MASARYKOVA UNIVERZITA V BRNĚ
FAKULTA INFORMATIKY

Ochrana webových serverů proti vybraným útokům

DIPLOMOVÁ PRÁCE

Michal Šafránek

Brno, podzim 2005

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: doc. RNDr. Václav Matyáš, M.Sc., Ph.D.

Poděkování

Mnohokrát děkuji svému vedoucímu doc. RNDr. Václavu Matyášovi, M.Sc., Ph.D. za jeho trpělivost, shovívavost, příkladné vedení, cenné rady, připomínky, pomoc a všechen čas, který mi věnoval.

Dále děkuji svým rodičům za jejich trpělivost, pochopení, lásku a obětavost, bez kterých si nedokážu představit svůj život takový, jaký dnes je.

Můj dík za pomoc při testování a kontrole obsahu práce patří také mým kolegům z firem Web4U s.r.o. a Communicon s.r.o., zejména Martinu Kukačkovi, Ivě Urbanové a Otu Zimmermannovi.

Shrnutí

Tato práce popisuje problematiku ochrany webových serverů proti útokům typu odepření služby (*denial-of service*, dále jen „DoS“) a útokům na autentizační mechanismy. Na základě těchto informací poté navrhuje seznam opatření, která dopad DoS útoků snižují. Následně je též navržen nový autentizační mechanismus, který je odolný proti většině běžných útoků. Jedním z druhotných cílů při návrhu tohoto mechanismu byla zpětná kompatibilita na úrovni rozhraní vůči existujícím webovým aplikacím.

V úvodní kapitole je čtenář seznámen s principem fungování TCP/IP a HTTP. Následuje kapitola popisující útoky na HTTP, která je rozdělena na dvě části – DoS útoky a útoky na autentizační mechanismy. Další kapitola podrobně rozebírá jednak evoluci návrhu nového autentizačního mechanismu a jednak jeho konečnou verzi, která je následně podrobena analýze z hlediska ochrany proti dříve popsaným útokům. Kapitola též obsahuje detaily referenční implementace. Předposlední kapitola popisuje výsledky testování navrženého autentizátoru v praxi – test z pohledu uživatele a test výkonový. Závěrečná kapitola obsahuje shrnutí celé práce a možné výhledy do budoucna. K práci je přiloženo CD, které obsahuje referenční implementaci navrženého systému a celý text v elektronické podobě.

Systém se při praktickém testu osvědčil – poskytuje dostatečný výkon a zároveň jej uživatelé shledali vyhovujícím. Dále je zmíněna možnost masového nasazení systému, neboť i kdyby nebylo možno využít přímo referenční implementaci, určenou pro webový server Apache, je text práce dostatečným výchozím podkladem pro implementace alternativní.

Klíčová slova

Autentizační mechanismus, hádání hesel, http, krádež sezení, ochrana „http basic“ autentizace, ochrana webových serverů, Turingův test, útoky na autentizační mechanismy, útoky typu odepření služby.

Zadání

Cílem práce je po odpovídající analýze problému navrhnout a alespoň v omezené míře v praxi otestovat systém pro ochranu webových serverů, který by detekoval útoky odepření služby (*denial-of service*) na protokol HTTP a eliminoval je (nebo alespoň snížil jejich dopad na únosnou míru). Dále by měl zajistit:

1. zvýšenou odolnost proti hádání hesel ať již metodou hrubé síly nebo metodou slovníkovou;
2. zpětnou kompatibilitu na úrovni rozhraní vůči webovým aplikacím (CGI, PHP);
3. odolnost proti pokusům o „ukradení“ relace.

Tento systém by měl být implementován jako patch/modul pro Apache httpd.

Místo této strany bude vložena kopie oficiálního zadání.

Obsah

Kapitola 1

Úvod

V této kapitole se pokusím v dostatečném rozsahu seznámit čtenáře s jednotlivými protokoly a mechanismy, jejichž znalost je pro pochopení dalšího textu klíčová. Musím však upozornit, že vzhledem k rozsahu práce byly některé části jednotlivých protokolů zjednodušeny, nebo byl jejich popis záměrně zcela vynechán.

Pokud jste dostatečně obeznámeni s principem fungování TCP/IP a HTTP protokolu, autentizačními mechanismy v HTTP a mechanismy pro správu sezení[1], můžete tuto kapitolu bez obav přeskočit.

1.1 Jemný úvod do TCP/IP

Pod zkratkou TCP/IP se skrývá celá rodina protokolů, které umožňují komunikaci v rámci heterogenních počítačových sítí, z nichž nejznámějším reprezentantem je Internet.

Tato rodina protokolů je dobrým příkladem dělby práce, neboť jednotlivé protokoly vytvářejí hierarchii a řeší různé podproblémy přenosu dat. Začněme tedy od nejnižších vrstev.

1.1.1 Internet Protocol (IP)

IP [1] je prvním z této rodiny protokolů. Jeho úkolem je doručovat jednotlivé „balíčky dat“ (packety[2]) uvnitř sítě od odesílatele k příjemci. Přesněji je zodpovědný za nalezení cesty od odesílatele k příjemci a za základní zajištění integrity packetu (tedy ochraně proti náhodné modifikaci při přenosu).

Komunikující partneři jsou v síti jednoznačně identifikováni IP adresou o délce 32 bitů[3], která se pro zjednodušení zapisuje jako čtyři čísla od 0 do 255, oddělená tečkami.

---

1. Rozhodl jsem se používat termín „sezení“ namísto „relace“, neboť mi přijde vhodnější.
2. V textu budu využívat anglické označení *packet* skloňované podle vzoru hrad, neboť mi jeho překlad přijde na úkor srozumitelnosti textu.
3. Zde uvažuji IP verze 4, neboť IPv6 má ještě dalekou cestu k masovému nasazení.

V adresním prostoru je vyhrazeno několik sítí pro privátní použití – tyto tzv. „privátní adresy“ [2] nejsou v Internetu použity.

Je vhodné upozornit, že IP neposkytuje garanci doručení jednotlivých packetů, ani ověření identity odesílatele.

1.1.2 Internet Control Message Protocol (ICMP)

ICMP [3] se nachází na stejné logické úrovni jako IP, ačkoli služeb IP pro svoji funkci využívá. Jeho hlavním úkolem je signalizace různých neobvyklých stavů, které se při přenosech dat vyskytnou – ať již jde o informaci o neexistenci cesty k cíli, informaci o zahození packetu nebo o žádost o zpomalení vysílání.

Tento protokol opět žádným způsobem nezajišťuje ani garanci doručení ani ověření identity odesílatele.

1.1.3 Transmission Control Protocol (TCP)

Protokol TCP [4] staví na pevných základech protokolu IP a nabízí užitečné rozšíření.

Tím je abstrakce datových přenosů pomocí zavedení představy spolehlivého obousměrného virtuálního okruhu (spojení). Aplikace využívající služeb TCP již tedy nemusí uvažovat jednotlivé packety a obstarávat kontrolu správnosti doručení dat.

Aby bylo možné sestavit více paralelních spojení mezi komunikujícími partnery, je kromě zdrojové a cílové adresy k jednoznačné identifikaci jednotlivých spojení využíván ještě zdrojový a cílový port, což je 16-ti bitové číslo.

Správné pořadí přenášených dat stejně jako kontrolu doručení implementuje TCP jednoduchým způsobem – obě strany si při vytváření spojení navzájem dohodnou dvě sekvenční čísla (pro každého z partnerů jedno). Při odeslání dat odesílatel do packetu v položce *Sequence Number* (dále jen „SN“) TCP hlavičky připojí sekvenční číslo prvního bajtu odesílaných dat. Při přijetí dat naopak příjemce informuje odesílatele pomocí položky *Acknowledgment Number* (dále jen „AN“), jaké je sekvenční číslo prvního bajtu za celistvým blokem dat, který do této chvíle úspěšně přijal.

Na základě vypočteného rozdílu mezi přijatým AN a SN posledního odeslaného bajtu je odesílatel schopen odvodit, zda je nutné opakovat odeslání nějakých dat díky ztrátě (nebo poškození) některého z packetů při přenosu.

Pro účely signalizace jsou v TCP hlavičce (mimo jiné) nastavovány následující příznaky:

- SYN – žádost o synchronizaci sekvenčního čísla na hodnotu uvedenou v SN, tento příznak je významný při sestavování spojení;
- ACK – AN obsahuje platná data (bez tohoto příznaku nebere příjemce AN v potaz);
- PSH – žádost o okamžité předání přijatých dat vyšší vrstvě (aplikaci);
- RST – informace o násilném ukončení spojení;
- FIN – informace o jednostranném uzavření spojení ze strany odesílatele (neboť již nemá další data k odeslání).

Na základě předchozího popisu je nyní možné uvést přesný postup vytváření spojení, který se díky třem vyměněným zprávám nazývá třícestné potvrzování (*three-way handshake*):

1. Iniciátor spojení (A) odešle partnerovi (B) TCP packet s nastaveným příznakem SYN a SN obsahujícím náhodné číslo;
2. B odešle A TCP packet s nastavenými příznaky SYN, ACK a dále SN obsahujícím náhodné číslo a AN obsahujícím přijaté SN + 1;
3. A odešle B TCP packet s nastaveným příznakem ACK a AN obsahujícím přijaté SN + 1.

Po této synchronizaci sekvenčních čísel mohou oba partneři začít výměnu dat. Spojení zaniká v momentě, kdy oba partneři uzavřou spojení pomocí FIN packetu, nebo pokud jeden z partnerů násilně spojení ukončí zasláním RST packetu.

1.1.4 Network Address Translation (NAT)

Díky zmenšujícímu se počtu nepřidělených IP adres byla v devadesátých letech minulého století publikována úprava IP protokolu [5], která umožňuje „zamaskovat“ síť o N počítačích za M veřejných adres. Principem je přepis zdrojové a/nebo cílové adresy v jednotlivých packetech procházejících přes router provádějící NAT (dále též „překlad adres“).

Tato úprava s sebou bohužel nese několik nevýhod – nejpodstatnější z nich je nemožnost navázat spojení s počítačem umístěným v síti za routerem provádějícím překlad adres. Pro velké množství běžně používaných

protokolů (DNS, SMTP, POP3, IMAP, HTTP, HTTPS, telnet, SSH, ICQ . . .) není tato nevýhoda naštěstí nijak omezující. Pro ostatní protokoly (např. pro FTP) byly buď implementovány další podpůrné mechanismy, a nebo je jejich použití v síti s překladem adres vyloučeno.

1.2 HyperText Transfer Protocol (HTTP)

HTTP [6] je bezstavový textový protokol vystavěný nad TCP/IP, určený primárně k přenosu hypertextových informací – tedy textových dat doplněných o odkazy na další dokumenty (a multimediální data). Protože jde o vcelku komplexní protokol, popíši nejdříve jeho základní variantu a tu průběžně v několika částech doplním o bližší informace o jednotlivých rozšířeních.

1.2.1 Základní funkce

Klient se připojí ke vzdálenému hostiteli komunikujícímu HTTP protokolem (který naslouchá zpravidla na TCP portu 80) a odešle požadavek, kterým žádá o určitý prostředek (zdroj) na daném serveru. První řádek požadavku obsahuje, mezerami oddělené, následující parametry:

- Metodu pro přístup k prostředku. Nejčastěji jde o metodu `GET`, kterou klient jednoduše žádá o zaslání prostředku. Ostatní metody se používají k odeslání formulářových dat/souborů (`POST`), odeslání libovolných dat (`PUT`), získání informací o prostředku (`HEAD`), diagnostice (`TRACE`) a dalším činnostem.

- Univerzální identifikátor prostředku (URI)[4]. Tento jednoznačně identifikuje prostředek (soubor nebo vzdálenou funkci), o který má klient zájem. Pokud jde o vzdálenou funkci, je též možné specifikovat rozšiřující parametry jako součást URI (u statického souboru to nemá smysl).

- Verzi protokolu, kterou klient použil pro sestavení požadavku. Server může odpovědět nižší verzí protokolu, má-li k tomu dobrý důvod (například klientem specifikovanou verzi nepodporuje).

Další řádky obsahují jednotlivé doplňující hlavičky, které upřesňují požadavek a zároveň informují server o schopnostech klienta. Každá hlavička

4. Universal Resource Identifier.

je umístěna na novém řádku a obsahuje dvě části: klíč (název) a hodnotu, navzájem oddělené dvojtečkou a mezerou.

Požadavek je ukončen odesláním prázdného řádku – po přijetí tohoto oddělovače začne server zpracovávat klientův požadavek.

Odpověď serveru je svou strukturou podobná požadavku – první řádek tentokrát obsahuje mezerami oddělené následující hodnoty:

- Verzi protokolu použitou k sestavení odpovědi.
- Třímístný číselný stavový kód rozlišující typ odpovědi. Jeho první číslice určuje povahu odpovědi. Kódy začínající číslicí `2` značí úspěch, `3` značí přesměrování, `4` chybu klienta, `5` chybu serveru. Mezi nejběžnější kódy patří:
  - `200` – úspěch,
  - `301` – trvale přesunuto (na adresu v hlavičce `Location`),
  - `302` – dočasně přesunuto,
  - `401` – požadována autentizace,
  - `404` – nenalezeno,
  - `500` – vnitřní chyba serveru.

  Přesný seznam a význam jednotlivých kódů je možné nalézt v RFC o HTTP [6].
- Popis stavového kódu, který může obsahovat další mezery.

Další řádky opět obsahují doplňující hlavičky a za nimi, oddělené prázdným řádkem, následuje tělo odpovědi.

Po odeslání těla odpovědi server uzavře spojení[5].

1.2.2 Příklad požadavku a odpovědi

Následující požadavek žádá o prostředek `/catalog/show.cgi` na serveru `example.com` (což je zřejmé z hlavičky `Host`). Jedná se o vzdálenou funkci, neboť jí jsou předány dva parametry: `id` a `cat`. Z dalších hlaviček je jasné, že si klient přeje obdržet buď HTML nebo prostý text (`Accept`) a že preferuje výsledný dokument v českém, případně anglickém, jazyce (`Accept-Language`).

5. HTTP verze 1.1 obsahuje rozšíření umožňující klientovi provést několik požadavků v rámci jednoho spojení.

```
GET /catalog/show.cgi?id=153&cat=pets HTTP/1.0
Host: example.com
User-Agent: Wejn's ultimate HTTP client (Linux)
Accept: text/html, text/plain
Accept-Language: cs, en
```

Server na takový požadavek může odpovědět například následovně:

```
HTTP/1.0 200 OK
Date: Wed, 02 Nov 2005 16:14:56 GMT
Server: Apache
Connection: close
Content-Type: text/plain; charset=iso-8859-2

Výpis položky 153 z katalogu domácích mazlíčků
-----------------------------------------------
Jméno: Obojek pro psa, průměr 20 cm
Materiál: kůže
Barva: hnědá
Cena: 150,- Kč
...
```

1.2.3 Autentizace

Autentizace v HTTP protokolu je řešena pomocí doplňujících hlaviček a signálem k zahájení autentizace je již zmíněný stavový kód `401` [7]. Existují dvě standardní schémata pro autentizaci:

- `Basic` – jednoduché schéma, kdy se uživ. jméno a heslo (dále jen „CREDS“) přenáší nešifrované.

- `Digest` – složitější schéma, v kterém se využívá metody výzva-odpověď (*challenge-response*). Místo CREDS klient odesílá jen výsledek jednocestné funkce aplikované na CREDS a unikátní hodnotu (`nonce`), díky čemuž je vyloučeno odchycení CREDS po cestě.

Obě schémata mají stejný princip fungování: Server při přijetí požadavku na autentizovaný prostředek ověří CREDS, získané z doplňující hlavičky `Authorization`, oproti databázi platných CREDS. Pokud jsou uživatelem odeslané CREDS platné, je přístup povolen (a požadavek je dále zpracováván jako kterýkoliv jiný). V opačném případě (a zejména pokud

klient vůbec hlavičku `Authorization` neodeslal) je klientovi vrácen chybový kód `401` spolu s doplňující hlavičkou `WWW-Authenticate`, která obsahuje podklady potřebné k autentizaci.

1.2.4 Správa sezení – cookies

Jak již bylo na začátku zmíněno, HTTP je bezstavový protokol – to znamená, že každý požadavek je naprosto nezávislý na ostatních. To s sebou nese jisté výhody (například triviální způsob obnovy po dočasném výpadku síťového spojení), ale též jednu zásadní nevýhodu: neexistuje mechanismus, jak spolehlivě uchovávat stav – ať již jen mezi jednotlivými požadavky nebo v mezidobí, kdy je klientský program neaktivní (vypnutý).

Z toho důvodu byl HTTP protokol rozšířen o jednoduchý mechanismus pro správu sezení – cookies[6] [8, 9].

Cookie je „malé množství textu“, které může být klientovi předáno serverem v rámci odpovědi na libovolný z jeho požadavků – a to v doplňkové hlavičce `Set-Cookie`. Pokud se klient rozhodne cookie akceptovat, odesílá jméno dané cookie a její hodnotu při dalších požadavcích v hlavičce `Cookie`.

Hlavička `Set-Cookie` může obsahovat kromě jména a hodnoty dané cookie ještě několik nepovinných parametrů ovlivňujících chování klienta:

- `Domain` – doménové jméno omezující platnost cookie;
- `Path` – prefix cesty, v rámci které je cookie platná;
- `Secure` – příznak, zda je zapovězeno odesílat cookie přes nezabezpečené (nešifrované) spojení;
- `Max-Age` – omezení doby, kdy je daná cookie platná.

Dnešní HTTP klienti navíc zpravidla kontrolují, zda je pole `Domain` shodné s aktuální doménou, aby server nemohl nastavit cookie pro jinou doménu.

1.2.5 HTTP skrz proxy server

Posledním zajímavým rozšířením HTTP protokolu, které zde budu popisovat, je zasílání požadavků přes tzv. „proxy server“ (dále jen „proxy“).

Při využití proxy se klient nepřipojuje přímo na cílový server, ale na prostředníka (*proxy*), se kterým komunikuje upravenou verzí HTTP. Proxy poté

6. V textu budu využívat anglické označení *cookie* a jeho množné číslo *cookies*, neboť mi překlad tohoto termínu přijde na úkor srozumitelnosti textu.

požádá cílový server normálním HTTP protokolem o klientem požadovaný prostředek a po obdržení odpovědi tuto vrátí klientovi.

Úprava HTTP pro komunikaci přes proxy server spočívá v modifikaci první řádky požadavku, kde klient namísto `URI` uvádí celé `URL`, tedy `URI` obohacené o jméno protokolu a doménové jméno cílového serveru.

Některé proxy do požadavku pro cílový server vkládají různé hlavičky s identifikací (IP adresou) klienta (namátkou `X-Forwarded-For`), který si daný prostředek vyžádal, ale není radno se na věrohodnost obsahu těchto hlaviček jakkoli spoléhat.

Kapitola 2

Útoky na HTTP

2.1 Útoky typu odepření služby – DoS

Útoky DoS[1] je možné nalézt téměř u každého protokolu, včetně HTTP(S) [10]. Jejich cílem je úplné znepřístupnění serveru poskytujícího služby, nebo alespoň citelné omezení jeho schopnosti odpovídat na požadavky klientů. Díky jejich relativní dostupnosti i technicky méně zručným uživatelům jsou velmi rozšířeny – není totiž třeba žádných složitých procedur či nákladného zařízení. Navíc existuje široká paleta nástrojů připravených pro tyto účely [11]. Typickým příkladem distribuované varianty takového útoku byl destruktivní kód internetového červa MyDoom [12], který měl za cíl znepřístupnit internetové stránky společnosti Santa Cruz Operation (SCO) v únoru 2004. Útočníkovým cílem je vždy vyčerpání nějakého omezeného prostředku, následkem čehož je server neschopen obsluhovat další klienty. Útoky DoS lze rozdělit dle dvou zásadních pohledů – viz 2.1.1 a 2.1.2.

2.1.1 Rozdělení dle zdrojů, který se snaží napadnout / vyčerpat

Útoky na tabulku procesů
Tyto útoky jsou prováděny vytvořením velkého množství paralelních spojení na napadený server – za účelem vyčerpání volných položek tabulky procesů, jehož důsledkem je nemožnost obsluhy legitimních požadavků ostatních klientů. Tento typ útoků je velmi účinný vůči webovému serveru Apache [13], který má navíc pro obsluhu požadavků vyhrazeno obvykle jen 256 „slotů“ (procesů).

Útoky na síťovou konektivitu
Velmi běžné útoky, které se snaží enormním množstvím požadavků a nebo jiným mechanismem „ucpat“ linku spojující napadený server se zbytkem Internetu. Jiným mechanismem se rozumí například zahlcení linky pomocí příchozích ICMP echo-reply packetů, které jsou zasílány na cílový server

1. Zkratka anglického *Denial-of-Service*.

ostatními systémy v Internetu (viz *SMURF attack* [14, 15, 16]) nebo *DNS-response flooding*, který využívá „amplifikace“ díky řádově větší odpovědi na krátký dotaz. Ovšem nezřídka se objevují i útoky normálním zahlcením linky díky požadavkům ze systému, který je připojen k Internetu rychlejší linkou než napadený server.

Útoky na síťovou vrstvu
Podobně jako předchozí typ útoků je i tento zaměřen na vyčerpání prostředků umožňujících komunikaci po síti. Na rozdíl od předchozího útoku je ale cílem síťová vrstva napadeného serveru – zpravidla tabulka spojení, kterou se útočník snaží zaplnit výlučně svou aktivitou. Mezi typické zástupce patří SYN-flood [17, 18], tedy zasílání velkého množství žádostí o nové spojení – SYN packetů, aniž by útočník dále dodržoval protokol a dokončoval sestavování spojení odesláním ACK packetů na přijatou SYN/ACK odpověď.

Útoky na procesorový čas, případně další systémové zdroje (paměť, I/O)
Mechanismus těchto útoků je velmi podobný útokům na síťovou konektivitu, avšak využívá jiného „úzkého místa“ – zátěže procesoru, paměti, případně I/O subsystému. Cílem útočníka je zahltit server množstvím na systémové prostředky náročných operací (například komplexní SQL dotazy, manipulace s grafickými daty, složité výpočty).

Útoky na chybu aplikace či designu
Tyto útoky jsou méně časté, i když stejně účinné. Jedná se o zneužití chyby, která způsobí buď násilné ukončení běhu serveru nebo nadále omezí jeho schopnost obsluhovat požadavky.

2.1.2 Rozdělení dle provedení útoku

Z jediného počítače
V tomto případě útočník k vyčerpání některého z výše uvedených zdrojů využije vlastní počítač. Nejčastěji je cílem vyčerpání tabulky procesů obrovským množstvím paralelních spojení, neboť implementace nástroje k takovému útoku ve vyšším programovacím jazyce zabere nanejvýš několik desítek řádek.

Navíc lze takový útok provést i pomocí utility Apache Bench, která je součástí open-source webového serveru Apache – stačí jen vhodně zvolit počet paralelních požadavků.

Detekce a obrana nevyžaduje žádné enormní úsilí. Obvykle stačí znemožnit komunikaci s útočícím počítačem na úrovni packetového filtru.

Z jediného počítače skrz proxy servery
Namísto přímého spojení s napadeným počítačem je využito spojení přes proxy server(y), které mohou zaručit nejen anonymitu, ale také maskování útoku – několik paralelních přístupu z více IP adres je méně nápadné než desítky přístupů z jediné IP adresy. Obrana v tomto případě již není triviální, neboť znemožnění komunikace pro jednotlivé proxy servery znamená zároveň znemožnění přístupu legitimním klientům využívajícím tuto odstíněnou proxy.

Z mnoha počítačů, případně i s využitím proxy serverů
Tento útok je též známý jako DDoS (*distributed denial of service*) [11], tedy distribuovaný útok odepření služby. Útočník zde využívá k útoku množinu počítačů (desítky až stovky tisíc) pod jeho kontrolou k znepřístupnění cílového serveru. Toto je oblíbený mechanismus působení internetových červů (MyDoom, MSBlast) [12, 19] a DDoS nástrojů (Trinoo, Stacheldracht) [20, 21].

2.2 Útoky na autentizační mechanismy

Jejich cílem je získání neoprávněného přístupu k autentizovaným prostředkům, službám či datům napadeného serveru. Podle způsobu provedení je lze rozdělit na několik variant – od neoprávněného přístupu podvodným získáním autentizátoru již připojeného klienta (nebo klientů) až po úplné prolomení autentizačního mechanismu. Níže zmíněné varianty jsou specializací obecných principů na oblast prolomení autentizačních mechanismů a tudíž nepopisují případy, kdy dojde k zcizení přihlašovacích údajů například pomocí programu zaznamenávajícího stisknuté klávesy (keyboard sniffer). Následující seznam si neklade za cíl být úplným a podrobným, ale spíše nastínit problémová místa, která může potenciální útočník zneužít.

2.2.1 Útok na klienta (resp. na konkrétní aktivní sezení)

Tato třída útoků je zaměřena na získání identity konkrétního uživatele nebo uživatelů. Možností provedení tohoto útoku je mnoho:

Krádež identifikátoru sezení (s použitím zákeřného kódu)
Elegantní způsob získání neoprávněného přístupu je pomocí krádeže identifikátoru sezení aktuálně autentizovaného uživatele. Tato metoda je navíc díky nedůslednosti programátorů webových aplikací v poslední době velmi populární, neboť nemalé množství těchto programátorů odmítá důsledně dodržovat zásady bezpečného programování webových aplikací. K prove-

dení takového útoku je nutné, aby napadená aplikace umožňovala přímé vložení útočníkem definovaného HTML kódu do stránky, která je zobrazena uživateli (oběti). Takový kód má potom za úkol donutit prohlížeč oběti vykonat nějakou činnost, která povede k vyzrazení identifikátoru sezení útočníkovi.

Konkrétní útoky se samozřejmě liší dle použitého mechanismu správy sezení, nicméně nejznámější zástupci jsou:

A. **Krádež obsahu „Referer" hlavičky**
Tuto metodu je možné aplikovat oproti mechanismům správy sezení, které využívají k sledování sezení identifikátor umístěný v URL – buď v cestě nebo jako parametr předávaný mezi stránkami. Tato metoda je sice v RFC o HTTP [6] nepřímo označena jako nebezpečná, nicméně se s ní stále můžeme setkat v široké škále aplikací, včetně velmi populárního jazyka PHP [22]. V něm je volitelně (a transparentně) využita k sledování sezení v případě, že prohlížeč nepodporuje/blokuje cookies.

Princip je jednoduchý – do stránky v autentizované oblasti je vložen odkaz na externí objekt (zpravidla obrázek, flash nebo jen klasický odkaz na stránku) ležící na webovém serveru, který je pod kontrolou útočníka. Jakmile se pokusí prohlížeč oběti získat tento objekt, vyzradí identifikátor sezení v obsahu hlavičky `Referer`. Ta totiž obsahuje plnou URL, z které je na vyžádaný objekt odkazováno.

B. **Krádež cookie nebo uživ. jména a hesla – XSS/XST**
Krádež identifikátoru sezení, který je předáván jako cookie, je poněkud komplikovanější, neboť cookie nelze získat pouhým vložením libovolného objektu. Ke krádeži cookie je nutné využít skriptovací jazyk prohlížeče (obvykle JavaScript). Skript v tomto jazyce je zodpovědný za vygenerování a vložení příslušného HTML odkazu do stránky. Takový odkaz potom jako součást své URL obsahuje přímo identifikátor sezení a útočník jej opět může jednoduše zaznamenat na svém serveru. Příkladem takového skriptu může být:

```
<script language="JavaScript">
document.write("<img src='http://evil.server/steal.cgi?id=" +
    escape(document.cookie) + "' width=1 height=1>");
</script>
```

Tato metoda krádeže se nazývá *cross-site scripting* (XSS) [23].

Podobnou metodou jako XSS je *cross-site tracing* (XST) [24], která umožňuje získat nejen cookies, ale i uživatelské jméno a heslo použité při

„HTTP basic“ autentizaci. Tento mechanismus je opět o řád složitější než XSS: v případě XST je prohlížeč skriptem donucen provést „normální “ požadavek do autentizované oblasti pomocí metody TRACE, která namísto obvyklé GET/POST vrací zpět jako odpověď vše, co bylo odesláno v rámci požadavku. Důsledkem toho získá skript v odpovědi veškeré hlavičky odeslané na server, speciálně tedy autentizační údaje a cookies. Tyto je potom možné opět odeslat na vzdálený server, stejně jako v případě XSS.

XST samozřejmě obchází i tzv. „HttpOnly cookies“ [25] rozšíření, které pro speciálně označené cookies znemožňuje jejich získání pomocí skriptovacího jazyka a tedy zabraňuje zneužití pomocí XSS. Toto rozšíření je každopádně k dispozici jen v prohlížeči Internet Explorer verze 6 SP1 (a vyšší).

Nepříjemnou vlastností uvedených krádeží je, že změna hesla daného účtu nemusí mít žádný vliv na identifikátor sezení a tudíž je možné, že útočník bude mít přístup i po změně hesla. To samozřejmě záleží na konkrétní implementaci mechanismu správy sezení.

Krádež autentizátoru „fixací “ sezení
Session fixation attack [26] je zajímavou variací krádeže cookie. V praxi tento útok probíhá tak, že útočník donutí prohlížeč oběti nastavit identifikátor sezení (*session cookie*) na útočníkem definovanou hodnotu a vyčká než se oběť přihlásí do systému. Donucovacím prostředkem může v tomto případě být cookie s vhodnou hodnotou pole `domain` – známým případem takového problému jsou domény .co.uk (a podobné), neboť prohlížeče obecně ignorují příliš obecné pole `domain` pouze v případě, kdy je hodnotou doména prvního řádu. Podobný problém se dá očekávat i u všech hostingových služeb poskytujících domény třetího řádu, stejně jako u služeb pro dynamické DNS [27]. Řešení tohoto problému tkví ve změně identifikátoru sezení okamžitě po autentizaci uživatele.

Krádež autentizátoru nepřímým způsobem
Nepřímým způsobem zde rozumíme například krádež cookie z (po lokální síti sdíleného) disku oběti, pasivním odposlechem packetů na cestě mezi obětí a serverem, sledováním záznamů na proxy serveru (jehož služeb oběť využívá) atp.

Krádež uživatelských údajů nepřímým způsobem
Tato třída útoků na klienta je uvedena spíše pro zamyšlení, protože se tématu práce dotýká jen okrajově. Mezi nepřímé metody krádeže uživ. jména a hesla patří například:

A. **DNS spoofing**
Metoda je založena na podvržení falešné adresy serveru, se kterým se klientský prohlížeč pokouší navázat spojení [28].

B. **Phishing útoky**
Útok funguje na principu „šíření poplašné zprávy", kdy je klient například vhodně formulovaným e-mailem vystrašen, že mu hrozí finanční ztráta (nebo jiné nebezpečí), pokud okamžitě neprovede nějakou akci. Nejčastěji je zmíněnou akcí vyplnění soukromých údajů na stránce odkazované z daného e-mailu. Vyplnění údajů je však, ke smůle klienta, provedeno na serveru, který se pouze zdá býti oficiálním webem dané instituce. Ve skutečnosti je navštívený server plně pod kontrolou útočníka – stejně jako osobní data vyplněná důvěřivým klientem [29].

C. **ARP spoofing**
Metoda využívá podvržení ARP[2] adresy brány sítě útočníkem, což mu následně umožňuje odposlouchávat komunikaci mezi obětí a zbytkem sítě [30].

D. **Využití spyware**
Spyware je označení pro různé druhy softwaru, které mj. neautorizovaně sbírají osobní informace na uživatelském počítači [31].

2.2.2 Útok na server či aplikaci

Tato třída útoků je zaměřena na získání přístupu do systému bez pevné vazby na konkrétního uživatele. Většinu z těchto útoků je též možné považovat za nebezpečnější než útoky na klienta, neboť dochází k prolomení samotného autentizačního mechanismu, nejen konkrétního uživatelského účtu.

Slovníkový útok nebo útok hrubou silou na přihlašovací údaje
Slovníkovým útokem je myšleno postupné zkoušení přihlašovacích údajů výběrem z předem připravené kolekce. Výčet způsobů pro získání dostatečně kvalitního slovníku je nad rámec této práce, nicméně zahrnuje mimo jiné: extrakce hesel z kompromitovaných serverů podobného zaměření (podobné uživatelské základny), triviální kombinace obecně známých slabých hesel (barvy, jména, letopočty, data narození. . .) spolu se známou množinou uživatelských jmen atp. Útok hrubou silou potom odpovídá přesně

2. *Address Resolution Protocol* (ARP) je protokol překladu IP adres na fyzické adresy (MAC) uvnitř Ethernet sítě.

definici, tedy vyzkoušení všech možných kombinací (nebo alespoň velkého množství z nich).

Oba útoky jsou velmi populární u serverů využívajících jako autentizační mechanismus HTTP autentizaci (dle definice v RFC o HTTP autentizaci [7]), zejména pak variantu „basic“. Díky bezstavovosti HTTP (autentizační údaje se odesílají s každým požadavkem) totiž není možné jednoduše implementovat ochranné prvky – jako například maximální počet nesprávných zadání přihlašovacích údajů za jednotku času (mimo jiné i díky existenci proxy serverů). Faktem je, že nejrozšířenější implementace HTTP serverů [32] – Apache a Microsoft IIS v této oblasti neposkytují žádnou ochranu.

Prolomení logiky autentizačního mechanismu
Jedná se o zneužití chyby v samotném mechanismu (algoritmu), který je zodpovědný za ověřování přihlašovacích údajů – například vložením nepředpokládaného vstupu – jehož následkem je útočník nesprávně autentizován jako některý z uživatelů, zpravidla administrátor. Zevrubný popis několika případů takových útoků lze nalézt v článku „Dos and Don'ts of Client Authentication on the Web“ [33].

Útok na (nedostatečný) mechanismus uchovávání identifikátorů sezení (zpětná analýza autentizačního tokenu)
Tento typ útoků je zaměřen na autentizační mechanismy využívající k autentizaci uživatelů autentizační tokeny v podobě cookies nebo parametrů v URL. Jeho základem je podvržení uměle vytvořeného autentizačního tokenu, který je chybou návrhu autentizačního schématu nebo algoritmu zodpovědného za ověřování tokenů nadále považován systémem za platný a útočník je tudíž autentizován jako uživatel. Detailní informace je opět možné nalézt v článku „Dos and Don'ts of Client Authentication on the Web“ [33].

Útok hrubou silou na identifikátor sezení
Varianta předchozích útoků – ovšem hrubá síla je zde uplatněna oproti autentizačnímu tokenu jako celku nebo MAC[3] [34, kap. 9] chránícím jeho integritu.

Útok pomocí zvýšení práv (privilege escalation)
Tento útok je založen na zneužití práv nebo znalostí přímo nesouvisejících se samotným autentizátorem, díky kterým je útočník schopen buď odhalit nějakou chybu v logice aplikace (a následně prolomit logiku autentizačního mechanismu) nebo je schopen zajistit si autentizaci jiným způsobem (na-

3. *Message Authentication Code* – využití jednosměrné funkce pro zajištění integrity přenášené zprávy.

příklad přidáním uživatele nebo identifikátoru sezení do databáze, ke které získal neoprávněný přístup).

Kapitola 3

Ochrana proti útokům a existující obranná řešení

Popis ochranných mechanismů jsem se rozhodl z důvodu lepší čitelnosti opět rozdělit na dvě části a to na ochranu proti DoS a ochranu proti útokům na autentizační mechanismy.

3.1 Ochrana proti DoS

Ochrana proti DoS útokům patří mezi netriviální problémy i v dnešní době. Slabých míst, která může případný útočník využít, je totiž, jak již bylo naznačeno v předchozí kapitole, mnoho.

Ochrana na úrovni síťové vrstvy
Na síťové vrstvě je možné využít jednak přímo ochranných prvků jednotlivých operačních systémů (OS) a jednak specializovaných přídavných zařízení nebo modulů.

V oblasti ochranných prvků v rámci OS je typickou volbou použití vestavěného packetového filtru (umožňujícího mimo jiné omezit maximální počet spojení z konkrétní IP adresy nebo podsítě) a například povolení tzv. „SYN-cookies“ [35]. Tato technika upravuje reakci OS na žádost o vytvoření spojení – příjem SYN packetu. Namísto vyhrazení místa v tabulce spojení je jako odpověď odeslán SYN/ACK packet, který má iniciální sekvenční číslo (*Initial Sequence Number*, dále jen „ISN“) nastavené na výsledek jednocestné (hash) funkce aplikované na zdrojovou a cílovou adresu, zdrojový a cílový port a časově omezenou tajnou informaci. Při přijetí ACK packetu je potom ověřena shoda vráceného ISN pomocí stejného výpočtu. V případě shody je spojení přijato, v opačném případě odmítnuto. Tato metoda tedy efektivně znemožňuje provedení SYN-flood útoku, neboť nelze vyčerpat tabulku spojení pomocí pouhého zahlcení cílového systému proudem SYN packetů.

Komerčně nabízených ochranných prvků existuje nepřeberné množství, jedním z příkladů může být DefensePro [36], což je aktivní síťový prvek (switch), který zároveň funguje jako aplikační firewall chránící připojené

systémy proti DoS/DDoS útokům, virovým útokům, útokům na aplikace (protokoly), pokusům o průnik atd.

Ochrana na úrovni HTTP protokolu
V oblasti ochrany proti DoS útokům na úrovni protokolu HTTP lze obecně nalézt softwarové produkty, které se snaží buď omezovat využití prostředků serveru podle přednastavených limitů a nebo se snaží útok rozpoznat dle průvodních příznaků a následně jej neutralizovat.

Typickým zástupcem omezování je modul mod_dosevasive [37] pro Apache, který umožňuje nastavit limit na počet požadavků se stejným URL za jednotku času a maximální počet požadavků z konkrétní IP adresy za jednotku času na konkrétního potomka (obslužný proces). Pokud některá IP adresa překročí daný limit, je na administrátorem definovanou dobu umístěna na černou listinu (tedy na požadavky je okamžitě odpovězeno chybovým kódem bez jakékoli snahy o jejich uspokojení). V případě, že přicházejí další požadavky během doby, kdy je IP adresa na černé listině, je „trestný čas“ počítán znovu od začátku. Modul je též schopen upozornit administrátora emailem na tuto situaci nebo spustit definovaný příkaz (program) – například pro umístění IP adresy na černou listinu na firewallu.

Oproti tomu Zeus Webserver [38], jenž je velmi podobný Apache, poskytuje nejen možnost nastavení limitů na jednotlivé IP adresy nebo skupiny IP adres, ale zároveň obsahuje i mechanismy, které se snaží rozpoznat probíhající útok. Tyto informace jsou navíc sdíleny všemi servery uvnitř clusteru, takže je ochrana více provázaných serverů jednodušší.

3.2 Ochrana proti útokům na autentizační mechanismy

V této oblasti existuje pestrá škála různých řešení, neboť možných útoků a zranitelných míst zde existuje podstatně více. Opět se pokusím přiblížit jednotlivé obranné mechanismy spolu s typickým reprezentantem.

Filtrování požadavků
Tato metoda patří mezi velmi obecné metody ochrany proti útokům na aplikace – a to nejen na jejich autentizační subsystémy. Principem je kontrola příchozích požadavků na známé vzory nebo potenciálně nebezpečné konstrukce před předáním požadavku cílové aplikaci. Díky této vstupní kontrole lze odchytit problémové požadavky ještě před tím, než by jejich zpracování mohlo způsobit chybu v cílové aplikaci. Vhodně definovanými filtračními pravidly lze například velmi významně omezit riziko chyb vyvolaných vložením SQL kódu (*SQL injection*) [39] nebo nepřímo odstínit jed-

noduché formy zneužití chyby přetečení zásobníku (*buffer overflow*), pokud ve filtru zakážeme sekvence, které obvykle odpovídají zákeřnému kódu.

Příkladem jednoduchého *SQL injection* filtru je regulární výraz [40]:

```
/(delete\s+from|insert\s+into|select\s.*\sfrom)/i
```

aplikovaný na libovolná vstupní data v rámci požadavku. A podobně jednoduchým je příklad triviálního *buffer overflow* filtru, opět v podobě regulárního výrazu:

```
/\x90{6,}/
```

Tento filtr využívá faktu, že nemalé množství (zejména primitivních) exploitů[1] na přetečení zásobníku chyby obsahuje „doplnění“ (*padding*) v podobě `NOP` instrukce[2], jejíž ASCII kód je 144, tedy 90 v hexadecimálním zápisu.

Ochranu filtrováním požadavků můžeme nalézt například v modulu mod_security [42], který je určen pro Apache a nebo v již výše zmíněném Zeus Webserveru, který též disponuje nástrojem pro filtraci vstupních dat.

Zabezpečení „HTTP basic“ autentizace analýzou požadavků
Díky bezstavovosti HTTP je „HTTP basic“ autentizace snadným cílem pro útoky – každý požadavek totiž musí znovu obsahovat přihlašovací údaje a útočník tedy jedním každým požadavkem může otestovat platnost zadaných údajů.

Jednou z metod jak zabezpečit „HTTP basic“ autentizaci je analýza příchozích požadavků, resp. omezování maximálního počtu neúspěšných přihlášení z konkrétní IP adresy.

Komerční systémy Killer Security [43] a PennyWize [44] implementují přesně tento kontrolní mechanismus plus další zajímavé doplňky pro zmírnění následku průniku na nějaký uživatelský účet. Průnik je určen překročením limitu počtu zemí (nebo podsítí typu „C“), ze kterých se daný uživatel úspěšně přihlásil za definovanou jednotku času.

V obou produktech je analýza prováděna pomocí relativně jednoduchého Perl skriptu, který je napojen pomocí nepojmenované roury (`pipe`) na standardní logovací mechanismus Apache (mod_log_config), který mu touto rourou předává data o aktuálních přístupech (vzdálenou IP adresu,

1. *Exploit* označuje techniku nebo konkrétní program určený ke zneužití chyby v nějakém jiném programu.
2. Bližší informace o technikách pro zneužívání chyb přetečení zásobníku je možné nalézt v článku „Smashing The Stack For Fun And Profit“ [41].

uživatelské jméno, status požadavku, počet odeslaných bajtů, jméno serveru). Skript poté v pravidelných intervalech kontaktuje server poskytovatele, kterému předá nasbíraná data a zároveň dostane jako odpověď seznam akcí, které má provést (zablokování účtů / IP adres).

Rozdíly mezi těmito systémy jsou nepodstatné, ačkoli autoři systému Killer Security nabízejí (oproti úplatě) jeho implementaci i pro jiné platformy než je Apache.

Při své práci jsem mj. realizoval řešení, které je podobné výše popisovaným komerčním systémům. Jeho popis můžete nalézt na konci této kapitoly – v části 3.2.1.

Použití vlastní správy sezení
Díky nesnadnosti zabezpečení „HTTP basic" autentizace volí často programátoři webových aplikací implementaci vlastního systému správy sezení včetně vlastního autentizačního mechanismu, což je poměrně logická volba. Takové systémy zpravidla obsahují propracované techniky ochrany proti hádání hesel či dalším útokům.

Obskurní metody ochrany proti slovníkovým útokům
Dalšími možnostmi pro ochranu autentizovaných částí webů chráněných pomocí „HTTP basic" autentizace jsou různé CGI skripty, které rozličnými způsoby znemožňují (nebo znesnadňují) použití existujícího softwaru pro hádání hesel. Některé z nich obsahují zajímavé či roztomilé myšlenky (například dvojí přihlašování před přístupem – první pomocí uživ. jména spolu s PIN a druhé již „HTTP basic" autentizací, skrývání adresy autentizované oblasti pomocí matoucího JavaScript kódu atp).

3.2.1 Vlastní řešení I

Moje prvotní řešení bylo principem podobné systémům Killer Security a PennyWize. Na rozdíl od nich však analýza probíhá přímo uvnitř procesu, který je napojen na Apache a tudíž není nutná komunikace se vzdáleným serverem. Jednotlivá blokování jsou prováděna pomocí vytváření prázdných souborů v předdefinovaném adresáři.

Toto první řešení se skládá ze dvou částí. První část je samotný analyzátor logů, který je pomocí následující direktivy:

```
CustomLog "|/data/checker/checkit.rb" "%v %a %>s %u %r"
```

v httpd.conf napojen pomocí nepojmenované roury svým standardním vstupem na logovací mechanismus Apache. Díky speciálnímu formátu logované řádky tak má přístup k seznamu všech požadavků pro daný virtu-

ální server obsahující všechny potřebné informace: jméno virtuálního serveru, vzdálenou IP adresu, stavový kód požadavku, uživatelské jméno, první řádku požadavku. Na základě stavového kódu jsou potom rozpoznány neúspěšné pokusy o přihlášení (kód 401). Pokud jejich počet přesáhne pro konkrétní IP adresu určitý limit za danou časovou jednotku, je vytvořen soubor pojmenovaný dle útočníkovy IP adresy:

```
/data/checker/rules/_ip_$IP
```

Druhou částí je úprava autentizačního modulu mod_auth, který je zodpovědný za „HTTP basic“ autentizaci v Apache. Upravený modul při každém požadavku ověřuje existenci souboru:

```
/data/checker/rules/_ip_$IP
```

a pokud takový soubor existuje, je přístup ihned odmítnut s kódem 401, který klientovi dává na vědomí, že se autentizace nezdařila. Dále je ještě testována existence souboru:

```
/data/checker/rules/_user_$USER
```

která dává možnost jednoduchým způsobem zamítnout přístup danému uživateli bez nutnosti jeho odstranění z `htpasswd` souboru.

Kapitola 4

Nový koncept

V předchozí kapitole byla nastíněna již existující řešení bránící proti útokům typu DoS a útokům na autentizační mechanismy (resp. mechanismy správy sezení). V této kapitole bude popsán nový koncept, který má za cíl komplexně řešit obě oblasti problémů.

V případě ochrany autentizačního mechanismu bude též uvedena diskuse předchozí (pracovní) verze s pořadovým číslem II, neboť uvažované koncepty (ač jsou nedostatečné) dobře ilustrují různá úskalí autentizačních mechanismů, která nebývají na první pohled patrná (případně nejsou obecně brána v potaz).

4.1 Ochrana proti DoS

Jak již bylo nastíněno v předchozích kapitolách, je nutné implementovat ochranu proti několika různým způsobům DoS:

- vyčerpání tabulky spojení;
- zahlcení velkým množstvím paralelních spojení;
- vyčerpání systémových prostředků.

Ochrana operačního systému proti *vyčerpání tabulky spojení* je řešena povolením „SYN-cookies“: v případě Linuxu je nutné při překladu jádra zapnout volbu `CONFIG_SYN_COOKIES` a následně tuto ochranu aktivovat např. pomocí:

```
sysctl -w net.ipv4.tcp_syncookies=1
```

neboť tato funkce není automaticky po startu jádra aktivní.

Ochrana proti *zahlcení velkým množstvím paralelních spojení* je realizována stavovým firewallem iptables [45], zejména pak jeho modulem `connlimit`, který umožňuje omezit maximální počet paralelních spojení z dané IP adresy nebo podsítě. Vhodným výchozím nastavením je například:

```
iptables -A INPUT -p tcp --syn --dport 80 -m connlimit \
        --connlimit-above 30 -j REJECT --reject-with tcp-reset
```

čímž omezíme maximální počet paralelních spojení na port 80 (HTTP) z individuální IP adresy na 30. Při překročení limitu budou další spojení odmítána – při pokusu o spojení bude toto serverem uzavřeno (klientovi bude odeslán RST packet). Alternativou může být ještě použití:

```
iptables -A INPUT -p tcp --syn --dport 80 -m connlimit \
        --connlimit-above 30 -j DROP
```

kdy je namísto okamžitého resetu packet zahozen. V takovém případě dochází po uplynutí určité doby k opakování požadavku, což pro koncové uživatele může mít pozitivní efekt v případě dočasného překročení limitu – prohlížeč nevrátí okamžitě chybu, ale čeká.

Ochrana proti *vyčerpání systémových prostředků* je poskytována v základní míře modulem mod_dosevasive, který (jak již jsem zmínil v části 3.1 na straně 18) omezuje počet možných požadavků jednotlivých klientů za danou časovou jednotku.

Možnou budoucí optimalizací je využití jiného mechanismu počítání těchto požadavků, například přímo uvnitř mateřského procesu Apache nebo modulem stavového firewallu, které by přineslo přesnější měření počtu požadavků a rychlejší reakci na případný útok.

Ačkoli je toto konkrétní aplikační řešení vázáno na Apache, nic nebrání implementaci jeho obecného principu v libovolném webovém serveru. V případě *select-based*[1] serverů jako je lighttpd [46] bude implementace dokonce o řád jednodušší, neboť odpadá nutnost případné meziprocesové komunikace.

4.2 Ochrana autentizačního mechanismu

Mým záměrem je navrhnout a implementovat autentizační mechanismus a správu sezení, která bude odolná proti co největšímu počtu útoků popsaných v části 2.2 na straně 11.

4.2.1 Pracovní verze (vlastní řešení II)

Hlavní myšlenkou této verze návrhu bylo striktně oddělit autentizační mechanismus od mechanismu správy sezení, neboť jejich provázání je hlavním

1. Model webového serveru, který obsluhuje všechny požadavky v rámci jediného procesu.

zdrojem problémů při obraně proti útokům na uživatelské údaje (například v případě „HTTP basic“ autentizace) – protože dochází k ověřování uživ. údajů při každém požadavku, nelze jednoduše odlišit pokusy o hádání hesel od legitimních požadavků. Teoreticky je sice možné blokovat IP adresy se zvýšeným počtem požadavků s neplatnými údaji, ale v praxi se díky existenci proxy serverů běžně dostaneme do situace, že zablokováním takového proxy serveru znemožňujeme zároveň přístup oprávněným uživatelům.

Ochrana autentizačního mechanismu
V této verzi byl tedy autentizační mechanismus chráněn pomocí tzv. veřejných Turingových testů (*public Turing tests*) [47], jejichž cílem je odlišit požadavky prováděné strojem od požadavků prováděných uživatelem (respektive zajistit, aby každý požadavek o autentizaci vyžadoval vyřešení úlohy, která je triviální pro člověka a velmi obtížně řešitelná algoritmicky. Jednoduchým příkladem takové úlohy je například nutnost přepisu zkresleného textu z obrázku do vstupního pole ve formuláři:

Na první pohled je zřejmé, že obrázek obsahuje anglické slovo „now“, ačkoli jsou písmena značně rozrušena umístěním na barevný podklad a přeškrtáním pomocí bílých čar. Existující software ovšem v současné době nezvládá rozumně řešit ani takto jednoduché úlohy rozpoznávání „poškozených“ znaků a tak je zmíněná úloha obtížně řešitelná bez interakce člověka.

Jelikož tato metoda přináší netriviální výpočetní zátěž (vytváření testu na serveru) a jisté uživatelské nepohodlí (zapamatování hesla v prohlížeči nefunguje správně), byla naivní implementace optimalizována pomocí protokolu z článku „Securing Passwords Against Dictionary Attacks“ [48], jak je popsáno dále.

Optimalizace naivní implementace Turingova testu
Autory navržený dvoufázový systém po úspěšném přihlášení zašle uživateli autentizovanou cookie (tedy takovou cookie, kterou nelze zfalšovat), jež ho nadále zprošťuje povinnosti absolvování Turingova testu při příštím přihlášení. Systém si tedy v první fázi od uživatele vyžádá jeho uživatelské jméno a heslo a v druhé fázi si může vyžádat ještě vyřešení Turingova testu, pokud uživatel nezašle zároveň platnou autentizovanou cookie zmiňovanou výše. Tím je tedy zvýšen uživatelský komfort.

Snížení režie je implementováno v rámci reakce na neplatné uživatelské jméno nebo heslo v první fázi. V takovém případě systém deterministicky rozhodne (na základě zadaného už. jména a hesla) zda s pravděpodobností $1 - p$ ($0 \leq p \leq 1$, například $p = 0,05$) přihlášení okamžitě zamítne, a nebo jej (s pravděpodobností p) nechá před zamítnutím vyřešit Turingův test. Volbou parametru p je potom určeno jakou režii výsledný systém bude mít v případě pokusu o slovníkový útok. Je zřejmé, že při volbě $p = 1$ jde o naivní implementaci (viz výše) s vyšším uživatelským komfortem.

Mechanismus správy sezení

Mechanismus správy sezení (dále jen „MSS“) opět vycházel z již existujícího řešení – udržování sezení pomocí cookie.

S tímto mechanismem, realizovaným pomocí cookie nebo statického identifikátoru v URL, se můžeme setkat v naprosté většině aplikací. Podstatnou nevýhodou statického identifikátoru je zranitelnost v případě nevhodně naprogramované aplikace, která tento MSS využívá (viz krádež s použitím zákeřného kódu v části 2.2.1 na straně 11).

Tuto nevýhodu sice částečně odstraňují doplňkové mechanismy („uzamykání“ sezení na IP adresu, zneplatnění po krátké době nečinnosti), nicméně tyto dle mého názoru vytvářejí více problémů než řeší. Pokud pominu snížení komfortu vzhledem k nutnosti se znovu přihlašovat v případě delší nečinnosti (čtení delšího textu, psaní odpovědi na e-mail), je o mnoho závažnějším problémem nemožnost využití těchto mechanismů (nebo celé aplikace) pokud se uživatel nachází za M:N překladem adres (M:N NAT) nebo za tzv. „load-balancing proxy“[2]. V takovém případě je ochrana uzamčením adresy buď neaplikovatelná nebo na obtíž (uživatel může být kterýmkoli požadavkem násilně odhlášen).

Mnou navrženým řešením bylo zavedení jistého dynamického elementu do MSS. Hlavní myšlenkou bylo, že pokud budou identifikátory sezení periodicky obměňovány (časově omezeny), poroste obtížnost úspěšného útoku na takový systém pomocí krádeže identifikátoru. Ačkoli tím nelze útok úplně vyloučit, bude jeho praktické provedení téměř nemožné. Zamykání na IP adresu bylo plánováno jako volitelný doplněk pro uživatele, kterým to jejich způsob přístupu k Internetu neznemožňuje.

Systém využíval dva časově omezené identifikátory sezení předávané jako „HttpOnly cookies“[3]:

1. krátkodobý (IK), určený k autorizaci přístupu do chráněné oblasti;

2. Proxy, která může každý požadavek odeslat z jiné zdrojové adresy.
3. Krátký popis viz část B na straně 13.

2. jednorázový (IJ), určený k periodické výměně krátkodobých identifikátorů.

Pro periodickou výměnu IK bylo nutné v prohlížeči ponechat „přihlašovací okno“ otevřené a samotný přístup k autentizované části provádět v okně jiném – nově otevřeném.

Aby bylo zamezeno krádežím IJ, bylo přihlašování do autentizované oblasti (a následná výměna IK) prováděno v rámci poddomény (*subdomain*) původní domény, což znemožňovalo přístup k IJ přes skriptovací jazyk prohlížeče z dané aplikace, neboť objekt `document.cookie` (a další) není nijak přístupný, pokud se domény v oknech neshodují (zejména je tedy vyloučen přístup přes objekt `window.opener`). S tím také souvisela nutnost vhodného nastavení pole domain pro IK tak, aby byla tato cookie dostupná z domény aplikace.

V přihlašovacím okně byl tedy prohlížeč instruován, aby periodicky obnovoval aktuální stránku, čímž byla zajištěna pravidelná výměna IK. Zároveň s každým obnovením docházelo k výměně IJ, takže případná krádež a použití IJ útočníkem bylo okamžitě patrné – uživateli nebyl obnoven přístup k autentizované oblasti a přihlašovací okno hlásilo chybu.

Další podrobnosti
Návrh se věnoval ještě do detailu popisu:

1. zpětné kompatibility s již existujícími mechanismy („HTTP basic“ autentizace) uvnitř datových struktur Apache („podvržení “ informace o uživatelském jménu náležícímu k danému požadavku);

2. propagace informací o sezení k dalším modulům pomocí proměnných prostředí;

3. dalších podpůrných mechanismů pro zvýšení bezpečnosti.

Tyto části pracovní verze II jsem se rozhodl zde nepopisovat.

4.2.2 Zranitelnost pracovní verze II

Zranitelnost pracovní verze plyne z předpokladu, že se zavedením dynamického identifikátoru ztíží (nebo v lepším případě znemožní) útok pomocí XSS/XST. Tento se díky skriptovacím jazykům v moderních prohlížečích ukázal jako neplatný.

Pracovní verze je sice odolná proti jednorázové krádeži IK (například vložením kódu, který je popisován v části 2.2.1 na straně 11), nicméně nechrání proti opakovanému odesílání IK útočníkovi během celého sezení.

Takový útok je možné realizovat s využitím pokročilých možností jazyka JavaScript – odesílání HTTP požadavků s využitím XMLHttpRequest [49] a modifikace dokumentu za běhu funkcemi pro manipulaci s DOM [50].

Alarmující demonstrací této techniky je například software XSS-Proxy [51], který umožňuje (mimo jiné) kompletní vzdálenou kontrolu nad prohlížečem uživatele po provedení XSS útoku.

Díky takovým širokým možnostem nelze považovat za bezpečnou žádnou správu sezení založenou na libovolných stavových informacích, které jsou dostupné skriptovacímu jazyku prohlížeče. Je dobré též upozornit na skutečnost, že XSS-Proxy navíc neutralizuje i pozitivní efekt uzamčení sezení na IP adresu, případně šifrování přenášených dat pomocí SSL.

V další části se tedy budu věnovat novému návrhu, který tyto bezpečnostní problémy odstraňuje.

4.2.3 Možné cesty k novému návrhu

Jak jsem již v předchozí části zmínil, klientský skriptovací jazyk má přístup téměř ke všem údajům souvisejícím s právě prohlíženou stránkou[4]. Je tedy téměř nemožné bezpečně uchovat nějaké sdílené tajemství mezi serverem a klientem (v tomto případě identifikátor sezení). Jsem přesvědčen, že další komplikování modelů s udržováním sezení pomocí cookies a parametrů v URL je slepá cesta a že je tedy nutno se vydat trochu jiným směrem.

Z jednoduchých variant připadají v úvahu minimálně dvě:

- rozšíření prohlížeče pomocí pluginu, který bude obstarávat autentizaci;
- úprava vlastností „HTTP basic“ autentizace, aby byla odolná proti známým útokům.

Z hlediska nasazení v prostředí Internetu je první varianta (ač daleko čistší po stránce návrhu) téměř nemyslitelná – pokud nějaká služba nefunguje bez nutnosti speciálních pluginů, je ve většině případů odsouzena k záhubě. Navíc je nutné uvážit nutnost naprogramovat plugin pro několik prohlížečů (běžících na různých operačních systémech), jenž jsou v současné době široce využívány.

Z těchto čistě praktických důvodů jsem se rozhodl jít druhým směrem – úpravou „HTTP basic“ autentizace tak, aby tato byla zpětně kompatibilní a zároveň odolná proti hádání hesel a dalším útokům.

4. Například k URL, hlavičkám, cookies, celému obsahu dokumentu.

4.2.4 Nový návrh (vlastní řešení III)

HTTP autentizace je, kromě problému s hádáním hesel, zranitelná ještě pomocí XST útoku (popis viz část B na straně 12), který dovoluje získání uživatelského jména a hesla (dále jen „CREDS“). Oba útoky se budu snažit v tomto návrhu ošetřit.

Úprava „HTTP basic“ autentizace
Základní myšlenkou ochrany proti hádání hesel je podmínění úspěšnosti HTTP autentizace zasláním cookie, která je uživatelskému prohlížeči přidělena po úspěšném splnění veřejného Turingova testu (viz část 4.2.1 na straně 24) a slouží tedy jako důkaz. Tímto způsobem znemožníme automatizované hádání hesel, neboť útočníkův program nemá možnost Turingovy testy úspěšně řešit.

Uživatel je při pokusu o přístup do autentizované oblasti nejdříve přesměrován na stránku, která jej žádá o splnění Turingova testu. Jakmile tento test splní, je přesměrován zpět do autentizované oblasti a dále se postup (z pohledu uživatele) neliší od standardní HTTP autentizace – je vyzván k zadání CREDS a po zadání platných hodnot je mu umožněn přístup. Jako ochrana proti hádání hesel je cookie po případném několikanásobném zadání neplatných CREDS znehodnocena.

Je zřejmé, že tato cookie může, stejně jako cookies udržující sezení, vinou některého z dříve popisovaných útoků padnout do rukou útočníkovi. Vzhledem k jejímu účelu má však tato skutečnost jen nepatrné následky, v porovnání s únikem identifikátoru sezení v jiných MSS. Navíc je použit další mechanismus, který dále snižuje dopad takové situace – při prvním správném zadání CREDS je k dané cookie (na serveru) asociováno zadané uživatelské jméno. Při dalších přístupech s touto cookie jsou pak ignorována jiná uživatelská jména (počítána jako neplatné zadání CREDS). Výsledkem tedy je, že při úniku cookie získá útočník maximálně možnost vyzkoušet několik CREDS než dojde k jejímu znehodnocení. Navíc by pro úspěšné hádání musel znát již předem správné uživatelské jméno a snažit se uhodnout pouze heslo.

Posledním nutným mechanismem je zneplatnění cookies s nepřiřazeným uživatelským jménem po několika málo minutách. Tím zajistíme nejen periodický úklid nepoužitých cookies, ale také znemožníme dávkové hádání hesel, kdy útočník nějakou dobu získává platné cookies a následně je využije k hádání hesel pomocí slovníku.

Protože slovní popis nepostihuje přesně všechny detaily mnou navržené modifikace „HTTP basic“ autentizace, uvádím v další části její pseudokód

v jazyce Ruby [52]. Kód je koncipován jako DSL[5] [53] a navržen spíše jako čitelný pro člověka než proveditelný strojem. Minimální úpravou a doplněním chybějících metod bychom však získali funkční implementaci.

Ochrana proti XST
Pro neutralizaci XST útoků je nutné ošetřit obsluhu `TRACE` metody na straně Apache. Protože RFC o HTTP [6] zmiňuje `TRACE` metodu jako volitelnou, jeví se mi jako ideální řešení zpracování `TRACE` v Apache úplně zakázat. Původně byla tato úprava implementována jako součást autentizačního modulu, ale vzhledem k tomu, že podobný mechanismus byl přidán do oficiální distribuce Apache od verze 2.0.55[6] v podobě konfigurační direktivy `TraceEnable`, byla moje obsluha `TRACE` z finální verze vynechána. Pro ochranu proti XST tedy doporučuji v konfiguračním souboru nastavit `TraceEnable` na hodnotu `off`.

4.2.5 Pseudokód autentizátoru v syntaxi jazyka Ruby

```
class TuringAuth < Apache::Authenticator::Basic
  N = 4

  def authenticate_request(r)
    return redirect_to :turing unless r.cookie?

    unless r.cookie.valid? && r.cookie.attempts.size < N
      r.cookie.discard!
      log :invalid_or_overused_cookie, r
      return redirect_to :turing
    end

    return :failure unless r.credentials?

    unless r.credentials.valid?
      r.cookie.attempts << r.credentials
      return :failure
    end

    unless r.cookie.bound?
      r.cookie.binding = r.credentials
      return :success
```

5. *Domain Specific Language*, neboli doménově specifický jazyk, je programovací mini-jazyk, který je, na rozdíl od univerzálních programovacích jazyků, zaměřen na snadný zápis algoritmů v konkrétní oblasti problémů (problémové doméně).
6. V případě předchozí stabilní řady (1.3.X) je tato vlastnost dostupná od verze 1.3.34.

```
    end

    unless r.cookie.binding == r.credentials
      r.cookie.attempts << r.credentials
      log :binding_doesnt_match, r
      return :failure
    end

    return :success
  end
end
```

4.2.6 Pseudokód obsluhy Turingova testu

```
class TuringHandler < Apache::Handler
  def handle_request(r)
    unless r.post?
      r.body = TuringForm.new
      return :success
    end

    ch = TuringChallenge.find_by_id(r.params[:id])
    raise "špatné řešení" if ch.nil?
    # ^^ resp. řešení neznámé instance problému

    if ch.valid_solution?(r.params[:solution])
      r.cookie = TuringCookie.new
      return redirect_to :protected_area
    end

    raise "špatné řešení"
  rescue Exception => e
    r.body = TuringForm.new(:error => e.to_s)
    return :success
  end
end
```

4.3 Analýza nového řešení

V této části budou diskutovány bezpečnostní přínosy a důsledky výše navrženého konceptu – tedy jednotlivé útoky, proti kterým tento koncept chrání, a zároveň nově vzniklá rizika/omezení. Protože je řešení složeno ze dvou nezávislých celků, bude i tato část rozdělena na dvě nezávislé podčásti za-

bývající se analýzou bezpečnosti. V první bude diskutována oblast ochrany proti útokům DoS, v druhé pak nově navržený autentizační mechanismus.

Na závěr potom zmíním možné problémy s kompatibilitou, které koncept může způsobit v případě přítomnosti nestandardních rozšíření instalovaných do prohlížeče.

4.3.1 Analýza ochrany proti DoS

V části 2.1 na straně 9 byly popsány běžné útoky typu DoS. Implementace výše navrženého systému je odolná vůči velké části z nich, ačkoli téměř nechrání vůči zbytku.

Systém spolehlivě chrání oproti nedistribuovaným formám útoku na tabulku procesů díky omezení maximálního počtu paralelních spojení z dané IP adresy – jednak pomocí `connlimit` modulu pro iptables [45] a jednak pomocí `mod_dosevasive` modulu pro Apache. Dalším úspěšně odstíněným útokem je nedistribuovaná varianta zahlcení linky pomocí enormního množství požadavků, neboť zde opět zasáhne mod_dosevasive s jeho limitem počtu požadavků na IP adresu za jednotku času. Posledním spolehlivě odstíněným útokem je útok na síťovou vrstvu – aktivací „SYN-cookies“ je znemožněn SYN-flood útok.

Mezi útoky, které systém nedokáže spolehlivě odstínit, pak patří distribuované varianty výše zmíněných útoků, útok na systémové zdroje a útoky na chybu aplikace.

Vhodným (avšak finančně náročným) rozšířením stávajícího systému, které eliminuje některé z neodstíněných tříd útoků, je rozhodně nasazení výkonného stavového firewallu nebo produktu podobné funkcionality jako je DefensePro (viz ochrana proti DoS v části 3.1 na straně 17).

Dalším užitečným doplňkem je pak striktní packetový filtr (odstínění ICMP packetů) s možností omezení toků (*traffic shaper*) na straně poskytovatele konektivity, který by měl citelně snížit riziko DoS útoků na síťovou konektivitu i některé typy útoků na síťovou vrstvu. Tento doplněk není uveden jako součást mého řešení, neboť v praxi brání nasazení takových mechanismů neochota poskytovatelů konektivity.

Spíše pro zamyšlení (a námět pro budoucí práci) je pak možnost využití informací publikovaných v seznamu veřejných proxy serverů (*Open Proxy List*) [54]. Efektivní využití takového seznamu pro ochranu HTTP serverů bude nejspíše vyžadovat úzkou integraci se síťovou vrstvou nebo firewallem hostitelského operačního systému.

4.3.2 Analýza autentizačního mechanismu

V části 2.2 na straně 11 byly popsány běžné útoky na autentizační mechanismy. Zde se pokusím diskutovat proveditelnost těchto útoků oproti novému konceptu a zároveň – v případě že je útok proveditelný – ukázat jaké důsledky může mít.

Krádež identifikátoru sezení nebo CREDS
Vzhledem k tomu, že systém nevyužívá žádný identifikátor sezení, je tento útok neaplikovatelný.

Jediná cookie, kterou systém využívá, plní úlohu důkazu, že při přihlašování byl použit prohlížeč obsluhovaný člověkem a tudíž nešlo pouze o počítačový program. Případná krádež této cookie neposkytuje útočníkovi žádnou výhodu, neboť již má přiřazené uživatelské jméno a tedy je k hádání hesel nepoužitelná bez znalosti tohoto uživatelského jména. I v případě jeho znalosti poskytuje nejvýše stejný počet pokusů, které poskytuje nově vygenerovaná cookie po splnění Turingova testu.

Krádeži CREDS pomocí XST je zamezeno díky odstranění TRACE metody, takže ani v tomto směru útočník nezískává žádnou výhodu.

Krádež identifikátoru „fixací“ sezení
Stejně jako v předchozím případě je tento útok neaplikovatelný. V případě nastavení cookie na nějakou útočníkovi známou hodnotu je uživatel buď okamžitě přesměrován na Turingův test a nebo (v případě, že šlo o platnou cookie) požádán o zadání svých CREDS.

Můžeme předpokládat, že pokud útočníkovým cílem nebylo usnadnit uživateli přihlašování do systému (tím, že za něj vyřeší Turingův test), nezískává opět žádnou výhodu.

Krádež identifikátoru sezení nepřímo
V tomto případě je nutné uvážit následující dvě možnosti:

- krádež cookie:
  v takovém případě je závěr naprosto shodný se závěrem u bodu „Krádež identifikátoru sezení nebo CREDS“;
- krádež CREDS:
  k takové by mohlo dojít odposlechem packetů nebo sledováním provozu na proxy serveru. Bohužel tento útok má šanci na úspěch, dokud nebude celá autentizovaná část přístupná pouze přes HTTPS protokol.

Teoretickou možností do budoucna (až se zlepší stav podpory „HTTP digest“ autentizace v prohlížečích) je úprava „HTTP digest“ autentizace obdobným způsobem jako stávající úprava „HTTP basic“.

Útoky (slovníkový a hrubou silou) na přihlašovací údaje
Jedna z motivací pro tvorbu zcela nového konceptu byla odolnost proti automatizovanému „hádání“ hesel pomocí hrubé síly nebo slovníku. Jak bylo již výše popsáno, je nutné při přihlášení vyřešit Turingův test, kterým přihlašující-se klient prokazuje, že při přihlášení došlo k interakci s člověkem a nikoli pouze s nějakým automatizovaným programem.

Prolomení logiky autentizačního mechanismu
Vzhledem k jednoduchosti konceptu, důslednému dodržování principů bezpečného programování a faktu, že implementace byla provedena formou úpravy stávajícího kódu modulu pro „HTTP basic“ autentizaci, si dovoluji tvrdit, že tento útok není možný.

Útok na mechanismus uchovávání identifikátorů sezení
Tento útok je opět neaplikovatelný, neboť systém nevyužívá ani cookies ani parametr v URL k identifikaci sezení.

Útok hrubou silou na identifikátor sezení
Vzhledem k tomu, že systém nepoužívá žádné identifikátory sezení, je tento útok opět neaplikovatelný.

Útok hrubou silou na použité cookies vidím spíše jako teoretickou možnost bez možnosti praktické realizace, neboť jak je zřejmé z části 4.4.4 na straně 37, jsou cookies náhodně generované identifikátory o délce minimálně 256 bitů. V případě cookies, které ještě nejsou přiřazené žádnému uživatelskému jménu, je jejich životnost navíc v řádech minut, což vylučuje jakýkoli praktický útok. V případě přiřazených je zase nemožné odvodit z dané cookie platné uživatelské jméno, takže úspěšnost hádání hesla přes takovou cookie je řádově nižší než u nepřiřazených.

Útok pomocí zvýšení práv
Tento útok je opět možný, a jediná možnost jak jeho riziko minimalizovat tkví v přiměřené péči o zabezpečení serverů, na kterých je systém využit.

4.3.3 Možné problémy s kompatibilitou

Vzhledem k tomu, že koncept ke správné funkci autentizace vyžaduje předávání cookies, může způsobit problémy při používání externích správců pro stahování obsahu (download manager). Protože nemám praktické zkušenosti s takovými programy a tudíž nejsem schopen ověřit, zda v takových případech dochází ke korektnímu předání cookies, uvádím tuto možnou nekompatibilitu jako varování pro případné zájemce o nasazení tohoto konceptu v praxi.

4.4 Implementační detaily

Protože je implementace obrany proti DoS popsaná dostatečně podrobně již v úvodním textu, budu se v této části věnovat pouze implementaci ochrany autentizačního mechanismu.

Mým původním záměrem bylo implementovat tuto ochranu jako dva spolupracující FastCGI [55] skripty. Bohužel jsem v průběhu implementace odhalil několik klíčových nedostatků FastCGI modulu, které tuto cestu vyloučily. Proto jsem zvolil alternativní přístup, jehož výsledkem jsou tři spolupracující podsystémy:

1. Autentizační modul do Apache2 v jazyce C.
2. Autentizační server v jazyce Ruby.
3. CGI skript v jazyce Ruby implementující Turingův test.

Dále se budu věnovat jednomu každému z nich.

4.4.1 Autentizační modul

Modul vznikl úpravou původního mod_auth z distribuce Apache, což je modul obstarávající „HTTP basic“ autentizaci. Podstatou úpravy bylo odstranění veškerého kódu pro ověřování přihlašovacích údajů a následné delegování této úlohy na rutinu, která komunikuje přes UDP s autentizačním serverem. Tomu předá nejen uživatelské údaje (jméno, heslo), ale též cookie. Na základě odpovědi serveru pak vrátí příslušný stavový kód nebo uživatele přesměruje na výše zmíněný CGI skript.

Podstatnou změnou je tedy fakt, že modul samotný neobsahuje žádnou autentizační logiku a je pouze „otrokem“ autentizačnímu serveru.

Jeho konfigurace je podobná konfiguraci mod_auth, vyžaduje však nastavení několika dalších parametrů. Jejich význam je zřejmý z následujícího okomentovaného příkladu:

```
<Directory ”/var/www/webserver/htdocs/secured/”>
    AuthType Basic
    AuthName ”Protected area”

    # Adresa a port autentizačního serveru
    TuringAuthServerAddr ”127.0.0.1”
    TuringAuthServerPort ”28645”
    # Kolikrát opakovat pokus o spojení se serverem?
```

```
    TuringAuthRetries 2
    # Jak dlouho čekat na odpověď?
    TuringAuthWait 1
    # Jméno cookie
    TuringAuthCookieName "turing_passport"
    # URL Turing CGI skriptu
    TuringAuthURL "http://webserver/turing/"
    # Logovat všechna přesměrování na Turing CGI?
    TuringAuthLogRedirects On
    # Logovat selhání autentizace při nezadaném
    # uživatelském jménu a heslu?
    TuringAuthLogAuthFailNoCreds On

    require valid-user
</Directory>
```

Výchozí hodnoty jednotlivých direktiv a bližší informace o funkci modulu je možné nalézt ve zdrojovém kódu na přiloženém CD.

Je dobré upozornit, že díky implicitnímu předpokladu o fungování „HTTP basic" autentizace ve zdrojových kódech Apache je pro správnou funkci modulu nutné, aby byl první při zpracování obou autentizačních fází. V opačném případě bude uživatel dotázán na jméno/heslo namísto přesměrování na výše zmíněný CGI skript a po následném zadání teprve přesměrován. Není to sice úplnou překážkou ve fungování, ale navržený protokol je tím narušen. Z toho důvodu modul registruje svoji obsluhu obou fází (`authenticate_basic_user`, `check_user_access`) jako `APR_HOOK_FIRST` místo původního `APR_HOOK_MIDDLE`, což by mělo zajistit přednost před mod_auth bez ohledu na pořadí načtení modulů.

4.4.2 Autentizační server

Autentizační server obsahuje komunikační rozhraní s oběma zbývajícími prvky systému – jednak komunikuje pomocí UDP s autentizačním modulem (implementace autentizační logiky je až na drobné odlišnosti v objektovém modelu shodná s pseudokódem v části 4.2.5 na straně 29) a „komunikuje" s CGI skriptem, se kterým sdílí adresář určený k ukládání PStore[7] databází udržujících stavové informace o jednotlivých autentizačních cookies.

Implementace serveru je velmi přehledně rozdělena do několika tříd, které plní jednotlivé funkce a jejich provázání (*coupling*) je nízké. Výhodou takového řešení je velmi snadná náhrada jednotlivých částí kódu, například

7. PStore je knihovna umožňující transakční přístup ke stromu objektů, který je udržován v souborech.

výměna souborové databáze hesel za databázi uloženou v nějakém RDBMS. Zároveň bych chtěl upozornit, že implementaci některých tříd považuji pouze za referenční – pro reálný provoz by bylo vhodné zavést optimalizace, které však povedou k znepřehlednění kódu.

Konfigurace serveru se provádí zadáním příslušných přepínačů z příkazové řádky. Přehled možných voleb je zde (původní nápověda získaná pomocí `--help` byla počeštěna a přeformátována):

```
Použití: authserv.rb [volby]

Možné volby:
    -p, --port PORT
      Port na kterém poslouchat, standardně: 28645
    -n, --max-attempts ATTEMPTS
      Maximální počet pokusů o autentizaci před
      znehodnocením dané cookie, standardně: 4
    -f, --pw-file FILE
      Soubor s hesly, standardně: ./.htpasswd
    --[no-]crypted-passwords
      Je soubor kryptovaný? Standardně: ano
    -s, --store-dir DIR
      Adresář, kam ukládat cookies, standardně: ./store/
    -l, --unbound-cookie-lifetime TIME
      Životnost (ve vteřinách) cookies, které nemají
      přiřazené uživatelské jméno, standardně: 300
    -v, --verbose
      "Užvaněný" provoz - tedy výpis informačních hlášení,
      která jsou normálně potlačena
    -h, --help
      Tato nápověda
```

4.4.3 CGI skript implementující Turingův test

Skript ke svojí práci využívá knihovnu pro generování Turingových testů s prostým názvem Turing. Tuto knihovnu jsem během tvorby diplomové práce implementoval a následně zveřejnil jako open-source (pod GPL licencí) na Rubyforge [56].

Díky vysoké abstrakci výše zmíněné knihovny je implementace Turingova testu záležitostí správné konfigurace a jedné metody, která zajišťuje vygenerování nové autentizační cookie poté, co uživatel úspěšně splní test.

Konfigurace parametrů se pro zjednodušení provádí přímo uvnitř CGI skriptu. Mezi nezbytně nutné parametry patří následující hodnoty:

```
# Adresář sdílený s autentizačním serverem
Settings.cookie_store = '/var/www/webserver/cookie-store/'

tcgi_config = {
    # URL prefix (cesta) ke generovaným obrázkům
    :imagepath => "/turing/imgs/",
    # cesta v rámci systému souborů k adresáři s obrázky
    :outdir => '/var/www/webserver/htdocs/turing/imgs',
    # PStore obsahující data (id, odpověď, čas)
    # jednotlivých testů odeslaných klientům
    :store => '/var/www/webserver/turing.pstore',
    # adresa na kterou přesměrovat při úspěchu
    # (musí být absolutní URL)
    :redirect_to => 'http://webserver/secured/',
}

# Jméno cookie (musí být stejné s konfigurací modulu)
$cookie_name = 'turing_passport'
# Expirace dané cookie ve vteřinách
$cookie_expiry = 30*24*3600 # 30 dní
```

Další (nepovinné) parametry je možné najít v dokumentaci knihovny Turing. Tyto parametry umožňují ovlivnit životnost vygenerovaných testů, vzhled a rozměry generovaných obrázků, použité HTML šablony, použitý slovník a další aspekty chování. Dokumentace též stručně popisuje způsob, jakým je možné knihovnu rozšířit o další druhy generátorů.

Skript je též schopen bez úprav pracovat jako FastCGI skript – jediným nutným předpokladem je instalace `fcgi` knihovny pro Ruby a správné nastavení FastCGI podpory v konfiguračním souboru Apache.

4.4.4 Generování identifikátorů autentizačních cookies

Identifikátory autentizačních cookies jsou tvořeny zpracováním náhodných dat, která jsou získávána ze znakového zařízení (*character device*) `/dev/urandom`, pomocí jednocestné funkce SHA-2 o volitelné velikosti výstupu. Standardně je velikost výstupu nastavena na 256 bitů, ale je možné použít i 384 nebo 512 bitů.

Samozřejmě je možné generátor pouhou změnou konstanty upravit tak, aby jako svůj vstup využíval zařízení `/dev/random`, jehož výstup obsahuje pouze náhodné bity odvozené z náhodných hardwarových událostí[8].

Důvod, proč jsem zvolil `/dev/urandom` jako implicitní tkví v tom, že je, na rozdíl od `/dev/random`, schopen poskytovat velké objemy kvalitních pseudonáhodných bitů[9] a navíc je u něho garantováno, že nikdy nezablokuje proces v důsledku nedostatku náhodných bitů (způsobeného nedostatkem náhodných hardwarových událostí). Bližší informace o implementaci obou náhodných blokových zařízení je možné nalézt ve zdrojových kódech jádra operačního systému Linux – v souboru `drivers/char/random.c`.

Zároveň je ošetřena i situace, kdy dojde k výjimce při generování náhodného identifikátoru (například vinou neexistence zadaného blokového zařízení) – v takovém případě je pomocí SHA-2 zpracován vstup složený z identifikátoru procesu, aktuálního času a dvou náhodných čísel z interního pseudonáhodného generátoru v Ruby.

8. Mezi sledované události patří: prodlevy mezi stisky kláves, prodlevy mezi přerušeními (*interrupts*) a další nedeterministické události, které případný útočník nemůže jednoduše sledovat.
9. Bity generuje pomocí pseudonáhodného generátoru (PRNG) založeného na SHA-1, jehož vnitřní stav je inicializován a průběžně měněn (*seeded*) náhodnými bity z hardwaru.

Kapitola 5

Praktické výsledky

V této kapitole se krátce zmíním jednak o ohlasech uživatelů a jednak o výsledcích testování výkonu, které jsem provedl.

5.1 Ohlasy uživatelů

Mnou implementovaný systém byl po dobu jednoho týdne podroben testování skupinou uživatelů z řad mých kolegů v práci, neboť jím byla nahrazena autentizace do firemního intranetu, která je normálně řešena klasickou „HTTP basic“ autentizací bez dalších úprav[1].

Ohlas uživatelů na změnu byl veskrze pozitivní. Ukázalo se, že většina z nich je s Turingovými testy již seznámena[2], takže pro ně jejich vyplnění nepředstavovalo větší problém. Jediným zaznamenaným problémem s Turingovými testy byla občasná snížená čitelnost některých instancí (zejména vytvořených metodou `Blending`).

Mezi uživateli oceněné vlastnosti nové autentizace patří:

- zpětná kompatibilita (odpadla nutnost znovu zadávat heslo, které je uloženo (zapamatováno) prohlížečem);
- trvalé uložení autentizační cookie (odpadla nutnost opět plnit Turingův test při dalším přihlášení).

Zároveň jsem obdržel negativní zpětnou vazbu po zveřejnění „Turing“ knihovny – byl jsem upozorněn, že v současné verzi knihovna nijak neřeší problém zrakově postižených uživatelů. Jako námět na možné řešení uvedl autor připomínky odkaz na textovou verzi Turingova testu, která využívá relativně jednoduché slovní matematické úlohy [57].

---

1. Rozumné zvýšení odolnosti této autentizace proti hádání hesel bylo dosaženo volbou „politiky hesel“ – hesla jsou náhodně generována systémem, uživatelé nemají možnost si své heslo zvolit.
2. Zejména díky registraci na freemail službách (seznam, yahoo . . .), ale také díky odesílání SMS přes web do mobilních sítí Oskar a Eurotel.

5.2 Testování výkonu

Protože je nově navržený systém programován ve vyšším jazyce, je testování výkonu poměrně důležitou součástí vývojového cyklu – nezřídka se totiž stává, že díky vysoké abstrakci dochází k neúnosnému zpomalení. To je obvykle způsobeno režií interpretru daného jazyka nebo díky skryté složitosti některých elementárních operací.

Proto jsem se při testování výkonu postupně zaměřil na:

- režii spojenou s generováním Turingova testu;
- režii autentizačního modulu.

V následujících částech 5.2.1 a 5.2.2 provedu jen stručné shrnutí dosažených výsledků. Detailní informace (včetně kompletních zdrojových kódů testovacích skriptů a mnou získaných testovacích hodnot) je možné nalézt na přiloženém CD.

Všechny testy byly provedeny na mém domácím počítači (Athlon64 3200+, 1GB RAM) s operačním systémem Gentoo Linux a jádrem 2.6.13-gentoo-r2.

5.2.1 Režie spojená s generováním Turingova testu

Testování režie při generování Turingova testu bylo provedeno měřením odezvy na požadavky při měnícím se počtu paralelních spojení. V rámci každého módu (CGI, FastCGI) bylo pro každý test vyžádáno 1000 Turingových testů – vždy s různým počtem paralelních spojení (1, 5, 20, 50).

Z následujících výsledků je patrné, že CGI mód je přibližně třikrát pomalejší než FastCGI. Zároveň je zřejmé, že počet paralelních požadavků nemá podstatný vliv na průměrnou dobu odezvy:

	průměr (ms)		pož/s	
paralelně	cgi	fcgi	cgi	fcgi
1	270.63	87.33	3.70	11.45
5	268.58	84.00	3.72	11.90
20	265.42	83.54	3.77	11.97
50	235.65	87.49	4.24	11.43

Výsledky testu mě však upozornily na potenciální problém: Turing knihovna využívá služeb PStore knihovny pro ukládání informací o vygenerovaných Turingových testech. PStore však při každém požadavku znovu načítá celou databázi objektů z disku (a po změně ji ukládá zpět).

S počtem vygenerovaných (a ještě nesplněných Turingových testů) však roste i čas potřebný pro tuto operaci. Ačkoli je nárůst časové náročnosti pro načtení informací o 1000 vygenerovaných Turingových testech (což je myslím více než obvyklé využití) stále neznatelný, bylo by myslím do budoucna vhodnější využít pro ukládání informací nějaký jiný databázový systém – například SQL databázi, případně obdobu `AuthCookie` databáze z autentizátoru.

5.2.2 Režie autentizačního modulu

Test autentizačního modulu jsem provedl jako porovnávací oproti standardnímu mod_auth v Apache. Opět jde o test s měřením odezvy na požadavky při rozdílném počtu paralelních spojení. Navíc jsem zavedl ještě jednu proměnnou – počet CREDS v souboru s hesly (dále jen „PU“).

Výsledek tohoto měření mě překvapil – očekával jsem horší reakční dobu mého modulu oproti mod_auth při nízkém PU a naopak daleko lepší odezvu při vysokém PU. Zmíněným překvapením je fakt, že můj modul dosahuje lepších výsledků již při nízkém PU. Samozřejmě dosahuje daleko lepších výsledků ve chvíli, kdy je PU $\geq$ 1000, neboť soubor s hesly načítá jen při změně.

Protože program „Apache Bench“, kterým jsem prováděl měření, není schopen poskytnout data s vyšší přesností než 1 ms, musel jsem snížit frekvenci svého procesoru (*CPU*) na 1000 MHz, abych byl schopen poskytnout alespoň rámcově přesné výsledky. Z toho důvodu je dobré následující tabulku brát s jistou rezervou:

	průměr (ms)		pož/s	
PU	mod_auth	turing	mod_auth	turing
1	3.12	1.27	321.03	788.02
10	3.11	1.28	321.44	779.42
100	4.27	1.36	234.36	736.38
1000	7.64	1.35	130.94	740.19

Kapitola 6

Závěr

Zadáním této diplomové práce bylo navrhnout, implementovat a otestovat systém pro ochranu webových serverů proti různým druhům útoků. Výsledný systém měl navíc zajistit odolnost autentizačních mechanismů proti hádání hesel a krádeži sezení.

Mojí snahou bylo poskytnout čtenáři kromě samotného řešení také potřebný základ v podobě principů fungování jednotlivých protokolů a zejména pak v podobě popisu jednotlivých zranitelných míst, která jsou pro jednotlivé útoky využívána. Neméně zajímavým je rozbor ochranných prvků, které jsou pro odstínění popsaných útoků běžně používané. Práce navíc obsahuje i jistou „evoluci" mých řešení, která ilustruje úskalí spojená s návrhem nových autentizačních mechanismů – předchozí verze jednotlivých řešení obsahovaly netriviální chyby znemožňující jejich úspěšné použití v masovém měřítku.

Kromě již zmíněného podrobného popisu související problematiky vidím přínos práce i ve výsledném systému, který je možné díky zpětné kompatibilitě jednoduše použít v praxi. Pokud by nasazení systému bránila platformová nekompatibilita, je jednoduché vytvořit vlastní implementaci na základě textu této práce.

Zajímavý pohled na mnou analyzované bezpečnostní problémy nabízí i excelentní kniha Michala Zalewského „Silence on the Wire" [58], která čtenáři nabízí mnoho brilantních konceptů důstojných dalšího prozkoumání.

Protože se jedná o rozsáhlou problematiku, nabízí se několik možných cest pro budoucí výzkum, případně rozšíření vytvořeného řešení:

- Pro ochranu proti DoS útokům by mohlo být výhodné využít informací uložených v různých seznamech veřejných proxy serverů, jako je např. Blitzed [54].
- Další cenné informace by bylo možné získat detailní analýzou chování systému, který je pod reálným DoS útokem, neboť simulace DoS útoků nemají z hlediska analýzy a návrhu nových ochran patřičnou vypovídací hodnotu.

- Možnou optimalizací ochrany proti DoS by mohlo být též využití centrálního počítání příchozích požadavků uvnitř mateřského procesu Apache nebo modulem stavového firewallu, což by mohlo přinést rychlejší reakci na případný útok.
- Pro lepší ochranu autentizované oblasti by mohlo být vhodné vytvořit analogickou implementaci i pro „HTTP digest“ autentizaci, pokud se ovšem podpora této metody v prohlížečích do budoucna zlepší.
- Vhodným doplňkem by též byla implementace alternativních Turingových testů, které jsou řešitelné i zrakově postiženými uživateli.
- Z hlediska nasazení do praxe by mohlo být výhodné změnit způsob ukládání informací o Turingových testech tak, aby bylo možné celý systém nasadit i pro systémy, které rozkládají zátěž na více strojů. Tím by se zároveň ošetřil již zmíněný problém s velkým množstvím aktivních Turingových testů (viz část 5.2 o výkonovém testu).

Zadání práce bylo splněno, neboť jak jsem již v částech 4.3.1 na straně 31 a 4.3.2 na straně 32 zmínil, je navržený systém odolný proti nedistribuovaným variantám DoS útoků a naprosté většině běžných útoků na autentizační mechanismy. Zároveň byl navržený autentizační mechanismus otestován koncovými uživateli (viz část 5.1 na straně 39), kteří jej shledali vyhovujícím. Stejně tak testy výkonu (viz část 5.2 na straně 40) ukazují, že je daný systém dostatečně rychlý i pro běžné nasazení v praxi.

Literatura

[1] J Postel. Internet Protocol. Defense Advanced Research Projects Agency. 1981. http://www.ietf.org/rfc/rfc791.txt (2006-01-01)

[2] Y Rekhter, B Moskowitz, D Karrenberg, GJ de Groot, E Lear. Address Allocation for Private Internets. Network Working Group. 1996. http://www.ietf.org/rfc/rfc1918.txt (2006-01-01)

[3] J Postel. Internet Control Message Protocol. Network Working Group. 1981. http://www.ietf.org/rfc/rfc792.txt (2006-01-01)

[4] J Postel. Transmission Control Protocol. Defense Advanced Research Projects Agency. 1981. http://www.ietf.org/rfc/rfc793.txt (2006-01-01)

[5] K Egevang. The IP Network Address Translator (NAT). Network Working Group. 1994. http://www.ietf.org/rfc/rfc1631.txt (2006-01-01)

[6] R Fielding, J Gettys, J Mogul, H Frystyk, L Masinter, P Leach, T Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. 1999. http://www.ietf.org/rfc/rfc2616.txt (2006-01-01)

[7] J Franks, P Hallam-Baker, J Hostetler, S Lawrence, P Leach, A Luotonen, L Stewart. HTTP Authentication: Basic and Digest Access Authentication. 1999. http://www.ietf.org/rfc/rfc2617.txt (2006-01-01)

[8] D Kristol, L Montulli. HTTP State Management Mechanism. Network Working Group. 1997. http://www.ietf.org/rfc/rfc2109.txt (2006-01-01)

[9] D Kristol, L Montulli. HTTP State Management Mechanism. Network Working Group. 2000. http://www.ietf.org/rfc/rfc2965.txt (2006-01-01)

[10] CERT Coordination Center. Denial of Service Attacks. 1997-2001. http://www.cert.org/tech_tips/denial_of_service.html (2006-01-01)

[11] D Dittrich. Distributed Denial of Service (DDoS) Attacks/tools. 1999-2004. http://staff.washington.edu/dittrich/misc/ddos/ (2006-01-01)

[12] Wikipedia. Mydoom. 2004. http://en.wikipedia.org/wiki/Mydoom (2006-01-01)

[13] Apache Foundation. The Apache HTTP Server Project. 1995-2004. http://httpd.apache.org/ (2006-01-01)

[14] CA Huegen. The latest in denial of service attacks: ”smurfing” description and information to minimize effects. 2000. http://wejn.org/smurf.txt (2006-01-01)

[15] Wikipedia. Smurf attack. 2004. http://en.wikipedia.org/wiki/Smurf_attack (2006-01-01)

[16] CERT Coordination Center. CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks. 1998-2000. http://www.cert.org/advisories/CA-1998-01.html (2006-01-01)

[17] Wikipedia. SYN flood. 2004. http://en.wikipedia.org/wiki/SYN_flood (2006-01-01)

[18] CERT Coordination Center. CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks. 1996-2000. http://www.cert.org/advisories/CA-1996-21.html (2006-01-01)

[19] Wikipedia. Blaster worm. 2003-2004. http://en.wikipedia.org/wiki/Blaster_worm (2006-01-01)

[20] D Dittrich. The DoS Project’s ”trinoo” distributed denial of service attack tool. 1999. http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt (2006-01-01)

[21] D Dittrich. The ”stacheldraht” distributed denial of service attack tool. 1999. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt (2006-01-01)

[22] PHP: Hypertext Preprocessor. http://php.net/ (2006-01-01)

[23] CERT. Malicious HTML tags embedded in client Web requests. 2000. http://www.cert.org/advisories/CA-2000-02.html (2006-01-01)

[24] J Grossman. Cross-Site Tracking (XST). Whitehat Security. 2003. http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf (2006-01-01)

[25] Microsoft Corporation. Mitigating Cross-site Scripting With HTTP-only Cookies. 2004. http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp (2006-01-01)

[26] M Kolšek. Session Fixation Vulnerability in Web-based Applications. Acros security. 2002. http://www.acros.si/papers/session_fixation.pdf (2006-01-01)

[27] Dynamic Network Services. http://dyndns.org/ (2006-01-01)

[28] Wikipedia. DNS cache poisoning. 2005. http://en.wikipedia.org/wiki/DNS_poisoning (2006-01-01)

[29] Wikipedia. Phishing. 2005. http://en.wikipedia.org/wiki/Phishing (2006-01-01)

[30] Wikipedia. ARP spoofing. 2005. http://en.wikipedia.org/wiki/ARP_Spoofing (2006-01-01)

[31] Wikipedia. Spyware. 2005. http://en.wikipedia.org/wiki/Spyware (2006-01-01)

[32] Netcraft. Webserver survey archives. 2004. http://news.netcraft.com/archives/web_server_survey.html (2006-01-01)

[33] K Fu, E Sit, K Smith, N Feamster. Dos and Don'ts of Client Authentication on the Web. Proceedings of the 10th USENIX Security Symposium. 2001. http://www.pdos.lcs.mit.edu/papers/webauth:tr.ps.gz (2006-01-01)

[34] AJ Menezes, PC van Oorschot, SA Vanstone. Handbook of Applied Cryptography. CRC Press. August 2001. ISBN: 0-8493-8523-7. http://www.cacr.math.uwaterloo.ca/hac/ (2006-01-01)

[35] J Lemon. Resisting SYN flood DoS attacks with a SYN cache. Usenix, Proceedings of the BSDCon 2002 Conference. 2002. http://www.usenix.org/publications/library/proceedings/bsdcon02/full_papers/lemon/lemon.pdf (2006-01-01)

[36] RadWare. DefensePro. http://www.radware.com/content/products/dp/default.asp (2006-01-01)

[37] mod_dosevasive modul pro Apache httpd. http://www.nuclearelephant.com/projects/mod_evasive/ (2006-01-01)

[38] Zeus Webserver. http://www.zeus.com/ (2006-01-01)

[39] Wikipedia. SQL injection. 2004. http://en.wikipedia.org/wiki/SQL_injection (2006-01-01)

[40] JEF Friedl. Mastering Regular Expressions, Second Edition. O'Reilly Media, Inc. July 2002. ISBN: 0-596-00289-0. http://www.amazon.com/exec/obidos/ASIN/0596002890 (2006-01-01)

[41] Aleph One. Smashing The Stack For Fun And Profit. Phrack magazine, issue 49, file 14. 1996. http://www.phrack.org/phrack/49/P49-14 (2006-01-01)

[42] I Ristic. ModSecurity – Open Source Web Application Firewall. 2005. http://modsecurity.org/ (2006-01-01)

[43] Killer Security. http://killersecurity.com/ (2006-01-01)

[44] PennyWize – password protection online. http://pennywize.com/ (2006-01-01)

[45] The netfilter/iptables project. http://netfilter.org/ (2006-01-01)

[46] lighttpd a secure, fast, compliant and very flexible web-server. http://lighttpd.net/ (2006-01-01)

[47] Carnegie Mellon University. The CAPTCHA Project. 2002-2004. http://captcha.net/ (2006-01-01)

[48] B Pinkas, T Sander. Securing Passwords Against Dictionary Attacks. Proceedings of the ACM Computer and Communications Security Conference. 2002. http://pinkas.net/PAPERS/pwdweb.pdf (2006-01-01)

[49] Apple. Dynamic HTML and XML: The XMLHttpRequest Object. http://developer.apple.com/internet/webcontent/xmlhttpreq.html (2006-01-01)

[50] World Wide Web Consortium. Document Object Model. http://www.w3.org/DOM/ (2006-01-01)

[51] A Rager. XSS-Proxy: A tool for realtime XSS hijacking and control. http://xss-proxy.sourceforge.net/ (2006-01-01)

[52] Y Matsumoto. Ruby: Programmers' Best Friend. http://ruby-lang.org/ (2006-01-01)

[53] M Fowler. Domain Specific Language. http://martinfowler.com/bliki/DomainSpecificLanguage.html (2006-01-01)

[54] Blitzed Open Proxy Monitor. http://opm.blitzed.org/info/ (2006-01-01)

[55] MR Brown. FastCGI Specification. Open Market, Inc. 1996. http://www.fastcgi.com/devkit/doc/fcgi-spec.html (2006-01-01)

[56] M Šafránek. Turing – Ruby implementation of Captcha. 2005. http://turing.rubyforge.org/ (2006-01-01)

[57] G Kistner. Ruby Quiz – Math Captcha. 2005. http://rubyquiz.com/quiz48.html (2006-01-01)

[58] M Zalewski. Silence on the Wire – A Field Guide to Passive Reconnaissance and Indirect Attacks. No Starch Press. April 2005. ISBN: 1-59327-046-1. http://www.amazon.com/exec/obidos/ASIN/1593270461 (2006-01-01)