Version 1.0

Mike Wiesner (info@mwiesner.com)

Kerberos als Unternehmsweites Single-Sign-On

Single-Sign-On wird immer häufiger von Unternehmen gefordert. Die Vorteile liegen klar auf der Hand: Der Arbeitsfluss wird nicht durch ständige Kennworteingaben unterbrochen, der Benutzer muss sich nicht so viele Kennwörter merken und wählt daher erfahrungsgemäß ein komplexeres Kennwort, der Administrationsaufwand sinkt und letztlich steigt auch die Sicherheit, da sensible Daten nur noch auf einem System vorgehalten werden müssen. SSO macht allerdings nur dann Sinn, wenn das verwendete Protokoll auch auf breiter Ebene unterstützt wird.

Hier hat sich in den letzten Jahren das Open Source Protokoll Kerberos immer mehr hervorgetan. Mittlerweile wird es unter anderem auf Linux, MS Windows (ab 2000), Sun Solaris und Mac OS unterstützt. Teilweise gibt es sogar mehrere Implementierungen, die aber alle untereinander kompatibel sind, sodass ein Windows-Client ohne Probleme mit einem Linux-Server kommunizieren kann.

Auch beim Anwendungsprotokoll lässt Kerberos einem freie Wahl, sodass es ohne Probleme mit z.B.: ssh, http, ldap oder auch eigenen Protokollen verwendet werden kann. Zu den Programmen die Kerberos unterstützen zählen unter anderem Firefox, MS Internet Explorer, OpenSSH, OpenLDAP, Apache HTTP-Server, Cyrus und viele mehr. Für die eigene Softwareentwicklung stehen APIs für C, C#, Perl und Java zur Verfügung.

In diesem Paper werden zunächst Grundlagen zu SSO und Kerberos auf kompakte Weise vermittelt. Anschließend geht es um die Voraussetzungen für ein Unternehmsweites SSO und die möglichen Probleme die dabei auftreten können. Um das ganze anschaulicher zu machen werden schließlich noch einige Beispiele aus der Praxis vorgestellt und als Fazit die Vor- und Nachteile von Kerberos sowie die Alternativen dazu vorgestellt.

Grundlagen SSO

Viele IT-Landschaften bestehen heute aus verteilten, dezentralen Diensten, welche untereinander oft nur lose Koppelungen haben. So ist es durchaus üblich das beispielsweise Linux-Server mit Windows und Mac OS-Clients betrieben werden. Die Clients verwenden dann auch noch Unternehmensanwendungen auf Basis von Client-/Server- oder Web-Technologie und natürlic Standardproduktie wie E-Mail und Browser. Die Anwendungen selbst sind dabei in verschiedenen Programmiersprachen implementiert wie beispielsweise C, C++ oder JAVA und verwenden diverse Protokolle, unter anderem HTTP, IMAP, SSH oder IIOP.

Diese Vielfalt ermöglicht zwar eine Unabhängigkeit der einzelnen Komponenten, erschwert jedoch die Kommunikation untereinander. Deshalb setzt sich im Bereich der Unternehmenssoftware auch immer mehr eine Service-Orientiere-Architektur (SOA) durch. Dabei werden einzelne Dienste (Service) angeboten und über fest definierte Schnittstellen (meist SOAP oder REST) zur Verfügung gestellt.

Ein Problem was sich durch diese Verteilung allerdings auch ergibt, ist das der Anwender sich in irgendeiner weiße diesen System gegenüber authentifizieren muss. Klassischer weiße geschieht dies durch die Eingabe von einem Benutzernamen und einem Kennwort, was bei der Vielzahl der Systeme jedoch für den Anwender sehr schnell zur Last werden kann. Weitere Ansätze verfolgen die Authentifizierung anhand von Smart Cards, welche am Client angeschlossen werden und dort die Authentifizierung übernehmen, ohne das ständig ein Kennwort eingegeben werden muss. Damit geht es dann nicht darum, das ein Anwender etwas wissen muss (wie ein Kennwort), sondern das er etwas besitzt (wie eine Smart Card). Allerdings setzt dies voraus, das jedes der Systeme diese Art der Authentifizierung unterstützt und ein Wechsel auf ein anderes Authentifizierungs-Verfahren ist dann auch nicht ohne weiteres möglich.

Das Ziel muss also sein, die Art der Authentifizierung unabhängig von den System wählbar und austauschbar zu machen. Es ist also eine Abstraktionsschicht nötig, auf welche die System aufsetzen und welche die Clients verwenden um sich diesen gegenüber zu Authentifizieren.

Da dies dann auf dem Client passieren muss ergibt sich zwangsläufig das Problem, wie der bestätigte Benutzername bei Client-/Server-Anwendungen auf den Server kommt, ohne das dieser dabei verändert werden kann. Die Abstraktionsschicht muss also auch dieses Problem adressieren und eine Möglichkeit zur Verfügung stellen, sich nach einer gültigen Authentifizierungen bei mehreren Systemen anzumelden, welche auf verschiedenen Servern zur Verfügung gestellt werden.

Genau diese Probleme sollen durch Single-Sign-On (SSO) gelöst werden. Das SSO-System führt dabei einmalig eine Authentifizierung durch und stellt dann Möglichkeiten zur Verfügung, sich gegenüber weiteren Systemen zu authentifizieren. Dazu stehen im Idealfall Bibliotheken und/oder APIs für

Programmiersprachen zur Verfügung, sodass Anwendungsentwickler das SSO-System einbinden können.

Im Bereich der SSO-Systeme haben sich in den letzten Jahren vor allem die Ticket basierten System durchgesetzt. Hier wird das Kennwort, oder allgemeiner die Credentials, nur einmal benötigt und anschließend werden auf Basis von Verschlüsselungstechniken und Validatoren jeweils für einzelnen Dienste Tickets erzeugt, welche die Dienste dann prüfen können. Die Tickets lassen jedoch keine Rückschlüsse auf die Credentials zu und können auch weder durch den Benutzer noch durch dritte verändert werden. Zumindest sollten dies die Basisanforderungen an ein Ticket-System sein, da ja die Sicherheit durch den Einsatz eines solchen Systems nicht schlechter werden soll.

Womit auch schon der nächste Punkt erreicht wäre: Wie sieht es dabei denn mit der Sicherheit aus? Solange das Verfahren in sich sicher ist ergibt sich häufig eher eine Verbesserung der Sicherheit, oder zumindest keine Verschlechterung. Die Gründe dafür sind recht einfach: Bei SSO müssen die Authentifizierungsdaten nur in einem System vorgehalten werden. So ist z.B. das Kennwort nur beim Ticketserver bekannt, alle anderen System kennen es nicht. Dadurch verringert sich schon mal die Angriffsfläche. Weiterhin wird das Kennwort auch nur einmal benötigt, und muss so weniger oft über das Netzwerk geschickt werden. Und als letzter Punkt zeigt meine Erfahrung auch, das wenn die Benutzer ihr Kennwort nur einmal eingeben müssen, das sie dann eher bereit sind ein komplizierteres und damit sichereres Kennwort zu wählen.

Aber wie gesagt, das SSO-Verfahren an sich muss sicher sein, da sonst ein Angreifer im schlimmsten Fall Zugriff auf alle Systeme bekommen würde. Daher sollte man gerade an dieser Stelle mit bedacht auswählen und prüfen.

Das beste SSO-Verfahren nutzt allerdings auch nichts, solange es keine vernünftigen Bibliotheken bzw. APIs für die Programmiersprachen gibt. Und natürlich sollte es auch außerhalb von Eigenentwicklungen unterstützt werden, wie beispielsweise dem Betriebssystem.

Ein Verfahren was sich auf die Fahnen schreibt diese Voraussetzungen zur erfüllen ist Kerberos.

Kerberos

Grundlagen und Historie

Kerberos ging aus dem 1983 gestarteten Athena-Projekt des MITs hervor und liegt seit 1993 in der derzeit aktuellen Version 5 (siehe RFC1510) vor. Es ist ein Ticket basiertes Authentifizierungsverfahren, welches innerhalb eines vertrauten Netzwerks eine sichere, gegenseitige Authentifizierung mittels Single-Sign-On ermöglicht. Vertrautes Netzwerk bedeutet dabei keinesfalls, das die Leitungen abhörsicher oder der gleichen sein müssen, sondern das die Dienste explizit in diese Netzwerk aufgenommen werden müssen und beim Ticketserver registriert sein müssen. So kann also ohne Genehmigung nicht jeder einfach einen eigenen Service anbieten. Das Netzwerk kann sich dabei auf mehrere IP-Subnetze oder sogar über das Internet erstrecken, da Kerberos so entwickelt wurde, das ein Angreifer mit abgefangenen Paketen keinen Schaden anrichten kann. Durch diese Registrierung ist auch einen gegenseitige Authentifizierung möglich, sodass sich der Client auch sicher sein kann das er mit dem richtigen Dienst verbunden ist, was bei SSL ja auch verwendet wird. Und letztlich wurde Kerberos auch von Anfang an als SSO-System entworfen und nicht etwa später um diese Funktionalität erweitert.

Das Verfahren basiert vollständig auf symmetrischer Verschlüsselungstechnik. Es gibt also nur einen Schlüssel zum ver- und entschlüsseln. Die heute oft verwendete Asymmetrische Verschlüsselung, bei der es einen öffentlichen und einen privaten Schlüssel gibt, stand damals einfach noch nicht zur Verfügung, doch durch den speziellen Aufbau der Tickets ist diese auch nicht nötig. Was den Vorteil hat das die symmetrischen Algorithmen deutlich weniger CPU-Last erzeugen als ihre asymmetrischen Verwandten, und das ist auch heute noch ein nicht zu unterschätzender Vorteil.

Damit ein Dienst Kerberos-Tickets verarbeiten kann muss dieser zuerst beim Kerberos-Server, der auch Key Distribution Center (KDC) genannt wird, registriert werden. Dazu wird ein Service-Principal erzeugt, welches folgendes Format hat:

```
Dienstkennung/Servername@REALM
```

Also z.B. für HTTP auf dem Host „server.secpod.de“ und in der Kerberos-Domäne (Realm) „SECPOD.DE“:

```
HTTP/server.secpod.de@SECPOD.De
```

Wichtig dabei ist, das der Realm immer groß geschrieben werden muss. Üblicherweise wird hier auch ein Default-Realm hinterlegt, sodass die Angabe dann nicht mehr notwendig ist. Beim erzeugen dieses Service-Principals wird auch ein Service-Key erzeugt, bei dem eine Kopie im KDC gespeichert wird und eine Kopie als sogennante Keytab-Datei exportiert wird. Diese Keytab benötigt der Dienst dann um die Kerberos-Tickets zu validieren. Gleichzeitig ermöglicht dies auch die gegenseitige Authentifizierung, da nur der „echte“ Dienst, der

diese Keytab hat die Tickets öffnen kann. Das Verfahren im Detail zu beschrieben würde jetzt jedoch den Rahmen sprengen.

Verbreitung und API

Wie bereits in der Einleitung erwähnt ist ein SSO-Verfahren nur dann sinnvoll wenn aus auch APIs für die gängigen Programmiersprachen gibt und es bereits eine gewisse Verbreitung bei den Betriebssystem gefunden hat.

Für Kerberos gibt es mittlerweile für die gängigsten Betriebssysteme passende Implementierungen. So wird es z.B. seit Windows 2000 als zentrales Authentifizierungsprotokoll verwendet. Dazu hat Microsoft eine eigene, nahezu Standardkonforme, Implementierung erstellt, sodass also ein Windows Domänencontroller gleichzeitig ein KDC ist. Im Linux/Unix-Bereich stehen diverse Open-Source-Implementierung wie MIT-Kerberos und Heimdal zu Verfügung und auch unter Mac OS X und Solaris gibt es eine Unterstützung. Mit AFS steht sogar ein verteiltes Dateisystem zur Verfügung das auf Kerberos aufbaut.

Auf Seite der Anwendungen gibt es Kerberosunterstützung beim Apache Webserver (mit mod_auth_kerb), MS IIS, OpenSSH, PAM, Samba, OpenLDAP, Dovecot (imap + pop), Postfix, Apple Mail.app und viele mehr.

Eine API für Kerberos gibt es zwar, diese ist jedoch nicht für die direkte Verwendung gedacht. Die Entwickler von Kerberos haben viel mehr eine Abstrakte API entwickelt, welche den Namen GSSAPI trägt, was für Generic Security Services API steht, wobei Kerberos derzeit das einzige Verfahren ist welches diese API nutzt. Für GSSAPI gibt es in vielen Programmiersprachen passende Implementierungen, unter anderem in C und Java. Durch die abstrakte API ist die Anwendung also nicht direkt mit Kerberos verbunden, sodass es ohne Änderung am Anwendungscode austauschbar ist (falls es einmal ein besseres System gibt). Ein weiterer Vorteil ist auch, das die GSSAPI wesentlich einfacher ist als die Kerberos-API.

Zusätzliche zu GSSAPI gibt es noch den SPNEGO (Simple and Protected GSSAPI Negotiaton Mechanism), der das Aushandeln des benutzten Authentifizierungsverfahren festlegt. Dies wird hauptsächlich bei Kerberos via http verwendet, da hier verschiedenste Browser mit verschiedenen Server kommunizieren müssen und dafür ein einheitlicher Weg festgelegt werden musste. Mittlerweile wird SPNEGO via http von Firefox, Safari, Internet Explorer, Apache Webserver (mit mod_auth_kerb) und dem IIS unterstützt.

Es ist also eine ausreichende Verbreitung und Unterstützung für Programmiersprachen vorhanden um Kerberos als Unternehmsweites Single-Sign-On System in Betracht zu ziehen. Durch die standardmäßige Verwendung in Microsoft Windows Netzwerken ist es ohnehin bereits in den meisten Firmen schon im Einsatz, sodass es bei Eigenentwicklung einfach nur benutzt werden muss.

Das Unternehmensweite SSO?

Die Grundlagen sind also soweit klar, doch ist Kerberos wirklich die richtige Wahl um unternehmsweit eingesetzt zu werden? Wie oben bereits erwähnt wird es das sowieso schon, nämlich dann wenn man sich in einem Windows 2000 Netzwerk befindet. Auch bei Solaris ist Kerberos das empfohlene System, sodass auch hier die Wahrscheinlichkeit hoch ist das es bereits eingesetzt wird.

Jetzt kann man sich natürlich fragen wieso es dann nicht schon längst jeder verwendet? Ein Problem ist sicherlich die mangelnde Lektüre dazu. Zu Kerberos selbst gibt es im wesentlichen nur ein Buch (Jason Garmin, Kerberos – The Definitve Guide) und die Artikel im Internet sind auch nicht so zahlreich, bzw. setzen oft schon genauere Vorkenntnisse voraus. Im Bereich der Softwareentwicklung gab es auch längere Zeit keine Unterstützung für Java, und genau damit werden viele Unternehmensanwendungen erstellt. Seit Java 1.4 gibt es diese zwar, allerdings sind auch hier bei der Benutzung detaillierte Kerberos-Kenntnisse notwendig. Es scheitert also auf an dem mangelndem Know-How und nicht an technischen und unternehmerischen Hürden.

Um die Verwendung von Kerberos z.B. in Java zu vereinfachen wird derzeit an einer Integration der GSS-API und SPNEGO in das Open Source Java Security Framework „Spring Security“ (www.springsecurity.org) gearbeitet.

Also sollte es auch von dieser Seite keine echten Probleme geben und wir können uns einige Beispielszenarien ansehen:

Beispiel 1

Das erste Beispiel besteht aus Windows 2000/XP-Clients, einem Windows 2003 Server mit Active Directory und einem Debian GNU/Linux Server mit Apache HTTP.

Das Ziel soll sein, das sich der Benutzer bei einer PHP-Webanwendung, welche auf dem Apache läuft, per SSO transparent authentifiziert.

Da die Clients Windows 2000/XP benutzen steht Kerberos schon mal zur Verfügung, es muss also lediglich am Server gearbeitet werden. Für den Apache Webserver gibt es glücklicherweise ein Modul welches Kerberos mittels SPNEGO versteht. Der Apache muss also um das Modul mod_auth_kerb erweitert werden und es muss ein Service-Principal samt Keytab beim Windows 2003 Server erzeugt werden.

Dazu wird ein Benutzer im Active Directory angelegt und anschließend mittels des Microsoft Tools ktpass dieser Benutzer mit einem Service-Principal verknüpft und die Keytab-Datei erzeugt. Der Service-Principal muss dabei folgendes Format haben:

```
HTTP/HostName@REALM
```

Wobei HostName durch den DNS-Namen des Webservers und REALM durch den Kerberos-Realm (= Windows Domänen Name) ersetzt werden müssen.

Damit wird dann das mod_auth_kerb konfiguriert und kann ab dann auf bestimmte URLs angesetzt werden. Der Apache macht dann mittels SPNEGO eine transparente Authentifizierung und stellt den Benutzernamen als Apache-Umgebungsvariable der Webanwendung zur Verfügung. PHP muss sich also gar nicht um Kerberos kümmern, sondern prüft lediglich die Apache-Variable REMOTE_USER ab.

Der Endeffekt ist, das Benutzer die sich im Firmennetzwerk bereits angemeldet haben auch automatisch bei der Webanwendung angemeldet werden. Das gleiche würde übrigens auch mit Linux und Mac-Clients funktionieren. So können also auch Linux-basierte Dienste nahtlos in ein Windows-Netzwerk integriert werden, ohne das dabei Windows-spezifischer Code verwendetet werden muss, sodass die Clients auch zukünftig nicht auf Windows festgelegt sind.

Benutzern die nicht am Firmennetzwerk angemeldet sind kann entweder der Zugang verwehrt werden oder über Basic-Auth ein Anmeldedialog präsentiert werden, bei dem Sie sich mit der Windows-Kennung anmelden können. Der Apache macht dann in diesem Fall vollständig die Kerberosanmeldung.

Natürlich kann man die Apache-Variable auch mittels Perl oder einem dahinter geschaltetem Apache Tomcat (Java) auslesen.

Beispiel 2

Das zweite Beispiel bezieht sich mehr auf die Systemadministratoren. Auch hier befinden wir uns in einem Windows 2003 Netz mit Linux und Solaris-Servern. Das Ziel soll sein, das sich die Administratoren mit einer Kennung auf allen Systemen anmelden können und das diese Kennung mittels SSH transparent propagiert wird.

Um ein einheitliches Anmelden zu ermöglichen müssen zuerst sämtliche Linux und Solaris-Systeme beim KDC (Windows 2003 Server) registriert werden. Dazu wird also wieder ein Service-Principal samt Keytab erzeugt (siehe Beispiel 1), bei dem HTTP durch host ersetzt wird und der ServerName durch den Hostname (FQDN). Nun wird auf dem Host das PAM-Modul für Kerberos aktiviert und die Authentifizierung läuft ab sofort über Kerberos.

Innerhalb einer solchen Sitzung steht dann auch wieder der SSO-Mechanismus von Kerberos zur Verfügung, sodass z.B. mittels OpenSSH ein transparentes anmelden auf anderen Servern möglich ist. Auch für Windows stehen dabei Patches für den SSH-Client PuTTY zur Verfügung.

Die Autorisierungsinformationen wie Gruppen, UID, etc. kommen dabei weiterhin von dem zuvor verwendeten Systemen wie z.B. einer passwd-Datei oder aus einem LDAP-Server, lediglich die Authentifizierung läuft über Kerberos. Der Benutzer braucht also kein lokales Kennwort mehr welches z.B. in einer

shadow-Datei gespeichert ist.

Dadurch werden die Kennwörter nur an einer Stelle gespeichert und man muss sich nicht bei jeder SSH-Sitzung neu authentifizieren.

Beispiel 3

In diesem Beispiel wird ein eigens in Java entwickeltes Client-/Serverprogramm verwendet, bei dem das Betriebssystem-Login (Windows und Linux!) übernommen werden soll.

Hierzu gibt es einige Lösung, die entweder unsicher oder sehr unsicher sind. Meistens wird aus irgendwelchen Variablen oder Kernelparameter versucht den Benutzername auszulesen und diesen dann auf den Server zu übertragen. Dabei gibt es allerdings zwei Probleme. Erstens, dieses auslesen kann leicht manipuliert werden; zweites, wie soll dieser Benutzername zum Server übertragen werden, ohne das der Benutzer selbst ihn ändern kann? Hier kommen dann oft Verschlüsselungen zum Einsatz, da dies aber auf dem Client selbst passieren muss steht der Schlüssel hier auch zur Verfügung, sodass jeder beliebige Benutzername damit verschlüsselt werden kann. Es ist also nicht einfach damit getan den Namen zu übernehmen, sonder wir benötigen auch eine Unterstützung um diesen sicher zum Server zu bekommen.

Kerberos ist hierfür die erste Wahl, da hier der Benutzername vom KDC bereits verschlüsselt zum Client übertragen wurde, und zwar mit einem Schlüssel, den nur der Service selbst kennt. So sind also Manipulationen jeglicher Art ausgeschlossen. Der Schlüssel ist wie in den anderen beiden Beispielen wieder eine Keytab welche mit einem Service Principal verbunden ist. Damit Kerberos in dieser Umgebung genutzt werden kann muss also im Client das holen eines Service-Tickets implementiert werden. Dieses wird dann zum Server übertragen, welcher es dann auf seine Gültigkeit prüft. Besonders ist hierbei auch, das der Server durch die vorher ausgetauschte Keytab das Ticket ohne Kommunikation mit dem KDC prüfen kann. Das Service-Ticket ist übrigens nur ein Byte Array und kann auf jede beliebige Art und Weise zum Server übertragen werden. Beim Einsatz eines Security Frameworks lässt sich dies oft sogar ohne Änderung im Business-Code durchführen.

Java hat dabei seine eigene Kerberos-Implementierung und muss sich somit nicht auf ein Vorhandensein einer Kerberos-Installation verlassen, weder auf dem Client noch auf dem Server. Wenn es feststellt das auf dem Client kein Kerberos vorhanden ist kann es so konfiguriert werden das ein Anmeldedialog aufgeht und Java die Kerberos-Anmeldung durchführt.

Fazit

Wie man sieht ist Kerberos für den Aufbau eines Unternehmensweiten SSO-Systems bestens geeignet, da die Infrastruktur häufig schon bereits vorhanden ist oder auch leicht aufgebaut werden kann. Die Unterstützung bei den Standardprodukten ist mittlerweile auch schon relativ weit, doch das interessante ist sicherlich das verwenden bei den eigenen Unternehmensanwendungen, was durch die Integration in Java und die Möglichkeit der Nutzung mit Spring Security auch kein echtes Problem mehr darstellt.

Ein Problem kann jedoch das KDC als Single Point of Failure darstellen. Wenn dieses ausfällt ist keine Anmeldung mehr bei sämtlichen Systemen möglich. Deswegen muss es auf jeden Fall redundant ausgelegt werden, was aber alle Implementierung unterstützen. Weiterhin müssen die System auf denen das KDC läuft auch besonders vor Angreifern geschützt werden, da hier eine Kompromittierung fatale Folgen haben kann. Dafür haben dann aber die Dienste selbst keine sensiblen Authentifizierungsdaten mehr, sodass sich die Schutzmaßnahmen auf einen kleinen Kreis von Systemen beschränken.

Die Einführung bzw. der Ausbau von Kerberos bedarf jedoch auch einem gewissen Know-How, da es wie bei jedem SSO-System um sensible Daten geht und sich Fehler im Design auf das ganze Unternehmen auswirken können.

Trotz dieser Nachteile überwiegen meiner Meinung nach deutlich die Vorteile und das nötige Know-How lässt sich entweder im Selbststudium oder mit externen Unterstützung erlangen. Um noch mehr Sicherheit zu erlangen kann auch das komplette System einem Security-Audit unterzogen werden.

Eine häufige Fehlerursache noch am Schluss: Wenn Kerberos auf einmal nicht mehr funktioniert, einfach mal die Uhrzeiten kontrollieren. Diese müssen stets synchron sein (+/- 10 Minuten) da in jedem Ticket ein Timestamp zum Verhindern von Replay-Attacken enthalten ist.