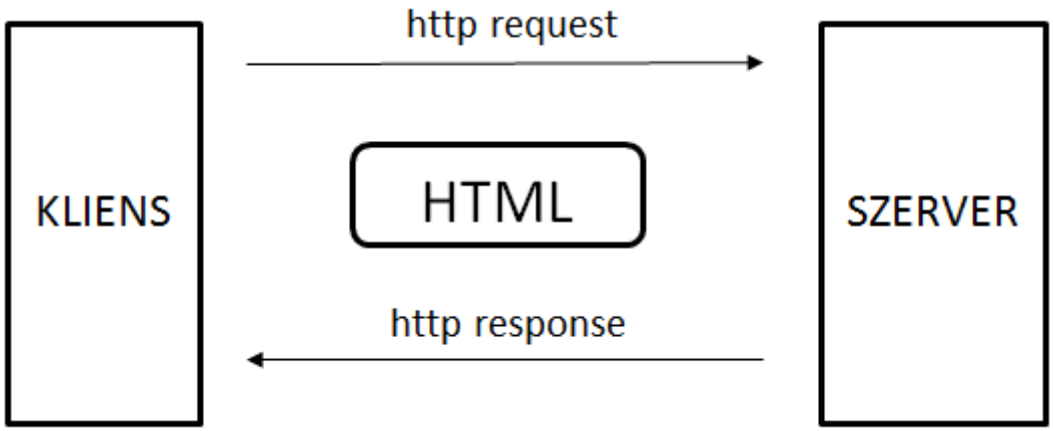
4. Egy webalapú szoftvert többfelhasználós működésre kell kialakítani. Tervezze meg a webalapú szoftver többfelhasználós működését, definiálja a működéshez szükséges komponenseket, és alakítsa ki a szükséges biztonsági megoldásokat!

– Ismertesse a webalapú rendszerek felhasználókezelési megoldásait és jellemzőit!
– Ismertesse a jelszókezelésre és tárolásra vonatkozó alapelveket!
– Mutasson be - egy webalapú szoftver esetében - egy teljes körű felhasználókezelési megoldást!
– Ismertesse a leggyakoribb felhasználói fiókok elleni támadási módszereket, és tegyen javaslatot az ellenük való védekezésre!

HTTP protokoll felépítése, működése:



A HTTP (HyperText Transfer Protocol) egy információátviteli protokoll a világhálón. Az eredeti célja a HTML lapok publikálása és fogadása volt. A HTTP egy kérés- válasz alapú protokoll kliensek és szerverek között. A kommunikációt mindig a kliens kezdeményezi. A HTTP klienseket gyűjtőnéven user agent-nek is nevezik. A user agent jellemzően, de nem feltétlenül web böngésző. A HTTP protokoll az OSI vagy a TCP/IP rétegmodell esetében is az alkalmazási rétegben helyezkedik el.

A HTTP kérés:

Egy HTTP kérés első sora mindig egy **nyitósor**, amely megadja a metódust, amellyel a kérés érkezett, a forrás elérési útvonalát és az aktuálisan használt HTTP verzióját, például

GET /images/logo.gif HTTP/1.1. vagy POST /feldolgoz/feldolgoz.php HTTP/1.1

Ezt a sort követheti tetszőleges számú **header** sor, **paraméter: érték** alakban, például így (böngésző küldi a kiszolgálónak):

User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36

és/vagy:

Accept: text/plain, text/html

Accept-Language: hu

A header sorok végét egy üres sor jelzi, melyet az opcionális üzenettest követ. A sorokat a **CRLF** (azaz kocsi vissza + soremelés) karakterpárral kell elválasztani. A headerek végét jelző üres sor csak ezt a két karaktert tartalmazhatja, nem lehet benne szóköz és tabulátor sem.

Majd az **opcionális üzenettest** következik.

HTTP protokoll többféle metódust definiál, amelyek a megadott erőforráson végzendő műveletet határozzák meg. Két legismertebb ezek közül:

GET : A megadott erőforrás letöltését kezdeményezi. Ez messze a leggyakrabban használt metódus. Nem kapcsolódik űrlaphoz, könyvjelzők közé felvehető, gyorsítótárazható, csak ASCII karakterek küldhetők (tehát bináris adatok nem) és maximalizált az így küldhető adatmennyiség (kb.2000 karakter). Amennyiben GET metódussal küldünk adatokat a kiszolgálónak, ezek az URL-en keresztül kerülnek átadásra, így érzékeny adatok küldésére nem javasolt.

POST : Feldolgozandó adatot küld fel a szerverre. Például HTML űrlap tartalmát. Az adatokat a kérésben lévő az üzenettest tartalmazza. Nem gyorsítótárazható, könyvjelzők közé nem vehető fel, bináris adatok is küldhetők (pl.fájlfeltöltés) és tudjuk a konfigurációs fájlokban módosítani a küldhető adatok méretét/mennyiségét (php.ini). A POST metódus mindig form elemhez kötődik.

A HTTP válasz:

A válasz első sora a **státuszsor/állapotsor**, amely verzió - státuszkód - indoklás alakú. A státuszkód egy három számjegyből álló szám, az indoklás egy angol nyelvű üzenet. Az előbbit inkább gépi, az utóbbit inkább emberi feldolgozásra szánták. Megadja az aktuálisan használt HTTP verzióját is. pl.:

HTTP/1.1 200 OK

A státuszkódok jelentését az RFC 2616 tartalmazza részletesen:

1xx: Informatív – Tájékoztató jellegű, pl.kérés megkapva.

2xx: Siker – Sikeres válasz. A kérés megérkezett; értelmezve, elfogadva.

3xx: Átirányítás – A kérés megválaszolásához további műveletre van szükség.

4xx: Kliens hiba - A kérés szintaktikailag hibás vagy nem teljesíthető (pl. 404 hiányzó elem).

5xx: Szerver hiba - A szerver nem tudta teljesíteni az egyébként helyes kérést (túlterhelt, hibásan kialakított kód).

A státuszsor után **header** sorok következhetnek a HTTP kérésnél látott módon paraméter:érték alakban. Pl.

Date: Mon, 23 May 2016 22:38:34 GMT;

Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) ;

Content-Type: text/html;

Charset=UTF-8

A header sorokat itt is üres sor zárja (CRLF – kocsivissza+soremelés), melyet az opcionális üzenettest követ.

Ha a státuszkód hibára utal, akkor a kliens megjelenítheti a hibaüzenetet, hogy tájékoztassa a felhasználót a hiba természetéről.

A HTTP/0.9 és 1.0 verziókban a kapcsolat egy kérés-válasz után lezáródik. A HTTP/1.1 verzióban bevezettek egy mechanizmust a kapcsolat életben tartására, így a kapcsolat újra felhasználható további kérésekhez. A kapcsolat életben tartását hívják idegen szóval perzisztenciának, az ilyen kapcsolatot pedig a perzisztens jelzővel illetik.

A HTTPS protokoll szintaktikailag megegyezik a HTTP-vel, de jelzi a böngészőnek, hogy használni kell az SSL/TLS titkosító réteget az adatforgalom védelme érdekében. Az SSL (Secure Sockets Layer) különösen célszerű a HTTP esetében, mert akkor is nyújt némi védelmet, ha csak a kommunikáció egyik oldala hitelesített. A szállítási réteg és az alkalmazási réteg között helyezkedik el, így még egy réteget ad a protokollveremhez. Az SSL módosítja a HTML alkalmazástól kapott adatot (feldarabolja, tömöríti, hasheli, titkosítja) és csak ezt követően adja át a szállítási rétegnek, hogy eljuttassa a célpontba.

A HTTP (és HTTPS) egy állapot nélküli protokoll. Az állapot nélküli protokollok előnye, hogy a szervernek nem kell nyilvántartania felhasználói információkat az egyes kérések kiszolgálása között.

Munkamenet kezelés (session-kezelés)

A HTTP/HTTPS protokoll nem teszi lehetővé számunkra, hogy megállapítsuk vajon két különböző kérés ugyanattól a felhasználótól érkezett-e. Ezért hívják a HTTP/HTTPS protokollt állapot-mentes protokollnak. Két egymás után érkezett kérés származhat ugyanattól a felhasználótól is, de a világ két különböző pontján lévő felhasználótól is. Így a felhasználók azonosításához valamilyen más módszert kell alkalmaznunk.

Amennyiben egy adott weboldalon végbemenő munkamenet alatt képesek vagyunk a felhasználót azonosítani és nyomon követni, akkor lehetővé tudjuk tenni, hogy a jogosultsági szintjének megfelelő tartalomhoz férjen hozzá, illetve ennek megfelelő műveleteket hajthasson végre az oldalon.

Munkamenetek használatával lehetővé válik, hogy a kiszolgáló oldalon azonosítani tudjuk a felhasználót és ellenőrizni tudjuk, hogy különböző kérések ugyanattól a felhasználótól

érkeztek-e. Meg kell különböztetnünk az **authentikáció** és az **authorizáció** fogalmát. Az előbbi azt jelenti, hogy a felhasználó azonosítja magát, tehát például a weboldalunkon megadja a felhasználónevét és jelszavát a belépéshez. Az authorizáció ezzel szemben egy folyamat, amely során azt ellenőrizzük, hogy egy felhasználó milyen tartalmakhoz, funkciókhoz fér hozzá a jogosultságai alapján (pl. rendszergazdai jogosultsággal rendelkezik-e és kezelheti-e egy webáruház megrendeléseit stb.). Ez utóbbit szükség esetén akár többször is el kell végezni.

A munkamenet megvalósítása az alábbi lépésekből áll php esetében:

1.**A munkamenet indítása**. Szükség lesz egy olyan egyedi azonosítóra, amit a kiszolgálón generálunk és átadunk a kliens oldalra. Majd egy újabb kliens kérés küldésekor a kliens ezt az azonosítót visszaküldi a kiszolgálóra.

PHP-ben a munkameneteket indíthatjuk a ***session_start()*** függvény hívásával. Ekkor a függvény ellenőrzi, hogy van-e folyamatban munkamenet, ha nincs, akkor létrehoz egyet.

A munkamenet indításával létrejön egy munkamenet azonosító, ún. ***session ID***. A munkamenet élettartama alatt a kliens oldalon tárolódik el. Ez a tárolás megoldható az alábbi módokon:

- süti segítségével tároljuk el a kliens oldalon,
- átadható a kiszolgálónak URL-en keresztül (GET metódussal minden egyes kérés URL-jéhez hozzáfűzzük),
- rejtett (hidden) űrlap mezővel is tárolhatjuk.

Ezek alkalmazásának biztonsági kérdéseit minden esetben meg kell fontolni és sajnos mindegyik módszernek van hátránya is. Pl. az url-hez könnyen hozzá lehet férni vagy a felhasználó az érvényes sessionID-t tartalmazó url-t küldi el valakinek, aki ezt felhasználva visszaélhet vele. Ezen kívül az is probléma ebben az esetben, ha elmentünk magunknak egy linket amikor be vagyunk jelentkezve (az aktuális sessionid-vel együtt), később, miután már egyszer kijelentkeztünk az oldalról a linkre kattintva nem azt látjuk amire számítunk, hiszen az url-ben lévő sessionid közben lejárt.

A rejtett űrlap mező használata nem mindig megoldható, hiszen minden alkalommal űrlapot kellene alkalmazni, illetve szintén manipulálható a kliens oldalon (böngészőben megtekinthetjük az oldal forráskódját, ahol látható a rejtett űrlapmező is, illetve a benne lévő sessionid is).

A sütik hátránya, hogy letiltható, így amennyiben a felhasználó nem engedélyezi őket a böngészőjében, akkor sütiben nem tudjuk eltárolni a munkamenet azonosítót.

Összességében a legjobb megoldás, ha elődlegesen sütiben próbáljuk meg eltárolni a munkamenet azonosítót, és ha a felhasználó nem engedélyezi a sütik használatát, akkor próbálkozunk egyéb módokon. A Php esetében mindig a php.ini-ben kell megnézni, hogy a süti létrehozása engedélyezett-e (alapértelmezetten igen), és sütik hiányában engedjük-e az URL-en keresztül átadni a sessionID-t (engedélyezni kell mert alapértelmezetten kikapcsolt).

2.**Munkamenet változó regisztrálása**. A munkamenet folyamatához létezik egy külön szuperglobális változó, ez a **$_SESSION** tömb. Ebben tudjuk eltárolni a kiszolgáló oldalon a felhasználó munkamenetéhez szükséges egyéb adatokat. Létrehozzuk a tömbnek egy új elemét és adunk neki értéket, pl. $_SESSION[„felhasznalo”] = felhasznalonev; A munkamenet végéig használható és segítségével ellenőrizhető, hogy a felhasználó be van-e jelentkezve.

3.**Munkamenet változók használata**. Ahhoz, hogy a létrehozott munkamenet változót használni tudjuk, szükséges a session_start() függvény meghívása. Ez teszi lehetővé, hogy a $_SESSION szuperglobális tömbből elérjük a szükséges adatokat (a munkamenet változót/változókat). Tehát minden olyan kód elején(!!) meg kell hívni a session_start() függvényt, ahol munkamenetet használunk, különben megszakad a folyamat.

4.**Munkamenet megszüntetése**. Egyik lehetséges mód, ha manuálisan töröljük a munkamenet változót: unset($_SESSION[„felhasznalo”]) és felszabadítjuk a munkamenet azonosítót a session_destroy() függvényt meghívva. A php.ini fájlban lehetőségünk van arra, hogy előre meghatározzuk a munkamenet érvényességi idejét másodpercben kifejezve. Amennyiben ez az idő letelik, a munkamenet törlődik.

Web alkalmazások biztonsága

Felhasználó által bevitt értékek szűrése:

Általános szabályként elmondható, hogy soha nem elég feltételezni, hogy egy űrlapból származó érték a vártaknak megfelelő értékek valamelyike lesz - minden esetben ellenőrizni is kell ezt. Az ellenőrzés történhet a front-end és a back-end oldalon is. Javasolt mindkét oldalon ellenőrizni.

A front-end oldalon történő ellenőrzés felhasználóbarát(abb), ráadásul a felhasználó sokkal hamarabb (akár rögtön) kap egy visszajelzést. Hátránya, hogy a front-end kód a böngésző fejlesztői eszköztárában megtekinthető és manipulálható. Front-enden ellenőrizhetünk HTML5 attribútumok segítségével is (required, pattern, minlength, maxlength stb), ekkor a böngésző maga blokkolja az űrlap küldését, amennyiben nem megfelelően kitöltött elemeket tartalmaz. Amennyiben a mezők a vártaknak megfelelően kerültek kitöltésre, az űrlap valid, ha nem, akkor invalid lesz. Visszajelzéseinket formázhatjuk CSS pszeudo osztályok segítségével is (pl.:required, :invalid, :valid). A HTML5 validálás kikapcsolható, ehhez az invalid attribútum használata szükséges.

Másik lehetőség, hogy valamilyen kliens oldali nyelv (pl. javascript) segítségével ellenőrizzük az űrlapot. Ez történhet a fent ismertetett módszerrel együtt, illetve a fenti módszert kikapcsolva is. Javascript segítségével lehetőségünk van arra, hogy a böngésző alapértelmezett validációhoz kapcsolódó visszajelzéseit felülírjuk. Rendelkezésünkre áll egy API is, amely a mai modern böngészők mindegyikében megtalálható. Ez az API (constraint validation API) hozzáférést biztosít a Javascript kódból a HTML5 néhány eleméhez (button,

input, select, textarea, fieldset és output) és különböző tulajdonságok segítségével validálhatjuk az űrlapot Javascript-ben (pl. validity.valid vagy validity.tooLong stb).

Használhatunk szabályos kifejezéseket, amelyekkel meghatározhatjuk, hogy milyen adatokat várunk a felhasználótól. A Javascript-ben a RegExp objektumhoz tartozik a test() metódus, amellyel megvizsgálhatjuk, hogy egy általunk meghatározott mintának megfelel-e a felhasználó által megadott érték.

A back-end oldalon történő ellenőrzés az alkalmazásunk utolsó biztonsági kapuja. Habár nem olyan felhasználóbarát, mint a front-end visszajelzés, nem szabad elfelejtkezni róla, hiszen ezek a kódok sokkal nehezebben hozzáférhetők és manipulálhatók a felhasználók által.

Először is tisztítsuk meg a beérkező karaktereket a fehérköz karakterektől, PHP-ban pl. a **trim()** segítségével. Ez a függvény a karakterlánc elejéről és végéről tünteti el a fehérköz karaktereket (pl. szóköz, tabulátor stb).

Alkalmazzunk **szabályos kifejezéseket** a back-end oldalon is. Nézzük meg, hogy a felhasználó által megadott karakterlánc a mintának megfelelő-e. Ehhez PHP-ben elérhető függvény a **preg_match().** Amikor a felhasználótól érkező adatokat adatbázisba mentjük, biztosnak kell lennünk benne, hogy nem tudnak rosszindulatú utasításokat küldeni és ezzel nem kívánt műveletet végrehajtani. Ennek kivédésére használhatjuk pl. a **mysqli_real_escape_string**() függvényt, amely az SQL-ben speciális jelentéssel bíró karakterek elé egy \ karaktert helyez, ezzel problémákat előzve meg. Illetve ugyanígy az **addslashes()** függvény is védőkarakterrel látja el a problémás karaktereket, a **stripslashes()** pedig eltávolítja a védőkaraktereket kiíratás előtt. A problémás karakterek pl. az idézőjelek, aposztrófok, perjelek, \n, \r karakterek stb.

A másik eset, ami ellen védekeznünk kell a HTML jelölők segítségével bejuttatott rosszindulatú utasítások (Cross Site Scripting). Például <script></script> tagek közé ágyazott utasítás bejuttatása egy input mezőn keresztül. Ennek kivédésére használjuk pl. a **htmlspecialchars()** függvényt vagy a **htmlentities()** függvényt. Ez utóbbi minden HTML entity-vel jelképezhető karaktert lecserél a karakterláncban.

Ellenőrizzük a felhasználók által megadott karakterláncok hosszát is, hogy védekezni tudjunk a túlcsordulások ellen. Természetesen nem lehetünk tisztában azzal, hogy pontosan mi a korlátja a változók hosszának, amely még nem okoz túlcsordulást, de a várt adatoknak megfelelően megbecsülhetünk egy értéket, aminél hosszabbat nem fogadunk el. PHP-ban a **strlen()** függvény használható, ám az ékezetes karakterek esetében helytelen eredményt kapunk általa. Ebben az esetben alkalmazzuk inkább az **mb_strlen(),** amelynek megadhatjuk a karakterkódolást is, ezzel az ékezetes karakterek esetében is a tényleges karakterszámot kapjuk vissza.

Webszerverünk és webalkalmazásunk biztonságossá tétele

A legbiztonságosabb alkalmazás is feltörhető, ha maga a kiszolgáló nem biztonságos.
Rendszerünk biztonságát a legegyszerűbben úgy tudjuk elősegíteni, ha gondoskodunk arról, hogy az általunk használt szoftvernek mindig a legfrissebb és legbiztonságosabb változata fusson.

Szánjunk időt arra, hogy alaposan áttanulmányozzuk a webszerver konfigurációs beállításait (Apache esetében a **httpd.conf** fájlt) beállításait. Természetesen erre akkor van lehetőségünk, ha mi magunk üzemeltetjük (vagy van felügyeleti jogunk) a webkiszolgálót és hozzáférünk annak httpd.conf fájljához.
Abban az esetben, ha ez nem így van (tehát hosting szolgáltatást veszünk igénybe) akkor a **.htaccess** használatával felül tudunk írni bizonyos beállításokat, amelyeket a saját webalkalmazásunkra szeretnénk érvényesíteni. A .hataccess beállítások mindig azon könyvtárra vonatkoznak, ahová fel vannak töltve és az ott található alkönyvtárakra. Ebben a fájlban rengeteg olyan beállítást adhatunk meg, amivel a saját alkalmazásunk biztonságát növelni tudjuk, illetve jobbá tehetjük az alkalmazásunkat. Néhányat kiemelve:

- saját, egyedi hibaüzenetek megjelenítése a felhasználónak a http protokoll szokásos hibaüzenetei helyett, pl. ha egy általa keresett fájl nem található vagy jogosulatlan kérést intéz a kiszolgáló felé stb.
- alapértelmezett nyitó fájl megadása (pl. ne az index.html vagy index.php nyíljon meg automatikusan
- könyvtár tartalmának listázása a böngészőben (vagy ennek letiltása vagy bizonyos kiterjesztésű fájlok listázásának letiltása stb.)
- fájlok hozzáférésének letiltása a weben keresztül (pl. include-olt fájlok stb.)
- hozzáférés letiltása bizonyos IP cím(ek)ről vagy domain(ek)ről (vagy épp tiltott mindenhonnan és csak bizonyos helyekről engedélyezzük)
- szebb és olvashatóbb URL-ek használata

Az adatbázisszerverek biztonsága

Az adatbázisok védelménél végig kell gondolni, hogy honnan, milyen módon érhető el az adatbázis. Az egyik lehetőség, hogy hálózaton keresztül férünk hozzá az adatbázishoz, közvetlenül vagy valamilyen alkalmazáson keresztül. A hálózaton keresztül történő elérésnek számos sérülékeny pontja van, amelyet a támadó kihasználhat. Az adatbázis szerverek adatbázis és konfigurációs állományaihoz a futtató operációs rendszeren keresztül is hozzá lehet férni.
Néhány fontosabb szempont:

Felhasználók és a jogosultsági rendszer: Nézzük meg, hogy az adatbázishoz kiknek van hozzáférése és ők milyen jogosultságokkal rendelkeznek. A rendszergazdán kívüli

felhasználóknak csak azokhoz az adatbázisokhoz és táblákhoz szabad hozzáférnie, és csak azokat szabad tudnia módosítani, amelyekkel ténylegesen dolga van. A GRANT utasítással adhatunk jogokat felhasználóknak vagy csoportoknak négy különböző szinten: globális, adatbázis, tábla és oszlop. A kiosztott jogosultságokat bármikor meg tudjuk változtatni.

A mintaadatbázisokat - amelyek teszteléskor jól jöhetnek - élesben történő használatkor célszerű törölni.
Az adattáblák mezőinek megfelelő adattípusokat válasszunk, illetve a mezők hosszát is korlátozzuk a várt adatoknak megfelelően. Például ha felhasználónevet tárolunk egy mezőben, akkor annak felesleges pl. 2040 karakternyi helyet hagyni.

Kapcsolódás az adatbázisszerverhez: Sok adatbázisszerver funkciói között megtaláljuk azt a lehetőséget, hogy titkosított kapcsolaton keresztül (általában a Secure Sockets Layer - SSL néven ismert protokollt használva) kapcsolódjunk hozzájuk. Ha bármikor is a nyilvános interneten keresztül kell adatbázisszerverhez kapcsolódni, titkosított kapcsolatot kell használni - amennyiben elérhető ilyen.

Webes alkalmazások esetén gyakran a kliens oldalon megadott adatok alapján hajtunk végre adatbázis műveleteket (pl. regisztráció esetében INSERT, adatok módosítása esetén UPDATE vagy egy felhasználó törölni szeretné magát egy nyilvántartásból DELETE).

Soha nem szabad megbízni a kliens oldalról érkező adatokban, tehát ezeket már eleve alaposan szűrni kell a korábban taglalt módszerekkel.

A webes alkalmazásunkon keresztül – ahol felhasználói adatok alapján hajtunk végre műveleteket - soha ne rendszergazdai jogosultságokkal csatlakozzunk az adatbázis kiszolgálóhoz. Legyen erre külön felhasználó az adatbázishoz és annak nevében történjék a csatlakozás és a műveletek végrehajtása.

Figyeljünk arra is, hogy kódjainkban megadásra kerül az adatbázishoz kapcsolódás esetében a felhasználónév és jelszó is. Ezeket mindenképp külön állományban tároljuk, aminek a hozzáférését korlátozzuk. A szükséges helyeken akár változókon keresztül adjuk át ezeket az értékeket.

Az alkalmazás tesztelésekor hasznos hibaüzenetek helyett éles használatkor már célszerű olyan visszajelzést megjeleníteni, ami a felhasználók számára nem árul el fontos információkat az adatbázisunkról és tábláinkról (pl. nem szerencsés egy olyan hibaüzenet megjelenítése, amelyben szerepel az adatbázisunk vagy valamely táblánk/mezőnk neve). Erre mindig nagyon figyeljünk oda a tesztelés után. Természetesen visszajelzést kell adnunk a felhasználónak, ha hiba történik, de nem mindegy milyen formában. A hibákat kezeljük le, illetve használjuk a log fájlokat a hibák monitorozására.

Jelszókezelésre vonatkozó szabályok, jelszavak tárolása

Hívjuk fel a felhasználók figyelmét arra, hogy ne szótári szavakat használjanak jelszónak, illetve ne engedjük meg, hogy a jelszó megegyezzen a felhasználónévvel. Ügyeljünk rá, hogy kellő hosszúságú jelszót válasszanak maguknak és a jelszó ne csak betűkből álljon (tartalmazzon számokat, egyéb karaktereket, kis-és nagybetűket is).

A felhasználók neveit és jelszavait mindig célszerűbb adatbázisban tárolni, mint egyszerű fájlokban. Így biztonságosabb és gyorsabb is a kezelésük.

A jelszavakat ne tároljuk egyszerű szövegként, ezzel szükségtelen biztonsági kockázatot vállalva. Hash (hasító) függvények használatával növelni tudjuk a jelszavak biztonságát. PHP-ben számos ilyen hash függvény elérhető, amelyeket alkalmazni tudunk (korábban pl. md5() – message digest, sha1() – secure hash algorithm, stb. Ma már inkább az sha256() vagy sha512() javasolt). Ezek nem teszik lehetővé a jelszavak visszafejtését akkor sem, ha a jelszavak illetéktelen kezekbe kerülnének. A kiválasztott hash függvénynek megfelelően pedig gondoskodjunk arról, hogy a tikosított jelszavak számára elegendő hely legyen az adatbázisban (pl. sha1 estén 40 karakternyi hely szükséges, valamely hash függvény esetében pedig ennél jóval többre van szükség). A hash vagy hasító függvények a bemeneti értékből egy megadott algoritmus alapján egy fix hosszúságú hasító értéket állítanak elő. Elvárás velük szemben, hogy működésük determinisztikus legyen, tehát az algoritmusuk csak a bemeneti érték bájtjaitól és azok sorrendjétől függjön.

Természetesen a szótári szavak feltörése a kiválasztott hash függvény ismeretében nagyon egyszerű, ezért a felhasználók jelszavait érdemes sózással még biztonságosabbá tenni. Ez azt jelenti, hogy egy általunk meghatározott konstanssal vagy algoritmus segítségével előállított véletlen karakterlánccal kiegészítjük a felhasználó által megadott jelszót, majd ezzel együtt adjuk át egy hash függvénynek.

Az is lehet egy megoldás, ha két vagy több hash függvényt választunk ki és úgy titkosítjuk le a jelszavakat.

Felhasználói fiókok elleni támadások

Szótármódszer: ez a módszer arra alapoz, hogy a felhasználók a saját nyelvükben előforduló szavakat használnak jelszónak. A támadók ugyanazzal az eljárással titkosítják a szótári szavakat, amellyel mi a jelszavakat, így össze tudják hasonlítani az így kapott karakterláncot. Mivel a hash függvények egy adott szót/karakterláncot mindig ugyanazzá a karakterlánccá alakítják át, így a szótári szavakat nagyon könnyen és gyorsan fel lehet törni ezzel a módszerrel. Természetesen, ha a támadók nincsenek a titkosítási módszerünk birtokában, nehezebb dolguk van, ám léteznek ún. szivárványtáblák, amelyek például szótári szavak többféle titkosítási eljárással előállított változatait tárolják, és ha a támadónak szerencséje van, a mi jelszavunk titkosított változata is közöttük lesz.

Nyers erő (brute force): A nyers erőt alkalmazó támadásnál egy gyors számítógép használatával kísérlik meg kitalálni a jelszavakat vagy visszafejteni egy titkosítási kódot. A támadó gyors egymásutánban kellően nagy számú lehetőséget próbál ki ahhoz, hogy

hozzáféréshez jusson vagy feltörje a kódot. A nyers erő (brute force) módszerét alkalmazó támadások szolgáltatás-megtagadást okozhatnak a rendkívül magas forgalom következtében egy meghatározott erőforrásnál vagy a felhasználói fiók zárolásával is járhat.

Általánosságban elmondható, hogy minél hosszabb jelszót választunk, annál több ideig tart az összes lehetséges kombinációt kipróbálni. Ahogy nő a jelszó hossza, exponenciálisan nő a lehetséges kombinációk száma. Illetve a hosszon kívül a számok és speciális karakterek használata szintén jócskán megnehezíti a jelszavak feltörését. A biztonság növelhető azzal is, ha bizonyos számú sikertelen próbálkozás után zároljuk a felhasználó fiókját, hogy meggyőződhessünk arról vajon támadás történt-e vagy a jogos tulajdonos zárta ki magát a rendszerből.