

# HTTP Protocol And Web Security Overview



# License & Disclaimer

2

## License Information

This presentation is licensed under the  
Creative Commons BY-NC License



To view a copy of the license, visit:

<http://creativecommons.org/licenses/by-nc/3.0/legalcode>

## Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

# Goal

3

- Present the history of the HTTP
- Show the key features of the protocol
- Present the definition of Web Security and give a classification of the attacks
- List useful tools commonly used in Web Security

# Prerequisites

4

## ➤ Lecture:

➤ *NS\_0.1 – Network Fundamentals*

# Outline

5

- HTTP History
- Key Features and Overview of HTTP
- Security and Web Security
- Tooling

# Outline

6

- HTTP History
- Key Features and Overview of HTTP
- Security and Web Security
- Tooling

# HTTP History

7

- HTTP was introduced at the beginning of the '90s
- The first version of the protocol, **HTTP 0.9**, was released under the World Wide Web initiative
  - Extremely simple
  - Released in 1991
- <https://www.w3.org/Protocols/HTTP/AsImplemented.html>

# HTTP History

8

- HTTP initial goal was to **share documents**
- Every document was (and still is) written in **HTML**
  - The first version of the language, HTML 1.0, is a barebone language whose main goal was to format texts and to connect them through hyperlinks
- The **first** example of a **browser** for this language was called "WorldWideWeb"



# HTTP History

9

- HTTP
- Every
- The
- The

## World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#), [Policy](#), November's [W3 news](#), [Frequently Asked Questions](#).

### [What's out there?](#)

Pointers to the world's online information, [subjects](#), [W3 servers](#), etc.

### [Help](#)

on the browser you are using

### [Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#), [X11 Viola](#), [NeXTStep](#), [Servers](#), [Tools](#), [Mail robot](#), [Library](#))

### [Technical](#)

Details of protocols, formats, program internals etc

### [Bibliography](#)

Paper documentation on W3 and references.

### [People](#)

A list of some people involved in the project.

### [History](#)

A summary of the history of the project.

### [How can I help?](#)

If you would like to support the web..

### [Getting code](#)

Getting the code by [anonymous FTP](#), etc.

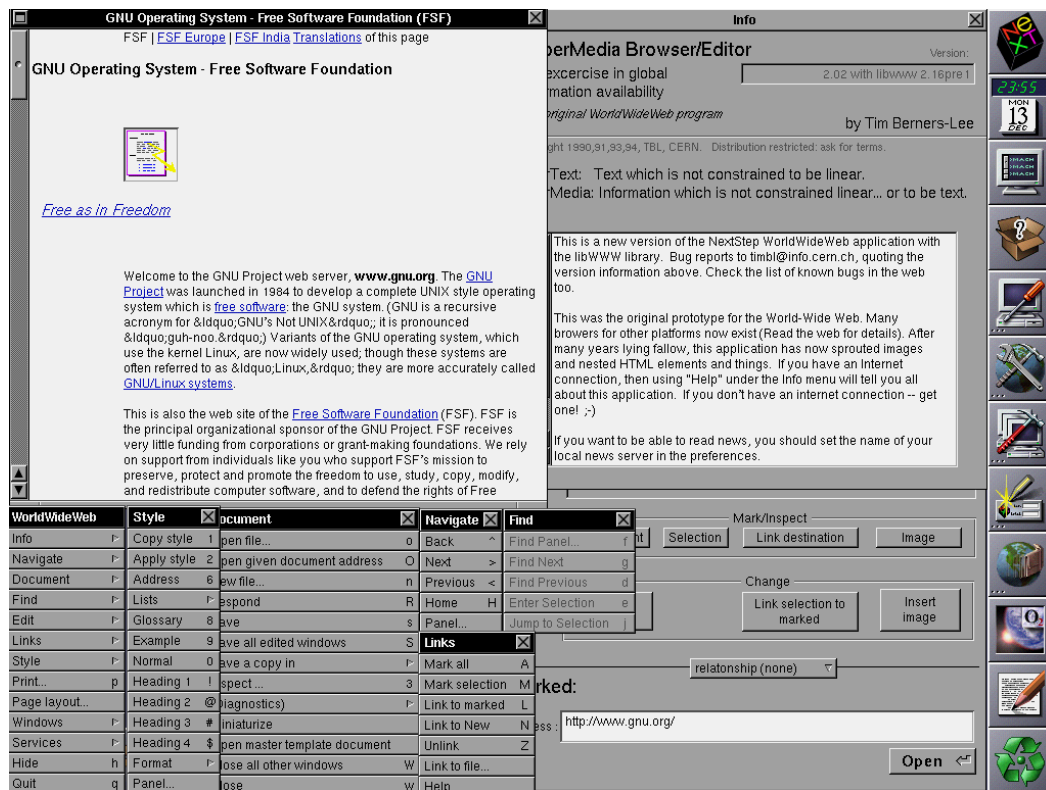
FROM: W3.ORG

HTML  
are bone  
d to  
age was

# HTTP History

10

- HTTP
- Every
- The
- lan
- cor
- The f
- calle



HTML  
are bone  
d to  
age was

# HTTP - History

11

- Through the years, **browsers became more and more complex**
- **Mosaic** in mid-1993 brought new features, such as the possibility to embed images into web pages
- Several software houses started to develop their own browser, adding new features to defeat the concurrence
  - HTML enchantments
  - JavaScript
  - Plugins such as Java/Flash

# HTTP - History

12

- This race (called the "**browsers war**") led to a vast diversity of standards
- Each browser implemented its own (often undocumented) heuristics to maintain compatibility with other browsers
  - Often ignoring all the security implications

# HTTP - Present

13

- In an effort to mitigate this anarchy, in **1994** the **W3C consortium** was created
  - The goal was to set mandatory web standards for vendors
- Eventually, vendors started to follow these standards, and by now the vast majority of the problems introduced by the "browsers war" are solved

# Outline

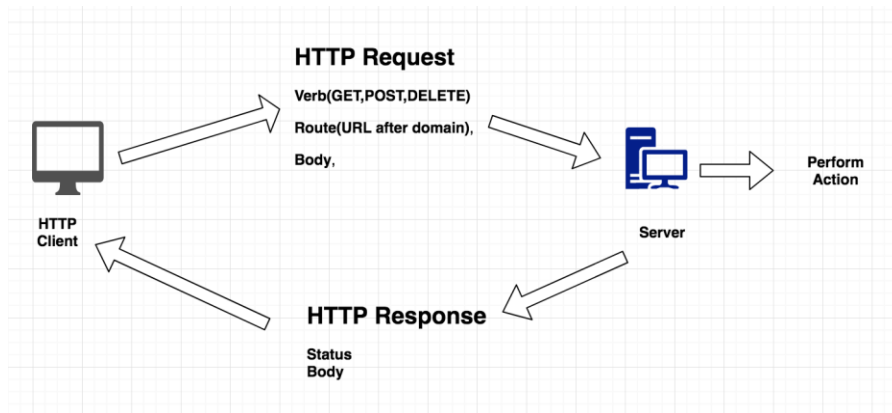
14

- HTTP History
- **Key Features and Overview of HTTP**
- Security and Web Security
- Tooling

# HTTP Overview

15

- Defined in **RFC2616**<sup>1</sup>
- High level protocol (Application level in the ISO/OSI stack)
- Mainly on TCP
  - Ports 80/443
  - HTTPS <-- HTTP over TLS
- Human readable
- Client/Server architecture
- Stateless



<https://tools.ietf.org/html/rfc2616>

# HTTP Overview

16

- HTTP is about **resources**
- A resource is an asset that a client requests to access, and it can be
  - A HTML file
  - An image
  - An information
  - ...



# HTTP Overview

17

- Resources are uniquely represented with **URLs**, acronym of
  - Uniform Resource Locator
- URLs are defined in RFCs **1738**<sup>1</sup> (URLs) and **3986**<sup>2</sup> (URIs)

1: <https://tools.ietf.org/html/rfc1738>

2: <https://tools.ietf.org/html/rfc3986>

# HTTP Overview - URLs

18

- An URL may be the following:

`http://foobar.com:8080/view.php?id=1`

# HTTP Overview - URLs

19

- An URL may be the following:

`http://foobar.com:8080/view.php?id=1`

Schema

- The **schema** specifies the protocol used, i.e. http or https

# HTTP Overview - URLs

20

- An URL may be the following:

`http://foobar.com:8080/view.php?id=1`

Host

Port

- The **host** and the **port** represent the address to which the client should connect and send the request

# HTTP Overview - URLs

21

- An URL may be the following:

`http://foobar.com:8080/view.php?id=1`

URL-Path

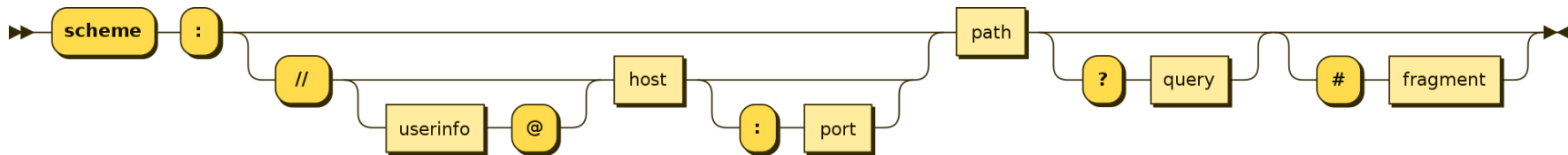
- The **URL-path** is the resource requested to the server

# HTTP Overview - URLs

22

- An URL may be the following:

`http://foobar.com:8080/view.php?id=1`



# HTTP Overview - URLs

23

- Optionally an URL can also contain an username and a password. These are used if the server requires an HTTP Basic authentication. An example of such url is the following one:

`http://admin:password@foobar.com/`

# HTTP Overview - URLs

24

- The **URL-path** has its own syntax in http, but since its interpretation is reserved to the backend, this syntax may vary from web app to web app
- Usually, it is in the form of

`/<directory>/<file>?<query>#fragment`



# HTTP Overview - URLs

25

- The <directory>/<file> part is used to represent the "physical" location of the resource on the filesystem of the server
- Right now, it is more a "virtual" location

<https://twitter.com/Twitter/status/1212237037631352832>

# HTTP Overview - URLs

26

- The <query> is optional and it can be used to send information to the backend
- It is a dictionary with a **key-value representation**, generally in the form of

`varname1=value1&varname2=value2`

# HTTP Overview - URLs

27

- The fragment is a piece of information that is reserved for **clients**
- Clients never send it to the server
- If the fragment appears in a request, the server will ignore it

# HTTP Overview - URLEncoding

28

- Looking at the URL syntax one may notice that some characters have a **special meaning** in a URL
  - The character "#" is used for segments, and the server will always ignore every character after it
  - The character "&" is used as a variable separator in the URL-Path
  - ...

# HTTP Overview - URLEncoding

29

- What if we want to send the text «hello &#» in a GET variable?

`http://foobar.com/?var=hello &# world`

- Because the fragment is reserved for clients, the server will **ignore** the word «world»

# HTTP Overview - URLEncoding

30

- This problem is solved using a **particular encoding**, that converts every character in a "not harmful" representation
- This encoding is called “**URL encoding**” or “**Percent Encoding**”

# HTTP Overview - URLEncoding

31

- This encoding is very simple
  - Take the hex value of a character you want to encode, and then prepend a "%" symbol

# == %23

- Every reserved character in a URL must be urlencoded
- Every non-printable character must be urlencoded
- Spaces can be represented either with %20, or with the plus sign (+)

# HTTP Overview - URLEncoding

32

- So the following not valid URL

`http://foobar.com/?var=hello &# world`

- Is rewrote as

`http://foobar.com/?var=hello+%26%23+world`



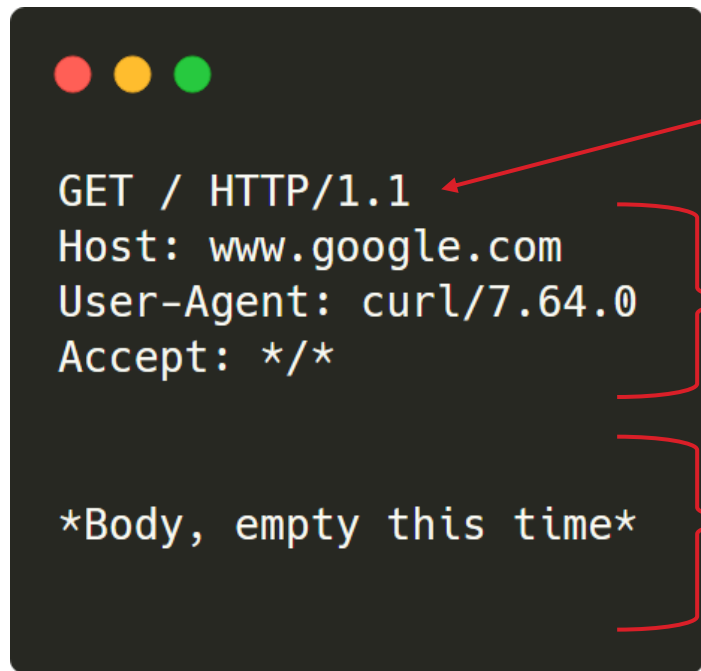
# HTTP Overview

33

- **Requests** and **Responses** are HTTP messages composed of three different parts
  - The Request/Response line
  - The Header Fields
  - A Body (optional)
- Every line in the Request/Response is terminated by the "CR;LF" sequence: `\r\n` or `0x0d0a` in binary
- An empty line separates the last header field from the body

# HTTP Overview - Requests

34



```
GET / HTTP/1.1
Host: www.google.com
User-Agent: curl/7.64.0
Accept: */*

*Body, empty this time*
```

Request Line

Header Fields

Body Field

# HTTP Overview - Requests

35

- The Request line is composed of
  - A **method**
    - "we want to do X with the resource"
  - The **resource** for which we are doing the request
  - And, finally, the **protocol version**

GET / HTTP/1.1

# HTTP Overview - Methods

36

- Tell the server "what we are doing" with the resource
- Standard methods
  - GET
  - POST
  - OPTIONS
  - HEAD
- It is also possible to define custom methods

# HTTP Overview - Body

37

- **Generic data** sent to the server
- Its type (or encoding) is defined by the **Content-Type** header
- It can be **encoded in different ways**:
  - application/x-www-form-urlencoded
  - text/plain
  - ...
- It can also have a **custom encoding**:
  - application/json
  - foo/bar
  - ...

# HTTP Overview - Headers

38

- Headers are used to send **additional data** to the server
- Serialized in the form **name: value**
- Some are mandatory:
  - **Host**
  - **Content-Encoding/Content-Length** if there is a body

# HTTP Overview - Responses

39

- A Response is very similar to a Request
- It differs only for the **first line**, which is called "status-line"
- This line is mandatory, and tells the client the type of the response and the version of the protocol used to make the response

# HTTP Overview - Responses

40

HTTP/1.1 200 Ok

Host: 127.0.0.1:5000

Date: Thu, 12 Mar 2020 10:31:38 GMT

Connection: close

X-Powered-By: PHP/7.4.3

**Hello World!**

Status-Line

Header Fields

Body field



# HTTP Overview – Status Line

41

- The status-line composed by the **version of the protocol**, an **integer number**, and a **string**
- The number is called **status code**. Status codes are divided into five categories:
  - 1\*\*: Informational Response
  - 2\*\*: Success
  - 3\*\*: Location change
  - 4\*\*: Client Error
  - 5\*\*: Server Error

# HTTP Overview – Status code

42

- Some common status codes are
  - 200: The request was successful
  - 400: The request was malformed
  - 404: The requested resource could not be found
  - 500: The server had a critical error, and could not complete the request

# HTTP Overview - Cookies

43

- In order to make HTTP stateful, **cookies** were introduced
- Cookies are **text information** that a web client receives and stores from a server, and sends back within every request to the host
- They are used mainly for
  - Session management
  - Personalization
  - Tracking



# HTTP Overview - Cookies

44

- HTTP servers can set cookies with the response header field **Set-Cookie**
- Cookies can also be set client-side via JavaScript
- Cookies are composed by a name, a value, and some meta-information
  - The origin (e.g., the server which sends the cookie)
  - The expire date
  - Some security policies

# HTTP Overview - Cookies

45

## Response Headers

Access-Control-Allow-Credentials: **true**

Access-Control-Allow-Headers: **X-Requested-With, Content-Type, X-Codingpedia**

Access-Control-Allow-Methods: **GET, POST, DELETE, PUT**

Access-Control-Allow-Origin: **\***

Content-Length: **65**

Content-Type: **application/json**

Date: **Tue, 23 May 2017 08:44:06 GMT**

Server: **GlassFish Server Open Source Edition 4.1**

Set-Cookie: **token=y9cHZlrjGqSlipT;Version=1;Comment=;Domain=;Path=/;Max-Age=3600;Expires=Tue, 23 May 2017 09:44:06 GMT**

X-Powered-By: **Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.1 Java/Oracle Corporation/1.8)**

- The expire date
- Some security policies

# HTTP Overview - Cookies

46

- Browsers will send back cookies to the server in accordance with its scope
- The scope is the "origin" in which each cookie was created
  - If a cookie named "foo" is set by "www.google.com", it cannot be sent to "www.microsoft.com", but only to "www.google.com"

# HTTP Overview - Cookies

47

- In addition to the origin, other security policies can be set
  - Secure and HTTPOnly
  - SameSite
    - None
    - Strict
    - Lax <-- The default on Chrome

# Outline

48

- HTTP History
- Key Features and Overview of HTTP
- **Security and Web Security**
- Tooling



# Security

49

- As the name suggests, security is about the protection of "something"
- When dealing with computers, we normally identify this "something" with **information**
- Security wants to ensure three main properties of information
  - Confidentiality
  - Integrity
  - Availability

# Security

50

## ➤ Integrity

- Maintaining the accuracy and completeness of data

## ➤ Confidentiality

- Data must be accessible only to whom is authorized to

## ➤ Availability

- Data must be accessible when needed

# Security

51

- A vulnerability is a **weakness in a system** that permits an attacker to violate one or more of the three previous properties
- Every vulnerability must have an **impact**
  - How this vulnerability of this system violates one or more of the three principles?

# Security

52

- The best way to find how a vulnerability impact a system is to assume the point of view of an attacker
- This is done testing the application in an offensive maner
- This activity is called **penetration testing**

# Web Security

53

- Web security applies to vulnerabilities that affect web applications.
- Typically, web applications are the most exposed assets to an attacker
- And http is really fragile..

# Web Security

54

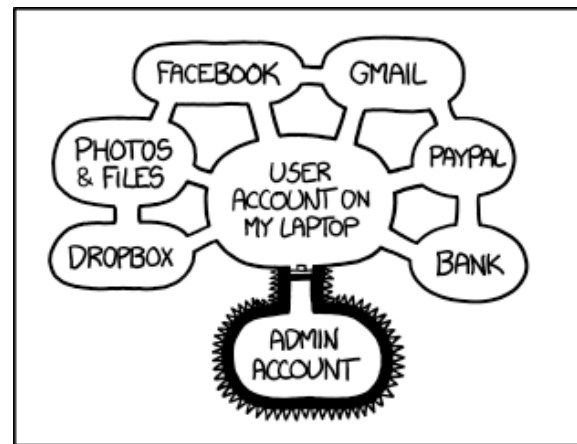
- HTTP was created with the intent to serve **static documents**
- The protocol is **simple by design**
  - It is a **stateless** protocol, since there was no need to keep track of the current client
  - **Documents were simple**, there was no need for animations
  - The **security was not a big concern**, at the beginning there was not much to protect on the web

# Web Security

55

- But now we have
  - **Dynamic generated pages**, e.g. scripts that generate pages on-the-fly
  - Exceptionally **complex pages**: HTML CSS, JavaScript, WebAsm, plugins... and a lot more
  - A lot of **secrets to protect**
- Such complexity leads to a **huge attack surface**

The web is a mess...



IF SOMEONE STEALS MY LAPTOP WHILE I'M LOGGED IN, THEY CAN READ MY EMAIL, TAKE MY MONEY, AND IMPERSONATE ME TO MY FRIENDS, BUT AT LEAST THEY CAN'T INSTALL DRIVERS WITHOUT MY PERMISSION.

# Web Security

56

- Web security is about the security of web assets, e.g. everything that runs over HTTP
  - **Server-Side Security:** The impact affects the remote server
  - **Client-Side Security:** The impact affects the client
    - Note: This does not mean that it is a vulnerability of the browser!
  - Rule of thumb: *If you need to send a link to the victim, then probably it is a client-side vulnerability*



# Web Security

57

- In order to find vulnerabilities, there are two main methodologies
  - **Blackbox**
    - We **do not know anything** about the system we are attacking
  - **Whitebox**
    - We **know everything** about the system, we have the source code, we can debug it, ...

# Web Security – BlackBox

58

- **Enumeration:** the **more information** we have about the system **the better**
  - Look at the functions an application implements
  - Try to input random things. If you have to insert a number, try to insert some letters, and look at what happens
- **Try and error:** because we do not know anything about the system, we **need to try attacks in order to uncover problems**

# Web Security – WhiteBox

59

- When testing on a WhiteBox environment there is much more a tester can do:
  - A **Static Analysis** of the code
  - A **Dynamic analysis** of the code

# Web Security – WhiteBox

60

- In a WhiteBox environment a tester can discover deeper issues than within a BlackBox environment
- In this way, the tester has an advantage over an attacker, because it can discover more flaws in less time and with less skills
- Because of this, WhiteBox testing is **more effective** than a BlackBox testing

# Outline

61

- HTTP History
- Key Features and Overview of HTTP
- Security and Web Security
- Tooling

# Web Security – Some useful tools

62

- Browser
- Curl/wget
  - A Command line utility to make http requests
- Python requests
  - A useful python library to do http requests
- Burp suite/zap proxy
  - live edit raw http requests and response
- Test server (php dev & httpsimplepython)
- Ngrok
  - creates a public http/tcp tunnel to your machine

# On-the-fly HTTP server

63

## ➤ PHP

- A very fast-to deploy test server
- Serves every file inside the directory it was launched from and executes .php scripts
- `php -S 127.0.0.1:5000`

Launch it from a test directory! You don't want to leak your .ssh directory !

# ngrok

64

- What if you need a public server?
  - VPS
  - Ngrok: <https://ngrok.com/>



# ngrok

65

- Ngrok allows one to create tunnels
- You can run it with the command
  - `$ ngrok http 5000`
    - Create a http tunnel and redirect every request to the local port 5000
  - `$ ngrok tcp 5000`
    - Create a tcp tunnel and redirect every connection to the local port 5000

ngrok by @inconsreveable

(Ctrl+C to quit)

Session Status online  
Account bonaff (Plan: Free)  
Update update available (version 2.3.35, Ctrl-U to update)  
Version 2.3.34  
Region United States (us)  
Web Interface http://127.0.0.1:4040  
Forwarding http://1da23a85.ngrok.io -> http://localhost:5000  
Forwarding https://1da23a85.ngrok.io -> http://localhost:5000

Connections	t1	opn	rt1	rt5	p50	p90
	0	0	0.00	0.00	0.00	0.00

Filter by

All Requests

Clear

GET / 200 OK 1.34ms

half a minute ago Duration 1.34ms IP 109.115.64.3

GET /

Summary

Headers

Raw

Binary

Replay

Headers

Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*; q=0.8
Accept-Encoding	gzip, deflate
Accept-Language	en-US,en;q=0.5
Dnt	1
Host	85804868.ngrok.io
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (X11; Linux x86_64; rv:73.0) Gecko/20100101 Firefox/73.0
X-Forwarded-For	109.115.64.3

200 OK

Summary

Headers

Raw

Binary

Ask a question

# HTTP Protocol And Web Security Overview



CYBER  
CHALLENGE.IT



CYBERSECURITY  
NATIONAL  
LABORATORY