Biztonságos programok fejlesztése és a web alapú rendszerek biztonsági sajátosságai

Vázlat a Nyíregyházi Egyetem
IT biztonság II. című tantárgyához

Állapot: szűk körnek szánt, itt-ott bővített vázlat, nagyon sokkal bővítendő

Szerző: Mátó Péter <mato.peter@gmail.com>

Verzió: v4.rc1 – 2016.05.12

Tartalomjegyzék

1. Web biztonság területe

1.1. Technikai oldalról

Web böngészők

Egyéb HTML megjelenítőt használó programok

- Mail kliensek
- RSS olvasók
- Mobil eszközök felülete és szoftverei
- Operációs rendszerek felületei és szoftverei
- IoT

Web szerverek

Web proxy-k

DNS szerverek

NTP szerverek

1.2. Logikai oldalról

Biztonságos kommunikáció

Internet bank

Üzenet kezelő oldalak (e-mail, fórumok)

Magánélet

Internetes zaklatás

Szülői felügyelet

DansGuardian

http://ubuntuforums.org/showthread.php?t=843 510

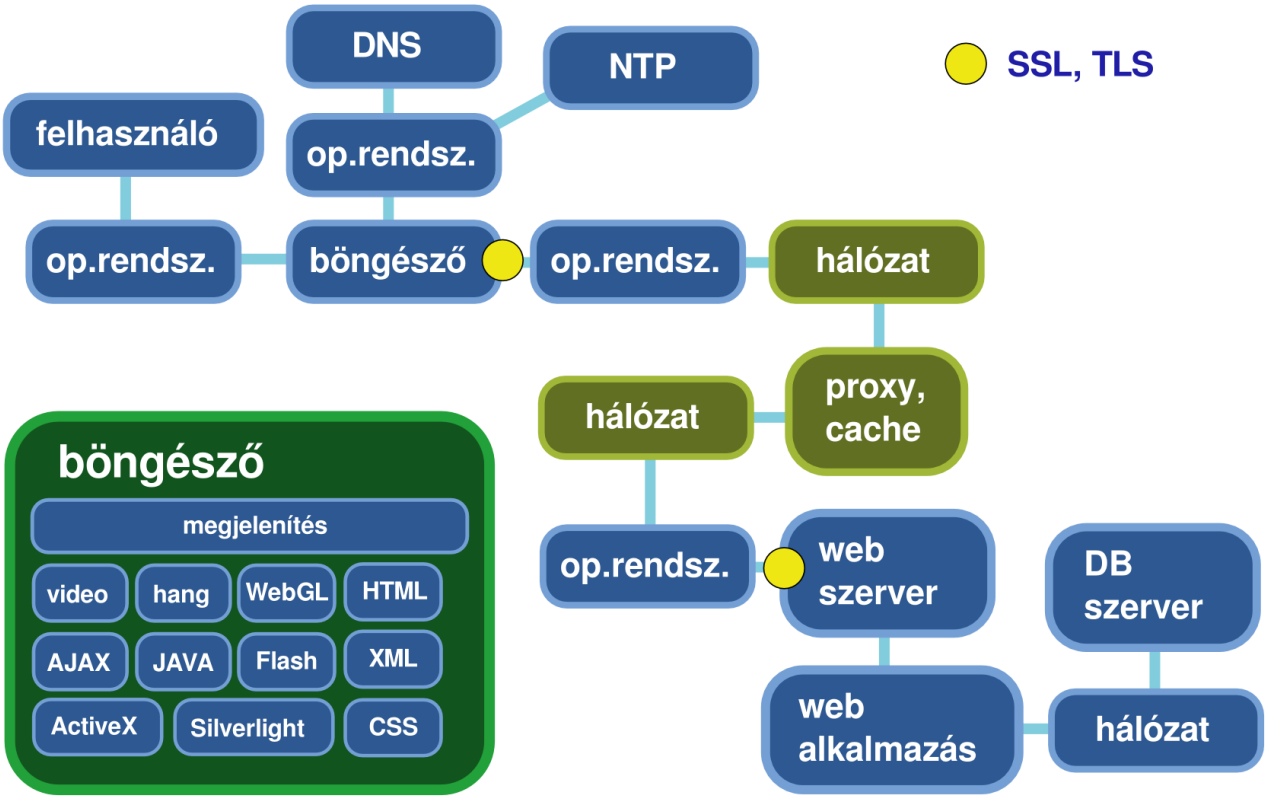
1.3. A web működése

Webszerver - Ha van az információs forradalomban barrikád, akkor a webszerver biztosan ott áll a tetején, az első sorban. A webszerver olyan program, amely HTTP segítségével adatokat szolgáltat a hozzá kapcsolódó klienseknek. Lehet monolitikus vagy moduláris felépítésű, általános célú vagy célszerver. Néhány pihent ember a sebesség érdekében még a Linux rendszermagba is beépített egy egyszerűbb webszervert (időközben a fejükhöz kaptak, és kivették). Viszonylag sok webszerver létezik, de ma a világot vitathatatlanul az Apache nevű szabad szoftver uralja. A világ webszervereinek 67.21%-a (2004. márciusi adat) Apache rendszert futtat, ami feltehetőleg kiemelkedő teljesítményének, tudásának, robusztus, biztonságos felépítésének köszönhető. A webszerver feladatából és a rajta átfolyó, gyakran értékes és bizalmas adatokból adódóan a támadók egyik legkedveltebb célpontja. A webszerverekre leselkedő veszélyekről és lehetséges elhárításukról lesz a legtöbb szó a fejezet további részében.

Egy fizikai szerver több virtuális szerver adatait is szolgáltathatja, egy webszerver feladatát nem ritkán igen nagy mennyiségű kiszolgálóból álló fürt látja el. Amennyiben egy weboldal kiszolgálását több fizikai gép látja el, a különböző gépek egymástól független részfeladatokat is elláthatnak.

A webszerver hagyományosan a 80-as porton várja a kéréseket, amelyen HTTP-n, nem titkosítva kommunikál. A HTTP-t nem egészítették ki TLS lehetőséggel, helyette használható a HTTPS protokoll, amely hagyományosan a 443-as porton figyel, ő a szabványos HTTP SSL-be csomagolt változata.

A webszerver kétféle forrásból veheti a kliensnek átadandó adatokat. Amennyiben az adatok előre el vannak készítve, akkor statikus tartalomról beszélünk, ha a webszerver a kérés pillanatában állíttatja elő valamilyen mechanizmus segítségével, akkor pedig dinamikus vagy aktív tartalomról. Biztonsági szempontból az első verzió lényegesen szerencsésebb, mivel ilyen esetben tartalmat előállító mechanizmus nem okoz biztonsági kockázatot, így csak a webszerver és a kiszolgálón lévő egyéb szolgáltatások esetleges hibáira kell felkészülni. Ezeknek a biztonsági hibáknak a kihasználása egy jól felépített rendszerben igen nehéz. A "Biztonságos szoftver futási környezet" fejezetben leírt lehetőségek tervezett, következetes használatával a szerverprogram és a kapcsolódó szolgáltatások fejlesztési hibáiból adódó kockázat minimalizálható. A rendszer biztonsága tovább fokozható egy megfelelően, failsafe módon beállított alkalmazás szintű tűzfallal, erről majd egy későbbi alfejezetben lesz szó.



A dinamikus tartalom olyan lehetőségeket ad, amelyek statikusan nem vagy csak komoly kényelmetlenségek árán lenne elérhető, ezért a világon folyamatosan növekszik a dinamikus lapok aránya. Ma már a legegyszerűbb lapot is dinamikus generátorral állítják elő, mondván a hírek adatbázisban való tárolása lényegesen leegyszerűsíti az oldalak szerkesztését. Ez kétségkívül igaz, azonban fontos megjegyezni, hogy a dinamikus lapok a legtöbb esetben (pl. a keresők tipikus kivételek) könnyen átalakíthatóak statikussá, csak előre le kell generálni a dinamikus tartalomgenerátor kimenetét. Erről még szólunk a későbbiekben. Mivel a dinamikus tartalom előállítását végző mechanizmusok számos igen komoly biztonsági probléma forrásai lehetnek, így az ezek által okozott kockázatokkal valamint a lehetséges védelmi megoldásokkal a későbbiekben részletesen foglalkozunk.

Web böngésző, browser vagy kliens - Amint azt korábban már leírtuk, minden valószínűség szerint a böngészők esztétikus és ergonómikus felületének is köszönheti a WWW gyors térhódítását. A böngésző nem más mint a kliens-szerver működésű HTTP kliens oldali része. Kapcsolódik a webszerverre és ő jeleníti meg a szerver által szolgátatott lapokat. Sokan hajlamosak megfelejtkezni arról a tényről, hogy a web böngészők is támadhatóak, nem kell hozzá más, mint egy rosszindulatú szerver adminisztrátor vagy egy rosszul megírt szerver, amely lehetővé teszi a felhasználóinak, hogy egymást megtámadják. Sajnos egyik eset sem ritka, így a böngészők biztonsági kérdéseivel is foglalkoznunk kell.

A kliens rendszerek biztonságának szempontjából talán a beágyazott objektumok megjelenítése jelenti a komolyabb veszélyt. Mivel a képeken kívül igen sok különböző beágyazott tartalom lehetséges, melyek sokaságára a böngészőket nemigen lehetne felkészíteni, így a komolyabb böngészők beépülő modul vagy plugin támogatást nyújtanak. Ez azt jelenti, hogy a tartalom MIME típusa alapján (amelyet a szerver közöl vagy biztonsági szempontból rosszabb esetben a kliens automatikusan felismer) a böngésző előveszi a szükséges megjelenítő modult, és annak segítségével mutatja meg a tartalmat. Ez a tapasztalatok szerint növeli a böngésző sebezhetőségét.

Web gateway vagy proxy - Alkalmazásszintű tűzfalak HTTP és HTTPS protokoll szűrésére kialakított egysége. A legtöbb proxy lehetővé teszi a cím, és felhasználó alapú hozzáférés vezérlést. A proxy-tól függ, hogy milyen szinten lehet a HTTP-t korlátok közé szorítani, erről bővebben írunk később. Lehet transzparens, ekkor a web böngésző nem igényel plusz beállítást. Valamilyen módszerrel a kérések be vannak terelve a proxy-ba, ami a kérésből vagy az eredeti célcímből eldönti, hogy a böngésző eredetileg hová akart kapcsolódni, és oda nyitja meg a szerver oldali kapcsolatát. Fontos tudni, hogy a transzparencia alatt általában azt szokás érteni, hogy a böngésző szempontjából a proxy átlátszó-e vagy nem. Viszonylag kevés tűzfal teszi lehetővé annak eldöntését, hogy a szerver szempontjából átlátszó-e egy proxy. Ennek akkor lehet jelentősége, ha a szervernek a naplózás vagy a statisztikák miatt szüksége van a kliensek eredeti forráscímére. Ha egy webproxy nem átlátszó, akkor a böngészőnek be kell állítani, hogy hol érhető el, vagy autoproxy-t kell használni. A proxy kifejezést gyakran használják a web gyorsítótárak megnevezésére is, de ez nem helyes, a gyorsítótár funkciója egészen más.

Web gyorsítótár (webcache) - Olyan szerver, amelyen keresztül megy a WWW forgalom, és ha egy olyan lapot kérnek tőle, amelyet már korábban is, akkor bizonyos körülmények között nem fogja a teljes lapot újra lekérni, hanem a már lekért és eltárolt adatot adja tovább. A gyorsítótár beállításaitól függ, hogy milyen esetben kéri újra az adatot. Megfelelően beállított web gyorsítótár jó esetben akár 40%-kal is csökkentheti a vonali terhelést. Hibás beállítások esetén a felhasználók azt tapasztalják, hogy a web gyorsítótár "ragaszkodik" az elavult tartalomhoz, és csak komoly küzdelem árán lehet rászedni, hogy frissítse a web gyorsítótárban tárolt lapot. A gyorsítótárak gyakran bizonyos tűzfal funkciókat is átvesznek (azonosítás, feljogosítás, fekete vagy fehér listás hozzáférés vezérlés stb.), de semmiképpen nem helyettesítik az alkalmazás szintű tűzfalakat, csak kiegészítik azokat. Nagy tudása, robusztussága és biztonsága miatt a leggyakrabban használt web gyorsítótár szabad szoftver, a neve Squid [squid].

1.4. A web ma

Aktív kliens oldal (GMail)

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

HTML5 a maga aktív képességeivel

WebGL

1.5. A web alapú rendszerek kommunikációja

HTTP (Hypertext Transfer Protokoll)

1.5.1. A HTTP legfontosabb jellemzői

Állapotmentes

Nyílt, lehallgatható

Szöveg orientált

Ellopható, módosítható - pl. Man in the middle attack

1.5.2. A protokoll általános működése

Szöveges, nem titkosított protokoll.

Legfontosabb metódusai: HEAD, GET, POST

A kódolásokról

1.5.3. A web alapú rendszerek kommunikációjának védelme

A HTTP szabvány nem ismeri a titkosítás fogalmát. A WWW születésekor ez nem volt szempont, hisz a protokoll fizikai publikációk megosztására lett kitalálva. Miután széles körben elterjedt, nyilvánvalóvá vált, hogy a bizalmas adatok átvitelét valamilyen úton meg kell oldani. A HTTP általánosan használt titkosított változata a HTTPS, ami nem más, mint egy hagyományos HTTP kapcsolat SSL kapcsolatba csomagolt változata. Létezik még egy Secure HTTP nevű protokoll, amely azonban nem terjedt el széles körben, a kliensek és szerverek általában nem ismerik.

A szerver választásánál figyelni kell rá, hogy ismeri-e a HTTPS protokollt, és ha igen, akkor milyen biztonsági jellemzőket lehet beállítani. Ha a szerver telepítésének pillatatában nincs szükség a HTTPS használatára, akkor is

szerencsés olyan szervert választani, amely lehetőséget ad a használatára, mert a tapasztalat azt mutatja, hogy a szerverek élete során általában szükségessé válik a HTTPS használata. Minimálisan el kell várni egy HTTPS-t beszélő webszervertől, hogy elő lehessen írni kötelező titkosítást egy domain-re, a kliensek kulcsainak érvényességét képes legyen minél több módon ellenőrizni (tanúsítvány aláírásának és érvényességi idejének ellenőrzése, CRL-ek használata, lehetőleg OCSP használata). Fontos, hogy a kliensek kulcsának ellenőrzését kötelezővé lehessen tenni. Igazán kifinomult SSL motorral beállíthatók az elfogadható titkosító algoritmusok, így szükség esetén kizárható a kliens által ismert túl gyenge titkosító algoritmus használata. A titkosítást minden olyan esetben célszerű használni, amikor a felhasználók adatai (akár csak email cím) mozognak a böngésző és a szerver között. A dinamikus tartalom veszélyeinél erről még szólunk.

Amennyiben a szerver bizalmas adatokat tárol, akkor ügyelni kell rá, hogy ne csak a struktúra egy szintjére írjunk elő kötelező azonosítást és hozzáférés védelmet, mert a struktúrában mélyebben elhelyezkedő adatok közvetlen hivatkozással elérhetők maradnak. Ha tehát a szerveren előírjuk a /titok könyvtárra az azonosítást, és az azonosítatlan felhasználóknak tiltjuk a hozzáférést, akkor még nem lehetünk biztosak benne, hogy a titok könyvtárban lévő állományok által hivatkozott tartalom is védett. Ha az adott szint alatt nincs bekapcsolva a védelem, és onnan egy URL kiszivárog az Internetre (például megjelenik egy levelezőlista archívumában), akkor azt nagy valószínűséggel illetéktelenek végig fogják kutatni. Egy átmeneti hibából adódóan az adataink igen hosszú időre elérhetőek lehetnek, mivel a legnagyobb keresőrendszerek egy időre a lapok a tartalmát is eltárolják, így hiába javítjuk ki a szerver védelmét, a keresőben még hosszú ideig hozzáférhető a véletlenül kiszivárgott információ. Hasonló a helyzet a web gyorsítótár szerverekkel. Ha egy oldalt betáraztak, akkor az adat a cache-ből kinyerhető (ha nem is mindenki által, de a cache adminisztrátorok által mindenképpen).

Ezért különösen körültekintőnek kell lenni olyan webszerver építésénél, amely bizalmas információkat tesz elérhetővé. Lehetőség szerint egy web domain-en belül ne keverjük a különböző érzékenységi szintű adatokat, így a teljes webszerverre beállítható egy egységes hozzáférés vezérlési politika. Ezzel elkerülhető a véletlen félrekonfigurálás.

Lehetőség szerint a hozzáférést korlátozzuk hálózati szinten is, így az azonosító rendszer hibája nem okoz akkora gondot. Ha csak a jogosultak tartományaiból elérhető a webszerver (a hálózati hozzáférés vezérlést egy tűzfallal megoldva), akkor egy esetleges hiba esetén a potenciális támadók száma a töredéke a védelem beállítása nélkülinek.

1.5.4. Ismétlés: SSL, TLS

A szimmetrikus titkosítás (ismétlés)

Az aszimmetrikus titkosítás (ismétlés)

1.5.5. 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 ellenőrizése

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??

Ingyenes tanúsítvány kibocsátó szervezetek

Let's Encrypt - https://letsencrypt.org/

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 hozzáférésének szabályzása 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 valódiságáról 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

1.6. A felhasználó azonosítása és feljogosítása

1.6.1. Emberek és robotok megkülönböztetése

webspam, adatgyűjtés ellen

Egy egyszerű Turing teszt: a capcha-k

1.6.2. Identifikáció, authentikáció, authorizáció (ismétlés)

DAC, MAC

1.6.3. Azonosítási módszerek

Belső egyedi azonosító (természetes szám)

Lehetőleg e-mail cím a fehasználói belépésre

1.6.4. A jelszavakról általában

http://code.google.com/p/cryptsetup/wiki/FrequentlyAskedQuestions#5._Security_Aspects

Jelszó biztonság (hossz, karakterek)

Csatorna függő: lehallgathatóság

Jelszó tárolás (plain pro és kontra)

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

Jelszó generálása

Jelszó bekérése, cseréje

A HTTP protokoll saját azonosítási mechanizmusai

Realm fogalma

Challange-Response

Szerver:

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

Kliens:

```
Authorization: Basic YXR5YTptZWdhc3p1cGVydGl0b2s=
```

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...

1.6.5. 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.

1.6.6. Felhasználó kizárása (lock out)

A megtámadott felhasználó belépési limitjét kihasználja

1.6.7. 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

VB Script

Hasonló, mint a JS, de csak Windowson működik

JavaScript

Újabban felkapták, Web 2.0

AJAX

Az internetes Javascript-ek korlátozva vannak

Volt már hiba, amitől fájlokat tudtak manipulálni

1.6.8. Web kereső motorok által okozott veszélyek

Lenyomozhatóság

Bizalmas adatok kerülnek ki

a robotok meglelik

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

1.6.9. 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:

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

1.6.10. A webalkalmazás jogai és az oprendszer védelmi rendszere

A webalkalmazás felhasználója és csoportja(i)

A processzek közti kommunikáció

TCP/IP

szignálok

Az elérhető állományrendszer

1.6.11. Speciális veszélyforrások, új kihívások

Adobe Flash

Google Gears

Adobe Air

HTML5 – local storage, web sql

1.6.12. 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

https://addons.mozilla.org/hu/firefox/addon/edit-cookies/

1.6.13. 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.0TP.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.tvll.hu">http://www.tv11.hu</a>
```

IDN megtévesztés:

```
<a href="http://www.p%D0%B0ypal.com/">paypal.com</a>
http://www.utf8-chartable.de/unicode-utf8-table.pl?start=1024
```

A státuszsor támadása

1.6.14. Fizetés a hálózaton

2. A program biztonság alapjai

2.1. A programok támadásának módjai

A program működésének befolyásolása adatainak megváltoztatása által

A rendszerben lévő kód végrehajtása nem megfelelő sorrendben

Támadó kód bevitele és végrehajtása

3. Általános programozási hibák és elkerülésük

A programozási hibák két fő csoportba sorolhatók: a nyelvfüggetlen és a nyelvtől függő programozási hibákra.

3.1. A nyelvfüggetlen programozási hibák

Az első hibacsoport olyan, általában program logikai hiba, amely lehetővé teszi a program védelmi rendszerének kikerülését, vagy a program olyan tevékenységre kényszeríthető, amelyet a tervezők eredetileg nem szerettek volna (például olyan állományok megnyitása és továbbítása, amelynek semmi köze a programrendszer működéséhez).

3.1.1. A .. hibacsoport

Ilyen hiba a webszervereknél jól ismert .. (dotdot) hiba. A program ellenőrzi, hogy a hivatkozott állomány elérési útjának eleje a megengedett könyvtáron belül van-e, azt azonban nem veszi észre, hogy az operációs rendszerek állományrendszereinek sajátosságai miatt használható ".." hivatkozással a támadó az engedélyezett könyvtárból vissza tud lépni, így a támadó olyan állományokhoz férhet hozzá, melyek nincsenek a konfigurációban megadott könyvtárban. Az a legmulatságosabb, hogy nemcsak a szakirodalom írja le ezt a hibát számtalanszor, hanem a HTTP szabvány is megemlékezik róla, és ennek ellenére rendszeres visszatérő látogatója a biztonsággal foglalkozó levelező listáknak, mintha a szabvány azt írná, hogy a webszerverek első verziójába kötelező implementálni. Feltehetőleg a webszerverek fejlesztői nemigen jutnak el a szabvány olvasásában a "Biztonsági figyelmeztetések" fejezetig. Sajnos.

3.1.2. Átmeneti állományok kezelési hibái

Egy másik tipikus nyelvfüggetlen hiba az átmeneti állományok hibás kezelése. Helytelenül kezelt átmeneti állományokkal a fejlesztő szándékai ellenére információ szivároghat ki a rendszerből, vagy szélsőséges esetben akár a rendszer más részeinek működését is befolyásolni, bénítani lehet. Szerencsére csak helyben támadható, tehát csak ha valaki jogosultságot szerzett a rendszerhez. Ezek a problémák ritkán detektálhatók, orvosolhatók automatikus eszközökkel. Egyes programozási nyelvekhez kínálhatnak bizonyos hibák elkerülésére alkalmas eszközöket (pl. perl -T [perltaint], RaceGuard [raceguard], fordítók szemantikai elemző képességei /nem inicializált változó.../), de ezek csak a problémák egy nagyon kicsiny részére nyújthatnak megoldást. Ennek a problémacsoportnak a megoldása kizárólag a fejlesztők továbbképzésével oldható meg.

3.1.3. 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 = $_GET['year'];
if (!preg_match("/^[0-9]{1,2}$/", $month)) die("Bad month, please re-enter.");
if (!preg_match("/^[12][0-9]{3}$/", $year)) die("Bad year, please re-enter.");
exec("cal $month $year", $result);
print "<PRE>";
foreach ($result as $r) { print "$r<BR>"; }
print "</PRE>";
```

3.1.4. 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

3.1.5. 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.

3.1.6. Abszolút út bejárás (Absolute Path Traversal)

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

helyett

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

3.1.7. Kódolási problémák, élő nyelv-specifikus hibák

Betű szerinti rendezés módosulása

3.1.8. 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
```

3.1.9. 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)

3.2. Nyelv- és platformfüggő programozási hibák

Védelmi szempontból a nyelvfüggő programozási hibák lényegesen érdekesebbek, mert sok esetben megelőző módszerek segítségével meg lehet akadályozni a kihasználásukat. Mivel a hálózati rendszerek fejlesztésének tipikus programozási nyelve a C, és mivel ez a fejezet csak egy rövid bevezető a programozási hibákba, ezért most csak a C nyelv tipikus hibái közül nézünk meg egyet, amely az elmúlt évek legtöbb programhibájáért volt felelős. A hiba neve: puffer túlcsordulás (buffer overflow, becenevén BOF). Sok alfaja létezik, a hibát kihasználó támadási módok elemzéséről írt tanulmányokkal egy kisebb könyvtár megtölthető lenne.

```
int buffer_overflow_me(char *str)
{
    char buffer[20];

    strcpy(buffer, str);
    printf("Buffered data: %s\n", buffer);

    return 0;
}
```

1. példa: C programozási nyelvű függvény tipikus hibával

A hiba lényege, hogy a C nyelvben a hívott függvények lokális adatait a rendszer olyan memóriaterületen tárolja, amely közel van a függvény visszatérési címének tárolt értékéhez. Amennyiben a fejlesztő nem ellenőrzi, hogy egy pufferbe mennyi adatot olvas be, ahogy ez az 1. példa állatorvosi ló függvényénél látszik, akkor elképzelhető, hogy a gonosz támadó a pufferbe támadó programrészletet (shellcode) tud bejuttatni, és a memória további területének átírása során a visszatérési cím is felülíródhat. Bizonyos C könyvtári függvények (pl. strcpy, sprintf stb.) nem ellenőrzik, hogy mennyi adatot írhatnak egy memóriaterületre, így ezek használata esetén a támadó elégséges méretű kódot és egyéb adatot juttathat be a rendszer sikeres megtámadásához. Ha a visszatérési címet a támadónak sikerül úgy módosítani, hogy az a pufferbe korábban bejuttatott kódra mutasson, akkor máris a hibás programot futtató felhasználó jogosultságaival rendelkezik, és kicsit sarkítva az általa bejuttatott programmal a megtámadott felhasználó nevében azt tesz, amit akar. Ha nem is képes a vezérlést átvenni, még lehetősége lehet a program működésének befolyásolására.

Amennyiben a program biztonsága kritikus szempont, akkor szerencsés ezeknek a függvényeknek a használatát messze elkerülni, ez azonban egy már létező rendszernél nem lehetséges.

A példa kedvéért nézzünk meg egy Perl nyelvű tipikus hibát is. A nyelvre jellemző, hogy scriptnyelv és ördögi keveréke az egyéb programozási nyelvek hasznosnak ítélt darabjainak. A perl hírhedt hibája a varázslatos open (magic open). Azért hívják varázslatosnak, mert nagyon sokrétű. Lehetőséget ad fájlok megnyitására (írás, olvasás, átírás), külső program futtatására úgy, hogy a bemenetét a perl script adja vagy úgy, hogy a külső program kimenetét a script olvashatja. Azt, hogy éppen mit szeretne csinálni a felhasználó, az open függvény úgy dönti el, hogy a megnyitandó szöveg tartalmaz-e cső jelet (|), és ha igen, akkor hol. Amennyiben a szöveg elején talál egyet, akkor elindítja a megadott külső programot, és visszaad egy csak írható fájlkezelőt (file handle), ha a szöveg végén, akkor hasonlóan, csak a fájlkezelő csak olvasható lesz. Mi lehet ezzel a probléma? Nézzünk egy egyszerű kis perl függvényt a 2. példában.

```
sub please_hackme_by_wrong_chars
{
    print("Give me the filename with path: ");
    my $filename = <>;

    open(FILE, "$filename");
    my @contents = <FILE>;
    close(FILE);

    print(join(", ", @contents), "\n");

    return 0;
}
```

2. példa: Perl nyelvű függvény tipikus hibával

Jól látható, hogy a fejlesztő vakon megbízik a felhasználóban, és nem ellenőrzi le, hogy milyen szöveget visz be a $filename változóba. Ha a felhasználó elég szemfüles, akkor rájöhet, hogy tetszőleges állomány tartalmához hozzáférhet a program nevében, hisz a megnyitandó állomány neve nincs ellenőrizve. Kis töprengés után a támadó feltehetőleg arra is rájön, hogy ha állománynév helyett egy parancsot ad meg cső jellel lezárva (|), akkor azt a program lefuttatja, és annak kimenetét írja ki. Így a támadó közvetlenül programot tud futtatni az ellenőrizetlen bevitel miatt.

Számos ilyen, gyakran elkövetett biztonsági hiba lehetősége rejlik szinte minden programozási nyelvben. Sajnos sok esetben csak a programozók továbbképzése ad teljeskörű megoldást ennek a hibaforrásnak a megszűntetésére, itt azonban több esetben, automatikus megoldás is lehetséges.

Ilyen esetben sem kilátástalan a helyzet. Meg lehet annyira nehezíteni a támadók dolgát, hogy a rendszer feltörése annyiba kerüljön (idő és pénz), hogy ne érje meg a befektetett munkát. Megakadályozható, hogy a támadó bizalmas adatokhoz férjen hozzá, továbblépjen a rendszerről vagy egyéb illegális tevékenységre használja azt. Ezekről a megoldási lehetőségekről szól a fejezet további része.

3.2.1. A C nyelv sajátos hibái

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

puffer túlcsordulás / stack v buffer overflow, overrun

halom túlcsordulás / heap overflow

dupla felszabadítás / double free

formázó sztring hiba / format string bug

return to libc

3.2.2. Szkriptnyelvek (php, perl, python...)

include

magic open

3.2.3. 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

Az adatok XML reprezentációja

XPath injection

4. Jó tanácsok jó programokhoz

A következőkben olyan jó gyakorlatokat gyűjtöttem össze, melyek nagy segítséget adnak a megfelelő minőségű és biztonságú programok fejlesztéséhez.

4.1. Célunk a programozás minőségével kapcsolatban

Helyes működés

Stabilitás

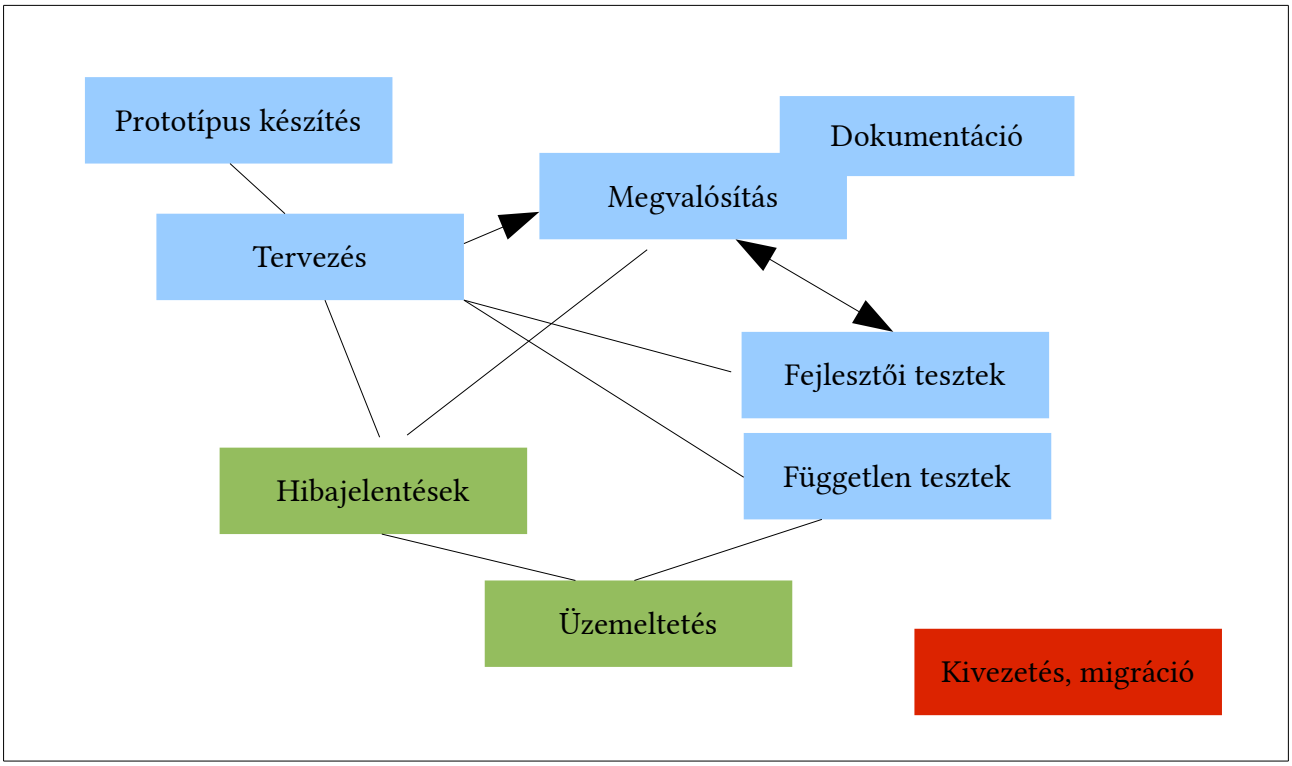
Biztonságos működés

4.2. A programozási projekt életútja

Prototípus

Tervezés

Megvalósítás



4.3. Hibatűrő megvalósítás

Inkább lépjen ki, mint hibásan működik

Hibatűrő megvalósítás – pl.: try catch finally

4.3.1. 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 terv rétegei megfelelnek egymásnak

4.3.2. Fejlesztői minőségkultúra

4.3.3. Megfelelő programnyelv kiválasztása

Fontos szempontok

- hozzáértés
- átláthatóság, érthetőség
- egységbe zárás
- információ elrejtés
- szigorú típusosság

4.3.4. 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

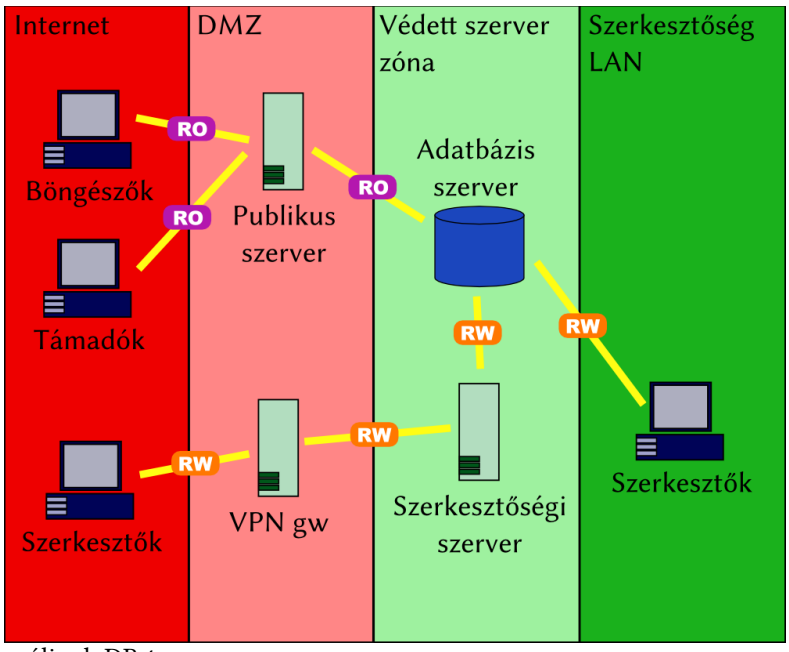
4.3.5. Erre kialakított fejlesztési módszertan

Általános biztonságos programozási módszertan: a CC szabvány

Common Criteria (ISO/IEC 15 408:1999)

Egy adott feladatra fejlesztett programban a hibák számát a fejlesztés alatt jól definiált módszerekkel csökkenteni lehet, elvileg akár nullára. Erre a problémára nyújt megoldást például a CC (Common Criteria) [cc], amely ma már nemzetközi szabvány. A CC egy nagyszerűen felépített rendszer, amely kikényszeríti, hogy a fejlesztők minden felmerülő fenyegetettséget felmérjenek, és a fejlesztési terveket ez alapján készítsék el. A gond az, hogy egy program CC szerinti fejlesztése nagyon komoly költségekkel jár, ami természetesen a fejlesztő céget terheli. Ez tehát ellene hat a használatának. Mivel azonban ma már biztonsági területen gyakran elvárják a felhasználók, hogy a biztonságos szoftverek rendelkezzenek CC minősítéssel, így a fejlesztő cégek a profitorientáltság miatt gyakran kompromisszumot hoznak, és egy gyenge követelményeket kitűző PP (Protection Profile) és/vagy ST (Security Target) alapján tervezik meg a TOE-t (TOE: Target of Evaluation), ezzel elérve az elsődleges célt, az EAL4-es (EAL: Evaluation Assurance Level) CC minősítést, azonban kikerülve a valódi feladatot: egy biztonságos rendszer fejlesztését. Érdemes egy pillantást vetni a CC szerint minősített biztonsági szoftverek ST-jére. Sajnos gyakran az látszik rajtuk, hogy elkészítésüknél nem a biztonság fokozása, csak a papír megszerzése volt a cél.

4.3.6. 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

4.4. A hibák elkerülése: A beérkező adatok ellenőrzése

4.4.1. 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

4.4.2. A beérkező adatok ellenőrzése

Lásd ajánlott irodalom [1]

Hossz ellenőrzés

Karakter osztály ellenőrzése

Reguláris minta illesztés

4.5. A hibák elkerülése: A környezet védelmi megoldásai

4.5.1. 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

Fálj 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

5. 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

[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/

Egyebek:

http://weblabor.hu/cikkek/munkamenetkezeles1

http://weblabor.hu/cikkek/munkamenetkezeles2

http://weblabor.hu/cikkek/phpbiztonsag

http://php.net/manual/en/security.php

http://code.google.com/p/cryptsetup/wiki/FrequentlyAskedQuestions#5._Security_Aspects

6. Demonstrációk

6.1. Firefox

dns névfeloldás

6.2. Chrome, mint biztonsági probléma

Küldi az adatokat, ha kell, ha nem :(

6.3. User auth lehallgatása

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

6.4. Demó minta programocskák

6.5. A gagyiszerv.pl