Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Danis Mihály

WEBES HÍRCSATORNÁKBAN ALKALMAZHATÓ NYOMKÖVETÉSI TECHNIKÁK VIZSGÁLATA

KONZULENS

Gulyás Gábor
György

BUDAPEST, 2011

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Danis Mihály**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2011. 12. 09.

.....

Danis Mihály

Összefoglaló

Az webkettő új, információ-vezérelt szolgáltatásai kétség kívül számtalan új lehetőséget hoztak az internet világába. Térnyerésüknek köszönhetően a tartalommegosztás vált a világméretű hálózat egyik fő mozgatórugójává. Az újdonsült funkciók előtérbe kerülése remek üzleti lehetségekkel kecsegtetett és kecsegtet a mai napig is, hiszen a felhasználókhoz hatalmas mennyiségű információ jut el nap mint nap.

Ez viszont nem csak előnyöket, hanem hátrányokat is hordoz magában. Sok szervezet, cég azzal a kizárólagos céllal jött létre, hogy nyomon kövessék a felhasználókat illetve a róluk megszerzett információkból profilokat építsenek fel, amelyet vagy saját maguk használnak fel, vagy pedig más cégeknek adnak el. Az internetező privátszférája - az ilyen típusú tevékenységeknek köszönhetően - egyre veszélyeztettebbé és sebezhetőbbé válik a támadásokkal szemben. Mivel a közeljövőben ez a sebezhetőség az új technikák létrejöttével csak fokozódni fog, ezért saját adataink védelmére egyre több figyelmet kell fordítanunk.

Sokféle webes nyomkövetést ismerünk, viszont ezeket a módszereket jelen szakdolgozat első részében a különböző webes hírfolyamok aspektusából vizsgálom majd meg. Először a hagyományos és az újabb webes nyomkövetési technikákat fogom bemutatni, majd rátérek a tesztelő alkalmazások jellemzésére is, végül a felépített szempontrendszer alapján fogom elemezni a hírcsatornákon alkalmazható nyomkövetési, és támadási technológiákat.

A szakdolgozat második részében a megfigyelésre alkalmasnak bizonyult módszerek webes jelenlétet fogom felmérni az összeállított hírfolyamok listája alapján. Arra leszek kíváncsi, hogy az oldalak milyen nyomkövetési technikákat alkalmaznak, és azokat milyen mértékben, ezután az eredményeket összesítem, következtetéseket vonok le belőlük, és végül lehetőségeket ajánlok a privátszféra védelmének maximalizálása érdekében.

Abstract

The new, information-driven services of Web 2.0 have brought many new opportunities to the world of internet. Due to their conquer, content-sharing has become one of the key moments of the worldwide network. The new features, that are coming to the front, had great business opportinities, and they still have to this day, because users receive huge amounts of information every day.

It brings not only advantages but disadvantages as well. Many companies and services have been created for the only reason to track users and build a profiles from the information obtained about them, and to either use it themselves or sold it to other companies. The privacy of surfing users - thanks to this type of activities - becomes more endangered and more vulnerable to attacks. Since this vulnerability will only intensify in the near future, with the born of new techniques, so the protection of our own data should get more and more attention.

There are many web-based and well-known tracking methods, however, in the first part of this thesis, I will examine these methods in the aspect of various web feeds. First, I am going to demonstrate the traditional and the new web-based tracking techniques too. Then I come to the characterization of test applications, and finally I am going to use the own-built criteria system to analyze tracking and attacking techniques of web feeds.

In the second part of the thesis I am going to estimate the presence of used methods on the feeds of the compiled list that are worth to check. I will find out what kind of tracking techniques are pages using, and to what extent are they doing it. After that, I will aggregate the results, draw inferences from them and recommend options to maximize the protection of privacy.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani Gulyás Gábornak, akihez a rendszeres konzultációkon illetve azokon kívül is bátran fordulhattam, ha segítségre volt szükségem; és aki hasznos ötleteivel, tanácsaival érdemben tudta támogatni jelen szakdolgozat elkészültét.

1 Bevezetés

Mai világunkban mindannyian rendelkezünk bizonyos alapvető jogokkal, viszont sok esetben nem is vagyunk vele tisztában, hogy ezek valóban megilletnek minket. Az egyik legfőbb ilyen jog a privátszféra védelme, azaz maga a tény, hogy ne legyünk kiszolgáltatva egyetlen kívülálló személynek sem. Az információs társadalom exponenciális fejlődésével és az internet megjelenésével tevékenységeink, személyes adataink nem is sejtjük, hogy milyen nagy veszélyeknek vannak kitéve. Az adatvezérelt Web 2.0-s szolgáltatások megjelenésével, az internet, tagadhatatlanul beleivódott a mindennapjainkba. A web interaktivitása, mobilitása és könnyen kezelhetősége miatt szinte már észre sem vesszük, hogy mennyi információt osztunk meg másokkal a különböző közösségi portálokon, videó- és képmegosztó oldalakon, fórumokon, blogokon, mikroblogokon vagy akár webes hírcsatornákon keresztül, és így sebezhetővé válunk a privátszférát érintő sokrétű webes támadások ellen.

Mi hasznuk van ezekből a támadóknak? A válasz egyszerű: hogy az internet adta új lehetőségeket kihasználva előnyre, pénzügyi haszonra, vagy hasonló, pozitív jellegű juttatásra tegyenek szert. Az évek során a magánszféra ellen irányuló támadási módszereket meglehetősen sokoldalúan alkalmazzák. Közöttük szerepelnek a hirdetők, webes áruházak, adatgyűjtők, a különböző felügyelő szervek és maguk az internet-szolgáltatók is [1]. A támadókat számos cél motiválhatja, amikor a felépített profilokat üzleti illetve haszonszerzési céllal használják fel. Többnyire a saját érdekeik, a saját szolgáltatásaik könnyebb hirdetése és eladása a cél, viszont ha a támadó komplett felhasználói profil-adatbázissal rendelkezik, akkor a birtokában lévő információkkal akár meg is károsíthatja a gyanútlan felhasználót. A támadók célja tehát elsősorban a profilkészítés, vagy más néven profilírozás: a lehető legteljesebb és legszélesebb körű profilok létrehozása, valamint a begyűjtött adatok felhasználásával a saját, és esetleg egyéb profilírozó szervezetek információs igényeinek kielégítése. A cégek döntő hányada, ha információgyűjtésről van szó, akkor nem csupán egy-egy konkrét információra kíváncsi, hanem minél próbál megszerezni. Ezeket az adatokat a különböző profilírozó cégek fel is használják főként marketing tevékenységekre -

ilyenek például a személyre szabott, célzott hirdetések, illetve a dinamikus árazás -, célzott adathalászatra (phishing-re), sőt akár identitáslopásra is.

Az ránk leselkedő veszélyforrások közé sorolhatjuk az XML alapú ún. RSS és Atom feed-ek adta lehetőségek kihasználásával történő támadásokat. Ezek az internetes hírcsatornák lehetővé teszik a felhasználóknak, hogy anélkül olvassák el a legfrissebb híreket, hogy az adott oldalakat meg kelljen nyitniuk, hiszen ezt a böngésző, vagy hírolvasó megteszi helyettük. A webes hírfolyamok a hírek rövid összefoglalóját tartalmazzák, az eredeti oldalra mutató linkkel, képekkel, videókkal és egy egyéb multimédiás tartalmakkal együtt.

Manapság a legelterjedtebb böngészők szinte mindegyike rendelkezik hírcsatorna-olvasási támogatással, vagy ha nem, akkor bővítmények telepítésével ez a funkció könnyedén elérhetővé válik. Napjainkban egyre népszerűbb a folyamolvasó weboldalak használata is. Az előzőekből következően nem túlzás kijelenteni, hogy hírek rendszeres letöltésével, azaz a feedek olvasásával, ha azokba valamilyen rosszindulatú szkriptet helyeztek el, akkor könnyedén célpointjai lehetünk az előbb említett webes támadásoknak. Viszont ami nagyon fontos: az információk gyűjtésére és feldolgozására a felhasználó tudta és legfőként beleegyezése nélkül kerül sor. Ez a fajta metódus megfosztja a híreket olvasó személyeket a saját adatai feletti tulajdonlás jogától, kiszolgáltatottá és sebezhetővé téve őket.

Az ilyen, privátszféra elleni támadási esetek ma már mindennaposnak mondhatók, ami arra ad okot, hogy azonosítsuk, felmérjük a ránk leselkedő veszélyeket, illetve különböző védelmi mechanizmusokkal, technikákkal felkészüljünk privát adataink védelmére. A jogi szabályozás nem a megfelelő ütemben követi az információs társadalom technológiai fejlődését. Hiába is tenné, hiszen ez még mindig nem lenne elegendő a probléma kezelésére. Ezért jöttek létre az ún. PET-ek, azaz Privátszféra Erősítő Technológiák, amelyek személyes adatok gyűjtésének, feldolgozásának és felhasználásának korlátozását és az érintett személy védelmét biztosító alapelvek, szabályok, eljárások, adatkezelés eszközök és módszerek összességét jelentik [2]. A PET módszerek segítségével hatékony kísérlet tehető az optimális adatvédelem biztosítására. A privátszféra biztonsága tehát egyre inkább közös érdekünk, amit mindenkinek szem előtt kell tartani és tenni annak érdekében, hogy szintje megfelelően magas maradjon.

2 Technikák és eszközök bemutatása

A weben ránk leselkedő potenciális veszélyforrásoknak alapvetően három különböző típusát ismerjük: az információs szuperhatalmak szolgáltatásain belüli információszerzés, a publikus adatforrások alapján történő profilírozás és a nyomkövetéses profilírozás [3]. A mi szempontunkból főként a legutóbb említett profilépítési technikák, eszközök lesznek fontosak, így a továbbiakban ezeket fogom részletesen taglalni.

2.1 Hagyományos webes nyomkövetési technikák

A profilírozó cégek legfőbb célja, hogy minél szélesebb körű profilokat építsenek fel a felhasználókról. Ezeket a profilokat vagy saját maguk használják fel, vagy pedig eladásra kínálják más szervezeteknek, ezáltal minden személyre szabhatóvá válik: a reklámok, webshopok árai, akciós csomagjai stb., amelyre az adott weboldalon történő felhasználói tevékenységek illetve a weboldalak közötti mozgások megfigyelése és elemzése nyújt lehetőséget [1]. Így tiszta kép kapható arról, hogy a felhasználó éppen melyik pillanatban melyik linkre kattint, azaz melyik hírt olvassa el, melyik terméket nézi meg, vagy melyik az a hivatkozás, amit meg sem nyit. Más módszer is adódik a megfigyelésre: az internetező tevékenysége valamely mértékben rekonstruálható a kiszolgálónál, vagy a felhasználónál keletkezett adatbázisok, naplók alapján is. Ezek alapján az alábbiakban több olyan általános nyomkövetési technika kerül bemutatásra, amely alkalmas felhasználói adatok gyűjtésére.

2.1.1 A nyomkövetés technikái

Az alábbi technikák - bár napról napra fejlődnek - alapvetően egyszerű módszereket alkalmaznak az információszerzés terén. Igyekeznek a beszerzett adatokból egy olyan egyedi azonosítót generálni (vagy egy olyan párosítást, megfeleltetést végrehajtani), amelyből egyértelműen megállapítható, hogy adott felhasználóhoz melyik profil vagy profilok tartoznak.

2.1.1.1 IP cím

Az IPv4 cím ma már nem biztosít egyértelmű azonosíthatóságot a dinamikus IP címek és NAT mellett, hiszen nehéz úgy összekötni profilokat címekkel, ha azok a címek vagy folyamatosan változnak, vagy egy felhasználóhoz több is tartozik (vagy

akár több felhasználóhoz egy). Megoldást jelent ezen információ kombinálása más információkkal: az IP-címet jelenleg vehetjük majdnem mindig szükséges, de nem elégséges feltételnek. IPv6 címek bevezetése esetén lesz akkora címzési bázis (10^36 db), hogy akár minden egyes elektronikai eszközhöz egyedi cím lesz rendelhető. Így a különböző tracking oldalak látogatóikról begyűjtött információk alapján megtudhatják, hogy melyik IP cím, melyik oldalt töltötte le és melyik időpillanatban.

2.1.1.2 Böngésző süti (Tracking Cookie)

A süti tulajdonképpen egy a számítógépen tárolt karakterlánc, amely 4 részből áll: név, érték, lejárati idő és elérési útvonal. A kiszolgáló nem fogja minden egyes látogatásunkkor megkérdezni ugyanazokat az információkat, hanem azokat első látogatásunk után sütikben tárolhatja. Persze csak akkor, ha élünk ezzel a lehetőséggel. Fontos megemlíteni itt, hogy egy adott szerver csak a saját elérési útját állíthatja be, más szerverét nem. A cookie-k megszületésével a HTTP (HyperText Transfer Protocol) protokollt sikerült állapotokkal gazdagítani. Ha az érem másik oldalát nézzük, a szerver által elhelyezett fájl tracking célokat is szolgálhat. Ebben az esetben a string egy egyedi azonosítót fog képezni, amely ha a felhasználó újra letölti az oldalt, akkor automatikusan elküldésre kerül a szervernek, amely így értesül róla, hogy a megfigyelt meglátogatta az oldalt.

2.1.1.3 Flash süti: LSO (Local Shared Object)

Az Adobe Flash Player összes verziója által támogatott Local Shared Object-ek olyan objektumok, amelyeket Flash programok használhatnak helyi adattárolásra, még pedig 100KB alapértelmezett tárméretig [4]. Fontos tudni róluk – ahogy ez a HTTP sütikre is igaz -, hogy az adott domain-től származó LSO-k nem olvashatók más domain által, tehát a “www.pelda1.hu” tárolt adatai nem olvashatók “www.pelda2.hu” oldalról. Az illegális használatot tekintve, az LSO-k képesek a böngésző sütik visszaállítására (backup cookie) is, miután a felhasználó törölte azokat. Ezek a Flash sütik tökéletesen alkalmasak nyomon követésre, ilyenkor PIE-knek, azaz Persistent Identification Element-eknek nevezzük őket.

2.1.1.4 HTML5 adatbázis

A HTML5 (HyperText Markup Language 5) specifikáció a HTML4 és az XHTML1 (eXtensible HTML 1) új verziója - a hozzájuk kapcsolódó DOM2 (Document Object Model 2) HTML Application programming interface-szel együtt – amely többek

között a mostani böngészők által használt plugin-ek (pl. Flash, Silverlight) leváltását lesznek hivatottak szolgálni. Igaz, hogy a szabvány még mindig fejlesztés alatt áll, viszont egyes böngészők már most támogatják egyes funkcióit. Az egyik legfontosabb funkció kétségkívül az új kliensoldali tárolási módszer, amely ugyan remek jövőbeli innovációs lehetőségekkel rendelkezik, viszont elég sok kritika érte a megfelelő szintű biztonság hiánya miatt. A tárolást az alábbi metódusokra osztották:

- Session storage (Viszony tároló)
- Global storage (Általános tároló)
- Local storage (Helyi tároló)
- Database storage (Adatbázis tároló)

Megjegyzendő, hogy a Global storage csak a HTML5 specifikáció korábbi változataiban volt elérhető; a WHATWG (Web Hypertext Application Technology Working Group) a Local storage-re cserélte, még pedig azért, hogy ne lehessen velük domain-eket meghatározni. A különböző tároló típusok később kerülnek részletesebb ismertetésre.

2.1.2 A technikákra épülő módszerek

Miután megismertük a nyomkövetéshez elengedhetetlen eszközöket, ezen részben az azokra épülő módszerek lesznek részletesebben taglalva [1].

2.1.2.1 Webpoloska

A lehallgatás eszköze. A webpoloskák általában egy 1x1 pixel nagyságú GIF vagy PNG formátumú, legtöbbször a háttérszínnel megegyező, vagy átlátszó képek, amelyek úgy tudják garantálni a külső forrásból való lekérést, hogy az oldalon böngésző felhasználó semmit sem sejt a letöltődésükről [5]. Az adott kép egy harmadik fél szerveréről töltődik le, így a web bug tulajdonosa könnyedén tudomást szerezhet a megfigyelt internetező IP céméről vagy például böngészője típusáról is. Ha a webpoloskákat sütik olvasásával kombináljuk, akkor az ún web beacon-öket kapjuk.

2.1.2.2 Szuper-perzisztens süti (Evercookie)

Az Evercookie-t lényegében egy extrém módon ellenálló sütiként képzelhetjük el [6]. A nagyfokú perzisztensségét abból nyeri, hogy több tárban helyez el információkat a felhasználóról, tehát több, a böngésző által támogatott tárolási mechanizmust használ fel. Az Evercookie okos: ha észreveszi, hogy a felhasználó

akármelyik helyről törölt egy sütit, úgy, hogy legalább egy helyen, egy tárban maradt egy példány, akkor azt rövid időn belül a többi helyen is újra létrehozza. A süti a következő tárolási módszereket használja:

- HTTP sütik
- Local Shared Object-ek
- Silverlight Isolated Storage
- Sütik tárolása gyorsítótárazott PNG képek RGB értékeiben; kiolvasás HTML5 canvas segítségével
- Sütik tárolása a webes előzményekben
- Sütik tárolása a HTTP Etag mezőkben
- Sütik tárolása a webes gyorsítótárban
- window.name gyorsítótárazás
- Internet Explorer userData tár
- HTML5 Session, Local, Global, Database tárak

Az Evercookie fejlesztője még további két fejlesztési lehetőséget is felsorol a módszer tökéletesebbé tételének érdekében [6].

2.1.2.3 Cross site scripting (XSS)

A Cross site scripting módszer azon alapul, hogy a támadó kártékony, kliensoldali kódokat ágyaz be más weboldalakba, így azok a scriptek akkor és abban a környezetben fognak lefutni, amikor és amelyben a felhasználó letölti az adott oldalt. A technika segítségével a kliensoldali védelmi rendszer kikerülhető, és rengeteg információ kerülhet a rosszindulatú kódok írójának birtokába, például HTTP sütik, bankkártya-adatok és egyéb személyes információk is.

2.1.2.4 Böngésző által küldött adatok

Egy weblap megnyitása során a böngészőnk automatikusan információkat küld a kiszolgálónak. Ilyen adatok többek között a felhasználó IP-címe, és ezáltal a hosztneve, használt böngészőjének típusa, a számítógépen futó operációs rendszer stb. A szerveren ezek a begyűjtött információk tárolódhatnak is, amelyek alapján az oldalt meglátogatott felhasználókról profilok készíthetők és ezáltal akár rossz szándékú felhasználásra is kerülhetnek.

2.1.2.5 Böngészőből JavaScript használatával lekérdezett adatok

JavaScript használatával számos kliensinformáció lekérdezhető, viszont ebben a pontban két módszert emelnék ki alkalmazására. Az első az ún. fingerprinting, vagyis az ujjlenyomatkészítés. Az adott kiszolgáló a felhasználóról megszerzett adatok - például az IP cím titkosított formában, az operációs rendszer megnevezése, vagy a képernyő felbontása - alapján a felhasználót egyedi módon tudja azonosítani, még pedig úgy, hogy a begyűjtött információk alapján egy egyedi kulcsot generál, amellyel az internetező későbbi látogatása során azonosítható lesz. A módszer egy implementációja kipróbálható a PET Portálon [7]. Hasonló elven alapul a Panopticlick[1] is.

A másik technika a History Stealing, azaz a böngészési előzmények felderítése. Lényege, hogy egy weboldal a képes megmondani a felhasználóról, hogy a lapon melyik Uniform Resource Locator-öket látogatotta meg. Ennek eldöntése többféle módon történhet: például rejtett URL-ek beillesztésével; ezek színe alapján eldönthető, hogy az adott URL-eket a felhasználó megnézte-e, vagy sem.

2.1.2.6 URL referer

Az URL referer egy HTTP kérés fejléc-mező, amelyet arra használnak, hogy megtudják a felhasználó melyik oldalról érkezett az adott weblapra. Fontos ezen mező esetében megjegyeznünk, hogy a felhasználó böngészési útja visszakereshető a referer mezőkön visszafelé történő lépkedéssel, ami lehetőséget ad a hirdetési szervezeteknek, hogy annak az oldalnak fizessenek többet, ahonnan több látogató érkezett hozzájuk. Ma már számos referer módosító és törlő plugin (ún. dereferer) is elérhető a böngészőnkhöz, sőt sok ilyen oldal is rendelkezésünkre áll a nagyobb biztonság elérése érdekében.

2.1.2.7 JavaScript cache problémák

A JavaScript gyorsítótárazási problémák kihasználása egy külső JavaScript fájl (.js) segítségével történik. Ebben a fájlban egy azonosítót tárolnak el egy változó formájában az azonosítót egyébként kliensoldalon tartalmazó süti pótlására, így ha az oldal betöltésekor nem található meg a felhasználó gépén a süti, akkor az létrejön a szerveren lévő JavaScript fájlból. Másik esetben pedig a kliensoldalon tárolt sütiből íródik az azonosító a kiszolgálón lévő fájlba.

[1] http://panopticlick.eff.org/

2.1.2.8 Spyware

A Spyware-ek, azaz magyarul kémprogramok olyan interneten terjedő alkalmazások, amelyek ritkább esetben csak a böngészési szokásaink, érdeklődési körünk megfigyelését és továbbítását hajtják végre. A legtöbb esetben viszont arra törekszenek, hogy megszerezzék a megfertőzött számítógépen található személyes információkat. Az esetek meglehetősen kis százalékában a felhasználó önként vállalja, hogy a program megfigyelje őt, ám többségében ezek az alkalmazások észrevétlenül települnek a felhasználó gépére és ezután is igyekszenek a háttérben, feltűnés nélkül munkálkodni.

2.2 Hírcsatornák

A webes hírfolyamokat napjainkban már hatalmas, szinte világméretű figyelem övezi, hiszen az internet dinamikus fejlődésével ezek is hozzájárulnak mindennapi információéhségünk kielégítéséhez. Ezért is fontos, hogy az alábbiakban részletesebben taglaljam a két típusát – az RSS illetve Atom hírcsatornákat -, hiszen ezek a későbbi vizsgálataink kulcsszereplői lesznek.

2.2.1 RSS folyam

Az RSS (Resource Description Framework Site Summary, vagy ahogy gyakrabban emlegetik Really Simple Sindication) a web feed-ek családjába tartozik. Segítségével különböző webes tartalmakat (például blog bejegyzések, hírek, audió és videó tartalmak) lehet közzétenni az interneten, szabványosított formában. Egy RSS dokumentum, felépítését tekintve XML alapokon nyugszik, viszont a szabvány előír bizonyos kötelező tag-eket, amiket egy csatornának kötelezően tartalmaznia kell. A szabványos felépítéshez szükséges még a folyam azonosításához nélkülözhetetlen metainformációk tartalmazása is. A dokumentum felépítése a következő:

```
<?xml version="1.0" encoding="utf-8"?>
<rss
version="2.0"xmlns:content=http://purl.org/rss/1.0/modules/content/xmlns:wfw=htt
p://wellformedweb.org/CommentAPI/>
<channel>
      <title>Teszt RSS2 csatorna</title>
      <link>http://research.pet-portal.eu/rss-test/</link>
      <description><![CDATA[Teszt RSS2 csatorna]]></description>
      <image><title>Teszt RSS2 csatornatesztkép</title>
         <link>http://research.pet-portal.eu/rss-test/</link>
         <url>http://www.pelda.hu/kepek/teszt.gif</url>
      </image>
```

```
11        <item>
12              <title>Cim</title>
13              <link>http://www.pelda.hu/hir11</link>
14              <pubDate></pubDate>
15              <description> Wed, 02 Oct 2002 15:00:00 +0200 <![CDATA[ példa szöveg
    ]]></description>
16        </item>
17  </channel>
18  </rss>
```

Amint látható, az XML és RSS meta-adatokat követően az rss és channel tag-eken belül először maga a csatorna leírása következik, majd az egyes feed itemeké. Fontos megjegyezni, hogy a *description* mezőkben elhelyezett szövegek Character Data (CDATA) formájában vannak feltűntetve, ez azt jelenti, hogy az XML Parser (értelmező) szövegként fogja csak őket figyelembe venni. Amint azt a későbbiekben látni fogjuk, a parser-nek tett megjegyzés eltávolítása esetén ebbe a mezőbe HTML, JavaScript, vagy akár egyéb kódok, illetve szkriptek ágyazhatók.

Az RSS csatornákon közölt tartalmakat webes alapú, alkalmazás alapú, vagy akár mobil eszköz alapú alkalmazásokkal, tehát böngészőkkel, vagy ún. aggregátor (feed olvasó) programokkal tölthetjük le és olvashatjuk el. A frissítés ezeknél a programoknál többnyire automatikusan történik, a beállított időközöknek megfelelően. A felhasználó az adott hírfolyam URI-ját (Uniform Resource Identifier-jét, azaz egységes erőforrás-azonosítóját) az olvasóba beütve vagy a folyam RSS ikonjára klikkelve tud feliratkozni a csatornára. Az olvasók feladata, hogy időnként ellenőrizzék, hogy keletkezett-e új bejegyzés a csatornán majd letöltsék, illetve egy felhasználói interfész formájában megjelenítsék azt. A fő ok, amiért az RSS feed-ek ennyire elterjedtté váltak az az, hogy használatukkal elkerülhető a publikáló weboldalak egyenkénti böngészése, ehelyett csak a feliratkozott oldalakról származó legújabb tartalmak kerülnek letöltésre, manuális böngészés nélkül.

2.2.2 Atom hírcsatorna

Az Atom elnevezéshez kettős jelentés kapcsolható. Egyrészt az Atom Syndication Format egy webes hírfolyamok családjába tartozó XML nyelvet jelent, másrészt az Atom Publishing Protocol (más néven AtomPub vagy APP) pedig egy HTTP protokoll, amely webes erőforrások létrehozására és frissítésére szolgál. Az Atom, bár jóval kevésbé elterjedt, mint az RSS, de mivel szinte ugyanazokkal a funkciókkal rendelkezik, és többek között a tartalom modellezésében, dátum

formátumokban és modularitásban tér el attól, ezért csak az RSS 2.0-hoz képesti XML element eltéréseket írom le [8].

RSS 2.0	Atom 1.0
author	author
category	category
channel	feed
copyright	rights
description	subtitle
description	summary és/vagy content
generator	generator
guid	id
image	logo
item	entry
lastBuildDate	updated
link	link
managingEditor	author vagy contributor
pubDate	published
title	title
ttl	-

2-1. ábra: RSS 2.0 és Atom 1.0 elemek összehasonlítása

Az RSS nagyobb népszerűségét és ezáltal előnyét annak köszönheti az Atommal szemben, hogy a korai implementációi előbb kerültek publikálásra. Így számos előnye ellenére is (például regisztrált MIME típus, XML névtér, relatív URI-k támogatása) alulmaradt vetélytársával szemben.

2.3 Böngészők

A mai nagyobb piacvezető böngészők már egytől egyig támogatják a hírfolyamok olvasását, és a hírforrások kezelését. Ezek az internetes alkalmazások lehetőséget biztosítanak arra, hogy az adott hírportál megnyitása nélkül válasszuk ki a legfrissebb híreket, amelyek (legtöbbször) mindössze a könyvjelzősávból elérhetőek, és csak azokat olvassuk el, amelyek minket feltétlenül érdekelnek. A tesztelés megkezdése előtt elengedhetetlen a megfelelő böngészők kiválasztása. A választás a piaci részesedés

és a globális elterjedtség első 5 helyezettjére esett, és azokon belül is a 2011. szeptemberében elérhető legfrissebb verziókra, amelyek rendre: Microsoft Internet Explorer (verzió: 9.0.8112.16421 : 64-bit), Mozilla Firefox (verzió: 6.0.2 : 32-bit), Google Chrome (verzió: 13.0.782.220 : 64-bit), Apple Safari verzió: (verzió: 5.1 : 64-bit) illetve az Opera által fejlesztett Opera (verzió: 11.51: 32-bit) [9].

2.3.1 Internet Explorer

A Microsoft cég, néhány éve teljesen integrálta operációs rendszereibe az Internet Explorert, üzleti előnyszerzés reményében, ami az akkori piaci részesedést tekintve sikerült is nekik [9]. Az Internet Explorer 9-es szériájáról elmondható, hogy jóval nagyobb mértékben támogatja a webes szabványokat, mint elődei, illetve talán ez az első verzió, amely tényleg elfogadható szabvány támogatással rendelkezik. Az új böngésző nagy technikai újítása a hardware gyorsított rajzolás a DirectX 11-ben bemutatott Direct 2D segítségével, így a weboldalak megjelenítése a többi böngészőhöz viszonyítva is villámgyors [10]. Ingyenes termékről van szó, amely a Windows operációs rendszerbe integrált, viszont csak Vistán és 7-en fut, így nem elérhető más típusú rendszereken. Támogatja az új HTML5 szabványt, az RSS híroldalak olvasását, a felső menüsáv a többi piacvezető böngészőhöz képest az Explorerben a legkisebb, mindössze 63 pixel. Fontos újítás még az ún. Tracking Protection rendszer, ami jóval nagyobb teret hagy a felhasználóknak privátszférájuk védelmének finomhangolására[2].

2.3.2 Firefox

Ingyenes, számos rendszeren (Windows, Linux és Mac OS X) használható, grafikus, nyílt forráskódú, több nyelven elérhető web-böngésző program. Fejlesztését az Internet Explorerrel vívott hajdani böngészőháborúban alulmaradt Netscape kezdeményezte ellenlépésként a Mozilla projekten keresztül. A nyílt forráskód ellenére, könnyen kezelhető és testre-szabható. Az aktuális, 7-es verzió gyors működésű, kisméretű program, amely maximálisan támogatja a legtöbb webes-szabványt és kifejezetten sokoldalú, így elég hamar a webfejlesztők kedvencévé vált. Ebben a verzióban a fejlesztők arra törekedtek, hogy a program az elődökhöz képest jelentős, kb. 20-30%-kal kevesebb memóriát használjon csak fel. Legfontosabb újítása, a plugin-ből kifejlődött Sync szolgáltatás, amely segítségével szinkronizálhatjuk könyvjelzőinket, jelszavainkat vagy előzményeinket más számítógépekkel, sőt még androidos

[2] http://ie.microsoft.com/testdrive/browser/trackingprotection/default.html

telefonunkkal is. Számtalan plugin letölthető hozzá, viszont egyes bővítmények lassuláshoz, vagy akár időnkénti összeomláshoz vezethetnek[3], illetve privát adatokat is könnyedén gyűjthetnek rólunk. A legutóbbi felmérések szerint a Firefox az aktuálisan legtöbbet használt böngésző a világon [9].

2.3.3 Chrome

Egy új, nyílt forrású böngésző, melyet a Google teljesen a nulláról kezdve épített fel, az internet mai viszonyaihoz igazítva. A program az interaktív weboldalak, webes alkalmazások és a JavaScript minél gyorsabb futtatását célozza meg. A formát illetően: egy lecsupaszított, egyszerűen lényegre törő, a saját beépített keresőjével ellátott böngésző eszköz. Platformtámogatását nézve, a Google böngészője használható Windows, Linux illetve OS X operációs rendszereken is. A Chrome - amely már a 14-es verziónál jár - egyik legérdekesebb funkciója az ún. Instant Pages. Bekapcsolásával már azelőtt megjelenik előttünk a megnyitandó oldal, mielőtt befejeztük volna címének begépelését. Erősen támogatja a legújabb webes szabványokat, illetve pl. beépített Flash lejátszóval és PDF olvasóval is rendelkezik. Az internetes keresés és böngészés alap funkciókon kívül, ki-ki tetszés szerinti kiegészítőkkel bővítheti, vagy éppen a Google eszközeit használva (Google Web Toolkit[4]) maga is készíthet hasonlókat. Fokozott elővigyázat szükséges az egyes ellenőrizetlen kiegészítések telepítésénél, hiszen ezek akár veszélyesek is lehetnek biztonságunk szempontjából.

2.3.4 Safari

A Safari web-böngészőt, az Apple fejleszti a Mac OS X operációs rendszeréhez, viszont újabban a Microsoft Windows operációs rendszerekhez is elérhető. A Safari előző verzióihoz képest az 5-ös széria már nem egy félkész érdekesség, sokkal inkább egy lehetséges alternatíva a mai vezető böngészők között. Jellemzői közé sorolhatjuk a privát böngészést, a többablakos megjelenítést, a beépített letöltésvezérlőt, a keresőt, a többféle beállítható tartalom-blokkolást, és a HTML5, valamint az RSS támogatását. Számítási és lapfeldolgozási sebességben kicsit lemarad az élmezőnytől, viszont nagy pozitívum, hogy OS X mellett Windows operációs rendszer támogatással is rendelkezik. Megjelenésében nem alkalmazkodik a Windows épp aktuális témájához, ezért meg kell szoknunk az egyszerű vonalvezetését. Hazai szempontból negatívum, hogy magyar

[3] http://kb.mozillazine.org/Firefox_crashes

[4] http://code.google.com/webtoolkit/

verzió nem létezik, ennek következtében számos előnyös tulajdonságai ellenére is nehezen elképzelhető, hogy a közeljövőben a hazai dobogó legfelsőbb szintjein helyezkedjen el.

2.3.5 Opera

Ingyenes, zárt forráskódú, és a legtöbb rendszeren hibamentesen használható internetes programcsomag. Elsősorban egy web-böngésző, de tartalmaz beépített levelező klienst, címjegyzéket, IRC alapú csevegő klienst, RSS és Atom hírolvasót, webes eszközöket, valamint még letöltésvezérlőt is. A jelenleg 11-es verziónál tartó Opera gazdag funkciókészlettel rendelkezik, sőt ezeket a funkciókat még tovább növelhetjük külső bővítmények, ún. Widgetek használatával. A jól átgondolt fejlesztés gyümölcseként sikerült megőrizni a kis méretet, a kompaktságot, valamit a gyorsaságot, és a kényelmes használatot is. Maximálisan támogatja a legtöbb mai webes szabványt. Presto elnevezésű motorját több nagyobb kereskedelmi partner is használja. A motor újabb változatai több javítást és optimalizációt tartalmaztak, például az eredetileg lassú ECMAScript motor mára az egyik leggyorsabb lett a modern böngészők között. Bár zárt forráskódú szoftverről lévén szó, asztali (desktop) változatának letöltése és használata ingyenes. Magyar nyelven is elérhető.

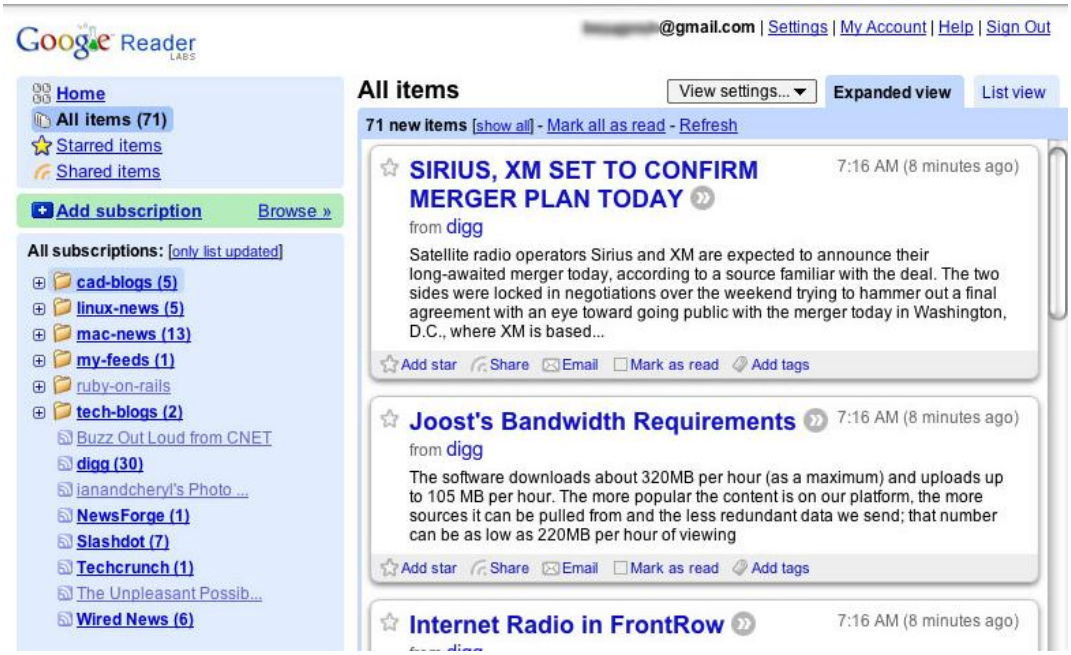
2.4 Hírolvasók

A böngészők mellett az ún. hírcsatorna-olvasó alkalmazások (vagy más néven aggregátorok) is lehetőséget teremtenek arra, hogy egyszerre több hírfolyamot is figyelemmel kísérjenek a felhasználók, akik feliratkoztak a csatornákra. Az említett programok mindezt természetesen megszabható automatikus vagy manuális frissítési idővel teszik. A csatornaolvasó-alkalmazásoknál a FeedDemon (verzió: 4.0.2), Readefine Desktop (verzió: 2.46) került terítékre. Bár böngésző alapú, de a teljességre törekvés érdekében a Google Reader is tesztelésre került, mint a manapság szinte legelterjedtebb RSS olvasó.

2.4.1 Google Reader

A Google Reader egy, a Google által fejlesztett és karbantartott webes felületű RSS olvasó szoftver, amely segítségével korlátlan számú hírcsatornát kezelhetünk és olvashatunk nap, mint nap. Funkciói meglehetősen sokrétűek: lehetőségünk nyílik még a hírek csillagozására (így eltárolhatók), webes közösségi oldalon való megosztására és kategóriákba rendezésére (taggelésére) is. Az olvasó már androidos mobilokra is

elérhető különálló alkalmazásként. A hírforrásokat OPML formátumból importálhatjuk a programba, valamint ugyanebbe a formátumba exportálhatjuk is őket. Az alkalmazás használatához elengedhetetlen egy web-böngésző alkalmazás (a Google természetesen a saját böngészőjét, a Chrome-ot ajánlja a legjobb teljesítmény elérése érdekében). A szolgáltatás 2009. októberétől már magyar nyelven is elérhető.

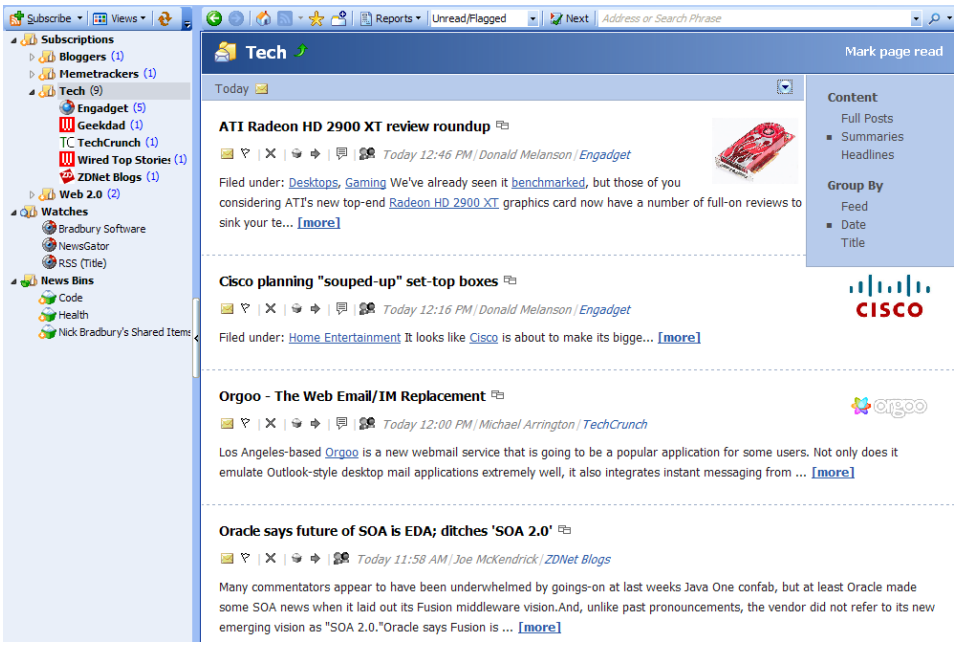


2-2. ábra: Google Reader[5]

2.4.2 FeedDemon

A FeedDemon, a fejlesztő cég állítása szerint a legnépszerűbb RSS olvasó Windows operációs rendszerre [11]. A hírcsatornák felvétele illetve kezelése pofonegyszerű, sőt számtalan hasznos tulajdonsággal rendelkezik. Híreinket szinkronizálhatjuk Google Readerrel, támogatja a tag-elést, a letöltött elemek közötti keresést és a hírfigyelést is. Utóbbi esetében a program a megadott kulcsszavakat figyeli, és csak azokat az elemeket jeleníti meg, ahol ezek a kifejezések előfordultak. Segítségével kedvenc podcast-jeinket is meghallgathatjuk, illetve iPod-unkra vagy egyéb eszközünkre is rátölthetjük őket. A jelenleg 4.0-s verziónál tartó alkalmazás a Windows Xp, Vista és 7-es rendszereket támogatja, viszont futtatásához szükséges még a Microsoft Internet Explorer 7-es vagy annál újabb változata.

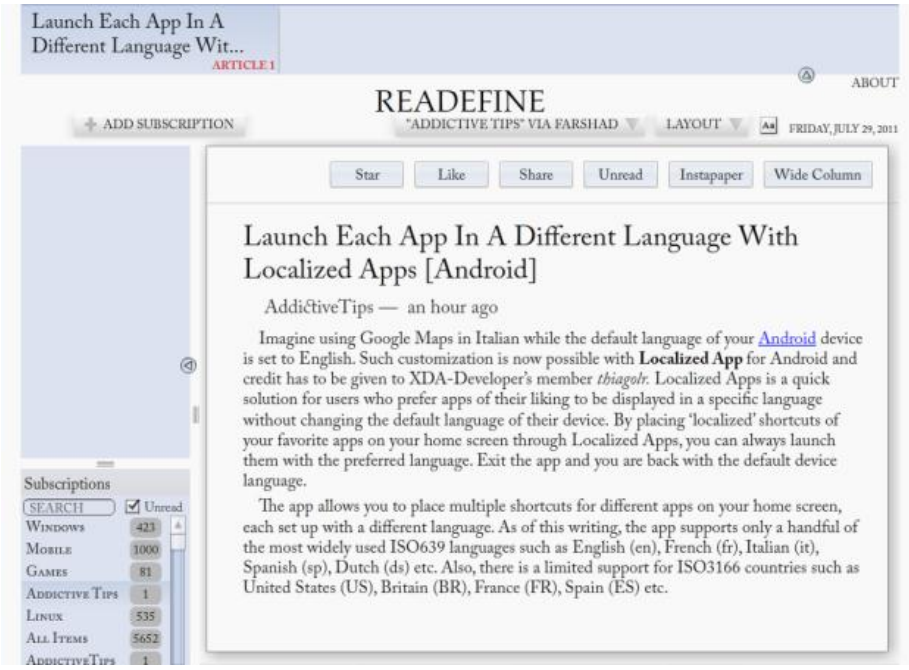
[5] http://www.stress-free.co.nz/google_reader_is_now_my_primary_rss_experience



2-3. ábra: FeedDemon[6]

2.4.3 Readefine Desktop

Adobe Air alapú, teljesen ingyenes, ráadásul kompatibilis Windows, Mac és Linux operációs rendszerekkel. A Readefine Desktop-ot főként Google Reader feed-ek olvasására tervezték, de megnyitható vele akármilyen hírfolyam, szöveges vagy akár HTML fájl is. Az alkalmazás felhasználói felülete a 2-4. ábrán látható.



2-4. ábra: Readefine Desktop

[6] http://www.file-extensions.org/feeddemon-file-extensions

A dizájn letisztult, megjelenése szinte magazinszerű, beállítási lehetőségei egyszerűek, ezzel együtt kissé hiányosak. A letöltött hírek megcsillagozhatók, azaz a kedvencek közé menthetők, lájkolhatók, sőt megoszthatók bármely internetes közösségi oldalon.

A tesztelő alkalmazások ismertetése után, a következő részben a tesztelés magja, azaz maga a szempontrendszer, illetve az ahhoz kapcsolódó technikák kerülnek részletes bemutatásra.

3 A böngésző- és hírolvasó teszt szempontrendszere

A szempontrendszert két fő részből áll, egy általános böngésző- és hírolvasó-tesztből, és egy specifikusabb, hírcsatorna-kezelési aspektusból történő tesztből. A vizsgálatok több részre, szekcióra bontása révén a cél egy átfogóbb, globálisabb kép alkotása volt a tesztelő alkalmazásokról.

3.1 Böngészők és olvasók alapvető funkciói

A vizsgálat első részében az olvasók alapvető funkciói kerültek tesztelésre. A hangsúlyt két dologra kellett fektetni: egyrészt azokat a funkciókat kellett közelebbről is megnézni, amelyek az RSS folyamok kezeléséhez, megjelenítéséhez nélkülözhetetlennek számítanak, másrészt pedig azokat a beállítási lehetőségeket kellett vizsgálni, amelyek az adatvédelem és az anonimitás szempontjából kiemelten fontosak lehetnek. Nem utolsó sorban, meg kellett nézni azokat, a csatornákhoz szorosan kapcsolódó opciókat is, amelyeket a felhasználó az adott programban könnyedén elérhet és igényeinek megfelelően finomhangolhat, testreszabhat. A tesztek egy része itt RSS vagy Atom csatornákra vonatkozik, egy része viszont kifejezetten az adott program képességeit vizsgálja, hírfolyamoktól függetlenül.

3.1.1 Fő funkciók

Privát böngészés és feed-ek támogatása. A legalapvetőbb funkciók ebben a szekcióban kerültek tesztelésre.

3.1.1.1 Privát böngészés támogatása

Az inkognitó üzemmódban történő böngészés, azaz a privát böngészés lehetőséget ad a felhasználónak arra, hogy úgy keressen fel weboldalakat, hogy azokról akármilyen információ is mentésre kerülne [12]. A beállítás következtében a sütik, űrlapadatok, jelszavak, böngészési előzmények nem kerülnek tárolásra a merevlemezen, és a gyorsítótárazott fájlok, és a Flash vagy Silverlight cookie-k is csak az adott session végéig, azaz a privát tab bezárásáig.

3.1.1.2 RSS támogatása

Az RSS feed-ek támogatásán nem a csatorna megjelenítését értjük, hanem a folyamra való feliratkozást és vagy a könyvjelzősávhoz hozzáadhatóságot. Ez a funkció mindenféle külső bővítmény nélkül került tesztelésre. Kivételt képez ez alól a Firefox,

hiszen ott egyáltalán nem volt rövid, képet vagy akár videót tartalmazó pár mondatos leírás vagy előnézet a hírfolyamban lévő hírekről (mint az összes többi alkalmazásnál), csupán a könyvjelzőkhöz hozzáadás lehetősége, ezért a böngésző a Sage plugin használatával lett tesztelve.

3.1.1.3 Atom támogatása

Az Atom csatornák támogatásának vizsgálata szintén az előző pontban leírtak szerint történt.

3.1.2 Időbeli és frissítési tényezők

Nem mehettünk el szó nélkül az időbeli és frissítési paraméterek mellett sem. Az aggregátorok egyik legfontosabb tulajdonsága – a korábban leírtak szerint - az, hogy a hírfolyamokat automatikusan, bizonyos időközönként frissítik. A felhasználó ezt többnyire programtól függő határok között meg is választhatja. Bár érdekességnek számít, de ellenőrzésre került még a frissítés során letöltött hírek darabszáma is.

3.1.2.1 RSS feed frissítés gyakorisága

A mérés Wireshark protokoll-analizátor programmal[7] lett végrehajtva, a hírolvasó által küldött HTTP kérések rögzítésével. A vizsgálat teljessége érdekében az értékelés az alábbi négy szempont alapján történt:

- aktív használat mellett (normál üzemmódban): folyamatos böngészés mellett
- kevésbé aktív használat mellett (normál üzemmódban): csak egy weboldal van megnyitva a háttérben
- aktív használat mellett (privát üzemmódban): inkognitóban történő folyamatos böngészés mellett
- kevésbé aktív használat mellett (privát üzemmódban): privát módban, ha csak egy oldal van nyitva a háttérben
- alkalmazás indításakor: az aggregátor vagy böngésző indításakor a csatorna meglátogatása nélkül letölti-e azokhoz a folyamhoz tartozó új híreket, amelyekre a felhasználó feliratkozott

Látni fogjuk, hogy az alpontok megkülönböztetése nem mindig számított, tehát nem feltétlenül okozott különbséget a frissítési mechanizmusoknál.

[7] http://www.wireshark.org/

3.1.2.2 Letöltött hírek száma alapértelmezetten

A vizsgálat ebben az esetben a hírcsatornák automatikus ellenőrzésénél és frissítésénél alapértelmezetten és maximálisan letöltött hírek számára irányult.

3.1.3 Tárkezelés

Bár a HTML5 szabvány és a hozzá tartozó adatbázis kezelés még mindig fejlesztés alatt áll, viszont több böngésző már most támogatja egyes funkcióit. A mi szempontunkból az egyik legérdekesebb funkció kétségkívül az új kliensoldali tárolási rendszer. A jövőre nézve remek lehetőségekkel rendelkezik, egyelőre azonban az általa nyújtott biztonság szintje nem megfelelő, azaz nem mutat előrelépést a jelenlegi tárolási metódusokhoz képest. Mint ahogy az már a 2.1.1.4-es pontban is szerepelt, négyféle tárolási metódust különböztetünk meg a szabványban:

- Database storage (Adatbázis tár)
- Global storage (Globális tár)
- Local storage (Helyi tár)
- Session storage (Viszony tár)

A HTML5 szabvány arra épít, hogy a különböző tárolási metódusai mind-mind a merevlemezen tárolják az adatokat, amely ugyan nem annyira helyfüggetlen, mintha minden adatunkat felhőben tárolnánk, viszont jóval gyorsabb és megbízhatóbb [13]. A fő okok, amelyek a WHATWG szerint perdöntő erejűek a kliensoldali tárolás mellett:

- nagyobb válaszarány
- csökkentett terhelés a szerveren
- magasabb fokú elérhetőség (nem kell minden egyes alkalommal bejelentkezni)

Akárhogy is, mikro szinten valószínüleg még egy jó ideig szükséges lesz adatokat tárolnunk a gépünkön, még ha csak minimiális mennyiségűt is.

Ebben a tesztrészben az előbb említett oldal segítségével le lett tesztelve, hogy melyik böngészők támogatják a lenti tárolási metódusokat. A hírolvasók ez esetben böngészőként lettek használva, persze csak ha volt ilyen lehetőség. A különböző adattárolási metódusokról részletesebben a 3.2-es részben lesz szó.

3.1.3.1 Sütik alkalmazáson belüli kézi írása

Bár mellékes információnak számít, de az adott alkalmazáson belül azt leellenőriztük, hogy a menüből vagy címsorból elérhető-e egy olyan beállítás a felhasználó számára, ahol különböző oldalak által beállított sütiket módosítani tudja.

3.2 Hírcsatorna-kezelési funkciók

Vizsgálataink magját a hírcsatorna-kezelési funkciók alapos tesztelése képzi. Itt már csak hírfolyamok aspektusából történt a tesztelés, és sorra kerültek többek között a különböző multimédiás tartalmak, a felhasználóról beszerezhető kliensinformációk elérése, sok, újdonságnak számító tárolási metódus, illetve egyéb megszerezhető információk is. A teszteknél a törekvés arra irányult, hogy minél több módszer, technika megvalósításra kerüljön, és ezáltal kézzel fogható és kipróbálható legyen. Sok olyan új technológiát, trükköt fogunk közelebbről megnézni, amelyet még elég kevesen implementáltak, ezzel is bizonyítva a terület széleskörű, dinamikus és intelligens fejlődését.

Mielőtt elkezdődött a tesztelés, természetesen szükség volt egy saját webes hírfolyam elkészítésére is, hiszen csak így lehetett érdemben kipróbálni az egyes módszereket, technikákat. Mint az már korábban említésre került, egy RSS csatorna elkészítésére esett a választás, hiszen jóval elterjedtebb vetélytársánál, az Atomnál. A csatorna PHP nyelven íródott, működését tekintve pedig az egyszerűség volt a fő szempont: a hírek egy MySQL táblából kerülnek kiolvasásra, ezután a PHP kód egy érvényes és szintaktikailag helyes XML kódot generál a kiolvasott hírekből, így azok a szerveroldali kód hatására publikálásra kerülnek. Szükséges még tudnunk azt is, hogyan is épül fel egy hír, azaz minimálisan - illetve a mi esetünkben - hány mezőből állhat. A legfontosabbak a következők: title (cím), link, create_date (létrehozás dátuma) és a description (leírás) mező. Számunkra a legutóbbi a leglényegesebb, hiszen – amint azt már említettük -, ez az a hely ahova a megfigyelők rosszindulatú kódokat illeszthetnek, és ezáltal információkat szerezhetnek meg rólunk.

A vizsgálatok során kétségkívül az egyik legnagyobb felfedezésnek a HTML iframe tag használata bizonyult. Ez a tag lehetőséget ad arra, hogy egy weboldalba egy másik weboldalt illesszünk be egy megadott méretű frame, azaz keret formájában. Az implementáció a következőképpen néz ki:

```
<iframe src="pelda.html" width="300" height="300">
<p>A böngészője nem támogatja az iframe-et.</p>
</iframe>
```

Látni fogjuk, hogy a kódot az adott hír description mezőjébe ágyazva könnyedén használhatjuk a különböző, pl. JavaScript technikákat, sok esetben még akkor is, ha azok alapvetően szűrve vannak (lásd pl. a Google Reader esetében). Az iframe tag-ek csak akkor kerültek használatra, hogyha az adott nyelvű kódok direkt beágyazással nem, vagy csak részben működtek.

3.2.1 Kliensinformációk begyűjtése

A webet böngészve nem is sejtjük, hogy mennyi információt küldünk el magunkról bizonyos szervereknek. A támadóknak és profilépítőknek számos lehetőség van a kezükben a nyomkövetésre, sőt az információgyűjtést többféle módszerrel, többféle nyelven és szemlélet szerint is megtehetik. Könnyen megtudhatják rólunk, hogy milyen típusú böngésző fut a gépünkön, mi az IP-címünk, a hoszt nevünk, milyen oldalról érkeztünk, milyen telepített plugin-ek találhatók a rendszerünkben, milyen asztali felbontást használunk, azt milyen bitmélységgel, sőt még azt is, hogy milyen operációs rendszer fut a számítógépünkön. A különböző technikák sorra vételével és kipróbálásával konkrét megszerzendő információ mellett meg fogom nézni, hogy azok működőképesek-e az egyes alkalmazásokban.

3.2.1.1 JavaScript-ből

A JavaScript objektumalapú szkript-nyelvet manapság már megszámlálhatatlanul sok weboldalon használnak. Segítségével a statikusan megjelenített tartalom dinamikusabbá és esztétikusabbá tehető. A szkripteket legtöbbször HTML kódba ágyazva, vagy külső, .js fájlként behívva hasznosítják, és kinyerhetők velük a legkülönfélébb kliensinformációk is, ahogy azt a következő is példa szemlélteti:

```
<div id="example"></div>
<script type="text/javascript">
txt= "<p>Browser Name: " + navigator.appName + "</p>";
txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";
document.getElementById("example").innerHTML=txt;
</script>
```

Ezen egyszerű szkript segítségével az adott hírt letöltve böngészőnk típusa és verziószáma fog megjelenni az oldalon.

3.2.1.2 Szerver oldalon

A PHP (PHP Hypertext Preprocessor) szkript-nyelvet legtöbbször dinamikus weblapok készítésére használják. Amikor egy PHP oldalt akarunk lekérni, akkor a szerveroldali kiszolgáló először feldolgozza az abban lévő parancsokat, utasításokat, és csak egy generált HTML kimenetet küldi el a böngészőnek, így a szkript kliensoldalról nem is látható. Kliensinformációkat is könnyedén elérhetünk a használatával:

```
<?php
echo $_SERVER['HTTP_USER_AGENT'] . "\n\n";
$browser = get_browser(null, true);
print_r($browser);
?>
```

Ebben az esetben a beépített metódusoknak köszönhetően szintén a User Agent-ről, azaz a felhasználó által futtatott böngészőről fogunk információkat kapni (pl. böngésző verzió, operációs rendszer, támogatott CSS verzió, vagy sütik engedélyezettsége).

3.2.1.3 URL referer jelenléte

Az URL referer (amely az eredeti szabvány leírásában is hibásan lett bemutatva) egy HTTP fejléc mező és azt mondja meg, hogy az oldalra, amin éppen tartózkodunk, melyik oldalról közelítettük meg, azaz melyik oldalról linkeltünk rá. A referer jelenléte JavaScript használatával könnyen kideríthető, hiszen a *document.referrer* property pontosan ezt fogja nekünk megmondani; tartalmazni fogja az oldal URL-jét.

3.2.1.4 Flash-en keresztül

Az ActionScript egy a JS-hez nagyon hasonló, objektumorientált programozási nyelv, amelyet elsősorban Adobe Flash objektumok programozására fejlesztettek ki. A Flash programozás tulajdonképpen két fő részre válik szét: a megjeleníteni kívánt objektumok felrajzolására és a mögöttük lévő tartalom ActionScriptben történő leprogramozására. Egy AS kód publikálásával, kimeneti fájlként egy .swf fájlt kapunk, amelyet könnyedén beágyazhatunk HTML weboldalunkba. Szemléltetésként a következő kódot lehet felhozni:

```
import flash.text.Font;
var fonts:Array = Font.enumerateFonts(true).sortOn("fontName");
var fonts_array:Array = new Array();
for (var i:int = 0; i < fonts.length; i++) {
      fonts_array.push(new String(fonts[i].fontName));
}
display.text = fonts_array.toString();
```

A fenti AS publikált változatát a HTML oldalba történő beágyazással azt fogjuk megkapni, hogy a felhasználó számítógépén milyen System Font-ok, azaz milyen betűkészletek vannak telepítve.

3.2.1.5 SilverLight-on keresztül

A Silverlight keretrendszerhez fejlesztett programok többféle .NET programozási nyelvben[8] megírhatók, viszont ajánlatos a Microsoft által kiadott fejlesztési eszközöket használni hozzá. A kliensinformációk (jelen esetben a böngészőinformációk) megszerzése Silverlight-ban is lehetséges, még pedig a *System.Windows.Browser* névtérben található *BrowserInformation* osztály segítségével. Az *BrowserInformation* class-t a HtmlPage osztályon keresztül érhetjük el, amely szintén az előbb említett névtérben foglal helyet. Az osztály elérése szemléltetésképpen:

```
1  LayoutRoot.DataContext =
2  System.Windows.Browser.HtmlPage.BrowserInformation;
```

A lefordított,.xap kiterjesztésű, viszont ZIP formátumú kimenetet a következőképpen lehet a HTML kódba integrálni:

```
1  <object data="data:application/x-silverlight-2," type="application/x-
   silverlight-2" width="100%" height="100%"><param name="source"
   value="Pelda.xap"/></object>
```

3.2.1.6 Java-n keresztül

Az általános applet egy kisebb alkalmazás, amelyet egy adott feladat elvégzésére terveztek, és tevékenységeit egy másik, nagyobb alkalmazás keretein belül végzi, gyakran egy plugin formájában. A Java applet-eket a következőképpen foglalhatjuk össze röviden: a Java Virtual Machine által a böngészőben futtatott Java bájtkódok[9]. Az előbb említett típusú applet elkészítése során első lépésként az alkalmazás kinézetet kellett testre szabni, hogy az adatok szemléltethetőek legyenek, majd az adott kliensinformációkat az alábbi kód alkalmazásával lehetett megszerezni. Jelen példa a kliens gép IP-címét kérdezi le:

```
1  InetAddress thisIp = InetAddress.getLocalHost();
2  label.setText("IP:"+thisIp.getHostAddress());
```

A lefordított kód .class fájlját a HTML oldalba beincludolva kellett használni.

[8] http://msdn.microsoft.com/en-us/vstudio/dd643383

[9] Bájtkódok pl. a Java fordító által lefordított .jar illetve .class kiterjesztésű fájlok.

3.2.1.7 Elküldött HTTP fejlécek

A HTTP kérések vizsgálata során, az elküldött fejléctípusok kerültek rögzítésre Wireshark protokoll-analizátor alkalmazás segítségével. Ezek a fejlécek a HTTP tranzakciók paramétereit határozzák meg.

3.2.2 Multimédia és megjelenítési rendszer

A Web 2.0 új szolgáltatásai nem terjedhettek volna el ilyen mértékben a különböző multimédiás tartalmak létezése nélkül [14]. A képek, videók, az animációk illetve a weboldal díszítő elemek, sőt az oldalak teljes egésze ma már szinte egytől egyig PHP, JavaScript, AJAX, Flash, Silverlight, vagy újabban HTML5 technológiát használ. Ebben az alfejezetben ezen programozási módszerekkel létrehozott nyomkövetési technikák estek górcső alá.

3.2.2.1 Proxy funkciók

Bevett biztonságot növelő módszerek közé sorolhatjuk a Proxy szerverek használatát. Ebben az esetben a kommunikálás kliens és szerver között egy beépülő, köztes, aggregálást biztosító szerver ékelődik be, amely ha szükséges megszűri vagy akár meg is változtatja a kéréseket vagy válaszokat. A tesztelés célja az volt, hogy kiderüljön, a hírbe elhelyezett multimédiás tartalmak a Proxy szerverről töltődnek-e le, vagy az eredeti, azaz a direkt forrásból. A lehetséges válaszok pedig: a teljes tartalom proxy-zásra került, csak az RSS, csak a képek, vagy egyik sem.

3.2.2.2 Webpoloskák blokkolása

A Vizsgálat három részből állt, és azt volt hivatott ellenőrizni, hogy First party és Third party cookie-k beállíthatók-e, illetve, hogy harmadik féltől származó képek megjelenítésre kerülnek-e. Az utóbbinál elég volt egy külső domain-ről származó kép manuális elhelyezése a hírben, viszont a másik kettőnél már más volt a helyzet. A képen keresztüli süti beállításnál figyelnem kellett a metódusok helyes sorrendjére. Először cookie beállítás, majd a kép elküldése következik, és ami a legfontosabb, a Content-Type HTTP header mezőt is helyesen kell meghatározni, hiszen miután az adott képet elküldjük, már nincs lehetőség sütik beállítására. A First party cookie-k beállítása tehát a következő módon tehető meg:

```
1   $CookieNev = "Sütim";
```

```
$CookieErtek = "hello";
$CookieKonyvtar = "/";
$CookieEddigTart = time() + 31*24*60*60;
setcookie($CookieNev,$CookieErtek,$CookieEddigTart,$CookieKonyvtar,0);
$image=
"R0lGODlhBQAFAJH/AP///wAAAMDAwAAAACH5BAEAAAIALAAAAAAFAAUAAAIElI+pWAA7\n";  //kep
base64 kodolva van
header('Content-type: image/gif');
echo base64_decode($image);
```

Ha a PHP fájlt egy HTML img tag-gel használjuk, akkor a kimeneten egy 1x1 pixeles átlátszó kép fog megjelenni, és a süti ugyanarról a domain-ről fog származni, amelyen a szkript található.

Third party sütik esetében pedig, lévén, hogy a sütik egy külső domain-ről azaz egy harmadik féltől származnak, a host-ot, amelyhez be akarjuk állítani az adott cookie-t, a CookieDomain változóban például a következő módon adhatjuk meg:

```
$CookieDomain = '.'.preg_replace('/^www\./','',"pelda.hu");
$CookieDomain = preg_replace('/:\d+$/','',$CookieDomain);
setcookie($CookieNev,$CookieErtek,$CookieEddigTart,$CookieKonyvtar,CookieDomain)
;
```

Így a saját domain-t meglátogatva nem ahhoz állítódott be süti, hanem egy külső domain-éhoz, ezáltal úgy kezelhetnek minket, mintha már jártunk volna azon az oldalon.

3.2.2.3 JavaScript futtatás

A teszt első részének célja egyszerű volt: annak kiderítése, hogy a hír description mezőjébe direktben elhelyezett JavaScript kódok lefutnak-e, tehát hogy a <script> tag-ek szűrésre kerülnek-e, illetve például a HTML body, form, keyboard, mouse vagy image esemény attribútumok használhatóak-e JS függvények meghívására. Szükséges volt kipróbálni a Flash-ből meghívottJavaScript működését is, a fentiekben említett illetve bemutatott ActionScript használatával. Példa az AS alkalmazására:

```
import flash.external.ExternalInterface;
var info:Object = ExternalInterface.call("getBrowserInfo");
display.text = info.toString();
```

A kód a *getBrowserInfo()* JS függvényt fogja meghívni, viszont ezt csak akkor tudja megtenni, ha az adott függvény nevesített, és ha ugyanabban a névtérben található, mint hívó szkript. A beágyazott objektum a következőképpen néz ki:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
             id="JSPelda" width="500" height="375"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab">
<param name="movie" value="Pelda.swf" />
<param name="quality" value="high" />
```

```
<param name="bgcolor" value="#869ca7" />
<param name="allowScriptAccess" value="sameDomain" />
<embed src="Pelda.swf" quality="high" bgcolor="#869ca7"
width="500" height="375" name="JSPelda " align="middle"
play="true" loop="false" quality="high" allowScriptAccess="sameDomain"
             type="application/x-shockwave-flash"
             pluginspage="http://www.macromedia.com/go/getflashplayer">
</embed></object>
```

Egy fontos dolgot ki kell emelni az előző kóddal kapcsolatban. A beágyazott objektumnál meg kell adnunk, hogy mely esetekben engedélyezzük a névtérben levő JavaScript függvények elérését (*getBrowserInfo()*). Az allowScriptAccess-nek három lehetséges értéke lehet: always (engedélyezés minden esetben), sameDomain (csak akkor hívható a függvény, ha a HTML oldal és az SWF fájl azonos domain-ről származnak), never (tiltás). Esetünkben a sameDomain volt a megfelelő választás.

3.2.2.4 Megjelenítés

Privátszféránk védelme érdekében sok esetben jobbnak bizonyul, ha a különböző tartalmak megjelenítése, és alkalmazások futtatása a mi engedélyünktől függ, azaz mi magunk mondhatjuk meg, hogy melyik tartalom jelenhet meg azon a webhelyen, ahol éppen tartózkodunk. Az RSS feed-ben elhelyezett képek, Flash és Silverlight tartalmak ezt voltak hivatottak vizsgálni, vagyis, hogy az adott program kért-e alapértelmezetten engedélyt a megjelenítésükhöz illetve megjelenítette-e azokat. A teszt Flash esetében beágyazott object-tel és iframe-mel, Silverlight esetében pedig JavaScript és iframe felhasználásával zajlott, és ha már legalább az egyik féle beágyazási módszer működött, akkor az igennek számított. A válaszlehetőségek a következők voltak: megjeleníti az adott tartalmat; a megjelenítés beállításoktól függ, illetve nem jeleníti meg azt.

3.2.2.5 Flash Local Shared Object beállítása

Mint az már említésre került, a Local Shared Object-ek, vagy más néven Flash sütik, egy tárterületet jelentenek a Flash alkalmazások számára a felhasználó merevlemezén, legfeljebb 100KB nagyságig. A könnyen ellenőrizhetőség érdekében fontos tudnunk, hogy az LSO-k hol helyezkednek el a helyi lemezen:

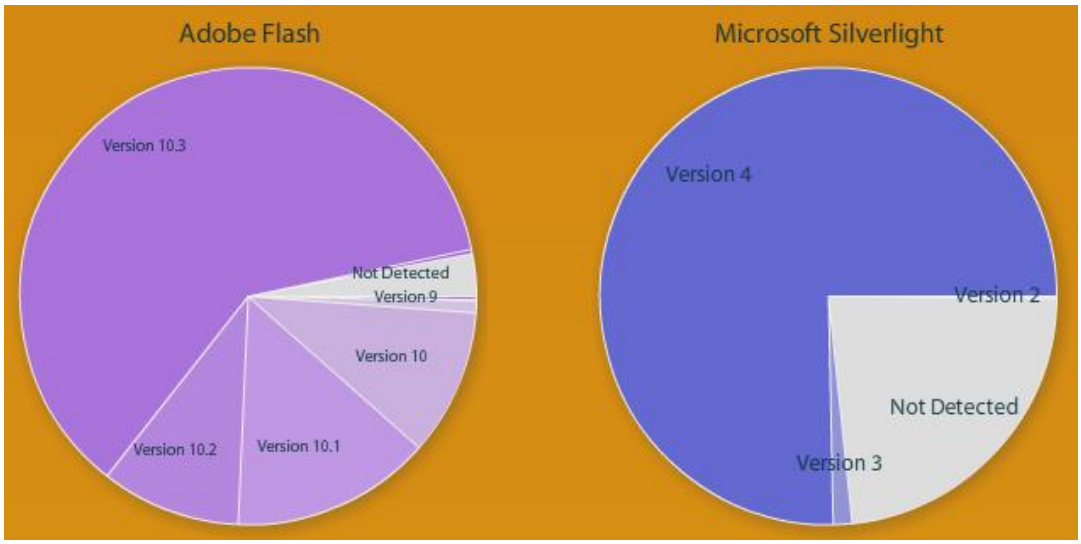
c:/Users/username/user_domain/AppData/Roaming/Macromedia/Flash Player/#SharedObjects/web_domain/path_to_application/application_name/object_n ame.sol

A */#SharedObjects/* könyvtárat felkeresve, és az adott domain-t kiválasztva, egyszerűen ellenőrizni tudjuk, hogy a domain állított-e be flash sütiket. A flash sütik beállítása a következőképpen tehető meg:

```
var mySharedObject:SharedObject = SharedObject.getLocal("ElmentendoAdat");
mySharedObject.data.firstName = "Pelda";
mySharedObject.flush();
display.text = mySharedObject.data.firstName;
```

3.2.3 Silverlightés Flash elterjedtsége

Kiegészítésül érdemes megnéznünk az alábbi kimutatást a Silverlight és Flash globális elterjedtségéről. A statisztika több mint 119 weboldal adatait tartalmazza, több mint 5 millió egyedi böngészőről, a legutóbbi 30 nap alapján. Az információk a 2011.október 2.-i állapotot tükrözik [15]:



3-1. ábra: Flash és Silverlight elterjedtsége

Észrevehetjük, hogy a több mint 5 millió egyedi látogató számítógépének kb. 75%-án volt található telepített Silverlight, míg ez a szám Flash esetében sokkal meggyőzőbb, kb. 97%-os értéket mutat. A előny azzal magyarázható, hogy az Adobe jóval korábban dobta piacra termékét: a Microsoft majd 10 év késéssel tette ezt csak meg. A cég reményei szerint a platformjuk fel fogja venni a versenyt a Flash-sel, hiszen elmondásuk szerint a megoldás lehetőséget ad a WMV, azaz Windows Media Video formátum zökkentőmentes használatára böngészőnkben. Hogy ez a jóslat mennyire fog beválni, és egyáltalán ezek a platformok életben tudnak-e maradni az új HTML5-ös szabvány elterjedésével, az a közeljövőben fog csak kiderülni.

3.2.4 Tárkezelés

Az internet teljes értékű kihasználásához állapotokra van szükségünk. Bár a sütik manapság már elég elavultnak számítanak – és a HTML5 igyekszik is leváltani őket - egyelőre nélkülözhetetlenek ahhoz, hogy egy adott oldal meglátogatásánál ne kelljen minden egyes alkalommal megadnunk például a felhasználónevünket és jelszavunkat, hanem elég azt csak a legelső alkalommal megtennünk, és az egészen addig érvényes lesz, amíg a cookie le nem jár. Az előbb említett megoldásokon kívül meg fogunk még vizsgálni egyéb tárolási módszereket, amelyek új lehetőségeket rejthetnek a jövőre nézve.

3.2.4.1 HTML5 adatbázis kezelés

Mivel a HTML5 szabvány a 3.1.3-as pontban már bemutatásra került, ezért ebben a részben csak a különböző tárolási metódusokról és azok kipróbálásáról fog szó esni, egyszerű példákkal szemléltetve.

Ha egy kicsit nagyobb mennyiségű adattal van dolgunk, mint a többi tár esetében, akkor az egyik legfontosabb szempont, hogy azt strukturáltan, egy adatbázisban tudjuk eltárolni, így az véletlenszerűen is elérhető lesz. A HTML5 ezért vezette be az ún. **Database storage**-et, azaz Adatbázis tárat, amely lehetővé teszi az adatok SQLite táblákban tárolását, bár egyszerű és gyors, mégis néhány problémával is küszködik[10]. Tudomásul kell vennünk, hogy ez nem egy elsődleges adatbázis, sokkal inkább az adatok ideiglenes, és offline, azaz internetes kapcsolat nélküli elérésére létrehozott adatbázis. Egy példakód a helyi adatbázis elérésére:

```
1  var database = openDatabase("Adatbazis nev", "Adatbazis verzio");
2  database.executeSql("SELECT * FROM teszt", function(eredmeny1) {
3  // muveletek az eredmenyekkel
4  database.executeSql("DROP TABLE test", function(eredmeny2) {
5  // muveletek ismet
6  alert("A masodik query-m is sikeresen vegrehajtodott");
7            });
8        });
```

A tár egyelőre számos biztonsági problémával küszködik. A helyi adatok nem titkosítottak és akárki számára elérhetők, és nincs lehetőség a számítógépen tárolt adatbázis szinkronizációjára sem a szerveren tároltéval.

A HTML5 korai változatai mutatták be az ún. **Global storage**-et, magyarul Globális tárat. Ez röviden egy memória szeletet jelent, amelyet a böngésző azon

[10] Ilyenek például a külső kulcs megszorítások.

perzisztens adatok tárolására használhat fel, amelyeket nem kell elküldenie a szervernek, viszont az adatok JavaScript-ből és Flash-ből is elérhetőek. Használata a következőképpen tehető meg:

```
globalStorage[''].foo = 'bar'; //foo minden weboldal számára elérhető
globalStorage['hu'].foo1 = 'bar1'; //foo1 csak a .hu végződésű weboldalaknak elérhető
globalStorage['pelda.hu'].foo2 = 'bar2'; //foo2 kizárólag a pelda.hu lapnak elérhető
```

Miután a böngészők implementálták a saját verziójukat a storage-ből, a WHATWG megváltoztatta az egész tár specifikációját. A Global storage-et Local storage-re cserélték, ahol a domain-ek nem előre megadhatók, hanem az adat, amit a böngésző tárol, automatikusan összekapcsolódik azzal a domain-nel, amelyen az adott script fut.

A **Session storage**-et azaz a Viszony tárat lényegében a HTTP sütik kiterjesztésének, módosításának tekinthetjük, néhány kifejezetten előnyös változtatással. A cookie-k már nem csak kilobyte nagyságnyi területet foglalhatnak el a merevlemezünkön, hanem kiterjedésük akár több megabájt is lehet. A Session adatok nem kerülnek automatikus elküldésre a HTTP kérések folyamán, sokkal inkább a fejlesztő kiválaszthatja, hogy mely kulcs-érték párokat szükséges továbbítani. A Viszony tárak segítségével az adatok a böngésző tab-ekhez kötöttek, tehát két felhasználó is bejelentkezhet egyszerre ugyanabból a böngészőből, két tab-et megnyitva. Egy egyszerű példa a tár használatára:

```
sessionStorage.setItem('keresztnev', 'Pelda');
document.write("Your name is: " + sessionStorage.getItem('keresztnev'));
```

Megjegyzendő, hogy a Session adatok az adott kapcsolat végeztével (tehát az ablak bezárásával) automatikusan megszűnnek.

A **Local storage** helyi tár, és az ehhez kapcsolódó localStorage JavaScript objektum funkcionálisan azonos az előbb említett sessionStorage objektummal, mindössze két fő különbséggel:

- Perzisztencia: a Helyi tárat hosszú távú tárolásra tervezték, tehát az adott böngészőablak bezárása után is létezni fog.
- Elérhetőség: a localStorage adatok az összes megnyitott böngészőablak számára elérhetők, míg a Viszony adatok csak abban az ablakban, amelyekben létrejöttek.

Használatuk a következő példával lehet szemléltetni:

```
localStorage.setItem('keresztnev', 'Pelda');
```

```
document.write("Üdv: " + localStorage.getItem('fullname'));
```

Az előbb említett HTML5 tárakból a globális tároló kivételével mindegyik tesztelésre kerül, még pedig a hírfolyamban elhelyezett kódok segítségével próbálunk majd meg a felhasználó számítógépen információkat eltárolni.

3.2.4.2 Böngésző specifikus lehetőség

A kizárólag az Internet Explorer által használt userData storage-et session-ök közötti perzisztens adattárolásra fejlesztették ki. A tárolt adatok mennyiség jócskán nagyobb lehet, mint a sütiké, viszont ez dinamikus, tehát a domain biztonsági szintjétől függően változik. Rövid példák a működésére, a userData viselkedés beállítására:

```
.storeuserData {
behavior: url(#default#userData);}
```

Egy oPersistForm nevű formból kapott input elmentése:

```
var oPersist=oPersistForm.oPersistInput;
oPersist.setAttribute("sPersist",oPersist.value);
oPersist.save("oXMLBranch");
```

Majd az elmentett adat formba betöltése:

```
var oPersist=oPersistForm.oPersistInput;
oPersist.load("oXMLBranch");
oPersist.value=oPersist.getAttribute("sPersist");
```

Megjegyzés: a userData storage-ben tárolt adatok nem titkosítottak, tehát akármilyen alkalmazás hozzáfér a bennük lévő információkhoz, ezért fontos, hogy a számunkra érzékeny adatokat ne itt tároljuk.

3.2.4.3 Egyéb tárolási lehetőségek

Sorra kellett venni több új, kreatív tárolási módszert, amelyek esetleg érdekesek és elterjedtebbek lehetnek a közeljövőben. Először vegyük a Web cache-ben, azaz a böngésző gyorsítótárában eltárolt sütiket. A módszer azon alapul, hogy kényszerítjük az adott alkalmazást, hogy minél tovább tárolja az adatainkat, például képeinket vagy akár JavaScript fájlainkat. A HTML5 új Cache manifest módszere pont ezt hivatott elvégezni, még pedig a következő módon:

```
<!DOCTYPE  HTML>
<html manifest="teszt.manifest">
<head>
<title>Teszt</title>
<script src="teszt.js"></script>
</head>
```

Miután helyesen beállítottuk a manifest attribútumot a html element-en, a beinclude-olt .js fájl segítségével sütiket kell írunk, majd a létehozott teszt.manifest fájlba az alábbi sorokat szükséges illeszteni:

```
CACHE MANIFEST
/test.js
```

A módszer alkalmazásával tehát a JavaScript fájlunkat tudjuk a böngészőnk vagy olvasó alkalmazásunk gyorsítótárában eltárolni. Ez a merevlemezen több könyvtárban helyezkedik el, attól függően, hogy éppen melyik alkalmazással nyitottuk meg az adott oldalt.

Tárolhatunk információkat a *window.name*, DOM (Document Object Model) property segítségével is. A tárban 2-32 MB-nyi adat tárolható, mégpedig JavaScript nyelv használatával. A property-ről elmondható, hogy csak azoknak a böngésző tab-eknek lesz üres a *window.name* értékük, amelyeket maga a felhasználó nyitott meg, a többi tab-nek viszont nem. Egy illusztráció a működésének szemléltetésére:

```
myWindow=window.open('','MsgWindow','width=200,height=100');
myWindow.document.write("<p>This window's name is: " + myWindow.name + "</p>");
```

A példa először egy ablak létrehozását, majd a nevének kiírását mutatja. Fontos megjegyezni, hogy a property segítségével csak az adott session idejére elérhetőek az abban tárolt adatok, tehát az adott ablak vagy tab bezárásával azok megszűnnek.

Érdekes módszer a sütik tárolására a PNG formátumú képek RGB értékeiben való tárolás. Ha a HTTP sütit a átküldik, az adatok RGB értékekre lesznek átkonvertálva, és a PNG kép 20 éves cache lejárati idővel kerül kirajzolásra. Ha valamilyen okból kifolyólag az adott képgeneráló PHP fájl a süti nélkül kerül elérésre, mert a felhasználó pl. törölte azt, akkor a kód egy “Not Modified” HTTP üzenettel tér vissza, amely arra kényszeríti a böngészőt, hogy a gyorsítótárából töltse be az eltárolt képet. A kód az alábbi:

```
if (!$_COOKIE["evercookie_png"]){
      header("HTTP/1.1 304 Not Modified");
      exit;
}
```

A kliensoldali kód ezután a képet egy HTML5 canvas-re helyezi el, majd pixelenként kiolvasva visszakonvertálja sütivé.

3.2.4.4 Előzmények és elévülési információk

Az következő információk, név szerint egyes elemek (pl. sütik, képek) elévülési ideje a cache-ben (gyorsítótárban), és az RSS URL-jének előzményekbe bekerülése

manuálisan lettek megkeresve, még pedig vagy internetes források alapján - főleg a fejlesztő cégek honlapjáról -, vagy pedig az adott alkalmazás beállítási lehetőségei között. Olyan eset is adódott sajnos, hogy érdemleges információk nem voltak fellelhetők ezen adatokkal kapcsolatban, illetve olyan eset is, hogy ezek a szempontok nem voltak értelmezhetők az adott programra. Az idők, a táblázat ide illő részen belüli első négy pontjánál órában értendők.

3.2.5 Egyéb információk

A teljesség igénye nélkül került ebben a részben néhány olyan információ beszerzésre, amelyeket a profilírozók előszeretettel használnak, ha a megszerzett adatokból egy unique (egyedi) ID-t szeretnének generálni [7]. Az alábbi kliensinformációk mind JavaScript segítségével lettek begyűjtve:

- Képernyő felbontás és színek bitmélysége
- Időzóna
- Telepített betűkészletek

Részletesebben szót ejteni csak a legutóbbiról fogok, hiszen ennek az információnak nehezebb a megszerzése, mint a másik kettőnek. Ebben az esetben a fő motívum az ún. dlgHelper objektum volt, amely lehetőséget biztosít a színskála beállítások, a formázási beállítások, és a telepített betűkészletek kollekciójának eléréséhez is. Az objektum adott oldalba ágyazása az alábbi módon lehetséges:

```
1  <object  id="dlgHelper"  classid="clsid:3050f819-98b5-11cf-bb82-00aa00bdce0b"
   width="0px" height="0px"></object>
```

A clsid egy olyan egyedi azonosító, amely egy COM (Component Object Model) osztálybeli objektumot azonosít, jelen esetben a Dialog Helpert. A felhasználó gépén telepített betűkészleteket a *dlgHelper.fonts* tömb tartalmazza.

4 Böngészők és hírolvasók vizsgálata

Ez a fejezet az előző, hármas pontbeli szempontrendszer szerint felépített tesztet hivatott bemutatni. Jelen teszt két részből tevődik össze: az általános, alapvető böngészési funkciók ellenőrzéséből, és egy sokkal hírcsatorna-specifikusabb, hírfolyam-kezelési aspektusból történő tesztből. Az eredmények az alábbi pontokban lévő táblázatokban láthatók tételesen összefoglalva. Mielőtt a bemutatást elkezdeném, szükséges néhány fontos megjegyzés tétele a böngészőkkel és azok tesztelésével kapcsolatban. A Mozilla Firefox, mivel a csatornák esetében csak könyvjelzősávos feliratkozásra volt lehetőség, és nem volt a hírfolyamból előnézet, a népszerű Sage RSS-olvasó bővítménnyel lett letesztelve. Megjegyzendő még a vizsgálattal kapcsolatban, hogy, amely teszt-esetben az adott funkció direkt beágyazással nem működött, ott az a bizonyos funkció iframe használatával is ellenőrizve lett. Ahogy az a táblázatokban is látszik, ezen HTML tag-ekre elég sokszor szükség volt, mondhatni jelentősen megkönnyítették a hírbe ágyazást. A további táblázatokban fellelhető rövidítések magyarázatai:

- VS: véges sok
- ∅: nem észlelhető
- +: megjeleníti
- -:nem jeleníti meg
- BF: beállításoktól függ
- BFT: beállításoktól függ, külön 3rd party tartalom lehetőséggel
- BBF: böngésző beállításaitól függ
- M: manuális
- Keresztben áthúzott cella: nincs információ
- Sötétebb háttér: iframe-mel tesztelve

A táblázatokban lévő üres helyek azt jelentik, hogy az adott böngésző nem támogatta az adott funkciót. Fontos még hangsúlyozni azt, hogy az alábbiakban látható eredmények saját méréseken illetve kutatómunkán alapulnak, tehát elképzelhetők eltérések más tesztekkel, publikációkkal összehasonlítva.

4.1 Böngésző-teszt eredmények

A vizsgálat kisebbik részét képző általános böngésző teszt eredményeit az alábbi táblázat tartalmazza. Jelen elemzés célja csupán annak ellenőrzése volt, hogy az említett hírolvasó alkalmazások támogatják-e a legszükségesebb és legalapvetőbb böngészési és csatorna-olvasási funkciókat.

Csoportok	**Alcsoportok**	**Értékelési szempontok**	**Böngészők**					**Hírolvasók**		
			Chrome	Firefox	Internet Explorer	Opera	Safari	Google Reader[11]	FeedDemon	Readefine Desktop
Fő funkciók		Privát böngészés támogatása	**x**	**x**	**x**	**x**	**x**			
		RSS támogatása		**x**	**x**	**x**	**x**	**x**	**x**	**x**
		ATOM támogatása		**x**	**x**	**x**	**x**	**x**	**x**	**x**
Időbeli és frissítési tényezők	RSS frissítés gyakorisága[12]	Aktív használat (normál)		**1**	**24**	**3**	**0.5**	**M**	**0.5**	**M**
		Kevésbé aktív használat (normál)		**1**	**24**	**3**	**0.5**	**M**	**0.5**	**M**
		Aktív használat (privát)		**1**	**24**	**3**		**M**		
		Kevésbé aktív használat (privát)		**1**	**24**	**3**		**M**		
		RSS frissítés alkalmazás indításakor		**x**	**x**	**x**	**x**	**x**	**x**	
		Letöltött hírek száma alapértelmezetten		**30**	**200**	**VS**	**VS**	**VS**	**200**	**30**
Tárkezelés	HTML5 adatbáziskezelés	Database Storage	**x**			**x**	**x**			
		Global Storage		**x**						
		Local Storage	**x**	**x**	**x**	**x**	**x**		**x**	
		Session Storage	**x**	**x**	**x**	**x**	**x**		**x**	
		Sütik alkalmazáson belüli kézi írása				**x**				

4-1. ábra: Böngésző teszt eredmények

[11] A Chrome böngészővel lett tesztelve.

[12] Az első négy szmepont órában van megadva.

A továbbiakban röviden sorra vesszük a főbb csoportok szerint, hogy, hogyan is alakultak a fenti táblázatban összefoglalt eredmények, azaz hogyan viselkedtek a tesztelt böngészők illetve hírolvasó alkalmazások az általános tesztben. Meg kell jegyezni, hogy a Safari néha elég furcsán szinkronizálta az egyes hírfolyamokat, főként amikor megnyomtam a frissítés gombot. Sok esetben nem töltötte le az új híreket (több hírfolyamon is ellenőriztem), így a böngésző tesztelése néhol kissé nehézkessé vált. Az alkalmazás újraindítása sem oldotta meg a problémát.

4.1.1 Fő funkciók

A Chrome volt az egyedüli böngésző, amely nem támogatta az RSS illetve Atom csatornákra való feliratkozást, csak egy előnézet megjelenítését, szemben a Firefox-szal, amely pont az ellenkezőjét nyújtotta funkcionalitásban, pedig kényelmi szempontokból talán fontos lehet, hogy ne csak a szalagcímeket lássuk, hanem egy pár mondatos leírást az adott hírről, esetleg a hozzá csatolt multimédiás tartalmakat, stb. A hírolvasóknál nem nyílt lehetőség privát böngészésre, sőt, a FeedDemon kivételével általános üzemmódban történő böngészésre sem, ami nem meglepő, hiszen nem erre fejlesztették ki őket.

4.1.2 Időbeli és frissítési tényezők

Érdekes eredménynek bizonyult, hogy a Safari böngésző privát módban, többszöri tesztelés mellett egyetlen alkalommal sem frissített, a normál üzemmódban mért fél óránként. Az Apple oldalán található információk szerint, ha a privát mód aktív, akkor a program megakadályozza a meglátogatott weboldalaknak, hogy a sütikhez és egyéb, a számítógépen tárolt személyes információkhoz hozzáférjenek, viszont az nem tisztázott, és nem is igazán található róla információ a weben, hogy a mi esetünkben magát az automatikus frissítést is megakadályozná [16].

4.1.3 Tárkezelés

A Local illetve Session tárakat nem meglepetés, hogy az összes böngésző támogatta, hiszen mindegyukük implementálta a saját mechanizmusait ennek érdekében, illetve ezek a technikák bizonyulnak az egyik legígéretesebb HTML5 újításoknak. A szabvány Database storage szolgáltatásának létjogosultsága egyelőre kétségeket vet fel, hiszen számos biztonsági problémával küszködik, amely következtében még szélesebb körű támogatottsága egyelőre várat magára.

Megemlítendő tény még ebben a témakörben, hogy a HTML5 Global storage ugyan már nincs benne a hivatalos szabványban – mivel a WHATWG a lokális tárra cserélte, viszont a Mozilla böngészője, a tesztelt, 6-os verzióban még mindig támogatta.

4.2 Hírcsatorna olvasó teszt eredményei

A szempontok a témakörben egyszerűek voltak: megtalálni azokat a kiskapukat, azokat a technikákat, amelyeket esetleg érdemes lehet a következőkben nyomkövetési téren fontolóra venni, azaz melyek azok a módszerek, amelyeket a nagyobb internetes cégek nyomkövetésre és az azon alapuló profilépítésre használhatnak fel. A teszt eredményeit a következő táblázatban láthatjuk. Amely paraméterek az előző, 3-as fejezetben szerepelnek, viszont a táblázatban nem, azok információtartalma kevés volt, ezért nem lettek a szempontok között felsorolva.

Csoportok	**Alcsoportok**	**Értékelési szempontok**	**Böngészők**					**Hírolvasók**		
			Chrome	Firefox	Internet Explorer	Opera	Safari	Google Reader[11]	FeedDemon	Readefine Desktop
Kliensinformációk	Elérhető kliensinformációk	JavaScript-ből	**x**							**x**
		Szerver oldalon	**x**			**x**				**x**
		URL referer jelenléte	**x**	**x**	**x**	**x**	**x**	**x**	**x**	
		Flash-en keresztül	**x**			**x**				**x**
		Silverlight-on keresztül	**x**							
		Java-n keresztül				**x**				
Multim. és engedélyr.	Poloska funkciók	Képen keresztüli süti beállítása	**x**	**x**	**x**	**x**				**x**
		Third party képek	+	-	+	+	-	+	+	-

Csoportok	Alcsoportok	Értékelési szempontok	Böngészők					Hírolvasók		
			Chrome	Firefox	Internet Explorer	Opera	Safari	Google Reader[11]	FeedDemon	Readefine Desktop
Multimédia és engedélyrendszer	P. f.	Third party sütik	**BFT**	**BFT**	**BFT**	**BF**	**BFT**	**BF**[13]	**BF**[14]	
	Javascript futtatás	Script tag-ek szűrése	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
		Script szűrése attribútumoknál	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
		Flash-ből hívott JavaScript	**x**							**x**
	Megjelenítés	Kép megjelenítése	**BF**	-	**BF**	**BF**	+	+	+	+
		Flash megjelenítése	+	-	-	+	-	+	+	+
		Silverlight megjelenítése	+	-	-	-	-	-	-	-
		Flash Local Shared Object beállítása	**x**			**x**				**x**
Tárkezelés	HTML5 adatbáziskezelés	Database Storage	**x**			**x**	**x**	**x**		
		Global Storage		**x**						
		Local Storage	**x**	**x**	**x**	**x**	**x**	**x**	**x**	
		Session Storage	**x**	**x**	**x**	**x**	**x**	**x**	**x**	
	Egyéb tárolási lehetőségek	Sütik tárolása a Web cache-ben	**x**			**x**				**x**
		window.name caching	**x**			**x**				
		Sütik tárolása a PNG képek RGB értékeiben	**x**							**x**

[13] Adott böngésző beállításaitól függ.

[14] IE böngésző beállításaitól függ.

Csoportok	Alcsoportok	Értékelési szempontok	Böngészők					Hírolvasók		
			Chrome	Firefox	Internet Explorer	Opera	Safari	Google Reader[11]	FeedDemon	Readefine Desktop
Tárkezelés	Előzmények és elévülési információk	Előzményekbe bekerül-e az RSS URL				**x**				
		Kép elévülése a gyorsítótárban[15]		**90**		**0.2**		**BBF**		
		Elem elévülése a cache-ben[15]		**90**		**0.2**		**BBF**		
Egyéb információk		Képernyő felbontás és színek bitmélysége	**x**							**x**
		Időzóna	**x**							**x**
		Telepített betűkészletek	**x**			**x**				**x**

4-2. ábra: Hírcsatorna olvasó teszt eredményei

A részletes elemzés tehát a főbb csoportok alapján fog történni ebben az esetben is, csak a leglényegesebb, legkiemelkedőbb momentumokat említve.

4.2.1 Kliensinformációk

A Google böngészője elég meggyőzően, vagy ha adatvédelem szempontjából nézzük, akkor meglehetősen rosszul szerepelt a kliensinformációk többféle módszerrel történő kinyerésénél. A Java tesztnél egy alkalmazás esetében probléma akadt. A Chrome a Java 1.7-es verzió használata esetén nem jelenítette meg az applet-eket, viszont egy korábbi, a kellően stabil 1.6 update 11-es csomaggal kipróbálva a megjelenítés tökéletesen működött. Remélhetőleg ezt a problémát hamarosan kijavítják. Érdekes eredménynek számított, hogy a beágyazott Silverlight alkalmazást egyedül a Chrome tudta csak futtatni, illetve a beágyazott Silverlight videót is csak ez a böngésző

[15] Napban értendő

tudta megjeleníteni a hírfolyamon; és például a Microsoft, - tehát maga a technika megvalósítója - által fejlesztett Internet Explorer pedig nem.

4.2.2 Multimédia és megjelenítési rendszer

A képen keresztüli (tulajdonképpen First party) süti beállítás több alkalmazásnál is sikeresen működött. A saját hoszton lévő képeket kivétel nélkül az összes böngésző alapértelmezetten megjelenítette, viszont érdekes volt, hogy a harmadik féltől származó, azaz Third party képeket is. A legtöbb alkalmazásnál lehetőség volt az opció finomhangolására, azaz a harmadik féltől származó tartalmak akár teljes blokkolására is. Nem volt meglepő, hogy a JS szkript tag-jei és az azokhoz tartozó attribútumok egyaránt szűrésre kerültek, azaz direkt beágyazás esetén a böngésző *script interpreter*[16]-je nem értelmezte őket. Ez a szűrés akár magán az adott hírfolyam-olvasó weboldalon is megtörténhetett, mint például a Google Reader esetében történt, viszont ezek a problémák ActionScript, illetve HTML iframe használatával sok esetben megkerülhetőek voltak. Megemlítendő még, hogy a beágyazott Flash applikációkat a több alkalmazás is megjelenítette, viszont érdekes tény, hogy közülük egyik sem kért hozzá engedélyt.

4.2.3 Tárkezelés

A HTML5 storage-ek már az általános részben bizonyos szempontok szerint megvizsgáltam, viszont itt a cél az RSS csatornán keresztüli működés tesztelése volt. Meglepetésre, az összes applikáció, amely az előző, (4.1-es) fejezetben jól szerepelt, az itt is: az eredmények hajszál pontosan ugyanazok lettek, ami azt jelenti, hogy sikerült a HTML5 tárakban eltárolnom a specifikus adatokat az adott híreken keresztül. Az egyetlen böngésző specifikus tárolási lehetőség - a userData tár - köztudottan az Internet Explorer böngésző számára készült, tehát csak ez az alkalmazás tudja kezelni, viszont a tesztelés során nem sikerült információkat eltárolni benne. Az Evercookie funkciók, név szerint a web cache-es, és PNG képes süti tárolás, illetve a window.name attribútum használatával történő tárolás elég kevés esetben működött. Érdekes tény, hogy a Google Reader minimalista asztali alteregója, a Readefine Desktop, a Chrome mellett szinte az

[16] Ha a weboldalunkon HTML, XML vagy például JavaScript kódokat helyezünk el, akkor az adott böngésző saját interpreter-je értelmezi ezeket. Ennek, és a további feldolgozásnak köszönhetően nem a plain text jelenik meg a képernyőn, hanem a lefuttatott kód eredménye.

összes módszert támogatta, a window.name gyorsítótárazás kivételével, amely utóbbi magától értetődő, hiszen a programban tab-ek sincsenek. Az előzményekbe csak az Opera feed olvasó alkalmazásában került be az RSS csatorna linkje, az automatikus frissítések alkalmával. Megemlíthető még, hogy a különböző tárolt elemek elévülési idejére vonatkozóan elég kevés információt sikerült fellelni az interneten, illetve az adott alkalmazások beállítási lehetőségei között.

4.2.4 Egyéb információk

Az egyéb, JavaScript-ből illetve Flash-ből kinyert információk esetében a Chrome és a Readefine Desktop is eredményesnek mutatkozott. A kliensoldalon telepített betűkészleteket JS-tel nem lehetett kinyerni, viszont ez Flash segítségével sikerült, így tehát a táblázat ezt az eredményt tartalmazza.

4.2.5 Elküldött HTTP fejlécek

Célszerű volt a kliens által a szervernek küldött HTTP fejléceket a jobb szemléltethetőség érdekében egy külön táblázatba gyűjteni. A teszt azt volt hivatott eldönteni, hogy a különböző esetekben pluszként elküldött fejléc mezőket esetleg fel tudjuk-e használni arra, hogy például hamis információkat küldésével megtévesszük a kliensoldalt, pontosabban a böngészőt, és így egy esetleges gyorsítótárból való kiolvasást kényszerítsünk ki. Az adott böngészők illetve hírolvasó alkalmazások a függelékben található táblázatban fellelhető mezőket küldték el adott HTTP tranzakciók folyamán. A teszt ezeket a fejléceket általános körülmények között volt hivatott megvizsgálni, tehát normál böngészési módban és semmilyen nemű alkalmazás-beállítások módosításával. A felfedezett, és lényegesnek bizonyult fejlécek magyarázata:

- Accept: böngésző által elfogadott média típusok
- Accept-Charset: böngésző által elfogadott karakterkészletek
- Accept-Encoding: böngésző által elfogadott kódolások
- Accept-Language: böngésző által elfogadott nyelvű válaszok
- Cache-control: gyorsítótárban tárolás tiltása/engedélyezése
- Connection: milyen típusú kapcsolatot részesít a UA előnyben
- Host: szerver domain címe
- Referer: előzőleg meglátogatott oldal címét tartalmazza
- User-Agent: böngésző azonosító-típust azonosító sztring

A felsorolt mezők közül kétségkívül a referer számunkra a legérdekesebb. Ez a karakterlánc tartalmazza annak az oldalnak az URL-jét, amelyről a jelenlegi oldalra navigáltunk. Ez lehetőséget biztosít a profilírozó és főleg a hirdető cégeknek arra, hogy a hozzájuk érkező vásárlók száma alapján tudjanak arányosan fizetni a hirdetést elhelyező odalaknak. Egyes esetekben ezek a szervezetek úgy próbálnak minél jobb helyezést elérni egy adott kereső rangsorában, az oldaluk látogatottságán növelni vagy egyszerűen csak minél több pénzt keresni, hogy spoofolják, azaz meghamisítják a referer-t, ezzel generált forgalmat létrehozva az általuk üzemeltetett weblap számára.

A teszteredményeket tekintve tehát, láthatjuk, hogy a számunkra hasznos HTTP mezők közül, a referer - a User Agent mellett -, milyen fontos motívuma lehet a további felméréseknek, vizsgálatoknak. Az eredményeket tartalmazó táblázat a függelékben található.

5 Megfigyelésre alkalmasnak talált technikák felfedezése

Mivel az előző fejezetekben vizsgált technikák már részletesen bemutatásra kerültek, ezért jelen részben csak a megfigyelésre koncentrálok, tehát csak azokat a módszereket sorolom fel, amelyek webes megfigyelésre alkalmasnak bizonyultak. A felsorolás nem kizárólag a hírcsatornák aspektusából történik majd, - hiszen az RSS és Atom folyamokon történő lehallgatás csak kisebb része a lehallgatások globális számának – hanem sokkal inkább néhány általánosabb, és népszerűbb weblap elemzése alapján, hagyományos, és újabb internetes nyomkövetési technológiák felhasználásával egyaránt. A vizsgálatok során kikristályosodott számomra, hogy olyan technikákat, technológiákat kell keresnem, amelyek a lehető legtöbb böngészőben és hírcsatorna-olvasó programban működnek és a hozzájuk szükséges esetleges pluginek a lehető legszéleskörűbben elterjedtek. A táblázatok eredményei alapján, az alábbi pontokban részletezett technikákat érdemes a begyűjtött dokumentumokban keresni. Ezek azok a módszerek, amelyek a legtöbb esetben működtek, így a legtöbb alkalmazásban sikeresen lehetett használni őket.

5.1 URL referer jelenlétének vizsgálata

Az URL referer HTTP fejlécmező egy kivétellel az összes böngészőben és hírolvasó programban jelen volt. Ezen fejlécmező a következő nyelveken írt kifejezésekkel kérdezhető le: az egyik a JavaScript *document.referrer* Document Object Model property-je, a másik pedig a User Agent, azaz a böngésző által beállított PHP-beli *$_SERVER* tömb HTTP*_REFERER* element-je. Az *$_SERVER* tömböt a web szerver hozza létre és számos különböző információt tartalmaz, a fejlécektől kezdve, az elérési utakon át, a különböző szkriptek tárolási helyéig. A PHP kódot nem tudom ellenőrizni, hiszen annak forrását nem fogom látni, viszont ha az adott oldal az először említett elemet JS segítségével kérdezi le, és a lekérdezett adatot fel is dolgozza, akkor biztosak lehetünk benne, hogy a weboldal tulajdonosai kíváncsiak arra, hogy honnan, melyik más page-ről érkeztünk a weblapjukra.

5.2 HTML iframe-ek használatának felderítése

A teszt egyik legfontosabb észrevétele kétségség kívül a HTML iframe tag-ek meglepően sikeres használata volt. A segítségükkel létrehozott HTML elemekbe

akármilyen külső, tehát más oldalról, illetve domain-ről származó dokumentumot, objektumot vagy különböző multimédiás tartalmat lehet elhelyezni. Fontos itt megjegyeznem, hogy az elnevezett, egyedi iframe-ekbe a target attribútum alkalmazásával a dokumentum bármely, box utáni pozíciójából lehetőség nyílik tartalmakat elhelyezni, még pedig a következő módon:

```
<iframe name="FoFrame" src="pelda1.htm"></iframe>
<!-- … -->
<a href="pelda2.htm" target="FoFrame">Pelda</a>
```

A fenti példából következően tehát nem elég csak azt vizsgálni, hogy van-e az adott oldalon iframe tag, hanem annak a nevét is fel kell jegyezni és azt is hozzá kell adni a keresési kulcsszavakhoz. Kérdéses esetekben a crawler alkalmazás lefutása után manuálisan is célszerű megnézni és ellenőrizni a nevesített frame-ek előfordulásait.

5.3 Webpoloskák keresése

Ha webpoloskákról esik szó, akkor többnyire 1x1 pixeles, átlátszó, GIF, vagy PNG kiterjesztésű képekre kell gondolnunk [5]. A keresés során legelőször is a megfelelő kép létrehozását vagy beincludolását végrehajtó részeket kell felderítenünk. HTML kódban erre az *img* tag lesz a segítségünkre, azaz ezek jelenlétét kell majd keresnünk. Szem előtt kell tartanunk azt is, hogy a modern webpoloskákat már nem csak a klasszikus módon helyezik az adott weblapon, hanem például HTML iframe, style, script, input link, embed, és object tag-ek segítségével is. Így nem hagyatkozhatunk csupán az *img* tag-ekre, hanem ha biztosra akarunk menni, akkor minden egyes megtalált PNG vagy GIF kiterjesztésű képet (és esetenként azok forrását) manuálisan is ellenőrizni kell. Ha a kódban találtunk ilyet, már erősen gyanakodhatunk, hogy internetes lehallgatóval van dolgunk.

5.4 Flash objektumok vizsgálata

A Flash típusú tartalmakat a vizsgált programok szép számban megjelenítették, így ezeket is egy lehetséges alternatívaként kell számon tartanunk a nyomkövetést illetően. Flash objektumokat kétféle módon lehet egy HTML dokumentumba ágyazni. A régebbi módszer az embedded object-tel történő beágyazás, míg az újabb és egyre elterjedtebb az iframe tag használatával végrehajtott beágyazás. Az előbbinél a keresési kulcsszavaink az *object*, illetve az *embed* lehetnek, míg az utóbbinál maga az *iframe* tag, sőt mindkettőnél használható, és (ha nem videóval van dolgunk) talán a legegyszerűbb

megoldás, ha a fájl kiterjesztésére, az *swf*-re keresünk rá. Flash objektumok dokumentumba ágyazása történhet JS segítségével is, viszont mivel a későbbiekben bemutatott crawler alkalmazás az oldalhoz kapcsolódó .js fájlokat is begyűjti, és azokban is keres, ezért ez nem jelenthet problémát.

5.5 HTML5 local és session tárak

Az új generációs, HTML5-ös helyi és viszony tárak figyelemre méltó szereplése következtében ezek ellenőrzését sem hagyhatom ki a weblapok vizsgálatából. A begyűjtött XML dokumentumokban először egy iframe tag-et kell keresni, majd ha ezt megtaláltuk, akkor a beágyazott HTML5 oldalon végigmenve a *localStorage* illetve a *sessionStorage* kulcsszavakat kell felhasználni, hogy kiderítsük, történik-e tudtunk nélküli információtárolás, illetve, hogy az adott tár tartalma el lesz-e küldve a szervernek egy későbbi HTTP kéréssel együtt. Fontos emlékeznünk, hogy ezen táraknál a weblapot fejlesztő dönti el, hogy melyik kulcs-érték párok kerüljenek csak elküldésre. Megjósolható, hogy ezek a tárolási módszerek, egyszerűségüknek köszönhetően a közeljövő legelterjedtebb tárolási technikái között lehetnek.

5.6 Kliensinformációk lekérése

Bár a különböző kliensinformációk begyűjtése a tesztben nem volt elég sikeres, viszont - mint az később látni fogjuk – érdemes volt hozzávenni a következő pontban található vizsgálat szempontjaihoz. Ezen információk ellenőrzése - mivel többféle nyelvről van szó, illetve ezek az adatok gép által csak nehézkesen lennének feldolgozhatók - kézi ellenőrzéssel történik, viszont csak azokban az esetekben, amelyeknél már legalább egy nyomkövetésre utaló jelet találtam.

6 Ismert technikák jelenlétének vizsgálata weboldalakon

A fejezet néhány látogatottabb weboldal elemzéséről és az abból adódó eredmények alapján, a lapokon fellelhető nyomkövetési technikák jelenlétéről fog szólni. A népszerűbb cégek legnagyobb része csak az adott statisztikákhoz feltétlenül szükséges információkra kíváncsi, például hogy hányan látogatták az oldalaikat, a felhasználók milyen rendszerességgel tértek vissza hozzájuk, illetve hogy látogatóik honnan milyen országból interneteznek, vagy éppen milyen oldalról tértek be hozzájuk. Vannak viszont olyan szervezetek is, mint ahogy az már említésre került, akik nem csak ennyire etikus módszerekkel, technikákkal igyekeznek személyes adatokat gyűjteni a felhasználókról. Ezen pont célja annak kiderítése lesz, hogy az ismertebb, és a tesztek során megfigyelésre alkalmasnak ítélt nyomkövetési technológiák milyen mértékben vannak jelen a vizsgált hírfolyamokon. A teszt során egy ún. crawler alkalmazás lesz a segítségemre.

6.1 Web robotokról általánosan

A webes crawler-ök (vagy más néven web pókok, web robotok) programok illetve automatizált szkriptek amelyek az internetet böngészik többféle célból kifolyólag [17]. A crawler programokat főleg arra használják, hogy egy naprakész másolatot készítsenek velük a meglátogatott oldalakról, amelyeket később egy kereső-motornak feldolgozásra és indexelésre küldenek el, a lehető leggyorsabb internetes keresés érdekében. Alkalmazási területükbe tartoznak még többek között a különböző karbantartási feladatok, a halott linkek illetve az adott weboldal változásainak felderítése, HTML kódok validálása, szerzői jogok megsértésének ellenőrzése, specifikus adatok begyűjtése, többféle multimédiás tartalom vagy objektum keresése, illetve bizonyos email címek címlisták alapján történő kinyerése is. Működését tekintve, a bot, vagy hívhatjuk akár szoftver-ügynöknek is, egy URL listát - ún. seedeket - kap bemenetként és azokon végigiterálva először az adott oldal elemeit ellenőrzi, felfedezi és azonosítja az oldalon található összes hiperhivatkozást, majd hozzáadja őket a meglátogatandó URL-ekhez – vagyis a frontierekhez -, ezzel bővítve saját listáját, és ezt folytatja rekurzívan újra és újra. A program az iteráció során HTTP kéréseket küld a szervereknek, hogy az adott dokumentumot lekérje, pontosan úgy, ahogy azt a

böngészőnk is teszi, amikor rákattintunk egy linkre. A crawler alkalmazásokat tehát automatizált hyperlink felderítőknek is tekinthetjük.

Ha a robot implementálására kerül sor, ügyelnünk kell rá, hogy mindenképp a megfelelő, és főként optimális keresési algoritmust valósítsuk meg. Így nem foglaljuk feleslegesen a rendelkezésre álló sávszélességet, és nem terheljük feleslegesen a linkelési lánchoz tartozó erőforrásokat sem. Ezen optimalitás megvalósításának könnyítésére találták ki a The Robots Exclusion Protocol[17]-t, vagy röviden robots.txt protokollt, amely segítségével egy szöveges fájlban megadhatjuk például, hogy melyek azok a könyvtárak, amelyeket mindegyik, vagy csak egy specifikus crawler hagyjon figyelmen kívül, azaz ne navigáljon olyan linkre, amely az adott könyvtárakra, vagy azokon belülre mutatnak. A robots.txt fájl tartalma a következő formában képzelhető el:

```
User-agent: Pelda-Bot
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /junk/
```

Jelen esetben a “Pelda-Bot” nevű robotnak azt javasoljuk, hogy ne nézze meg az adott könyvtárakat, hiszen ott nem lesz számára hasznos információ. Két probléma merül fel ezzel a protokollal kapcsolatban, az egyik, hogy csak ajánlás az oldalt meglátogató botnak, tehát ezt az iránymutatást figyelmen kívül is hagyhatja, a másik, hogy a robots.txt szöveges fájl publikus, tehát mindenki számára látható, hogy melyek azok a könyvtárak, amelyeket az oldal üzemeltetője nem szeretné, ha crawler-ök is felhasználnák.

6.2 Tematika és applikáció

A 6.3-as pontban részletezett tesztet két, kis mértékben elkülönülő részre osztottam. Az első rész a testreszabott crawler alkalmazás segítségével egy automatizált link-felderítés és adatgyűjtés, a második pedig azon elementek, linkek illetve különböző erőforrások manuális vizsgálata, amelyek ellenőrzését valamilyen okból kifolyólag automatizáltan nem lett volna érdemes megvalósítani. Szerencsések vagyunk, hiszen a mi esetünkben nem szükséges számtalan oldalt begyűjteni, hanem a csomópontok kifejtésénél csak egy adott mélységig kell elmennünk, így nem az alkalmazás optimalizáltsága fog előtérbe kerülni, hanem az, hogy minél hatásosabban működjön az erőforrások felderítése.

[17] http://www.robotstxt.org/robotstxt.html

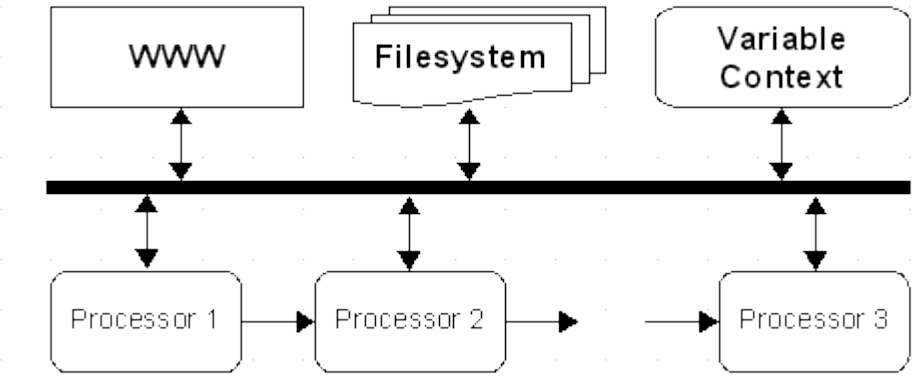
6.2.1 Felhasznált crawler alkalmazás

A számos crawler keresési, illetve begyűjtési algoritmus közül számunkra a legjobban hasznosíthatónak kétség kívül egy olyan robot-implementáció bizonyul, amely a megadott elérési utat csökkenő sorrendben bontja ki, természetesen csak minimális számú szintet feljebb lépve. Ebben az esetben a crawler alkalmazást arra használom fel, hogy a lehető legtöbb adatot letöltsem a megadott weblapról, a megadott URL-ben a legmélyebb ponttól kezdve fokozatosan felfelé haladva a könyvtárstruktúrában. Miután végeztünk az adott oldalhoz tartozó könyvtárak kibontásával, következhetnek azok az oldalak, amelyekre a weblapon egy vagy akár több hivatkozás található. Ennél a link-mélységnél meg is állhatunk, hiszen minket csak az adott oldal hírcsatornája érdekel, kivéve azokat az eseteket, amelyeknél a további felderítést egy specifikus erőforrás vagy tartalom nem indokolja. Az algoritmus erőssége, hogy segítségével hatásosan megtalálhatók a különböző nem izolált és izolált erőforrások egyaránt, illetve azok az erőforrások is, amelyekre a robot az előzetes információgyűjtés során egyáltalán nem talált hivatkozást.

Mivel a felhasználóbarát környezet megteremtése ma már alapkövetelmény egy adott weboldalon, ezért az interneten tárolt adatok szinte minden esetben formátaltan jelennek meg, amelyek egyáltalán nem mondhatók “robot-barát”-nak. Ha az érem másik oldalát nézzük, ott sem bíztatóbb a helyzet, hiszen az adatok manuális másolása és beillesztése nehézkes, hibákkal teli, és egyes esetekben szinte lehetetlen. Nem is beszélve arról, hogy a kód-összeolvasztások, design pattern-ök alkalmazása már sokszor szerveroldalon megtörténik, így a kimeneten csak egy csomag HTML weblap lesz a kliens számára hozzáférhető. A problémát a crawler alkalmazás fogja nekünk megoldani.

A méréshez egy szerb fejlesztőcsapat által implementált Web-Harvest alkalmazást, illetve API-t használtam fel [18]. A program egy Java alapú, nyílt forráskódú, tehát könnyedén testreszabható és fejleszthető webes adatgyűjtő eszköz, amely főleg a HTML/XML alapú weboldalakra koncentrál. Ehhez a mai adatbányászathoz feltétlenül szükséges összes technikát és technológiát ismeri és alkalmazza; ilyenek például az XSLT, Xquery, vagy a reguláris kifejezések. A Web-Harvest alkalmazásban minden kibontási procedúra felhasználó által definiált, egy XML alapú konfigurációs fájl használatával. Minden egyes konfigurációs fájl leírja a processzorok, azaz a feldolgozó-egységek számára, hogy mely taszkokat hajtsák végre

és azokat milyen sorrendben. A processzorok az adott taszkokat, ahogy az a 6–1-es ábrán is látható, csővezetékszerűen hajtják végre, tehát az egyik végrehajtó egység kimenete a másik egység bemenete lesz.



6-1. ábra: Pipeline végrehajtás [18]

Az adatbányászat illetve erőforrás-kibontás eredménye a végrehajtás során létrehozott fájlokban vagy a változó környezetből lesz elérhető. Egy rövid példa a konfigurációs fájl szemléltetésére:

```
<xpath expression="//a[@shape='rect']/@href">
<html-to-xml>
<http url="http://www.somesite.com/"/>
</html-to-xml>
</xpath>
```

A program a konfigurációs fájl ezen részének végrehajtásánál rendre a következő lépéseket hajtja végre:

- a HTTP processzor letölti a tartalmakat a megadott URL-ről
- a HTML-to-XML processzor tisztázza, hogy a HTML XHTML tartalmat fog létrehozni
- az Xpath végrehajtó megkeresi a specifikus linkeket az előző lépésben létrehozott XHTML dokumentumban, majd egy URL szekvenciát ad eredményül.

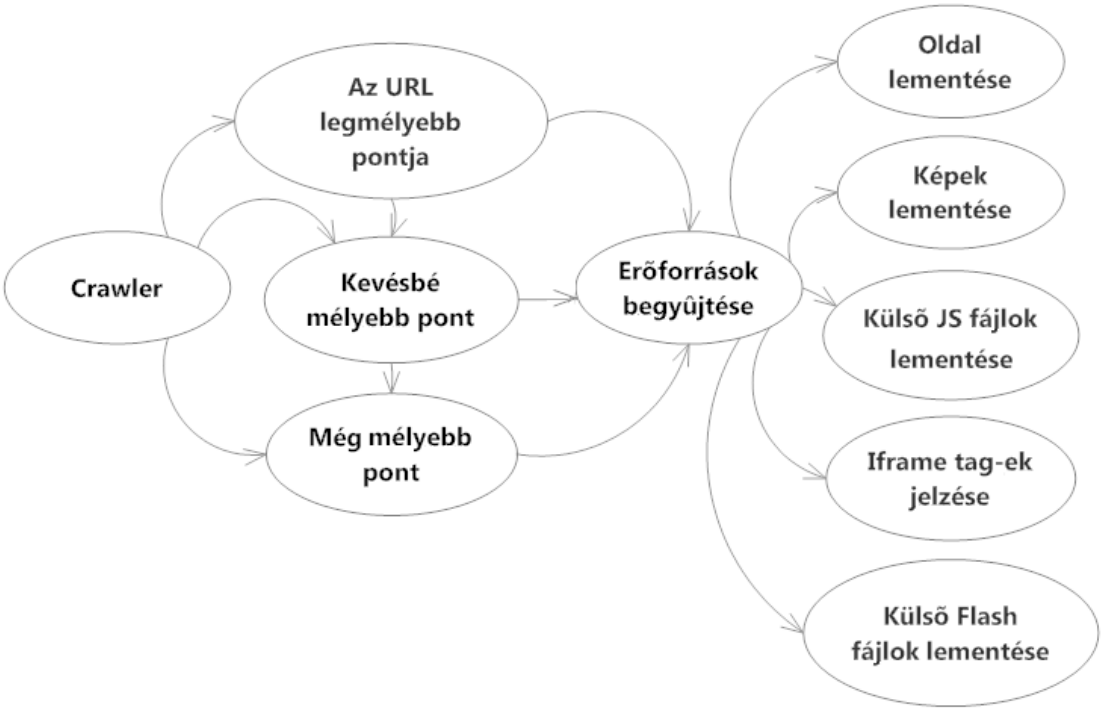
Az alkalmazás számos hasznos feldolgozó egységet támogat a változók manipulálása érdekében. Kezeli a feltételes elágazásokat, a loop-okat, függvény-funkciókat, fájl műveleteket, a HTML/XML lapok feldolgozását, sőt támogatja a kivételkezelést is.

Az elkészített konfigurációs fájlt az előző fejezet eredményei alapján állítottam össze. Öt különböző részt tartalmaz, amelyek rendre a következők:

- adott rss oldal lementése

- a lapon található képek lementése,
- külső JavaScript fájlok lementése,
- iframe tag-ek ellenőrzése
- külső Flash swf fájlok vizsgálata.

A crawler működése egy oldal esetén a 6-2. ábrán látható. Miután az alkalmazás végzett az adott lappal, azokon az oldalakon folytatja a kifejtést, amelyekre a weblap, valamelyik pontján legalább egy hivatkozást tartalmazott.



6-2. ábra: Crawler működése egy adott URL esetén

Mivel RSS csatornák esetében XML oldalakat kell begyűjtenünk, esetenként minimális számú HTML oldal kivételével, ezért hasznos funkció a HTTP fejlécek ellenőrzése, amelyek használatával csak akkor szükséges lekérnünk az adott oldalt, ha az az előbb említett két formátum egyikében íródott. Miután megtörtént a megfelelő lapok lekérése, a következő lépések az oldalak értelmezése (parse-olása), a teljes lap feldolgozása, a linkek megtalálása, majd azok listához hozzáadása, feltéve persze, hogy a megtalált link RSS vagy ATOM hírfolyam. A fenti bekezdésekben leírtak miatt tehát szükséges egy kisfokú domain korlátozás bevezetése is, hogy a crawler legfeljebb melyik domain-ig, illetve az adott domain-en melyik könyvtárig mehet el. Amennyiben ezt a korlátozást nem tennénk meg, akkor a program nagy valószínűséggel az egész

internet felderítésére vállalkozna, vagyis addig lépkedne a linkeken, amíg el nem akadnának, vagy amíg le nem állítanánk.

6.2.2 Teszt-weboldalak listája

A tesztelt lapok URL listájának összeállításánál a fő szempont az volt, hogy az oldalak a jelenleg globálisan legtöbbet látogatott, hivatalos és nagy szervezetek által üzemeltetett RSS oldalak közül legyenek kiválasztva és kerüljenek felderítésre illetve feldolgozásra. Az URL-ek több toplista alapján lettek összeállítva, többek között az AdPlanner mérései nyomán, a Google által felállított, legtöbbet látogatott oldalak listájából szemezgetve [19][20][21][22]. Ez a toplista az egyedi látogatók maximális száma alapján méri az adott weblapok népszerűségét. Hogy az itthoni vonatkozásokat is figyelembe vegyük, a listában nem kizárólag nemzetközi oldalak szerepelnek, hanem több magyar nyelvű weblap is szerepet kapott, hogy a hazai nyomkövetési helyzet illetve hozzáállás is minimálisan fel legyen térképezve. A tesztelt hírcsatornák listája a függelékben található.

6.3 A nyomkövetés-teszt eredményei

Az elemzett negyven RSS hírfolyamból tizenegynél lehetett találni valamely nyomkövetési technika alkalmazására utaló jelet. Az eredmény meglepőnek számít, tekintve, hogy az oldalak, szinte mindegyike hivatalos és nagy cég illetve társaság által üzemeltetett. A táblázatban ennek megfelelően csak azokat a webes hírfolyamokat tüntettem fel, amelyeknél legalább egy, a nyomkövetés szándékára utaló jel előfordult. Az oldalak neve utáni szám a Google által összeállított toplistán elfoglalt helyezéseket jelenti, persze csak abban az esetben ha az adott lap szerepelt a listán. Erre azért volt szükség, hogy fel lehessen mérni, hogy a követő oldalak mennyire népszerűek, mennyire látogatottak, illetve melyikek azok a nagyon népszerű oldalak, amelyek követnek.

		Nyomkövetési technikák alapja				
		URL referer	HTML iframe	Webpoloska	Flash objektum	Kliensinformációk
Hírfolyamok	ABC News (315.)			A		
	PC World (941.)			A		
	MSNBC			A		
	NY Daily News (832.)	x		A,P,B	x	x
	Blogthings			A		
	Slashdot		x	A		
	USA Today (397.)			A		
	Scientific American			A		
	FastCompany			A		
	Extremetech		x			x
	Microsoft (9.)	x		A,P,B	x	x

6-3. ábra: Nyomkövetési technikák jelenléte a hírcsatornákon

Láthatjuk, hogy a táblázatban szereplő hírcsatornák közül majdnem a fele rajta volt a Google Top 1000-es listáján, a Microsoft pedig ráadásul igen előkelő helyen.

6.3.1 URL referer jelenlétének vizsgálata

Az URL referer jelenlétét természetesen csak a kliensoldalon lehetett ellenőrizni, ebből kifolyólag, csak az XML illetve HTML kódokban tudtuk ezt megtenni. Két oldal esetében fordult elő, hogy a document.referer property-t haszálták volna, ez a két oldal pedig a Microsoft, illetve a NY Daily News volt. Mindkét hírfolyam a könyvtárstruktúrában az RSS csatornához képest feljebb elhelyezett szkripteket haszálta, így a crawler alkalmazás ezeket is kifejtette és lementette a manuális ellenőrzésre.

6.3.2 HTML iframe-ek használatának felderítése

Iframe-eket a vizsgált listából mindössze két hírcsatorna használt. Az Extremetech a létrehozott boxokat csak különböző videó-megosztó oldalakról származó tartalmak megjelenítésére használta, viszont a Slashdot folyama nyomkövető hirdetéseket helyezett el bennük.

6.3.3 Webpoloskák keresése

A tesztelt oldalak túlnyomó többsége az 1x1-es méretű képeket csupán az adott HTML elementek helyre igazítására, illetve az elhelyezés finomhangolására használta, azaz spacer-ként alkalmazta őket. Az weblapok közül a fentiekben már említett NY Daily News és a Microsoft állított be és ellenőrzött sütiket. Elég sok oldalnál előfordult, hogy nem webpoloskákat, hanem nyomkövető hirdetéseket helyeztek el a lap valamely részén és ezek a hirdetések túlnyomó többségben csak akkor küldtek információt a tulajdonosaiknak, ha a felhasználó rájuk kattintott. Meg kell jegyezni, hogy a táblázatban a nyomkövető hirdetéseket, a webes poloskákat, és a web beacon-öket a következő betűkkel jelöltem: A, P, B.

6.3.4 Flash objektumok vizsgálata

Behívott swf kiterjesztésű fájlok, beágyazott hirdetések, videók, vagy egyéb, Flash alapú objektumok szinte egyik esetben sem fordultak elő az ellenőrzött hírcsatornákon. A Microsoft és a NY Daily News odalak viszont tartalmaztak olyan szkripteket is, amelyek például a felhasználó számítógépén telepített Flash vagy Silverlight plugin verziójára voltak kíváncsiak.

6.3.5 HTML5 local és session tárak

A vizsgált feed-ek közül egyetlen oldal sem tartalmazott HTML5 alapú oldalra mutató linkeket, így a helyi- és viszony tárakat sem használta egyik lap sem. Ez nem meglepő, hiszen az új szabvány illetve az abban alkalmazott technológiák még nagyon újak és ebből kifolyólag nem annyira elterjedtek mint az előző HTML verzió esetében. A teszt eredménye ennek megfelelően a táblázatban sem lett feltüntetve.

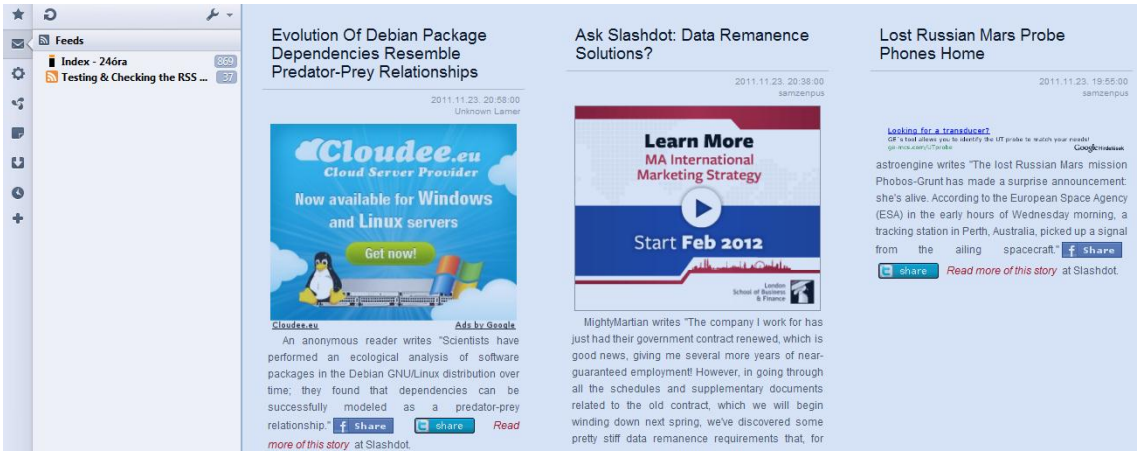
6.3.6 Kliensinformációk

A különböző kliensinformációk megszerzése volt a harmadik legnépszerűbb adatgyűjtési metódus a webpoloskák és a követő hirdetések után. Az adatok

természetesen JavaScript felhasználásával lettek lekérve a felhasználótól. Az oldalak, a gyűjtési folyamat során nem csak a különböző telepített beépülők verziójára, a böngészők típusára, a felhasználó számítógépén a futó operációs rendszerre, az asztali felbontásra, annak bitmélységére, hanem számos más elérhető információra is kíváncsiak voltak.

6.3.7 Egyéb észrevételek

Várható volt, hogy számos hírfolyam használni fog hirdetéseket a kódba ágyazva, viszont talán elsőre nem volt ennyire egyértelmű, hogy ezek a hirdetések szinte minden esetben nyomkövetési célzattal és a felhasználó szokásait illetve ízlését meghatározandó kerülnek elhelyezésre. Megjegyzendő (bár nem meglepő tény), hogy azokon a hírfolyamokon lévő hirdetések, amelyek a Google-től származnak, még a Google Reader-ben sem kerültek szűrésre.



6-4. ábra: Google hirdetések a Slashdot csatornán

A fenti kép az Opera RSS-olvasó programjával megjelenített Slashdot hírfolyamot ábrázolja. Ilyen és ehhez hasonló nyomkövető hirdetéseket lehetett látni a vizsgált oldalak elég nagy százalékánál.

6.4 Kiemelt esetek

Érdemes kiemelni két meglehetősen kirívó, feltűnő esetet, amelyek elég szép számban alkalmaztak valamilyen nyomkövetési és információgyűjtési technikát. Mindkét oldal a tracking függvényeket az adott domain-ek könyvtárstruktúrájában feljebb elhelyezett JavaScript fájlokban tárolta. Az alábbiakban említett hírfolyamok által használt technikák kódrészletei a függelékben találhatók.

6.4.1 NY Daily News

A NY Daily News az egyik legjobb példa arra, hogy napjainkban milyen nyomkövetési technológiákat lehet, és szokás alkalmazni a webes hírcsatornákon. A hírfolyamon a GoogleAd hirdetések mellett saját tracking scriptek is előfordultak a crawler által begyűjtött *ywa.js* fájlban. A fájl tartalmaz többek között document.referer property használatot, süti-beállítást illetve olvasást, a böngésző típusának lekérdezését, és egyéb telepített pluginek, főleg Flash és Silverlight információk gyűjtését is.

6.4.2 Microsoft

A Microsoft esetében letöltött szkriptek talán a legszemléletesebbek, a függvények beszédes neve, illetve az első néhány kikommentezett sor miatt. Követési metódusai definiálására az oldal két JavaScript fájlt használt, amelyek a következők: *ms.js* illetve a *wt.js*. Az első szkriptnél a következő módszereket figyelhetjük meg: web beacon (amely a webpoloska és süti-beállító is egyben), Silverlight és Flash adatok beszerzése, sőt egyéb kliensinformációk gyűjtése is. A második esetben – az előbbiekben felsoroltakon kívül – előfordulnak olyan függvények, amelyek például a böngésző nyelvét, a monitor képernyő-felbontását illetve a Java Virtual Machine engedélyezettségét kérdezik le. A Microsoft, a megszerzett adatokat a függény neve (WebTrends) alapján, a felhasználók által használt alkalmazások és plugin-ek feltérképezésére, illetve azokból kimutatások készítésére használja fel.

6.5 Tendencia

Összességében elmondható, hogy a vizsgált oldalak majdnem egyharmad részében találtam valamilyen információszerzésre utaló nyomot, amelyek, mint azt később láthattuk, be is igazolódtak. A táblázatban szereplő tizenegy oldal mindegyikén lehetett találni követő hirdetést vagy webes poloskát illetve web beacon-t, illetve elég népszerű módszernek bizonyult még a különböző kliensinformációk begyűjtése is. Érdekes volt látni, hogy a táblázatban szereplő oldalak közül öt oldal rajta volt a Google toplistáján, amely azt bizonyítja, hogy a nyomkövető oldalak kellően népszerűek ahhoz, hogy nagy veszélyt jelentsenek a jövőre nézve.

A tendencia azt mutatja, főleg a két kiemelt oldal, a Microsoft, illetve a NY Daily News példájából kiindulva, hogy azok a weblapok, amelyek egy kliensinformáció-szerzési technikát alkalmaztak, azok többet is. Nem volt tehát olyan

hírfolyam, amelyik például csak a user monitorának felbontását szerette volna megtudni, mert egy kimutatás, illetve teszt céljából arra szüksége lett volna (egy szavazás ugyan etikusabb, viszont a kézzel fogható eredmények csak később kiértékelhetőek), hanem a lehető legtöbb információt próbálta megszerezni a felhasználó számítógépéből. A teszt másik, talán legszembeötlőbb eredménye kétségkívül a webpoloskák, web beacon-ök és hirdetések rendszeres, illetve nagy mértékű használata volt.

Felmerülhet bennünk a kérdés, hogy miért is jó ez mindkét félnek: a hirdetőknek, illetve azoknak a szervezetnek, amelyek az adott hirdetést elhelyezik az oldalukon. A hirdetők fő célja az, hogy a hirdetéseiket minél többen megnézzék, illetve minél többen kattintsanak rájuk, hiszen ebből származik a profitjuk, sőt mindkét fél profitja [12]. Az elhelyező cégek a teljesítményük alapján szerzik a bevételüket, tehát, hogy hányan navigáltak a hirdetett oldalra az ő oldalukról egy adott időszak alatt. Ezek az oldalak, annak érdekében, hogy a lehető legtöbb érdeklődőt csábítsák a hirdetéseik megnyitására, felhasználói profilokat szereznek be (profilírozással vagy akár vásárlással), hogy ezen információk alapján személyre szabott hirdetéseket tudjanak létrehozni és megjeleníteni a felhasználók számára. Abban az esetben, ha a hírfolyamokon alkalmazott hirdetések megmaradnak ilyen formában, azaz nem keltenek nagy feltűnést, nem lehetetlenítik el a hírek olvasását, és nem használnak felugró (pop-up) ablakokat, akkor ezek a reklámok még némely esetben hasznosnak is bizonyulhatnak.

Már csak egy fontos kérdés maradt hátra; a haszonszerzésen kívül pontosan mire is használják még fel az oldalak a megszerzett felhasználói adatokat? A Microsoft esetében a tevékenység célja a fent leírtak szerint kitalálható, viszont a másik, szóban forgó oldalnál (a NY Daily News-nál) erre csak következtethetünk. Mindkét oldal célja valószínűleg az aktuális webes trendek felmérése, és az ahhoz kapcsolódó opcionális profilépítés lehet; illetve ezen cégek számára az információk nagy adatbázisokban eltárolása, statisztikákban összesítése és szemléltetése is fontos szempont lehet, hiszen ezeknek köszönhetően könnyedén elemezhetővé válnak az összegyűjtött adatok.

6.6 Védekezés most és a jövőben

Látható, hogy a privátszféra, illetve a privátszféra védelme egyre fontosabbá válik az életünkben, a mindennapjainkban hiszen olyan óriásvállalatok működése mellett, mint a Facebook vagy a Google egyre jobban kell figyelnünk arra, hogy ne

adjunk meg magunkról információkat bizonytalan eredetű oldalakon, és ami még fontosabb, ügyelnünk kell arra is, hogy ne is tudjanak adatokat lopni illetve begyűjteni rólunk. Fontos, hogy olyan technikák, alkalmazások illetve beépülők is szóba kerüljenek, amelyek használatával sikeresen vehetjük fel a versenyt a különböző, privátszférát megcélzó támadások ellen. Az alábbiakban néhány javaslatot írok le a bemutatott sebezhetőségek orvoslására.

A vizsgált böngésző és hírolvasó alkalmazások, illetve weboldalak közül a privátszféra védelmét tekintve a leghatékonyabbanak kétség kívül a Google Reader bizonyult. Az oldal a saját hirdetésein kívül minden, esetleg kártékony céllal elhelyezett tartalmat szűrt a tesztek folyamán, így elnyerte a legnagyobb védelemmel rendelkező olvasó díját is. Az optimális biztonsági funkciókkal rendelkező hírolvasó használata mellett ügyelnünk kell arra is, hogy a megfelelő beépülőket is feltelepítsük az alkalmazáshoz. Az összes vizsgált böngészőhöz telepíthetőek ilyen plugin-ek, és ezek közül is olyanokat érdemes beszereznünk, amelyek a megfelelő módon tudják szűrni a veszélyes tartalmakat. Az alábbi plugin-ek illetve alkalmazások mind-mind ezt a funkciót hivatottak szolgálni [23]:

- pop-up blokkoló: linkek megnyitásakor a felugró ablakok blokkolására használható
- phising védelem: célzott adathalászat ellen nyújt védelmet, a spoofed (hamisított) weboldalak kiszűrésével
- webploska blokkoló: eltávolítja (illetve nem is tölti le) az adott lapon található webes lehallgatókat
- webploska jelző: jelzi az oldalon található webpoloskákat
- reklámszűrő: blokkolja az oldalon található hirdetéseket, azaz szimplán nem jeleníti meg őket
- dereferrer: kitörli az URL referer értékét, így nem lehet tudni hogy honnan érkeztünk az adott oldalra
- User Agent String cserélő: böngésző azonosítójának megváltoztatása, elfedése
- flash blokkoló: a flash tartalmak nem futnak automatikusan, csak akkor ha engedélyezzük őket
- proxy szerver: a valódi kliens és kliens által küldött HTTP kérések elfedése egy, a kommunikáció közé ékelt szerver segítségével

- anonim böngészők: webes, vagy alkalmazáson belül beállítható proxy-k, amelyek például titkosítási és tartalomszűrési szolgáltatásokat nyújtanak
- tűzfal: hálózaton keresztüli illetéktelen behatolások megakadályozására alkalmas
- vírusirtó: a számítógépen lévő vírusok, különböző szkriptek és tracking sütik távolíthatók el vele; célszerű valós idejű védelmi funkciókkal használni
- egyéb módszerek: természetesen számos más módszer illetve bővítmény is létezik védelmünk maximalizálására

A számítógépünkre történő behatolást, illetve a webes mozgásunk megfigyelését nem akadályozhatjuk meg, erre univerzális módszer egyelőre nem létezik, viszont az előbb említett bővítmények, illetve alkalmazások használatával jócskán megnehezíthetjük a támadók, információ-gyűjtők dolgát.

A jövő webes biztonságát illetően nem túl bíztató a helyzet. A jövő számítógépei valószínűleg a mostaniakhoz képest jóval nagyobb feldolgozási sebességgel, tárhely- illetve memóriakapacitással fognak rendelkezni. A védelmi mechanizmusok sajnos nem a megfelelő ütemben fejlődnek, illetve ez a fejlődés a támadó oldalon sokkal nagyobb mértékű. Egyesek szerint a jövőben globális rendszerek fogják biztosítani a felhasználók rendelkezési jogát személyes adataik felett [12]. Egy ilyen megoldás koncepciója az ún. PRIME[18] projekt, amely egy átfogó, a privátszféra védelmét erősítő identitás menedzsment rendszer felépítését tűzte ki célul. Utódja a PrimeLife[19] projekt.

6.7 Crawler fejlesztési lehetőségek

Érdemes néhány szót ejteni arról is, hogy az elkészített crawler alkalmazás milyen további fejlesztési lehetőségeket tartogat számunkra. A meglévő program magját egy kiforrottabb architektúrába átemelve, egy komplexebb, optimalizáltabb és nagyobb volumenű adat-begyűjtőt lehetne elkészíteni. Ezalatt többek között a feldolgozás sebességét, a felismerhető technikák körének bővítését, és a csomópontok kifejtésének még optimalizáltabb végrehajtását kell érteni. A csomópontok kifejtését egy dinamikusan változó domain-korláttal lehetne optimalizáltabbá tenni, így a cralwer nem fejtene ki feleslegesen sok csomópontot, illetve nem kerülne végtelen ciklusba sem. Hasznos funkció lenne még, hogy az odalakon található linkek segítségével kifejtett

[18] https://www.prime-project.eu/

[19] http://www.primelife.eu/

csomópontokból egy gráf kerülne felépítésre, így ezen gráf segítségével könnyedén fényt lehetne deríteni az egyes oldalak, illetve az odalakon belüli erőforrások kapcsolatára is. A csomópontok kifejtésének sebességét a futtatási szálak emelésével lehetne növelni, tehát a program így többszálúvá válhatna. A felismerhető technikák körét is hasznos lenne bővíteni, illetve hogy az egyes lementett szkripteket ne kelljen manuálisan ellenőrizni. Az alkalmazás azt is megvizsgálhatná, hogy az adott fájlban milyen információszerzési technikák kerültek használatra.

7 Összefoglalás

Jelen munka legelső egysége a hagyományos webes nyomkövetési technikák bemutatásának lett szentelve. Még a munkálatok elkezdése előtt körvonalazódott, hogy ezen technológiák kizárólagos használata nem lesz elegendő a számunkra, hiszen egyrészt a tesztet a hírfolyamok aspektusából kell elvégezni, másrészt a vizsgálatok nem csak a jelenleg elterjedt módszerekre kell, hogy koncentráljanak, hanem a kibontakozóban lévőkre is. Ezért kerültek a teszt szempontok közé többek között a HTML5 storage módszerek, az Evercookie néhány meggyőző tárolási funkciója illetve egyéb, nem teljesen szokványosnak mondható kliensinformációk begyűjtése is.

A komplexebb és átfogóbb eredmények érdekében, a fő vizsgálatokat két részre osztottam: az első egy általános, a böngészők és hírolvasó alkalmazások alapvető funkcióit leellenőrző teszt volt, míg a második egy specifikusabb, kifejezetten a hírcsatornák aspektusából történő teszt volt. Az eredményeket táblázatosan összefoglaltam, ezt követően részletesebben megnéztem azokat a módszereket, amelyek a vizsgálatok alapján nyomkövetésre alkalmasnak bizonyultak. A tesztek fontos felfedezésnek számított az iframe-ek egyszerű alkalmazása, hiszen ez a különböző tartalmak beágyazását nagy mértékben megkönnyítette.

A következő, nagyobb lélegzetű rész az előző pontokban bemutatott technikák alkalmazásán alapult. Megnéztem, hogy ezek a technológiák mennyire jól használhatók, népszerű, illetve nagy cégek által üzemeltetett hírfolyamokon. Ahhoz, hogy ezen tesztet hatékonyan végrehajthassam, szükségem volt egy adat-begyűjtő, azaz crawler alkalmazás elkészítésére is. A program konfigurációs fájlának elkészítésekor – amely a kifejtést vezérelte – nem az optimalitás volt a fő szempont, sokkal inkább az hogy a lehető legtöbb nyomkövetésre utaló jelet felkutassuk és feldolgozzuk. Végül a crawler segítségével begyűjtöttem a mellékelt listában szereplő negyven oldalt, majd azok alapján megbecsültem a nyomkövetési technikák webes jelenlétét.

A tendencia azt mutatja, hogy a hírcsatornák csak kis százaléka alkalmaz nyomkövetési módszereket, és hogy csak a nagyobb weboldalak fertőzöttek jelentősen, a kisebbek kevésbé. Az eredmény két oldalnál kiemelkedő, a Microsoft illetve a NY Daily News esetében. Mindkét oldal széleskörű nyomkövetést használ, illetve az általuk alkalmazott technikák – mint az a böngésző- és hírolvasó tesztből is kiderült - tekinthetők információgyűjtés szempontjából a legelterjedtebbnek. Az előbb említett két oldal célja a megszerzett felhasználói adatokkal feltehetően az aktuális webes trendek

felmérése, és az ahhoz kapcsolódó opcionális profilépítés; ezeket a profilokat pedig vagy saját maguk használják fel, vagy pedig más szervezeteknek adják el hasznuk maximalizálása érdekében.

Ábrajegyzék

Irodalomjegyzék

[1] Gulyás Gábor, Schulcz Róbert, Imre Sándor, „Comprehensive Analysis of Web Privacy and Anonymous Web Browsers”. Proceedings of the Joint SPACE and TIME Workshops, 2008.

[2] Székely Iván, „A privátszférát erősítő technológiák”, Információs Társadalom 2008. VIII. évf. 1. szám, 20-34. old.

[3] Paulik T., Földes Á.M., Gulyás G.Gy, „Privát Adatok publikálása a weben” Alma Mater, 2011

[4] A. Soltani, S. Canty, Q. Mayo, L. Thomas, C. J. Hoofnagle, „Flash Cookies and Privacy”
Elérhető: http://ssrn.com/abstract=1446862
2009.

[5] Hullám Gábor, „A web bug technológia — barát vagy ellenség?”, BME ITM
2005. március

[6] Evercookie - virtually irrevocable persistent cookies [Online]
http://samy.pl/evercookie/
Megtekintve: 2011. szeptember 28.

[7] PET Portál és Blog, Fingerprint [Online]
http://pet-portal.eu/fingerprint/
Megtekintve: 2011. október 18.

[8] RSS Wikipedia [Online]
http://en.wikipedia.org/wiki/RSS_(file_format)
Megtekintve: 2011. október 17.

[9] W3Schools, Browser Statistics [Online]:
http://www.w3schools.com/browsers/browsers_stats.asp
Megtekintve: 2011. október 1.

[10] Michael Muchmore, „Internet Explorer 9” [Online]
http://www.pcmag.com/article2/0,2817,2369163,00.asp
Megtekintve: 2011. október 5.

[11] FeedDemon [Online]
http://www.feeddemon.com/
Megtekintve: 2011. október 6.

[12] Gulyás Gábor György, „Anonim-e az anonim böngésző?”
2006. március, Alma Mater

[13] Arpan Dhandhania, “HTML 5: Client-side Storage” [Online]
http://www.webreference.com/authoring/languages/html/HTML5-Client-Side/
Megtekintve: 2011. október 10.

[14] O'Reilly, Tim, „What Is Web 2.0” http://www.oreilly.com. [Online]
http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
2005. szeptember 30.

[15] Rich Internet Application Statistics [Online]
http://riastats.com/
Megtekintve: 2011. október 2.

[16] Safari Features [Online]
http://www.apple.com/safari/features.html
Megtekintve: 2011. október 4.

[17] Shalin Shah, “Implementing an Effective Web Crawler” [Online]
http://www.devbistro.com/articles/Misc/Implementing-Effective-Web-Crawler
Megtekintve: 2011. november 3.

[18] Web-Harvest Introduction [Online]
http://web-harvest.sourceforge.net/
Megtekintve: 2011. november 20.

[19] Google, “The 1000 most-visited sites on the web” [Online]
http://www.google.com/adplanner/static/top1000/
Megtekintve: 2011. november 6.

[20] Live Journal [Online]
http://www.livejournal.com/syn/list.bml
Megtekintve: 2011. november 6.

[21] Bloglines [Online]
http://www.bloglines.com/topblogs
Megtekintve: 2011. november 6.

[22] A magyar web legjava [Online]
http://www.topmagyar.com/
Megtekintve: 2011. november 6.

[23] List of Firefox Extensions [Online]
http://en.wikipedia.org/wiki/List_of_Firefox_extensions
Megtekintve: 2011. november 27.

Függelék

Csoportok	Alcsoportok	Fejléc típusok	Böngészők					Hírolvasók		
			Chrome	Firefox	Internet Explorer	Opera	Safari	Google Reader	FeedDemon	Readifine Desktop
Kliensinformációk	Elküldött HTTP fejlécek	Accept	x	x	x	x	x	x	x	x
		Accept-Charset	x	x						
		Accept-Encoding	x	x	x	x	x	x	x	x
		Accept-Language	x	x	x	x	x	x	x	x
		Authorization								
		Cache-control	x							
		Connection	x	x	x	x	x	x	x	x
		Cookie								
		Content-Length								
		Content-MD5								
		Content-Type								
		Date								
		Expect								
		From								
		Host	x	x	x	x	x	x	x	x
		If-Match								
		If-Modified-Since								
		If-None-Match								
		If-Range								
		If-Unmodified-Since								
		Max-Forwards								
		Pragma								
		Proxy-Authorization								
		Range								
		Referer	x	x	x	x	x	x	x	
		TE								
		Upgrade								
		User-Agent	x	x	x	x	x	x	x	x
		Via								
		Warning								

Elküldött HTTP fejlécmezők

URL lista:

1. Yahoo! News: http://news.yahoo.com/rss/world
2. CNN: http://rss.cnn.com/rss/edition.rss
3. MSNBNC: http://rss.msnbc.msn.com/id/3032091/device/rss/rss.xml
4. Google News: http://news.google.com/news?pz=1&cf=all&ned=us&hl=en&output=rss
5. New York Times: http://feeds.nytimes.com/nyt/rss/World
6. Fox News: http://feeds.foxnews.com/foxnews/most-popular
7. Digg: http://digg.com/news.rss
8. NY Daily News: http://www.nydailynews.com/index_rss.xml
9. ABC News: http://feeds.abcnews.com/abcnews/topstories
10. BBC News: http://feeds.bbci.co.uk/news/rss.xml
11. Wall Street Journal: http://online.wsj.com/xml/rss/3_7085.xml
12. MSN: http://msn.com/rss/news.aspx
13. Microsoft: http://www.microsoft.com/presspass/rss/RSSFeed.aspx
14. Slashdot: http://rss.slashdot.org/Slashdot/slashdot
15. Apple – Top 10 Just Added Tunes: http://ax.itunes.apple.com/WebObjects/MZStore.woa/wpa/MRSS/justadded/limit=10/rss.xml
16. Formula1: http://www.formula1.com/rss/news/latest.rss
17. USA Today: http://rssfeeds.usatoday.com/usatoday-NewsTopStories
18. Fast Company: http://www.fastcompany.com/rss.xml
19. Stereogum: http://feeds.feedburner.com/stereogum/cBYa
20. Extremetech: http://www.extremetech.com/feed
21. About Animals / Wildlife: http://z.about.com/6/g/animals/b/rss2.xml
22. Cool Tools: http://feeds.feedburner.com/CoolTools
23. Simply Recipes: http://feeds.feedburner.com/elise/simplyrecipes
24. Popsugar: http://feeds.feedburner.com/popsugar
25. MTV News: http://www.mtv.com/rss/news/news_full.jhtml
26. Fanblogs: http://www.fanblogs.com/site.xml
27. Scientific American: http://rss.sciam.com/ScientificAmerican-News
28. Pc World Latest Technology News: http://feeds.pcworld.com/pcworld/latestnews
29. Postsecret: http://postsecret.blogspot.com/feeds/posts/default
30. Blogthings: http://feeds.blogthings.com/blogthings/NewestQuizzes
31. XKCD: http://www.xkcd.com/rss.xml
32. Penny Arcade: http://penny-arcade.com/feed
33. Prohardver – Hírek és tesztek: http://prohardver.hu/hirfolyam/anyagok/rss.xml
34. HWSW: http://www.hwsw.hu/xml/latest_news_rss.xml
35. Gamestar: http://hu.gamestar.feedsportal.com/c/33352/f/566121/index.rss
36. Index: http://index.hu/x?t=/24ora/rss/
37. Portfólió: http://www.portfolio.hu/rss/cikkek/all.xml

38. Hvg.hu: http://www.hvg.hu/rss
39. Totalcar: http://totalcar.hu/24ora/rss/
40. Köpönyeg: http://www.koponyeg.hu/idojaras_rss.php?regios=1

NY Daily News, ywa.js fájl részletei:

```
//…
YWAT.prototype.setCookie = function (name, value, off) {
    var expiry, cookie, d;
    d = new Date();
    d.setTime(d.getTime() + (off * 1000));
    expiry = (off > 0) ? "; expires=" + d.toGMTString() : "";
    if (off < 0) {
        expiry = "; expires=Thu, 01-Jan-1970 00:00:01 GMT";
    }
    cookie = name + "=" + value + expiry + "; path=" + this.FPCP + ((this.FPCD
!== "") ? ("; domain=" + this.FPCD) : (""));
    document.cookie = cookie;
};
YWAT.prototype.deleteCookie = function (name) {
    return this.setCookie(name, "1", -1);
};
YWAT.prototype.getCookie = function (name) {
    var start, end, dc, pos;
    dc = document.cookie;
    pos = dc.indexOf(name + "=");
    if (pos !== -1) {
        start = pos + name.length + 1;
        end = dc.indexOf(";", start);
        if (end === -1) {
            end = dc.length;
        }
        return dc.substring(start, end);
    }
    return "";
};

//…

YWAT.prototype.track = function (d, i) {
    var cs, pt, t, r, its;
    t = "";
    r = document.referrer;
    YWA.errorId = this.idx;
    its = [];
    if (YWA.is(this.REFERRER) && this.REFERRER.length > 0) {
        r = this.REFERRER;
    } else {
```

```
        if      ((navigator.userAgent.indexOf("Mac")      >=      0)      &&
(navigator.userAgent.indexOf("MSIE 4") >= 0)) {
            r = document.referrer;
        } else {
            if (d) {
                YWA.windowOnerror = window.onerror;
                window.onerror = YWA.ywaOEH;
                if (document.location !== top.location) {
                    try {
                        r = top.document.referrer;
                        t = top.location.href;
                    } catch (e2) {
                        its[0] = "&nr=webkit";
                    }
                }
            } else {
                its[0] = "&nr=t";
            }
        }
    }
    if (YWA.windowOnerror) {
        window.onerror = YWA.windowOnerror;
    } else {
        window.onerror = null;
    }
    this.pp();
//…
};
```

Microsoft, ms.js részletek:

```
//..
function MscomGetSlvVersion(control) {
    var slv = "",
        slVMax = (new Date).getYear() - 2004;
    for (var i = slVMax; i > 0; i--) for (var j = 9; j >= 0; j--) {
        slv = i + "." + j;
        if (control.IsVersionSupported(slv)) return slv
    }
    return slv
}
function MscomHandleSession() {
    var currDate = new Date;
    tz = currDate.getTimezoneOffset();
    var currTime = currDate.getTime();
    currTimeAsString = currTime.toString();
    var cookiePre = "",
        index = document.cookie.indexOf(sessionCookieName + "=");
    if (index == -1) {
```

```
        sessionId = currTimeAsString;
        if (cookieDisabled == 1) return;
        cookiePre = sessionCookieName + "=" + sessionId
    } else {
        var start = index + sessionCookieName.length + 1,
            end = document.cookie.length;
        cookiePre = sessionCookieName + "=" + document.cookie.substring(start, end)
    }
    document.cookie = cookiePre;
    index = document.cookie.indexOf(sessionCookieName + "=");
    if (index == -1) cookieDisabled = 1
}
//..
function MscomBeacon() {
    try {
        var src = [];
        src.push(window.location.protocol + "//c.microsoft.com/trans_pixel.aspx?");
        pvInfo.length = 0;
        MscomSetPVInfo();
        src.push(pvInfo.join(""));
        if (clickInfo != "") src.push(clickInfo);
        MscomInitMeta();
        if (metaTags != "") src.push(metaTags);
        if (customTags != "") src.push(customTags);
        if (customInfo != "") {
            src.push("&cot=5");
            src.push(customInfo)
        }
        var srcString = src.join("");
        if (srcString.length > 2048) srcString = srcString.substring(0, 2042) + "&tr=1";
        if (document.images) {
            imgArray[imgArrayIndex] = new Image;
            imgArray[imgArrayIndex].src = srcString;
            imgArrayIndex++
        } else document.write('<IMG ALT="" BORDER="0" NAME="bImg" WIDTH="1" HEIGHT="1" SRC="' + srcString + '"/>');
        clickInfo = "";
        customInfo = ""
    } catch (e) {}
}
//..
```

Microsoft, wt.js részletek:

```
WebTrends.prototype.dcsVar=function(){
      var dCurrent=new Date();
      var WT=this.WT;
```

```
4         var DCS=this.DCS;
5         WT.tz=parseInt(dCurrent.getTimezoneOffset()/60*-1)||"0";
6         WT.bh=dCurrent.getHours()||"0";
7         WT.ul=navigator.appName=="Netscape"?navigator.language:navigator.userLangu
  age;
8         if (typeof(screen)=="object"){
9
          WT.cd=navigator.appName=="Netscape"?screen.pixelDepth:screen.colorDepth;
10           WT.sr=screen.width+"x"+screen.height;
11        }
12        if (typeof(navigator.javaEnabled())=="boolean"){
13           WT.jo=navigator.javaEnabled()?"Yes":"No";
14        }
15        if (document.title){
16           if (window.RegExp){
17              var                                                          tire=new
  RegExp("^"+window.location.protocol+"//"+window.location.hostname+"\\s-\\s");
18              WT.ti=document.title.replace(tire,"");
19           }
20           else{
21              WT.ti=document.title;
22           }
23        }
24 //..
25 }
```