



DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES

Bevezetés a Web biztonságba

VIHIBB01 – Kódolás és IT biztonság (2020)

Gazdag András

CrySyS Lab, BME
agazdag@crysys.hu



M Ű E G Y E T E M 1 7 8 2

Contents

- **Web biztonság**
 - Általános problémák
 - Szerver oldali támadások
 - Kliens oldali támadások





WEBES RENDSZEREK FELÉPÍTÉSE

Egy weboldal betöltése

1. Megadott URL-ről tartalom letöltése (Universal Resource Locator)
 - a) Domin név IP címre fordítása
 - b) TCP kapcsolat felépítése az IP címre a 443-as porton (régében 80-as)
 - c) Egy HTTP GET kérés küldése a megadott elérési útvonallal
 - d) HTML tartalom fogadása a HTTP kapcsolaton keresztül
2. Eredmény: HTML (Hyper Text Markup Language) formátumú adat
3. HTML értelmezése
 - a) Újabb kérések küldése a HTML-ben talált linkekre
 - Képek
 - Stíluslapok
 - Szkriptek
 - Frémek
 - b) Az értelmező által talált Javascript lefuttatása
4. Tartalom megjelenítése
5. Az event loop elindítása, Javascript események kezelése

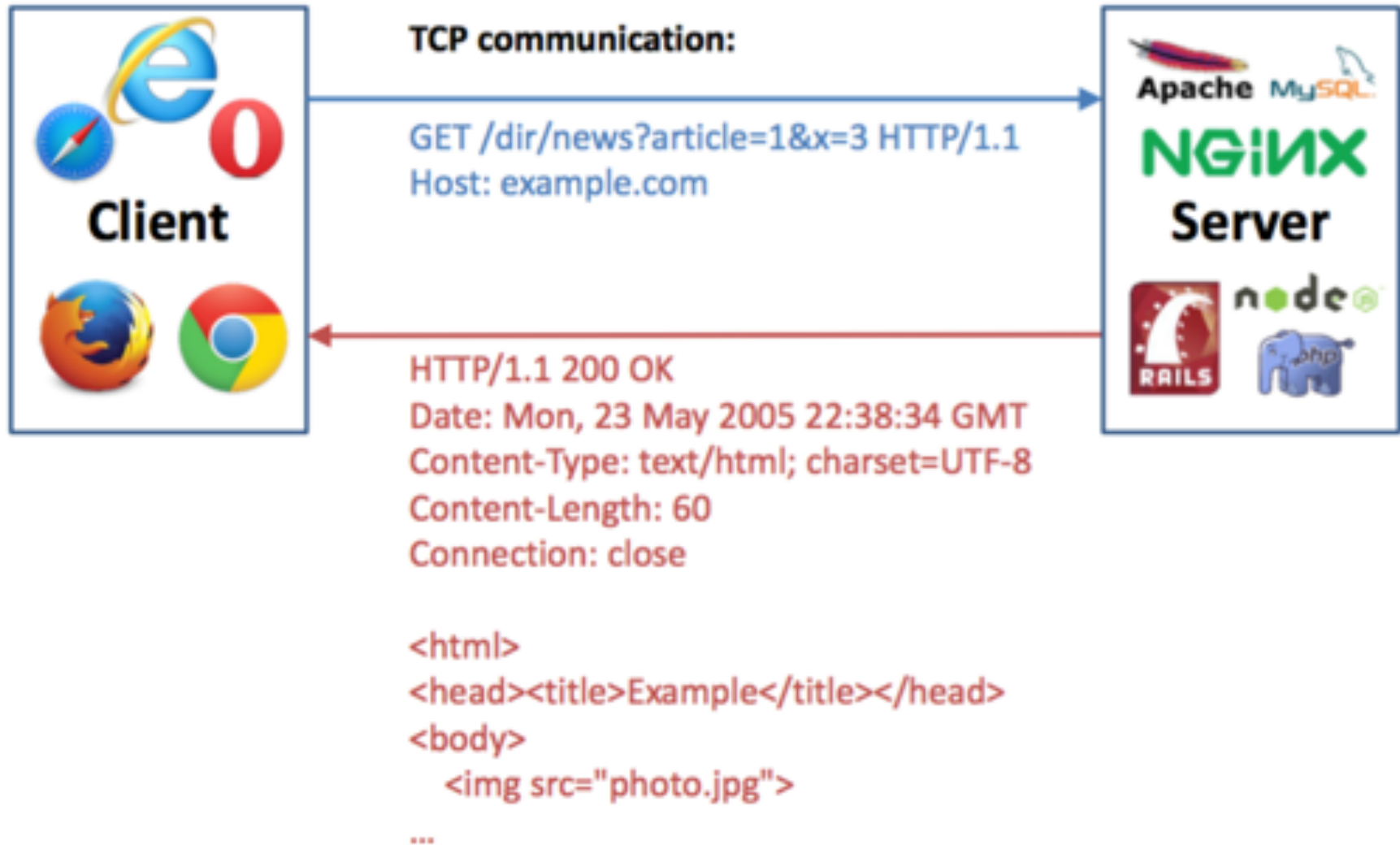
Universal Resource Locator (URL)

URI = `scheme:[//authority]path[?query][#fragment]`
 `authority = [userinfo@]host[:port]`

`https://www.crysys.hu/education/?q=bprof#top`

1. `https:` – séma
2. `//www.crysys.hu` – hoszt
3. `/education/` – elérési útvonal
4. `?q=bprof` – lekérdezés paramétere
5. `#top` – fragmens azonosító

HyperText Transfer Protocol (HTTP)

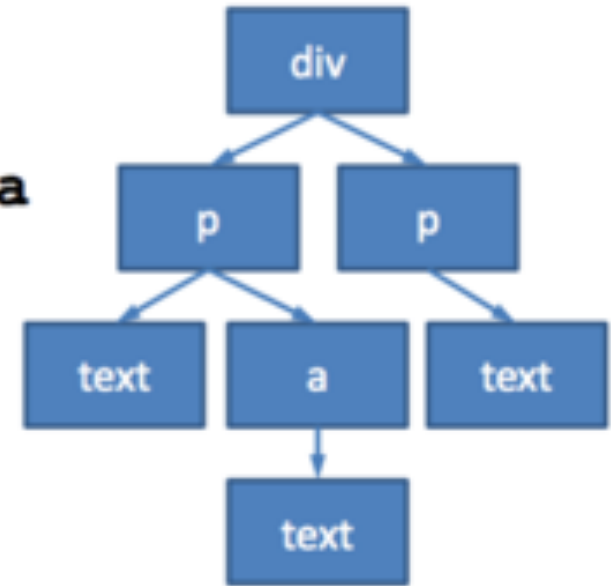


HyperText Markup Language (HTML)

```
<html>                                     Parsing, processing ↓
<head>
  <title>Example</title>
  <link href="/style.css" rel="stylesheet"
      type="text/css">                     HTTP GET /style.css
</head>
<body>
                     HTTP GET /photo.jpg
  <script src="/code.js">
</script>                                   HTTP GET /code.js
  <iframe src="/comments.html" id="comments">
</iframe>                                   HTTP GET /comment.html
</body>
</html>
```

Document Object Model (DOM)

```
<div>
  <p>
    This is a paragraph with a
    <a>link</a>
  </p>
  <p>second paragraph</p>
</div>
```

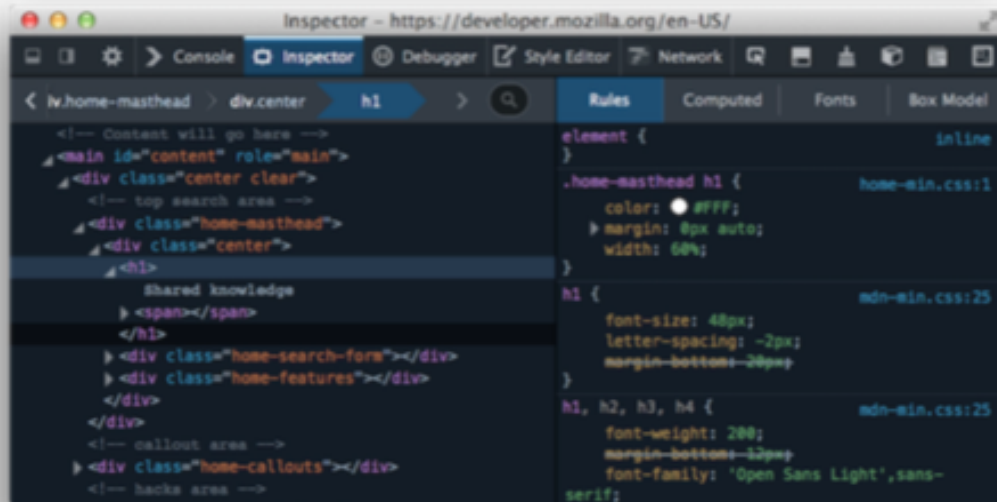


- DOM
 - A HTML tartalom memóriabeli reprezentációja
 - Fa struktúra készül belőle
 - Javascript API-n keresztül elérhető és manipulálható

Developer Tools Böngészőkben

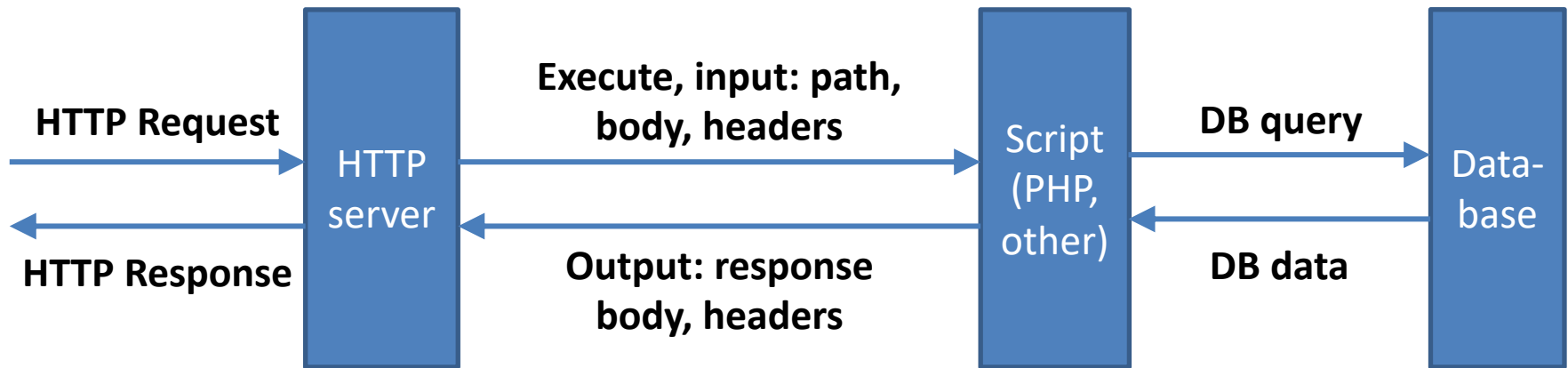
- Minden elterjedt böngésző rendelkezik egy beépített fejlesztői eszköztárral
 - Chrome DevTools
 - Safari Web Inspector
 - Firefox Developer Tools
 - ...

- Elemezni és debuggolni lehet
 - HTTP kéréseket
 - DOM-ot
 - JavaScript-et

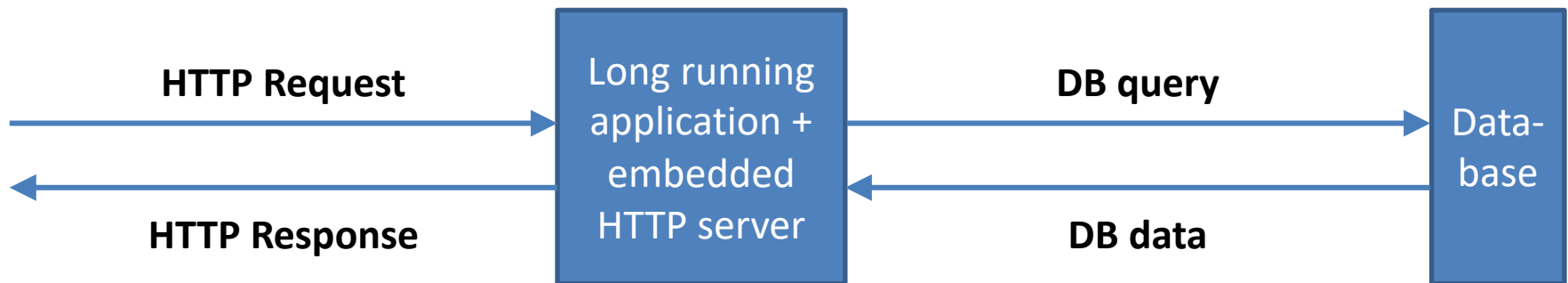


Szerver oldali architektúra

- Klasszikus architektúra (pl: PHP)

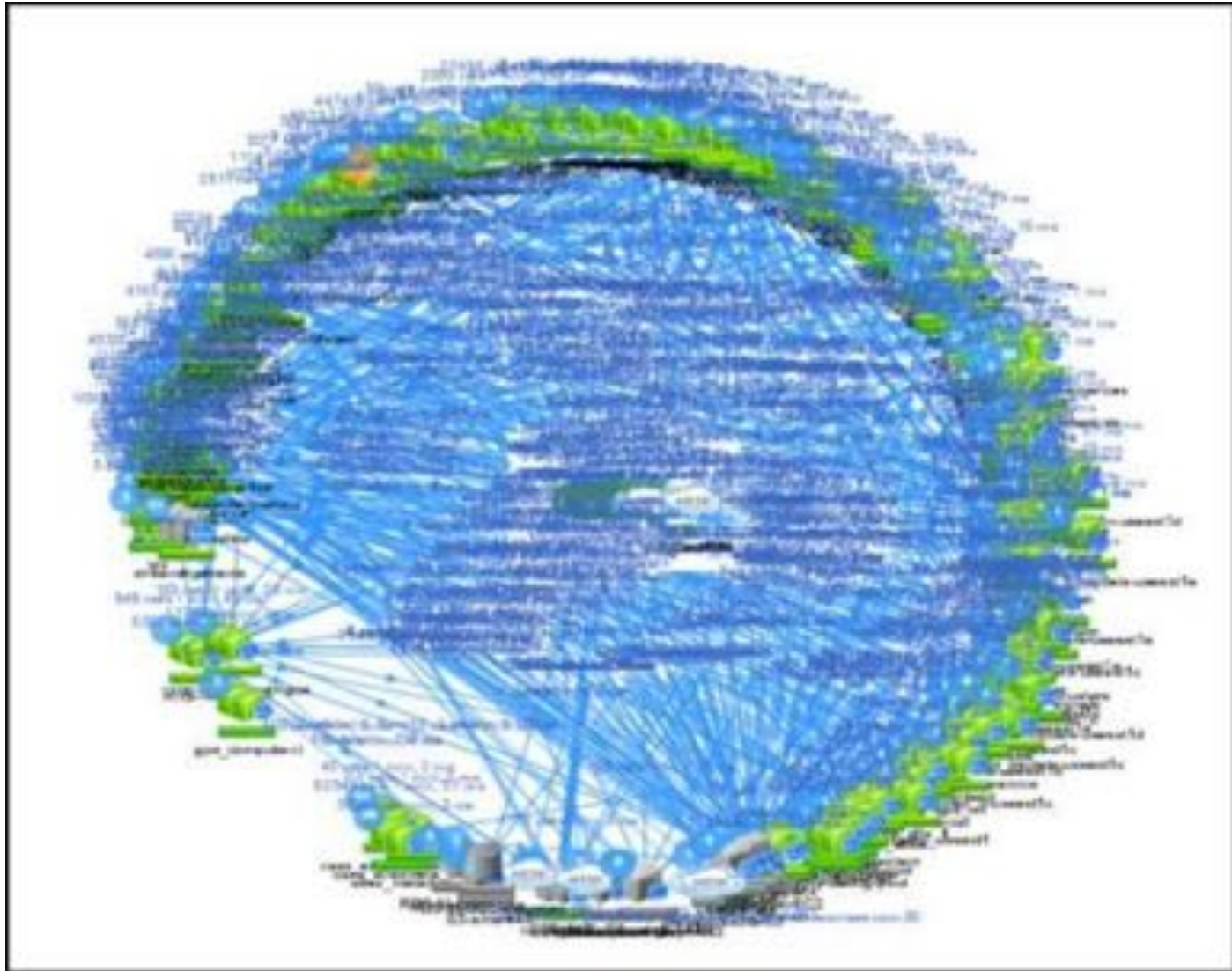


- Esemény alapú architektúra (Python Tornado, Node.js, stb):



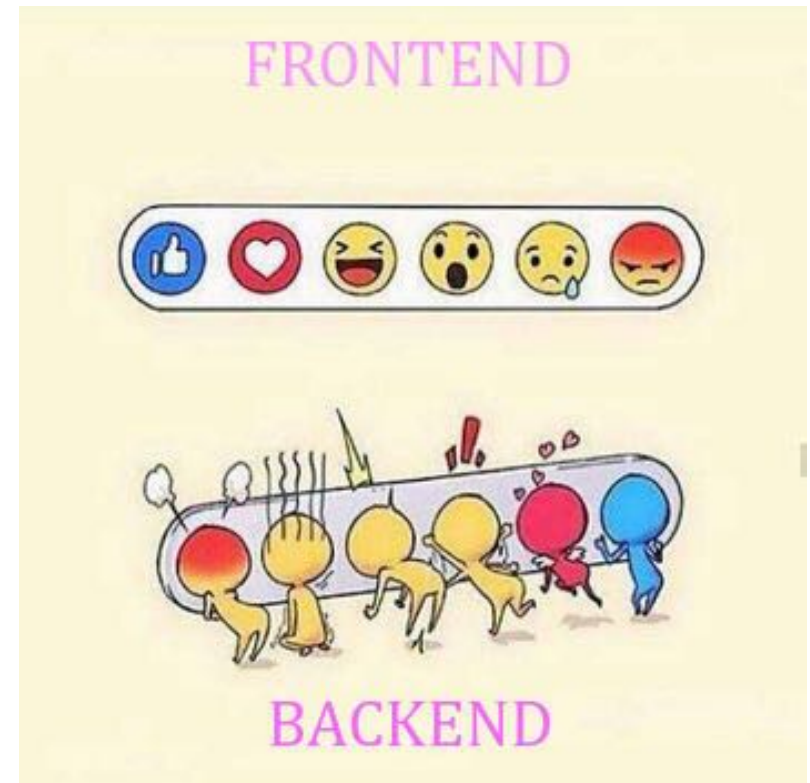
- + terhelés elosztók, elosztott adatbázisok, CDN, stb.

Netflix Architektúra...

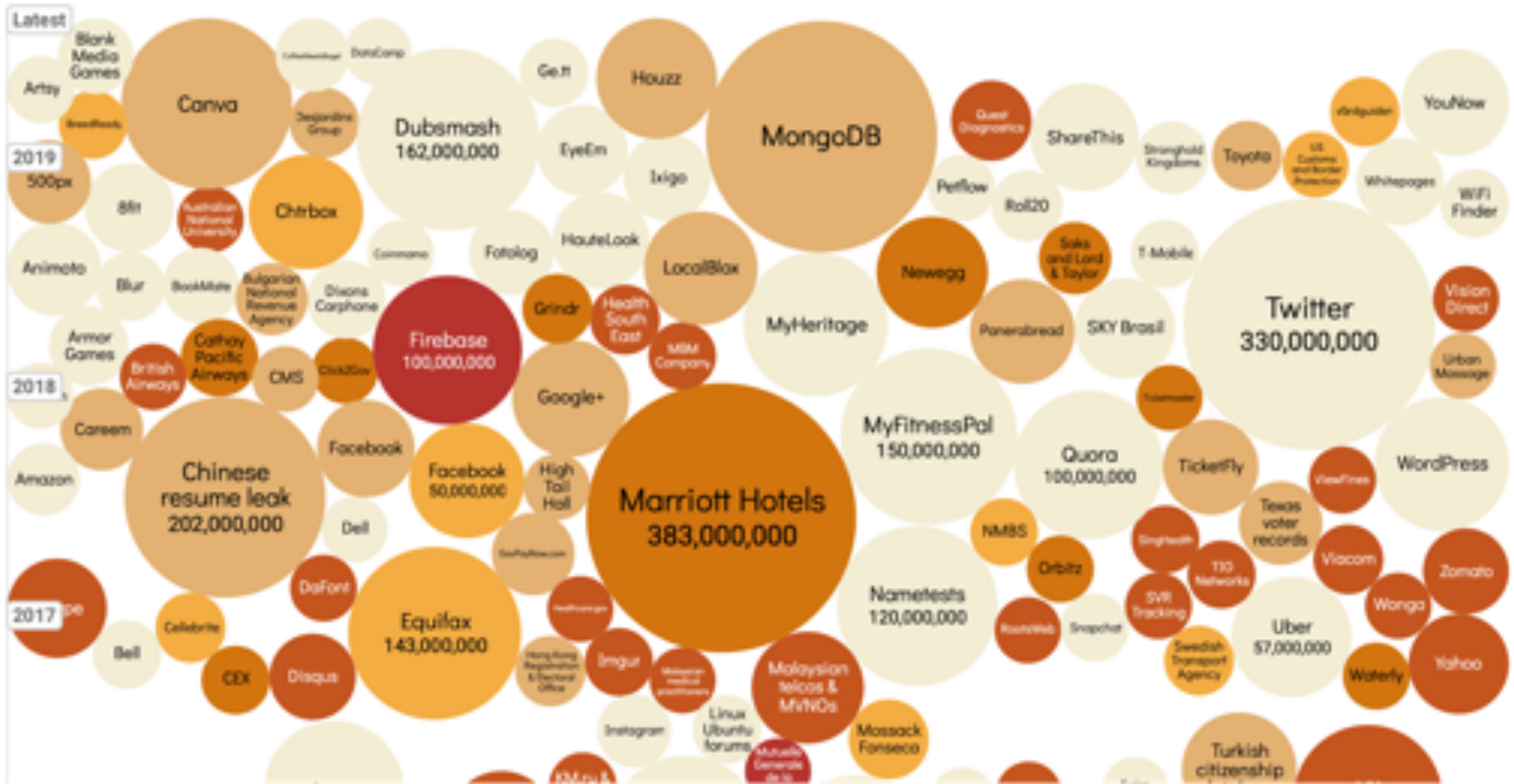


Web biztonság problémái

- Biztonságos kommunikáció a böngésző és a szerver között
 - Hitelesítés és biztonságos csatorna (→ TLS)
 - Session lopás veszélye
- Kliens oldali támadások
 - Cross Site Scripting
 - ...
- Szerver oldali támadások
 - SQL injection
 - ...



Jelentős incidensek



Előadás célja

- Általános biztonsági koncepciók bemutatása
- A leggyakoribb támadások működésének a megértése
 - Cookie lopás, SQL injection, Cross Site Scripting (XSS), stb.
- Védekezési lehetőségek áttekintése és megértése
 - Bemenetek megfelelő szűrése és ellenőrzése
- Nem cél: támadáshoz használt eszközök terjesztése (a bemutatott támadási példák általában nem futtathatók le közvetlenül, csak illusztrációként szolgálnak)



- Open Web Application Security Project (OWASP)
- Nemzetközi non-profit szervezet a web biztonság javítására
- Céljuk, hogy felhívják a figyelmet a webes biztonsági kérdésekre, hogy a cégek és a fejlesztők jó döntéseket tudjanak hozni
- Mindenki részt vehet a munkájukban, és az összes anyaguk ingyenes és jogtiszta elérhető
- Nem javasolnak fizetős megoldásokat, hogy anyagi függetlenségükkel is biztosítsák a pártatlanságukat

OWASP Top 10

- A 10 legjelentősebb veszély egy web alkalmazásra
- A legfrissebb verzió belőle a 2017-es lista

- Ez nem egy feladatlista!
- Csak a prioritások meghatározásában segít!
 - A lista végére érve nem állhatunk meg azzal, hogy most már biztonságos az alkalmazás. Ezek a lista elemek csak a legvalószínűbb veszélyeket jelentik, ezekkel érdemes kezdeni a biztonság növelését, azonban nem teljes a lista, ennél több veszély is felmerül egy weblap fejlesztése és üzemeltetése során
 - Más dokumentumok is hasznosak lehetnek még:
 - » OWASP Developer's Guide
 - » OWASP Cheat Sheet sorozat

- https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP Top 10 - 2017

- A1 – Injection
- A2 – Broken Authentication
- A3 – Sensitive Data Exposure
- A4 – XML External Entities
- A5 – Broken Access Control
- A6 – Security Misconfiguration
- A7 – Cross-Site Scripting (XSS)
- A8 – Insecure Deserialization
- A9 – Using Known Vulnerable Components
- A10 – Insufficient Logging & Monitoring

OWASP Top 10

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

10-nél nincs vége!

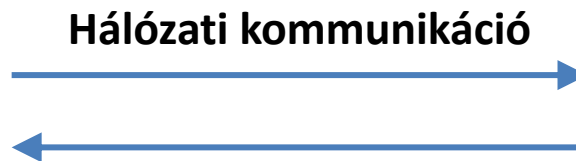
- Több száz további veszély ismert még web alkalmazásokkal kapcsolatban, amelyekre van megoldási javaslat a többi OWASP segédanyagban:
 - <https://github.com/OWASP/DevGuide>
 - https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series
- Ezek ismerete mindenki számára erősen ajánlott, aki web alkalmazást vagy web API-t fejleszt
- A sérülékenységek hatékony felismeréséhez további hasznos információk találhatóak még az OWASP Testing Guide-ban.
 - https://www.owasp.org/index.php/OWASP_Testing_Project



SZERVER OLDAL

Támadó modell

- A támadó a szerverhez csak hálózati kommunikáción keresztül fér hozzá
 - Web alkalmazások esetén ez leginkább HTTP forgalmat jelent
- Támadó célja lehet:
 - **Adat kiolvasása** a szerverről
 - **Adat módosítása** a szerveren
 - Szerver elérhetetlensége a többi felhasználó számára (**Denial of Service**)





SQL INJECTION

SQL

- SQL = Structured Query Language
- Adatbázis lekérdező nyelv relációs adatbázisok számára

ID	NAME	PASSWORD	EMAIL
1	joe	SG2MB45BK	joe@example.com
2	jane	SZ634BT723	jane@example.com
3	john	KJZ245JZ78J	john@example.com
4	jill	FU3489VI76	jill@example.com

```
SELECT * FROM users WHERE name=jill AND password=FU3489VI76
```

- Több dialektusa létezik



SQL injection

- Nagyon gyakori probléma: egy SQL lekérdezése előállítása string műveletekkel **külső** adatok alapján:

```
String userName = $_GET['user'];  
String password = $_GET['password'];  
String query = "SELECT * FROM users WHERE name = '"  
                + userName + "' AND password = '"  
                + password + "'";  
db.execute(query);
```

- Művelet elvárt eredménye:

```
SELECT * FROM users WHERE name=jill AND password=FU3489VI76
```

- Azonban, ha a támadó a bemenet részeként SQL utasítást is megad, az alapjaiban módosíthatja az eredeti lekérdezést.

Például: name ' OR 'a'='a,

```
SELECT * FROM users WHERE name = 'whoever' AND  
                password = 'name ' OR 'a'='a''
```


Tipikus SQL injection támadások

```
SELECT * FROM users WHERE username=<user> AND password=<pwd>
```

- SQL komment string: #, --, /*
 - DBMS függő a támadás

```
username=a' # comments out the rest of the query
```



Tipikus SQL injection támadások

```
SELECT * FROM users WHERE username=<user> AND password=<pwd>
```

- A UNION kulcsszó használata

- Ezzel lehet adathoz hozzáférni, ami másik táblában található
- Az esetben működik csak, ha a két lekérdezés azonos számú oszlopot tartalmaz az eredményben (a támadónak meg kell ismernie a táblák felépítését)

```
username=a' union select 1,2,3,* from users #
```

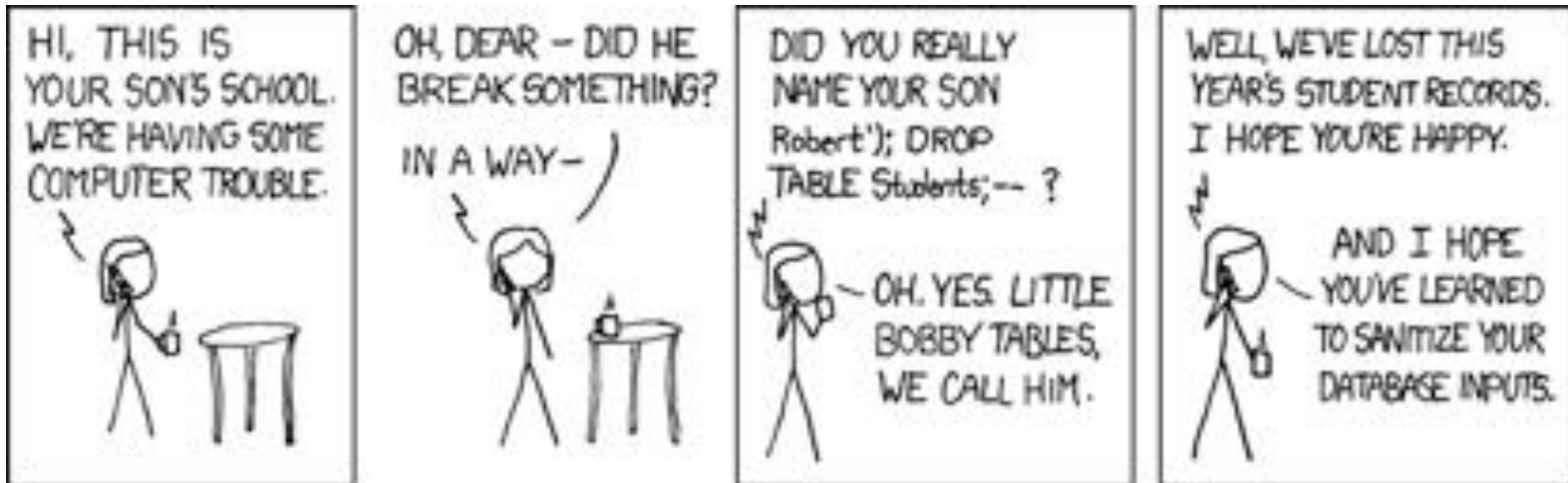


Tipikus SQL Injection támadások

```
SELECT * FROM users WHERE username=<user> AND password=<pwd>
```

- Query stacking segítségével több utasítás összefűzhető
 - ; segítségével el lehet választani több utasítást egymástól
 - DBMS és platform függő ez a támadás

```
username=a'; DELETE * from users #
```



Blind és időzítés alapú SQL injection

- Abban az esetben, ha a lekérdezésnek semmilyen olvasható eredménye nem látszik (hibaüzenet sem), akkor lehet adatszivárgás?
 - **Igen!** Bitenként így is hozzá lehet férni az adatbázis tartalmához, abban az esetben, ha szerver a válaszában felhasználja az adatbázis tartalmát
- Igaz/hamis SQL lekérdezések
 - Ha látszik az oldal viselkedéséből a lekérdezés eredménye, akkor olyan lekérdezéseket kell összeállítani, aminek vagy bármi az eredménye, vagy üres halmaz egy feltételtől függően
 - ' `AND <something_returning_true_or_false> #`
 - Ha nem látszik semmi az eredményből, akkor a feltételtől függően késleltetni kell a lekérdezés eredményét (`sleep` utasítás), amely detektálható
- Ilyen támadásokat általában automatizáljuk (pl: SQLMap)

SQL injection programok



- De facto sztenderd megoldás SQLMap
- Képes automatikusan kihasználni az egyszerűbb hibákat
- Könnyen bővíthető a komplex esetek kezelésére:
 - Python "tamper script"-ek segítségével

```
[11:07:01] [INFO] target URL appears to have 3 columns in query
[11:07:01] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns'
injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [
y/N] N
sqlmap identified the following injection points with a total of 25 HTTP(s) requests
:
---
Place: GET
Parameter: id
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1 AND 3362=3362

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
  Payload: id=1 AND (SELECT 9338 FROM(SELECT COUNT(*),CONCAT(0x3a6976743a,(SELECT
(CASE WHEN (9338=9338) THEN 1 ELSE 0 END)),0x3a766b663a,FLOOR(RAND(0)*2))x FROM INFO
RMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
```

Védekezés

- Kerülni kell a dinamikus SQL lekérdezéseket
 - Amikor csak lehet statikusan megírt lekérdezéseket kell futtatni
 - » Ilyen lekérdezések nem változnak futásidőben, így nem is sérülékenyek
- A bemeneteket szűrni és ellenőrizni kell
 - A speciális karaktereket escape-elni kell, kikényszeríteni a sztenderd név konvenciók használatát, stb.
- Parameterized statement-ek használata
 - Speciális karakterek jelzik a bemenetek helyét
 - A bemenetek speciális függvények segítségével kerülnek behelyettesítésre
- Állítsunk be megfelelő jogosultságokat az adatbázison
 - Pl: a webszerver felhasználónak csak olvasási joga legyen alapból, ha lehet

Parameterized statements

- A legtöbb környezetben létezik implementáció hozzá
- A felhasználói inputok úgy kerülnek behelyettesítésre, hogy a DBMS garantáltan nem fogja azokat a kód részeként értelmezni
- Python példa:

```
connection = mysql.connector.connect(...)
cursor = connection.cursor(prepared=True)

sql_query = """SELECT * FROM Users WHERE
                EmailAddr=%s AND Password=%s"""
cursor.execute(sql_query, (username, password))
```

SQLI HALL-OF-SHAME

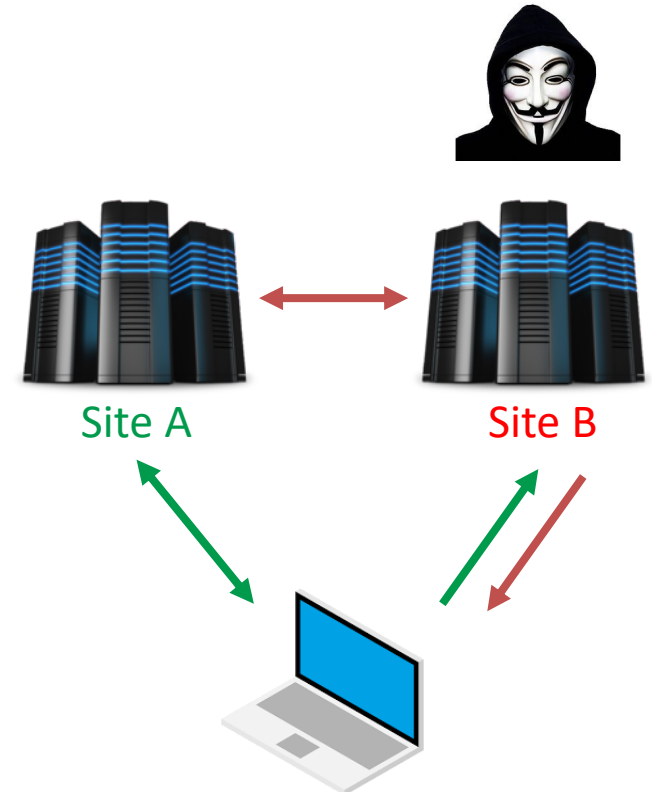
Company	Date	Results	Reference
Cisco	2020-08	Critical vulnerabilities in data center network manager	Cisco Addressed Multiple Bugs In Data Center Network Manager
Wordpress	2020-07	wordpress theme vulnerable to xss and sqli	WordPress NexosReal Estate Theme 1.7 Cross Site Scripting / SQL Injection
Cisco	2020-07	web management interface vulnerable	Cisco SD-WAN vManage Software SQL Injection Vulnerability
GitHub / GitLab via Waydev	2020-07	OAuth tokens stolen and being used in further attacks	Hackers stole GitHub and GitLab OAuth tokens from Git analytics firm Waydev
Joomla	2020-06	possible exposure of redis or proxy credentials	[20200706] - Core - System Information screen could expose redis or proxy credentials
QB8 LLC	2020-06	data wiped via sql injection in corrupt email from HIBP	Email from HavelBeenPwned wipes helpdesk tickets
SAP	2020-06	vulnerabilities (patch available) in Adaptive Server Enterprise (ASE) database product used by thousands of enterprises worldwide	Install latest SAP Adaptive Server Enterprise patches, experts urge
QIWI Payment Service	2020-06	3 vulnerabilities risking user data and system availability	Critical Vulnerability Found In Russian Payment Service QIWI
Rockwell	2020-05	Industrial software can be compromised with malicious EDS files	Hackers Can Target Rockwell Industrial Software With Malicious EDS Files
Sophos	2020-05	Firewall appliances being attacked	THIS WEEK IN SECURITY: FIREWALL 0-DAY, APPLE'S RESPONSE, AND AN ANDROID BLUETOOTH BUG



KLIENS OLDAL

Kliens oldali támadó modell

- A felhasználó be van jelentkezve A oldalon (A originhez tartozik)
 - A bejelentkezés süttikkel történik
 - Az oldal jelenleg nincs megnyitva
- A felhasználó meglátogatja a B oldalt (B originhez tartozik)
- B oldal kártékony kódot tartalmaz
- B oldalnak nem szabad tudnia:
 - Adatot olvasni az A oldalról
 - Módosítani adatot az A oldalon
 - Hozzáférnie az A oldalhoz tartozó süttikhez (megszemélyesítéses támadás)



Kliens oldali biztonsági modell

- **Origin = scheme + domain + port**
- Alapszabály: **origin = biztonság határa**
- Egy oldal hozzáférhet:
 - Más oldalak tartalmához, ha arra explicit jogosultságot kap
 - Más oldalak erőforrásaihoz, ha azt a Same Origin Policy megengedi
 - Helyi erőforrásokhoz, ha azt a felhasználó explicit megengedi
- Ez a modell mellett egy kártékony oldal nem tud kárt okozni, csak ha:
 - A legitim oldal hibát tartalmaz (XSS, CSRF, ...), amit a támadó ki tud használni
 - A böngésző hibát tartalmaz, amit a támadó ki tud használni

Same Origin Policy

Egy weboldal csak a saját originjéhez tartozó erőforrásokhoz férhet hozzá korlátozás nélkül. Más originhez tartozó erőforrásokra a korlátozások vannak.

- A biztonság alapszabálya a kliens oldalon
- A korlátozás függ az erőforrás típusától és a hozzáférés módjától
- Példák
 - Más domainhez tartozó sütiket nem lehet kiolvasni
 - Nem lehet sem olvasni, sem módosítani azokat a tabokat vagy frame-eket, amelyek más domainhez tartozó tartalmat jelenítenek meg
 - HTTP kérések *küldése és a válasz olvasása* csak a saját domainre lehetséges
 - Más domainről származó scripteket, stíluslapokat és képeket be lehet ágyazni, a tartalmukat azonban nem lehet kiolvasni

Same Origin Policy – Alapszabályok

SO = korlátozza a same origin policy (nem lehetséges)

SO = nem korlátozza a same origin policy (lehetséges)

Elv:

- Megjeleníteni vagy beágyazni tartalmat, HTTP kérést küldeni **SO**
- Kiolvasni tartalmat **SO**
- **Képek** megjelenítése **SO**; pixel értékek kiolvasása **SO**
- **Scriptek** beágyazása, végrehajtása **SO**; kód olvasása **SO**
- **Stíluslapok** beágyazása, végrehajtása **SO**; utasítások kiolvasása **SO**
- **Frame-ek és iframe-ek**
 - URL beállítása **SO**; URL kiolvasása **SO**
 - Tartalom kiolvasása **SO**
- **Navigáció** **SO**
- **Formok elküldése (= navigation + POST)** **SO**



JAVASCRIPT BIZTONSÁG

Javascript

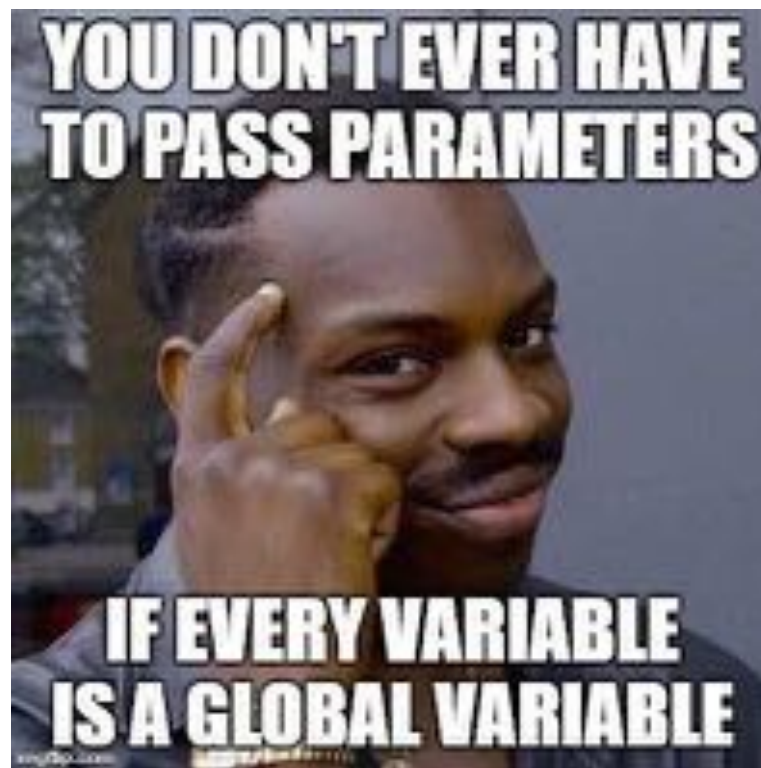
- 10 nap alatt létrehozott nyelv
 - Brendan Eich találta ki 1995-ben (Netscape)
- ECMA standardizálta 1996-1997-ban
 - Aktuális verzió: 11. kiadás (2020. június)
- Stabil technológia sok éve már

JavaScript előfordulásai

- **HTML-ben:** `<script>...</script>`
- **HTML elemként:** ``
- **Távolról:** `<script src="example.com/glob.js"></script>`
- **CSS:** `body{background:url("javascript:alert('XSS')")}`
- **Trükkösen:** ``

JavaScript Global Object

- A JavaScript eredendően egy globális nyelv
 - Változók alapból a globális scope-hoz tartoznak (kivéve, ha függvényen belül lettek definiálva)
 - Függvényeknek globális a scope-ja
 - Objektumok a globális prototípusból örökölnek
- Következmények
 - Minden script, ami egy adott originből származik, hozzáférhet minden más script változóihoz, függvényeihez, objektumaihoz
 - Távolról betöltött scriptek a lokálisokkal egyformán vannak kezelve
 - XSS komoly veszélyt jelent



JavaScript veszélyei

- Kártékony céllal írt JavaScript kód képes lehet a következőkre:
 - Hozzáférhet másik script változóihoz
 - Hozzáférhet másik script függvényeihez
 - Felülírhatja a böngésző natív metódusait
 - Elküldhet adatot bárhova
 - Figyelheti a billentyűleütéseket
 - Kiolvashatja a sütiket
 - A felhasználói kattintás és a JavaScript kattintás egyenértékű

Mi a hiba? – Kliens oldali biztonság

```
function check()
{
  if ( SHA1(document.auth.pass.value) ==
      '43206512b209ba29cb5c642edc85bdac133354fe' )
  {
    window.location.href="admin_page.aspx";
  }
  else
  {
    alert('Invalid password!');
  }
}
```

- A bejelentkezés a kliens oldalon van megvalósítva
- **Kliens oldalra írt kód sosem nyújt biztonságot!**
 - Sosem szabad semmilyen kliens oldalon megvalósított funkcióra támaszkodni biztonság szempontjából
 - A kliens oldali ellenőrzés csökkentheti a szerver terhelését és javíthatja a felhasználói élményt
 - » **De az ellenőrzéseket meg kell ismételni a szerver oldalon!**

Kliens oldali JavaScript biztonsága

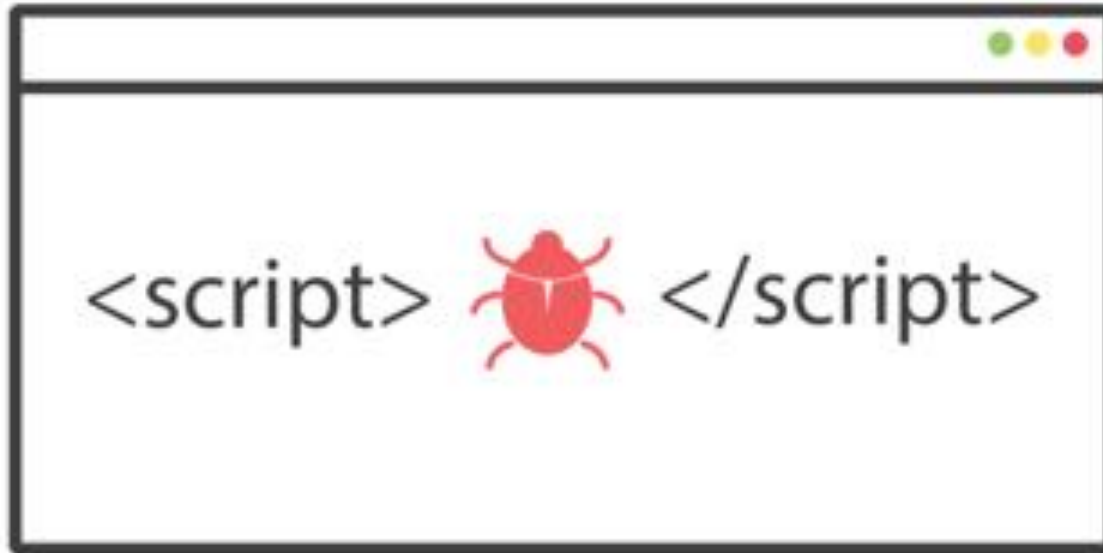
- JavaScript kód a böngészőn belül kerül értelmezésre és végrehajtásra a kliens oldalon
- Ismeretlen kód futtatásából származó veszélyek csökkentése céljából a böngészők különböző korlátozásokat alkalmaznak
 - Minden kód sandboxban kerül végrehajtásra, hogy ne tudjon kárt okozni a helyi számítógépen
 - Minden kód végrehajtását korlátozza a Same Origin policy, hogy ne tudjon kárt okozni egy másik weboldalon

JavaScript kód védelme

- Egyre több funkcionalitás kerül a kliens oldali alkalmazásokba
 - Egyre jelentősebb értéket képviselnek -> a kód védelme fontossá vált
- Megoldás: **obfuszkáció**
 - Változók, függvények, stb. átnevezése
 - Kód struktúrájának a módosítása (pl: utasítás sorrend)
 - Adat transzformációk (pl: [[]] → [], szöveg átkódolás)
- Natív kód estén a cél az automatikus kód visszafejtés megnehezítése
- JavaScript: a kód értelmezése nehezebb az ember számára
- Az obfuszkáció nem teszi biztonságossá a kódot! Ettől nem tűnt el sérülékenység!
 - **Security by Obscurity**

JavaScript kód védelme

```
<script>
eval(
(![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(![]+[])[+!+[]]+(![]+[])[
+[]]+(![]+[])[(![]+[])[+[]]+(![]+[])[[]])[+!+[]+[]+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+[]]+
(![]+[])[!+[]+!+[]+!+[]]+(![]+[])[+!+[]])[!+[]+!+[]+[]+[]]+[+!+[]]+(![]+[])[(![]+[])[
+[]]+(![]+[])[[]])[+!+[]+[]+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+[]]+(![]+[])[!+[]+!+[]+!
+[]]+(![]+[])[+!+[]])[!+[]+!+[]+[]+[]]
)
</script>
```



CROSS-SITE SCRIPTING (XSS)

Cross Site Scripting

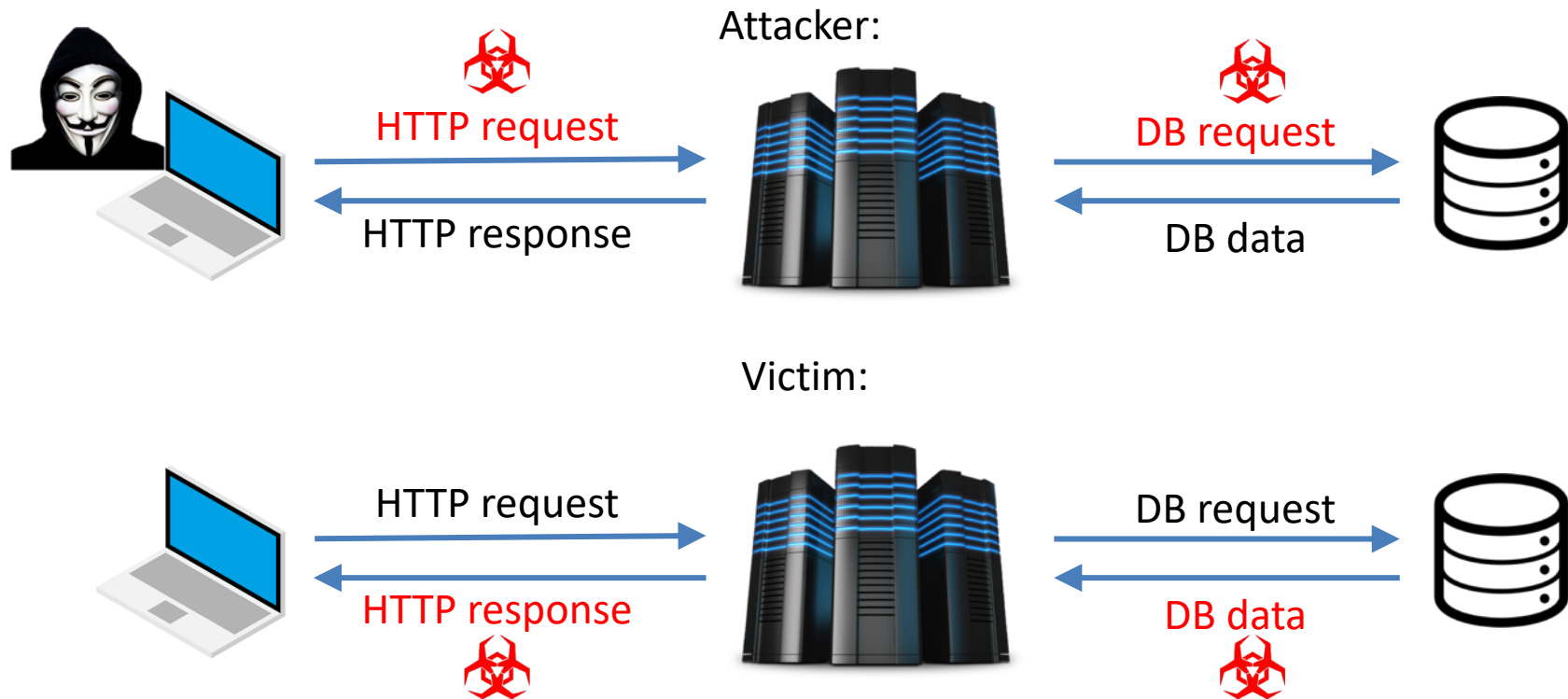
Cross Site Scripting (XSS): a támadó képes JavaScript kódot futtatni egy másik originen belül

- A legveszélyesebb kliens oldali támadás
- A kártékony kód lényegében bármit megtehet a támadás során
- Több különböző altípusát is megkülönböztetjük a támadásnak, az alapján, hogy:
 - A kártékony kód tárolására kerül-e a szerver oldalon
 - A HTML tartalom összeállítása hol történik meg

Cross Site Scripting típusai

1. Persistent/Stored XSS

- A kártékony kód a szerver oldalon eltárolásra kerül
- Például: kommentben, üzenetben, felhasználói adatban, stb.
- Trigger: az áldozat meglátogatja a fertőzött oldalt



Cross Site Scripting típusai

2. Reflected XSS

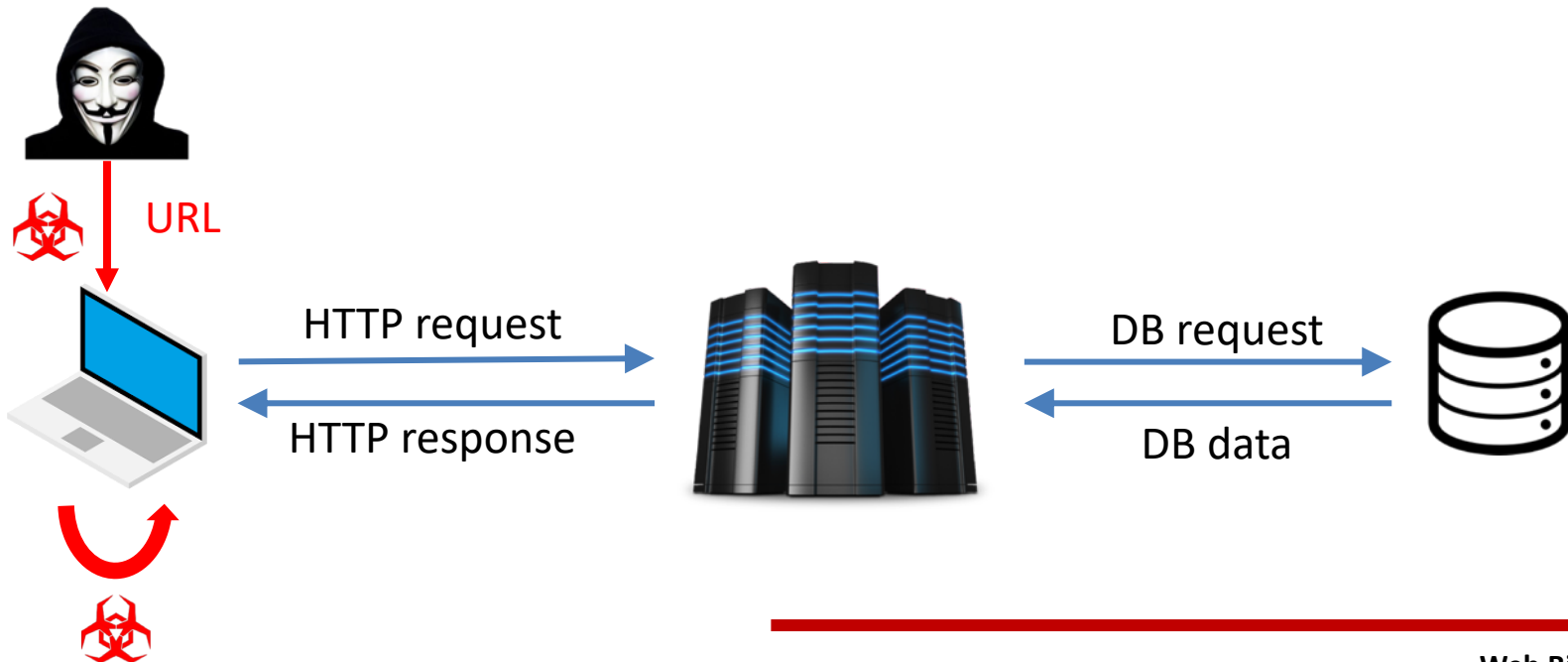
- A kártékony JS egy GET/POST paraméteren át fertőz
- A szerver oldal felhasználja a paraméter értékét a HTML előállításánál
- Trigger: a felhasználó meglátogatja a sérülékeny oldalt a kártékony URL-en keresztül



Cross Site Scripting típusai

3. DOM alapú XSS

- A támadás tisztán a kliens oldalon történik
- HTML összeállítása a kliens oldalon van megvalósítva (pl: SPA)
 - » `x.innerHTML = attacker_controlled_variable;`
- Trigger: a felhasználó meglátogatja a sérülékeny oldalt a kártékony URL-en keresztül



Cross Site Scripting - megelőzés

- **Kívülről származó adatot ellenőrizni kell mielőtt az a HTML-be kerül!**
- **A kontextus fontos!** Különböző ellenőrzéseket kell végegni!
 - `<p><?php echo $user_comment; ?></p>`
 - `" >`
 - `<script>n = "<?php echo $user_name; ?>";</script>`
- **Tiltó lista alapú megközelítés vagy egyszerű törlés nem elég!**
 - Törlés esetén: `<scr<script>ipt>` → `<script>`
 - Egy tiltó lista **sosem** teljes!
- **Megoldás HTML elemek és tulajdonságok esetén:**
 - HTML entity encoding
 - PHP: `htmlspecialchars()`
- [OWASP XSS Prevention Cheat Sheet](#)

Cross Site Scripting - védekezés

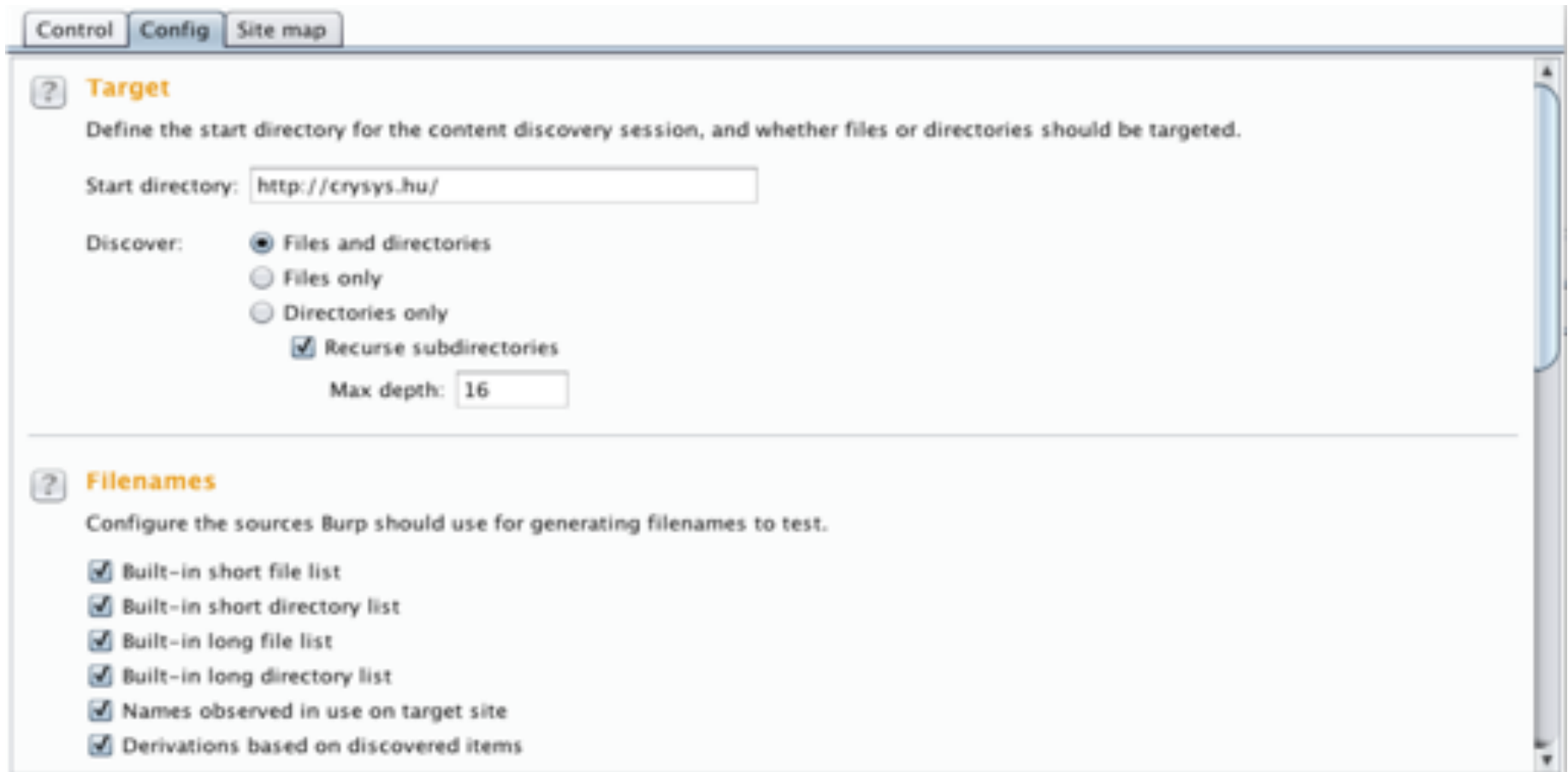
- Hogyan tegyük az XSS támadást nehézé/lehetetlenné?
- **HTTP-only sütik**
 - Egy tulajdonsága a sütiknek
 - Nem lehet JS kódból hozzáférni → session lopás lehetetlen XSS-en át
- **Content Security Policy (CSP)**
 - HTTP fejléc
 - Megadja, hogy honnan tölthet be tartalmat a böngésző
 - Egy megadott URL-re jelzi, ha szabálysértés történik
 - Kiindulási elv:
 - » Ne engedjük az inline script lehetőséget (sokszor nehéz megvalósítani)
 - » Ne engedjük az `eval()` függvény használatát
 - További ötletek:
 - » Csak megadott domainről engedjük a scriptek, képek, objektumok betöltését
 - » Adjuk meg, hogy mely oldal ágyazhatja be frame-be a mi oldalunk



WEB SECURITY SCANNERS

Web Application Security Proxies

- Fízetős: [Burp Suite](#), open source: [Zed Attack Proxy](#)
- HTTP traffic inspection, web spider, **content discovery module**



Bug Bounty Programok

- A böngésző fejlesztők biztonsági stratégiájának a része
- **Cél: a sérülékenységeket a gyártónak adják el és ne a feketepiacon**
- Komolyabb összegeket is lehet keresni
 - [Mozilla](#): ~15000 USD
 - [Chrome](#): <= ~30000 USD
- [Pwn2own](#)
 - Évente megrendezett esemény a CanSecWest konferencián
 - Teljes támadást kell bemutatni
 - Nagyon magas díjakkal
- [Internet Bug Bounty](#)
 - Általános webhez tartozó sérülékenységek, sandbox escape, stb.

További források

- <https://venturebeat.com/2018/03/05/wordpress-now-powers-30-of-websites>
- <https://developers.slashdot.org/story/17/10/29/0441205/why-do-web-developers-keep-making-the-same-mistakes>
- <https://www.hpe.com/us/en/insights/articles/the-owasp-top-10-is-killing-me-and-killing-you-1710.html>
- <https://codecurmudgeon.com/wp/sql-injection-hall-of-shame>
- <https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html>
- [OWASP XSS Prevention Cheat Sheet: https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
- [Content Security Policy: http://www.w3.org/TR/CSP/](http://www.w3.org/TR/CSP/)
- [Internet Bug Bounty: https://hackerone.com/internet-bug-bounty](https://hackerone.com/internet-bug-bounty)

Ellenőrző kérdések

- Hogyan épül fel egy URL?
- Mik a leggyakoribb web biztonsági problémák?
- Mi a célja az OWASP projektnek?
- Általában mi az első lépése egy támadónak egy szerver ellen?
- Mi az általános probléma injection típusú támadásoknál?
- Mutasson egy példát egy SQL injection támadásra!
- Hogyan működik egy blind SQL injection?
- Hogyan működik egy időzítés alapú SQL injection?
- Hogyan kell védekezni egy SQL injection ellen?

Ellenőrző kérdések

- A kliens oldalon hogy definiáljuk a biztonsági határt?
- Mi az a Same Origin Policy?
- Mit limitál a Same Origin Policy egy XMLHttpRequest elküldésekor?
- Hol jelenhet meg JavaScript kód?
- Mit tud megtenni egy kártékony JavaScript kód?
- Mi az a XSS támadás?
- Milyen típusai vannak az XSS támadásnak?
- Hogyan lehet védekezni XSS támadással szemben?
- Mi az a Content Security Policy?