

Biztonságos webalkalmazások fejlesztése

Kovács Ferenc (Tyrael)

Escalion Hungary Kft.

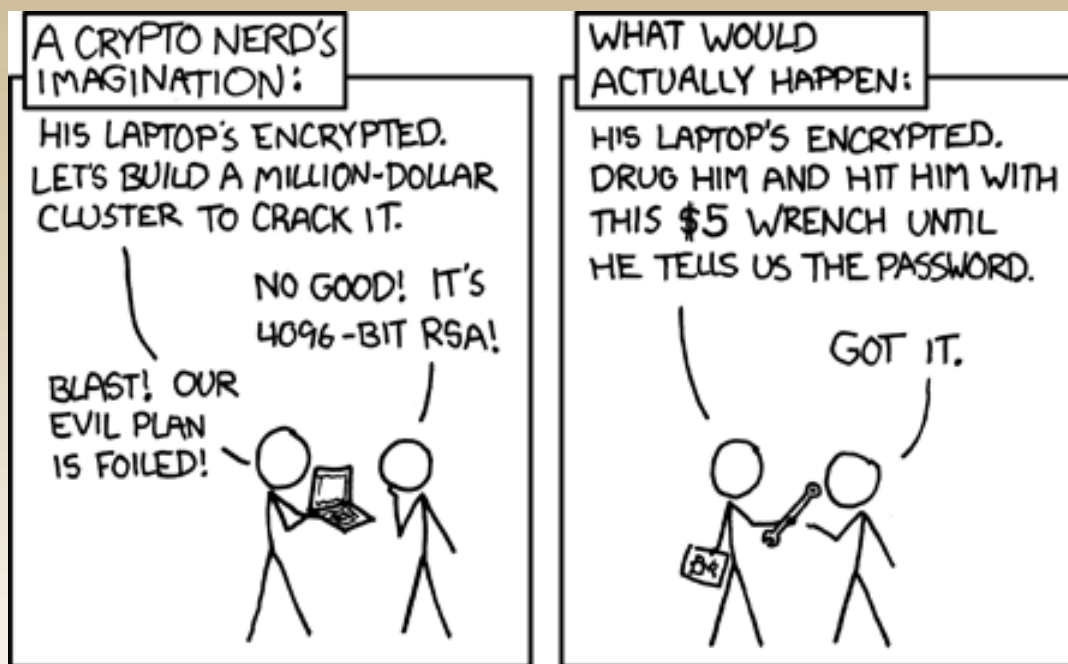
Fejlesztési vezető

1. Mi az a biztonság?
2. Miért fontos a biztonság?
3. Mit kell megvédeni?
4. Kitől kell megvédeni?
5. Hogyan védekezzünk?

1. Mi az a biztonság?

- Az informatikai biztonság céljai tartalmazzák többek között az adatok és eszközök védelmét az illetéktelen hozzáféréstől (Confidentiality), az engedély nélküli módosítástól (Integrity), és biztosítja a rendszer zavartalan működését (Availability). (CIA model)
- A biztonság egy folyamat, nem pedig egy állapot. — Bruce Schneier, *Secrets and Lies*
- A biztonság egy lánc; csak annyira erős, mint a leggyengébb láncszem. — Bruce Schneier, *Secrets and Lies*
- Nagyon sokszor az ember jelenti a leggyengébb pontot a láncban és közismerten felelős a biztonsági rendszerek hibáiért. — Bruce Schneier, *Secrets and Lies*

1. Mi az a biztonság?



2. Miért fontos a biztonság?

- Ha az alkalmazottak nem tudják, vagy nem értik, hogy hogyan kell a bizalmas adatokat kezelni, biztonságba helyezni, akkor nemcsak azt kockáztatod, hogy az egyik legfontosabb üzleti értéked (az információ) nem megfelelő kezekbe kerülhet, de azt is, hogy megsérted az egyre növekvő számú adatkezelési szabály és előírás valamelyikét. Továbbá egy másik nagyon fontos üzleti értéked kockáztatod: a céged jó hírnevét. — Rebecca Herold, "Managing an Information Security and Privacy Awareness and Training Program" 2005

2. Miért fontos a biztonság?

- Mert nem jó, ha ellopják a cég pénzügyi nyilvántartását.
- Mert nem jó, ha ellopják a felhasználók szenzitív adatait.
- Mert nem jó, ha elérhetetlenné teszik a cég szolgáltatásait.
- Mert nem jó, ha a cég infrastruktúráját használják ugrópontként más informatikai támadások végrehajtásához (spam, ddos, malware hosting, etc.).
- Mert nem jó, ha ellopják a cég valamely termékének forráskódját.
- Mert nem jó, ha deface-elik a cég weboldalát.
- Mert nem jó, ha megsemmisítik a cég vásárlói adatbázisát.
- Mert nem jó, ha nyilvánosságra kerül, hogy a cég nem tudja megvédeni a informatikai infrastruktúráját.

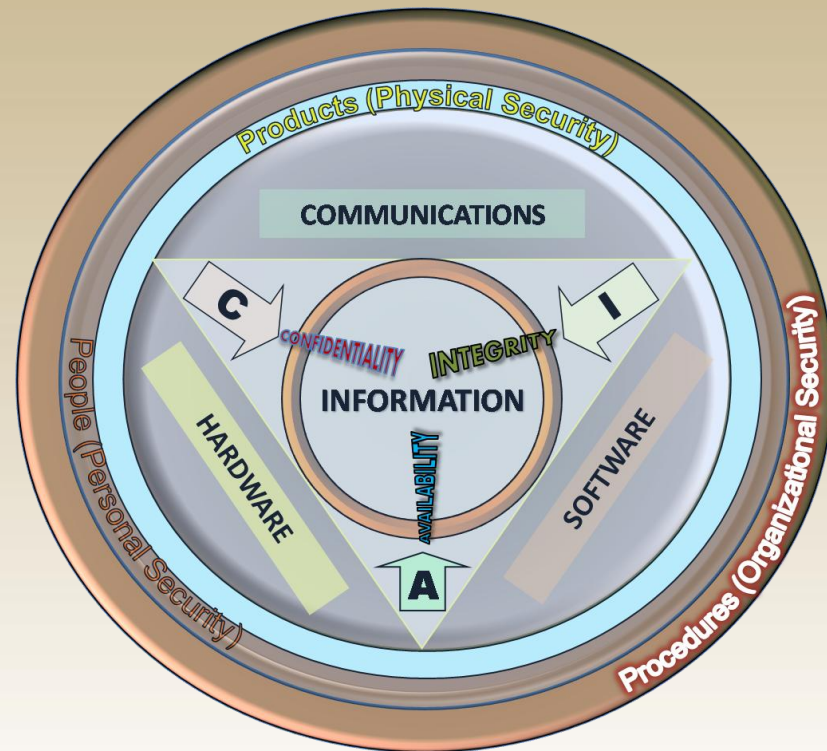
3. Mit kell megvédeni?

- Mindent -> Rossz válasz.
- Nem minden egyformán fontos.
- Kell egy tételes lista a legfontosabb értékekről(Assets), és az ezekre leselkedő veszélyekről(Threats), valamint a potenciális sebezhetőségekről(Vulnerabilities).
- Fel kell mérni, hogy az adott értékeket milyen kontrollokkal védjük, ezek mennyire sebezhetőek, mennyire valószínű, hogy ezen értékeket támadás éri.
- Fel kell mérni, hogy az egyes értékek mennyire fontosak: a sikeres támadás, mekkora mértékű kárt okoz(Impact).
- A kockázat(Risk) megadható az ismert fenyegetések, a sebezhetőségek, valamint az okozott kár szorzataként.

3. Mit kell megvédeni?

- Ez gyakorlatilag a kockázatelemzés/kockázatkezelés témaköre.
- A kiszámolt kockázat alapján lehet a szükséges lépéseket megállapítani, a fenyegetések ellen kontrollokat létrehozni, csökkentve ezzel a három tényezőn keresztül a kockázatot.
- Az okozott kár megállapításánál három szempontot szokás vizsgálni: bizalmasság(Confidentiality), sértetlenség(Integrity), elérhetőség/rendelkezésre állás (Availability), ez az úgynevezett CIA modell.
- Ha ellopták a forráskódot, akkor sérült a bizalmasság, ha elrejtettek benne egy backdoort, akkor a sértetlenség, ha pedig letörölték a produkciós környezetből, akkor pedig az rendelkezésre állás szenvedett csorbát.

3. Mit kell megvédeni?



3. Mit kell megvédeni?

Asset Owner: A. Owner Asset: Credit Card Data							
	CONFIDENTIALITY public (0), restricted (1-5), confidential (6-9), secure (10)			INTEGRITY low (1-3), moderate (4-7), high (8-9, very high (10)		AVAILABILITY low (1-3), moderate (4-6), high (7-8), very high (9), mandatory (10)	
Impact Requirement (1-10)	10 / secure			10 / very high		8 / high	
Threats <i>list all that apply</i>	Disclosure	Theft	Loss	Hacking	Input Errors	Drive Failure	Power Failure
Vulnerability (1-10) <i>none (0), low (1-4), moderate (5-7), high (8-9), very high (10)</i>	10	3	1	8	2	5	2
Threat (1 to 100) <i>Impact X Vulnerability</i>	100	30	10	80	20	40	16
Risk Level <i>Low (1-33), Medium (34-67), High (68-100)</i>	High	Low	Low	High	Low	Medium	Low
Countermeasures <i>list all that apply</i>	Password Protection			Firewall	Data Input Forms Data validation		

4. Kitől kell megvédeni?

- Ez is része lehet a kockázatelemzésnek, mert nagyban függ a cégtől, de azért néhány példa:
 - Script kiddie-k
 - Vandálok (deface)
 - Profi „blackhat” hackerek, akár a konkurencia megbízásából.
 - Rosszindulatú volt vagy jelenlegi alkalmazottaktól: nekik jelentős többletinformációjuk lehet a rendszer belső működéséről, ami bőven kompenzálhatja az esetlegesen hiányzó informatikai tudást.
- A felsorolásból látszik, hogy nincs olyan site, vagy alkalmazás vagy rendszer amit ne akarna valaki feltörni. Hiába vagyunk „kicsik”, a vandálok meg a script kiddiek így is meg fognak találni, hiába vagyunk jól védettek, a profik nem fogják egykönnyen feladni.

5. Hogyan védekezzünk?

- A jó programozó olyan valaki, aki mindig körbenéz jobbra és balra, mielőtt átmegy az egyirányú úton. — Doug Linder
- Nekünk csak egyszer kell szerencsésnek lennünk. Neked minden egyes alkalommal. — Az IRA Margaret Thatcher-nek, egy megghiúsult merénylet után
- A fejlesztő nem csak a hibás kóddal tudja kompromitálni a rendszert:
 - A forráskód nem biztonságos tárolása, kezelése.
 - Nem megfelelő ellenőrzési nyomvonal(audit trail).
 - Nem megfelelő kód review, vagy tesztelési folyamat.
 - Nem megfelelően szeparált, vagy biztonságos fejlesztői környezet.
 - Nem megfelelően automatizált, vagy biztonságos élesítési eljárás.
 - Etc.

5. Hogyan védekezzünk?

- Biztonsági hiba, sebezhetőség:
 - Olyan gyengeség vagy hiba a rendszer tervezésében, megvalósításában, vagy üzemeltetésében amit kihasználva áthágható a rendszer biztonsági házirendje.
- Tehát szükségünk van biztonsági házirendre.
- Le kell dokumentálni az összes biztonsági szempontból meghozott döntést:
 - ACL: mihez ki fér hozzá, ha megoldható akkor legyen központi autentikáció a különböző rendszerekhez.
 - Milyen ismert hibák, gyengeségek vannak a rendszerben (lehet hogy azt mondjuk egy funkcióra, hogy úgy is csak trusted dolgozók használhatják, ezért megengedhető, hogy lazábbak a szabályok, viszont ha később megváltozna a hozzáférők köre, tudnunk kell, hogy mik azok a módosítások, amiket meg kell ejteni előtte)

5. Hogyan védekezzünk?

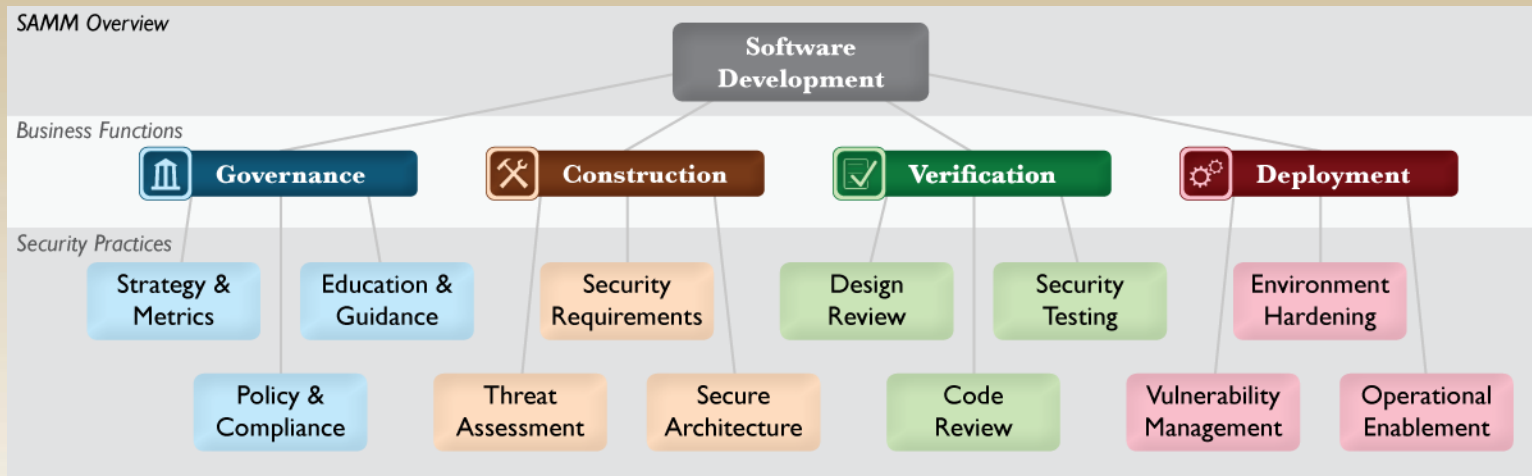
- Defense in depth:
 - A rendszerünk biztonsága soha ne egy komponens megfelelő működésén múljon, hanem a különböző biztonsági intézkedések egymást kiegészítve, erősítve alkossanak rendszert.
- Secure by design:
 - Már alapjaitól úgy építjük fel a rendszert, hogy szem előtt tartjuk, hogy támadásoknak lesz kitéve. Minden beérkező kérést, inputot, függvényhívást úgy kezelünk, hogy egy potenciális támadótól érkezett.
- Principle of least privilege:
 - Minden komponens, modul, függvény csak annyi információval és jogosultsággal rendelkezik, ami a feladata végrehajtásához szükséges.
- Security through simplicity
 - Az egyszerű rendszert könnyebb karbantartani, könnyebb a hibákat felfedezni, és kijavítani.

5. Hogyan védekezzünk?

- Separation of duties:
 - Azokat a feladatokat/pozíciókat, amik kritikusak akár a hiba lehetősége, akár a visszaélések veszélye miatt, szét kell bontani több lépésre, és a különböző lépéseket különböző személyhez rendelni.
 - Tehát például a produkciós környezetbe csak úgy kerülhet ki hibajavítás, hogy egy fejlesztő lefejleszti, a megrendelői oldalról valaki elfogadja, egy másik fejlesztő leellenőrzi és elfogadja, egy arra jogosult vezető jóváhagyja az élesítést, majd a rendszergazda élesíti.
- Audit trail
 - Legyen minden biztonsági vagy üzleti szempontból kritikus művelet biztonságos módon naplózva. A logok lehetőleg ne legyenek utólag módosíthatóak(append only).
- Fail-secure:
 - Ha bármi hiba lép fel a működés közben, akkor mindig a biztonságosabb módon álljon meg a program. Ne nyomjuk el a hibákat, és reméljük, hogy minden működni fog.

5. Hogyan védekezünk?

Secure Software Development Lifecycle: OWASP SAMM (Software Assurance Maturity Model)



5. Hogyan védekezzünk?


- Legyen Issue tracking rendszer, ahol folyamatosan dokumentálásra kerülnek a hibák, és követni lehet az életútjukat.
- Legyen kidolgozott code review folyamat, valamint nagyon hasznos tud lenni az automatizált tesztek is a hibák javításának a validálásában.
- Legyen külön fejlesztői és teszt környezetünk, ahol a hibákat és a javításokat megfelelően tesztelni lehet.
- A javított hibákra mindig írjunk regressziós teszteket, hogy ne tudjanak később visszakerülni a rendszerbe.
- A fejlesztési folyamataink legyenek jól definiáltak.
- A coding/development standards dokumentáció legyen betartatva, hiszen sokkal egyszerűbb így kiszűrni a hibákat, illetve nem elkövetni őket.

5. Hogyan védekezzünk?

- Az új kollégák legyenek megfelelő szinten a biztonságos fejlesztés területén és/vagy kapjanak alapos oktatást az első hetekben.
- Folyamatos képzéssel gyarapítani kell a fejlesztők tudását, illetve követni a trendeket a biztonságos fejlesztés témakörében.
- Rendszeres kódanalízist (manuális, illetve statikus elemző eszközökkel), illetve penetrációs tesztekkel kell végezni a kódon.
- A blackbox tesztekkel jobban lehet szimulálni a külső támadásokat, viszont a whitebox tesztekkel nagyobb lefedettséget lehet elérni.

5. Hogyan védekezzünk?

OWASP Top Ten

RISK	 Threat Agents	Attack Vectors → Security Weakness		Technical Impacts → Business Impacts	
		Exploitability	Prevalence	Detectability	Impact
A1-Injection		EASY	COMMON	AVERAGE	SEVERE
A2-XSS		AVERAGE	VERY WIDESPREAD	EASY	MODERATE
A3-Auth'n		AVERAGE	COMMON	AVERAGE	SEVERE
A4-DOR		EASY	COMMON	EASY	MODERATE
A5-CSRF		AVERAGE	WIDESPREAD	EASY	MODERATE
A6-Config		EASY	COMMON	EASY	MODERATE
A7-Crypto		DIFFICULT	UNCOMMON	DIFFICULT	SEVERE
A8-URL Access		EASY	UNCOMMON	AVERAGE	MODERATE
A9-Transport		DIFFICULT	COMMON	EASY	MODERATE
A10-Redirects		AVERAGE	UNCOMMON	EASY	MODERATE

5. Hogyan védekezünk?

- **A1 – Injection**

- SQL injection
 - **SELECT * FROM `users` WHERE `name` = '' OR '1'='1';**
 - Kivédhető megfelelő input validációval, az SQL injection kivédésére alkalmazhatunk prepared statement-eket, illetve a modernebb db libek már emulálni is tudják ezt a fajta paraméter bindingot, ezáltal könnyebbé téve az sql paraméterek escape-elését.
- Code injection (php, ruby, etc.)
 - eval is evil
 - Ha egy mód van rá, ne akarjunk ilyet csinálni, ha mégis, akkor nagyon szigorú input validációval.
- Local/Remote file inclusion
 - Remote file inclusion-t tiltsuk le, a helyi esetben pedig ismét csak szigorú input validáció.
- Shell injection
 - Ritkán szükséges, de ha mégis, akkor megfelelő input validáció/escapeelés.
- Ldap, xpath, etc.
- Mindegyik ilyen típusú hiba kivédhető megfelelő input validációval.
- FIEO: Filter Input, Escape Output

5. Hogyan védekezzünk?

- **A2 – Cross-Site Scripting (XSS)**

- Ez is injection, javascript a kliens oldalon(szerveroldalon generálva), csak annyira népszerű(a támadások jelentős részéért felelős), hogy saját pontja is van.
- Lehet perzisztens, vagy reflected a típusa, attól függően, hogy eltárolt adatból, vagy a lekérés paraméteréből injectálható a kód.
- Nem teljesen triviális a szűrése, validálása, mivel kontextustól függően eltérően kell escapelni:
[http://www.owasp.org/index.php/XSS \(Cross Site Scripting\) Prevention Cheat Sheet](http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
- A legtöbb elterjedt frameworkben van rá szűrő, ha lehet használd azokat, ha nincs, akkor OWASP AntiSamy, vagy htmlpurifier, legvégső esetben szigorú whitelist.
- A sütilopás az egyik legismertebb felhasználása az XSS-nek, de korántsem az egyetlen.
- A HttpOnly sütik ellopása sem okoz különösebb problémát a támadónak, ha már talált XSS-t az oldalon. (*XMLHttpRequest* vagy *TRACE method*).
- Jelszólopás form spoofolással vagy akár jelszókezelőből kiolvasva, esetleg keyloggerrel.

5. Hogyan védekezzünk?

• **A3 – Broken Authentication and Session Management**

- Munkamenet azonosítót nem utaztatunk url-ben (browser history).
- Ha megoldható, a beléptetett felhasználóval mindig csak titkosított csatornán keresztül kommunikáljunk, ne csak a belépésnél, különben el lehet lopni a session sütit.
- Legalább minden privilégiumváltásnál (kilépés, belépés, etc.) generáljuk újra a munkamenet azonosítót.
- A munkamenet azonosítót ne választhassa tetszőlegesen a kliens (session fixation).
- A munkamenet azonosítót megfelelően nagy random poolból generáljuk, hogy ne lehessen kitalálni, vagy megjósolni a kiosztott azonosítókat.
- A lejáratott munkameneteket töröljük a szerverről is, ne csak a kliensen múljon, hogy lejár-e, vagy sem.
- Ha megoldható, akkor kössük IP-hez(vagy akár csak geoIP alapján országhoz) a munkameneteket, megakadályozva ezzel a session hijacking-ot.
- Jelszavakat nem tárolunk plain alakban, csak erős kriptográfiával, elfelejtett jelszó feature-nél egyszer használatos jelszót, vagy ami még jobb linket küldünk ki, belépés után kötelező a jelszóváltoztatás.

5. Hogyan védekezünk?

- **A4 – Insecure Direct Object References**

- /node/1/edit -> nincs lekezelve, hogy joga van-e szerkeszteni az aktuális usernek az 1-es idjű tartalmat.
- Logikai hiba, mindig győződjünk meg róla, hogy az ACL-nek megfelelően joga van-e az adott felhasználónak az adott műveletre az adott rekordon.

5. Hogyan védekezzünk?

- **A5 – Cross-Site Request Forgery(CSRF)**

- A felhasználó meglátogja a támadó webhelyet, ami egy rejtett iframe-ben elküld egy get vagy post kérést, ami a felhasználó nevében végrehajtásra kerül a célpont oldalon.
- Nagyon elterjedt hiba, pedig a védekezés viszonylag egyszerű: Minden oldalad, ami új tartalmat hoz létre, meglévőt módosít, vagy töröl, az POST-tal legyen elküldve(szemantikus web), illetve minden ilyen esetben a legenerált form-ba generálj bele egy tokent, aminek a jelenlétét a form beküldésénél vizsgáld meg.
- A manapság elterjedt frameworkok már általában tartalmaznak transzparens CSRF védelmet, de lefejleszteni sem egy ördögösség.

5. Hogyan védekezünk?

- **A6 – Security Misconfiguration**

- A rendszergazdákkal közös feladat ennek a hibalehetőségnek a minimalizálása.
- A fejlesztőknek is nyomon kell követniük az általuk használt framework, osztálykönyvtárak biztonsági frissítéseit, ugyanúgy, ahogy a többi komponenst is naprakészen kell tartani.
- A framework biztonsági beállításait ellenőrizni, a fájlok jogosultságait szintén.
- Információszivárgás: produkciós környezetben publikusan elérhető helyen nem lehet a rendszer paramétereit listázó script (pl. `phpinfo()`), illetve a hibákat sem jelenítjük meg a látogatóknak, mert az is segítség a támadóknak a rendszer feltérképezésében.

5. Hogyan védekezzünk?

- **A7 – Insecure CryptographicStorage**

- Használj erős titkosítást, az md5, sha1(hash függvények) már reális idő alatt törhető, ráadásul mindkét algoritmusban találtak sebezhetőségeket, ami méginkább megkönnyíti a támadó dolgát.
- Ahol nincs szükséged arra, hogy az eredeti információ visszaállítható legyen(belépési jelszavak általában tipikusan ilyenek), ott használj egyirányú hash algoritmusokat (sha-2 család), rekordonként egyedi salt-tal, így a szivárvány táblákkal sem lehet gyorsítani a törési folyamatot.
- Ahol vissza kell tudni állítani az eredeti információt, ott az AES(szimmetrikus/egykulcsos titkosítás), illetve az RSA(asszimmetrikus/nyílt kulcsos titkosítás) használata javasolt.
- Ha titkosítasz, akkor győződj meg róla, hogy a titkosításhoz használt kulcs megfelelően védve van, illetve hogy nem tárolod/továbbítod valahol máshol titkosítatlan formában a szenzitív adatokat.
- Készülj fel, hogy kompromittálódás esetén le tudd cserélni a titkosításhoz használt kulcsokat.

5. Hogyan védekezzünk?

- **A8 – Failure to Restrict URL Access**

- A hiba abban rejlik, hogy bizonyos URL-ekre nem terjed ki a hozzáférés szabályzás, pedig szükség lenne rá.
- FrontController pattern használata segítségével könnyebb elkerülni, hogy valamilyen oldal kimarad a szórásból.
- Ha FrontController-t használasz, akkor nincs szükség rá, hogy a többi fájlod, amiket csak include-olsz, kint legyen a DocumentRoot alatt.
- A hozzáférés szabályozás terjedjen ki minden oldalra, és mindenki csak ahhoz az oldalhoz kapjon hozzáférést, amit a szerepe szerint neki látnia kell.

5. Hogyan védekezünk?

- **A9 – Insufficient Transport Layer Protection**
 - Az érzékeny adatokat csak titkosított csatornán keresztül szabad továbbítani.
 - A titkosításhoz megfelelően erős kriptográfiát használjunk.
 - A titkosított oldalakon minden erőforrás titkosított csatornán legyen lekérve, különben a felhasználónak megjelenhet egy figyelmeztetés, illetve a munkamenet azonosítója is kiszivároghat.
 - A munkamenet sütikre állítsuk be a secure flaget, így csak titkosított csatornán kerül elküldésre a kliens által.
 - Megfelelő tanusítványt használjunk: hiteles, megbízható kibocsájtó által lett kiadva, nem járt le, nem lett visszavonva, és az aktuális domain és a kiállított domain megegyezik.

5. Hogyan védekezünk?

- **A10 – UnvalidatedRedirects and Forwards**

- A hiba abból áll, hogy user inputban megadott url-re átirányít valamely oldalunk, ezáltal például egy emailben a támadó küld egy linket az áldozatnak, ami látszólag az áldozat által ismert megbízható site, de ha rákattint, akkor átirányítja a script a támadó oldalra.
- Input validációval kiszűrhető, illetve ha nincs szükség rá, akkor ne irányítsuk át a látogatóinkat paraméterben kapott külső domainekekre.

5. Hogyan védekezünk?

- **A10 – UnvalidatedRedirects and Forwards**

- A hiba abból áll, hogy user inputban megadott url-re átirányít valamely oldalunk, ezáltal például egy emailben a támadó küld egy linket az áldozatnak, ami látszólag az áldozat által ismert megbízható site, de ha rákattint, akkor átirányítja a script a támadó oldalra.
- Input validációval kiszűrhető, illetve ha nincs szükség rá, akkor ne irányítsuk át a látogatóinkat paraméterben kapott külső domainekekre.

5. Hogyan védekezzünk?

- **Egyéb tippek/trükkök**

- Kerüld a serializált blobokat. (Nehéz validálni, kibontásnál érhetnek meglepetések)
- Mindig legyél explicit, nem jó ötlet a nem várt inputoknál próbálni értelmes értékekre konvertálni a bejövő adatot.
- Figyelj oda a távoli JS/JSONP behúzására, lehet abból is XSS.
- Nézz utána a throttling fogalmának, DOS, brute-force kísérleteket jól lehet szűrni vele, akár csak azzal, hogy kilépsz a kód elejében, ha valaki ugyanarról az IP-ről/sessionról jelszó töréssel, vagy terheléses támadással próbálkozik, akár úgy, hogy ilyenkor feldobsz neki egy captcha-t.
- Ha van keresés az oldalon, akkor figyelj oda a like ‘%%’, illetve limit nélküli querykre.
- Fájl feltöltések külön mappába menjenek, amin nincs engedélyezve semilyen script futtatási lehetőség(AddHandler, etc.). Ugyanígy a mime-type-okkal is óvatosan bánj, csak a mime type-ból, vagy csak a kiterjesztésből nem biztos, hogy meg tudod mondani, hogy mi van a fájlban.
- A transzparenst WAF(web application firewall: mod_security, WebKnight, Zorp) jellegű rendszerek nem mindig jók, lehet benne false pozitív, illetve előfordulhat, hogy valamit nem csíp meg. Használhatod kiegészítőként, de ne ezen múljon az életed (emlékezz, defense-in-depth).

5. Hogyan védekezzünk?

- **Egyéb tippek/trükkök**

- A kritikus függvényeidre írd olyan teszteseteket, amik szándékosan hibás, hiányos adatokra várják a hibát.
- Innen egy lépéssel továbbgondolva kipróbálhatod a Fuzz testinget: hibás, váratlan, random adatokkal meghívogatni a rendszer különböző részeit, és figyelni, hogy hogyan viselkedik.
- Tanulj mások hibáiból, sokkal kellemesebb, hogyha nem azért tudsz időt szánni a biztonsági javításokra, mert felnyomta valaki a rendszeredet, és a menedzsment leüvölti a fejed.
- Ha gondolkoztál már rajta, hogy milyen jó lenne egy normális sql „tűzfal”, amit nem Lua-ban kell configolni, akkor nézd meg a Greensql-t.
- A pentesteléshez/vulnerability scanhez nézd meg a következőket: Nessus, OpenVAS, nikto, wapiti.
- A static analysis-hez nezd meg a Rats-ot, vagy a RIPS-et, de ez eléggé nyelv függő.
- A sütiknél megfelelően állítsd be a domaint és a pathot, ne legyen nagyobb a hatókör, mint szükséges.

5. Hogyan védekezzünk?

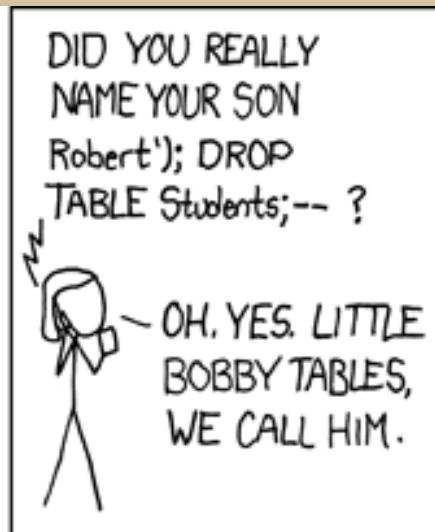
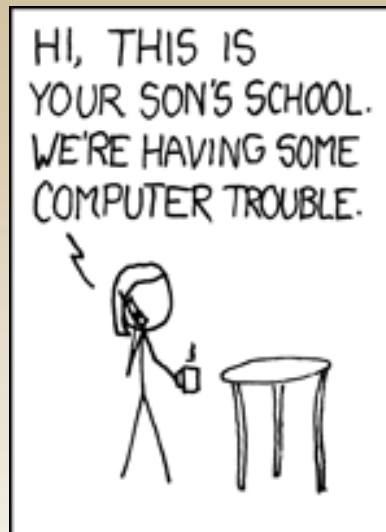
- **A végére néhány PHP specifikus dolog**

- A `register_globals` rossz, ne használd! De tényleg!
- A `magic_quotes` szintén csak a semminél jobb egy kicsit, ne használd, escapeld magadnak a paramétereket, vagy bízd a frameworkre, amit használsz.
- Suhosin legyen fent a szerveren, mert a buffer overflow-k nagy részét megfogja, sőt még az ilyen alap dolgokhoz is szükség van rá, mint hogy a végtelen rekurzió ne crasheltesse el a zend engine-t.
- Ha mindenképp saját magadnak akarod kezelni az input validációt, akkor mindenképp nézd meg az inspekt projectet, illetve használd a php-ban elérhető filter extensiont.
- Manuális statikus kódanalízisre próbáld meg a `bytekit/bytekit-cli` alkalmazásokat.
- Ha mindenképp akarsz transzparens megoldást a támadások ellen, akkor érdemes lehet vetni egy pillantást a PHPIDS projectre.
- `open_basedir` direktíva hasznos lenne, de sok extension nem veszi figyelembe, ha biztosra akarsz menni, akkor marad a `chroot/jail`.
- Kapcsold ki az **`allow_url_include`** -ot, és ha nincs szükséged rá, akkor az `exec/system/etc.` Függvényeket.

5. Hogyan védekezünk?



5. Hogyan védekezzünk?



Vége.

Kérdések?

<http://tyrael.hu/>

<http://slideshare.net/Tyrael/>

<http://twitter.com/#!/Tyr43l>