

# **Biztonságos programok fejlesztése és a web alapú rendszerek biztonsági sajátosságai**

**Vázlat a Nyíregyházi Főiskola  
IT biztonság II. című tantárgyához**

**Szerző: Mátó Péter <[mato.peter@andrews.hu](mailto:mato.peter@andrews.hu)>**

**Verzió: v3.4 - 2011.05.08**

# Tartalomjegyzék

Tartalomjegyzék.....	2
1.Web biztonság területe.....	4
1.1.A web működése.....	4
1.2.A web ma.....	4
1.3.A web alapú rendszerek kommunikációja.....	4
1.3.1.A HTTP legfontosabb jellemzői.....	4
1.3.2.A protokoll általános működése.....	4
1.3.3.Ismétlés: SSL, TLS.....	4
1.3.4.A HTTPS protokoll.....	5
1.4.A felhasználó azonosítása és feljogosítása.....	6
1.4.1.Emberek és robotok megkülönböztetése.....	6
1.4.2.Identifikáció, autentikáció, authorizáció (ismétlés).....	6
1.4.3.A jelszavakról általában.....	6
1.4.4.Azonosítási módszerek.....	6
1.4.5.Session kezelési hibák.....	7
1.4.6.Felhasználó kizárása (lock out).....	7
1.4.7.Kliens hibák.....	7
1.4.8.Web kereső motorok által okozott veszélyek.....	8
1.4.9.Szerver oldali támadások.....	8
1.4.10.A webalkalmazás jogai és az oprendszer védelmi rendszere.....	9
1.4.11.Speciális veszélyforrások.....	10
1.4.12.Sütikkel kapcsolatos problémák.....	10
1.4.13.Phishing.....	10
1.4.14.Fizetés a hálózaton.....	11
2.Általános programozási hibák és elkerülésük.....	12
2.1.Célunk a programozás minőségével kapcsolatban.....	12
2.2.A programozási projekt életútja.....	12
2.3.A legfontosabb általános fejlesztési problémák.....	12
2.3.1.Nem megfelelően ellenőrzött bemenet.....	12
2.3.2.Hibatűró megvalósítás.....	13
2.3.3.Hozzáférés vezérlési hibák.....	13
2.4.Általános jó tanácsok megfelelő minőségű programok fejlesztéséhez.....	13
2.5.Általános biztonságos programozási módszertan: a CC szabvány.....	14
2.6.Jogosultsági szintek, mint védekezési eszköz.....	14
2.7.A programok támadásának módjai.....	15
2.8.A működési környezetből adódó hibák.....	15
2.9.A programozási környezetből adódó hibák.....	15
2.9.1.Néhány általános programozási hiba.....	15
2.9.2.Elérési út bejárás (dot-dot bug, Path traversal).....	15
2.9.3.Abszolút út bejárás (Absolute Path Traversal).....	16
2.9.4.Kódolási problémák, élő nyelv-specifikus hibák.....	16

2.9.5.A C nyelv sajátos hibái.....	16
2.9.6.Szkriptnyelvek (php, perl, python.....)	16
2.9.7.Adatbázisokkal kapcsolatos hibák.....	16
2.9.8.Az adatok XML reprezentációja.....	17
2.10.A hibák elkerülése: A beérkező adatok ellenőrzése.....	17
2.10.1.A beérkező adatok csatornái.....	17
2.10.2.A beérkező adatok ellenőrzése.....	17
2.11.A hibák elkerülése: A környezet védelmi megoldásai.....	18
2.11.1.A programhibák elleni védelem szintjei.....	18
3.Ajánlott olvasmányok.....	20
4.Demonstrációk.....	21
4.1.Firefox.....	21
4.2.Chrome, mint biztonsági probléma.....	21
4.3.User auth lehallgatása.....	21
4.4.Demó minta programcskák.....	21
4.5.A gagyiszerv.pl.....	21

# 1. Web biztonság területe

Web böngészők

Egyéb HTML megjelenítőt használó programok

- Mail kliensek
- RSS olvasók
- Mobiltelefonok felülete és szoftverei

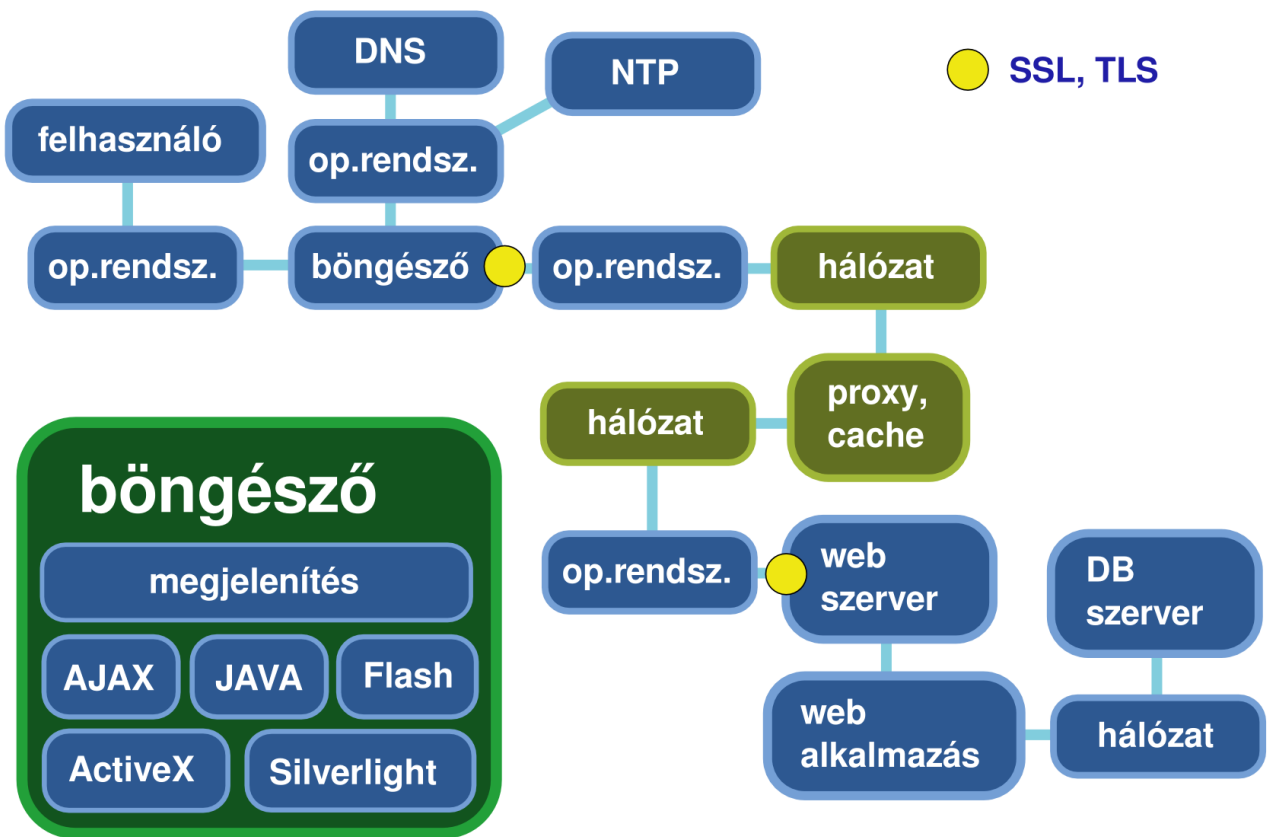
Web szerverek

Web proxy-k

DNS szerverek

## 1.1. A web működése

client - c.system - network - proxy - network - s.system - webservice - web app - rdbms - data



## 1.2. A web ma

Aktív kliens oldal (GMail)

HTML -> XML, XSL / XHTML / XFORMS ...

## 1.3. A web alapú rendszerek kommunikációja

HTTP (Hypertext Transfer Protokoll)

### 1.3.1. A HTTP legfontosabb jellemzői

Állapotmentes

Nyílt, lehallgatható

Szöveg orientált

Ellopható, módosítható - pl. Man in the middle attack

### 1.3.2. A protokoll általános működése

Szöveges, nem titkosított protokoll.

Legfontosabb metódusai: HEAD, GET, POST

A kódolásokról

### 1.3.3. Ismételés: SSL, TLS

A szimmetrikus titkosítás (ismételés)

Az aszimmetrikus titkosítás (ismételés)

### 1.3.4. A HTTPS protokoll

A HTTPS nem más, mint a HTTP egy SSL által titkosított csatornában. (Az SSL utódja a TLS, amely valójában csak konstansokban különbözik az SSL-től, így a továbbiakban összefoglaló néven csak SSL-ként hivatkozom rá.) A böngésző SSL protokollon kapcsolatot hoz létre a webszerverhez, és utána ezen a titkos csatornán a már megismert HTTP protokollt használja. Az SSL kissé leegyszerűsítve: a kommunikáló felek aszimmetrikus titkosítás segítségével, egy PKI által kibocsátott digitális tanúsítvánnyal azonosítják (az ellenőrzés nem kötelező, de a szervernek mindenképpen fel kell mutatnia egy tanúsítványt) a másik oldalt, majd egy egyeztetett szimmetrikus algoritmussal és kulccsal kezdenek beszélgetni.

Előnye, hogy a kliens nagy biztonsággal ellenőrizheti, hogy valóban ahhoz a szerverhez csatlakozott-e, akihez akart. Ha a szerver úgy van beállítva, akkor a kliensnek is be kell mutatnia a saját tanúsítványát, így az azonosítás két irányú lehet. Mivel a világon számos komoly biztonsági követelményeknek megfelelő tanúsító szervezet (CA - Certification Authority) állít ki hivatalos tanúsítványokat, így az egymást nem ismerő felek is nagy biztonsággal meggyőződhetnek a másik fél hitelességéről.

Hiányosságai:

- nagy processzor erőforrásigény
- nagy entrópia igény (ezt sokszor csak speciális hw beépítésével lehet megoldani)

### Tanúsítványok megfelelő ellenőrzése

CA aláírás

érvényességi idő

CRL

OCSP

## Tanúsítványok beszerzése

Tanúsítószervezetek által kiadott tanúsítványok

A felhasználó regisztrációjakor a szerver ellenőrzi a felhasználó tanúsítványának érvényességét (lásd HTTPS protokoll leírásánál), majd a webalkalmazás feljegyzi az adatbázisban a tanúsítvány kiállító szervezetét és a tanúsítvány sorszámát. Ezek után, ha a felhasználó az adott tanúsítvány segítségével kapcsolódik a szerverhez, akkor az alkalmazás minden más ellenőrzés nélkül tudja, hogy ki a kliens.

FIXME subject??

Saját CA által kiállított tanúsítványok

Ha lehetséges egy saját tanúsító szervezet felállítása (például egy nagyobb cég belső CA-ja), akkor a szerveren beállítható, hogy csak a saját CA által kiadott, érvényes tanúsítványokat fogadjuk el. Így közvetlen befolyása alá kerül a felhasználók hozzáféréseinek szabályozása a tanúsítványok szükség esetén való visszavonásával.

Ön-aláírt tanúsítvány használata

Az ön-aláírt tanúsítvány (angolul self-signed certificate) a legegyszerűbben elkészíthető, de a legkevesebb haszonnal járó módszer. Az egyetlen értelme, hogy használatával a böngésző és a webszerver képes titkosítva kommunikálni, de a kliens nem tud a szerver valóságáról meggyőződni, így az SSL használatának egyik legfontosabb előnye veszik el.

## Virtuális hosztok problémái

\*.domain.tld

SNI Server Name Indication (RFC 3546)

Nem minden böngésző támogatja őket

## 1.4. A felhasználó azonosítása és feljogosítása

### 1.4.1. Emberek és robotok megkülönböztetése

webspam, adatgyűjtés ellen

Turing teszt

Capcha

### 1.4.2. Identifikáció, autentikáció, autorizáció (ismétlés)

DAC, MAC

### 1.4.3. A jelszavakról általában

Jelszó biztonság ( hossz, karakterek )

Csatorna függő: lehallgathatóság

Jelszó tárolás ( plain pro és kontra )

Hash algoritmusok ( crypt, md5, sha1, sha256, sha512, mire jó a salt )

Jelszó generálása

Jelszó bekérése, cseréje

## 1.4.4. Azonosítási módszerek

### A HTTP protokoll saját azonosítási mechanizmusai

Realm fogalma

Challenge-Response

Szerver:

```
WWW-Authenticate: Basic realm="TitkosAdatok"
```

Kliens:

```
Authorization: Basic YXR5YTptZWdhc3plcGVydG10b2s=
```

Automatikus Response egy sütiben tárolva

### Alkalmazás szintű felhasználó azonosítás

#### Felhasználói név és jelszó

Lásd "A jelszavakról általában"

#### Süti (cookie) alapú azonosítás

A felhasználó által is módosítható (FIXME leírni miért)

Elrabolható (lehallgatás, XSS)

- IP-hez köthető
- lejárata szabályozható

#### Rejtett űrlapmező alapú azonosítás

#### URL alapú azonosítás

lehallgatható

cache-ek is látják

#### Tanúsítvány alapú azonosítás

Az alapokat lásd a HTTPS protokoll leírásánál. A web szerveren futó program a szervertől megkapja a felhasználó által átadott tanúsítványban lévő adatokat, így ezen adatok alapján ellenőrizhető a felhasználó kiléte. Ennek természetesen alapfeltétele, hogy a szerver megfelelően legyen beállítva, és csak megbízható tanúsító szervezetek által kiállított tanúsítványokat fogadjon el.

A továbbiakat lásd a HTTPS protokoll leírásánál.

#### Kétlépcsős azonosítás (challenge-response)

Pl. sms a felhasználónak, OTP módszerek (SKey, CryptoCard) és a tanúsítvány alapú azonosítás is ide tartozik...

## Single sign on

OpenID, Passport...

### 1.4.5. Session kezelési hibák

Megvalósítás: süti, rejtett mező és URL rablás

A session rablás ellen tárolhatjuk a REMOTE\_IP-t és a USER\_AGENT-et. De ezzel lehet problémája a felhasználónak (roaming userek, dinamikus címek, állapotszinkron több böngésző között...). Védekezés lehet még a süti érvényességének (lejáratának) hangolása a biztonsági követelményeknek megfelelően.

### 1.4.6. Felhasználó kizárása (lock out)

A megtámadott felhasználó belépési limitjét kihasználja

### 1.4.7. Kliens hibák

#### Kliens dinamikus tartalom

##### Java

Sandbox-ban futó appletek

Automatikusan elindulnak

Jó, ha a JVM biztonságos

Sajnos néha nem az, az applet bármit megtehet

##### ActiveX

A felhasználó eldöntheti, hogy mehet-e -

Nem ért hozzá, nem jól dönt

Biztonsági hiba esetén nem is kérdez, csak jön

##### JavaScript

Újabbban felkapták, Web 2.0

AJAX

Az internetes Javascript-ek korlátozva vannak

Volt már hiba, amitől fájlokat tudtak manipulálni

##### VB Script

Hasonló, mint a JS, de csak Windowson működik

### 1.4.8. Web kereső motorok által okozott veszélyek

Lenyomozhatóság

Bizalmas adatok kerülnek ki

a robotok meglelik

beteszik a kereső cache-be, aztán már nagyon nehéz levenni

## 1.4.9. Szerver oldali támadások

Rendszer elleni támadás (rendszer típusának megh.: nmap)

Web szerver elleni támadás (típus megh.: nc)

Web alkalmazás elleni támadás

RDBMS elleni támadás

### A támadó célja

Deface, massdeface

DoS, DDoS (pl. tcp connect flood)

Illetéktelen hozzáférés

Illetéktelen adatmódosítás

### Web szerver scannerek

Akár teljes hibaadatbázissal

### Szerver hibák

Azonosítás kijátszása

pl. .htaccess; jelszóval - brute force; IP szerint

SQL injection - a beléptető SQL parancs ellen

Feljogosítás kijátszása

.. bug

Példa:

```
http://www.example.com/WebCalendar/login.php?user_inc=../../../../../../../../etc/passwd
```

### Néhány gyakran használt támadási mód

#### Dupla kódolás (Double coding)

példa1:

```
"../" -> "%2E%2E%2f", "%" -> "%25"
```

```
"../" -> "%252E%252E%252F"
```

példa2:

```
<script>alert('XSS')</script> -> %253Cscript%253Ealert('XSS')%253C%252Fscript%253E
```

#### XSS v. CSS

példa1:

```
<script type="text/javascript">
```

```
var adr = '../evil.php?cakemonster=' + escape(document.cookie);  
</script>
```

példa2:

```
<script>  
password = prompt("Please enter your dial-up password","");  
...  
</script>
```

példa3:

```
<img src='$url'> <- "doesntexist.jpg" onerror='alert(document.cookie) "
```

Mivel JS, ezért célzott, akár egyénre szabott támadást tesz lehetővé

### Alternatív XSS leírás

```
<script>alert('y0u ar3 0wn3d!');</script>
```

helyett:

```
&\<script&\gt;alert&\#40;'y0u ar3 0wn3d!'\&\#41;;&\</script&\gt;
```

## 1.4.10. A webalkalmazás jogai és az oprendszer védelmi rendszere

A webalkalmazás felhasználója és csoportja(i)

A processzek közti kommunikáció

TCP/IP

szignálok

Az elérhető állományrendszer

## 1.4.11. Speciális veszélyforrások

Adobe Flash

Google Gears

Adobe Air

## 1.4.12. Sütikkel kapcsolatos problémák

### Sütis adatgyűjtés

A tárolt sütik és tartalmuk sokat elárulnak a felhasználó hálózati szokásairól, sőt szokásairól

Minek?

A támadó segítséget kaphat, hogy merre induljon

Irányított, személyre szabott reklám

CIA, NSA :)

Speciális sütik

flash cookie

LSO (Local Shared Object)

## Süti rablás

### Süti módosítása

Ezt a technikát csak nem titkosított sütinél használhatja a támadó

## 1.4.13. Phishing

### Megtévesztő levelek, weboldalak

#### Kis kitérő: SMTP

### Megtévesztő linkek, az URI támadása

A cél megváltoztatása:

```
<a href="http://evil.hu">http://www.otp.hu</a>
```

A domain a kukac után (felhasználónév használata):

```
<a href="http://www.otp.hu@evil.hu">http://www.otp.hu</a>
```

Egyszerű megtévesztés 1:

```
<a href="http://www.OTP.hu">http://www.OTP.hu</a>
```

Egyszerű megtévesztés 2:

```
<a href="http://www.cem.com">http://www.cern.com</a>
```

Egyszerű megtévesztés 3:

```
<a href="http://www.tv11.hu">http://www.tv11.hu</a>
```

IDN megtévesztés:

```
<a href="http://www.p%D0%B0ypal.com/">paypal.com</a>
```

### A státuszsor támadása

## 1.4.14. Fizetés a hálózaton

## 2. Általános programozási hibák és elkerülésük

### 2.1. Célunk a programozás minőségével kapcsolatban

Helyes működés

Stabilitás

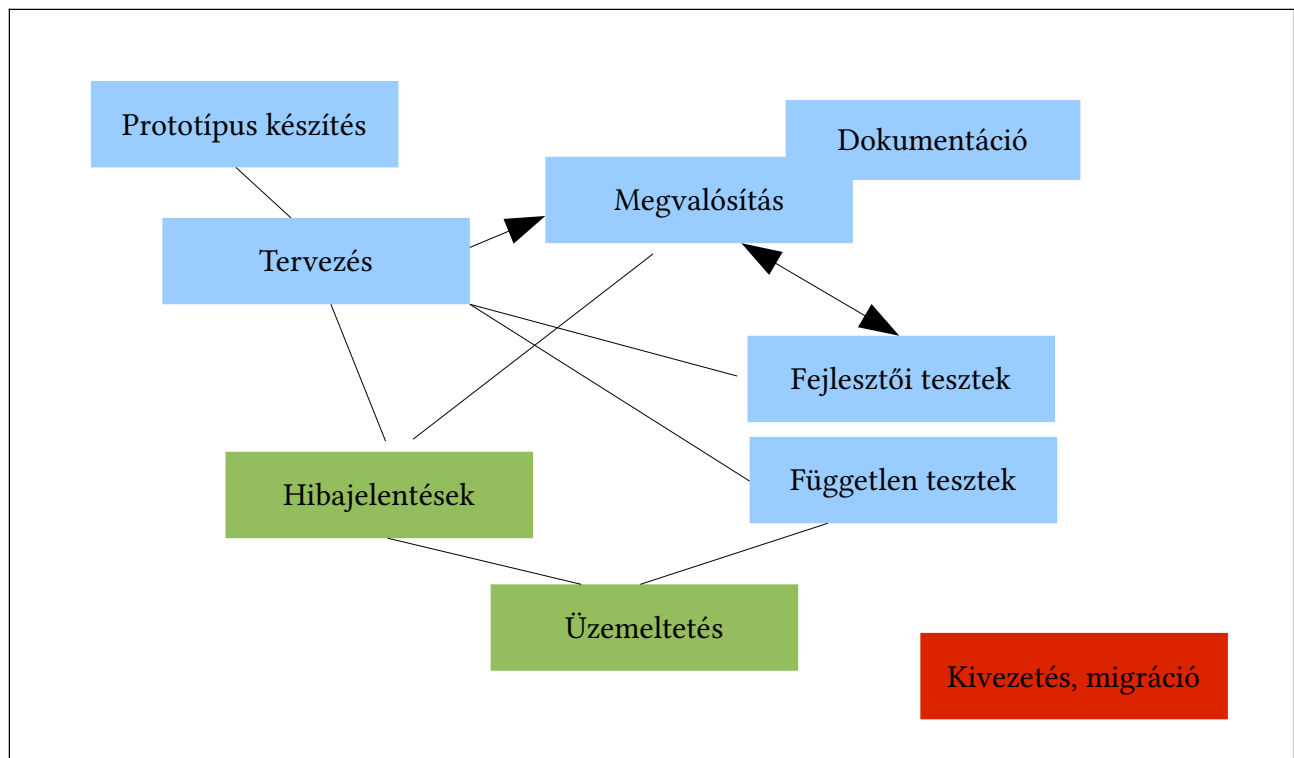
Biztonságos működés

### 2.2. A programozási projekt életútja

Prototípus

Tervezés

Megvalósítás



### 2.3. A legfontosabb általános fejlesztési problémák

#### 2.3.1. Nem megfelelően ellenőrzött bemenet

Fail open (perl):

```
if ( $input =~ /\.\./ ) { # hiba }
```

Fail safe (perl):

```
if ( $input =~ /^[a-zA-Z0-9\|]+\.[a-zA-Z0-9\|]+*$/ ) { # ok }
else { hiba }
```

Nincs ellenőrzés, hibás (php):

```
$month = $_GET['month'];
$year = $_GET['year'];
exec("cal $month $year", $result);
print "<PRE>";
foreach ($result as $r)
{ print "$r<BR>"; }
print "</PRE>";
```

Megfelelően ellenőrzive, helyes (php):

```
$month = $_GET['month'];
$year = $_GET['year'];
if (!preg_match("/^[0-9]{1,2}$/", $month)) die("Bad month, please re-enter.");
if (!preg_match("/^[0-9]{4}$/", $year)) die("Bad year, please re-enter.");
exec("cal $month $year", $result);
print "<PRE>";
foreach ($result as $r) { print "$r<BR>"; }
print "</PRE>";
```

## 2.3.2. Hibatűró megvalósítás

Inkább lépjen ki, mint hibásan működik

Hibatűró megvalósítás – pl.: try catch finally

## 2.3.3. Hozzáférés vezérlési hibák

A webroot-on belül elhelyezett adat könyvtárak, index fájl nélkül. Ha az apache automatikus indexelése be van kapcsolva akkor a támadó hozzáférhet az adatokhoz.

php programok mentése a webroot alatt .bak kiterjesztéssel

```
http://valaki.hu/getdata.php/data/publikus.pdf
http://valaki.hu/data/publikus.pdf
http://valaki.hu/data/
```

```
/var/www/valaki.hu webroot
/var/www/valaki.hu_data
```

## 2.4. Általános jó tanácsok megfelelő minőségű programok fejlesztéséhez

Precíz rendszerspecifikáció

pontosan meg kell fogalmazni a célokat funkcionális szempontból

a problémát részletekre kell bontani, a bonyolultság függvényében akár több rétegben

a részeknek megfelelő illesztő felülettel kell rendelkeznie, ha kell modulonként lehessen cserélni

meg kell győződni róla, hogy a tervek rétegei megfelelnek egymásnak

Fejlesztői minőségkultúra

Megfelelő programnyelv kiválasztása

Fontos szempontok

hozzáértés

átláthatóság, érthetőség

egységbe zárás

információ elrejtés

szigorú típusosság

Helyes programozási technikák

meg kell ismerni, és kerülni kell a tipikus hibákat (lásd később)

nehezen értelmezhető kódrészletek kerülése (minden fejlesztőnek olvasható legyen)

a hibára hajlamos programszerkezetek elkerülése (fejlesztési vs. futási sebesség)

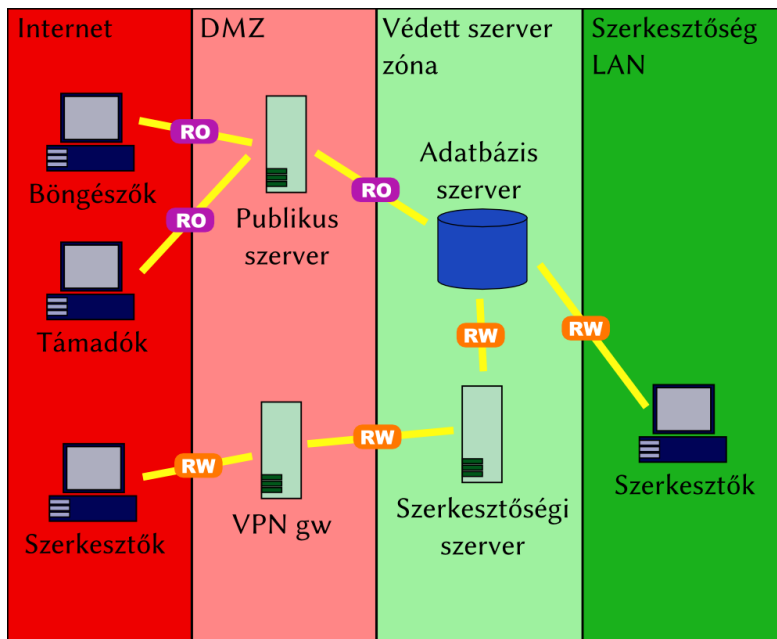
példa sql do

hasznos a programozási környezet beépített védelmi funkcióinak használata, pl. Perl: Tainted mód

## 2.5. Általános biztonságos programozási módszertan: a CC szabvány

Common Criteria (ISO/IEC 15408:1999)

## 2.6. Jogosultsági szintek, mint védekezési eszköz



Ha nem muszáj, ne használjunk DB-t

DB felhasználók használata

Jogosultságok minimalizálása időben és lehetőségekben egyaránt

Különböző biztonsági szintű felhasználók felület szeparációja

az egyszerű felhasználók nem is férhetnek hozzá az admin felülethez

hálózati leválasztás (lásd ábra)

hozzáférés csak tanúsítvány használatával

íráshoz csak olyan felhasználó használható, akinek van személyes felhasználója DB szinten is

## 2.7. A programok támadásának módjai

A program működésének befolyásolása adatainak megváltoztatása által

A rendszerben lévő kód végrehajtása nem megfelelő sorrendben

Támadó kód bevitele és végrehajtása

## 2.8. A működési környezetből adódó hibák

Külső hívások eltérítése (PATH, tcp redirect ...)

Erőforrások túlzott használata

Fájlrendszer teleírása

DOS túl sok kérés imitálásával (nagy mennyiségű GET, majd disconnect)

klasszikus DDOS (nagy mennyiségű kérés botnet segítségével)

SQL lekérdezések, mint DOS lehetőség (memória vagy processzor elfogyasztása)

## 2.9. A programozási környezetből adódó hibák

### 2.9.1. Néhány általános programozási hiba

Kezdőérték nélküli változók használata, ahol a támadó képes befolyásolni a kezdőértéket

Egész szám kezelési problémák

integer overflow, signedness bug (demo)

összeadásnál, szorzásnál, értékadásnál

### 2.9.2. Elérési út bejárás (dot-dot bug, Path traversal)

Dekódolás előtti ellenőrzésnél:

```
%2e%2e%2f -> ../
%2e%2e/ -> ../
..%2f -> ../
%2e%2e%5c -> ..\
%2e%2e\ -> ..\
..%5c -> ..\
%252e%252e%255c -> ..\
..%255c -> ..\
```

és így tovább.

## 2.9.3. Abszolút út bejárás (Absolute Path Traversal)

`http://testsite.com/get.php?f=list`

helyett

`http://testsite.com/get.asp?f=/etc/passwd`

## 2.9.4. Kódolási problémák, élő nyelv-specifikus hibák

Betű szerinti rendezés módosulása

## 2.9.5. A C nyelv sajátos hibái

Többnyire Neumann a hibás, minek kellett egy helyre tenni a kódot és az adatokat

puffer túlsordulás / stack v buffer overflow, overrun

halom túlsordulás / heap overflow

dupla felszabadítás / double free

formázó sztring hiba / format string bug

return to libc

## 2.9.6. Szkriptnyelvek (php, perl, python...)

include

magic open

## 2.9.7. Adatbázisokkal kapcsolatos hibák

### Relációs adatbázisok

A leggyakrabban az adatbázisban tárolják a kontroll adatokat is (struktúra, felhasználók...). Biztonsági szempontból ez nagyon nem szerencsés. Folyománya: DML-lel minden információ elérhető az adatbázisról (MySQL, PostgreSQL: INFORMATION\_SCHEMA; MS SQL: SYSOBJECTS)

A legfontosabb hiba, az SQL injection:

Az SQL sztring php-ben valahogy így néz ki:

```
select * from arucikkek where id=$_GET['id']
```

normális használat mellett ezt csinálná:

```
select * from arucikkek where id=1
```

a támadó így alakítja a működését:

```
select * from arucikkek where id=1000 union select nev,jelszo from users
```

A UNION SELECT használatához a támadónak tudnia kell hány darab (és milyen típusú) mező van az eredeti lekérdezésben

Nem csak az adatbázis adatai érhetők el

MySQL: LOAD\_FILE(), INTO OUTFILE, FROM DUMPFILE

MSSQL: xp\_cmdshell

## **Hierarchikus adatbázisok**

pl. LDAP esetén filter injection

### **2.9.8. Az adatok XML reprezentációja**

XPath injection

## **2.10. A hibák elkerülése: A beérkező adatok ellenőrzése**

### **2.10.1. A beérkező adatok csatornái**

#### **Az operációs rendszer csatornái**

Parancssor

Környezeti változók

Fájl olvasás

Fájl descriptorok

Fájl nevek

Fájl tartalma

Konfigurációs fájlok

#### **Hálózatról érkező adatok**

A HTTP protokoll elemei

URL

POST metódus esetén az adatok

HTTPS esetén a tanúsítványban szereplő adatok (subject, issuer)

Adatbázisból érkező válaszok

Memcached-ből érkező válaszok

Saját speciális kiegészítő szerver által adott válaszok

### **2.10.2. A beérkező adatok ellenőrzése**

Lásd ajánlott irodalom [1]

## Hossz ellenőrzés

## Karakter osztály ellenőrzése

## Reguláris minta illesztés

# 2.11. A hibák elkerülése: A környezet védelmi megoldásai

## 2.11.1. A programhibák elleni védelem szintjei

### A hardware védelmi funkciói

védelmi szintek

NX bit, memórialapok futtathatóságának beállítása

i386: nincs támogatva

amd64, sparc, powerpc, alpha: támogatott

processzor szintű virtualizáció

### Operációs rendszer beépített védelmi mechanizmusai

Felhasználói szintek

Többfelhasználós rendszerek

Fájl jogosultságok, ACL

partíciók csatolási opciói

Exec-shield

Gyenge cím véletlenszerűsítés

### Operációs rendszer kiegészítő védelmi mechanizmusai

SELinux, Apparmor, Grsecurity, PaX

### Fejlesztő környezet védelmi mechanizmusai

A nyelv biztonsági sajátosságai

Automatikus kódelemzők

Védelmi kiegészítő funkciók (pl. StackGuard, ProPolice)

### Fejlesztési módszertan védelmi funkciói

A CC garancia követelményei, mint segédanyag

## **Futtató környezet védelmi funkciói**

Apache biztonsági beállításai

PHP környezet biztonsági beállításai

Perl Tainted mód

### 3. Ajánlott olvasmányok

[1] Secure programming for Linux and Unix HOWTO - 5. és 11.2. fejezetek mindenképp, a többi ajánlott

<http://www.dwheeler.com/secure-programs/>

[2] Hogyan írjunk biztonságos programokat PHP nyelven

<http://www.cgisecurity.com/lib/php-secure-coding.html>

[3] OWASP - Open Web Application Security Project - minden, amit a webbiztonságról tudni érdemes

<http://www.owasp.org>

[4] A Google web biztonsággal foglalkozó lapja, előadások, tanulmányok

<http://code.google.com/intl/hu-HU/edu/security/index.html>

[5] MTA biztonsági tanulmány - 4.10. fejezet

[http://www.cert.hu/dmdocuments/MTA1\\_online.pdf](http://www.cert.hu/dmdocuments/MTA1_online.pdf)

[6] The WWW Security FAQ

<http://www.w3.org/Security/Faq/>

[7] CGISecurity.com - egy másik webbiztonsággal foglalkozó, nagyon alapos oldal

<http://www.cgisecurity.com/>

[8] A 25 leggyakrabban elkövetett fejlesztési hiba a SANS szerint

<http://www.sans.org/top25errors/>

<http://weblabor.hu/cikkek/munkamenetkezeles1>

<http://weblabor.hu/cikkek/munkamenetkezeles2>

<http://weblabor.hu/cikkek/phpbiztonsag>

## 4. Demonstrációk

### 4.1. Firefox

dns névfeloldás

### 4.2. Chrome, mint biztonsági probléma

Küldi az adatokat, ha kell, ha nem :(

### 4.3. User auth lehallgatása

tcpdump vagy Wireshark, és freemail.hu belépés titkosítás nélkül (lehetne szinte akármi)

### 4.4. Demó minta programocskák

### 4.5. A gagyiszerv.pl