СУ “Св. Климент Охридски”

Факултет по математика и информатика

Катедра “Софтуерни технологии”

Д И П Л О М Н А Р А Б О Т А

на тема

Използване на протокола Керберос за автентикация на потребителите при сървъри за приложения

Дипломант**:** Георги Димитров Димитров, фак. № M21299

Специалност**:** Информатика

Специализация**:** Разпределени системи и мобилни технологии

Научен ръководител**:** доц. д-р Александър Григоров

София

2007

Съдържание

1. Увод

1.1. Въведение

Автентикацията [1](от гръцки *αυθεντικός* = истински) е действието, при което се доказва, че нещо (или някой) е истинско, или по-точно е същото, за което се представя. Автентикация на обект се използва в смисъл на доказателство за неговия произход, докато автентикация на човек означава доказателство за неговата самоличност. В компютърните системи автентикацията е процесът на проверка на идентичността на обект, който се опитва да се представи пред компютърна система (сървър). Важността на автентикацията в компютърните системи е голяма, тъй като от правилния изход на този процес, зависят правата, с които в последствие ще разполагат потребителите.

Традиционните методи за автентикация често не са подходящи за употреба в широки компютърни мрежи, където има възможност трафика да бъде следен от недоброжелатели с цел откриване на чужди пароли (или други идентификационни средства). Това налага все по широката употреба на криптографски алгоритми и протоколи в процеса на автентикация.

Протоколът Керберос[2][3] е разпределена услуга за автентикация, която позволява на даден потребител да докаже своята идентичност пред сървъра, който иска да ползва, без да се пращат данни по мрежата, които могат да позволят на недоброжелател да се представя за него след това. Затова протоколът Керберос е един от най-разпространените и използвани методи за автентикация. Интегриран е в операционни системи като WINDOWS и LINUX, а също и в други операционни системи базирани на UNIX. Поради широкото му разпространение на ниско ниво, отварянето на протокола към различни криптографски алгоритми го прави привлекателен за използване и от много софтуерни продукти, работещи върху тези операционни системи.

1.2. Цел и задачи на дипломната работа

Дипломната работа представя използване съвместно на механизмите SPNego[8][10] и Керберос за автентикация на Web клиенти пред сървъри за приложения. В дипломната работа се разглеждат следните задачи:

Задача1 Изследване на характеристиките на протокола Керберос и подходите, чрез които този протокол решава основните проблеми за гарантирането на сигурна автентикация на потребителите.

Задача 2 Изследване на стандарта GSSAPI[4][5] за автентикация и защитена комуникация и спецификацията JAAS[7] в частта свързана с автентикация на потребителите.

Задача 3 Изследване на протокола SPNego за автентикация на потребителите.

Задача 4 Реализиране на модул за автентикация от спецификацията JAAS, който да използва протокола SPNego.

Задача 5 Реализиране на опростено Web приложение, чрез което да се тества модула за автентикация от точка 4. Модулът трябва автоматично да предоставя идентичността на потребителите от даден домейн, при заявка към тестовото приложение. При стартиране на тестовото приложение се извежда името на потребителя, с което той(потребителят) е автентициран за работа в домейна.

Задача 6 Да се изследват възможностите за прилагане на този подход за автентикация при различните версии на JDK, видовете WEB клиенти и други. Да се изследват критичните случаи при използването на въпросния механизъм за автентикация и подходи за разрешаването им.

Ограничение: За реализацията трябва да се използва сървър за приложения предоставящ възможност за използване на горе споменатите стандарти и спецификации. Модулът за автентикация и тестовото приложение трябва могат да се интегрират към съответната система и затова ще ползват някои функционалности специфични за дадения сървър, но те не трябва да са определящи за разработеното решение.

1.3. Структура на дипломната работа

Дипломната работа съдържа 11 глави. Главата „Увод” представя накратко съдържанието на дипломнатата работа. Следващите глави са:

- Протокол Керберос
- Стандартен интерфейс GSSAPI
- Протокол SPNego.

Те представят основните характеристики на въпросните технологии и стандарти, на които се базира реализацията на програмното задание. С повече подробности се разглеждат онези части, които конкретно се използват в програмната част на реализацията, докато някои части са

умишлено пропуснати или споменати накратко, с цел да се фокусираме основно върху проблемите свързани с решаването на поставените задачи. Следващите глави представят програмната част от дипломното задание. Главите са:

- Реализация
- Конфигуриране и тестване
- Проблеми и насоки за развитие

Те показват как на практика се използват разглежданите технологии. Описва се архитектурата на решението и се показва в общия случай и конкретно с пример как се използва то. Разглеждат се проблемите свързани с употребата на конкретното решение, както и по-общи проблеми породени от използваните технологии.

Следва главата „Заключение”, в която се прави равносметка за свършената работа. Дипломната работа завършва с главите:

- Използвана термини и съкращения
- Използвана литература
- Приложение

Тук се предоставя допълнителна и помощна информация свързана с процеса на работа върху разглежданите задачи.

2. Протоколът Керберос

Протоколът Керберос[2] е разработен през 80-те години в САЩ в Масачузетския технологичен институт (MIT). Първоначално Керберос е бил част от проекта Атина, като целта му е била да позволи потребители и услуги да се идентифицират един пред друг. Наречен е на името на триглавото куче от древногръцката митологията, което охранявало входа на подземния свят Хадес. Предполага се, че името Керберос (и по-точно символът куче с 3 глави) е избрано заради факта, че Керберос протоколът ясно дефинира 3 участника в комуникацията:

1.Потребител

2.Сървър за приложения

3.Специален сигурен трети сървър

Керберос използва за основа протокола Нийдам-шрьодер (Needham-Schroeder). Протоколът Керберос в момента е особено разпространен поради факта, че компанията Microsoft решава да използва този протокол като свой основен (по премълчаване) метод за автентикация за операционните системи Windows XP и Windows Server 2003. Като допълнение Mac OS X на Apple също така използва Керберос. Понастоящем се използва версия 5 на Керберос.

2.1 Предпоставки за възникването и характеристики на протокола Керберос

2.1.1 Условия за гарантиране на сигурността

Автентикационните протоколи представляват последователността от необходими действия, за да протече правилно автентикацията на клиента пред сървъра. Сигурността на тези протоколи зависи от гарантирането на следните условия:

- **Конфиденциалност на информацията** (Confidentiality) – предаването на данни между клиента и сървъра трябва да бъде осъществено, така че ако се послушва канала да не може да бъде откраднатa информация, която по-късно да бъде използвана за автентикация.
- **Интегритет на информацията** (Integrity) – това свойство, доказва че пристигналата информация на сървъра, е същата, която е изпратена от самият клиент, т.е. не е била променяна по трансферния канал. За тази цел обикновено се използват цифрови подписи,

които клиентът изпраща заедно със съобщението. При пристигане на съобщението, сървъра проверява дали съобщението отговаря на съпътстващия го цифров подпис.

- **Двустранна автентикация** (Mutual authentication) – освен че клиентът се представя пред сървъра, то и сървърът може да се идентифицира пред клиента. Например, ако се опитваме да заплатим някаква услуга или да закупим продукт по интернет, то трябва да знаем, че сървъра, на когото плащаме е наистина сървъра на тази компания, а не просто сървър, който се представя за такъв.
- **Невъзможност за отричане** (Non Repudiation) – това свойство доказва, че наистина автентикацията е минала, и клиентът или сървърът не може да отрече, че наистина тя е протекла. За тази цел отново се използват цифрови подписи. Например когато плащаме през интернет, ние искаме гаранция, че фирмата, от която от която закупуваме даден продукт, няма да отрече, че той вече е платен.

2.1.2 Проблемът с транспортиране на пароли

Голяма част от несигурността при компютърните мрежи идва от факта, че се предават пароли по мрежата (в най-незащитения случай като чист текст). Това означава, че всеки, който подслушва мрежата може да се сдобие с паролата на даден потребител и да се представя за него. Дори кодиране на паролата, например чрез използване на хеш функции, не може да помогне много

2.1.3 Нуждата от еднократна автентикация

Разрастването на големият брой системи за различни продуктивни нужди, довежда потребители и системни администратори пред проблема за поддръжка на съответния голям брой потребителски имена и пароли (или друг тип автентикационни данни) за всеки един потребител. Например даден потребител може да притежава няколко регистрации - с клиентски номер и някаква парола за някаква организация (например за плащане на сметки през Интернет), с някакво измислено име и парола за сайт за електронна поща и регистрация с ЕГН и някакъв сертификат за някаква банка например. Осигуряването на коректен достъп до тези системи понякога може да бъде досадна операция, когато при всяка една операция върху някоя от тези системи трябва да се предоставят автентикационните данни.

Еднократната автентикация (Single Sign On -SSO) предлага следното решение за този тип проблем - да се използва допълнително приложение/сървър „Доверител”, което да осъществява

сигурна връзка с различните приложения. Всяка една от системите, които ще достъпва потребителят трябва да имат доверие на този „Доверител”. Това означава, че когато „Доверителят” поиска да се изпълни някаква операция от името на потребителя, отделните системи няма да изискват допълнителни операции за потвърждаване на идентичността и ще изпълнят заявката.

2.1 Основни характеристики на автентикацията с Керберос

Протоколът Керберос предоставя възможност за автентикацията на потребителите в компютърни мрежи като позволява потребителите да предоставят своята идентичност по сигурен начин. Тази технологията предотвратява възможностите за подслушване на съобщенията и опитите за неправомерно повторното използване на прихванатите валидни съобщения. Особеност на Керберос протокола е необходимостта от използването на доверен трети участник в комуникацията. Като основни цели на протокола може да определим:

1. Да не се транспортират пароли (криптирани и некриптирани) по мрежата.
2. Защита от злоупотреба с подслушани и откраднати съобщения по мрежата (replay attack)
3. Да не се изисква въвеждане на потребителска парола за всякакви рутинни операции и услуги (поддръжка на SSO).

В архитектурата на протокола е залегнала идеята за работа с централизиран сървър, който всички компютри в мрежата приемат за сигурен и към него се пренасочват всички заявки за автентикация. Керберос предоставя и двустранна автентикация, която не само обезпечава, че потребителя зад клавиатурата (или клиентското приложение) е този за когото се представя, но също така доказва, че сървърът, с който той комуникира, е истински. Протоколът Керберос поддържа еднократна автентикация. Това означава, че веднъж след като е автентициран потребителя, неговите автентикационни данни се предават автоматично за всеки ресурс, който иска да ползва.

2.2 Основни понятия и реализация на Керберос

2.2.1 Сървър за Автентикация

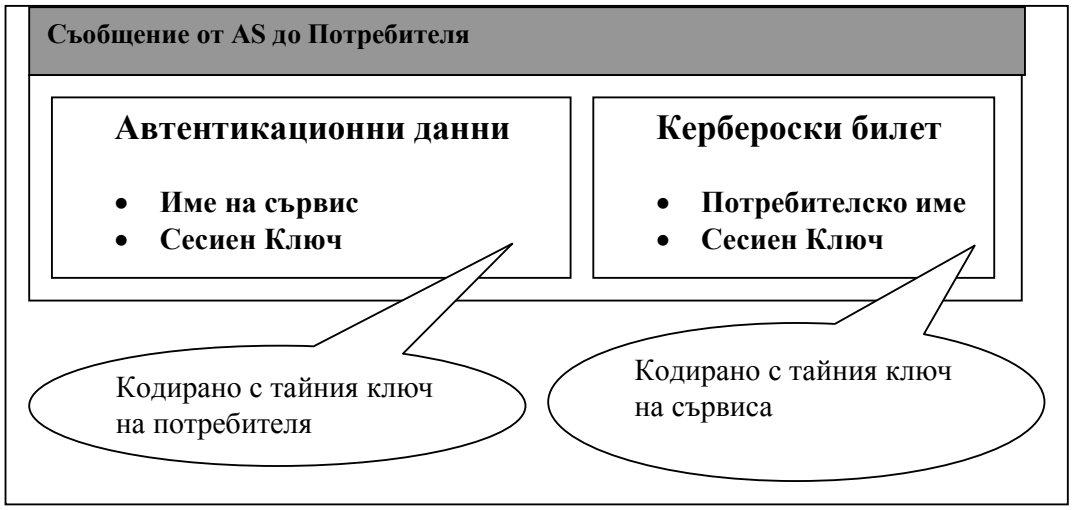
Протоколът Керберос[3] използва специален сървис(сървър) наречен Сървър за Автентикация (Authentication Server - AS). Идеята е потребителите и сървисите по мрежата да имат доверие на този Сървър за Автентикация, когато се идентифицират помежду си. И потребителите и сървисите трябва да са регистрирани на този Сървър за Автентикация (по-точно на сървърът се съхраняват споделени тайни ключове). Когато даден потребител се автентицира пред даден сървис се изпълнява следната процедура:

Фаза 1 Потребителят изпраща заявка до AS, която съдържа името на сървиса, с които ще се осъществява комуникация. Всеки може да изпрати такава заявка без проверка за идентичността.

Фаза 2 AS генерира ключ наречен **сесиен ключ(session key)**, който ще се използва по-нататък от потребителя и сървиса. AS генерира съобщение от 2 части:

- Първата част съдържа сесийния ключ и името на сървиса криптирани с тайния код на потребителя. Това представляват **автентикационите данни** (**credentials**) за потребителя.
- Втората част съдържа същия сесиен ключ и името на потребителя кодирани с тайния код на сървиса. Това представлява **Керберoския Билет (ticket).**

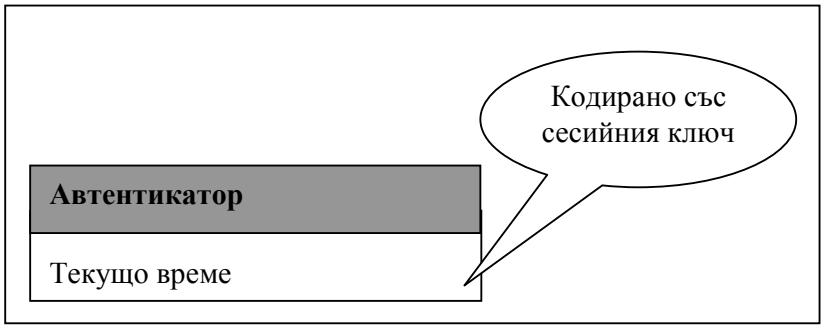
Това съобщение се изпраща към потребителя. Представено е на фигура 1.



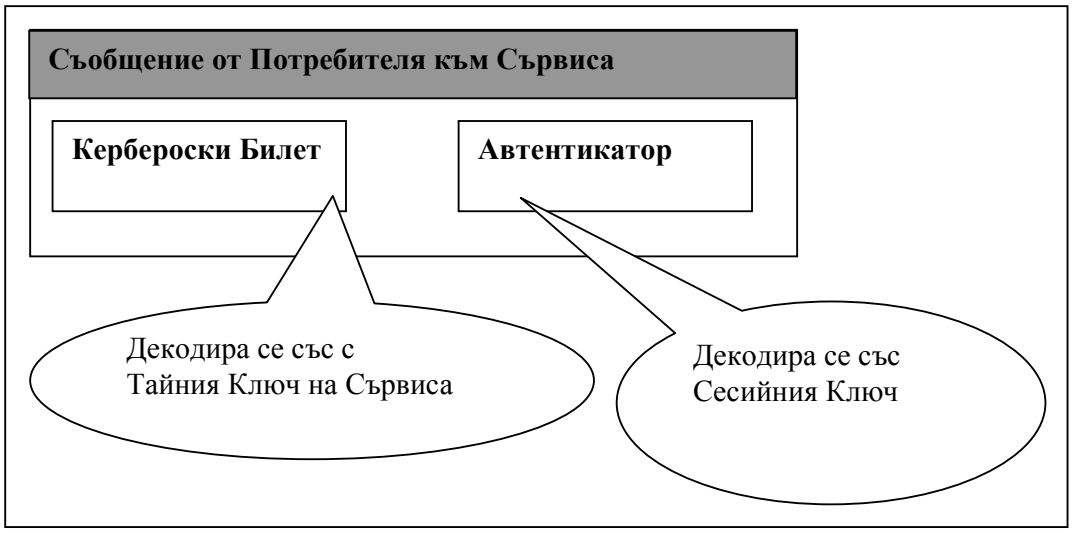
Фигура 1: Съобщение от Сървъра за Автентикация до Потребителя

Фаза 3 Потребителят получава съобщението и изпълнява следните стъпки:

1. Декодира частта Автентикационни Данни като използва своя таен ключ. Така разбира сесийният ключ за комуникация със сървиса. В този момент само потребителят знае сесийния ключ. Никой друг не може да декодира „Автентикационните Данни", защото само потребителят знае тайният ключ, с който е извършено кодирането.
2. Потребителят генерира ново съобщение базирано на точния час (timestamp) и го кодира със сесийния ключ. Това съобщение се нарича **"автентикатор".**
3. Потребителят изпраща Кербероския билет и Автентикатора към сървиса.



Фигура 2 Автентикатор



Фигура 3 Съобщение от Потребителя към Сървиса

Фаза 4 Сървисът получава съобщението от потребителя и изпълнява следните стъпки:

1.Декодира билета с неговия (на сървиса) таен ключ. Така и той разбира сесийния ключ.

2.След това сървисът използва сесийния ключ да декодира автентикатора. Понеже сървисът има доверие на Сървърът за Автентикация, той (сървисът) знае, че само

истинският потребител може да генерира такъв автентикатор. С това завършва автентикацията на потребителя пред сървиса.

Взаимна автентикация

Понякога е нужно сървисът също да се автентицира пред потребителя. Тогава сървисът изпраща автентикатора модифициран по някакъв определен начин (например като добави своето име) закодиран със сесийния ключ.

2.2.2 Сървър за издаване на билети

Едно от неудобствата, когато се използват пароли е, че всеки път, когато се достъпва даден сървис трябва да се въвежда някаква парола. Това може да бъде много досадно и като резултат потребителят може да започне да използва навсякъде една и съща парола. И освен това да направи тази парола лесна за писане и съответно за отгатване. Керберос елиминира използването на много пароли, но все още остава рискът от използване на лесна парола. Тук все още остава възможността някой да отгатне паролата, въпреки че тя се използва като споделена тайна, а не като съобщение, което се праща по мрежата. Керберос решава този проблем като използва още един сървър.

Това е **"сървърът за издаване на билети"(TGS).** Идеята на TGS е потребителят да въвежда паролата само веднъж. Кербероският Билет и сесийният ключ, които се получават се използват за всички (Кербероски) билети след това. Т.е. преди да се достъпи който и да е сървис, потребителят иска Главен Кербероски Билет от AS за да може да говори с TGS. **Главният Кербероски Билет** представлява всъщност **Билет за достъп до други билети** (ticket granting ticket –TGT). Той осигурява достъпа до билети за другите сървиси, които реално се използват от потребителя. Понякога се нарича "начален билет". Сесийният ключ за TGT е криптиран с тайния ключ на потребителя. Паролата се използва за да се декодира отговора от AS. След като получи TGT, всеки път когато потребителят иска да достъпи даден сървис, той (потребителят) иска билет не от AS, а от TGS. След това отговорът е кодиран не с тайният ключ на потребителя, а със сесийният ключ, който потребителят е получил при получаването на TGT. Така потребителската парола не е нужна за получаването на нов сесиен ключ, с който ключ ще се комуникира със конкретния сървис. Този TGT както и другите билети, които биват използвани се кешират.

Опростен пример за подобен сценарий

Потребител (човек) посещава дадено учреждение, показва си личната карта и тогава му издават временен пропуск, с който може да се легитимира в случай на нужда. В тези случай се използва временният пропуск, който дори и да бъде загубен или откраднат може да бъде обявен за невалиден и веднага може да се издаде нов пропуск (а не нова лична карта).

2.2.3 Център за предоставяне на ключове

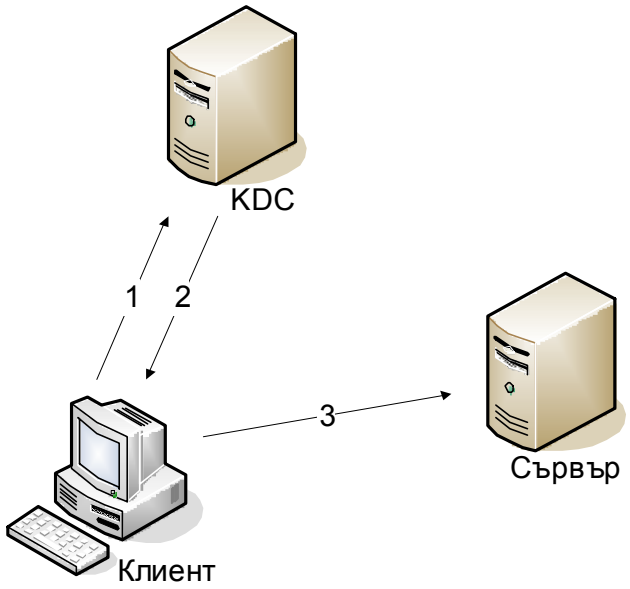
Керберос използва доверен трети участник(trusted third party), който има за цел да предоставя "ключове" (билети) за комуникация. Това е Центърът за предоставяне на ключове (Key Distribution Center - KDC). KDC логически се състои от 2 части:

1. Сървърът за автентикация (Authentication Server) и
2. Сървърът за предоставяне на билети (Ticket Granting Server).

Логически TGS и AS са отделени, но те могат и най-често се намират на една физическа машина. Затова най-често говорим за KDC и по-рядко конкретизираме, за кой от двата логически компонента конкретно става дума. KDC издава Керберос билетите използвани в комуникацията. Този сървър трябва да е достатъчно сигурен за да държи тайните ключове на всеки клиент и сървър в мрежата. Ключът, споделян с KDC формира основата, на която клиентът или сървърът приемат автентичността на билета, който получават. Керберoският билет е валиден за краен интервал от време. KDC съхранява архив от тайни ключове. Всички участници в компютърната мрежа (клиенти или сървъри) използват тайни ключове, които са известни само на самия собственик на ключа и на KDC. Който знае тайния ключ може да се представя за този, на когото е ключа. За да си комуникират два "потребителя" KDC им предоставя сесиен ключ(описан по-горе), посредством който те си гарантират сигурността на обмен на съобщенията между тях.

2.2.4 Първоначална размяна на билет

Фигура 4 показва необходимата последователност от стъпки и съобщения на клиента да докаже своята самоличност пред даден сървър. Клиентът използва тази размяна, когато за първи път се свързва със сървъра. Следващите връзки към същата система изискват само крайното съобщение от тази размяна (кеширането на билета при клиента елиминира необходимостта от първите 2 съобщения докато билетът е все още валиден).



1. Клиент → KDC: заявка за билет за Сървъра

2. KDC → Клиент: Автентикационни Данни, Керберoски Билет

3. Клиент → Сървър: Автентикатор, Керберoски Билет

Фигура 4: Издаване и използване на първоначалния билет.

В първото съобщение клиентът се свързва с KDC, идентифицира се с парола(хеш стойност на паролата), предоставя специален низ/маркер (съдържащ стойност за текущото време или някакъв друг уникален идентификатор на заявката). След получаването на съобщението, KDC изпраща автентикационните данни и билета, които са генерирани по описания по-горе механизъм. Билетът идентифициращ клиента, предоставя сесийния ключ, указва времето, за което е валиден и е кодиран с тайния ключ на сървъра. Тъй като билетът е кодиран с ключ, който е известен само на KDC и на сървъра, никой друг не може да го прочете или да промени самоличността на клиента указана в него. При получаването на отговора клиентът декодира автентикационните данни с тайният си ключ. След проверка на маркера (който евентуално може да е променен с някаква предварително зададена стойност), клиентът кешира билета и асоциирания сесиен ключ за по-нататъшно използване. В третото съобщение клиентът представя билета и току що генерирания Автентикатор пред сървъра. Както беше споменато, Автентикаторът съдържа текущото време и е кодиран със Сесийния Ключ. При получаването му Сървърът декодира билета, като използва своя таен ключ и извлича самоличността на клиента и Сесийния ключ. За да провери самоличността на клиента, Сървърът декодира Автентикатора (като използва Сесийния ключ от билета) и проверява дали отбелязаното в него

време е текущото. Успешната проверка на Автентикатора доказва, че клиентът притежава истинския Сесиен ключ, който може да бъде притежаван само след успешно декодиране на отговора от KDC. Тъй като отговорът е кодиран с тайния ключ на клиента, сървърът е основателно убеден, че самоличността му е тази, която е указана в полученият билет.Ако клиентът поиска обратна (взаимна) автентикация от сървъра, сървърът отговаря с ново съобщение, кодирано със сесийния ключ. Това удостоверява на клиента, че сървърът притежава Сесийния ключ, с който може да се сдобие единствено ако е способен да декодира билета. Тъй като билетът е кодиран с ключ, който е известен единствено на сървъра и KDC, този отговор удостоверява самоличността на сървъра.

2.2.5 Допълнителна размяна на билет

Основният Керберос протокол за автентикация позволява на клиент, който знае тайният ключ на потребителя, който в повечето случаи се базира на неговата парола, да се сдобие с билет за него, с който да се представя за този потребител пред всеки сървър, регистриран в AS. Процесът описан в предната точка трябва да се изпълнява всеки път, когато потребителят се автентицира пред нов сървър/сървис. Керберос поддържа еднократно идентифициране (SSO), като потребителят се автентицира веднъж, като предоставя име и парола и всяка следваща автентикация след това става автоматично. Очевидният начин за поддържането на този механизъм чрез кеширане на паролата може да бъде опасен. Въпреки че Керберос билетът и ключът асоцииран с него са валидни за кратък период от време, паролата на потребителя може да бъде използвана, за да се вземе билет и по този начин друг да се представя за него докато паролата не се промени. По-добрият начин, който се използва в Керберос, е да се кешират само билетите и кодиращите ключове, които ще са валидни за ограничен период от време.

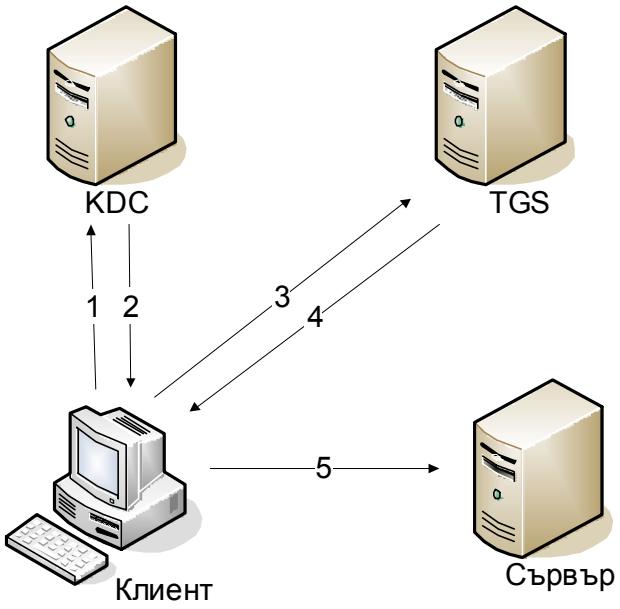
За да се намали риска от разобличаване на тайния ключ на клиента и за да се направи употребата на Керберос възможно по-невидима за него процесът описан в предната точка се използва главно за вземане на билет за TGS (Ticket Granting Server). Клиентът изтрива, копието на своя таен ключ след като се сдобие с билета (ticket granting ticket-TGT) за този сървър, наречен Главен билет.

Клиентът представя своя Главен билет (заедно с останалите данни) пред TGS, както би го представил пред всеки друг сървър, TGS проверява билета, автентикатора и придружаващата ги заявка и отговаря с билет за новия сървър. Защитената част от отговора е кодирана със сесийния ключ от Главния билет, така че на клиента не му е необходимо да помни оригиналния

си таен ключ, за да декодира и използва този отговор. Клиентът след това използва тези нови автентикационни данни за връзка с втория сървър.

След като автентикацията е установена, клиентът и сървърът споделят Сесиен ключ, който се праща по мрежата само в кодиран вид. Те могат да използват този ключ за предпазване на последвалата комуникацията между тях от разкриване или модификация. Керберос предоставя формати на съобщения, които могат да се ползват за гарантиране на цялостност и поверителност на съобщенията.

Раздаването на билети в Керберос протокола позволява на един потребител да се сдобие с билет и кодиращ сесиен ключ, като предостави краткосрочни удостоверения за самоличността си, без да е необходимо да се въвежда отново парола. Когато потребителят влезе първоначално в системата се извършва заявка за автентикация и като резултат се получават билет и кодиращ ключ за достъп до TGS. Този билет, наречен Главен билет има относително кратък живот (обикновено в рамките на 8 часа). След това, когато потребителят иска да се автентицира пред друг сървър се изисква нов билет от TGS, като се използва главния билет.



1. Клиент → KDC: заявка за TGT
2. KDC(AS) → Клиент: Автентикационни данни + TGT – съдържат Сесийния ключ за комуникацията Клиент-TGS
3. Клиент → TGS: заявка за билет за Сървъра : Автентикатор + TGT + маркер
4. TGS → Клиент: Автентикационни данни от TGS + Билет за Сървъра – съдържат Сесийния ключ за комуникацията Клиент-Сървър
5. Клиент → Сървър: Автентикатор + Билет за Сървъра

Фигура 5: Пълен Керберос протокол на автентикация.

Фигура 5 показва пълния Керберос протокол за автентикация. Съобщения 1 и 2 се използват само при първото влизане на потребителя в системата (когато потребителят си пусне компютъра и въведе паролата си за домейна), 3 и 4 се използват всеки път, когато потребителят трябва да се автентицира пред нов сървър (това става без допълнителни действия от Потребителя), а съобщение 5 се използва всеки път, когато потребителят трябва да се автентицира(например при нова заявка към съответния Сървър).

2.2.6 Керберос и използване на публични ключове

Тъй като един от механизмите, на които се базира Керберос е криптирането, то трябва да изясним за какво криптиране става дума. Керберос най-често използва само симетрично криптиране. При симетричното кодиране потребителят и сървърът трябва да знаят тайният ключ, на който се базира кодирането/разкодирането, т.е. потребителят трябва да е регистриран на KDC преди да започне да използва Керберос. Това изискване може да бъде избегнато ако се използват "публични ключове". Публичният ключ е общо достояние, а тайният ключ е достояние само на самия потребител.

Този подход може да бъде интегриран в Керберос по доста прост начин. Когато KDC (и по точно AS) генерират отговора за потребителя (който включва в себе си сесийния ключ за TGT) той няма да се криптира с тайния ключ на потребителя, а със специален ключ, който е криптиран с публичния ключ на потребителя.

Единствено потребителят може да разкодира това съобщение като използва собствения си таен ключ. Първо ще получи “случайния” ключ генериран от сървъра, а след това с него ще получи и сесийният ключ за TGT.

Но дори и да не е нужно споделянето на симетричен таен ключ, трябва да има някакво асоцииране на потребителите и публичните ключове. Иначе няма никаква гаранция че публичният ключ отговаря на някаква идентичност. Всеки би могъл да генерира публичен и таен ключ, да изпрати публичния на KDC и да се представи, за който и да е друг потребител. За да се избегне това публичните ключове се сертифицират от някакъв Гарант за сертификатите (certificate authority) - CA. Публичният ключ се подписва от този CA. CA кодира публичния ключ и идентичност с тайния ключ на самия CA. CA е някой(сървър), на когото се има доверие за тази операция. След това всеки, който иска да провери публичния ключ дали е истински, може да направи това като използва публичния ключ на CA. Предимството на подхода използващ симетричните ключове е, че не е нужно CA сървъри да се използват и да са работещи постоянно докато KDC автентицира потребителите.

3. Стандартен интерфейс GSSAPI

3.1 Основни характеристики

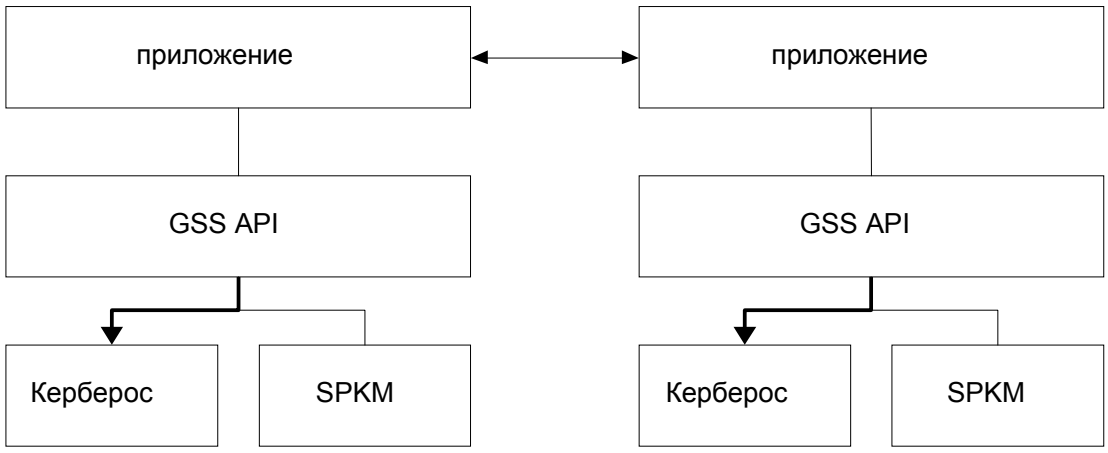
Приложенията често имат различни изисквания за сигурността и разчитат на множество технологии за да постигнат това. Основен проблем възниква, когато се предприеме смяна на една използвана технология с друга. За тази цел организацията IETF разработва стандарта GSSAPI (Generic Security Services Application Program Interface)[4], който предоставя унифициран интерфейс за автентикация и защитена комуникация, която изолира потребителя от ползваната технология. Този програмен интерфейс, обхваща следните услуги за сигурност:

- автентикация
- поверителност и целокупност на съобщенията
- последователност на предпазваните съобщения
- откриване на отговор
- делегация на автентикационни данни.

Технологиите за осигуряване на защита, т.н. "механизми за сигурност" имат възможността да избират да поддържат една или повече от тези услуги освен главната – автентикация, която задължително трябва да поддържат. За момента IETF е специфицирал два стандартни механизма за сигурност: протоколът Керберос (версия 5) и Механизъм с публични ключове(Simple Public Key Mechanism – SPKM). Програмният интерфейс GSSAPI е проектиран така, че да поддържа няколко механизма за автентикация едновременно, като дава възможност на приложението да избере един по време на изпълнението му. Всеки механизъм има свой уникален идентификатор Unique Object Identifier (OID) който е регистриран в IANA. Например протоколът Керберос 5 е с идентификатор **1.2.840.113554.1.2.2**. Този идентификатор всъщност представлява поредица от кодове:

{iso(1) member-body(2) Unitited States (840) mit(113554) infosys(1) gssapi(2) krb5(2)}.

Другата много важна характеристика на този програмен интерфейс е, че е базиран на низове. Резултатите от извикванията на интерфейса генерира неразбираеми низове, които приложението трябва да прати на отсрещната страна. Това позволява интерфейсът да е независим от транспорта. На фигура 6 схематично е показано как две приложения, които си комуникират посредством GSSAPI, се договарят да използват Керберос.



Фигура 6: избор на механизъм при реализацията на GSS-API

Съобщенията от GSSAPI могат да бъдат пращани без да се гарантира допълнителна сигурност, защото механизмът гарантира вродена сигурност на съобщенията. След като се обменят няколко съобщения до двете страни, приложението разбира, че е установена сигурна среда за комуникация (Security context). След това приложението вече може да обменя съобщенията като ги криптира/закодира посредством GSSAPI. Ограничение на GSSAPI е, че той стандартизира само автентикацията, но не и даването на права(authorization). Съществуват стандарти за GSSAPI за C и Java. C Java GSSAPI ще наричаме реализациите на GSSAPI от различните доставчици на JDK. Керберос 5 е задължителна част за всички реализации на Java GSSAPI, които се предоставят с J2SE, като те могат да поддържат и други протоколи за автентикация.

3.2 По-важни класове на Java GSSAPI

Тук ще разгледаме по-важните класове от реализацията на GSSAPI предоставяна от стандартните версии на JDK[5]. (Тези класове се използват в реализацията на предложената в дипломната работа библиотека SPNegoLib.)

3.2.1 Класът GSSManager

Този клас предоставя всички налични механизми за автентикация и е отговорен за създаването на инстанции от другите основни класове. Инстанция на обект от този клас се получава по следния начин:

```
GSSManager manager = GSSManager.getInstance();
```

3.2.2 Интерфейсът GSSName

Той представя съществуваща самоличност за целите на JavaGSS API. Обект от този тип се създава чрез следния метод:

```
GSSName GSSManager.createName(String name, Oid nameType) throws GSSException;
```

Например:

```
GSSName clientName = manager.createName("duke", GSSName.NT_USER_NAME);
```

Резултатът от изпълнението на този код ще доведе до създаването на GSSName представящо самоличността на потребителя "duke" във вид независим от механизма за защита. Например механизмът предоставящ Керберос версия 5 ще конвертира името до "duke@FOO.COM", където FOO.COM е локалната Керберос област(или Керберос реалм).

Когато самоличността, която се представя не е на потребител, а на сървис това трябва да се укаже при създаването на GSSName:

```
GSSName serverName =
manager.createName("nfs@bar.foo.com",GSSName.NT_HOSTBASED_SERVICE);
```

Механизмът на Керберос 5 ще свърже това име със специфичната за Керберос форма nfs/bar.foo.com@FOO.COM, която представлява сървис nfs вървящ на машината bar.foo.com.

Имплементацията на Сън (Sun Microsystems) е реализирана като контейнер, в който отделните компоненти извършат това конвертиране, когато техният механизъм се използва и след това запазва резултата. По това отношение имплементацията на GSSName е подобна на класа Subject в спецификацията JAAS[7], като дори може да съдържа същите елементи, който се запазват в множеството от самоличности в инстанцията от класа Subject, като използването им е ограничено в контекста на Java GSSAPI. Елементът, до който се конвертира името при Керберос имплементацията на Сън е инстанция на подклас на **javax. security.auth.kerberos.KerberosPricipal**.

3.2.3 Интерфейсът GSSCredential

Този интерфейс обхваща автентикационните данни (credentials) на една самоличност. Подобно на GSSName този интерфейс представлява контейнер за данните от множество механизми. Обект от този тип се създава по следния начин:

```
GSSCredential gssManager.createCredential(GSSName name,
                                          int lifetime,
                                          Oid[] desiredMechs,
                                          int usage);
```

Използване от страна на клиента

```
GSSCredential clientCredential =
   gssManager.createCredential(clientName,
                               8*3600,
                               desiredMechs,
                               GSSCredential.INITIATE_ONLY);
```

Тук инстанцията на GSSManager изисква от компонентите на механизмите, който са описани в desiredMechs, автентикационните данни, който принадлежат на clientName. Освен това указва ограничението, че те трябва да са от такъв тип, с който да може да се инициира заявка и тези данни да са с валидност 8 часа. Обектът получен от изпълнението съдържа елементи от подмножеството на избраните механизми, които имат автентикационни данни, които отговарят на изискванията. Елементът, зареждан от Керберос компонента, е инстанция на **javax.security.auth.kerberos.KerberosTicket** съдържащ главния билет, който принадлежи на потребителя.

Използване от страна на сървъра

```
GSSCredential serverCredential =
    gssManager.createCredential(serverName,
                                GSSCredential.INDEFINITE_LIFETIME,
                                desiredMechs,
                                GSSCredential.ACCEPT_ONLY);
```

Поведението при изпълнението на този код е подобно на изпълнението от страната на клиента с тази разлика, че автентикационните данни, който се искат са тези, който могат да приемат идващата заявка. Освен това тъй като сървърите са с дълъг живот тези данни се създават за дълъг период от време като указаното INDEFINITE_LIFETIME в примера. Елемента, зареждан от Керберос механизма е инстанция на класа **javax.security.auth.kerberos.KerberosKey** съдържащ личния ключ на сървъра. Тази стъпка обикновено е времеотнемаща и поради това приложенията обикновено я изпълняват по време на инициализацията си, за да се сдобият с всички автентикационни данни, които ще ползват по време на изпълнението си.

3.2.4 Интерфейсът GSSContext

Този интерфейс предоставя услуги за защита и на двете страни в комуникацията

- Създаване на обект GSSContext при клиента

```
GSSContext createContext(GSSName peer,
                         Oid mech,
                         GSSCredential clientCredential,
                         int lifetime) throws GSSException;
```

Той връща инициализиран контекст, който знае коя е отсрещната страна, с която ще комуникира и механизма за защита, който ще използва. Автентикационните данни на клиента са необходими за автентикацията пред отсрещната страна.

- Създаване на обект GSSContext при сървъра

```
GSSContext createContext(GSSCredential serverCredential) throws GSSException;
```

Този метод връща инициализиран контекст за получаващата страна. На този етап не е известно името на клиента, който ще прати заявка нито пък механизма, който ще бъде използван. Поради това, ако идващата заявка не е за този сървър представен чрез serverCredential или механизмът ползван от клиента няма елемент в serverCredential заявката ще пропадне.

3.2.5 Установяване на GSSContext

Преди да може да се ползва GSSContext трябва да се установи връзка, което става с размяна на съобщения между двата участника. Всяко изпълнение на метода за установяване на контекста генерира последователност от данни, които приложението по някакъв начин трябва да изпрати на отсрещната страна като използва канал за комуникация избран по свое усмотрение.

- Установяване на контекст от клиента

```
byte[]initSecContext(byte[] inToken, int offset, int len) throws GSSException;
```

- Установяване на контекст от сървъра

```
byte[] accesptSecContext(byte[] inToken, int offset, int len) throws
GSSException;
```

Тези два метода взаимно се допълват като входните данни приемани от единия са резултат от викането на другия. Първият низ се генерира от клиента при първото изпълнение на initSecContext. Аргументите подадени при това първоначално изпълнение на метода се игнорират. Последния генериран низ зависи от избрания механизъм за защита и параметрите на контекста. Броя на съобщенията необходими за автентикация в GSSAPI зависи от самия механизъм избран за осигуряване на защита, а също така зависи и от това дали се изисква едностранна или двустранна автентикация. Поради това двете страни трябва да изпълняват тези методи в цикъл до приключване на процеса.

Когато е избран Керберос това се извършва с едно пращане на съобщенията.

1. Клиентът извиква initSecContext(). Резултатът е низ, който съдържа Керберос заявка. Този низ се изпраща към приложението. За да генерира това съобщение Керберос компонента взема билет за сървъра, като използва главния билет. Издаденият билет е кодиран с тайния ключ на сървъра и е част от заявката към него.
2. Сървърът извиква acceptSecContext(). Тук се подава като параметър съобщението получено от клиента и методът декодира билета и автентицира клиента.

Ако не е необходима двустранна автентикация контекстите и на двете страни са установени и метода на сървъра acceptSecContext() няма да генерира изходни данни. Ако е необходима двустранна връзка тогава този метод ще генерира изходен низ, съдържащ отговора на сървъра. Този низ след това трябва да се прати на клиента и той да го подаде като параметър на метода initSecContext(). Едва след това контекста на клиента ще е установен.

Когато контекстът е инициализиран от страната на клиента, вече е ясно кой точно механизъм на защита е избран за използване. Средата на работа на Java GSSAPI може да вземе имплементацията на контекста от подходящия компонент предоставящ механизма за защита. Затова всички обръщения направени към обекта GSSContext се делегират на реализацията на контекста на избрания механизъм. Механизмът, който ще се ползва при сървъра, се установява след получаването на първото съобщение от клиента.

Следва примерен код показващ как се реализира клиентската страна[6].

```
GSSManager manager = GSSManager.getInstance();
GSSName clientName = manager.createName("duke",
                                        GSSName.NT_USER_NAME);
GSScredential clientCreds = manager.createCredential(clientName,
                                       8*3600,
                                       desireMechs,
                                       GSSCredential.INITIATE_ONLY);
GSSName peerName = manager.createName("nfs@bar.foo.com",
                                      GSSName.NT_HOSTBASED_SERVICE);
GSSContext secContext = manager.createContext(peerName,
                                   krb5Oid,
                                   clientCreds,
                                   GSSContext.DEFAULT_LIFETIME);
secContext.requestMutualAuth(true);

byte[] inToken = new byte[0];
byte[] outToken;
boolean established = false;

while (!established) {
  outToken = secContext.initSecContext(inToken, 0, inToken.length);
  if (outToken != null) {
    sendToken(outToken);
  }
  if (!secContext.isEstablished()) {
```

```
      inToken = readToken();
    } else {
      established = true;
    }
  }
```

Съответния код, който върви на сървъра е следният:

```
GSSManagem manager = GSSManager.getInstance();
GSSName serverName = manager.createName("nsf@bar.foo.com",
                                        GSSName.NT_HOSTBASED_SERVICE);
GSSCredential serverCreds = manager.createCredential(serverName,
                            GSSCredential.INDEFINITE_LIFETIME,
                            desiredMech,
                            GSSCredential.ACCEPT_ONLY);
GSSContext secContext = manager.createContext(serverCreds);
byte[] inToken = null;
byte[] outToken = null;

while (!secContext.isEstablished()) {
  inToken = readToken();
  outToken = secContext.acceptSecContext(inToken, 0, inToken.length);

  if (outToken != null) {
    sendToken(outToken);
  }
}
```

След като контекста е установен той може да бъде ползван за предпазване на съобщенията между клиента и сървъра. Java GSSAPI предоставя едновременно цялостност и поверителност на съобщенията. Двата метода, които осигуряват това се предоставят от GSSContext:

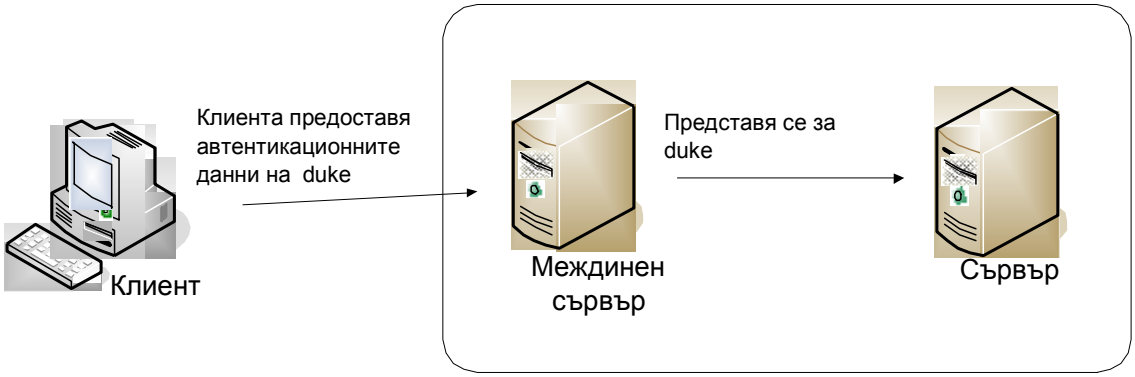
```
byte[] wrap(byte[] clearText,
            int offset,
            int length,
            MessageProp properties) throws GSSException;

byte[] unwrap(byte[] inToken,
              int offset,
              int length,
              MessageProp properties) throws GSSException;
```

Първият метод се използва за капсулирането на открит текст в низ, така че неговата цялост да е предпазена. Съществува вариант съобщението да се кодира, като това се указва в опциите, който се подават чрез properties параметъра. Метода wrap() връща неразбираем низ, който може да се прати на отсрещната страна. Оригиналният текст се възстановява чрез метода unwrap() като му се подаде полученият низ. Чрез параметъра properties се указва дали съобщението допълнително е кодирано.

3.3.6 Делегиране на автентикационни данни

Java GSS API позволява на клиента безопасно да предостави автентикационните си данни на сървъра, така че този сървър да може да създава контекст от името на клиента. Тази черта е

полезна за поддържането на еднократна идентификация (SSO). Фигура 7 изобразява схемата на делегиране на автентикационни данни[6].



Фигура 7: Делегиране на автентикационни данни.

За да предостави автентикационните си данни клиентът трябва преди initSecContext() да изпълни следния метод на GSSContext като подава true за параметъра state:

```
void requestCredDeleg(boolean state) throws GSSException;
```

Междинният сървър получава делегираните автентикационни данни посредством метода:

```
GSSCredential getDelegateCred() throws GSSException;
```

В случаите, когато се ползва Керберос, делегираните автентикационни данни включват единствено главния Кербероски билет, който се препраща като част от първия низ, изпратен от клиента. Използвайки този главен билет междинният сървър може да се сдобие с други Керберос билети, с който да се представя като клиента пред различните сървиси.

3.3 Стандартен начин за придобиване на автентикационните данни

Дотук описахме как едно приложения използва GSSManager.createCredential () за да се сдобие с автентикационните данни. Сега ще се фокусираме върху механизма за вземане на тези данни. Този механизъм сам по себе си не извършва автентикация. Вместо това се приема, че автентикацията е извършена преди да се ползват Java GSSAPI. Автентикационните данни се предполага, че са съхранени в някакъв кеш, с който механизмът е запознат. Методът взема референция към тези данни и ги предоставя във вид на инстанция на класа GSSCredential. В Java 2 е наложено ограничението кеша, който компонентите за механизмите в Java GGSAPI използват да бъдат точно публичните и тайните множества от автентикационни данни (public and private credentials) в обекта Subject (от спецификацията JAAS) на текущия контекст за контрол на достъпа. Този модел има предимството, че управлението на автентикационните

данни е просто и предвидимо от гледна точка на приложението. Приложение, което има необходимите права може да махне тези данни от обекта Subject или да ги поднови като използва стандартните Java класове.

3.3.1 Процедура за вземане на автентикационните данни, когато се използва протокола Керберос

1. Приложението извиква JAAS логин, който от своя страна извиква конфигурирания Krb5LoginModule[9].
2. Krb5LoginModule взема главния билет за потребителя от KDC или от съществуващ кеш и го запазва в личните автентикационни данни на обекта Subject.
3. Приложението получава попълнения обект Subject, след това извиква Subject. doAs/doAsPrivileged, който поставя обекта в контекста за контрол на достъпа на нишката, в която върви приложението.
4. Извиква се метода GSSManager.createCredential, като подава идентификатора за Керберос като желан механизъм.
5. GSSManager.createCredential иска от Керберос компонента автентикационните данни за създаване на контекст за сигурност.
6. Керберос компонента взема Subject обекта от текущия контекст за контрол на достъпа и търси в множеството от лични автентикационни данни за валиден Кербероски билет във вид на инстанция на KerberosTicket
7. Инстанцията на KerberosTicket се връща на обекта от тип GSSManager, който го записва в контейнера от обекти GSSCredential и го връща като резултат.

На сървърската страна, когато Керберос логин е успешен на стъпка 2, Krb5LoginModule записва в обекта Subject обект KerberosKey, който съдържа личния ключ на сървъра освен KerberosTicket. След това този KerberosKey се използва на стъпките от 5 до 7 и се използва за декодиране на билета получен от клиента. Описаният стандартен начин за сдобиване с необходимите автентикационни данни изисква наличието им в текущия Subject. Обикновено такива данни се слагат там след успешен JAAS логин.

4. Протоколът SPNego

4.1 Интегрирана Автентикация за Windows

Това е понятие свързано с компанията Майкрософт. Отнася се до протоколите за автентикация Kerberos и NTLMSSP[10]. IWA не е име на стандартен протокол. Използване на интегрирана автентикация означава, че механизмите за сигурност, които се използват на ниско ниво ще бъдат използвани в определен ред. По-конкретно, ако работи услугата предоставяща Керберос и билетът за Керберос комуникация може да бъде получен за дадена услуга (сървис) и останалите настройки позволяват да се използва Керберос автентикация (например настройките на интернет браузъра), тогава протоколът Керберос ще бъде заявен т.е. първо ще бъде направен опит за Керберос автентикация. Иначе ще се задейства автентикация с протокола NTLMSSP. Ако автентикацията с Керберос се задейства, но пропадне тогава ще се активира отново NTLMSSP. NTLMSSP е протокол за обмен на съобщения асоциирани с протокола за автентикация NTLM. Съкращението NTML произлиза от NTLan Manager – протокол за автентикация разработен от Майкрософт. Този протокол наследява LANMAN (Microsoft Lan Manager) и е съвместим с него.

4.2 Характеристики на SPNego

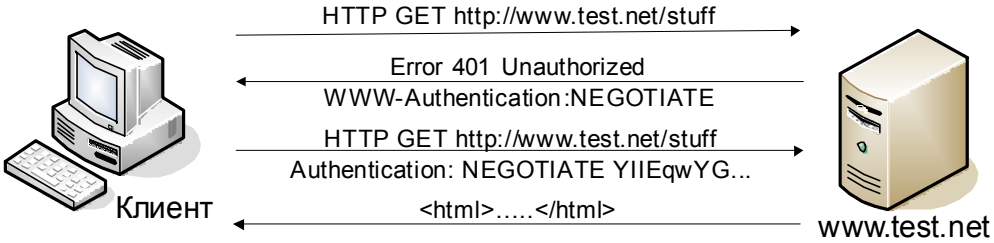
SPNego(Simple and Protected Negotiation Mechanism)[8] е стандартен механизъм, които се използва при интегрирана автентикация с Windows. Посредством този механизъм се определя протокола, с който потребителите ще се автентицират и се установява контекст за комуникация. SPNego има най-голямо разпространение от Майкрософт. Този протокол се поддържа от браузърът InternetExplorer 5.01 (и всички по-нови версии) и осигурява възможност за Single Sign-On чрез делегирана автентикация между операционната система и даден сървър за приложения. Интегрирана автентикация се поддържа също така и от по-новите версии на Mozilla (например Firefox). Сам по себе си Керберос не дефинира начина, по който той работи между web браузър и web сървър. На практика Майкрософт дефинира начина, по който се реализира тази интеграция. На фигура 8 е показана структурата на низовете използвани в комуникацията дефинирана в протокола SPNego.

Фигура 8: Структура на SPNego низ.

Протоколът SPNego има следните основни характеристики:

- Представлява обвивка на протокол базиран на GSSAPI.
- Позволява уговаряне на механизма за автентикация
- Поддържа всички GSSAPI механизми
- За протокола HTTP, низовете се разменят като HTTP хедъри между сървъра и браузъра

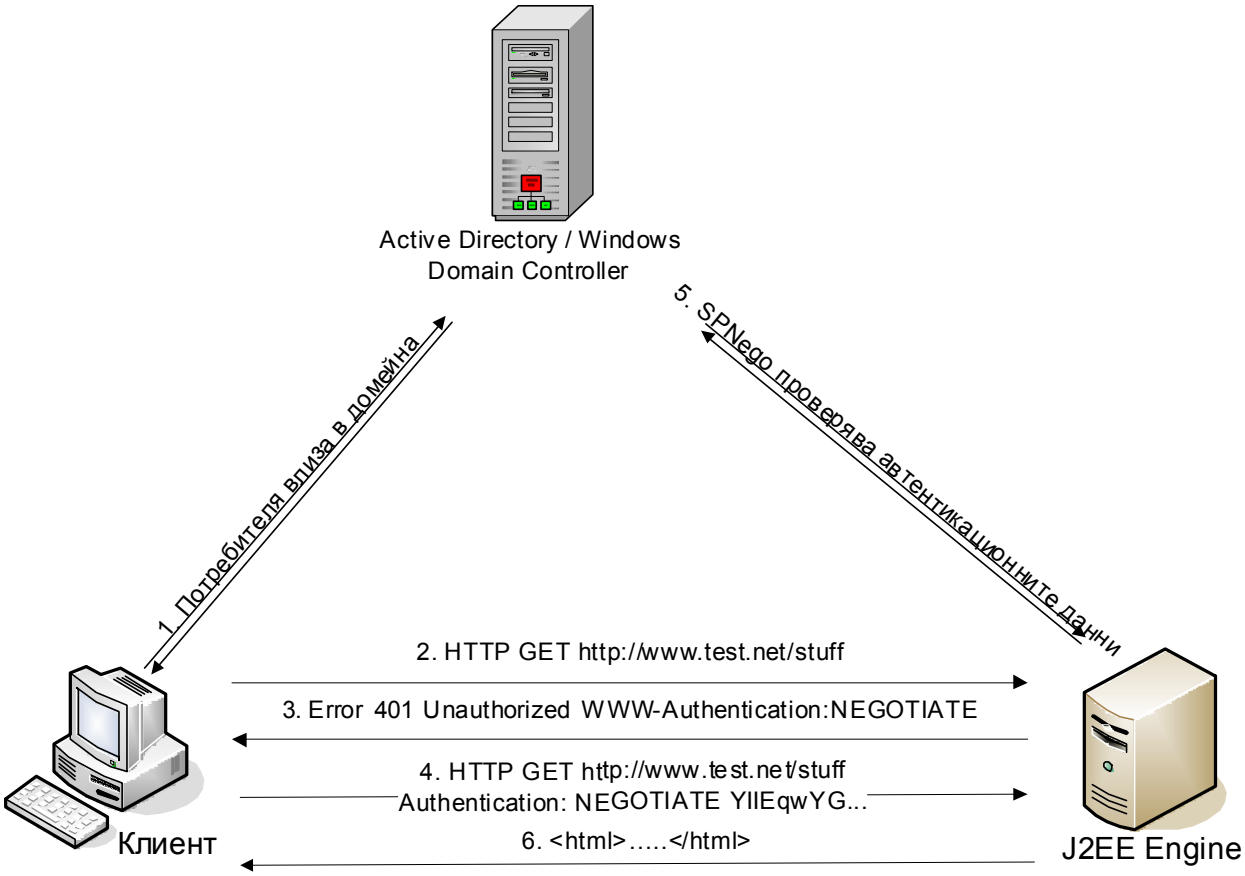
МеханизмътSPNego дефинира употребата на HTTP хедърите "**WWW-Authenticate**" и "**Authorization**", които са част от стандартизирания HTTP протокол, с тази разлика, че ги разширява със специфичен метод за GSSAPI - "**Negotiate**". Когато web сървър иска от клиента(браузъра) да се автентицира, чрез този метод, праща отговор **"401 Unauthorized"**. След това сървърът очаква от клиента да върне заявка с "Authorization" хедър съдържащ GSSAPI низа. На фигура 9 е показана обмяната на съобщенията имащи отношение към протокола SPNego при осъществяване на заявка до примерен сървър test.net с използване на интегрирана автентикация.



Фигура 9: Автентикация пред web сървър със SPNego

На практика пълният сценарий включва следните стъпки, които са илюстрирани на фигура 10:

1. Потребителят се автентицира в областта test.net при започване на работа на компютъра. Прави се заявка към домейн контролера и потребителят получава своя главен кербероски билет.
2. Потребителят иска да отвори защитена страница от **www.test.net.** Браузърът иска достъп до ресурс на сървъра с GET заявка.
3. Сървърът отговаря с код „HTTP Error 401 Unauthorized" и HTTP хедър „WWW-Autheticate: Negotiate". Започва SPNego протокола.
4. Браузърът разпознава, че машината, на която работи сървърът, е част от Керберос областта и праща отново заявката към защитената страница, като добавя HTTP Authorization хедър, който съдържа кербероския билет пакетиран като SPNego низ. (В случай, че няма такъв билет браузърът иска от домейн контролера билет за сървиса HTTP/www.test.net. Това се прави само първия път, когато е необходим такъв билет.)
5. Сървърът получава SPNego низа. Посредством конкретната Керберос имплементация (предоставена от JDK чрез библиотеките на GSSAPI) извлича името на потребителя от използвайки получения SPNego низ.
6. Проверява се дали потребителят има права за достъп и ако има такива позволява достъпа до ресурса.



Фигура 10: Керберос автентикация в J2EE Engine

4.3 NTLM протокол

Майкрософт използва Керберос като първи протокол за автентикация в Windows 2000 и Windows 2003 Active Directory домейни. Керберос се използва, когато потребителят се включи в Windows домейна при започване на работа на компютъра. Както споменахме вече, интегрираната автентикация поддържа двата механизма – Керберос и NTLM[14]. NTLM все още се използва в случаи като изброените:

- Потребителят се автентицира към сървъра с IP адреса си
- Потребителят се автентицира към сървър, който не е към същия домейн
- Не съществува никакъв Active Directory домейн

NTLM се поддържа най-вече за съвместимост с по-стари системи и в случаите когато SPNego не успее да сработи, NTLM автоматично се активира. Тогава сървърът получава NTLM низ, който се разпознава по началните си символи “TlRM”. Подобно на SPNego, NTLM

автентикацията работи по схемата на договарянето (challenge-response), която включва 3 съобщения. Ето как работи сценарият:

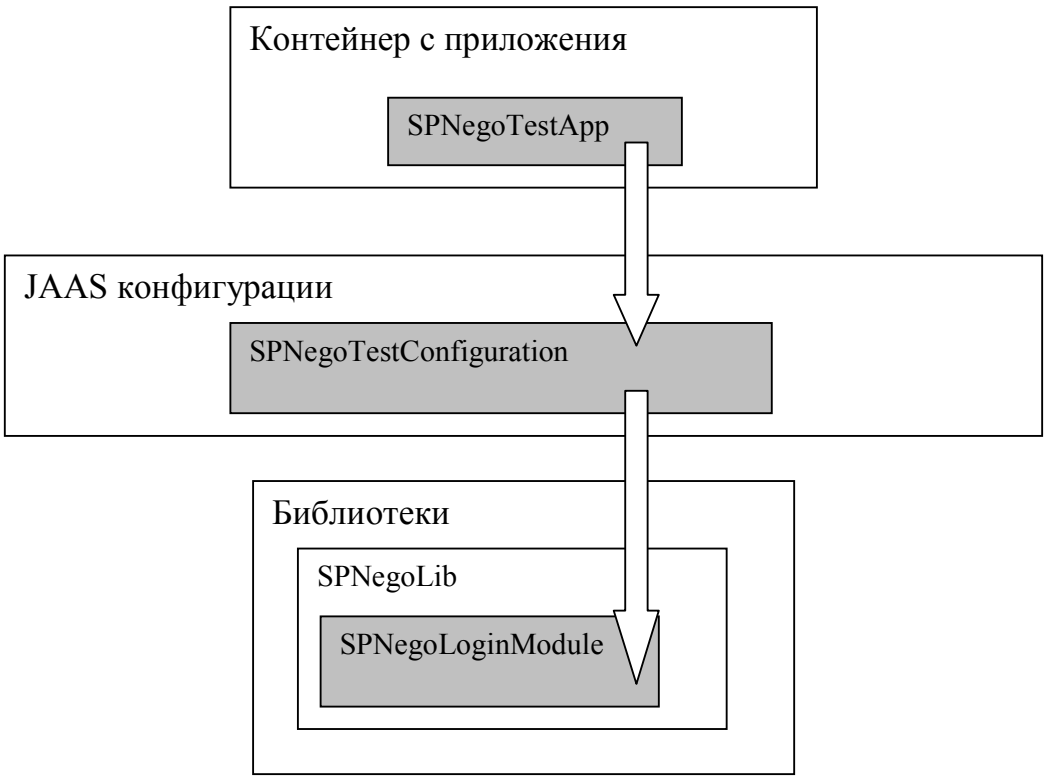
1. **Договаряне (negotiation)** – потребителят изпраща списък с поддържаните от него механизми, с които може да се автентицира пред сървъра
2. **Покана (challenge)** - Сървърът отговаря на потребителя със списък от механизми, които той одобрява и също поддържа. Най-вече това съобщение съдържа заявка за данните, с които потребителят ще се автентицира по първият механизъм, който и двамата поддържат.
3. **Автентикация (authentication)** – Потребителят отговаря като изпраща данните включващи домейна , потребителското име и други данни ако е нужно.

5. Реализация

5.1 Архитектура на решението

Една от задачите на дипломната работа е да представи решение използващо по-горе описаните технологии, чрез което да се демонстрират възможностите за автентикация на потребител регистриран в Windows домейна пред сървър за приложения. Функционалността, която осъществява автентикацията е реализирана като библиотека, която се инсталира на сървъра. Въпросната библиотека се казва SPNegoLib и по-нататък ще използваме това име за краткост. Тъй като основната част от работата по изпълнението на дипломното задание е съсредоточена върху тази библиотека, в тази глава основно ще разгледаме нейните най-важни детайли. SPNegoLib съдържа логин модул наречен SPNegoLoginModule. Този логин модул използва набор от помощни класове, които също за част от библиотеката. Детайлното описание на класовете включени в библиотеката както и фрагменти програмен код, са поместени в главата „Приложение". Тук ще се спрем само на най-ключовите помощни класове SPNegoInit и SPNegoTarg. Освен класовете от SPNegoLib, за работата на SPNegoLoginModule са необходими и някои други класове, които се използват като част от стандартните библиотеки работещи на конкретния сървър(в случая SAP AS Java 6.40). Част от тези класове са свързани с реализацията на платформата JAAS. Употребата им е сведена до ниво интерфейси, като по този начин максимално се абстрахираме от конкретния използван сървър. Други класове са свързани с извеждането на съобщения за откриване на грешки, за съпоставяне на автентицирания потребител от домейна към реална регистрация на потребител на сървъра и други. Кодът с участието на тези класове не е поместен в дипломната работа, тъй като това не е съществено за разясняването на алгоритъма и реализацията му. Повече разяснения по този въпрос са поместени в главата „Проблеми и насоки за развитие". За да се тества функционалността от SPNegoLib е реализирано малко Web приложение, което по-нататък ще наричаме SPNegoTestApp. Логически то може да се раздели на 2 части, които физически са поместени в два отделни WAR файла. Двата WAR файла съдържат няколко малки JSP файла. Подробности за реализацията на SPNegoTestApp няма да излагаме в тази глава, тъй като функционалността достатъчно ясно се разбира от описанието за употреба и тестване изложено в следващата глава. SPNegoTestApp трябва да се инсталира на сървъра и се конфигурира да използва логин модула от SPNegoLib. Това приложение ни помага да проверим дали успешно могат да се автентицират потребителите пред сървъра и в случай на грешка ни предоставя възможност да проверим дали сървърът е настроен правилно за да може да използва този тип автентикация.

Благодарение на спецификацията JAAS[7] можем да конфигурираме приложенията работещи на сървъра да използват различни механизми за автентикация без да променяме самите приложения. Връзката между приложението и механизмите за автентикация става чрез използването на JAAS конфигурации[13], в които указваме списък от логин модули, които дефинират правилата определящи как ще протече процеса на автентикация. За да осъществим връзката между SPNegoTestApp и SPNegoLib използваме JAAS конфигурация, която ще наричаме SPNegoTestConfiguration. Това е конфигурацията, която дефинира списъка от логин модули за автентикацията на SPNegoTestApp. (Тук няма да конктеретизираме как точно става задаването на връзката между JAAS конфигурация от логин модули и дадено приложение. За различните сървъри са възможни различни реализации, които не са тема на дипломната работа.) В списъка от логин модули добавяме логин модула SPNegoLoginModule, които е част от библиотеката SPNegoLib. Схематично тази връзка е показана на фигура 11. Ако потребителите успешно се автентицират, когато достъпват приложението SPNegoTestApp, то след това те успешно ще могат да се автентицират и пред други приложения, като единствено ще трябва да се променят техните (на приложенията) JAAS конфигурации.

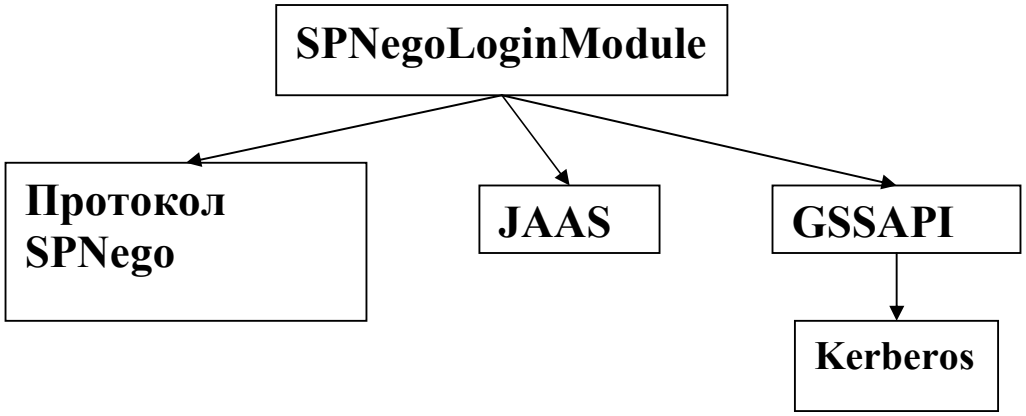


Фигура 11: Връзка между SPNegoLib, SPNegoTestApp и SPNegoTestConfiguration

5.2 Описание на библиотеката SPNegoLib

5.2.1 SPNegoLoginModule

Разработването на библиотеката SPNegoLib представлява основната програмна част свързана с работата по дипломното задание. Тук се осъществява връзката между технологиите описани в предходните глави. Най-важен клас в цялата библиотека е класът SPNegoLoginModule. SPNegoLoginModule реализира интерфейса LoginModule предоставен от спецификацията JAAS, изпълнява стъпките от протокола SPNego и посредством Java GSSAPI успява да извлече името на потребителя. Връзката между SPNegoLoginModule и разглежданите технологии схематично е показана на фигура 12.



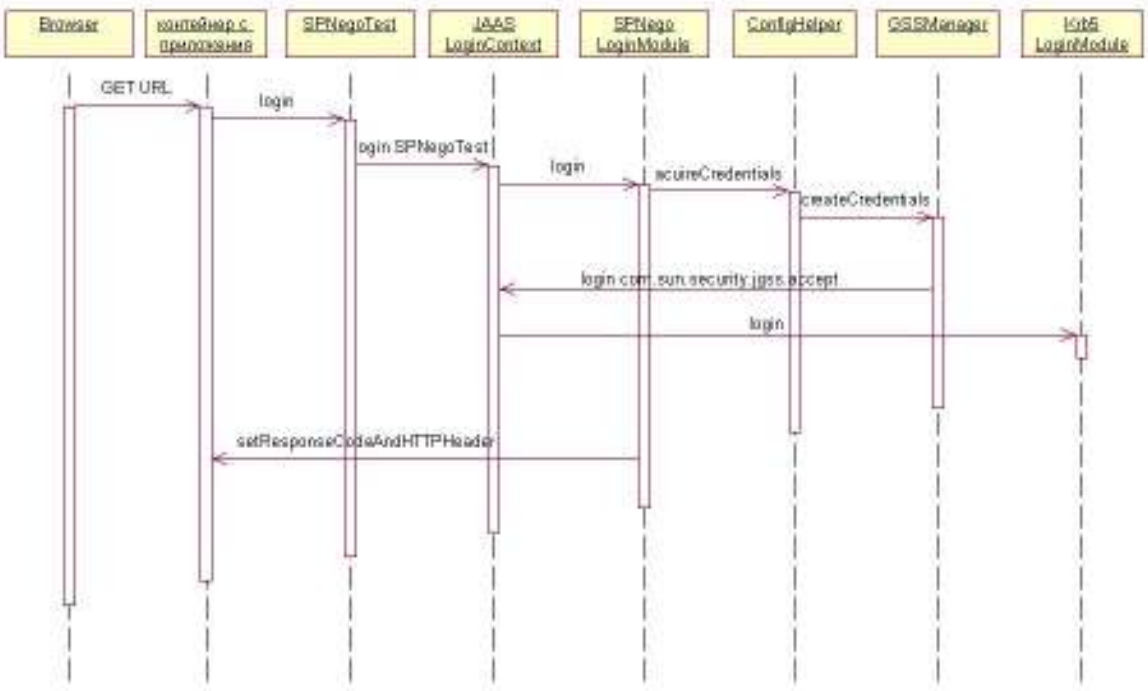
Фигура 12: SPNegoLoginModule и технологиите, на които се базира

SPNegoLoginModule изпълнява стъпките от протокола SPNego. Както стана ясно има 2 основни фази на на този процес.

5.2.1.1 Фаза 1

В първата фаза браузърът изпраща заявка към сървъра за дадено приложение. В този момент потребителят(браузърът) не знае дали се изисква някаква автентикация и ако се изисква такава не знае какви автентикационни данни са необходими на сървъра. Сървърът получава заявката и проверява дали е необходима автентикация. Ако е необходима тогава се проверява списъкът от логин модули дефиниран в JAAS конфигурацията. Сега (в случая с тестовото приложение SPNegoTestApp) сървърът зарежда логин модулите зададени в SPNegoTestConfiguration и по-конкретно SPNegoLoginModule. (За да опростим тестовия сценарии в SPNegoTestConfiguration използваме единствено SPNegoLoginModule и никакви други допълнителни механизми за автентикация.) SPNegoLoginModule проверява HTTP заявката за HTTP хедър „WWW-Authenticate”. В този момент такъв хедър не е получен. (Това е така, защото първо браузърът трябва да получи заявка за такъв хедър и чак тогава го праща.)

Сега логин модулът трябва да укаже използването на протокол SPNego от браузъра. Но преди това логин модулът проверява дали изобщо той би могъл да обработи съобщения от протокола SPNego. За целта се използват класовете от стандарта GSSAPI. Чрез тях логин модулът се автентицира пред домейн контролера. GSSAPI използва специален клас Krb5LoginModule. Този клас извършва автентикацията на сървъра пред домейн контролера. Ако това стане успешно тогава SPNegoLoginModule добавя HTTP хедър в отговора към браузъра. С това приключва първата фаза. Схематично стъпките, които се изпълняват, и класовете, чрез които става това, са изобразени на фигура 13



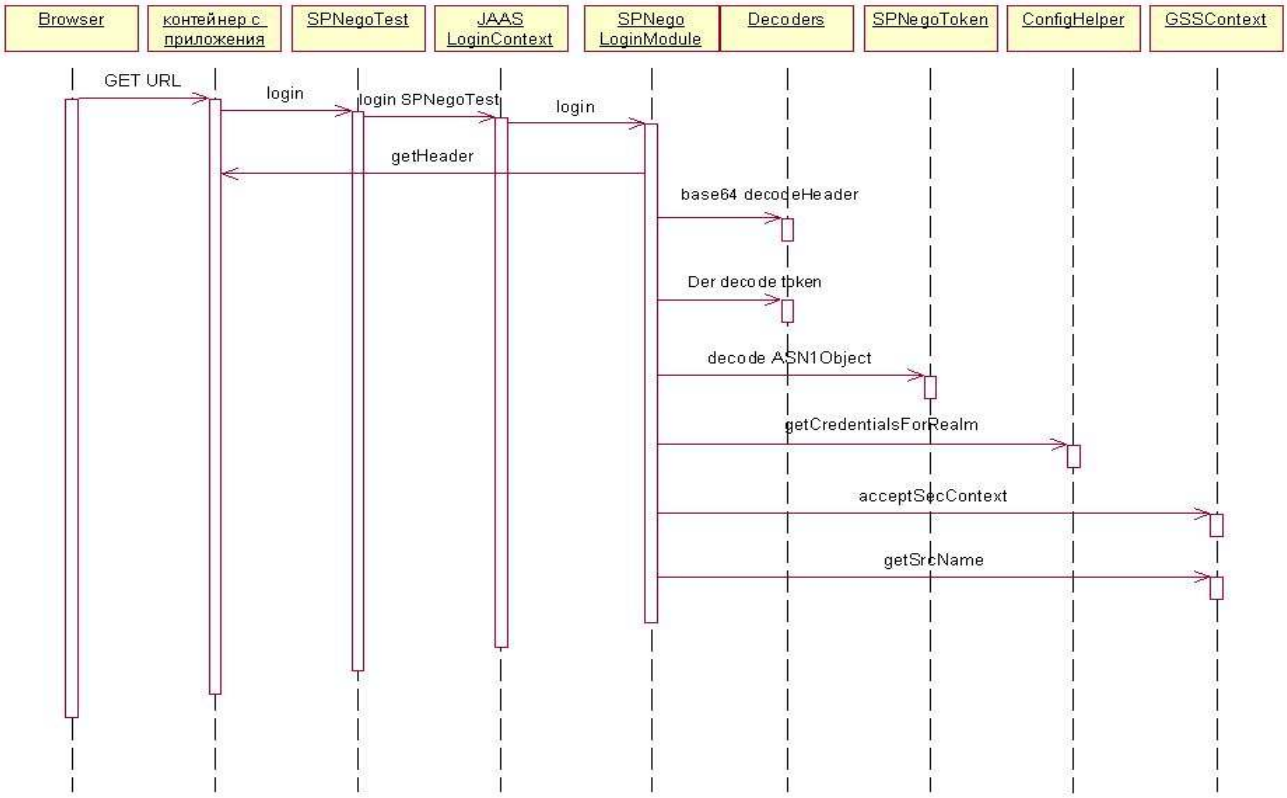
Фигура 13: Първа фаза на автентикацията

Пояснение: Механизмът на извикването на Krb5LoginModule е скрит в реализацията на съответната версия на GSSAPI. Както ще бъде разяснено по-нататък, SUN използва специална JAAS конфигурация com.sun.security.jgss.accept, която трябва да съдържа Krb5LoginModule. В повечето случаи това е единственият логин модул, който трябва да съдържа. В случаите с IBM JDK такава допълнителна конфигурация не е необходима.

5.2.1.2 Фаза 2

Във втората фаза браузърът изпраща HTTP хедър със съответното съобщение от протокола SPNego. Отново се зарежда списъка от логин модули. SPNegoLoginModule декодира

съобщението изпратено от браузъра като първо извършва Base64 декодиране, а след това декодира получената структура от формат ASN.1. След това полученият масив от байтове се използва за установяване на контекст за комуникация. В случай, че няма грешки контекстът се установява успешно и от него логин модулът получава името на потребителя. С това приключва втората фаза. Основните стъпки от процеса изпълняван във фаза 2, както и използваните класове, са изобразен на фигура 14



Фигура 14: Втора фаза на автентикацията

Пояснение: В случай, че не може да се установи контекст на базата на първия изпратен HTTP хедър, се налага използването на фаза 3, при която браузърът изпраща допълнителни автентикационни данни. Повече подробности за този случай са поместени по-нататък в тази глава.

5.2.1.3 Метод login() на SPNegoLoginModule

Сега ще разгледаме операциите, които се изпълняват в логин модула по време на двете фази на автентикацията. Интерфейсът LoginModule дефиниран от JAAS изисква дефинирането на 4 метода: login(), submit(), abort() и logout(). Извикването на всички обръщения, посредством които се извършва автентикацията на практика става по време на изпълнението на метода

login(). Затова ще разгледаме какво се случва в този метод и как се определя резултатът от опита за автентикация. По-горе дефинирахме 2 фази, които характеризирахме така:

Фаза1: Първоначално обръщение към сървъра без HTTP хедър.

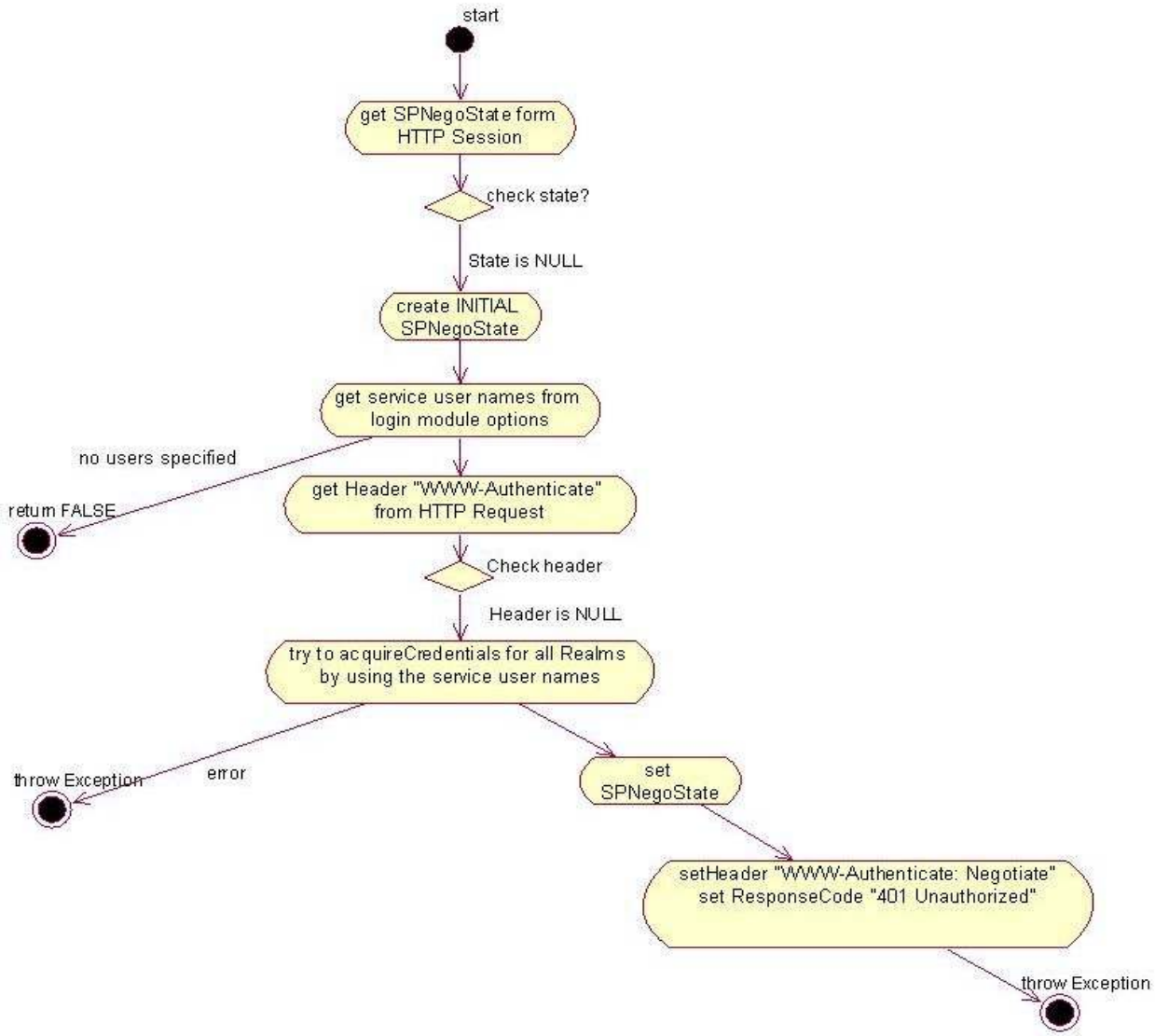
Фаза2: Второ обръщение към сървъра с SPNego билет в HTTP хедъра.

За да установим в коя фаза се намираме при започване на метода login() използваме обект от помощния клас SpNegoState. Той пази информацията за текущата фаза на автентикацията между отделните стъпки на договаряне. Логин модулът съхранява обект от този клас в HTTP сесията. В началото на метода login() се взема този обект от HTTP сесията и стойността на HTTP хедъра "WWW-Authenticate" ако има такъв изпратен. Основно на базата на тази информация ще бъде определен резултатът от опита за автентикация.

Нека разгледаме отделните случаи, през които може да премине изпълнението на метода login().

Случай 1. Статусът на SpNegoState е „НАЧАЛЕН" и HTTP хедър "WWW-Authenticate" не е изпратен. Логически това е първият случай, в която попадаме при първоначална заявка. Преди да бъде изпратен отговор със заявка за Керберос автентикация на потребителя, се проверява дали успешно може да се автентицира специалният сървис потребител, който идентифицира J2EE сървъра пред домейн контролера. Може да имаме дефинирани няколко такива потребителя в случай, че искаме да предоставим възможност за автентикация на потребителите от няколко домейна. Ако сървърът успешно се идентифицира използвайки сървис потребителите указани като опция на логин модула, тогава се създават обекти от класа GSSCredential, и се добавят в специален автентикационен кеш (реализиран във вид на Hashtable, в който добавяме обекти от класа GSSCredential), които кеш по-късно ще бъде използван за установяването на контекст (обект GSSContext). В случай, че няма указани никакви сървис потребители, то логин модулът не може да автентицира сървъра пред домейн контролера и цялостната автентикация се прекратява. Автентикацията на сървис потребителите на тази стъпка става чрез методите предоставени от класа GSSManager. Тук скрито от програмиста се използва Krb5LoginModule, който извършва опит за автентикация използвайки специални Java параметри, keytab файл и krb5.conf файл. Те се разглеждат по-подробно в главата показваща конфигурационните стъпки. Ако получим грешка на тази стъпка, то изобщо няма нужда да искаме от браузъра SPNego отговор, понеже няма да можем да го обработим. В този случай прекратяваме автентикацията. Ако успешно е автентициран сървърът пред домейн контролера тогава изпращаме отговор (HTTP response) с код 401 „Unauthorized" и HTTP хедър

“WWW-Authenticate: Negotiate”. С този специален хедър указваме на браузъра, че сървърът изисква (по–точно логин модулът) Керберос автентикация. Сега добавяме конструирания обект от класа SpNegoState в HTTP сесията, която се пази от сървъра. Целта е да се продължил процеса на автентикация от там, където е достигнал сега. В противен случай при следващо извикване на login() целият процес ще започне отначало. В този момент както стана ясно login() не може да завърши успешно автентикацията и затова се генерира изключение LoginException съгласно спецификацията JAAS[7]. На фигура 15 схематично е изобразен току-що описания алгоритъм.



Фигура 15: Метод login() по време на фаза 1

Когато браузърът получи въпросния отговор (HTTP response), той изпраща HTTP хедър „WWW-Authenticate: Negotiate YII…„. Със символите ”YII” започват всички коректни SPNego

низове. Ако браузърът не може да генерира SPNego низ, то той автоматично изпраща NTLM низ, който започва със символите “TlRM”.

Случай 2. Сега започва втората фаза на автентикацията. В този момент SpNegoState има статус „НАЧАЛЕН” или „НЕЗАВЪРШЕН” и HTTP хедър е изпратен за да може сървърът да открие името на потребителя. Първо проверяваме дали низът наистина е за SPNego, а не NTLM например. На практика проверяваме дали получения низ започва със съответният префикс. Ако започва с “TlRM”, то отхвърляме опита за автентикация. На този етап по-точна проверка не можем да направим. Ако низът не е реално SPNego съобщение, то по-нататък ще се получи грешка заради грешен формат. Сега използваме класа ThreadTokenCache за да проверим дали този SPNego низ не е бил вече обработван от текущия Java процес. Така си спестяваме повторното обработване на низа и директно връщаме потребителското име или някакво съобщение за грешка. Ако този SPNego низ не присъства в ThreadTokenCache започваме извличането на информацията посредством класовете от GSSAPI. В случай на първоначална автентикация на потребителя в ThreadTokenCache няма да бъде намерено потребителското име. За да успее да използва SPNego съобщението, логин модулът първо прилага декодиране Base64, а след това разкодира ASN.1 структурата и получаване ASN1Object. Този ASN1Object представлява SPNegoToken (който е или SPNegoInit или SPNegoTarg – разглеждат се по нататък). Сега вземаме под внимание статусът на SpNegoState. Тук разглеждаме 2 подслучая:

Подслучай 1. Ако е „НАЧАЛЕН”, то SPNegoToken трябва да е от тип SpNegoInit. Извличаме Керберос заявката (обект от тип KerberosApReq) и от тук получаваме и Кербероския билет (обект от тип KerberorTicket). От билета извличаме името на домейна, от който пристига SPNego съобщението. За този домейн трябва да имаме вече генериран обект GSSCredential, който можем да получим от автентикационния кеш. Ако няма такъв обект, то това означава, че потребители от този домейн няма да могат да се автентицират и цялостната автентикация се прекратява. Като използваме автентикационните данни (кеширания GSSCredential) инициализираме обект GSSContext. Сега вече извикваме метода GSSContext. acceptSecContext.

Подслучай 2. Ако статусът на SpNegoState е „НЕЗАВЪРШЕН”, то значи сме получили обект SpNegoTarg закодиран в HTTP хедъра. Тук извикваме метода GSSContext .acceptSecContext като му подаваме като параметър въпросният SpNegoTarg.

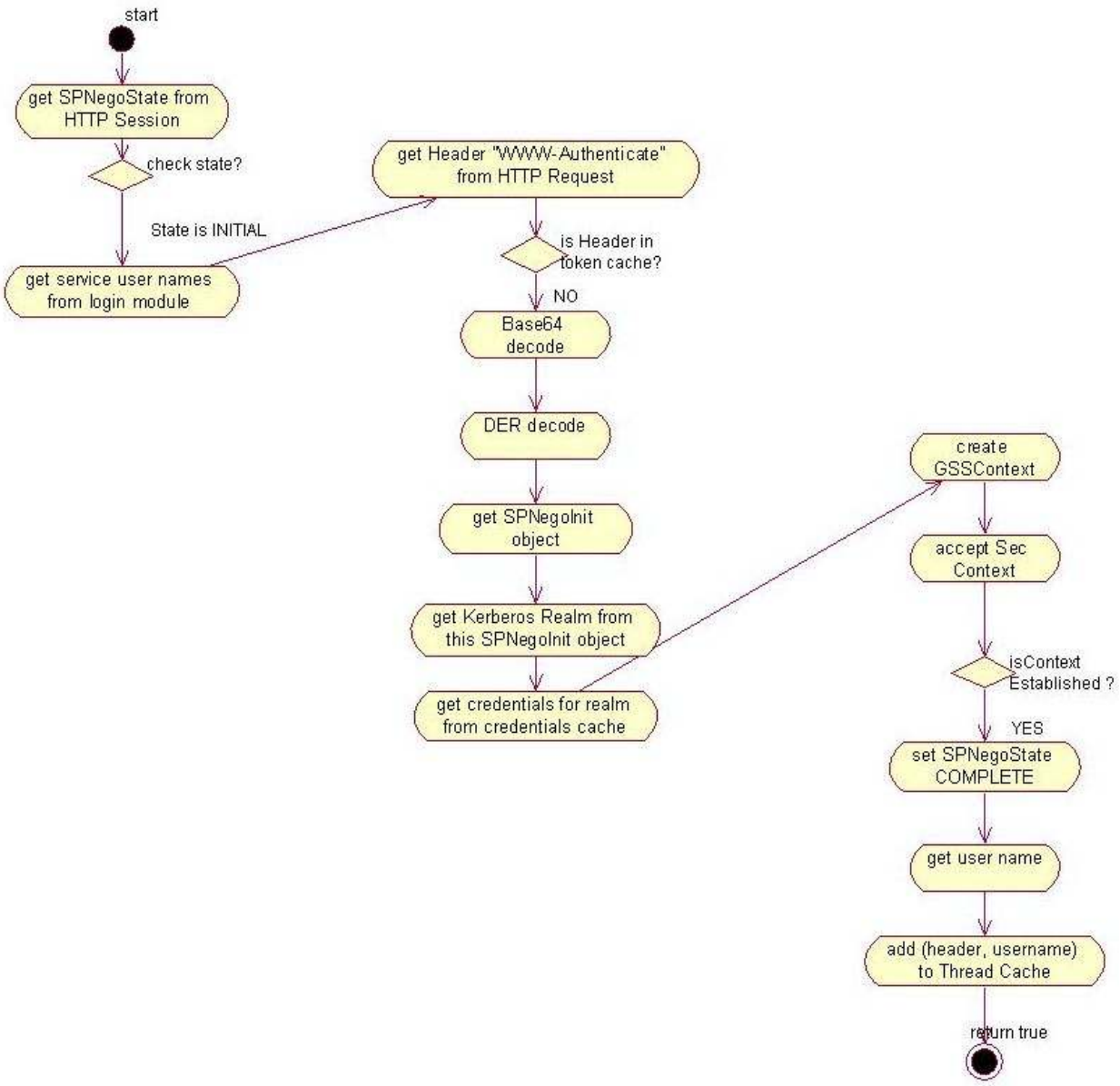
Ако установяването на контекст е успешно (GSSContext.isEstablished() == true), то можем да получим името на потребителя като извикаме GSSContext.getSrcName() – крайната цел е достигната, автентикацията е завършила успешно. Ако не е установен контекст все още, тогава трябва да изпратим резултатът от `acceptSecContext` отново към браузъра. Статусът на SpNegoState става „НЕЗАВЪРШЕН”. Преди да изпратим отговор към браузъра за да продължи договарянето трябва да подготвим новия хедър:

1.Конструираме ASN.1 структурата от масива от байтове получени от acceptSecContext.

2.Прилагаме кодиране DER.

3.Прилагаме кодиране Base64.

Получения низ изпращаме отново като HTTP хедър „WWW-Authenticate: Negotiate <отговор>”.В стандартния случай, когато пристигне коректен SPNegoInit, в края на фаза 2 методът login() успява да получи името на потребителя. Фигура 16 изобразява поредицата от стъпки схематично. С цел опростяване на фигурата разклонението в случая, когато GSSContext все още не е установен, не е добавен.

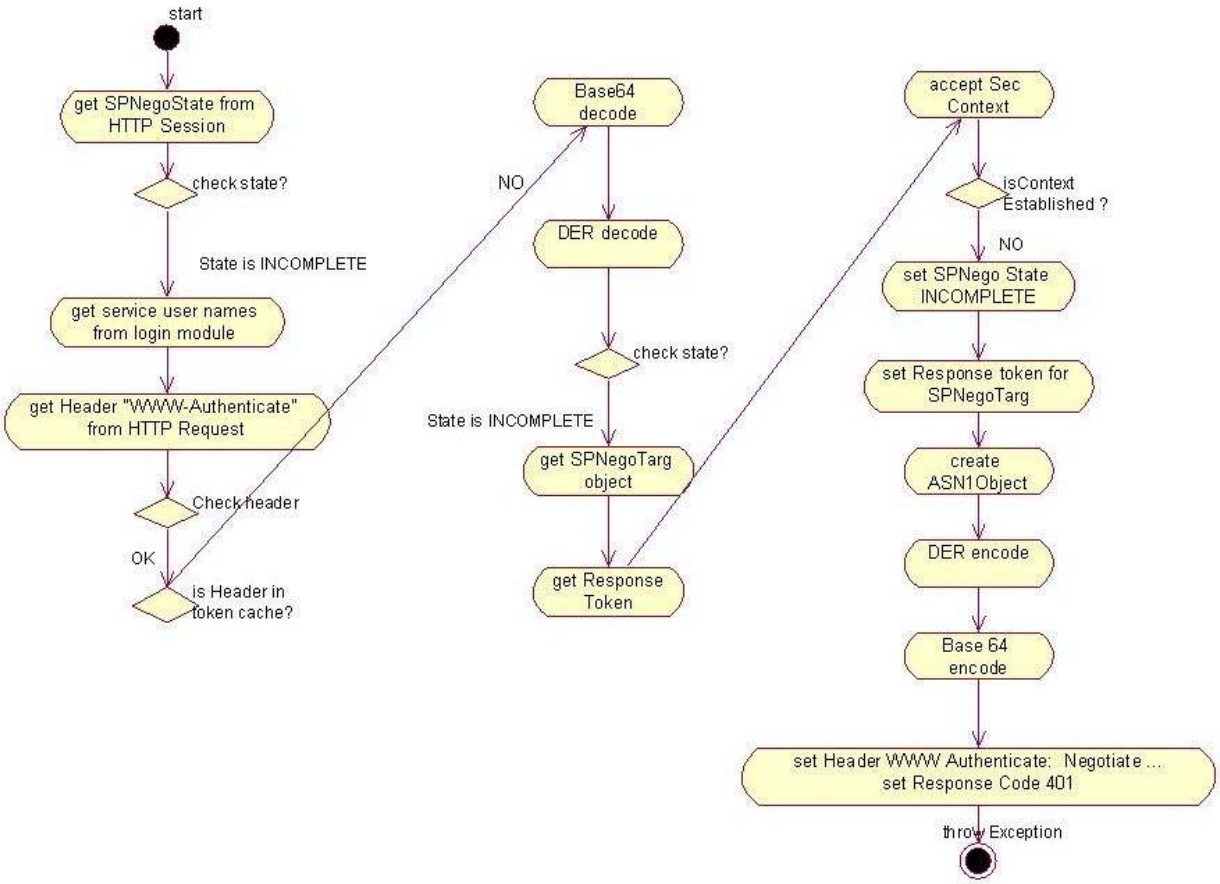


Фигура 16: Метод login() по време на фаза 2 в случая, когато всички операции завършват без грешки и успешно се открива потребителското име

Фигурите 15 и 16 показват поведението на метода login() в най-стандартния сценарий, при които ако коректно са изпълнени всички конфигурационни стъпки то автентикацията успешно приключва след края на фаза 2.

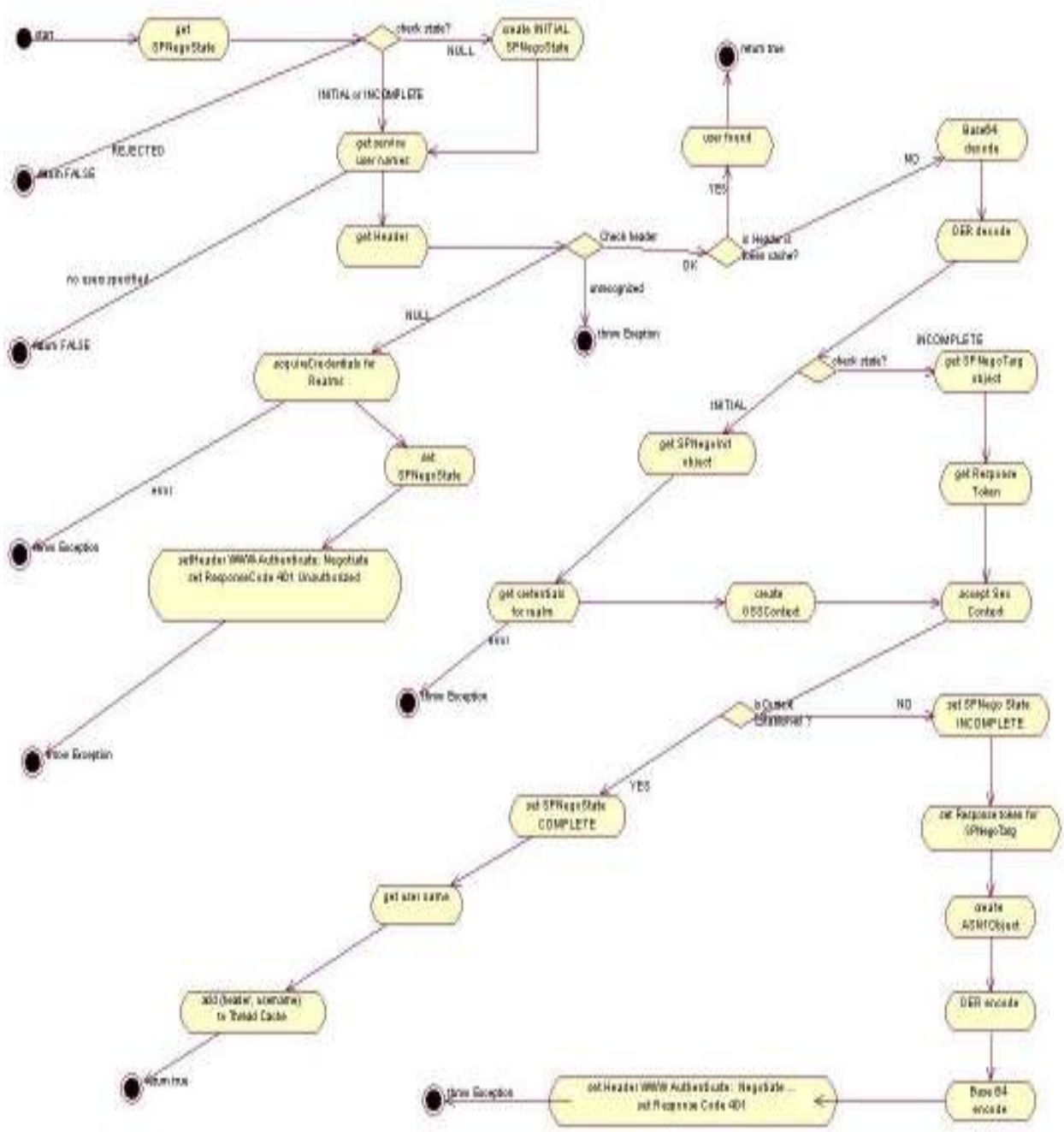
Фигура 17 показва един по-нестандартен сценарий, в който SpNegoState има статус НЕЗАВЪРШЕН и след извикването на GSSContext.acceptSecContext() контекстът все още не е установен. Тогава статусът продължава да е НЕЗАВЪРШЕН и отново се изпраща заявка за

продължаване на договарянето с браузъра. Този сценарий в реалността трудно се получава. Идеята на фигура 17 е най-вече да покаже един сценарий-комбинация от двете разклонения, които доста рядко могат да се случат дори и поотделно само – конструиране на обект SPNegoTarg в трета фаза на автентикация и изпращане на “WWW-Authenticate: Negotiate xxx” в следствие на неуспех след acceptSecContext().



Фигура 17: Метод login() по време на фаза 3, в случая на неуспешно установяване на контекст.

Фигура 18 показва в най-пълен вид логиката на метода login(). Изобразени са всички случай на грешки предизвикани от неправилно конфигуриране както и всички възможни разклонения. Фигура 18 съдържа в себе си фигурите 15, 16 и 17. За пълнота са включени и случайте, когато потребителското име е открито в ThreadTokenCache, както и случая с SpNegoState със статус ОТХВЪРЛЕН.



Фигура 18: Пълна схема на метода login()

5.2.2 Класовете SpNegoInit и SpNegoTarg

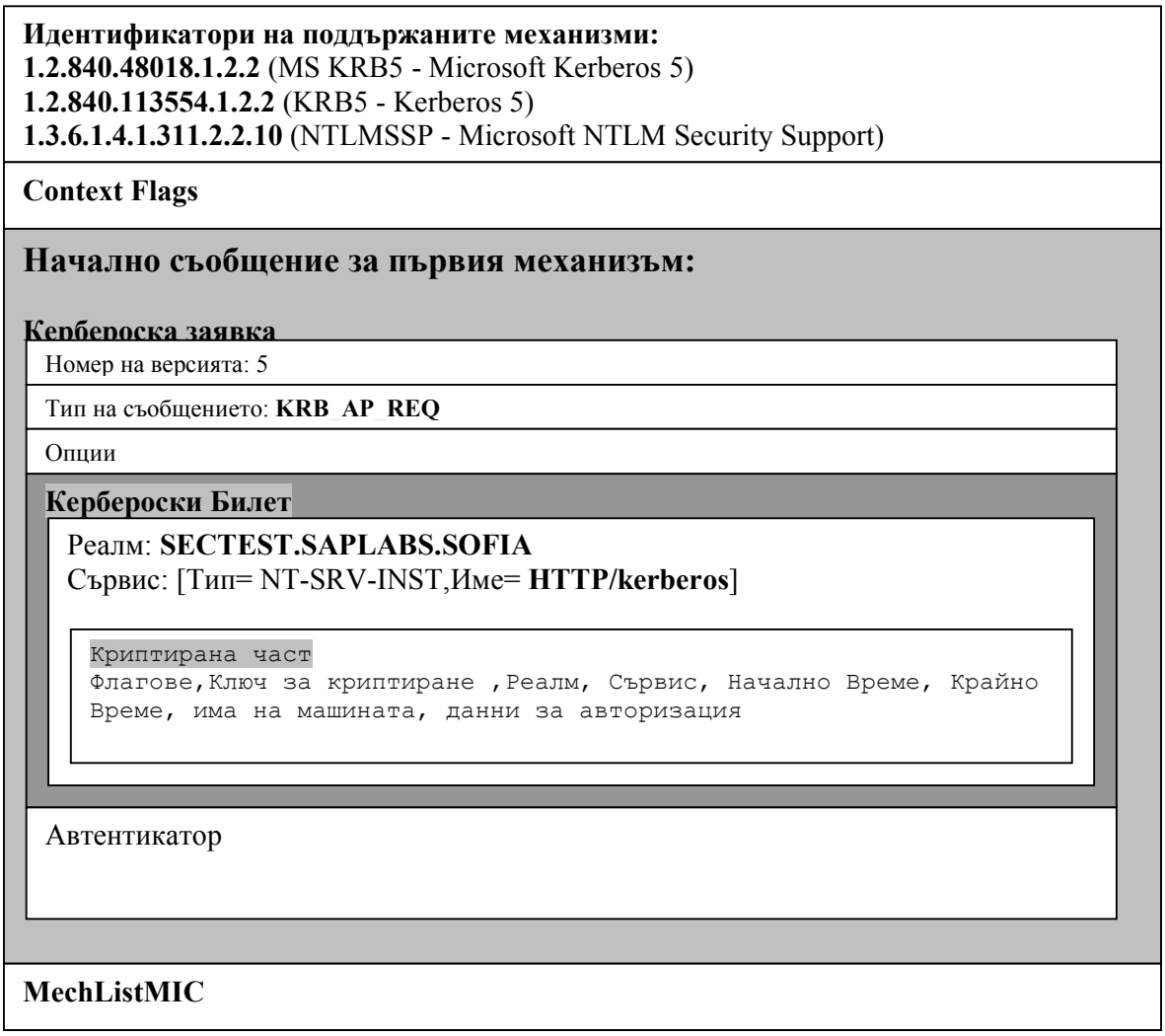
Помощните класове SPNegoInit и SPNegotTarg са едни от съществените за реализирането за стъпките от протокола SPNego. Както стана ясно в описанието на фаза 2 от изпълнението на метода login(), инстанции на тези 2 класа конструираме като от низа подаден на метода login()

във вид на HTTP хедър, приложим декодиране Base64 и след това конструираме обект ASN.1. От получения обект извличаме нужните полета от структурата и формираме съответната интанция на SPNegoInit или SPNegoTarg. Фигурите 19 и 20 показват структурата на обектите от тези 2 класа[11].

5.2.2.1 SPNegoInit

Обект SPNegoInit се конструира само при първото съобщение (първия хедър) от браузъра. В него като най-важни посочваме списъка от поддържаните механизми от клиентската машина и началното съобщение за първия механизъм. Когато списъкът от поддържаните механизми от предложени от браузъра и списъкът от механизми, които поддържа сървърът, нямат общ елемент, то тогава процесът на договаряне пропада и автентикацията не може да се извърши. В случай, че сървърът иска да автентицира потребителя с първия механизъм от списъка, тогава за целта сървърът използва, полето от структурата „начално съобщение за първия механизъм". То се изпраща още с първото съобщение за да не се налага още една допълнителна размяна на съобщения между браузъра и сървъра с единствена цел да се достави това съобщение. Когато първият механизъм за автентикация е Керберос, тогава „началното съобщение" представлява Кербероска заявка. При декодирането на Кербероската заявка конструираме обект от класа KerberosApReq. Част от Кербероската заявка е Кербероският билет, за който използваме помощният клас KerberosTicket. Фигура 19 показва структурата на обект от класа SPNegoInit с вложените обекти от класовете KerberosApReq и KerberosTicket.

Структура на SPNegoInit



Фигура 19: Схема на SPNegoInit с вложени Кербероска заявка и Кербероски билет[11].

Фигурите 44,45,46 от глава Приложение показват в детайли реални обекти от този тип.

5.2.2.2 SPNegoTarg

Обект SPNegoTarg се конструира, когато SPNegoInit не е достатъчен за установяване на контекст. Съобщение съдържащо тази ASN.1 структура се създава първоначално от сървъра и се изпращат към браузъра, а в последствие браузърът изпраща в новия хедър следващо съобщение от този тип. Когато сървърът не иска да извърши автентикация с предложения първи механизъм, то той(сървърът) изпраща закодиран обект SPNegoTarg, който единствено указва на браузъра за кои механизъм иска начално съобщение. В обекта получен от следващия

хедър изпратен от браузъра ще можем да видим отново идентификатора на механизма и началното му събщение. Фигура 20 показва структурата на обект от класа SPNegoTarg.

Структура на SPNegoTarg

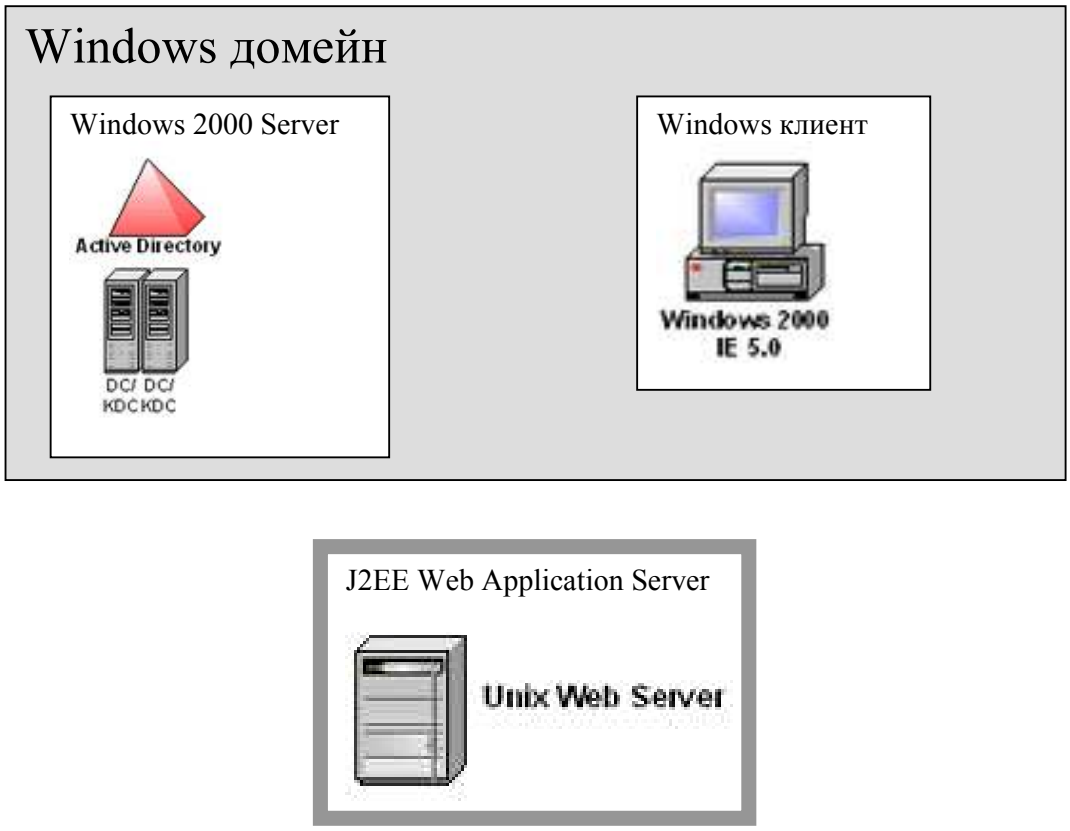
Резултат от договарянето **Завършен или Незавършен или Отхвърлен**
Поддържан Механизъм: 1.2.840.48018.1.2.2 (MS KRB5 - Microsoft Kerberos 5)
Съобщение-Отговор
MechListMIC

Фигура 20: Схема на SPNegoTarg[11].

Фигурите 50,51,52,53 от глава Приложение показват реални обекти от този тип.

6. Конфигуриране и тестване

В тази глава ще разгледаме настройките необходими за използването на SPNegoLib и SPNegoTestApp. Както беше показано вече, автентикацията с протокола Керберос включва три участника – KDC, Сървър за приложения и Клиентска машина. В нашата конкретна реализация ще използваме тестов Windows Домейн. За да изпълним стъпките необходими за конфигуриране на домейн контролера трябва да имаме администраторски достъп до съответния домейн контролер. От съображения за сигурност в повечето случаи само системните администратори имат такива права. Чрез настройките, които ще направим ще позволим на потребителите от домейна да достъпват тестовото приложение работещо на сървъра. Сървърът за приложения, който ще използваме е J2EE Web сървър за приложения. На фигура 21 схематично са показани трите машини използвани в сценария. По-конкретно в тази реализация използваме SAP AS Java, като възможно най-много ще се дистанцираме от този факт и ще изложим описанието във вид, такъв че без усложнения да можем да използваме реализирания сценарий и за сървъри за приложения разработени от други производители.



Фигура 21: Трите машини участващи в механизма за интегрирана автентикация

Фигура 21 изобразява трите машини имащи отношение към нашата реализация.

- Домейн Контролер настройван от системни администратори на мрежата/домейна, Windows 2000/2003 Server. Изпълнява ролята на Key Distribution Center.
- Клиентска машина с операционна система Windows. Браузър InternetExplorer 5.0 или по-нов. Клиентската машина е регистрирана в домейна и потребителят, който работи на тази машина също е регистриран и работи на машината като потребител в домейна, а не като локален потребител.
- Web Application Server(SAP AS Java 640) работи на машина с операционна система Unix, която не е част от Windows домейна. Операционната система е различна от Windows за да покажем, че сървърът за приложения може да работи на произволна операционна система. Със същата идея сървърската машина е извън домейна за да се покаже, че сценарият позволява това. Ясно е, че в най-опростеният случай сървърът за приложения може да работи на друга клиентска машина добавена в домейна и работеща на Windows.

6.1 Конфигуриране на домейн контролера

Настройките, които трябва да се изпълнят на домейн контролера в общия случай изискват администраторски потребител. В повечето случаи такива са само системните администратори. От характеристиките на механизма за автентикация е ясно, че потребителите, които ще използват автоматична автентикация трябва да са регистрирани като потребители на домейн контролера. Но също така е необходимо да бъде създаден и конфигуриран още един специален потребител.

6.1.1 Конфигуриране на сървис потребител

Процесът на Керберос автентикация използва KDC(в случая Домейн Контролер), за да автентицира клиента при първоначален „логон” на клиентската машина. За да се издаде Кербероски билет за комуникация между сървъра и браузъра, както и за да успее след това сървърът да извлече името на потребителя от SPNego съобщението, трябва да съществува регистрация на сървъра като потребител в домейна[10]. Този потребител представя сървъра като част от домейна, макар че физически той може и да не е в домейна както е в нашия пример. Този специален потребител трябва да изпълнява следните условия:

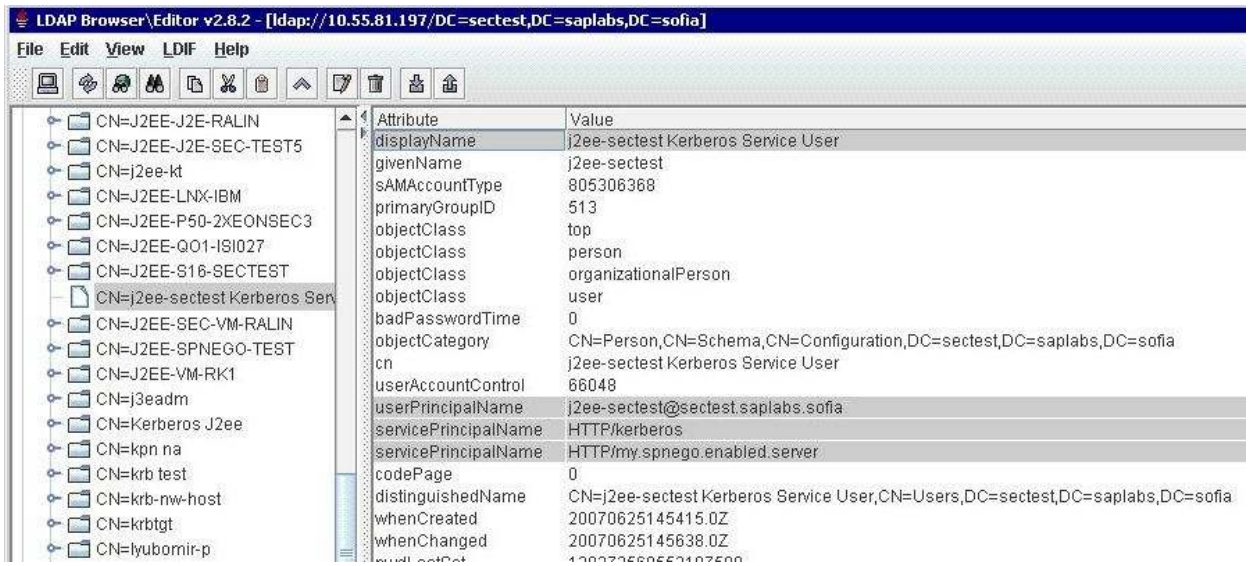
- паролата на потребителя не трябва да губи своята валидност никога
- потребителят трябва да използва DES кодиране

- за всяко едно от DNS имената, които могат да се използват за достъп до J2EE Сървъра трябва да се регистрира Service Principal Name (SPN) атрибут за служебния потребител. За целта се използва командата **setspn**.

Например ако сървис потребителят има аттрибут samaccountname **j2ee_spnego** и изпълним следните команди:

```
setspn –A HTTP/kerberos j2ee_sectest
setspn –A HTTP/my-spnego-enabled-server j2ee_sectest
```

Като резултат имената **kerberos** и **my-spnego-enabled-server** ще са регистрирани като SPN-и и асоциирани с потребителя за J2EE Сървъра създаден в Windows Домейн Контролера. В последствие от клиентския браузър потребителят ще може да използва URL http://**kerberos** или http://**my-spnego-enabled-server** и ще може да използва механизма SPNego, но няма да може да използва автоматична автентикация чрез URL http://<IP адрес> или http://<alias>. Това е така, защото браузърът ще направи заявка към домейн контролера за Кербероски сесиен билет за комуникация с потребителя, който има servicePrincipalName **HTTP/<URL адреса>.** На фигура 22 е показан списъкът от атрибути на j2ee-sectest като е използвана помощната програма LDAPBrowser. По-важните атрибути са маркирани с тъмен цвят.



Фигура 22: userPrincipalName и servicePrincipalName на сървис потребител

6.1.2 Генериране на keytab файл

Използване на ktab

Този файл може да бъде генериран посредством стандартна малка Java програма ktab, която се предоставя стандартно с дистрибуциите на JDK версия 1.4.2 или по-нови.

Тя има следния синтаксис:

ktab [-a <*principal_name*> <*password*>] [-k <*keytab_name*>]

в нашия случай изпълнението е следното:

ktab -a j2ee_sectest@SECTEST.SAPLABS.SOFIA <парола> -k my_keytab_file

Тъй като този подход изисква единствено стандартна версия на JDK, то е ясно че тази стъпка може да бъде изпълнена на всяка машина с JDK, не е задължително да е на Домейн Контролера. Поради по-голямата „универсалност" на този механизъм в доста случаи е по-удобно да се използва тази команда, а не командата ktpass описана по-долу.

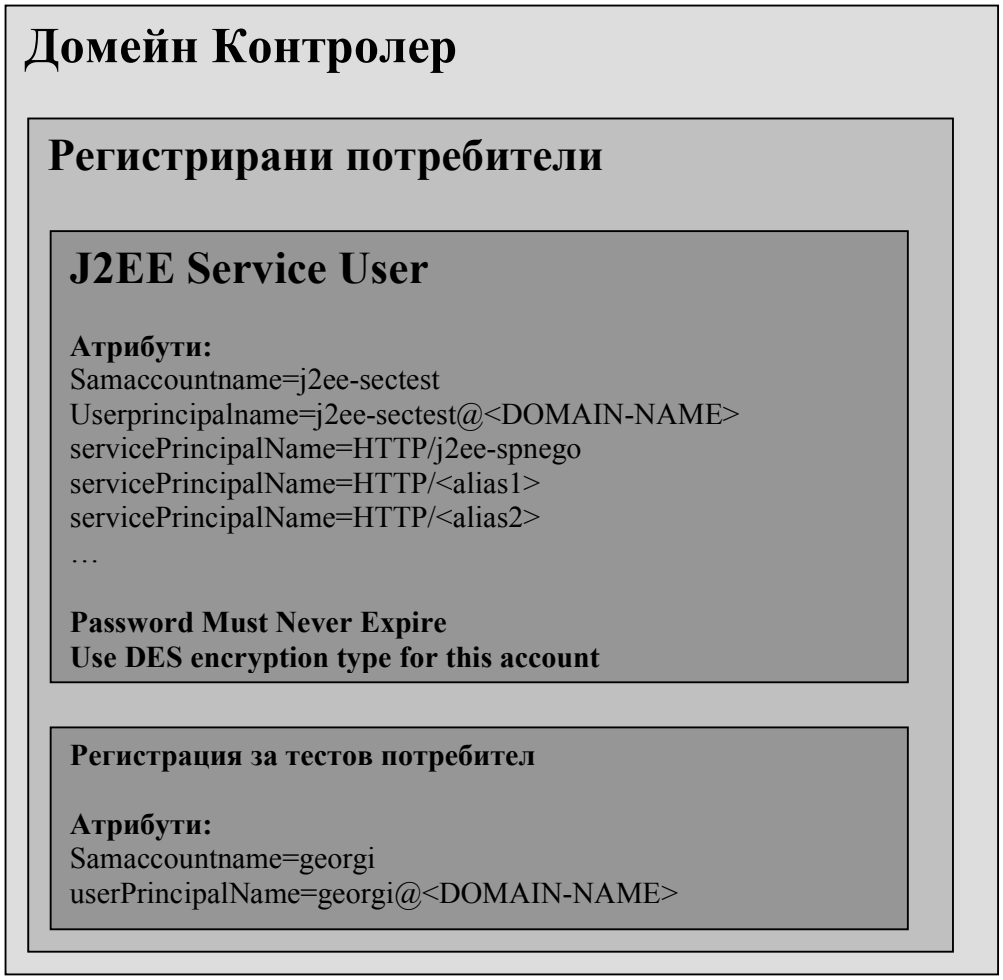
Използване на ktpass

Друг един подход е използването на командата предоставена от Microsoft – ktpass. Тя се изпълнява задължително на Домейн Контролера от потребител с администраторски права. Една примерна употреба има следния вид:

ktpass -princ HTTP/my-spnego-test@MY.KERBEROS.DOMAIN
-pass <парола> –out **my.kerberos.keytab** -mapUser
_spnego +DesOnly /crypto DES-CBC-MD5 /ptype KRB5_NT_PRINCIPAL

като резултат от тази операция се създава keytab файл с име **my.kerberos.keytab,** който свързва Service Principal Name HTTP/my-spnego-test@MY.KERBEROS.REALM с потребителя, който има атрибут samaccountname j2ee_spnego.

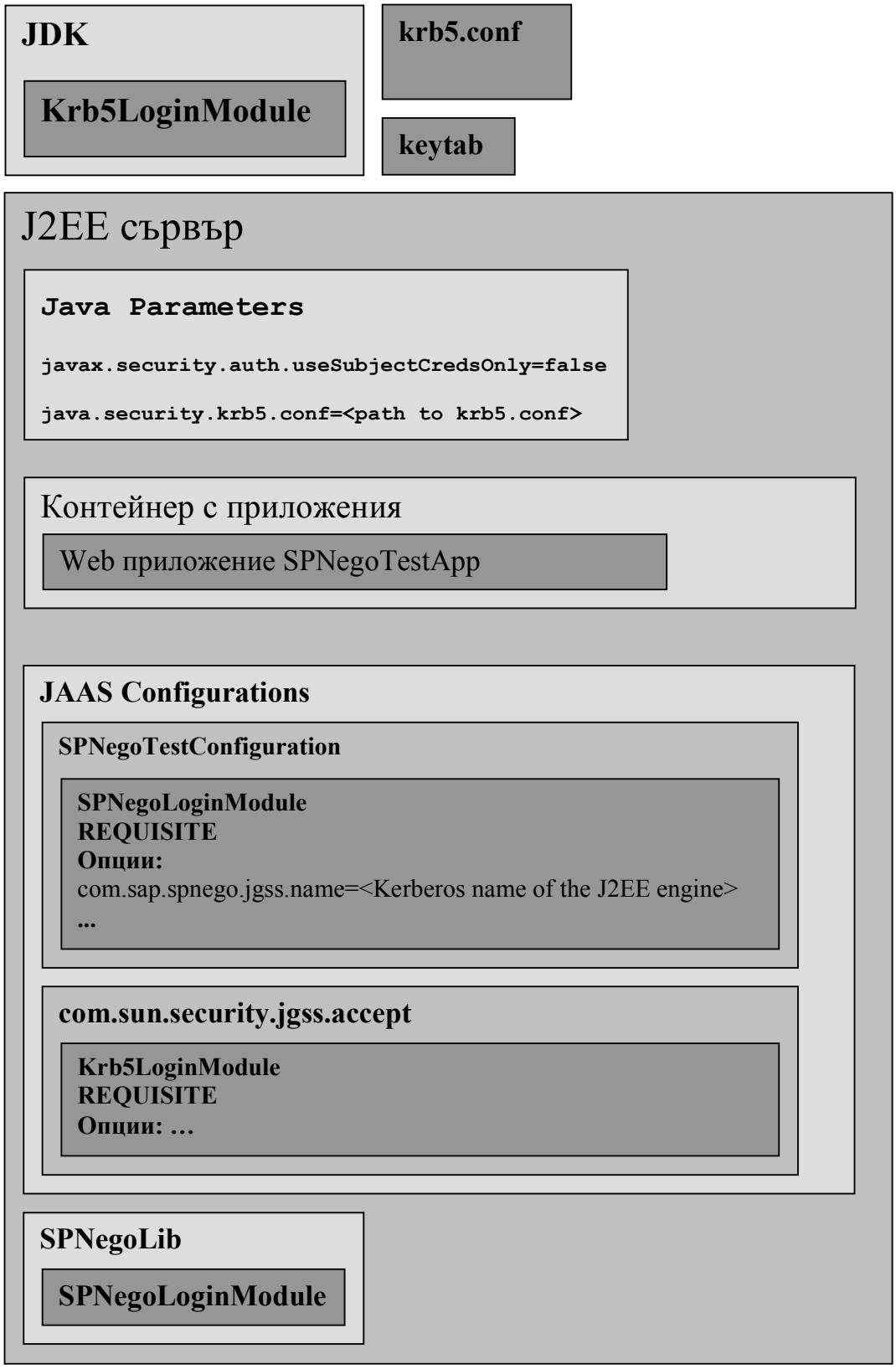
Понеже получения keytab файл в последствие трябва да се запише на файловата система на сървъра и понеже не всички могат да изпълняват тази администраторска команда, то като заключение можем да кажем, че е по-добре да генерираме keytab файла направо на машината на сървъра, където е ясно че има JDK. Фигура 23 показва общия вид на настройките на един тестов домейн контролер.



Фигура 23: Конфигуриране на домейн контролера

6.2 Конфигуриране на сървъра за приложения

Фигура 24 представя основните компоненти, които изискват настройки за да можем да използваме интегрирана автентикация за приложенията работещи на сървъра.



Фигура 24: Схема на основните компоненти и конфигурации на J2EE сървъра

Както стана ясно вече физически самата сървърска машина може да се намира извън домейна. Също така операционната система може да е различна от Windows. Освен инсталиран Java Development Kit (JDK) и работещ сървър за приложения други начални изисквания няма. (Както казахме, в нашия пример ще използваме инсталация на SAP AS Java 640) Следва поредицата от стъпки, които трябва да се изпълнят, заедно е разясненията към всяка една от тях. Подредбата на изпълняване на отделните стъпки не е задължителна, но точно ред на описването им е избран за да се улесни разбирането на необходимостта от всяка една отделна стъпка.

Стъпка 1: Инсталиране на тестовото приложение SPNegoTestApp

За целите на нашата демонстрация ще използваме тестово приложение, което е разработено и подготвено за инсталация (deploy) във вид на "ear" файл. (spnegoTest.ear).

Стъпка 2: Създаване на SPNegoTestConfiguration

За SPNegoTestApp трябва да зададем механизма за автентикация. За нашите цели това приложение ще използва JAAS конфигурация SPNegoTestConfiguration, чрез която на практика осъществяваме връзката между приложението и модулът за автентикация[13]. Въпросната JAAS конфигурация трябва да съдържа SPNegoLoginModule. Видът на конфигурацията е показан на фигура 25. Използваният флаг е "REQUISITE", макар че като единствен логин модул той може да използва какъвто и да е друг флаг – това няма да промени резултата. Опцията, която използваме е **com.sap.spnego.jgss.name=<Kerberos name of the J2EE engine>.** В нашия случай стойността на това поле ще е j2ee_sectest@SECTEST.SAPLABS.SOFIA,... или по-точно използваме атрибута **userPrincipalName** на сървис потребителя. Ако искаме да автентицираме потребителите от няколко домейна, трябва да добавим имената на сървис потребителите за всички домейни със запетая.

JAAS конфигурация SPNegoTestConfiguration за приложението SPNegoTestApp

Login Modules	Flag	Options
SPNegoLoginModule	REQUISITE	com.sap.spnego.jgss.name = j2ee_spnego@<DOMAIN-NAME>, j2ee_spnego_2@<DOMAIN-NAME-2>

Фигура 25: Употреба на SPNegoLoginModule в стек за автентикация

Стъпка 3: Инсталиране на библиотеката SPNegoLib

За да има смисъл SPNegoTestConfiguration, то трябва SPNegoLoginModule да съществува на сървъра. Самият модул за автентикация SPNegoLoginModule заедно с всички помощни класове, които той използва са подготвени за използване на сървъра като отделна библиотека Spnegolib.sda (Software Development archive), която трябва да се инсталира. До тук описаните стъпки са стандартни за произволно приложение, което иска да използва логин модул предоставян като част от някаква библиотека.

Стъпка 4: Създаване на JAAS конфигурация com.sun.security.jgss.accept

За конкретните нужди на Керберос автентикацията се налага да създадем още една JAAS конфигурация със специалното име **com.sun.security.jgss.accept**[6]. Тя се използва от SPNegoLoginModule. Тази JAAS конфигурация използва логин модула предоставен от доставчикът на JDK. В случай на SUN JDK специалният логин модул се казва **com.sun.security.auth.module.Krb5LoginModule**[9]**.** (За IBM JDK името на съответния логин модул е **com.ibm.security.auth.module.Krb5LoginModule,** но в случая с IBM не е нужно създаването на такава JAAS конфигурация.) Този логин модул се използва за да автентицира сървъра (и по конкретно специалния потребител, чийто атрибут **userPrincipalName** е зададен в опциите на SPNegoLoginModule на фигура 25) пред домейн контролера като използва keytab файла. Това е keytab файла, който показахме как се създава, когато разгледахме стъпките по конфигурирането на сървис потребителя. Фигура 26 показва пълния набор от опции за Krb5LoginModule използвани в **com.sun.security.jgss.accept**.

JAAS конфигурация com.sun.security.jgss.accept

Login Modules	Flag	Options
Krb5LoginModule	REQUISITE	debug=true, doNotPrompt=true, storeKey=true, useKeyTab=true, useTicketCache=true

Фигура 26: com.sun.security.jgss.accept и Krb5LoginModule.

Стъпка 5: Задаване на Java System Properties

От своя страна KerberosLoginModule използва специални Java System Properties, които трябва да зададем:

- javax.security.auth.useSubjectCredsOnly=false
- java.security.krb5.conf=<пълен път до файла **krb5.conf**>

Допълнително могат да се използват и следните Java System Properties, които ако са зададени както е показано малко по-долу ни позволяват да получаваме подробна информация от извикването на класовете от GSSAPI и в случаи на грешки да ни улеснят при разследването.

- В случай на SUN JDK

```
sun.security.krb5.debug=true
```

- В случай на IBM JDK

```
com.ibm.security.jgss.debug=all
com.ibm.security.krb5.Krb5Debug=all
```

Стъпка 6: Създаване на krb5.conf файл

На предната стъпка видяхме, че се използва krb5.conf файл, който има примерното съдържание показано на фигура 27.

Общ вид на krb5.conf файл

```
[domain_realm]
     <DNS домейн> = <Kerberos реалм с големи букви>

[libdefaults]
     default_keytab_name = <пълен път до файла keytab>
     default_realm = <Kerberos реалм с големи букви>
     dns_lookup_kdc = true
     default_tgs_enctypes=des-cbc-md5;des-cbc-crc
     default_tkt_enctypes=des-cbc-md5;des-cbc-crc

[logging]

[realms]
     <Kerberos реалм с големи букви>= {
     admin_server = <IP на домейн контролера>
     kdc = <IP на домейн контролера>
     }
```

Фигура 27: Съдържание на krb5.conf файла.

За да стане малко по-ясно нека приемем че нашият тестов домейн се казва SECTEST.SAPLABS.SOFIA, сървърът за приложения се намира в домейна SAPLABS.SOFIA с пълно име j2ee_server.saplabs.sofia, а домейн контролерът има IP адрес 77.66.55.44, а пътят до keytab е /home/security/Kerberos/keytab. Тогава горният формат приема вида показан на фигура28:

```
[domain_realm]
     .saplabs.sofia = SECTEST.SAPLABS.SOFIA

[libdefaults]
     default_keytab_name = /home/security/Kerberos/keytab
     default_realm = < SECTEST.SAPLABS.SOFIA
     dns_lookup_kdc = true
     default_tgs_enctypes=des-cbc-md5;des-cbc-crc
     default_tkt_enctypes=des-cbc-md5;des-cbc-crc

[logging]

[realms]
      SECTEST.SAPLABS.SOFIA {
     admin_server = 77.66.55.44
     kdc = 77.66.55.44
     }
```

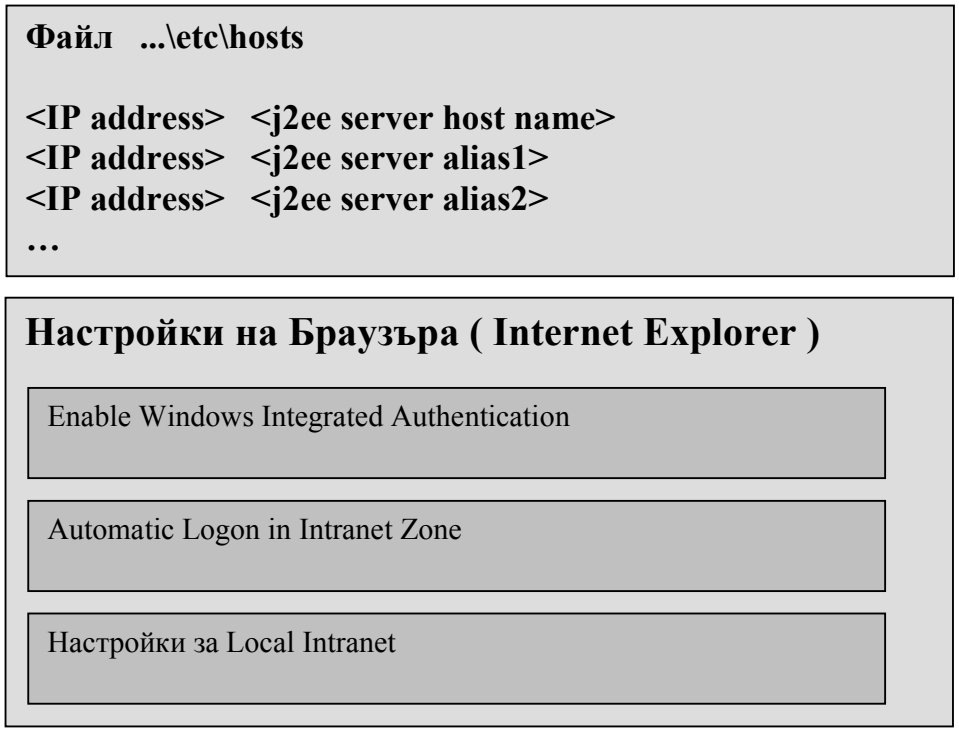
Фигура 28: Съдържание на krb5.conf файла използван в нашия тест.

Особености при използване на IBM JDK

GSSAPI за IBM JDK е реализиран, така че да използва единствено Java System Properties. На практика цялата информация нужна за автентикацията на J2EE сървъра пред домейн контролера се намира в krb5.conf файла. За разлика от SUN, IBM използва com.ibm.security.auth.module.Krb5LoginModule, но не се налага да се създават допълнителни JAAS конфигурации.

6.3 Конфигуриране на клиентската машина

Фигура 29 показва схематично настройките необходими които трябва да бъда изпълнени на клиентската машина. След това следват по-подробни инструкции.



Фигура 29: Настройки на клиентската машина

6.3.1 Добавяне на допълнителни имена за сървъра

В общия случай трябва клиентската машина да може да разпознава по някакъв начин името с което ще се обръщаме към сървъра. За нашите цели това най-лесно можем да направим като редактираме ...\etc\hosts файла. За версиите на Windows той се намира в C:\WINDOWS\system32\drivers\etc, като тук предполагаме, че операционната система е инсталирана на устройство C.

Например ако искаме да използваме машината j2ee-server.saplabs.sofia с две допълнителни имена (alias) my-spnego и my-kerberos-enabled-server и ако тя има IP адрес 10.55.70.85 то във файла добавяме

```
10.55.70.85   j2ee-server.saplabs.sofia
10.55.70.85   my-spnego
10.55.70.85   my-kerberos-enabled-server
```

6.3.2 Конфигуриране на браузъра

6.3.2.1 Конфигуриране на InternetExplorer

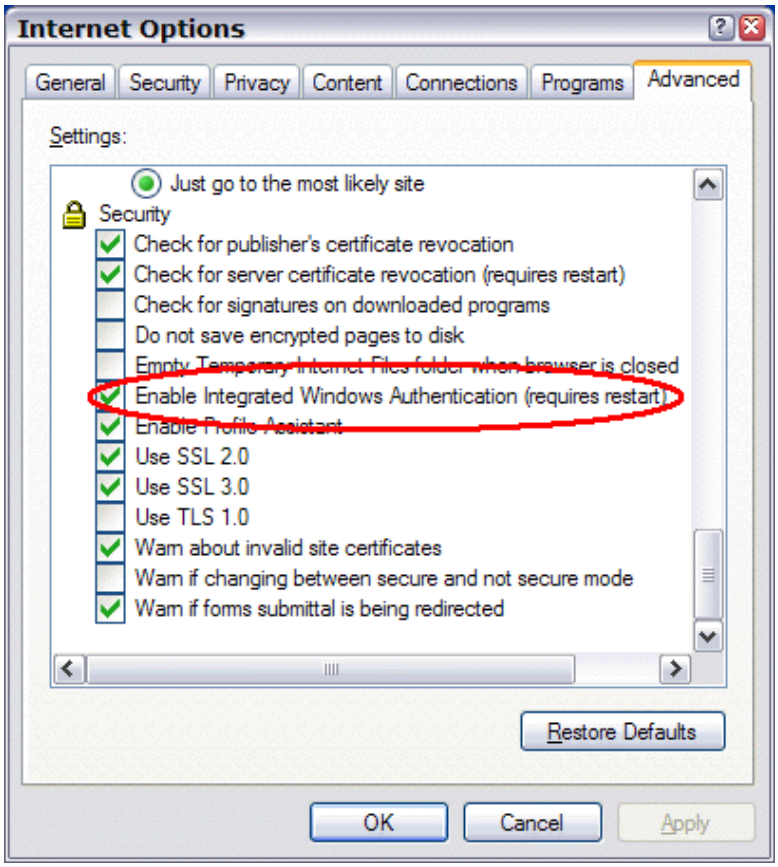
Стъпките за конфигуриране на Internet Explorer за да използва Керберос автентикация към J2EE Сървъра са следните[10]:

- **<u>да се позволи Windows Integrated Authentication</u>**.

За InternetExplorer това се задава така:

Процедура: Меню „Tools” ➔ „Internet Options…” ➔ таб „Advanced” ➔ Група „Security” ➔ Избираме „Enable Windows Integrated Authentication(requires restart)”

Както се вижда от наименованието на това поле, браузърът трябва да се рестартира преди да тестваме интегрирана автентикация.



Фигура 30: Включване на Windows Integrated Authentication

- **<u>да се позволи автоматично влизане в интранет зоната.</u>**

За InternetExplorer това се задава така:

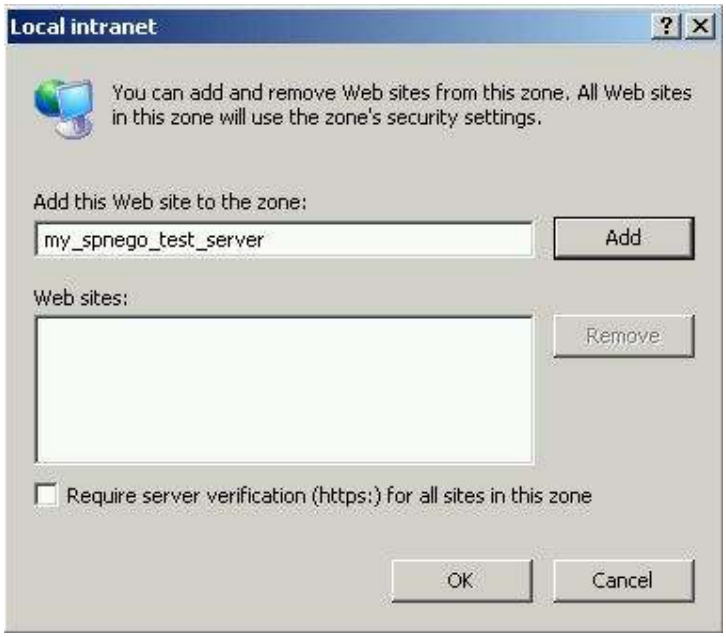
Процедура: Меню „Tools” ➔ „Internet Options” ➔ Таб „Security” ➔ „Local Intranet” ➔ Бутон „Custom Level” ➔ Избираме “Automatic logon only in Intranet Zone”

Фигура 31: Разрешаване на автоматично влизане в интранет зоната

- **<u>Да се добави DNS името на машината, на която се намира J2EE Сървъра към списъка на локалните интранет сайтове</u>**.

За InternetExplorer това се задава така:

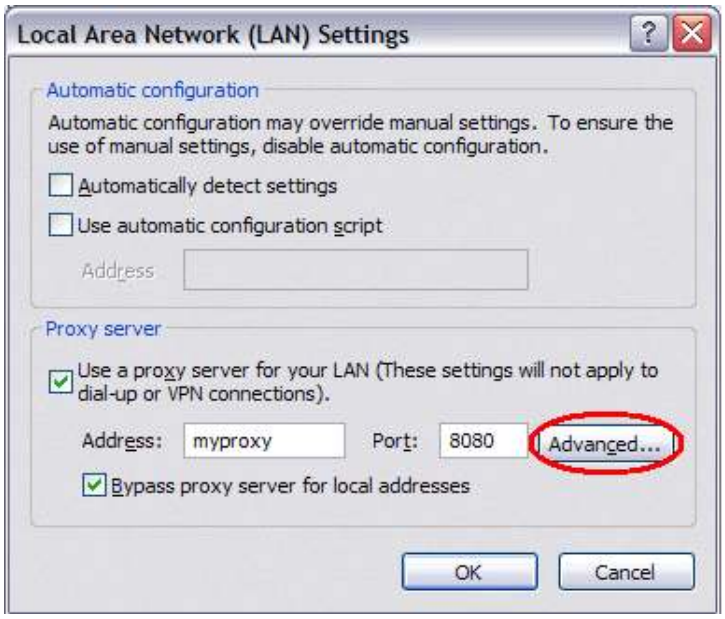
Процедура: Меню „Tools” ➔ „Internet Options” ➔ Таб „Security” ➔ „Local Intranet” ➔ Бутон “Sites” ➔ Бутон „Advanced” ➔ Добавяме името на машината където работи J2EE Сървъра към списъка със сайтове



Фигура 32: Добавяне към интранет зоната

- **<u>Да се добави IP адреса на сървъра към списъка от сайтове, за които да не се използва прокси (в случай че се използва такова)</u>**

Процедура: Меню „Tools” ➔ „Internet Options” ➔ Таб „Connections” ➔ „Local Area Network (LAN) Settings” ➔ Бутон “LAN Settings…” ➔ Избираме „Bypass proxy server for local addresses” ➔ Бутон “Advanced” ➔ Добавяме IP адреса на сървъра.



Фигура 33: Отказ от прокси сървър за локалните сайтове

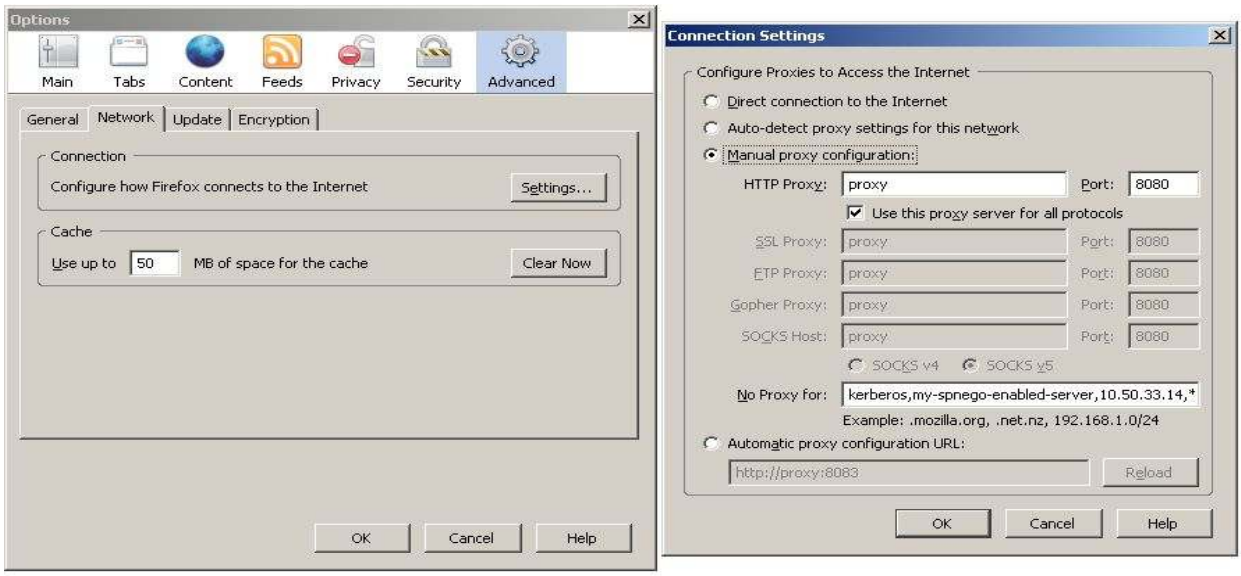
Фигура 34: Добавяне на сървъра към списъка на локалните сайтове.

6.3.2.2 Конфигуриране на Mozilla Firefox

Когато се конфигурира Mozilla процедурата е доста по-опростена[12]:

- **Да се добави адреса на сървъра към списъка от сайтове, за които да не се използва прокси (в случай че се използва такова)**

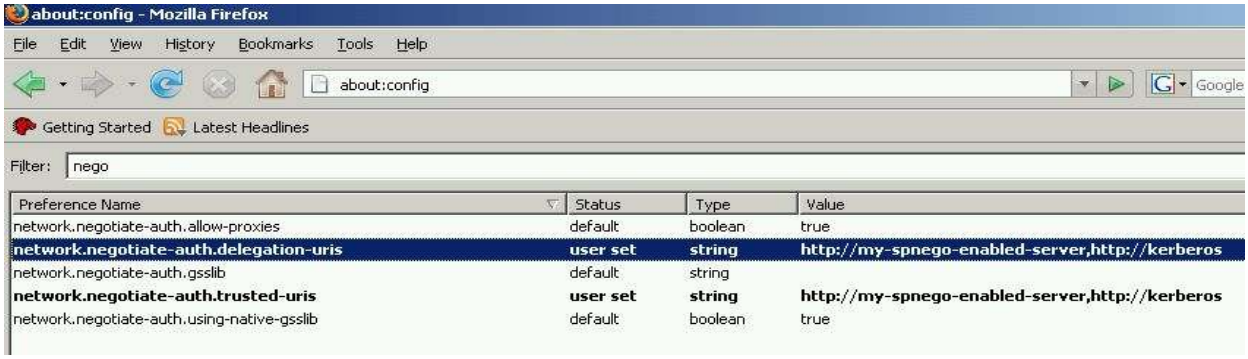
Процедура: Меню „Tools" ➔ „Options" ➔ Таб „Advanced" ➔ Таб „Network" ➔ Бутон "Settings" ➔ В полето „No proxy for:" добавяме адреса на сървъра.



Фигура 35: Добавяне на сървъра към списъка на локалните сайтове.

- **Да се разреши интегрирана автентикация**

Процедура: За целта се използва специалната страница с настройки, която се извиква като се въведе в полето за URL **"about:config"**. Филтрираме настройките по име, използваме префикса „nego". За **опциите network.negotiate-auth.delegation-uris и network.negotiate-auth.trusted-uris** въвеждаме http://<адреса на сървъра>. Ако искаме да разрешим интегрирана автентикация без ограничение на сайтовете, тогава просто указваме http:// без да задаваме адрес.



Фигура 36: Включване на интегрирана автентикация за Mozilla.

6.4 Използване на SPNegoTestApp

6.4.1 Успешен опит за автентикация

На фигура 37 е показан резултатът от извикването на SPNegoTestApp. Вижда се името на потребителя, който успешно е автентициран с SPNego. В случая името на потребителя е "**georgi-di**". Потребителят е използвал браузър Mozilla Firefox. В полето за интернет адреса виждаме, че е използвано името **„kerberos**" (т.е. servicePrincipalName=HTTP/kerberos).

Фигура 37: Успешно автентикация на потребителя с SPNego.

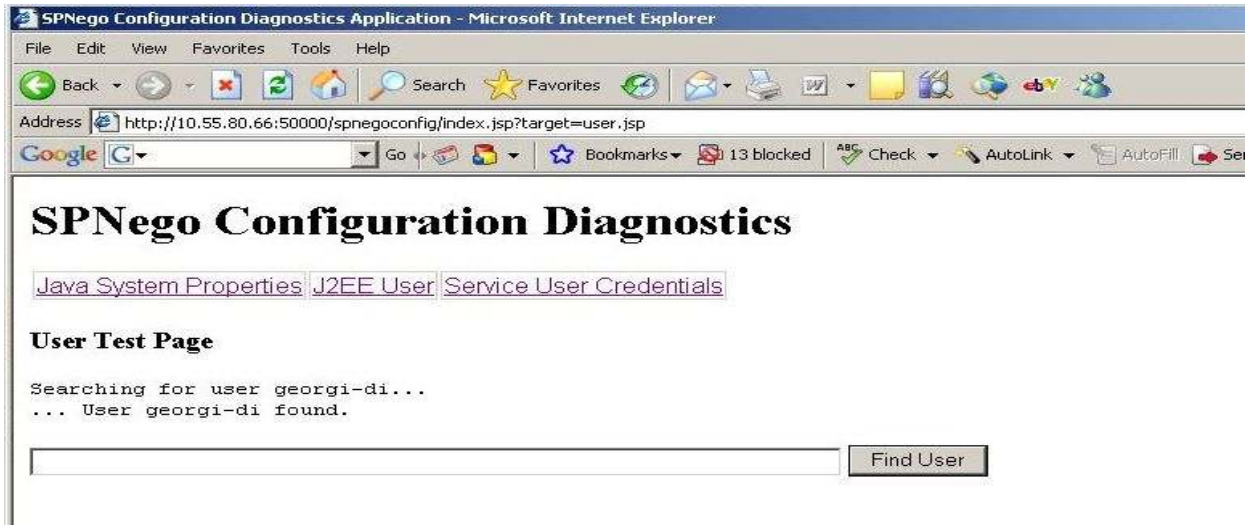
6.4.2 Неуспешен опит за автентикация

На фигура 38 се вижда резултата от неуспешен опит за автентикация с SPNego. Тук използваният адрес е "spnego_error". Този псевдоним е добавен във файла „hosts", браузърът открива сървъра, но понеже няма регистриран сървис потребител с атрибут servicePrincipalName=HTTP/spnego_error опита за автентикация не може да успее.

Фигура 38: Неуспешна автентикация на потребителя с SPNego.

6.4.3 Проверка за съществуването на потребителя

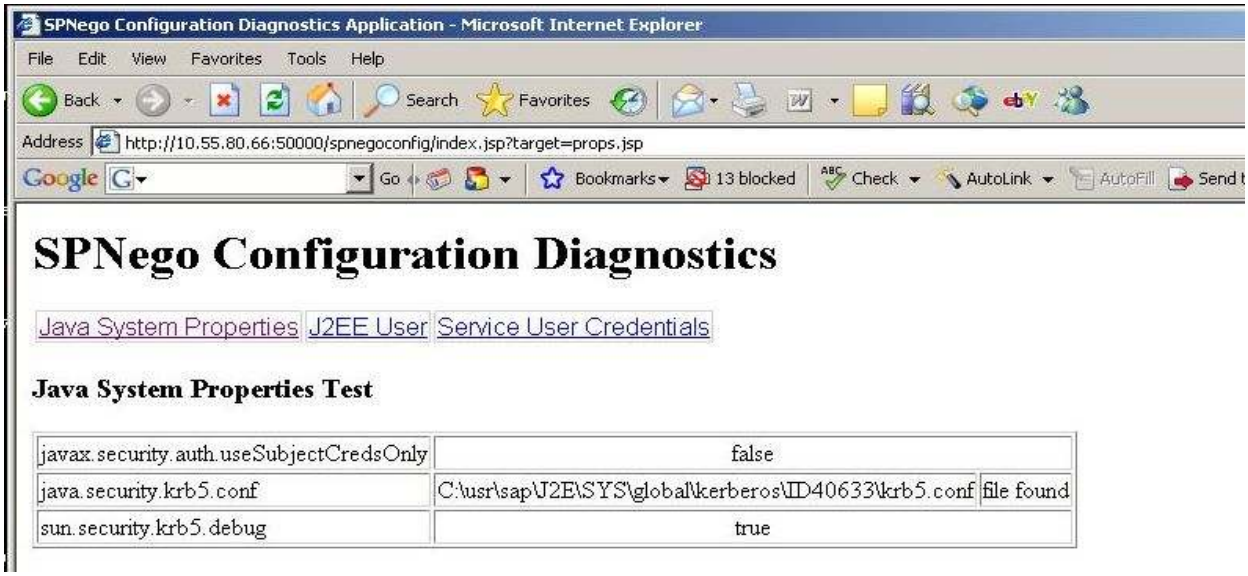
В случай, че не успее да се автентицира потребителя можем да проверим дали сървърът вижда потребител с такова име в неговата система с регистрирани потребители. Фигура 39 показва резултата от успешното откриване на потребителя georgi-di.



Фигура 39: Успешно откриване на потребителя в списъка от регистрирани потребители.

6.4.4 Проверка на Java System Properties

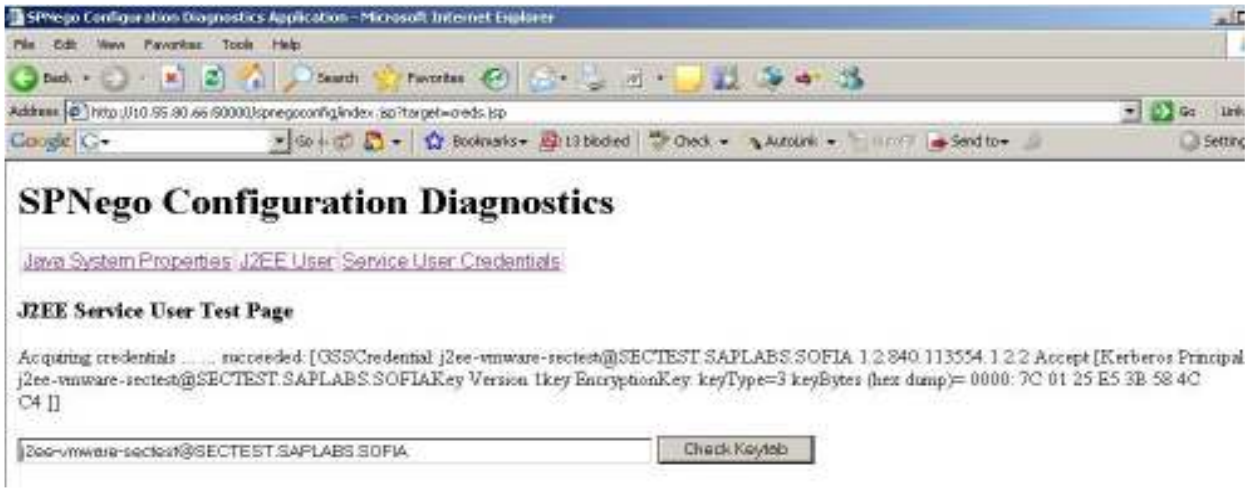
Ако потребителят съществува тогава можем да проверим дали правилно са настроени Java System Properties. В случай на неправално зададени стойности ще можем лесно да открием това със страницата Java System Properties на тестовото приложение. Фигура 40 ни показва как изглежда тази страница.



Фигура 40: Стойностите на Java System Properties свързани с SPNego.

6.4.4 Проверка на файла keytab

Тестовото приложение ни позволява да проверим коректността на файла keytab. Това правим като за даден сървис потребител проверим дали можем посредством указания файл keytab да получим автентикационни данни от домейн контролера. Фигура 41 ни показва успешен тест за примерен сървис потребител j2ee-vmware-sectest@SECTEST.SAPLABS.SOFIA. В текстовото поле се въвежда името на сървис потребителя по начина, по който този потребител се задава като опция на SPNegoLoginModule – въвежда се атрибута userPrincipalName. В случай, че конфигурираме сървъра за автентикация на потребителите от няколко домейна, можем да проверим отделните сървис потребители за различните домейни. На фигура 41 виждаме успешното получаване на автентикационни данни за сървис потребителя. Като допълнение виждаме резултата от извикването на метода toString() за получения обект GSSCredential. В случай, че файлът keytab е невалиден, има правописна грешка при въвеждането на името на сървис потребителя или някакъв друг проблем получаваме резултата показан на фигура 42.



Фигура 41: Успешно получаване на автентикационни данни.

Фигура 42: Неуспешно получаване на автентикационни данни.

В случай, че и трите страници на помощното приложение ни покажат, че настройките са правилни, това означава че сървърът е правилно настроен да използва SPNego автентикация. Ако и трите тестови страници ни покажат, че съответните настройки са коректни, то автентикацията би трябвало да премине успешно. Ако все пак потребителят не може да се автентицира при достъп до SPNegoTestApp, то значи проблемът е на машината на клиента.

7. Проблеми и насоки за развитие

Съществуват различни причини поради, които описаният механизъм за автентикация на базата на Керберос и SPNego не е удобен за ползване или създава определени проблеми. Те са предизвикани от факта, че процедурата по конфигуриране на трите машини участващи в механизма за автентикация е доста сложна и има много възможности за грешки при отделните стъпки. Дори когато основните правила описани в главата "Конфигуриране и тестване" са спазени, няма гаранция, че в последствие автентикацията няма да спре да работи.

7.1 Проблем със сървис потребителя

Нека разгледаме случай, в койтоизползваме сървис потребител с атрибути

userPrincipalName = myserviceuser@DOMAIN

servicePrincipalName=HTTP/kerberos_test

Ние очакваме, че когато въведем в полето за интернет адрес http://kerberos_test:50000/spnegoTest, процесът на автентикация ще направи опит да открие потребителя с атрибут servicePrincipalName=HTTP/kerberos_test, ще намери myserviceuser@DOMAIN и ще се опита да го автентицира пред домейн контролера с файла keytab генериран на базата на паролата, която ние сме задали първоначално. Но ако някой създаде друг сървис потребител със същия servicePrincipalName, то myserviceuser няма да бъде открит. Същия резултат ще се получи ако myserviceuser бъде изтрит от домейн контролера. И накрая ако е сменена паролата на myserveceuser, то нашият keytab файл ще се окаже безполезен. В тези случаи проблемът се счита за сравнително прост и той може да бъде открит като използваме помощното приложение за откриване на проблеми на сървъра за приложения.

7.2 Проблем с версията на JDK

Както стана ясно освен на коректността на SPNegoLoginModule, процесът на автентикация разчита и на доста други класове. Ако решим поради някаква причина несвързана с Керберос и SPNego, да използваме по-нова версия на GSSAPI,например като инсталираме по-ново JDK, то може да възникнат някои неочаквани проблеми. Например IBM IDK 1.4.2 SR7 генерира изключения при някои тривиални методи. По-конкретно ако използваме привидно безобидния код имащ за цел да изведе малко помощна информация без да се извършва нищо специфично

System.out.print(gssContext.toString()), то в посочената версия на JDK се получава неочаквано изключение.

```
org.ietf.jgss.GSSException, major code: 16, minor code: 0
major string: Operation unavailable or not implemented
minor string: Context not usable
```

В друг един случай подмяната на SUN JDK 1.4.2_13 със следващата версия 1.4.2_14 води до необратимо спиране на работата на така важния Krb5LoginModule. Получаваме

```
java.lang.NullPointerException
at java.lang.StringBuffer.append(StringBuffer.java:467)
at com.sun.security.auth.module.Krb5LoginModule.attemptAuthentication(Krb5LoginModule.java:576)
at com.sun.security.auth.module.Krb5LoginModule.login(Krb5LoginModule.java:475)
```

В такъв един случай най-простото решение е използването на версия на JDK, която е изчистена от такива странни изключения. Подмяна на използваното по-ново JDK към по-старо води до решение на проблема.

7.3 Проблем с клиентската машина

Други важни проблеми са свързани с клиентската машина. Както изяснихме след като сървърът заявки, че иска да използва Керберос за автентикация на потребителя, той очаква HTTP хедър съдържащ Кербероски билет. Тук отново разчитаме, че операционната система и браузърът ще работят според нашите очаквания. Оказва се обаче, че когато на Кербероския билет му изтече срока на годност, то в някои случаи операционната система не успява да вземе нов и тогава вместо SPNego браузърът изпраща NTLM хедър. Доказателство, че проблемът е в операционната система е фактът, че след рестартиране на клиентската машина, автентикацията отново сработва. В случаи като този единствен вариант е да се иска съдействие от Microsoft за разрешаване на проблема. За този конкретно проблем съществува hotfix KB899587.

Както показахме, файлът keytab може да се генерира посредством програмата ktpass предоставена от Microsoft. Оказва се обаче че в някои случаи има несъвместимост между генерирания формат на файла и различните версии на JDK. Тогава едно възможно решение е да се използва ktab от същата версия на JDK, която използва сървъра.

7.4 Използване на резервен механизъм за автентикация

Целта на реализираната библиотека и на тестовото приложения бяха да покажат как работи интегрирана автентикация с Керберос (SSO с Керберос) към дадено приложение. След като успеем да настоим цялата сложна система да работи по описания сценарий възниква въпросът как да използваме автентикация без SSO. Това може да се случи, например ако искаме на нашата машина за кратко да работи някой друг. Той ще използва нашия браузър и ще се автентицира автоматично без да може да използва своята регистрация на сървъра. В такива случаи трябва да се предвиди резервен вариант за автентикация. Това може да се постигне чрез използването на допълнителен логин модул в JAAS конфигурацията. Например логин модул за автентикация с потребителско име и парола или със сертификат. Необходимо е и още нещо. Трябва да предизвикаме грешка в процеса на SPNego, за да можем да използваме допълнителния логин модул (иначе преди да се активира допълнителният логин модул, потребителят автоматично ще се автентицира). За да постигнем целта има няколко подхода - да се използва IP адреса при достъп към сървъра (така ще се получи NTLM хедър) или временно да се изключи опцията "интегрирана Windows автентикация" (или някоя от другите настройки на браузъра). В най-стандартния сценарий потребителят иска да може да достъпва автоматично приложението от своята машина в домейна, а в случай, когато работи на машина извън домейна да може да достъпва сървъра без допълнителни промени. Тогава решението изисква единствено модифициране на стека от логин модули. Например използването на SPNegoLoginModule и хипотетичен логин модул за автентикация със сертификат ще позволят достъпа и в двата случая. Тук не е нужно да предизвикваме допълнителна грешка в SPNegoLoginModule тъй като ако заявката към сървъра е направена от машина извън домейна, браузърът няма да изпрати SPNego хедър.

Трябва да се обърне внимание на факта, че някои механизми също разчитат на HTTP хедъри и кодовете на HTTP отговора. В някои случаи е възможно даден механизъм за автентикация да не може да се използва заедно с SPNego заради такива конфликти. Например най-тривиалният механизъм за автентикация с потребителско име и парола използва HTTP хедър „WWW-Authenticate: Basic” задаван от сървъра и „Authorization: Basic QWxhZ...” изпращан от браузъра.

7.5 Реплициране на потребителите

Изяснихме, че при успешна автентикация сървърът ще открие името на потребителя във вида samaccountname@DOMAIN. В стандартния случай samaccountname (например ime.familia) е достатъчно уникално и ни позволява да открием коя регистрация на потребител на сървъра трябва да се използва. Но остава въпросът какво се случва ако имаме домейн контролер, на който се създават или премахват регистрации всеки ден или с по-голяма или по- малка честота. Един възможен подход е да се използва реплициране на потребителите на домейна като потребители и на сървъра. Възможно е в някои случаи сървърът да използва автоматично потребителите регистрирани в домейна като им задава някакви стандартни права. Например всички потребители регистрирани в домейна динамично се включват към група от потребители "DOMAIN". То този начин например ще можем много лесно да задаваме различни права на потребителите в зависимост от домейна, в който са регистрирани.

7.6 Насоки за развитие

Като се имат в предвид посочените дотук в тази глава проблеми и техни примерни решения можем да кажем, че най-голямо допълнително развитие на предложената реализация може да търси в областта на опростяване на конфигурационните стъпки и в добавянето на допълнителни функционалности подпомагащи откриването на възникнали проблеми. В дипломната работа се предлага една реализация свързана с един конкретен сървър за приложения. Поради тази причина се използват специфични класове, свързани с конкретната реализация на платформата JAAS. Биха могли да се разработят аналогични библиотеки и за други сървъри за приложения. Един примерен подход е да се реализира един абстрактен SPNegoLoginModule, който да дефинира задължителната функционалност, към която се стремих да се придържам в изложението на описаната тук реализация. Разработчиците на софтуер за различните сървъри ще имат задата да реализират абстрактните методи в зависимост от спецификите на техния сървър. Тези методи ще дефинират начините за извеждане на съобщения за грешка, връзката с контейнера за приложения ако не необходимо, получаването и задаването на стойности на хедърите на HTTP заявките и отговорите, съпоставянето на откритото име на потребител в домейна с реалната регистрация на потребител на сървъра и други.

8. Заключение

Автентикацията е критичен елемент в сигурността на една компютърна система. Традиционните методи за автентикация често не са подходящи за употреба в широки компютърни мрежи, където има възможност трафика да бъде следен от нарушител с цел откриване на пароли или други идентификационни средства. Употребата на криптографски алгоритми и протоколи в процеса на автентикация се налагат все повече с техническото и софтуерно развитие на комуникациите. В дипломната работа са описани основните характеристики на един от най-разпространените и използвани протоколи за автентикация. - Керберос. Широкото му разпространение го прави привлекателен за използване от много софтуерни продукти. В дипломната работа подробно са разгледани характеристиките на стандарта GSSAPI, благодарение на който Керберос става достъпен за употреба от програмистите чрез стандартните дистрибуции на JDK. Дипломната работа представя един начин за употреба на Керберос посредством протокола SPNego. Чрез разработената библиотека се предоставя възможност за използване на интегрирана автентикация на потребителите при достъп до приложенията от даден сървър. С цел тестване на функционалността и за откриване на проблеми по настройките на сървъра е разработено и тестово приложение, чрез което може да провери работата на библиотеката и улесни откриването на някои конфигурационни грешки. Представено е подробно обяснение на процесите извършвани от сървъра по време на автентикацията, когато се използва разработената библиотека. Описани са проблемите свързани с използвания механизъм за автентикация. Представени са някои отворени въпроси и идеи, които дават насока за бъдеща работа по допълнително развитие на предложената реализация. Като заключение можем да кажем, че дипломната работа изпълнява поставените задачи представени в главата „Увод”.

9. Използвани термини и съкращенията

- **Керберос**
  Протокол за мрежова автентикация разработен в Технологичният институт в Масачузетс, използва симетрични ключове за кодиране и изисква доверен трети участник в комуникацията.

- **Автентикация**
  Процесът на проверка на идентичността на някой или нещо. Установяването дали нещо е това, за което се представя.

- **Авторизация, упълномощаване**
  Процесът на определяне на правата на достъп до определени ресурси в дадена компютърна система.

- **Потребител**
  В този документ се използва в смисъла на физическо лице.

- **JAAS, Java Authentication and Authorization Service**
  Платформа за сигурността предоставяна като част от JDK, която дефинира набор от интерфейси използвани при процесите на автентикация и авторизация

- **GSSAPI**, **Generic Security Services Application Program Interface**
  Стандарт дефиниран от IETF адресиращ проблема с определянето на механизма за установяване на сигурен канал за комуникация

- **SPNego**, **Simple and Protected Negotiation Mechanism**
  Simple and Protected Negotiation Mechanism - механизъм използващ HTTP хедъри за договаряне на протокол за автентикация и обмен за съобщения за този протокол.

- **SSO**, **Single Sign On**
  Общо наименование за механизъм позволяващ автоматичен достъп до различни ресурси след еднократна успешна автентикация

- **Replay attack**
  Мрежова атака, при която прихванати данни се използват повторно с цел злоупотреба. Например изпращане на подслушана кодирана парола с цел представяне за някой друг.

- **Криптиране, кодиране**
  Процесът на модифициране на данните с цел скриване на информацията от недоброжелатели.

- **Сървър за Автентикация, Authentication Server, AS**
  Доверен сървър удостоверяващ идентичността на участниците в дадена комуникация посредством техните тайни ключове.

- **Сесиен ключ**
  Ключ, на които се базира симетричното кодирането използвано по време на комуникация между клиента и сървъра.

- **Автентикационни данни**, **credentials**
  Данни, чрез които се удостоверява идентичността, например име и парола, сертификат, Керберос билет и други.

- **Кербероски Билет**
  Съобщение издавано от AS, чрез което клиентът осъществява връзка със сървъра. От Кербероския билет сървърът извлича сесийния ключ, с който да кодира съобщенията преди да ги изпрати към клиента.

- **Автентикатор**
  Съобщение, кодирано със сесийния ключ използвано за удостоверяване на идентичността на клиента. Основно съдържа текущото време.

- **Сървър за издаване на билети, Ticket Granting Server, TGS**
  Сървър използван за издаване на билет за даден сървър след представяне на Главен Кербероски билет

- **Главен Кербероски Билет, Билет за достъп до други билети, TGT**
  Билет издаван от AS за комуникация с TGS.

- **Център за предоставяне на ключове, Key Distribution Center, KDC**
  Логическо обединение на AS и TGS.

- **Първоначална размяна на билет**
  Процесът на обмен на съобщения с AS с цел получаването на сесиен ключ за комуникация с TGS, въпросният сесиен ключ се нарича главен кербероски билет.

- **Допълнителна размяна на билет**
  Процесът на обмен на съобщения между клиента и TGS с цел получаването на сесиен ключ за комуникация с даден сървър.

- **Публични ключове**
  Ключове използвани за генериране на съобщения, които само собственикът на съответния таен ключ може да декодира.

- **Симетричен таен ключ**
  Ключ използван при операциите кодиране и декодиране на данните

- **Гарант за сертификатите (certificate authority), CA**
  Сървър удостоверяващ коректността на сертификатите.

- **IETF, Internet Engineering Task Force**
  Организация имаща за цел да дефинира стандартите използвани в интернет

- **Делегация на автентикационни данни**
  Предоставяне на автентикационните данни за даден сървър с разрешение той да ги използва за да се представя с тях пред други сървъри в случай на нужда

- **SPKM , Simple Public Key Mechanism**
  Механизъм за автентикация с публични ключове поддържан от някои реализации на GSSAPI

- **IANA, Internet Assigned Numbers Authority**
  Организация, която дефинира уникалните наименования и константи използвани в мрежовите протоколи

- **Unique Object Identifier (OID)**
  Поредица от числа идентифицираща даден обект регистриран от IANA

- **JDK, Java Development Kit**
  Набор от програми и библиотеки необходими за писането и стартирането на java приложения.

- **GSSContext, контекст**
  Статусът на единия от участниците в процеса на автентикация. "Контекст е установен" означава, че може да се изпращат съобщенията по сигурен канал.

- **JAAS Subject**
  Обект, представящ заявителя за дадена операция, който обект след успешна автентикация съдържа набор от идентичности (като например име по паспорт и ЕГН) и автентикационни данни(например сертификат)

- **Интегрирана автентикация, Integrated Windows Authentication (IWA)**
  Използва се в смисъл на определен режим на работа на дадена програма (например браузър), за който се използва Керберос като първоначален механизъм за автентикация

- **NTLMSSP - NT Lan Manager Security Support Provider**
  Сървис работещ на операционната система Windows, използва протокола NTLM

- **NTLM, NT LAN Manager**
  Протокол за автентикация дефиниран от Microsoft

- **LANMAN**
  Остарял протокол за автентикация използван от Microsoft Windows, заменен от NTLM

- **браузър ,Web browser**
  Програма осигуряваща възможност за достъп до информация получавана от сървър от локалната мрежа или интернет като се използва протокола HTTP.

- **хедър, HTTP header**
  Специални данни използвани от HTTP протокола за задаване на определени стойности имащи отношение към HTTP сесията

- **JAAS конфигурация**
  Списък от логин модули, заедно с техните флагове и опции.

- **Домейн контролер**
  Специален сървър, който отговаря за автентикацията и авторизацията на потребителите в домейна.

- **Java system properties**
  Набор от опции, които дефинират средата, в която работи виртуалната машина на Java.

- **keytab файл**
  Съдържа ключове за автентикация пред KDC, генерирани на базата на паролата на потребителя

- **krb5.conf файл**
  Съдържа основната информация необходима за дефинирането на настройките за протокола Керберос

- **Сървис потребител**
  Вид регистрация на потребител имаща за цел да включи машината на сървъра за приложения логически към списъка с потребители в домейна.

- **ASN.1, Abstract Syntax Notation One**
  Вид стандартизирана нотация използвана за описания на структури използвани при кодирането, декодирането и транспортирането на данни.

- **DER, Distinguished Encoding Rules**
  Набор от правила използвани за кодирането на обекти с нотацията ASN.1

10. Използвана литература

1. Дефиниция за автентикация
   http://en.wikipedia.org/wiki/Authentication

2. Kerberos Network Authentication Protocol
   http://web.mit.edu/kerberos/www/

3. The Kerberos Network Authentication Service (V5)
   http://www.ietf.org/rfc/rfc1510.txt

4. Generic Security Service Application Program Interface
   http://www.ietf.org/rfc/rfc2743.txt

5. Generic Security Service API Version 2 : Java Bindings
   http://www.ietf.org/rfc/rfc2853.txt

6. Java GSSAPI, Single Sign-on Using Kerberos in Java
   http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/single-signon.html

7. Java Authentication and Authorization Service (JAAS)
   http://java.sun.com/products/jaas/

8. The Simple and Protected GSS-API Negotiation Mechanism
   http://www.ietf.org/rfc/rfc2478.txt

9. Krb5LoginModule
   http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/spec/com/sun/security/auth/module/Krb5LoginModule.html

10. HTTP-Based Cross-Platform Authentication via the Negotiate Protocol / Kerberos Infrastructure Configuration
    http://msdn2.microsoft.com/en-us/library/ms995329.aspx

11. HTTP-Based Cross-Platform Authentication via the Negotiate Protocol / SPNEGO Tokens
    http://msdn2.microsoft.com/en-us/library/ms995330.aspx

12. Mozilla - Integrated Authentication
    http://www.mozilla.org/projects/netlib/integrated-auth.html

13. Login Modules and Login Module Stacks
    http://help.sap.com/saphelp_nw04/helpdata/en/8c/f03541c6afd92be10000000a1550b0/content.htm

14. Протокол NTLM
    http://en.wikipedia.org/wiki/NTLM

11. Приложение

11.1 Ообработване на SPNegoInit и SPNegoTarg

11.1.1 Обработване на SPNegoInit

Тук представяме подробно схематично описание на стъпките изпълнявани от сървъра при получаването на SPNegoInit съобщение с цел откриване на името на потребителя. Описаният сценарии ни показва случая, когато не е нужно допълнително договаряне.

Стъпка 1. Сървърът получава съобщение подобно на това представено на фигура 43.

```
YIIEwwYGKwYBBQUCoIIEtzCCBLOgJDAiBgkqhkiC9xIBAgIGCSqGSIb3EgECAgYKKwYBBAGCNwICCqKCBI
kEggSFYIIEgQYJKoZIhvcSAQICAQBuggRwMIIEbKADAgEFoQMCAQ6iBwMFACAAAACjggOUYYIDkDCCA4yg
AwIBBaEWGxRTRUNET00uU0FQTEFCUy5TT0ZJQaIbMBmgAwIBAqESMBAbBEhUVFAbCGtlcmJlcm9zo4IDTj
CCA0qgAwIBA6EDAgEDooIDPASCAzg2KkUwKAktW0Uodp1IHip1gS7Tu59S2Rwo1uvQ8qvXN3mcKio5qiQM
NJLq41gQ4q2w+0zTiLIe3D62hKSYfh7LlXsxhb8kPC3xVr9cW3hitTAQGa+wjTWEUE+7DMl6wDxPoU/sQT
q51obWXvcT83XkE/cMvT9c/8Hb30VcMMut3pohqUbLSEnPDDb9k2qN0L12ElXtkVXHySzM0MJdbpBZp2EC
6BmfZtgUuB9JU5H4EDP4eXQjsOhRxDd4K/pDf5maMkFWsGFNq3JYp65tX6gYMOYkHCjA2iKoAnmUg1ITb2
83Y8MLhHJSxGLlbn7jni7QVt0nLP0dEaF0bpojE/sEUFjZ7rSs6KmwAoBbvVRvX82P8MxXH2hrdegwxAC0
NjqjCTS35CgkfGjFRboQkC5JkzY8b3/BTxZKgZyfysdzjYI//XQfy0McmaYBMfCEQHfhI0TzjTpnfzDkSv
Jziimqv7V6JWQov3a2vO5JkYR/Meu45J9+Y/nNamWKz0ILmxVHCYOvjRE4xGg6QS1EqABBSJyFSLxe3D0X
ZXNwrIrdDxv75rxTyUI5KfABMJsNpF+0QhfpORK+Nh8cA1SCCyl34U5G4+sKJ2G+K5bQ7ipKTFThxHNvpT
Wny+yLnZnRLI5YA555YBVAseYXFUgc2m7i74VZXn+TBO0UEnlMZwV4burhocn4GSPSVlYDS6wGOmNJUULX
Y0otPpfA9xuqoggVQxU0iP5L8ik8BoqDG8Hjn9BWVxfaZef86rUfM6Vr/XZKGwoIakXLz/qSpcoEGgJz0Z
B45iXz7/DxfU0SszC5WMP7dbuSYnd04WcKZ2YFDiCZMM90YAh7twsMMRnM2X4eg5EtLRvZBWwBc/3p9qgo
2H6Qjie7eZcBvCL1dJ2eMo+ief4ycCTLi3vZtXip2XHaNoz+u15B7bD/FquhwC3+5cQGS7gJLsHsadyU2D
05ToQIZx9NNLVa294Zn2NR1L6uWc9nlRR0MXDvCHkF+NYKBdsrWBPiomkm/ndnWIMCave6M+WiDKaYycV9
zZJQhxRnEUqiVpH1iaQiUpQUVgvRk2fhhsiZvrPcbEO83zFQMnb3EIW5nyGGuqSBvjCBu6ADAgEDooGzBI
Gwd8QA61sjho0GqMB0m3fHconIDukNAKjQysCkzvYTi0g70TguIYbzm2EDUwXpU18Zkg+lwMOzVhCrU9Fi
kSAQexuxPY8Yusfe01XNdqP7/fR0HSWHzjLlN9kOiVMPowI99URwEuy/qDxQX0WGCzTiTJQXTP1dgRX8BW
Ox0H77AJhAABoE+L9jG/XzFlM+V3BiTB/wRZLB3OZXW2M+2pkWHGW+AQNDLmY5VxKbUMn0cQY=
```

Фигура: 43: Пример за валиден SPNego низ

Този низ започва задължително с **YII.** В случай на NTLM низ първите няколко символа за "**TlRM**".

Стъпка 2. Като резултат от Base64 и DER(ASN.1) декодиране се получава следната структура.

```
  Init Token {
    Supported Mechs {
      1.2.840.48018.1.2.2
      1.2.840.113554.1.2.2
      1.3.6.1.4.1.311.2.2.10
    }
    Mech Token 6082481692a864886f712... [1157 bytes]
}
```

Фигура 44: Декодиран SPNegoInit обект.

Фигура 44 представя списъка от поддържани механизми и началното съобщение съответстващо на първия поддържан механизъм. Съдържанието показано на фигурата е получено като резултат от извикването на метода toString() на реален обект от клас SPNegoInit.

Стъпка 3 Сървърът одобрява първия предложен механизъм, това е идентификаторът на Microsoft Kerberos 5

Стъпка 4 Сървърът разкодира структурата на Mech Token.

Първото съобщение за установяване на контекст за комуникация има вида представен на фигура 45. Различават се името на домейна и името на сървиса. На фигурата 45 в ляво са показани отделните байтове (по 16 на всеки ред разделени с „:"), а на съответния ред в дясно са изобразени ASCII символите на тези от тях, които имат някакво графично представяне. (Останалите са изобразени с точки.) Виждат се с удебелен шрифт имената на домейна и на сървиса.

```
60:82:04:81:06:09:2a:86:48:86:f7:12:01:02:02:01  `.....*.H..□....
00:6e:82:04:70:30:82:04:6c:a0:03:02:01:05:a1:03  .n..p0..l.......
02:01:0e:a2:07:03:05:00:20:00:00:00:a3:82:03:94  ........ .......
61:82:03:90:30:82:03:8c:a0:03:02:01:05:a1:16:1b  a...0.........□□
14:53:45:43:44:4f:4d:2e:53:41:50:4c:41:42:53:2e  □SECDOM.SAPLABS.
53:4f:46:49:41:a2:1b:30:19:a0:03:02:01:02:a1:12  SOFIA.□0□......□
30:10:1b:04:48:54:54:50:1b:08:6b:65:72:62:65:72  0□□.HTTP□.kerber
6f:73:a3:82:03:4e:30:82:03:4a:a0:03:02:01:03:a1  os...N0..J......
03:02:01:03:a2:82:03:3c:04:82:03:38:36:2a:45:30  .......<...86*E0
28:09:2d:5b:45:28:76:9d:48:1e:2a:75:81:2e:d3:bb  (.-[E(v.H-*u....
9f:52:d9:1c:28:d6:eb:d0:f2:ab:d7:37:79:9c:2a:2a  .R.□(......7y.**
39:aa:24:0c:34:92:ea:e3:58:10:e2:ad:b0:fb:4c:d3  9.$.4...X□....L.
88:b2:1e:dc:3e:b6:84:a4:98:7e:1e:cb:95:7b:31:85  ..-.>....~-..{1.
bf:24:3c:2d:f1:56:bf:5c:5b:78:62:b5:30:10:19:af  .$<-.V.\[xb.0□□.
b0:8d:35:84:50:4f:bb:0c:c9:7a:c0:3c:4f:a1:4f:ec  ..5.PO...z.<O.O.
...
```

Фигура 45: Началните байтове от съобщението Mech Token.

Mech Token представлява кербероска заявка KerberosApReq, която съдържа кербероски билет. Тези 2 обекта са показани на фигура 46. Тук се виждат основните полета от структурите, като отново се фокусираме върху името на домейна и името на сървиса.

```
KerberosApReq: [
  protocol version number: 5
  protocol message type : KRB_AP_REQ
  apOptions: [0, 32, 0, 0, 0]
  ticket: KerberosTicket: [
             tkt_vno: 5
             realm: SECDOM.SAPLABS.SOFIA
             sname: [type= NT-SRV-INST,value= HTTP/Kerberos]
          ]]
```

Фигура 46: Съдържание на KerberosApReq обект и съдържания в него KerberosTicket

Стъпка 5 Проверка дали се поддържа автентикация за получения домейн.

Сървърът проверява дали има указан сървис потребител, посредством който да установи контекст с домейн контролера. Такъв трябва да е бил указан в опциите на логин модула. В нашия пример за домейна SECDOM.SAPLABS.SOFIA е указан сървис потребител

j2ee-secdom@SECDOM.SAPLABS.SOFIA. Сървърът използва получените вече автентикационни данни (GSSCredential) от автентикационния кеш за да установи на контекст.

```
GSSCredential:
   j2ee-secdom@SECDOM.SAPLABS.SOFIA
   1.2.840.113554.1.2.2 Accept
   [Kerberos Principal j2ee-secdom@SECDOM.SAPLABS.SOFIAKey Version 1key
    EncryptionKey: keyType=3 keyBytes (hex dump)=0000: 83 2F 1A 2A 58 EA B0 C4
   ]
```

Фигура 47: Автентикационни данни на сървис потребителя.

Фигура 47 показва резултата получен след извикването на метода toString() за обект GSSCredential при SUN JDK.

```
--- GSSCredential ---
Number of mehanism credentials: 1
[1] Kerberos credential, mechanism: 1.2.840.113554.1.2.2
Owner: J2EE-AS4-USER1@SECTEST.SAPLABS.SOFIA
Usage: accept only
StartTime: 6/26/07 5:27 PM
InitLifeTime: unknown
AcceptLifeTime: indefinite
Krb5Client: J2EE-AS4-USER1@SECTEST.SAPLABS.SOFIA
Krb5Server: unknown
--- End of GSSCredential ---
```

Фигура 48: Автентикационни данни на сървис потребителя при IBM JDK.

Фигура 48 показва резултата получен след извикването на метода toString() за обект GSSCredential при IBM JDK.

Стъпка 6 Сървърът прави опит за установяване на контекст за комуникация.

Сървърът използва масива от байтове Mech Token като параметър за acceptSecContext() с цел установяване на контекст за комуникация. Ако всичко е коректно, сървърът получава съобщение от вида представен на фигурата 49.

```
60:68:06:09:2a:86:48:86:f7:12:01:02:02:02:00:6f        `h..*.H..□.....o
59:30:57:a0:03:02:01:05:a1:03:02:01:0f:a2:4b:30        Y0W..........K0
49:a0:03:02:01:03:a2:42:04:40:51:1f:18:81:e9:80        I......B.@Q□...
24:0d:13:0f:ca:07:bc:1f:aa:e3:e9:cb:e5:a1:78:a5        $.□.........x.
c4:56:09:e5:e1:2f:de:50:4c:42:b3:bf:dd:0b:84:61        .V.../.PLB.....a
b8:15:0a:9c:56:26:95:a4:26:a4:3b:90:e0:31:cd:9d        .□..V&..&.;..1..
eb:c5:b8:18:7e:35:f0:33:21:e1                          ...□~5.3!.
```

Фигура 49: Съобщение-резултат от опита за установяване на контекст.

Стъпка 7. Получаваме името на потребителя

Сървърът проверява дали контекстът е установен и ако е така може успешно да извлече от контекста името на потребителя, чрез GSSContext.getSrcName(). Полученото име е от вида **<samaccountname>@<DOMAIN NAME>** или по-конкретно

georgi@SECDOM.SAPLABS.SOFIA.След като сървърът получи името в този формат считаме, че автентикацията е приключила успешно.

11.1.2 Обработване на SPNegoTarg

За възпроизвеждането на сценария описан тук трябва сървърът да изисква механизъм за договаряне различен от този, който браузърът предлага като първи в неговия списък от поддържани механизми[11].

Стъпка 1.Получаване на първоначален хедър.

Сървърът получава съобщение подобно на това показано на фигура 43.

Стъпка 2.Извличане на обект SpNegoInit

Получава се SpNegoInit обект, който има вида показан на фигура 44.

Стъпка 3.Генериране на SpNegoTarg

В този момент сървърът не може да установи контекст, защото полученото съобщение не съдържа входния низ за механизъма предпочитан от сървъра. Затова се изпраща съобщение до браузъра, с което се указва какъв механизъм иска сървъра. Създава се обект SpNegoTarg, който има вида представен на фигура 50. Показаният обект има “response token”=null, защото единствената цел на този обект е да укаже механизма, с който сървърът иска да установи контекст за комуникация.

```
Targ Token {
    Neg Result accept_incomplete (1)
    Supported Mech 1.2.840.113554.1.2.2
  }
  Response Token [null]
}
```

Фигура 50: Обект от тип SpNegoTarg .

Стъпка 4. Генериране на нов хедър

Сървърът генерира Base64 кодирано съобщение, което съдържа обектът представен на предната стъпка. Фигура 51 представя низът, получен от Base64 кодирането на обекта представен на фигура 50.

```
oRQwEqADCgEBoQsGCSqGSIb3EgECAg==
```

Фигура 51: Base64 кодиран обект от тип SpNegoTarg

Стъпка 5. Получаване на нов хедър.

Сървърът изпраща низа от предната стъпка и след това получава нови SpNegoState и HTTP хедър. Новият хедър има вида показан на фигура 52. Първите символи винаги са „**oYI”.**

```
oYIJdjCCCXKiggS1BIIEsWCCBK0GCSqGSIb3EgECAgEAboIEnDCCBJigAwIBBaEDAgEOogcDBQAAAAAo4
IDuGGCA7QwggOwoAMCAQWhFxsVU0VDVEVTVC5TQVBMQUJTLlNPRklBohswGaADAgECoRIwEBsESFRUUBsI
a2VyYmVyb3OjggNxMIIDbaADAgEDooIDZASCA2CY00QidLm4/43p+uL60qMJ5iRRqnNGlCe1NDzkP/SVdL
sOnBM4SD5YMEEGk6KD1GiY9bf4ioK3EVtmTDtPI4K5TaYdBhMo4AGm8jxtwvkav0ebFv4fOcLHTKwsRPdm
SLe1Lglr1FuNrWQY7h5nOagDlsr1OZidXq5oISssq/4ovkDgnOvnyAPjqxJcpOoxadJpLxiahhw9shjNP0
yZGLl8i3OYue161/brd6BvERk+skBfUfcaDYSMG3e7iqzD1HAAk0fpROZ/p77PbFVCd98WXTiNfzulNi0y
cLmlUjoeG8HE9TcIlaJCt07iqlBbEeeZU84UT5oITxP7tU1DtMqylnfsr8luH9dg/SMxu8qr6CO1sIiS7v
fwXIhTL1tluGGLbAhPwWazl2ZbKSO+tCLH7RXQ13jsdFaEb4AP+SplAYyD8NZQgMJ9uJXwAm3LK1WYbPcH
5j82KOh4VMUC0iOlHziUyE68GrklQQ5FO3Gwu9vAcmxDtoNMUhPhexp+7Yr7iywUvEd7O1fxfJsyqS54hA
idfYjMCxEYbgo1iV3MYrFcpzvCB3pcu4BP7rLWgwq//+HXzbb2Uy7sV80z1FUZFlKPgERCgpgdJiB2/eQF
sT43a7Pmjf/LlTNPv5Pqw5DzPx3bvLHSQpDaqbRgeePGwXD96FEkKaZ5eTnBO1UrybxukPbJ6LSmpIO7uP
jEQ8eW2USxaa9rNIplRXXKaurI2VkxOBo0Yg521hwz7yHxYE6CX+H+cag+QErBfLbocE5924q0M0AYgBzY
M9/2r5vGQzfkQtP65w0N1+/SLZUBDFT7QdxCiF3ZcIVqxh3KvwxCIcVbLepili2sgwZhEjAzRYuWaMg9Le
hKUQibbmex/vDFVlrzh1B6XPY9+qUUMyafrv/EmpNoC5hAqwhYJtZUb67qZXMVXrIeO4nekNTOln+ML9xs
vJyMEe9cZCWn5Hw+ZU5pjkTJl3qqee/SulOVzwJ6Kn+7CvEc0e9K79CVgVsIgtR2h2j20Ej5EvLBF9mEGq
JqUKY8Me+xI7fQm7BWZNYlGzzYAhN7qYTbwgDCkpo4iu2We5Y3U7EWXNMLZZV1tGbMn7Z2wX83xO6YBT0V
ANnTezthL7yq2E/XBch2bEhM/k0dRg5B4+QZuUWd2F2kgcYwgcOgAwIBA6KBuwSBuDyiUEkZDFjmaMT7QC
```

```
680RwGPOG+Q/qG/OqC+wzhu57rV5LTqZIY2wZIW2tzVYBBlHZX7epJ8g/UklRoyqnYS4kYQg/hMmEdN47h
aUaaBuTMo1QHTJvtgCMrVATLgd7I+DWYR1b3k8fXP9bnAuqUzgonTS2+MJ6tI8bvSE3ZHFC05l39iwYTIY
3ecuDLb3vpprVshR/j5z7D8zVqMwNIhUKftsBjqAB236VUukZ2NUw5xIKoh7KTnBajggS1BIIEsWCCBK0G
CSqGSIb3EgECAgEAboIEnDCCBJigAwIBBaEDAgEOogcDBQAAAAAo4IDuGGCA7QwggOwoAMCAQWhFxsVU0
VDVEVTVC5TQVBMQUJTLlNPRklBohswGaADAgECoRIwEBsESFRUUBsIa2VyYmVyb3OjggNxMIIDbaADAgED
ooIDZASCA2CY00QidLm4/43p+uL60qMJ5iRRqnNGlCe1NDzkP/SVdLsOnBM4SD5YMEEGk6KD1GiY9bf4io
K3EVtmTDtPI4K5TaYdBhMo4AGm8jxtwvkav0ebFv4fOcLHTKwsRPdmSLe1Lglr1FuNrWQY7h5nOagDlsr1
OZidXq5oISssq/4ovkDgnOvnyAPjqxJcpOoxadJpLxiahhw9shjNP0yZGLl8i3OYue161/brd6BvERk+sk
BfUfcaDYSMG3e7iqzD1HAAk0fpROZ/p77PbFVCd98WXTiNfzulNi0ycLmlUjoeG8HE9TcIlaJCt07iqlBb
EeeZU84UT5oITxP7tU1DtMqylnfsr8luH9dg/SMxu8qr6CO1sIiS7vfwXIhTL1tluGGLbAhPwWazl2ZbKS
O+tCLH7RXQ13jsdFaEb4AP+SplAYyD8NZQgMJ9uJXwAm3LK1WYbPcH5j82KOh4VMUC0iOlHziUyE68Grkl
QQ5FO3Gwu9vAcmxDtoNMUhPhexp+7Yr7iywUvEd7O1fxfJsyqS54hAidfYjMCxEYbgo1iV3MYrFcpzvCB3
pcu4BP7rLWgwq//+HXzbb2Uy7sV80z1FUZFlKPgERCgpgdJiB2/eQFsT43a7Pmjf/LlTNPv5Pqw5DzPx3b
vLHSQpDaqbRgeePGwXD96FEkKaZ5eTnBO1UrybxukPbJ6LSmpIO7uPjEQ8eW2USxaa9rNIplRXXKaurI2V
kxOBo0Yg521hwz7yHxYE6CX+H+cag+QErBfLbocE5924q0M0AYgBzYM9/2r5vGQzfkQtP65w0N1+/SLZUB
DFT7QdxCiF3ZcIVqxh3KvwxCIcVbLepili2sgwZhEjAzRYuWaMg9LehKUQibbmex/vDFVlrzh1B6XPY9+q
UUMyafrv/EmpNoC5hAqwhYJtZUb67qZXMVXrIeO4nekNTOln+ML9xsvJyMEe9cZCWn5Hw+ZU5pjkTJl3qq
ee/SulOVzwJ6Kn+7CvEc0e9K79CVgVsIgtR2h2j20Ej5EvLBF9mEGqJqUKY8Me+xI7fQm7BWZNYlGzzYAh
N7qYTbwgDCkpo4iu2We5Y3U7EWXNMLZZV1tGbMn7Z2wX83xO6YBT0VANnTezthL7yq2E/XBch2bEhM/k0d
Rg5B4+QZuUWd2F2kgcYwgcOgAwIBA6KBuwSBuDyiUEkZDFjmaMT7QC680RwGPOG+Q/qG/OqC+wzhu57rV5
LTqZIY2wZIW2tzVYBBlHZX7epJ8g/UklRoyqnYS4kYQg/hMmEdN47haUaaBuTMo1QHTJvtgCMrVATLgd7I
+DWYR1b3k8fXP9bnAuqUzgonTS2+MJ6tI8bvSE3ZHFC05l39iwYTIY3ecuDLb3vpprVshR/j5z7D8zVqMw
NIhUKftsBjqAB236VUukZ2NUw5xIKoh7KTnBY=
```

Фигура 52: HTTP хедър с обект SpNegoTarg

Стъпка 6. Извличане на обекта SpNegoTarg

Отново се декодира това съобщение като сега се получава обект от класа SpNegoTarg, който съдържа данните за механизма, който сървърът указа. Полученият обект SpNegoTarg има вида показан на фигура 53.

```
Targ Token {
  Response Token 60824ad692a864886f712
  MIC 60824ad692a864886f712
}
```

Фигура 53: Обект SpNegoTarg изпратен от браузъра.

Стъпка 7. Установяване на контекст.

Сървърът използва масива от байтове „`Response Token`" като параметър за acceptSecContext() с цел установяване на контекст за комуникация. Сега вече се използва точният механизъм и контекстът се установява успешно. Като резултат сървърът отново получава името на потребителя.

11.2 Избрани фрагменти от кода с коментари

Както стана ясно най-важната част от кода се намира в SPNegoLoginModule и по-точно в метода login(). Затова тук ще направим преглед на използвания програмен код с коментари за по-важните стъпки.

Фрагмент 1

В началото на изпълнението на login() проверяваме дали за пръв път извикваме този метод или автентикацията е започнала преди това и сега продължава т.е. дали се намира процеса на автентикация във фаза 1, 2 или 3. За целта вземаме обекта SpNegoState, който съхраняваме в HTTP сесията. Ако няма такъв обект то тогава създаваме такъв като в конструктора му задаваме статус „НАЧАЛЕН" и след това този обект записваме в HTTP сесията. Прочитаме най-важният входен параметър – HTTP хедъра.Проверяваме дали процестът на договаряне е в начална фаза или незавършена.

```
 Object obj = getSessionAttribute( IConstants.SPNEGO_SESSION );
    SpNegoState spNegoState = null;
    if ( obj == null ) {
      spNegoState = new SpNegoState();
      this.setSessionAttribute( IConstants.SPNEGO_SESSION, state );
    } else {
      spNegoState = (SpNegoState) obj;
    }
    String receivedHeader = getHeader( IConstants.AUTH_HEADER_NAME );
    if (spNegoState.negstate == IConstants.SPNEGO_NEG_ACCEPT_INITIAL ||
        spNegoState.negstate == IConstants.SPNEGO_NEG_ACCEPT_INCOMPLETE)
…. //следва код за различните случай в зависимост от статуса и хедъра.
```

Фигура 54: Програмен код към фрагмент1

Фрагмент 2

Сега разглеждаме случая, когато имаме начално обръщение към сървъра и няма SPNego съобщение в HTTP хедър „Authenticate". Използваме помощния клас ConfigurationHelper. Прочитаме опцията на логин модула указваща сървис потребителите, които са във формат <име>@<ДОМЕЙН> и за полученият масив от имена на домейни ще проверяваме дали можем да автентицираме потребителите. Ако за поне един сървис потребител успеем да получим автентикационни данни, то тогава автентикацията може да продължи. Иначе имаме грешно конфигуриран сървър.

```
String[] realms = configurationHelper.getConfiguredRealms();
if (realms != null && realms.length != 0) {
    boolean areCredentialsAcquired = false;
     for (int i = 0; i < realms.length; i++) {
         String realm = realms[i];
          try {
              GSSCredential creds = configurationHelper.getCredentials(realm);
              credentialsAcquired = true;
            } catch (Exception e) {
               …
            }
     }
     if (!credentialsAcquired) {
        throw new LoginException("Can not get GSS credentials for at least one
Kerberos realm");
     }
} else {
    //SPNegoLoginModule is not configured to accept credentials from any Kerberos
realm.
    //Check option if IConstants.CONF_GSS_NAME exists and its value is valid.
    return false;
 }
```

Фигура 55: Програмен код към фрагмент 2

Фрагмент 3

Когато сме в случая на нова заявка обявяваме HTTP статус 401 Unauthorized, добавяме HTTP хедър „Authenticate: Negotiate”, който ще укаже на браузъра, че трябва да изпрати SPNego отговор, и записваме с сесията SpNegoState.

```
setStatus( HttpServletResponse.SC_UNAUTHORIZED );
setHeader( IConstants.WWW_AUTHENTICATE_NAME, IConstants.NEGOTIATE );
setSessionAttribute( IConstants.SPNEGO_SESSION, state );
```

Фигура 56: Програмен код към фрагмент 3

Фрагмент 4

Проверка дали полученото SPNego съобщение вече е било добавено към кеша. Ако го има вече в кеша и успешно е бил обработен и одобрен този низ тогава вземаме потребителското име получено при предното обработване на това SPNego съобщение и излизаме от метода login() с резултат true, ако не е бил одобрен тогава резултатът е false.

```
TokenStatus tokenStatus = threadTokenCache.containsToken( header );
 if ( null != tokenStatus) {
    if (tokenStatus.isAuthenicated) {
       userName = tokenStatus.userName;
       sharedState.put( IConstants.LOGIN_NAME, userName );
       return true;
    } else {
       return false;
    }
}
```

Фигура 57: Програмен код към фрагмент 4

Фрагмент 5

Обработването на полученият хедър става с метода doHandshake. Той връща като резултат нов HTTP хедър в случай,че контекст не бил установен. Ако се получи такъв хедър, то тогава го подготвяме за изпращане от сървъра към браузъра.

```
String headerToSend = doHandshake( spnegoState, receivedHeader );
if (headerToSend!= null ) {
  this.setStatus( HttpServletResponse.SC_UNAUTHORIZED );
  this.setHeader( IConstants.WWW_AUTHENTICATE_NAME, "Negotiate " + headerToSend);
}
```

Фигура 58: Програмен код към фрагмент 5

С това приключват фрагментите от метода login().

Фрагмент 6

Тук ще обърнем внимание на метода commit(). Основната цел на методът commit() е да добави новите идентичности (Principals) към обекта Subject в случая, когато потребителското име успешно е било извлечено от метода login(). Тъй като вече не е необходим обекта SpNegoState го премахваме от сесията.

```
public boolean commit() throws LoginException {
  if ( userName == null ) {
    return false;
  }
  Principal principal = new com.sap.engine.lib.security.Principal( userName );
  principals.add(principal );
  subject.getPrincipals().addAll( principals );
  SpNegoState state =
   (SpNegoState) getSessionAttribute( IConstants.SPNEGO_SESSION );
  release( state );
  setSessionAttribute( IConstants.SPNEGO_SESSION, null );
  threadTokenCache.cleanup();
  return result;
}
```

Фигура 59: Програмен код към фрагмент 6

Основен метод използван в метода login() е doHandshake(). Иизползва се за проверка на получения SPNego низ подаден като параметър. Ако в края на метода при извикването на gsscontext.acceptSecurityContext() резултатът е все още неустановен контекст, тогава doHandshake() ще върне нов SPNego низ кодиран в base64 формат. Ако има някаква грешка тогава генерираме изключение, автентикацията приключва, а SpNegoState получава статус „ОТХВЪРЛЕН”. Следващите няколко фрагмента показват по-съществените части програмен код използван за проверяването на хедъра.

Фрагмент 7

Тук показваме началните декларации на специфичните класове.Хедърът се декодира Base64 и Der и получаваме обект ASN1.

```
private String doHandshake( SpNegoState spnegoState, String receivedHeader )
                  throws SPNegoProtocolException {
  String responseHeader = null;
  ASN1Object asn1Object = null;
  SPNegoToken spnego = null;
  SpNegoInit spninit = null;
  SpNegoTarg spntarg = null;
  BasicSpNegoType basicspnego = null;
  int mechanismId = -1;
…//други декларации
  inputBytes = Base64.decode(receivedHeader);
  checkAuthorizationHeaderToken(inputBytes);
  asn1Object = DerCoder.decode( token );
```

Фигура 60: Програмен код към фрагмент7

Фрагмент 8

Тук конструираме обект SPNegoInit , след това генерираме обект за Кербероската заявка и след това извличаме Кербероския билет. Така извличаме Керберос реалма(домейна) и името на сървис потребителя. Вземаме от автентикационният кеш обекта GSSCredential и правим опит за установяване на контекст.

```
if (spnegoState.negstate == IConstants.SPNEGO_NEG_ACCEPT_INITIAL ) {
  spnegoToken = new SPNegoToken();
  spnegoToken.decode(asn1Object);
  basicspnego = spnegoToken.getSpnego();
  spninit = (SpNegoInit) basicspnego;
  gssintoken = spninit.getMechToken();
  KerberosApReq kerberosApReq = new KerberosApReq( gssintoken );
  KerberosTicket kerberosTicket = kerberosApReq.getTicket();
  String realm = kerberosTicket.getRealm();
  String principalName = kerberosTicket.getPrincipalName();
  try {
    GSSCredential credential = configurationHelper.getCredentials( realm );
    GSSManager gssManager = GSSManager.getInstance();
    GSSContext ctx = gssManager.createContext(credential);
    spnegoState.gsscontext = ctx;
    gssouttoken = spnegoState.gsscontext.acceptSecContext( gssintoken, 0,
gssintoken.length );
  } catch ( GSSException gsse ) {
    throw new SPNegoProtocolException(gsse.getMessage());
  }
```

Фигура 61: Програмен код към фрагмент 8

Фрагмент 9

Тук показваме случая на статус НЕЗАВЪРШЕН – използване на SPNegoTarg.

```
if ( state.negstate == IConstants.SPNEGO_NEG_ACCEPT_INCOMPLETE ) {
  spntarg = new SpNegoTarg();
  spntarg.decode( asn1Object);
  gssintoken = spntarg.getResponseToken();
  …
  try {
    gssouttoken = state.gsscontext.acceptSecContext( gssintoken, 0,
gssintoken.length );
  } catch ( GSSException gsse ) {
    throw new SPNegoProtocolException("Context establishment failed.");
  }
}
```

Фигура 62: Програмен код към фрагмент 9

Фрагмент 10

Тук показваме как получаваме името от установения контекст.

```
if ( spnegoState.gsscontext != null && spnegoState.gsscontext.isEstablished() ) {
  spnegoState.negstate = IConstants.SPNEGO_NEG_ACCEPT_COMPLETED;
  GSSName peer = spnegoState.gsscontext.getSrcName();
   peer = peer.canonicalize( new Oid( IConstants.OID_JGSS_MECHTYPE ) );
   username =  peer.toString();
   synchronized ( threadTokenCache ) {
     threadTokenCache.put(authorizationHeader, userName, kerberosPrincipalName );
   }
}
```

Фигура 63: Програмен код към фрагмент10

Фрагмент 11

Този фрагмент показва конструирането на нов HTTP хедър, когато контекст не бил установен от acceptSecContext().

```
try {
  //gss context negotiation continues
  spnegoState.negstate = IConstants.SPNEGO_NEG_ACCEPT_INCOMPLETE;
  spntarg = new SpNegoTarg();
  spntarg.setNegResult( IConstants.SPNEGO_NEG_ACCEPT_INCOMPLETE );
  spntarg.setSupportedMech( state.mech );
  // in case gssouttoken is null, we just send a Targ token back
  // to tell the browser which mechanism we intend to support.
  if ( gssouttoken != null ) {
    spntarg.setResponseToken( gssouttoken );
  }
  headerToSend = Base64.encode( DerCoder.encode( spntarg.toASN1Object() ) );
…
  }
} catch ( CodingException e ) {
  throw new SPNegoProtocolException("Unexpected decoding error.");
}
```

Фигура 64: Програмен код към фрагмент11

11.3 Преглед на помощните класове използвани в реализацията

11.3.1 Пакет spnego

- **`Iconstants`**

  Дефинира константите използвани в логин модула и помощните класове.

- **SPNegoProtocolException**

  Използва се като специален вид изключение генериран от процеса на SPNego автентикация.

- **SpNegoState**

  Съхранява 3 основни полета:

  1. статуса на автентикацията (начална, незавършена, завършена или отхвърлена)
  2. инстанцията на GSSContext посредством, която извършваме обръщенията към функционалността от GSSAPI
  3. идентификатора (ObjectID) на механизма за установяване на GSSContext

11.3.2 Пакет spnego.util

- **Base64**

  Използва се за кодиране и декодиране на base64 низове

- **ConfigurationHelper**

  Този клас кешира автентикационните данни за сървъра след като веднъж да били получени.

  Най-важният метод, на който трябва да се спрем е "**acquireCredentials**"

  Той извиква следния код

```
GSSManager gssman = GSSManager.getInstance();
GSSName gssname = gssman.createName( gssName, nametype );
GSSCredential gssCredential= gssman.createCredential( gssname,
                                GSSCredential.INDEFINITE_LIFETIME,
                                new Oid( strGSSMechOid ),
                                GSSCredential.ACCEPT_ONLY );
```

  Тук се автентицира специалният сървис потребител, който създадохме на домейн контролера и за който генерирахме keytab файла.

- **ShortLifetimeCache**

  Този клас реализира краткотраен кеш, наследява се от ThreadTokenCache.

- **ThreadTokenCache**

  Наследява ShortLifetimeCache. Целта на този клас е да предотврати обработването на SPNego низове повече от веднъж в един и същ Thread обект. Затова този клас пази SPNego низовете кеширани , а като ключ за търсене се използва текущият обект от клас Thread. Този

механизъм има за цел да предотврати "replay attacks" посредством следните 3 подхода едновременно използвани:

1.SPNego низовете могат да бъдат намерени само в правилния Thread обект, т.е. ако SPNego низът е бил одобрен и ако в последствие някой открадне този низ (SPNego token) ,то той няма да успее да злоупотреби, защото автентикацията(първоначално) е протекла в друг Thread.

2.Животът на този кеш е една секунда, което е достатъчно кратко. След този период кешът ще връща NULL при всички извиквания за обекти.

3.След успешна автентикация или неуспешна, поради някаква грешка, кешът се изтрива. Т.е. използва се само докато протича процеса на автентикация.

- **`TokenStatus`**

  Представлява една обикновена структура, която пази:

  1.булева стойност дали е успешна автентикацията или не

  2.KerberosPrincipalName

  3.Потребителско име, което евентуално може да е различно от KerberosPrincipalName

11.3.3 Пакет spnego.asn1

- **`BasicSpNegoType`**

  Основен клас за типа получената структура SPNego.

- **`SpNegoASN1`**

  Основен метод в този клас е decode(). Той извършва следните операции:

  1.Изчита и проверява идентификатора (ObjectID)на SPNego протокола

  2.Изчита и проверява типа на контекста

  3.Задава BasicSpNegoType от клас SpNegoInit или SpNegoTarg

- **`SpNegoInit`**

  наследява BasicSpNegoType, реализира интерфейса ASN1Type

```
NegTokenInit ::= SEQUENCE {
  mechTypes [0] MechTypeList OPTIONAL,
  reqFlags [1] ContextFlags OPTIONAL,
  mechToken [2] OCTET STRING OPTIONAL,
  mechListMIC [3] OCTET STRING OPTIONAL
}
```

- **SpNegoTarg**

наследява BasicSpNegoType реализира интерфейса ASN1Type

```
NegTokenTarg ::=  SEQUENCE {
      negResult [0]  ENUMERATED { accept_completed (0),
                                  accept_incomplete (1),
                                  rejected (2) }  OPTIONAL,
      supportedMech [1]  MechType     OPTIONAL,
      responseToken [2]  OCTET STRING OPTIONAL,
      mechListMIC   [3]  OCTET STRING OPTIONAL }
```

- **SPNegoToken**

Има 2 типа SPNego низове(tokens):

1. Init
2. Targ

Те използват различни кодирания[10]. SPNegoToken физически е TLV структура и има следния вид:

Тип=0x60 <байтове Дължина> {стойност: <SP Nego OID><SP Nego type specific> };

<SP Nego type specific> е Init или Targ, разпознава се по първия байт (0xa0 или 0xa1)

- **TLVParser**

Осигурява методи използвани при изчитането на TLV структури

- **KerberosApReq**

Използва се основно за извличане на Керберос билета от Керберос заявката изследва се масивът от байтове като TLV структура и се инициализират отделните полета (които също представляват TLV структури). По-конкретно една коректна Керберос заявка (KerberosApplicationRequest) има следното съдържание:

```
AP-REQ ::=  [APPLICATION 14] SEQUENCE {
      pvno[0]          INTEGER,
      msg-type[1]    NTEGER,
      ap-options[2]  APOptions,
      ticket[3]        Ticket,
      authenticator[4] EncryptedData
    }
```

- **`KerberosTicket`**

  Използва се за най-вече за получаване на Керберос реалма и KerberosPrincipalName. Извиква се от класа KerberosApReq. Следва съдържанието на тази структура:

```
Ticket ::= [APPLICATION 1] SEQUENCE {
      tkt-vno[0]  INTEGER,
      realm[1]    Realm,
      sname[2]    PrincipalName,
      enc-part[3] EncryptedData}
PrincipalName ::= SEQUENCE {
      name-type[0]     INTEGER,
      name-string[1]   SEQUENCE OF GeneralString}
Realm ::= GeneralString
```

11.4 Помощен софтуер използван при конфигуриране на системите

- **Помощна програма klist**

  Използва се за да проверим какви ключове има добавени към „keytab" файла и по-точно за кои потребители.

  Примерна употреба:

  klist -e -f -k -K keytab

  Примерен резултат от изпълнението на командата:

  Key tab: keytab, 2 entries found.
  [1] Service principal: kerberos_spnego@MY.KERBEROS.DOMAIN
  KVNO: 1
  Key type: 3
  Key: 0xefe125cfe09c59fd
  [2] Service principal: my_service_user@MY.KERBEROS.DOMAIN
  KVNO: 1
  Key type: 3
  Key: 0x2fa8dda2b24fe502

  Стойностите, които виждаме обозначени като „Service principal", отговарят на атрибута „**userPrincipalName"** за специалния потребител за J2EE Сървъра**.**

- **Помощна програма ldifde**

  Позволява бързо търсене на Active Directory Сървъра

  Примерна употреба:

  Ldifde –r "serviceprincipalname=HTTP/my.spnego.test" –f output.ldf

Резултатът съдържа атрибутите на потребителите със serviceprincipalname HTTP/my.spnego.test". Ако резултатът съдържа повече от един потребител, то автентикацията няма да сработи, защото Керберос протоколът няма да може да идентифицира кой точно потребител да използва за връзка с J2EE Сървъра. Резултатът трябва да съдържа точно един потребител и това трябва да е точно специалният сървис потребител, който дефинирахме по-горе и за който създадохме keytab файл.

- **Помощна програма <u>setspn</u>**

  Използва се за конфигурирането на сървис потребителите.

  1.Извежда списъка от регистрирани servicePrincipalName стойности

```
setspn -l Kerberos_spnego

Резултат:
   Registered ServicePrincipalNames for CN=
   Kerberos_spnego,CN=Users,DC=my,DC=kerberos,DC=domain:
   HTTP/my.kerberos.enabled.server
   HTTP/my.spnego.test
```

  2.Добавя нов SPN

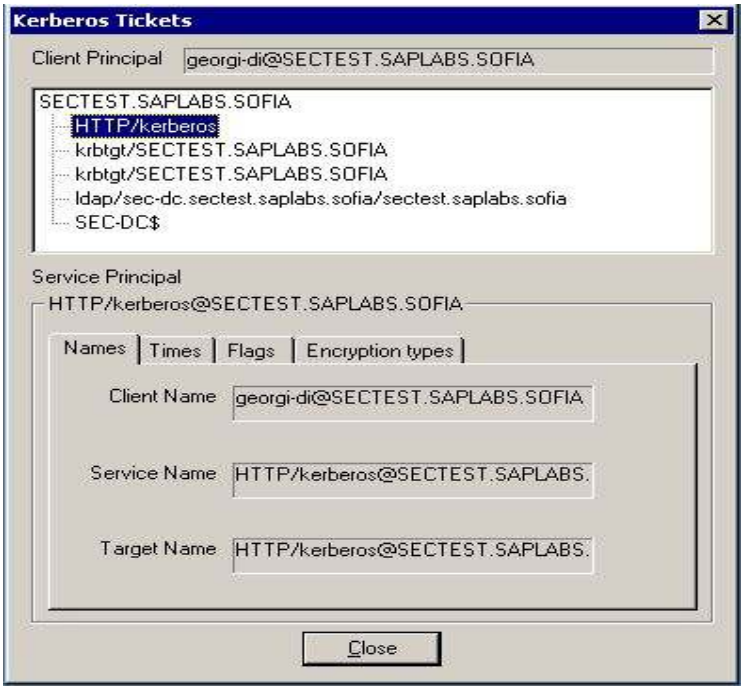
```
setspn -a HTTP /test Kerberos_spnego

Резултат:
   Registering ServicePrincipalNames for CN=
   Kerberos_spnego,CN=Users,DC=my,DC=kerberos,DC=domain
           HTTP /test
   Updated object
```

  3.Изтрива съществуващ SPN

```
setspn -d HTTP /spnego Kerberos_spnego
Резултат:
   Unregistering ServicePrincipalNames for CN=
   Kerberos_spnego,CN=Users,DC=my,DC=kerberos,DC=domain
   HTTP /spnego
   Updated object
```

- **Помощната програма <u>kerbtray</u>**

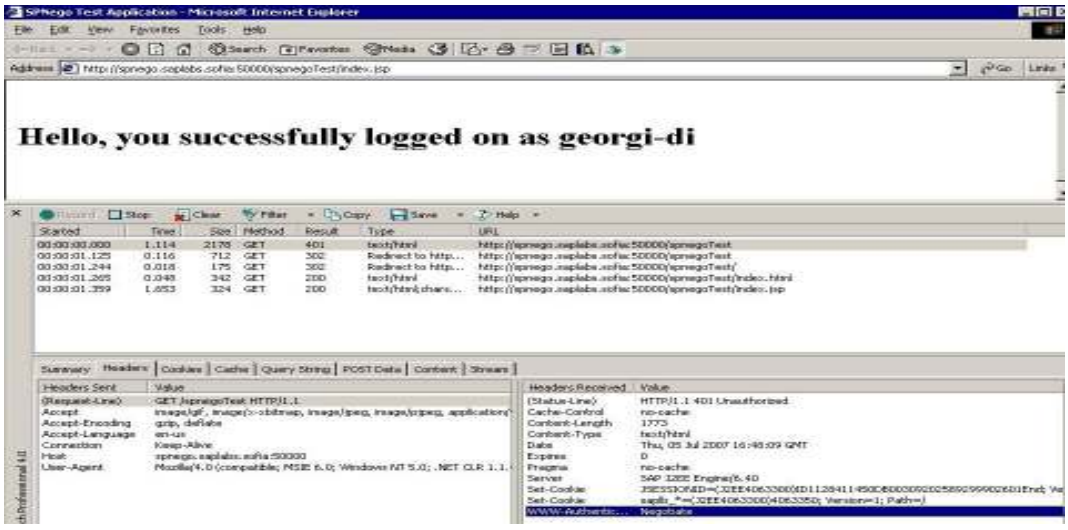
  Позволява да се видят Кербероските билети получени на клиентската машина. Фигура 65 показва полетата на Кербероския билет използван на машината по време на успешното тестване на SPNegoTestApp.



Фигура 65: Използване на kerbtray, маркиран е билетът на georgi-di за комуникация със сървиса HTTP/kerberos

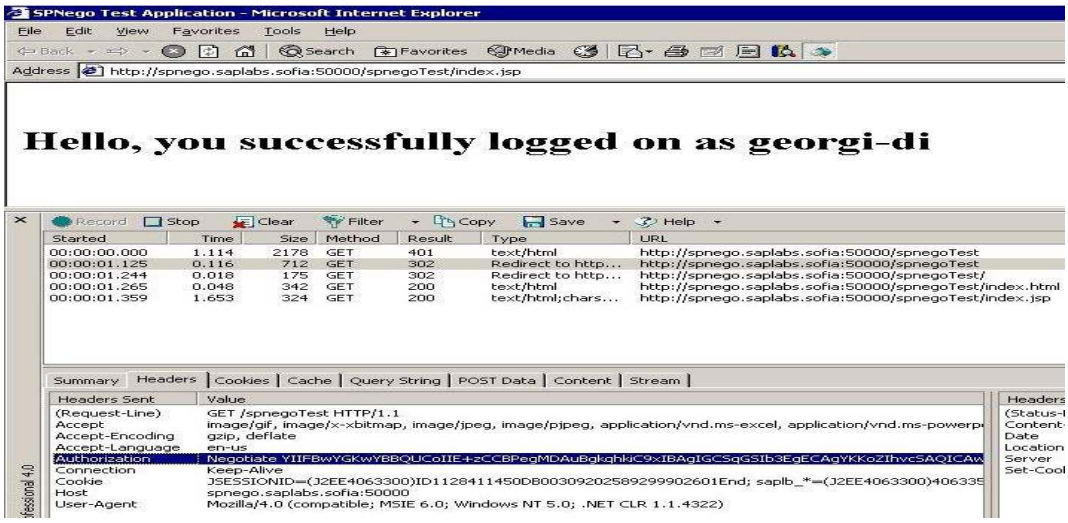
- **Помощната програма httpwatch**

  Позволява да се проследи HTTP трафика на клиента. Конкретно за нашата цел може да ни послужи като проследим дали правилно се изпращат и пристигат HTTP хедърите.Фигура 66 ни показва как изглежда HTTP watch. Маркиран е хедърът, който сървърът изпраща към браузъра.



Фигура 66: Използване на HTTPWatch за проследяване на WWW-Authenticate: Negotiate.

Фигура 67 ни показва маркиран HTTP хедъра, който браузърът изпраща към сървъра. В показания случай се вижда валидният SPNego низ съдържащ обект SpNegoInit, който успешно автентицира потребителят **georgi-di.** В случаите, когато браузърът не е конфигуриран правилно може да се види NTLM хедър, а в случаите, когато има нужда от допълнително договаряне може да се види и валиден низ съдържащ закодиран обект SpNegoTarg



Фигура 67: Използване на HTTPWatch за проследяване на HTTP хедър Authorization: Negotiate YII….