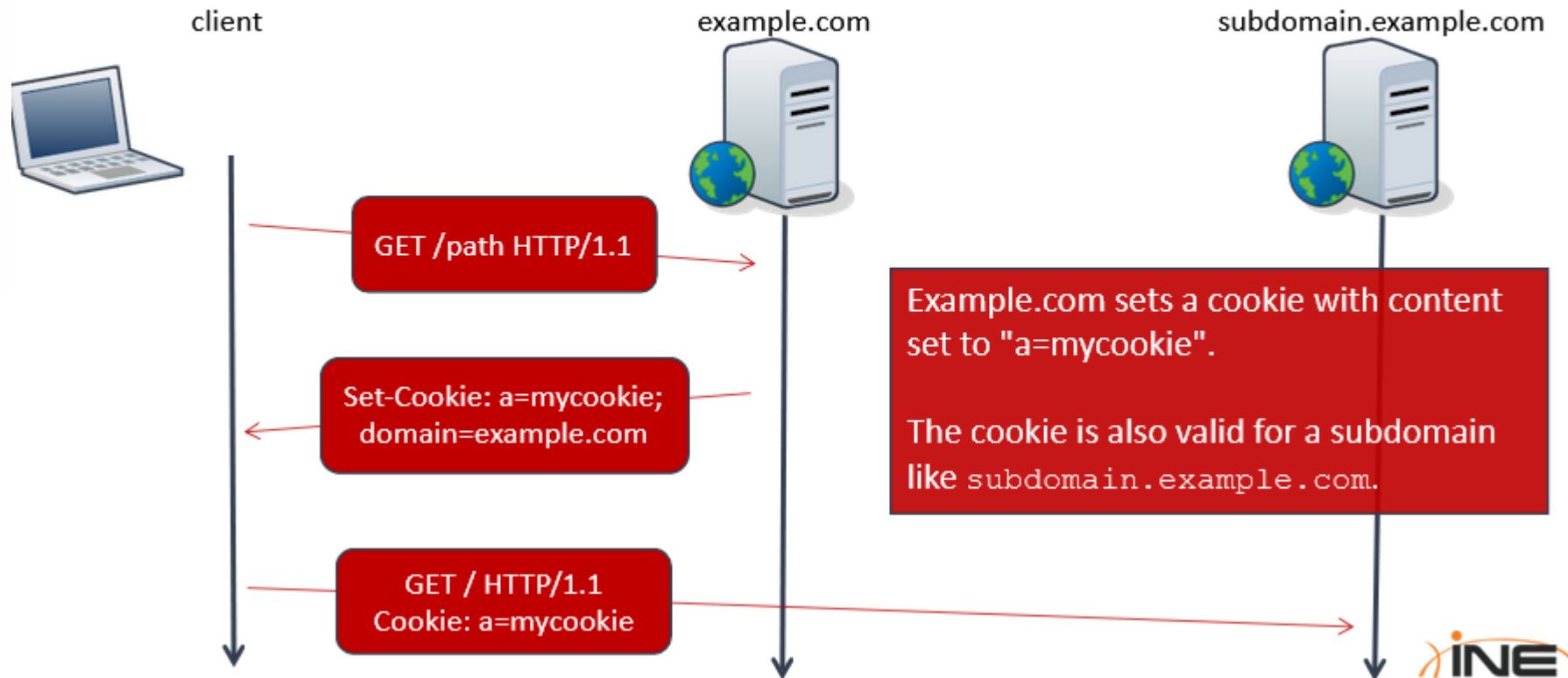


3.3.3 Cookie Domain

- + If the **server does not specify** the domain attribute, the browser will automatically set the domain as the server domain and set the cookie **host-only** flag; this means that the cookie will be sent **only to that precise hostname**.
- + In the following examples, you will see how a browser chooses to send or not to send a cookie.

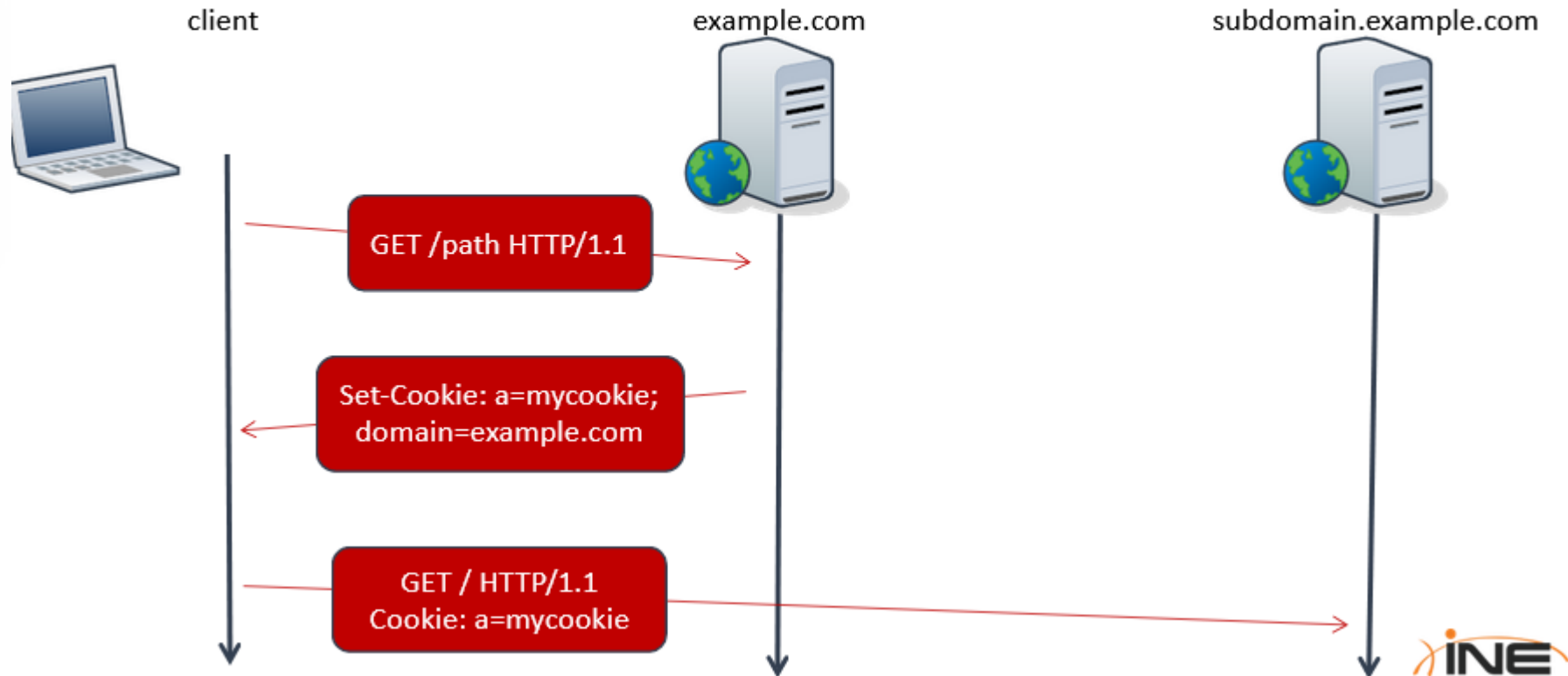


3.3.3.1 Cookie Domain Examples



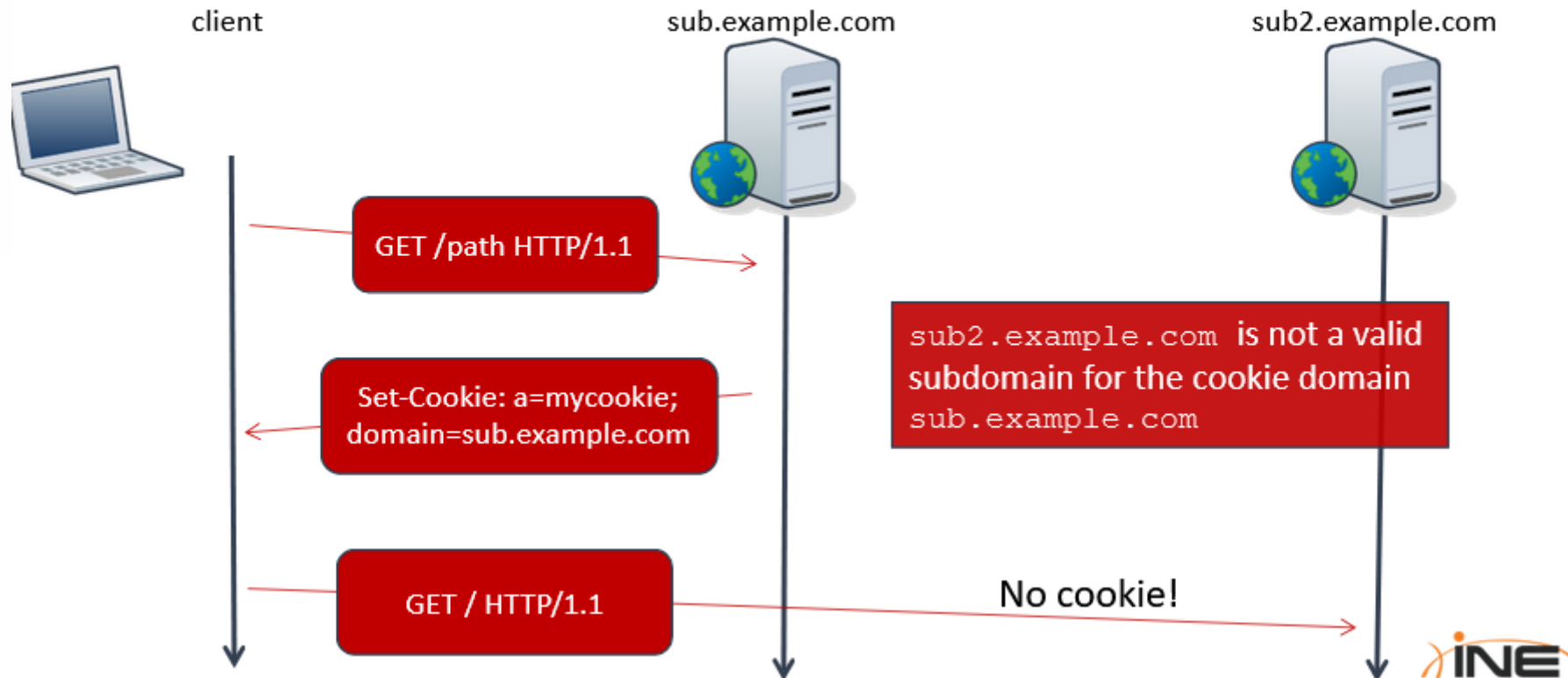


3.3.3.1 Cookie Domain Examples



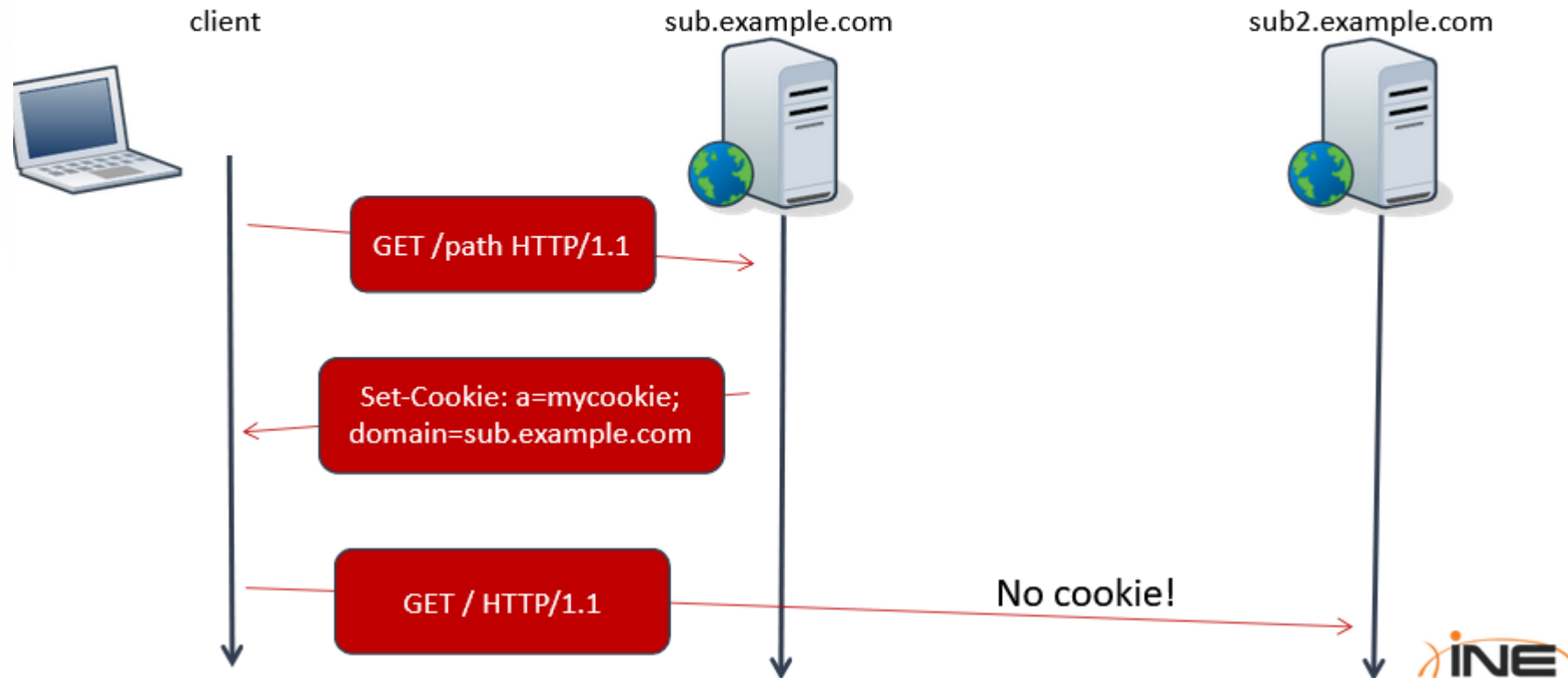


3.3.3.1 Cookie Domain Examples



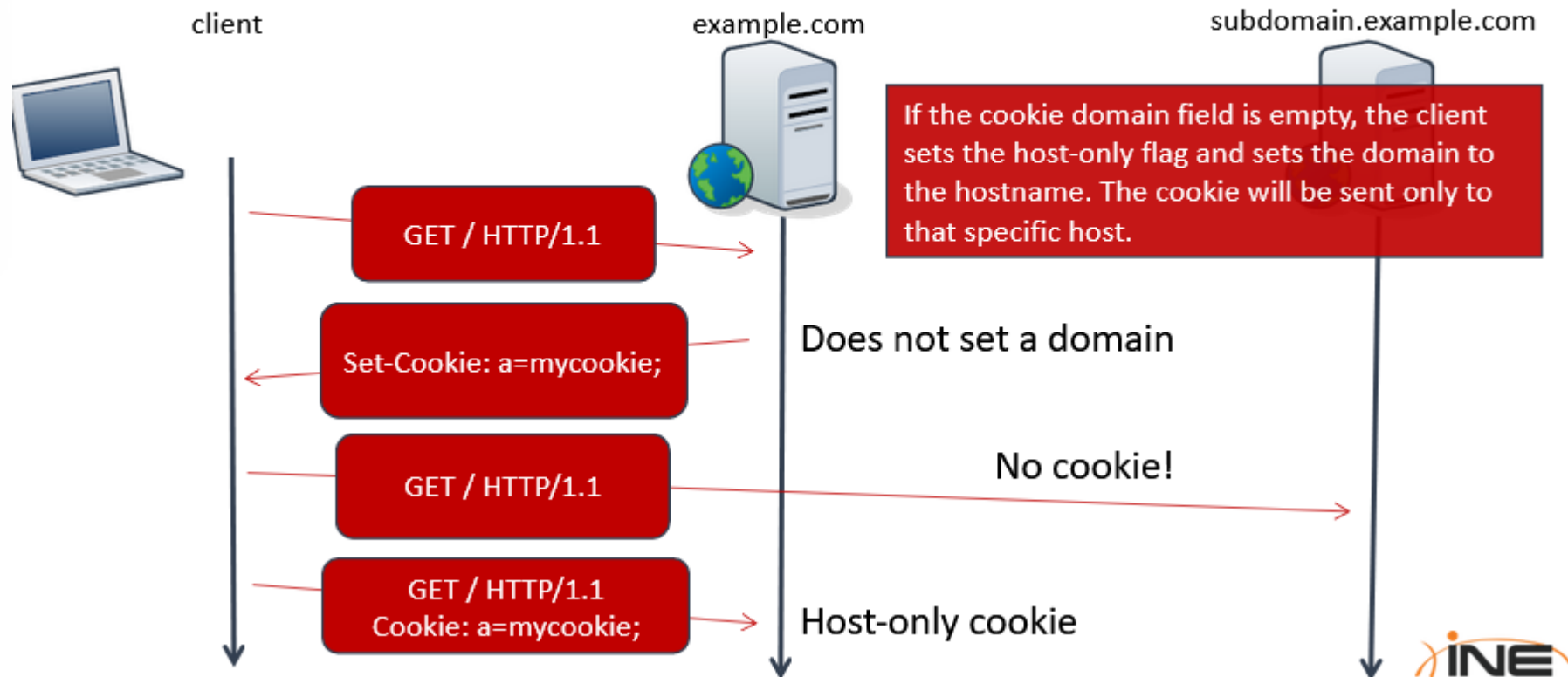


3.3.3.1 Cookie Domain Examples



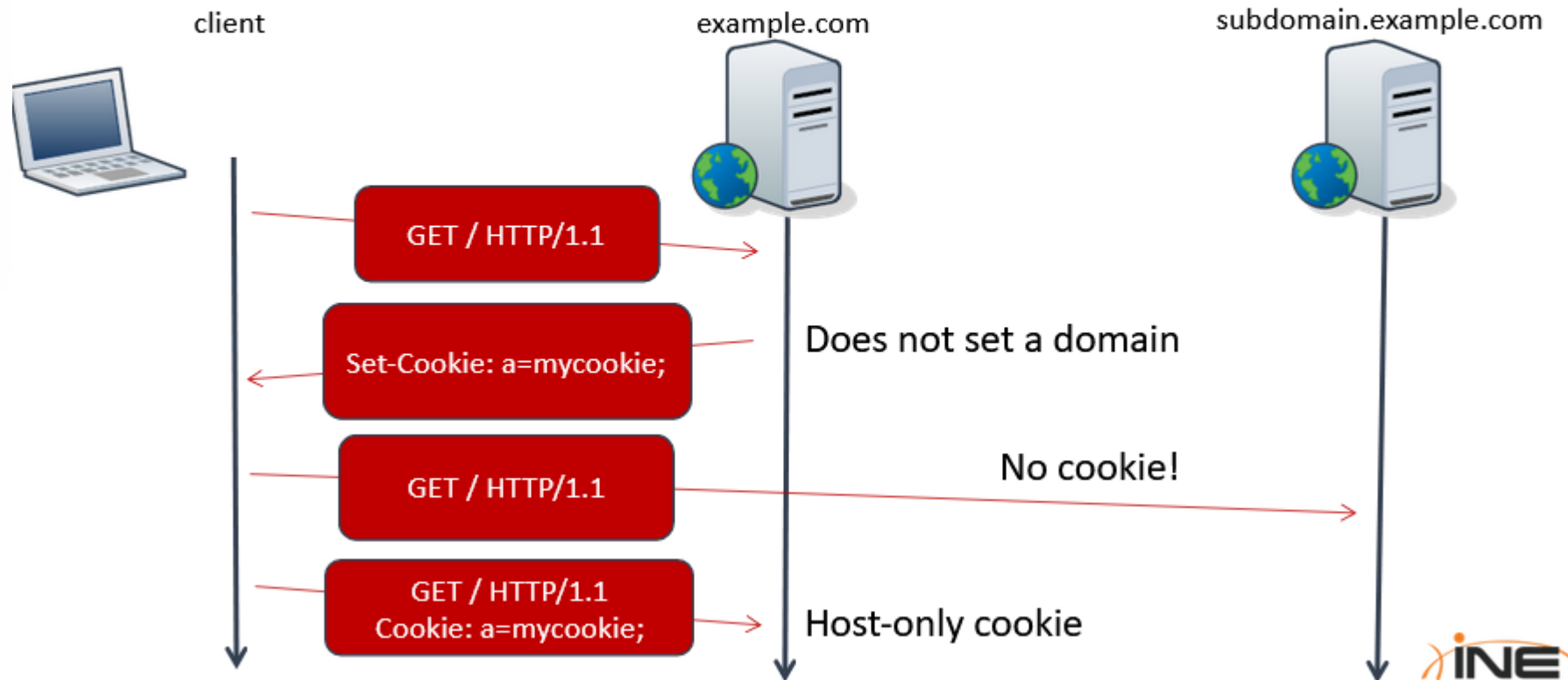


3.3.3.1 Cookie Domain Examples





3.3.3.1 Cookie Domain Examples





3.3.4 Cookie Path

- + As we saw previously, the **path** and the **domain** attributes set the **scope** of a cookie.
- + The browser will send a cookie to the **right domain and to any subpath of the path** field value.



3.3.4 Cookie Path

EXAMPLE

- + When a cookie has the path attribute set to:
 - + `path=/the/path`
- + The browser will send the cookie to the right domain and to the resources in:
 - + `/the/path`
 - + `/the/path/sub`
 - + `/the/path/sub/sub/sub/path`

but it will not send it to `/otherpath`.



3.3.5 Cookie Expires Attribute

- + The **expires** attribute sets the **validity time window** of a cookie.
- + A browser will not send an expired cookie to the server. Session cookies expire with the HTTP session; you will see more about that later in this module.



3.3.6 Cookie Http-Only Attribute

- + When a server installs a cookie into a client with the **http-only attribute**, the client will set the **http-only flag** for that cookie. This mechanism prevents JavaScript, Flash, Java and any other non-HTML technology from reading the cookie, thus preventing cookie stealing via XSS.
- + You will see how to exploit XSS vulnerabilities in the *Web Application Attacks* module.



3.3.7 Cookie Secure Attribute

- + **Secure flag** creates secure cookies that will only be sent over an HTTPS connection (they will not be sent over HTTP).



3.3.8 Cookie Content

EXAMPLE

- + A cookie can carry a number of values. A server can set multiple values with a single `Set-Cookie` header by specifying multiple `KEY=Value` pairs.

Set-Cookie: Username="john"; auth=1

- + Sets two values: one for username and one for auth.



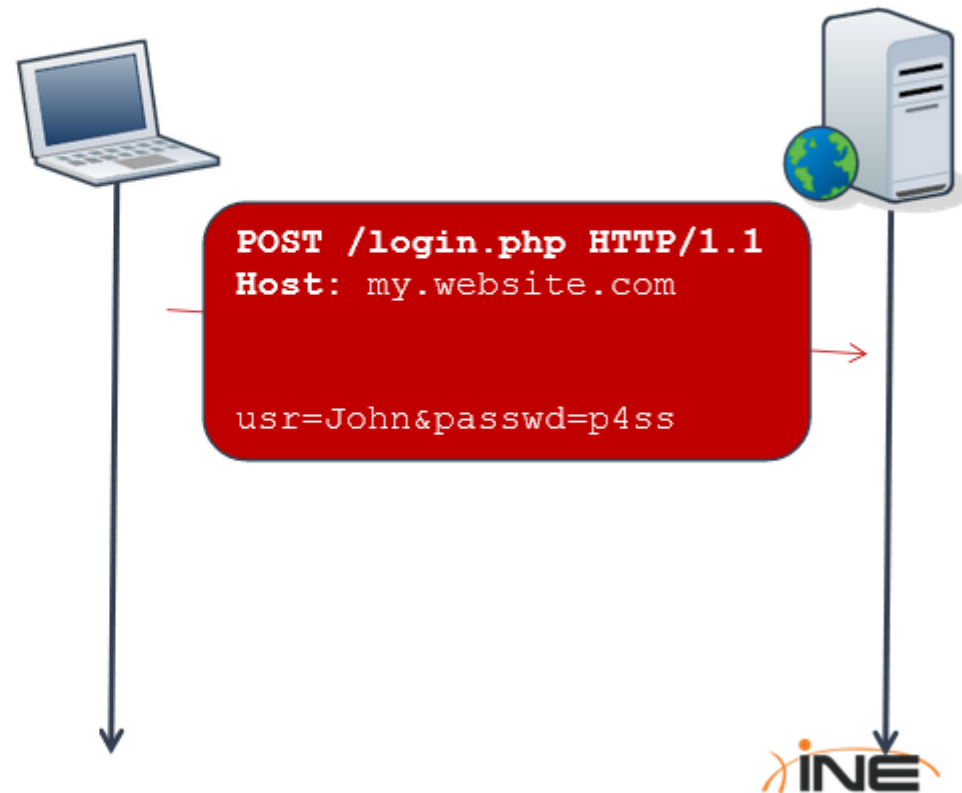
3.3.9 Cookie Protocol

- + [RFC6265](http://tools.ietf.org/html/rfc6265) states cookies format, how a server can install cookies and how a client can use them.
- + Let's see cookies in action with a simple example.



3.3.9 Cookie Protocol

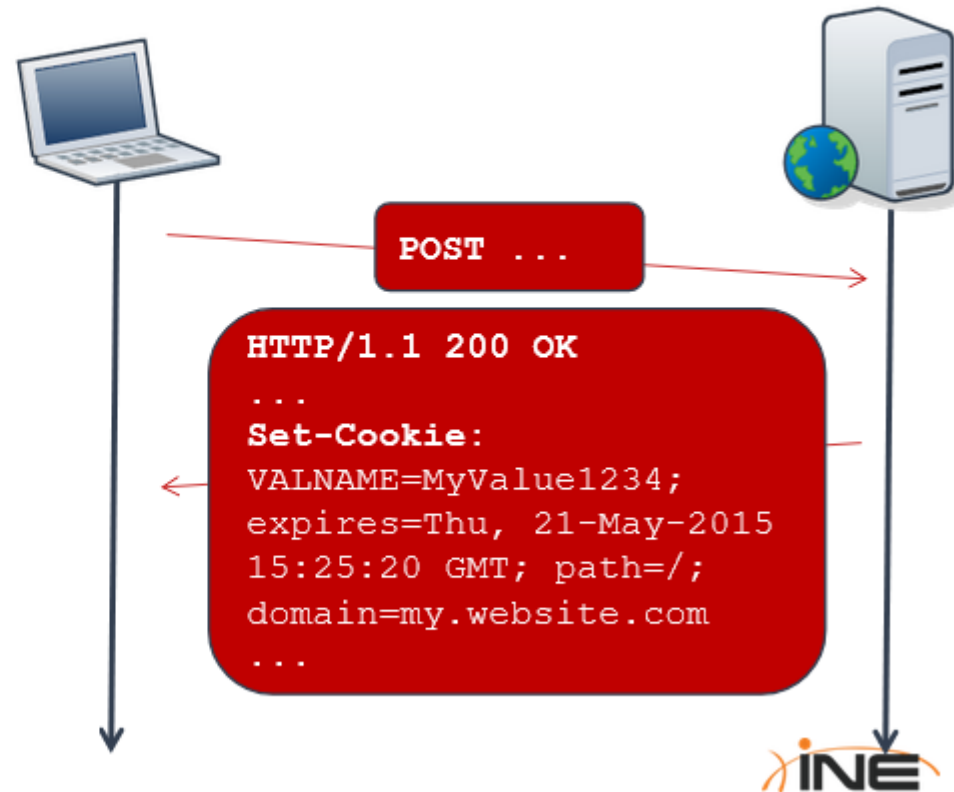
- + Cookies are often installed during a login.
- + In this example, the browser sends a **POST** request with the username and password.





3.3.9 Cookie Protocol

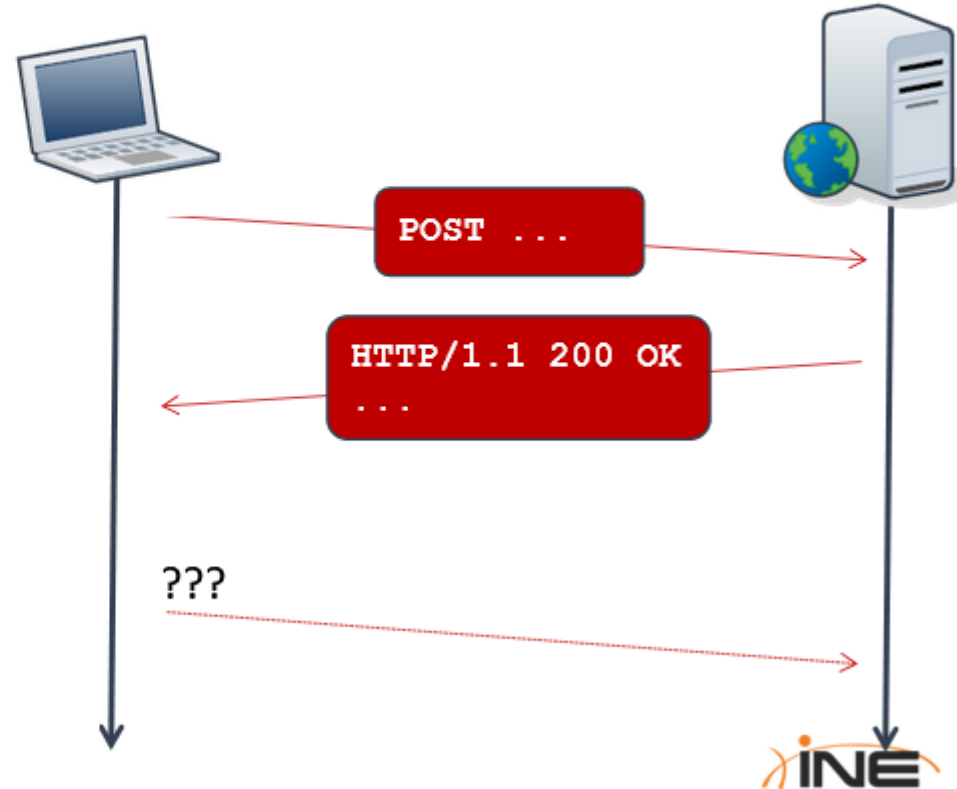
- + The server sends a response with a **Set-cookie** header field, thus telling the browser to install the cookie.





3.3.9 Cookie Protocol

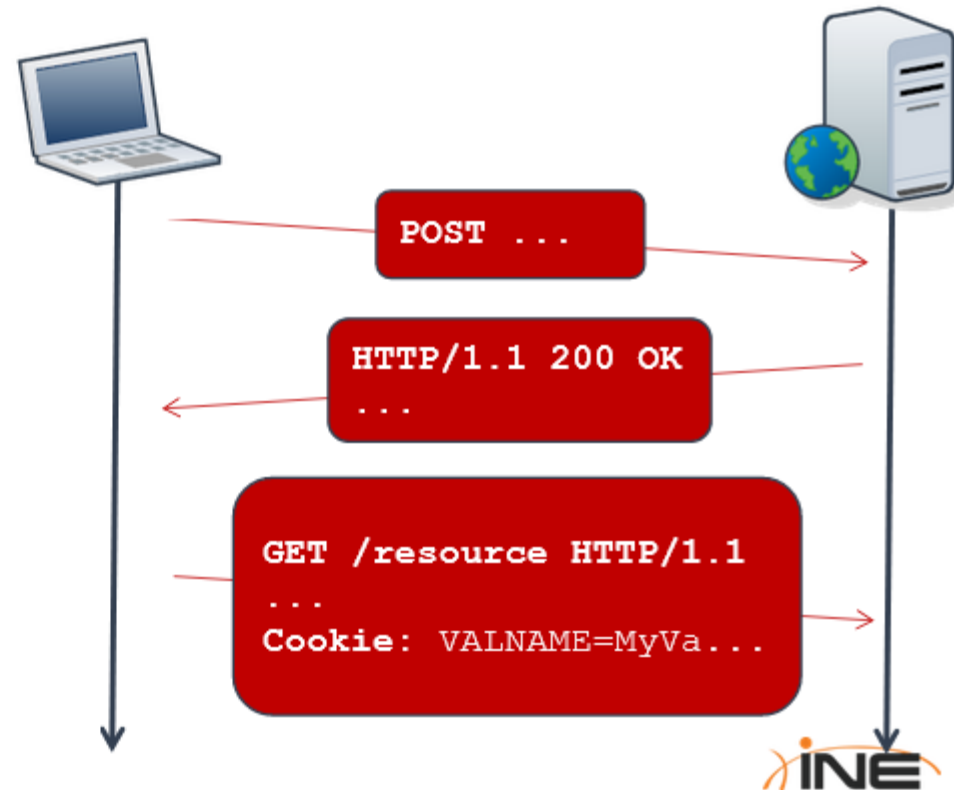
- + For every subsequent request, the browser considers:
 - + Domain
 - + Path
 - + Expiration
 - + Flags





3.3.9 Cookie Protocol

- + If all the checks pass, the browser will insert a **cookie:** header in the request.





Sessions



3.4 Sessions

- + Sometimes the web developer prefers to store some information on the **server side** instead of the client side; this happens to hide the application logic or just to avoid the back and forth data transmission typical of cookies.
- + **Sessions** are a mechanism that lets the website store variables specific for a given visit on the **server side**.
- + Each user session is identified by a **session id**, or token, which the server assigns to the client.

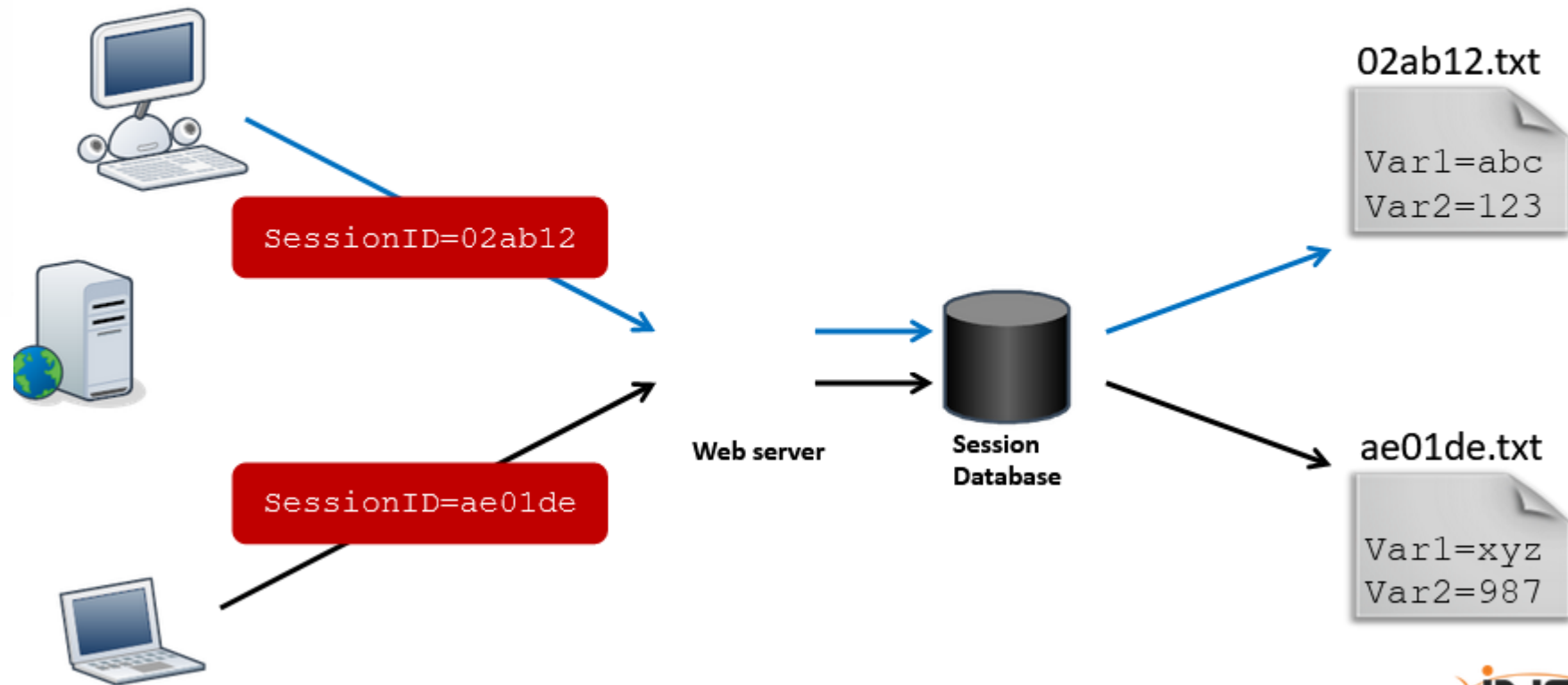


3.4 Sessions

- + The client then presents this ID for each subsequent request, thus being recognized by the server.
- + By means of the session ID, the **server retrieves the state of the client** and all its associated variables. The server stores Session IDs inside text files in its storage. You can find an example in the next slide.



3.4.1 Sessions Example





3.4.2 Session Cookies

Q

How does a web application install session IDs on a web browser?

A

By using session cookies.

+ It's now time to see how to use **HTTP sessions**!



3.4.2 Session Cookies

- + Session cookies just contain a single parameter value pair referring to the session.

```
SESSION=0wvCtOBWDH8w  
PHPSESSID=13Kn5Z6Uo4pH  
JSESSIONID=W7DPUBgh7kTM
```



3.4.2 Session Cookies

- + Websites running PHP install session cookies by using the "PHPSESSID" parameter name, while JSP websites use "JSESSIONID". Each development language has its own default session parameter name.
- + Of course, the web developer can also choose to use a custom parameter name.



3.4.2 Session Cookies

- + If needed, servers install session cookies after a browser performs some kind of activity, like:
 - Opening a specific page
 - Changing settings in the web application
 - Logging in



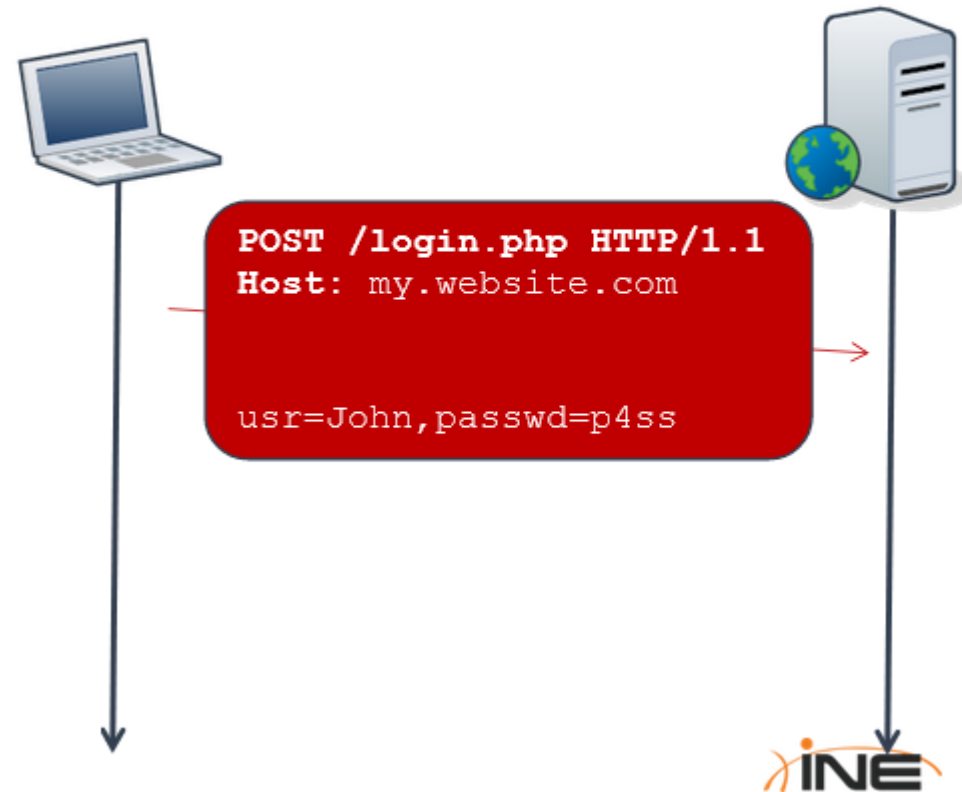
3.4.2 Session Cookies

- + The browser then uses the cookie in subsequent requests. A session could contain many variables, so sending a small cookie keeps the bandwidth usage low.
- + In the following example, you can see a session cookie in action.



3.4.2.1 Session Cookies Example

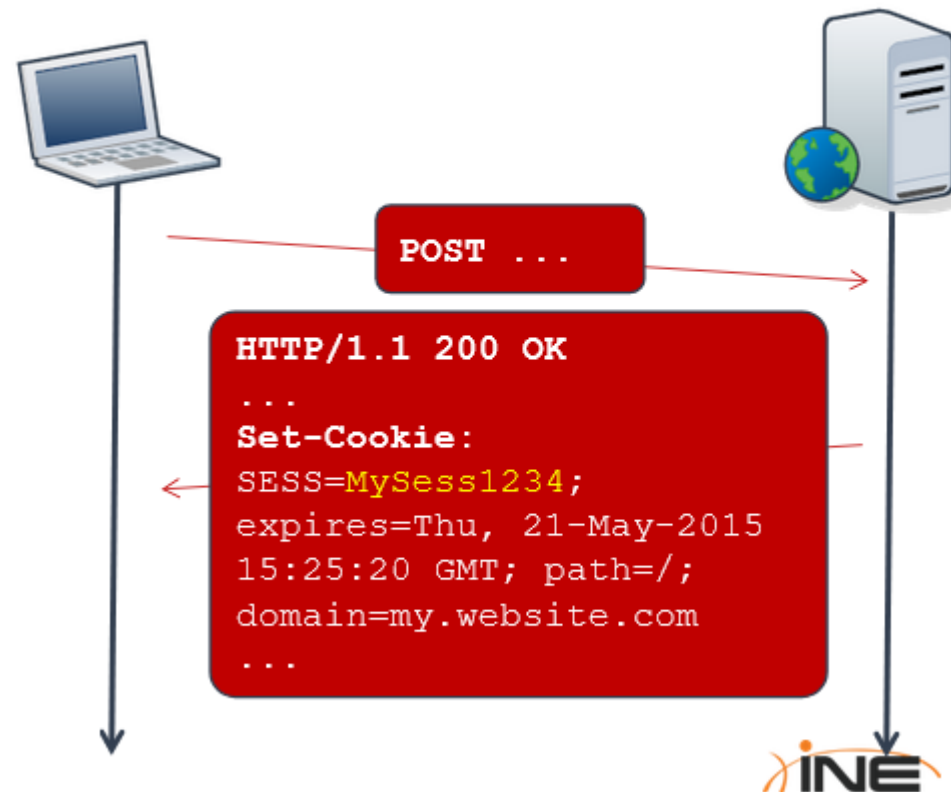
- + The client uses a login form to POST the user's credentials.





3.4.2.1 Session Cookies Example

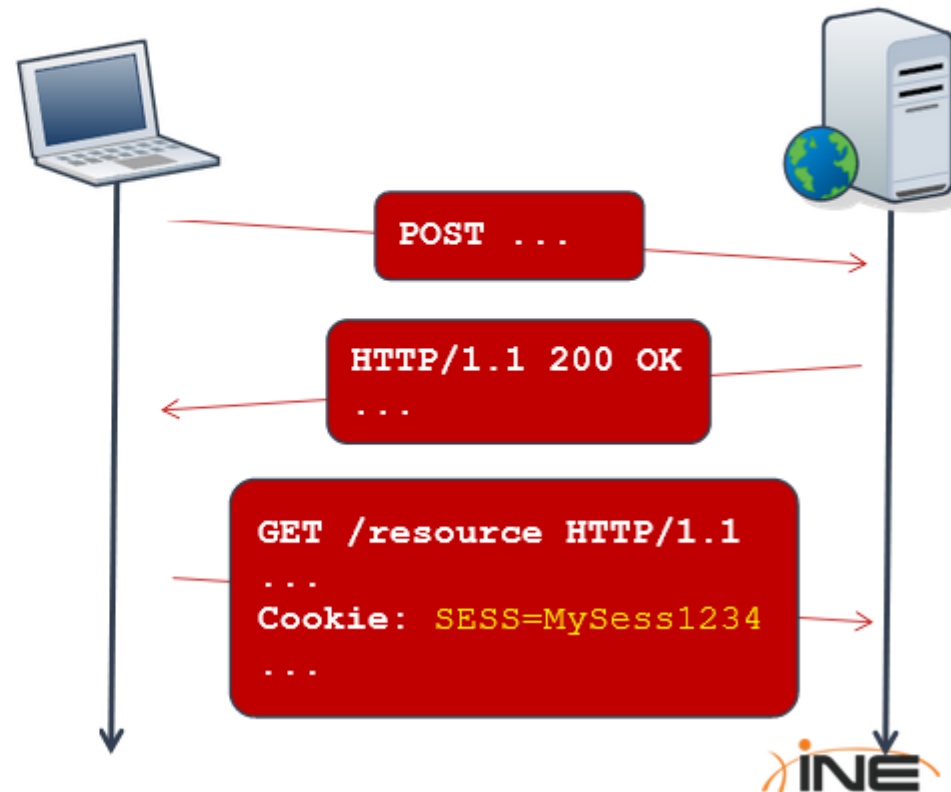
- + The server sends back a response with a Set-cookie header field.
- + The cookie contains the **session ID**.





3.4.2.1 Session Cookies Example

- + The browser will send back the cookie according to the cookie protocol, thus sending the **session ID**.





3.4.3 GET Requests

EXAMPLE

- + Session IDs can also be transmitted via **GET requests**.

```
http://example.site/resource.php?sessid=k27rds7h8w
```



Same Origin Policy



3.5 Same Origin Policy

- + **Same Origin Policy** (SOP) is a critical point of web application security.
- + This policy prevents JavaScript code from getting or setting properties on a resource coming from a **different origin**.



3.5 Same Origin Policy

+ The browser uses:

Protocol

Hostname

Port

+ To determine if JavaScript can access a resource: *Hostname*, *port*, and *protocol* **must match**.



3.5 Same Origin Policy

EXAMPLE

A JavaScript script on

- https://www.elearnsecurity.com:345/
protocol hostname port

can read resources from:

- `https://www.elearnsecurity.com:345/path`
- `https://www.elearnsecurity.com:345/path/2`



3.5 Same Origin Policy

EXAMPLE

+ But not from:

- `https://www.elearnsecurity.com/path`
(same protocol and domain but different port)
- `http://www.elearnsecurity.com:345/path`
(same port and domain but different protocol)
- `https://www.heralab.net:345/path`
(same port and protocol but different domain)



3.5.1 HTML Tags

- + Note that SOP applies only to the **actual code of a script**.
- + It is still possible to include external resources by using HTML tags like `img`, `script`, `iframe`, `object`, etc.



3.5 Same Origin Policy

- + The entire web application security is based on Same Origin Policy.
- + If a script on domain A was able to read content on domain B, it would be possible to steal clients' information and mount a number of very dangerous attacks.

