

FAKULTA INFORMATIKY MASARYKOVY UNIVERZITY



**Použití mechanismu Kerberos pro autentizaci
v HTTP**

BAKALÁŘSKÁ PRÁCE

Luděk Šulák

Brno, 2002

Prohlašuji, že tato 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.

Na tomto místě bych chtěl vyjádřit poděkování vedoucímu své bakalářské práce Mgr. Danielovi Kouřilovi za odborné vedení a cenné rady.

Shrnutí

Tato bakalářská práce se zabývala přidáním kerberovské autentizace do protokolu HTTP. Její součástí jsou upravené aplikace Apache a Mozilla.

Klíčová slova

WWW, HTTP, autentizace, GSS-API, Apache, Mozilla, Kerberos

Obsah

1	Úvod	1
2	Použité protokoly	2
2.1	Transport Layer Security protocol (TLS)	2
2.2	Kerberos	2
2.3	Hypertext Transfer Protocol (HTTP)	3
2.3.1	Autentizace v HTTP	3
3	Programování bezpečnostních aplikací	5
3.1	Generic Security Service Application Program Interface (GSS-API)	5
3.1.1	Pseudomechanismus SPNEGO	7
3.1.2	Příklad použití GSS-API	7
4	Zabezpečení WWW v projektu MetaCentrum	9
4.1	Dosavadní metoda autentizace	9
4.2	Využití systému Kerberos pro HTTP	10
4.3	Implementace návrhu	11
4.3.1	Apache	11
4.3.2	Mozilla	12
4.3.3	Příklad autentizované komunikace	13
5	Závěr	16
A	Obsah přiloženého CD	17

Kapitola 1

Úvod

Protokolem HTTP se stále častěji přenášejí důležitější a důvěrnější údaje, které je potřeba co nejefektivněji chránit proti odposlechu nebo zneužití. Uživatelé jsou často nuceni pamatovat si množství hesel, která je nutné zadávat do různých autentizačních systémů, což samozřejmě vede k tomu, že uživatelé používají stále stejná hesla, případně si je zapisují přímo na pracovní počítač nebo dokonce do souboru na disku.

Tato práce si dala za úkol obohatit protokol HTTP o kvalitnější a z pohledu uživatele pohodlnější metodu autentizace. Nevýhodou v této oblasti je to, že k většině používaných prohlížečů nejsou dostupné zdrojové texty, a proto je velmi nesnadné k nim přidávat nové vlastnosti bez spolupráce jejich vývojářů. Cílem práce bylo vytvořit nástroje použitelné v nejširším možném měřítku.

Kapitola 2

Použité protokoly

V této kapitole jsou popsány protokoly, které jsem použil během práce na mém bakalářském projektu.

2.1 Transport Layer Security protocol (TLS)

Protokol TLS [3] je nástupcem protokolu SSL. Hlavním cílem tohoto protokolu je vytvoření bezpečného kanálu mezi klientem a serverem. Protokol je navržen tak, aby mohl být použit jako mezivrstva mezi síťovým protokolem TCP a (téměř) libovolným aplikačním protokolem. Tímto způsobem lze bezpečně provozovat i protokoly, které neobsahují žádné vlastní bezpečnostní mechanismy.

Protokol TLS je založen na použití asymetrické kryptografie. Pro autentizaci komunikujících stran se používají X.509 certifikáty veřejných klíčů. V rozsáhlejších prostředích, je potřeba mít k dispozici systém důvěryhodných certifikačních autorit, které svým uživatelům vydávají certifikáty.

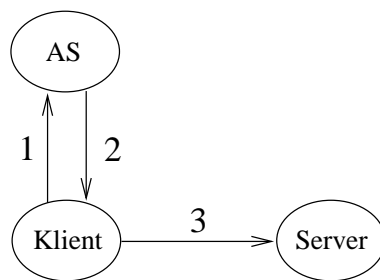
Mechanismus SSL se v současné době hodně používá např. pro zabezpečení protokolů POP3 pro stahování elektronické pošty nebo HTTP pro webovou komunikaci. Více informací o tomto mechanismu lze získat např. na stránkách projektu *OpenSSL* (<http://www.openssl.org>).

2.2 Kerberos

Kerberos [1, 2] je autentizační mechanismus, který používá symetrickou kryptografii. Každá organizace, která používá Kerberos si udržuje autentizační server (*AS* nebo také *KDC*), kde je seznam všech uživatelů dané organizace, spolu s jejich hesly. Princip fungování mechanismu Kerberos je znázorněn na obr. 2.1.

Pokud uživatel chce použít nějakou chráněnou službu, musí nejprve získat kerberovský lístek pro tuto službu. Lístky vydává AS (*KDC*), kterému se uživatel prokáže svým heslem. Lístky mají zpravidla platnost několik hodin (obvykle deset) a mohou být použity pouze uživatelem, pro kterého byly vydány.

Výhodou Kerbera je, že hesla mohou být umístěna centrálně a AS může zajistit spolehlivou verifikaci uživatelů. Nevýhodou je nutnost velmi spolehlivého zabezpečení *KDC*.



Obrázek 2.1: Základní princip mechanismu Kerberos

2.3 Hypertext Transfer Protocol (HTTP)

HTTP [4] je zřejmě nejpoužívanější protokol současného Internetu. Primárně byl navržen pro přenos HTML stránek z webových serverů, ale používá se i pro řadu jiných aplikací. Základním HTTP požadavkem je `GET <URL>`, kterým klient žádá o zaslání určité stránky ze serveru. Server požadavek zpracuje a v případě, že došlo k chybě, vrátí klientovi chybovou zprávu (např. 401 – Unauthorized) spolu s bližší specifikací chyby, ke které došlo. V případě, že požadavek byl korektně vyřízen, pošle server klientovi pozitivní odpověď (s kódem 200), jejíž součástí je i požadovaná stránka.

HTTP je plně textový protokol, tzn. že všechny HTTP zprávy mohou obsahovat pouze sedmibitové ASCII znaky. Veškerá binární data, která obsahují i osmibitové znaky je nutné před odesláním zakódovat pomocí algoritmu Base64.

HTTP je bezstavový protokol, což znamená, že server si neudrží žádný kontext o svých klientech, kteří k němu v minulosti přistupovali. Klient tedy musí posílat každý svůj požadavek zcela samostatně, bez závislosti na jiných požadavcích. Mimo jiné to znamená, že pokud server vyžaduje autentizaci pro přístup k určitým stránkám, musí se příslušné autentizační údaje posílat zvlášť s každou žádostí o chráněnou stránku.

2.3.1 Autentizace v HTTP

RFC 2617 [5] definuje dvě standardní autentizační metody pro HTTP – **Basic** a **Digest**. Metoda Basic je založena na posílání jména a hesla spolu s HTTP požadavkem. Server pak ověří tuto dvojici proti své databázi hesel a podle toho rozhodne o poskytnutí požadovaných informací. Základní nevýhodou tohoto mechanismu je posílání nešifrovaného hesla na server. Kdokoliv kdo má přístup ke komunikaci mezi klientem a serverem může snadno odposlechnout tyto autentizační údaje a později je zneužít tak, že se bude vydávat za uživatele, jehož heslo bylo takhle zachyceno. Vzhledem k těmto rizikům se tato metoda používá pouze v prostředí, kde je bezpečná síť, chráněná proti odposlechu. Lze jej také použít společně s mechanismem SSL (viz 2.1), který se používá pro šifrování přenášených dat. Protokol HTTP provozovaný nad SSL se označuje jako **https**.

Druhou definovanou metodou je Digest. Narozdíl od metody Basic ji lze bezpečně použít i nad nešifrovanými linkami, protože údaje posílané spolu s požadavkem nelze snadno zneužít. Metoda Digest je založena na principu jednocest-

ných funkcí a na faktu, že z výsledku jednocestné funkce je velmi obtížné zjistit původní argument. Mechanismus Digest je příkladem tzv. *Challenge–response* modelu, kdy server nejprve klientovi pošle náhodně vygenerované číslo, které klient spojí se svým heslem a na tento řetězec aplikuje jednocestnou funkci (zpravidla MD5 algoritmus), výsledek pak pošle zpět serveru jako svou odpověď. Server, který zná uživatelské heslo provede stejnou proceduru jako klient a porovná výsledek s klientskou odpovědí. V případě, že obě hodnoty jsou totožné, server poskytne klientovi požadovanou stránku.

Kapitola 3

Programování bezpečnostních aplikací

Vývoj aplikací, které řeší bezpečnostní problematiku je velmi náročný na implementační fázi, kdy je třeba velkou pozornost věnovat i nejmenším maličkostem, jejichž opomenutí by v budoucnosti mohlo způsobit vážné bezpečnostní problémy. Nedílnou součástí vývoje aplikací je proto zvolení vhodného programátorského rozhraní, které by mělo být dostatečně jednoduché, aby umožnilo tvorbu přehledného kódu. Na druhé straně je ovšem třeba pamatovat na požadavek maximální možné pružnosti aplikací a zvážit problémy, které by mohly nastat při převádění aplikace do prostředí s jinými bezpečnostními mechanismy, než pro jaké aplikace původně vznikala.

Každý vyspělý bezpečnostní mechanismus (např. Kerberos) má vlastní rozhraní, pomocí kterého může programátor psát aplikace, které využívají daný mechanismus např. pro autentizaci nebo šifrování dat. Takový přístup funguje dokud je aplikace používána v jednotném prostředí, pro které byla vytvořena. Ovšem v okamžiku, kdy přijde požadavek na přenos aplikace na jiný bezpečnostní mechanismus je zpravidla nutné všechny části, které byly napsány pro původní mechanismus vyměnit za nový kód. Takové změny jsou samozřejmě náročné a navíc zvyšují riziko vzniku chyb. Proto vznikají obecná rozhraní, která nabízejí realizaci bezpečnostních funkcí bez specifikací konkrétních autentizačních, či šifrovacích mechanismů.

3.1 Generic Security Service Application Program Interface (GSS-API)

Služby GSS-API jsou definovány v [6], mapování GSS-API na jazyk C lze nalézt v [7]. Zjednodušená ukázka části kódu serveru napsaného pomocí tohoto rozhraní je na obr. 3.1 (příklad byl převzat z [7]).

Jak je vidět z příkladu, autentizace pomocí GSS-API je založena na výměně *gss tokenů* mezi klientem a serverem. V těchto tokenech se přenášejí data konkrétního autentizačního mechanismu, jejich obsah a význam je však pro aplikaci neviditelný. Aplikace pouze musí zajistit přenesení tokenů na druhou stranu. Podobně i obsah *gss kontextu*, který si obě aplikace musí udržet v průběhu celé

```

gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;
do {
    receive_token_from_peer(input_token);
    maj_stat = gss_accept_sec_context(&min_stat,
                                     &context_hdl,
                                     cred_hdl,
                                     input_token,
                                     input_bindings,
                                     &client_name,
                                     &mech_type,
                                     output_token,
                                     &ret_flags,
                                     &time_rec,
                                     &deleg_cred);

    if (output_token->length != 0) {
        send_token_to_peer(output_token);
        gss_release_buffer(&min_stat, output_token);
    };
    if (GSS_ERROR(maj_stat)) {
        report_error(maj_stat, min_stat);
        if (context_hdl != GSS_C_NO_CONTEXT)
            gss_delete_sec_context(&min_stat,
                                   &context_hdl,
                                   GSS_C_NO_BUFFER);

        break;
    };
} while (maj_stat & GSS_S_CONTINUE_NEEDED);

```

Obrázek 3.1: Fragment kódu serveru

vzájemné komunikace je pro aplikaci neviditelný. Jak je vidět, GSS-API se nezaobírá vlastní komunikací mezi klientem a serverem. Je zcela na zodpovědnosti aplikace jakým způsobem přepraví token od klienta k serveru a opačně.

Pro nastavení parametrů autentizace má klient i server k dispozici řadu parametrů (např. klient může specifikovat, že vyžaduje i autentizaci serveru nebo nějakou specifickou ochranu přenášených dat). Tyto parametry mají vliv jak na délku přenášených zpráv, tak i na počet výměn, který je potřeba k provedení autentizace.

Podobně jako pro navázání autentizovaného spojení nabízí GSS-API funkce pro šifrování a dešifrování dat. Tyto funkce se jmenují `gss_wrap()` a `gss_unwrap()` a jsou opět nezávislé na použitém mechanismu nebo šifrovacím algoritmu.

Použití GSS-API je složitější jak na pochopení, tak na prvotní implementaci, ovšem u aplikací, u kterých se předpokládá, že budou používány v různých

prostředích se vyplatí investovat úsilí do použití tohoto rozhraní. Změna bezpečnostního mechanismu pak znamená pouze přelinkování aplikace s jinou gssapi knihovnou. Není potřeba žádná změna ve zdrojovém kódu aplikace.

3.1.1 Pseudomechanismus SPNEGO

GSS-API může být používáno nad velkou škálou autentizačních mechanismů. Vzhledem ke své obecnosti si musí klient a server nejprve nějak domluvit, jaké bezpečnostní mechanismy podporují (přes GSS-API rozhraní). V uzavřené skupině se to zpravidla řeší tak, že klientské aplikace (či spíše jejich uživatelé) „vědí“, že server podporuje určitý mechanismus a žádné domlouvání není potřeba. V rozsáhlejší prostředí však již je nutné, aby se aplikace před započítím vlastní autentizace domluvili, které mechanismy podporují a který z nich při své další komunikaci budou používat. K tomuto účelu vznikl mechanismus SPNEGO (*The Simple and Protected GSS-API Negotiation Mechanism*), jehož popis následuje ve zbývající části této podkapitoly.

Klient ve svém prvním tokenu nabídne jeden nebo více bezpečnostních mechanismů, které podporuje. Server buď akceptuje jeden z nabízených bezpečnostních mechanismů, případně pokud nepodporuje žádnou z nabízených možností, vrátí chybu a ukončí spojení. Server klientovi pošle token se specifikací vybrané metody. V této základní formě potřebuje protokol jednu výměnu navíc, aby určil používaný mechanismus, což je samozřejmě nevýhodné. Proto standard SPNEGO umožňuje, aby klient připojil inicializační token preferovaného mechanismu (tj. výsledek funkce `gss_init_sec_context()`, která implementuje tento preferovaný mechanismus), přímo do tokenu, kterým posílá serveru seznam svých metod. Po domluvení autentizačního mechanismu již probíhá komunikace tak, jak bylo naznačeno v kap. 3.1.

3.1.2 Příklad použití GSS-API

Na dokumentu [8] lze ukázat použití těchto obecných rozhraní v již zavedeném protokolu. Tento draft rozšiřuje specifikaci protokolu HTTP a zavádí nové autentizační schéma *Negotiate*, které využívá rozhraní GSS-API a mechanismus SPNEGO (oba pojmy byly popsány výše). Toto schéma je používáno aplikacemi MS Internet Information Services (IIS) a MS Internet Explorer (IE). Draft primárně předpokládá použití pro mechanismy Kerberos a Microsoft NTLM, ale vzhledem k obecným mechanismům, na kterých je založen, mělo by být snadné použít i jiné mechanismy.

Princip autentizace je podobný standardním mechanismům Basic a Digest. Pokud klient žádá o chráněnou stránku, server vrátí chybu 401, spolu s hlavičkou

```
WWW-Authenticate: Negotiate.
```

Klient pak opakuje svůj požadavek, ke kterému připojí autentizační hlavičku

```
Authorization: Negotiate EncBase64(output_token),
```

kde `output_token` je výsledek volání funkce `gss_init_sec_context()`. Funkce implementuje SPNEGO mechanismus tak, jak je popsáno v předchozí kapitole, tj. token obsahuje jak identifikaci použitého mechanismu (předpokládá se

Kerberos), tak první token z funkce `gss_init_sec_context()`, implementující mechanismus Kerberos.

Server přečte autentizační hlavičku, dekoduje parametr a zavolá funkci `gss_accept_sec_context()`, která zkontroluje, zda klient nabídl použití podporovaného mechanismu a v kladném případě zavolá odpovídající funkci konkrétního mechanismu `gss_accept_sec_context()`.

Kapitola 4

Zabezpečení WWW v projektu MetaCentrum

Cílem projektu *META Centrum* je vytvoření virtuálního počítače, který spojí výpočetní kapacity superpočítačových center v České republice do jednoho logického celku¹. Více informací o tomto projektu lze získat na <http://meta.cesnet.cz>. Uživatelé pak mají opravdu iluzi jednoho „počítače“. Základním autentizačním mechanismem v projektu *META Centrum* je Kerberos. Stručný popis mechanismu Kerberos lze nalézt v kapitole 2. Uživatelé mohou pomocí kerberovských lístků snadno používat většinu služeb, které souvisejí s chodem *META Centra*. Nicméně stále je několik oblastí, kde je vyžadována klasická autentizace pomocí jména a hesla. Jednou takovou oblastí je přístup do autentizovaného www prostoru *META Centra*. Cílem této bakalářské práce bylo vytvoření takových nástrojů, které uživatelům umožní, aby mohli své kerberovské lístky používat i pro přístup k autentizovanému webu.

4.1 Dosavadní metoda autentizace

V současnosti je autentizace na webech *META Centra* založena na faktu, že všechny rozšířené webové prohlížeče podporují autentizaci pomocí jména a hesla a zároveň podporují protokol SSL pro ochranu přenášených dat na server. Podrobnější informace o používaných autentizačních mechanismech lze najít v kapitole 2. V případě webových serverů *META Centra* se proto pro autentizaci používá autentizace typu **BASIC**, pomocí které uživatel předává serveru autentizační data potřebná pro získání kerberovského lístku (viz také kapitola 2). WWW server potom zkusí na základě těchto údajů získat lístek, jako by byl uživatel, který poslal tyto údaje. Tímto způsobem server ověří, že uživatel skutečně zná správné heslo pro získání příslušného lístku. Navíc server musí pomocí tohoto lístku získat lístek pro službu `khttp/<jméno-stroje>`, kde `<jméno-stroje>` je doménové jméno počítače, na kterém webový server běží (např. `scb.ics.muni.cz`). Takto získá server lístek „pro sebe“ a může pak porovnat tento lístek s příslušným šifrovacím klíčem, který má uložen na svém

¹V současné terminologii se používá pojem *grid*

disku. Tímto způsobem server ověří, že uživatelův lístek byl skutečně vydán správným KDC serverem, tj. že se nikdo nevydává za toto KDC a nevydává neoprávněně lístky.

Z předchozího popisu je zřejmé, že pro ověření jména a hesla je zapotřebí dvakrát kontaktovat KDC. Protože http je bezstavový protokol, musí být takové ověřování provedeno samostatně pro každou žádost klienta. Je zřejmé, že tento způsob je velmi náročný, protože k verifikaci uživatele je potřeba častá síťová komunikace mezi webovým serverem a KDC. S rostoucím počtem uživatelů pak roste zatížení serveru a prodlužuje se doba vyřizování klientských požadavků. Další nevýhodou a potenciálním bezpečnostním rizikem je to, že uživatel předává serveru veškeré své autentizační údaje, které jsou potřeba pro získání lístku. WWW server nebo jeho administrátor pak může zneužít tyto údaje pro padělání identity uživatele. Stejně v případě úspěšného průniku na webový server mohou být tyto údaje shromažďovány útočníkem, který je může později zneužít. Protože webové servery bývají častým cílem útoků, je toto nebezpečí velmi nepříjemné. Další nezanedbatelnou nevýhodou tohoto mechanismu, je to, že uživatel musí zadávat heslo při přístupu na chráněné stránky.

Naopak výhodou tohoto autentizačního mechanismu je to, že k webům *META Centra* lze přistupovat z libovolného prohlížeče, který podporuje autentizaci typu BASIC a šifrování pomocí SSL. Tyto vlastnosti ovšem mají téměř všechny webové prohlížeče, které se dnes používají.

4.2 Využití systému Kerberos pro HTTP

Problémy uvedené v předchozí kapitole by bylo možné elegantně vyřešit použitím Kerbera místo mechanismu BASIC. Bohužel současně používané prohlížeče nemají vestavěnou podporu jiných autentizačních mechanismů a vzhledem k tomu, že k většině těchto prohlížečů (tj. prohlížečů z rodiny Netscape nebo Internet Explorer) nejsou volně dostupné zdrojové texty, neexistuje jednoduchý způsob, jak k nim takovou novou autentizační metodu doplnit. V současné době se však situace zvolna mění a stále více uživatelů používá open-source prohlížeč Mozilla². Díky dostupným zdrojovým textům lze k tomuto prohlížeči daleko snadněji přidávat vlastnosti, které u něj uživatelům chybí.

Rovněž mechanismus Kerberos samotný se v posledních několika letech stále více prosazuje jako základ pro spolehlivé a bezpečné prostředí. Kerberos byl například zvolen jako základní autentizační systém pro operační systém Microsoft Windows 2000. Jako součást podpory Kerbera v tomto systému byla kerberovská autentizace přidána i do webových služeb, které jsou postaveny na tomto operačním systému. Autentizace založená na mechanismu Kerberos je tak podporována webovým prohlížečem Microsoft Internet Explorer a webovým serverem Internet Information Services. Nástin, jak je Kerberos použit v těchto komponentách byl zveřejněn v [8] a je blíže popsán v kapitole 3.

Vzhledem k plánovanému cíli této práce, tj. vytvoření nástrojů pro podporu kerberovské autentizace na webu, které budou použitelné co nejširší skupinou uživatelů a kvůli faktu, že prohlížeč Internet Explorer již implementuje a

²<http://www.mozilla.org>

(částečně) dokumentuje podporu tohoto mechanismu, jsem se rozhodl upravit webový server Apache a prohlížeč Mozilla tak, aby tyto komponenty implementovaly specifikaci [8].

4.3 Implementace návrhu

Pro implementaci jsem upravil server Apache verze 1.3.24³, který je nejrozšířenějším webovým serverem dneška a je rovněž použit na webech *META Centra*. Dále jsem upravil prohlížeč Mozilla verze 1.0rc1⁴. Jako implementace protokolu Kerberos jsem použil distribuci Heimdal verze 0.4e⁵, používanou na strojích *META Centra*. Pro kryptografické funkce jsem použil knihovnu OpenSSL verze 0.9.6d.

Vzhledem k tomu, že jsem neměl k dispozici implementaci mechanismu SPNEGO, použil jsem přímo implementaci nad Kerberem, která je součástí distribuce Heimdal. Pro přechod na SPNEGO mechanismus nejsou však potřeba žádné změny v kódu, stačilo by pouze slinkování všech komponent s odpovídajícími SPNEGO knihovnamí.

Zbývající část této kapitoly popisuje úpravy provedené v jednotlivých komponentách.

4.3.1 Apache

Apache umožňuje přidávat nové vlastnosti pomocí systému modulů, které exportují funkce implementující potřebnou novou funkcionalitu. Tyto funkce se volají z hlavního těla serveru během zpracování klientského požadavku. Tímto způsobem lze snadno rozšiřovat schopnosti Apache, bez nutnosti měnit hlavní kód aplikace. Vzhledem k stabilnímu rozhraní, kterým Apache s těmito moduly komunikuje a také nezávislosti modulů na hlavním kódu serveru lze snadno přenášet moduly mezi různými verzemi a konfiguracemi Apache. Pomocí modulu je implementována i současná kerberovská autentizace, využívající jméno a heslo.

Z pohledu autentizace je u modulu nejdůležitější funkce `check_user_id()`, která dostává jako parametr strukturu specifikující klientský požadavek a na základě těchto informací se pokusí klienta autentizovat. Pro implementaci draftu [8] jsem vytvořil modul `mod_auth_gssapi`, který ve funkci `check_user_id()` provádí kerberovskou, resp. GSSAPI autentizaci. Tato funkce nejprve ověří, že stránka požadovaná klientem vyžaduje kerberovskou autentizaci. V kladném případě hledá v požadavku uživatele hlavičku

```
Authorization: Negotiate 89a8742aa8729a8b028
```

Pokud taková hlavička není nalezena, vynutí modul vrácení chyby 401 (tj. Authorization Required), spolu s hlavičkou

```
WWW-Authenticate: Negotiate,
```

tak jak specifikuje draft [8]. Pokud je však hlavička

³<http://www.apache.org>

⁴<http://www.mozilla.org>

⁵<http://www.pdc.kth.se/heimdal/heimdal.html>

Authorization: Negotiate 89a8742aa8729a8b028
v klientském požadavku nalezena, dekoduje se předaný parametr pomocí algoritmu Base64 a výsledek se předá jako proměnná `input_token` pro volání funkce `gss_accept_sec_context()`. Pokud tato funkce vrátí OK, je klient akceptován a požadovaná stránka je mu poslána. V případě chyby je klientovi poslána chyba 401. V každém případě, pokud funkce `gss_accept_sec_context()` vrácí nenulový parametr `output_token`, jsou tato data zakódována pomocí Base64 a připojena k odpovědi v hlavičce

WWW-Authenticate: Negotiate B64(output_token)
Z důvodů kompatibility s klienty, kteří nerozumí autentizaci typu *Negotiate* vrací modul v případě chyby dvě hlavičky typu **WWW-Authenticate:**. Jednu s obsahem **Authorization: Negotiate** popsaným výše, druhou specifikující Basic metodu: **WWW-Authenticate: Basic realm="KRB5 Realm"**.

Modul `mod_auth_gssapi` zavádí dvě nové konfigurační volby, které mohou být specifikovány buď v globálním konfiguračním souboru Apache (v sekci `<Directory>`) nebo v souboru `.htaccess` v chráněném adresáři:

GssAuth Specifikuje, zda je pro daný adresář vyžadována kerberovská autentizace. Volbu `AuthType`, kterou Apache standardně používá pro tento účel nelze použít, protože Apache neumí kombinovat více těchto voleb pro jeden adresář a nebylo by tedy možné podporovat zároveň BASIC i GSS autentizaci

GssKrb5Keytab Specifikace souboru, který obsahuje kerberovský klíč pro službu `khttp/<jméno_stroje>`.

Z důvodů současné spolupráce obou autentizačních metod, tj. BASIC i GSS bylo nutné upravit i stávající modul `mod_auth_kerb` tak, aby v případě, že je vyžadována klientská autentizace vracel kromě hlavičky

WWW-Authenticate: Basic realm="KRB5 Realm",
kterou vracel dosud navíc i hlavičku

Authorization: Negotiate.

4.3.2 Mozilla

Celá aplikace je napsána v C++ a má přísně objektovou strukturu. Mozilla definuje abstraktní objekt `nsIHttpAuthenticator` (definici lze nalézt v adresáři `mozilla/network/protocol/http`), který slouží jako základ pro další objekty implementující konkrétní autentizační mechanismus. V současné verzi Mozilla implementuje oba standardní mechanismy, tj. Basic i Digest. Pro metodu *Negotiate* jsem vytvořil podobný objekt (`nsHttpGssapiAuth`). Další drobné úpravy bylo nutné provést i v jiných částech kódu, které se týkají autentizace. Např. Mozilla prohledává vnitřní cache s autentizačními údaji podle parametru `realm`, který dostane od serveru při použití obou standardních metod. Při použití metody *Negotiate* se však žádný takový parametr neposílá, proto bylo nutné změnit vyhledávání na základě požadovaného URL. Další změnou bylo zakázání zobrazování dialogu, který od uživatele požaduje jméno a heslo, protože pro Kerberovskou autentizaci se autentizační údaje získávají ze souboru na disku.

Dále bylo potřeba k hlavičkovému souboru `gssapi.h`, který je součástí Heimdalů doplnit direktivu `extern "C"` tak, aby jej bylo možné používat i z C++ programů.

Vzhledem k tomu, že při zpracování korektní odpovědi od serveru (tj. s kódem 200) používá Mozilla pouze omezený počet hlaviček, bylo by velmi obtížné zpracovávat `WWW-Authenticate`: hlavičky, které se v korektních odpovědích ignorují. Z toho důvodu je použita pouze jednodušší varianta modelu GSS-API, kdy se autentizuje pouze klient serveru a není vyžadována opačná verifikace. Stačí tak pouze jedno volání funkce `gss_init_sec_context()` a není tudíž nutné držet GSS kontext jako v jiných GSS aplikacích.

Zde je také vhodné připomenout, že všechny zmíněné autentizační metody, včetně *Negotiate* poskytují pouze autentizaci, nikoliv šifrování, či kontrolu integrity posílaných zpráv. Je proto potřeba doplnit tyto metody např. mechanismem SSL, který zajistí šifrování přenášených dat.

4.3.3 Příklad autentizované komunikace

V této kapitole uvádím, jak vypadá HTTP komunikace mezi upraveným klientem Mozilla upraveným serverem Apache. Výpisy byly pořízeny pomocí programu *Ethereal*⁶.

1. Klient žádá o chráněnou stránku `index.html`.

```
GET /index.html HTTP/1.1
Host: localhost:80
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0rc1)
           Gecko/20020527
Accept: text/xml, application/xml, application/xhtml+xml,
text/html;q=0.9, image/png, image/jpeg, image/gif;q=0.2,
text/plain;q=0.8, text/css, */*;q=0.1
Accept-Language: cs, en-us;q=0.50
Accept-Encoding: gzip, deflate, compress, identity
Accept-Charset: ISO-8859-2, utf-8;q=0.66, */*;q=0.66
Keep-Alive: 300
Connection: keep-alive
```

2. Server ze své konfigurace zjistí, že přístup ke stránce musí být autentizovaný a vrací chybu 401 – Unauthorized. Nabízí buď standardní metodu Basic nebo novou Negotiate.

```
HTTP/1.1 401 Authorization Required
Date: Mon, 27 May 2002 21:14:39 GMT
Server: Apache/1.3.24 (Unix) (Red-Hat/Linux)
WWW-Authenticate: Basic realm="KRB5 Realm"
WWW-Authenticate: Negotiate
Keep-Alive: timeout=15, max=100
```

⁶<http://www.ethereal.org>

```
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
\r\n
Data(409 bytes)
```

3. Klient si po obdržení chybové zprávy od serveru vyžádá a pošle znovu žádost o stejnou stránku, ke které přidá získaný lístek.

```
GET /index.html HTTP/1.0
Host: localhost:80
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0rc1)
          Gecko/20020527
Accept: text/xml, application/xml, application/xhtml+xml,
text/html;q=0.9, image/png, image/jpeg, image/gif;q=0.2,
text/plain;q=0.8, text/css, */*;q=0.1
Accept-Language: cs, en-us;q=0.50
Accept-Encoding: gzip, deflate, compress, identity
Accept-Charset: ISO-8859-2, utf-8;q=0.66, */*;q=0.66
Keep-Alive: 300
Connection: keep-alive
Authorization: Negotiate YIICDgYJKoZIhvcSAQICAQBuggH9MIIB+aADAg
EFoQMCAQ6iBwMFAAAAAACjggEdYYIBGTCCARWgAwIBBaEJGwdIT01FLkNaoiEwH
6ADAgEBoRgwFhsFa2h0dHAbDWthcmVsLmhvbWUuY3qjgd8wgdygAwIBEKEDAgEB
ooHPBIHM/0eEmG5Xe/0J010Eji50/ayDa1TwhvB1SgrCxy334jakDVFiQdmWr/J
83i/wvoQ04cnnQjg3z8Fw2VfP4oFrnSo7x7NGUmn0qPe6c+PSxDjb62F8vfp/y2
erJ266GRUspfeWLzLhHSttNp1ouIz5EhtPg375bQQYdJ0d8tdMSuah2Vxa1VMYy
sqkxaeLe0JFPALY3v8pYb4QXkg1jv4o7z62+qflgPHVb1Rry6sy9zAmosSYmJt+
208wuTA+MD11M+eSmU+Uze18+6N0pIHCMIG/oAMCARCigbcEgbQ0R4LKKpPHAqL
8WNTmcPy9yCwVhqPaBE6pU07tHbNbZ1qLQKkt3YiPHfzLehWD8YVryUpXQecUK2
RTwky1XxV4c70HcnytfRDxcA1ppDyaYft2Yv1NAHzEgOm/kMZ0yLVIGHWTpt/SD
e96YHT4QR1BqpviQqo2ew2wWftNTG7Iz3mmAAAnEbIvYqKsC4cf0tdavZtXwtXI/
69wk9mjV0gHTzkrmB9qEA0F+3Gk0ZbU57gFXZrA=
```

4. Server po úspěšném ověření lístku vrací kladnou odpověď, společně s požadovanou stránkou.

```
HTTP/1.1 200 OK
Date: Mon, 27 May 2002 21:22:02 GMT
Server: Apache/1.3.24 (Unix) (Red-Hat/Linux)
Last-Modified: Sat, 20 Apr 2002 16:32:12 GMT
ETag: "3c8a1-637-3cc1980c"
Accept-Ranges: bytes
Content-Length: 1591
Connection: close
```

Content-Type: text/html

\r\n

Data (1591 bytes)

5. Při dalších dotazech si již prohlížeč pamatuje, že server vyžadoval autorizaci, a proto posílá verifikační údaje přímo v první žádosti o stránku, tj. body 1. a 2. jsou přeskočeny.

Kapitola 5

Závěr

Během své práce jsem upravil www server Apache tak, aby podporoval plnohodnotnou kerberovskou autentizaci v protokolu HTTP. Analogické úpravy jsem provedl v open-source webovém prohlížeči Mozilla, který se v poslední době stává stále rozšířenější. Výsledkem mé práce je tedy prostředí, kde uživatelé mohou pro přístup k chráněným webovým stránkám používat své kerberovské lístky a nejsou nuceni do prohlížeče zadávat uživatelské jméno a heslo.

Rozšíření bylo implementováno podle specifikace od firmy Microsoft, ve které je popsáno, jak je Kerberos použit v webových aplikacích této firmy. Mělo by tedy být snadné používat silnou kerberovskou autentizaci i společně s těmito produkty.

Myslím, že používání systému Kerberos dává uživateli poměrně slušnou jistotu, že jeho data při komunikaci na Internetu nebudou zneužita. Samozřejmě, každý systém na ochranu dat se dá obejít, takže používání lepších metod ochrany pouze zvyšuje stupeň bezpečnosti komunikace, ale je rozhodně výhodnější udělat více než méně.

Příloha A

Obsah příloženého CD

V této příloze je zobrazen README soubor, který obsahuje návod na instalaci a konfiguraci jednotlivých komponent. CD obsahuje jednak zdrojové texty, jednak zkompilevané verze Mozilly pro Linux a Windows 2000 a verzi Apache pro Linux.

Předpokládám, že obsah tohoto CD je přístupný v adresáři /cdrom, Heimdal je nainstalován v /usr/heimdal, OpenSSL v /usr/local/ssl.

Instalace

=====

Mozilla

```
vitriol# cd /usr/heimdal/include
vitriol# patch < /cdrom/src/mozilla/heimdal.patch
```

```
vitriol$ cd ~/software
vitriol$ tar xzf /cdrom/src/mozilla/mozilla-source-1.0.rc1.tar.gz
vitriol$ cd mozilla
vitriol$ patch -p1 < /cdrom/src/mozilla/mozilla.patch
vitriol$ ./configure --enable-crypto --disable-mailnews
                    --disable-tests --disable-debug
                    --enable-optimize --enable-strip
```

Apache

```
vitriol$ cd ~/software
vitriol$ tar xzf /cdrom/src/apache/apache_1.3.24.tar.gz
vitriol$ LDFLAGS="-L/usr/heimdal/lib -L/usr/local/ssl/lib"
LIBS="-lgssapi -lkrb5 -lasn1 -lroken -lcrypto -lresolv"
INCLUDES=-I/usr/heimdal/include
./configure --prefix=/usr/local/apache
            --add-module=/cdrom/src/apache/mod_auth_kerb.c
            --add-module=/cdrom/src/apache/mod_auth_gssapi.c
```

```
vitriol$ make
vitriol# make install
```

Konfigurace

=====

Apache

Ukázkový konfigurační soubor je v /cdrom/src/apache/httpd.conf

Do sekce <Directory> adresáře, do kterého má být GSS

autentizován přístup musí být přidány následující direktivy:

```
GssKrb5Keytab "/usr/local/apache/conf/krb5.keytab"
```

```
GssAuth On
```

V krb5.keytab musí být lístek pro uživatele khttp/<jmeno_stroje>, soubor musí být vlastněn uživatelem nobody a čitelný pouze pro tohoto uživatele.

Po konfiguraci spusťte apache server:

```
vitriol# /usr/local/apache/bin/apachectl start
```

Mozilla

```
vitriol$ kinit
```

```
vitriol$ mozilla/mozilla
```

Ladicí výpisy lze zapnout pomocí proměnných NSPR_LOG_MODULES a NSPR_LOG_FILE (popis lze nalézt v mozilla/nsprpub/pr/include), např:

```
vitriol$ export NSPR_LOG_MODULES=nsHttp:5
```

```
vitriol$ export NSPR_LOG_FILE=/tmp/moz.log
```

Literatura

- [1] W. Stallings, *Network and Internetwork Security: Principles and Practice*, 1995.
- [2] J. Kohl, C. Neuman, *The Kerberos Network Authentication Service (V5)*, IETF RFC 1510, September 1993.
- [3] T. Dierks, C. Allen, *The TLS Protocol – Version 1.0*, IETF RFC 2246, January 1999.
- [4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, IETF RFC 2616, June 1999.
- [5] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, *HTTP Authentication: Basic and Digest Access Authentication*, IETF RFC 2617, June 1999.
- [6] J. Linn, *Generic Security Service Application Program Interface, Version 2, Update 1*, IETF RFC 2743, January 2000.
- [7] J. Wray, *Generic Security Service API Version 2 : C-bindings*, IETF RFC 2744, January 2000.
- [8] J. Brezak, Microsoft, *HTTP Authentication: SPNEGO Access Authentication As implemented in Microsoft Windows 2000*, draft-brezak-spnego-http-03.txt, February 2002, Internet draft, Kerberos WG, Work in progress.