Einblick in 802.1X – Port Based Network Access Control

Anlass: LVA 184.15 LU “Entwurf, Errichtung und Management von Datennetzen”
Lehrender: Univ.Lektor Dipl.-Ing. Dr.techn. Manfred Siegl
Autoren: Martin Jauernig
Stephan Zimmerer
Fridolin Ströhle

Abstract

In dieser Zusammenfassung wird versucht die im Rahmen der Laborübung zum Thema “802.1X – Port Based Network Access Control” gemachten Erfahrungen und gewonnenen Erkenntnisse zu vermitteln.

Der in der Übung an unser dreiköpfiges Team gerichtete simulierte “Auftrag” bestand darin, sich mit der Theorie des Sicherheitsstandards 802.1X vertraut zu machen, und sich in der praktischen Implementierung dieses Standards zu versuchen.

Überblick

Die strukturelle Gliederung richtet sich einerseits nach dem chronologischen Ablauf der Übung, namentlich der Zweiteilung in theoretische und praktische Erforschung des Standards, sowie andererseits nach der logischen Trennung der Themengebiete in die technischen Komponenten die das 802.1X-Setup konstituieren, und die auch ihren Niederschlag in der arbeitsteiligen tiefergehenden Beschäftigung des Teams mit den drei Hauptkomponenten Supplicant, Authenticator und Authentication-Server fand:

A.) Einführung in 802.1X

B.) Die Hauptkomponenten in der Theorie

1. Supplicant
2. Authenticator
3. Authentication-Server

C.) Die praktische Umsetzung

1. Das Setup im Überblick
2. Supplicant
3. Authenticator
4. Authentication-Server

D.) Fazit und Ausblick

Unsere Hauptquellen (siehe Referenzen) waren zum theoretischen Teil das Buch “Implementing 802.1X Security Solutions for Wired and Wireless Networks” von Jim Geier, IEEE Dokumente/Präsentationen sowie diverse Webseiten. Zum praktischen Teil diente uns hauptsächlich eine Webseite des finnischen Forschers Hrn. Teemu Rinta-aho als Vorlage.

A.) Einführung in 802.1X

Im Laufe der 1980er Jahre, nach der sich beschleunigenden Verbreitung von LANs (Local Area Networks) die nach dem IEEE (Institute of Electrical and Electronics Engineers: www.ieee.org) - Standard 802 implementiert wurden, entstand ein Bedarf nach der Möglichkeit den Zugang zu diesen Netzen zu beschränken, nachdem die physische Sicherheit von sich innerhalb von Gebäuden befindlichen Netzen durch häufigeren Bedarf nach Heimarbeit (remote login) und die zunehmende Anbindung an das Internet aufgeweicht wurde. [1]

1998 fand der erste große Schritt zur Port-Basierten Authentifizierung (Definition und Erklärung folgt) mit der Spezifizierung des Extensible Authentication Protocol (EAP) statt, das die Kommunikation zwischen einem zu authentifizierenden Client und dem Authentifizierungsserver festlegt. [2]

Im Jahr 2001 schließlich ratifizierte die IEEE den ursprünglichen 802.1X Standard (802.1X-2001), der auf EAP aufbaut und es zu EAPOL (EAP over LANs) erweitert, und das interface zwischen einem Client und einem Ethernet-Switch/Wireless Access Point definiert - 2004 erhielt der Standard in einer Revision dann seine bis jetzt gültige Fassung. [3]

Was genau spezifiziert nun 802.1X und wie funktioniert es? Hier muss man nun unterscheiden was allgemein üblich unter "802.1X" verstanden wird (und wie auch wir hier es verwenden), und was tatsächlich im Standard behandelt wird. Ersteres umfasst das System zur Zugangsbeschränkung zu einem LAN in seiner Gesamtheit, mit all seinen Komponenten und Protokollen, während letzteres nur ein Teilstück des Systems, nämlich EAPOL beschreibt. [4]

Wenn wir 802.1X sagen beziehen wir uns also “pars pro toto” (der Teil für das Ganze) auf ein System zur port-basierten Zugangsbeschränkung.

Zitat Geier:

“Actually, the control of access to a network involves a host of protocols and standards that are anything but simple. 802.1X is an important component, but several other standards and specifications, written by different organizations, form a complete 802.1X port-based authentication system. As examples, the IEEE standard that applies to port-based authentication is 802.1X, which addresses EAPOL, and the IETC provides RFCs for EAP, EAP-Methods, and RADIUS. All of these standards and specifications are needed to make a port-based authentication system operate. Some people mistakenly think that 802.1X does it all, but actually no single integrated standard
specifies all of the components needed to implement a complete port-based authentication system.” [5]

Nachdem jetzt geklärt sein sollte was 802.1X bezeichnet, gilt es den zweiten wichtigen Begriff zu definieren, der hier schon mehrmals verwendet wurde und schon recht gut “in a nutshell” beschreibt worum es in 802.1X eigentlich geht: “Port Based Network Access Control” - also die port-basierte Zugangsbeschränkung zu einem LAN.

So kommen wir schon zur Grundidee von 802.1X, dass nämlich einem potenziell böswilligen/unerwünschten/unauthorisierten Benutzer, genauer einem unbekannten Client der sich mit dem Netzwerk verbinden will, zum frühestmöglichen Zeitpunkt, bevor er überhaupt die Möglichkeit des Zugriffs auf Ressourcen des Netzwerks hat, der Zugang verwehrt wird, so lange bis seine Identität abgeglichen, und im positiven Fall, er als legitimer Benutzer authentifiziert wurde.

Und dieser Punkt ist eben genau jener eines physischen oder logischen Ports eines Layer-2 Netzwerkgerätes (meist ein Ethernet-Switch oder Wireless Access Point).

Vereinfacht gesagt besteht also die port-basierte Zugangsbeschränkung darin, den Datenverkehr des Clients mit dem Netzwerk über einen Port nur dann zu erlauben, wenn dieser mittels einer Authentifizierungsmethode beweisen kann, dass er dazu berechtigt ist, sonst wird ihm der Zugang verwehrt.

Geier vergleicht diesen Vorgang anschaulich mit dem einer Person die z.b. Zugang zu einem gesicherten Gebäude will, aber schon an der Tür von einem “Gatekeeper”, also z.b. Portier abgefangen wird. Die Person zeigt nun ihren Ausweis, welcher vom Portier zu einer Prüfstelle im Gebäude weitergereicht wird. Erst wenn diese Prüfstelle zustimmt dass die Person die zum Ausweis gehört das Gebäude betreten darf wird die Tür aufgemacht und die Person darf in das Gebäude. [6]

Somit schälen sich auch schon die wesentlichen Komponenten und Protokolle eines 802.1X-Systems heraus, als da wären.

Die Komponenten:

- Ein Benutzer/Client der sich mit dem Netzwerk verbinden will, in der 802.1X Terminologie als Supplicant (etwa: Antragsteller) bezeichnet.

- Ein Netzwerkgerät das den Supplicant an seinem logischen oder physischen Port “abfängt”, und ihm bis zur Authentifizierung den Zugang zum Netzwerk verwehrt (“Gatekeeper”) benannt als Authenticator.

- Eine Ressource innerhalb des Netzwerks die als “Prüfstelle” fungiert und bei Vorliegen des richtigen “Ausweises” den Authenticator anweist den Port für den Supplicant freizugeben, bezeichnet als Authentication Server, oder auch AAA-Server (für Authentication, Authorisation, Accounting). In manchen Implementierungen sind Authenticator und Authentication Server auch in einem Gerät vereinigt. In der Praxis ist der Server sehr oft ein sogenannter RADIUS (Remote Authentication Dial-In User Service)-Server.

Die Protokolle:

- Ein Protokoll, das den Supplicant mit dem Authentication Server kommunizieren lässt, damit dieser ihn authorisieren kann. Dies ist in 802.1X Systemen über verschiedene sogenannte EAP-Methoden verwirklicht, also Implementierungen des Extensible Authentication Protocol.

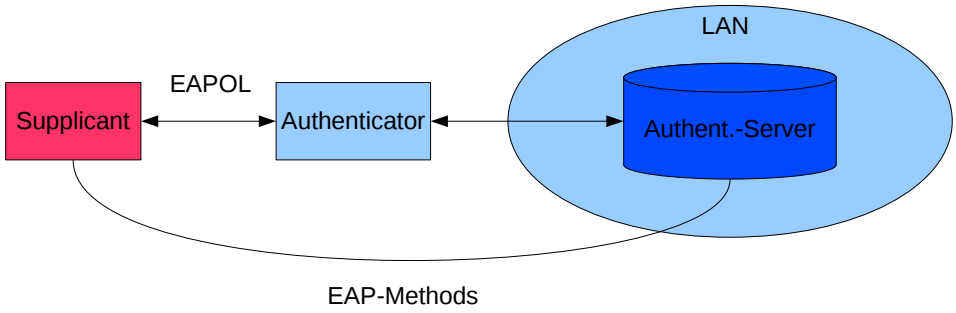
- Ein Protokoll, das den Supplicant mit dem Authenticator kommunizieren lässt, damit dieser in weiterer Folge den EAP-Datenverkehr weiterleiten und dem Supplicant den Port freigeben kann, dies ist realisiert über EAPOL (EAP over LANs).

Wir fassen also zusammen:

Es wird mit 802.1X versucht eine wesentliche Schwäche des 802 Standards zu beseitigen, nämlich dass jemand der physischen Zugang zum Netzwerk hat, praktisch keine weiteren Barrieren vorfindet, da diese im 802 Standard nicht vorgesehen waren.

Dabei wird einem Supplicant (beliebiges Ethernet-fähiges Client-Device, z.b. Desktop, Laptop, Handy...) vom sogenannten Authenticator (z.b. Ethernet-Switch, Wireless Access Point) der Zugang zum LAN dadurch verwehrt dass die Kommunikation über einen logischen oder physischen Port mit dem LAN gesperrt wird, die einzige vorerst erlaubte Kommunikation vom Supplicant zum Authenticator findet über das EAPOL-Protokoll statt, das wiederum die Kommunikation des Supplicants über eine EAP-Methode mit dem Authentifizierungsserver erlaubt. Wenn die Authentifizierung erfolgreich verlaufen ist, entsperrt der Authenticator den Port des Supplicants, und dieser hat normalen Zugriff auf das LAN.

Mehr als tausend Worte:



B.) Die Hauptkomponenten in der Theorie

1) Supplicant

Allgemein

Der Supplicant ist wie bereits erwähnt dafür verantwortlich, die Verbindung mit dem Authenticator aufzubauen und in Folge den für die Authentifizierung notwendigen Handshake durchzuführen. Zu Beginn des Authentifizierungsvorganges ist von Seite des Netzwerks gesehen der Supplicant unbekannt und hat damit keine Rechte.

Wird der Handshake erfolgreich abgeschlossen, schaltet der Authenticator die zum Supplicant gehörige Netzwerkadresse frei und erlaubt dem Gerät damit den Zugang zum gesicherten Netz. Wird der Handshake abgebrochen oder schlägt fehl, kann das Gerät entweder abgewiesen werden, es könnte aber auch, je nach Konfiguration, teil eines öffentlichen VLAN werden, dass etwa lediglich Zugang zum Internet bietet. Zusätzlich steht nun Information über das neu ans Netzwerk angeschlossene Gerät zur Verfügung (Standort, Benutzer, ...) die etwa zur Protokollierung von Nutzeraktivität oder zu Abrechnungszwecken verwendet werden kann. [8]

Protokoll

Die Aufgabe des EAPOL-Protokolls ist der Transfer von EAP-Nachrichten über ein LAN.
Meist handelt es sich bei EAPOL-Nachrichten einfach um EAP-Pakete die mit einem zusätzlichen Header versehen wurden, der vom Empfänger wieder entfernt wird. Die eigentliche Authentifizierung wird mit Hilfe dieser sogenannten Typ 0 EAPOL-Frames abgewickelt. In den EAP-Paketen wiederum wird die „Nutzlast“ der gewählten EAP-Methode transportiert. Daneben existieren noch weitere EAPOL-Frames die für speziellere Aufgaben, wie etwas das übermitteln von Schlüsselmaterial oder das Starten und Beenden des Authentifizierungsvorganges, verwendet werden. [7]
Nachdem ein Supplicant an einen Authenticator angeschlossen wurde, sendet der Authenticator ein EAP Request frame. Der Supplicant beantwortet dieses Paket mit einem EAP Response frame. Die Details des nachfolgenden Datenaustausches werden von der verwendeten EAP-Methode festgelegt. (etwa EAP-TLS). In jedem Fall werden die EAP-Daten die der Supplicant sendet in EAP Response frames verpackt, während der Authenticator die vom Authentication-Server kommenden Daten als EAP Requests weiterleitet.
Sendet der Authenticator nach Herstellung der physischen Verbindung kein EAP Response-Paket, kann der Supplicant die Authentifizierung mit dem weiter oben erwähnten EAPOL-Start frame initialisieren. Genauso kann die Verbindung mit EAPOL-Logoff beendet werden. [6]

Zu beachten ist außerdem, dass EAP standardmäßig keine Verschlüsselung oder sonstige Mechanismen zur Datensicherung bereitstellt. Soll der Authentifizierungvorgang weiter gesichert sein, muss diese Sicherung von Seiten der EAP-Methode erfolgen.

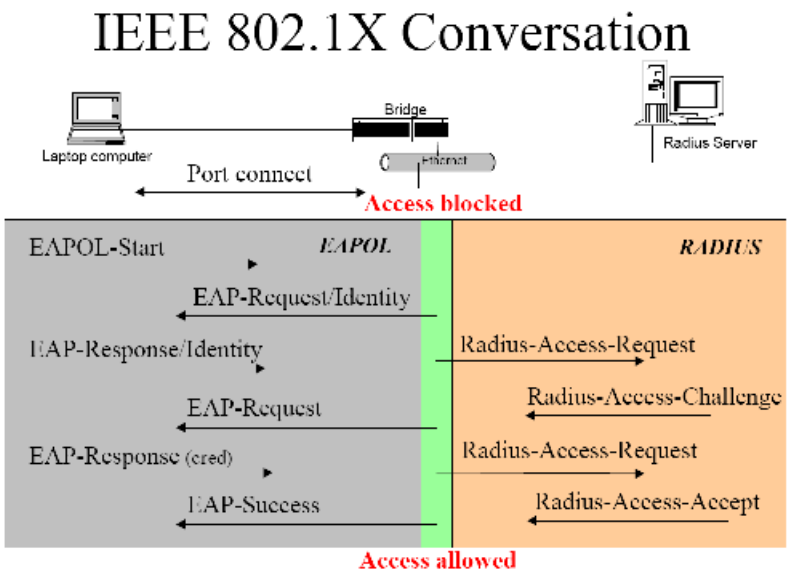
2) Authenticator

Wie schon erwähnt übernimmt der Authenticator einerseits die Rolle des "Gatekeepers" zum zu sichernden LAN, indem einem nicht-authentifizierten Client der logische oder physische Port gesperrt wird, und andererseits ermöglicht er die Kommunikation zwischen dem zu authentifizierenden Client und dem Authentifizierungsserver (mit EAP-Methoden) über das Protokoll EAPOL.

All dies wird über sogenanntes "Layering" erreicht, indem der EAP-Handshake, der zwischen dem Supplicant und dem Authentifizierungsserver stattfinden soll (das eigentliche Ziel der Kommunikation), im ersten Teilstück (vom Supplicant zum Authenticator und vice versa) in EAPOL Frames "verpackt" wird. [7]

Im zweiten Teilstück (vom Authenticator zum Authentifizierungsserver und vice versa) hingegen in das spezifische Serverprotokoll, in der Praxis meist RADIUS. (Zur Erinnerung, in 802.1X sind diese Teile des Systems nicht spezifiziert).

Die Einzelheiten der einzelnen Frames und die Abläufe der Handshakes würde hier zu weit führen, Interessierte seien an die Referenzen verwiesen, aber hier noch ein Bild um die Rolle des Authenticators und den Ablauf der Protokolle zu veranschaulichen (Quelle: [8]):



Zusammenfassend: Der Authenticator blockiert den Port des Supplicants, dieser kann nun ausschließlich mittels EAPOL-Frames dem Supplicant seine EAP-Pakete mitteilen, die dieser z.b. über das RADIUS-Protokoll an den Authentifizierungsserver weiterleitet. Somit ist die indirekte Verbindung zwischen Supplicant und Authentifizierungsserver hergestellt, und sie können den EAP-Handshake vornehmen. Das Ergebnis teilt nun der Server dem Authenticator mit, dieser wiederum öffnet den Port des Supplicants und erlaubt ihm damit den Zugang zum LAN.

Wie gesagt ist der Authenticator physisch meist ein 802.1X-fähiger Switch/Router/WLAN Accesspoint. In letzterem Fall firmiert die Fähigkeit zu 802.1X auch oft unter dem Namen "WPA-Enterprise".

3) Authentication Server

802.1X ist bei der Auslegung des Servers nicht sehr strikt. Die einzige Voraussetzung bildet dabei, dass es sich bei dem Server um einen AAA(Authentication, Authorisation, Accounting)-fähigen Server handeln muss. So ist es durchaus möglich Authenticator und Authentication Server zu vereinen, jedoch werden vor allem bei größeren Netzwerken eigenständige Server verwendet.

In der Praxis werden vorwiegend RADIUS-Server verwendet, welche quasi den de-facto-Standard bilden.

Gute Skalierbarkeit ist ebenso gegeben, da beliebig viele Authentication Server verwendet werden können. Hierzu können jedem Authenticator ein primärer Server und mehrere sekundäre Server zugewiesen werden.

Ob ein Supplicant Zugang zum Netzwerk erhält, entscheidet der Authentication Server, welcher Informationen über die zugelassenen Nutzer besitzt. Da der Supplicant nur indirekt über den Authenticator mit dem Server kommuniziert, muss sich natürlich auch der Authenticator ausweisen können. Dies geschieht zumeist über ein shared-secret.

Da 802.1X nicht genau festschreibt, welcher Server verwendet werden muss, läuft die Kommunikation zwischen Authenticator und Server nicht immer gleich ab und ist vom verwendeten Server abhängig. Da in der Praxis allerdings fast ausschließlich RADIUS Server verwendet werden, sei hier dieses Beispiel mittels RADIUS Protokoll kurz gezeigt, wobei hier nicht auf den exakten Aufbau der Message Frames eingegangen wird (Quelle [16]):

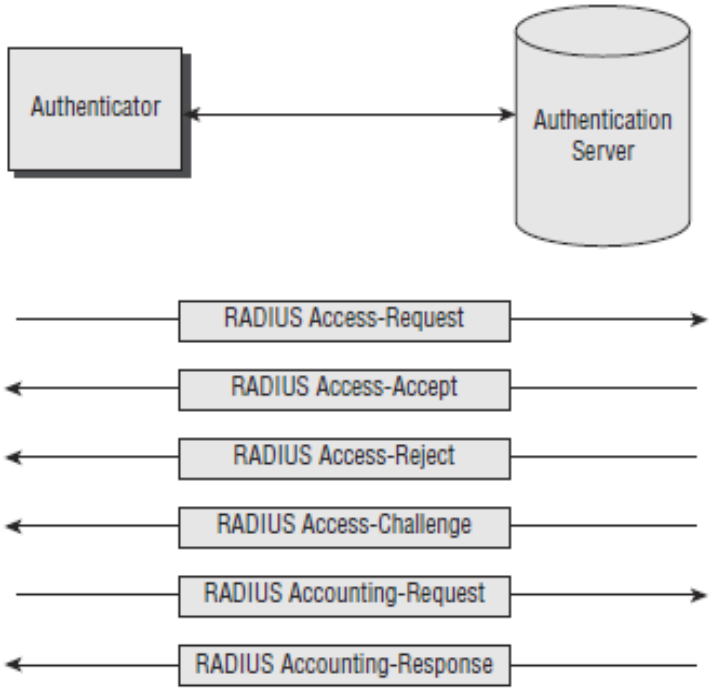


Figure 2-12: Communications between the authenticator and the authentication server

C.) Die praktische Umsetzung

1) Das Setup im Überblick

Die Anforderungen die wir uns selbst bezüglich der Umsetzung eines 802.1X Test-Setups stellten waren folgende:

- **es sollte mit uns zur Verfügung stehender Hardware realisierbar sein**
- **die verwendete Software sollte wenn möglich "freie" (Open-Source) Software sein**

Der Supplicant und Authentication Server stellten dem Anschein nach keine Schwierigkeit dar, da für diese mehrere frei erhältliche Softwarepakete sowohl für Windows alsauch für Unix Betriebssysteme zur Verfügung standen.

Der Authenticator bereitete uns schon mehr Probleme, da dieser, wie schon erwähnt, in der Praxis meist ein 802.1X-fähiger Router ist, und ein solcher stand uns nicht zur Verfügung. Hier eröffneten sich nun zwei prinzipiell von uns als gangbar gesehene Wege:

- **einen vorhandenen Router durch das bespielen mit frei erhältlicher Firmware 802.1X-fähig zu machen (z.b. mit OpenWrt [9]).**
- **mittels Software einen Authenticator zu "emulieren".**

Angesichts der Risiken der ersten Option (Stichwort "bricking") entschieden wir uns für Letzteres, was uns zusätzlich noch den Vorteil bot, dass wir quasi in den Authenticator "hineinschauen" konnten, konkret heißt das die Handshakes live verfolgen und mit Wireshark den Datenverkehr protokollieren konnten, was sich auch beim Debugging als hilfreich erwies.

Dies wurde durch die Software hostapd [10] ermöglicht, die einen Wireless Access Point auf einem Rechner mit Linux-Betriebssystem emuliert.

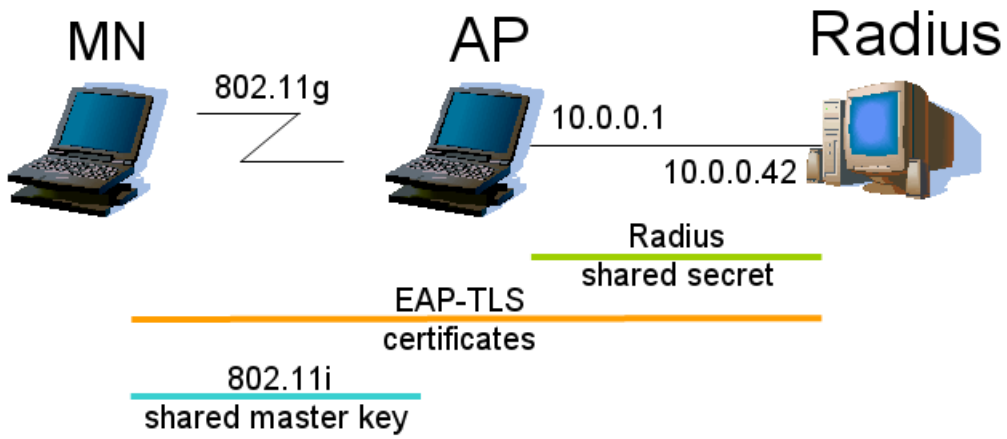
Als Supplicant dienten uns sowohl ein Rechner mit Windows/Linux Betriebssystem und passender Supplicant-Software (wobei sich nur jene unter Linux von uns erfolgreich implementieren ließ), alsauch ein Smartphone mit Android Betriebssystem.

Als Authentifizierungsserver verwendeten wir nach den nach eigener Aussage weit verbreitetsten RADIUS-Server der Welt, FreeRadius, ebenfalls auf einem Linux-Rechner.

Als EAP-Methode wählten wir EAP-TLS, allgemein als eine der sichersten Methoden angesehen.

Bei den Recherchen stießen wir auf die Webseite von Hrn. Teemu Rinta-aho [11], der genau so ein Setup realisiert und sehr genau beschrieben hat, was sich als großer Glücksfall für uns erwies. Im Wesentlichen haben wir seine Konfiguration übernommen, bis auf einige kleine Änderungen und zwei wesentlichen Erweiterungen, nämlich dass der Authenticator mittels einer simulierten Bridge in die Lage versetzt wurde nach erfolgter Authentifizierung dem Supplicant Zugriff auf das simulierte LAN zu geben (dies wurde mittels Ping zum Server überprüft), und einem Router der zwischen Authenticator und Server gesetzt wurde, und auch als DHCP-Server diente.

Bild des wesentlichen Setups (Quelle: [11]):



MN steht für "Mobile Node", also den Supplicant, AP für Access-Point, also den Authenticator.

Die EAP-Methode EAP-TLS benötigt Zertifikate sowohl für den Server alsauch den Supplicant. In der Praxis werden das meist von einer CA (Certificate Authority) beglaubigte Zertifikate sein, für unsere Zwecke genügten sogenannte self-signed certificates, die wir mit einem Script erstellten das ebenfalls unter [11] zu finden ist. Als Hürde erwiesen sich die unterschiedlichen Zertifikat-Formate für die verschiedenen Betriebssysteme, das Ausgangsformat war ".pem" für Linux, für den Android Supplicant ließ es sich erfolgreich in das ".p12" Format konvertieren, beim Windows-Supplicant stießen wir diesbezüglich auf Schwierigkeiten.

2) Supplicant

Ansatz 1: Xsupplicant unter Windows [19]

Xsupplicant ist ein frei verfügbarer Supplicant, der eine ganze Reihe von Protokollen unterstützt (unter anderem auch 802.1X für WLAN genauso wie für LAN). Da das primäre Betriebssystem des als Client vorgesehenen Computers Windows ist, wurde die Windows-Version von Xsupplicant verwendet. Über die graphische Oberfläche des Supplicant können die Parameter des Netzwerks eingestellt werden, zu dem eine Verbindung aufgebaut werden soll (Name, verwendete Verschlüsselung, ...). Es ist aber nicht möglich, neue Zertifikate als Identitätsmerkmal des Supplicant zu importieren. Es kann lediglich ein von Windows verwaltetes Zertifikat ausgewählt werden.

Unter Windows werden Zertifikate im allgemeinen Zentral von der „Microsoft Management Console“ verwaltet. Diese Konsole ist zwar relativ flexibel was das Format der Zertifikate anbelangt, sie verlangt aber nach zumindest einem Wurzelzertifikat, dass als solches dem System bekannt sein muss. Es ist durchaus möglich, ein Client-Zertifikat zu importieren, allerdings wird dies nur als vollwertiges Identitätsmerkmal erkannt, wenn es von einer vertrauenswürdigen Stelle signiert ist. Ist das (selbstsignierte) Root-Zertifikat, wie in unserem Fall, nicht korrekt importiert worden, kann Xsupplicant nicht auf das für diesen Zweck vorbereitete Client-Zertifikat zugreifen. Das Programm kann in Folge den Authentifizierungsprozess nicht starten.

Ausgelöst wurde dieses Missverständnis vermutlich durch die nicht besonders detaillierte Dokumentation der Windows-Version von Xsupplicant.

Ansatz 2: wpa_supplicant unter Linux

Im Gegensatz zu Xsupplicant ist wpa_supplicant ein Client, der auf WLANs spezialisiert ist. Neben dem 802.1X-Handshake übernimmt das Programm auch die Verschlüsselung der übermittelten Daten, ermöglicht Roaming.

Die Konfiguration des wpa_supplicant erfolgt über eine oder mehrere Konfigurationsdateien, in denen nicht nur Name und Art des gewünschten Netzwerks angegeben sind, sondern in denen auch Daten zur Identität des Benutzers, inklusive der Pfade zu den zu verwendenden Zertifikaten, eingetragen werden können. Hier sei ein beispielhafter Ausschnitt aus der von uns verwendeten Konfigurationsdatei angeführt [10, 20]:

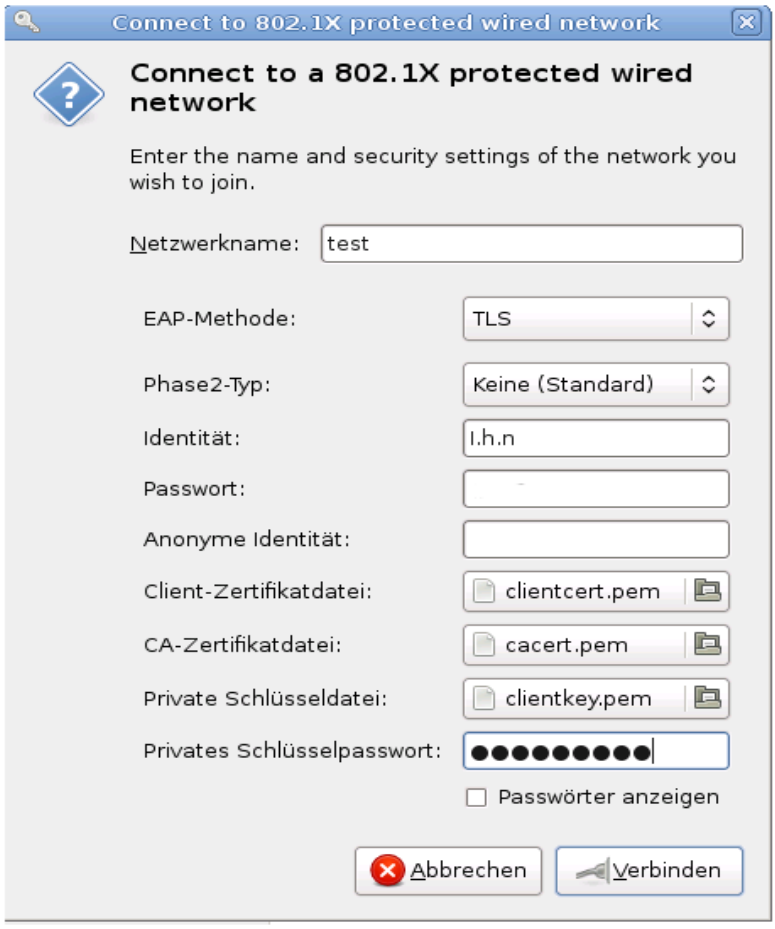
```
# path to UNIX socket control interface
```

```
ctrl_interface=/var/run/wpa_supplicant

# Scan for accesspoints, this may be required for hidden ssids
eapol_version=1
ap_scan=1
fast_reauth=1

network={
   scan_ssid=0
   ssid="test"
   proto=RSN # == WPA2/IEEE 802.11i (also WPA2 can be used as an alias for RSN)
   key_mgmt=WPA-EAP
   auth_alg=OPEN # list of allowed IEEE 802.11 authentication algorithms
   eap=TLS
   identity="I.h.n"
   ca_cert="/home/knoppix/Desktop/clientconf(pw4client)/cacert.pem"
   client_cert="/home/knoppix/Desktop/clientconf(pw4client)/clientcert.pem"
   private_key="/home/knoppix/Desktop/clientconf(pw4client)/clientkey.pem"
   private_key_passwd="pw4client"
}
```

Auf Seite der Authentifikations-Infrastruktur muss „client_cert“ bekannt und mit dem in „identity“ angegebenen String assoziiert sein.

Der start von wpa_supplicant mit einer solchen Konfigurationsdatei führte in unserem Fall zu einem ersten erfolgreichen Handshake zwischen Authenticatior und RADIUS-Server und dem Client-Computer.

Im weiteren Verlauf hat sich jedoch gezeigt, dass der mit Knoppix 6.0 mitgelieferte Netzwerkmanager als frontend für wpa_supplicant fungieren kann. Das folgende Menü ist über das Programm „nm-applet“ unter „Connect to 802.1X protected wireless/wired network“ zu erreichen:



Dies macht die Verwendung des wpa_supplicant unter Linux relativ einfach, zumindest in der von uns gewählten, relativ rudimentären Konfiguration. Allerdings ist durch die Struktur von EAP auch bei steigender Komplexität der Netzwerkarchitektur nicht zu erwarten, dass sich der Aufwand des Verbindungsaufbaus aus der Sicht des Anwenders beziehungsweise des Supplicant drastisch erhöht, zumal der Supplicant immer nur in Kontakt mit einer Stelle (dem Authenticator) steht. Schwierigkeiten könnten jedoch wie bereits erwähnt dann auftreten, wenn unterschiedliche Architekturen und Systeme unterschiedlicher Hersteller miteinander verbunden werden sollen.

3) Authenticator

Wie schon erwähnt wurde der Authenticator von uns mittels der Software hostapd [10] emuliert. Voraussetzung dafür ist allerdings ein WLAN-Interface das sich in den Master-Mode (als Access Point) schalten lässt, bzw. mit dem richtigen Chipset, glücklicherweise befand sich eine solche Karte in unserem Besitz, eine Liste der kompatiblen Karten findet man z.b. unter [12].

Hostapd ist 802.1X-fähig und hat sogar die Möglichkeit eines eingebauten freeradius-Servers, d.h. theoretisch hätten wir sogar auf einen separaten Server verzichten können, wir taten dies aber im Interesse der Erfahrungssammlung nicht.

Es ist im Grunde sehr einfach zu konfigurieren, es genügt ein einzelnes Konfigurationsfile, hier das von uns verwendete:

```
#das verwendete Wireless Interface

interface=wlan0
# die verwendete Bridge
bridge=br0

# Wo und in welchem Umfang werden Log-Meldungen ausgegeben
logger_stdout=-1
logger_stdout_level=0

# der verwendete Treiber
driver=nl80211

# Die Einstellungen für den Access-Point: SSID, Kanal, Details der verwendeten
# Wireless Einstellungen des Access-Points.
ssid=test
channel=1
hw_mode=g
ieee80211n=0
macaddr_acl=0
auth_algs=3
ieee8021x=1

# EAPOL-Key index workaround (set bit7) for WinXP Supplicant (needed only if
# only broadcast keys are used)
```

```
#eapol_key_index_workaround=0

# Die IP-Adresse des Authenticators
own_ip_addr=192.168.1.103

# Details des RADIUS authentication server
# IP-Adresse, Port und shared secret (Passwort)
auth_server_addr=192.168.1.101
auth_server_port=1812
auth_server_shared_secret=pw4radius

# Vom Access-Point akzeptierte WPA-Standards und Schlüsselmethoden
wpa=3
wpa_key_mgmt=WPA-EAP
wpa_pairwise=CCMP TKIP
```

Die angeführte emulierte Bridge "br0" wurde mittels des Linux-Befehls "brctl" erzeugt (in unserem Fall eine Bridge vom Wireless Interface "wlan0" zum Wired Interface "eth0"), die genauen Befehle zum Erzeugen einer Bridge findet man z.b. unter [13].

4) Authentication Server

Wir haben uns bei unserem Server für FreeRadius, da es sich dabei um einen etablierten und weit verbreiteten Open Source Server handelt. Dieser wird bei vielen großen Netzwerken verwendet, unter anderem auch eduroam (sieher [17]).

Bei unserem Server handelte es sich um einen Linux Rechner, auf dem FreeServer 2.1.10 lief. [18]

Wie bereits erwähnt benötigt der Server für EAP-TLS Zertifikate, welche wir mittels [11] erstelllt haben.

Weiters werden noch Konfigurationsdateien für die Clients, die User und die EAP Methode benötigt:

clients.conf:

```
# WLAN Access Points in 192.168.0.0/8 network
client 192.168.0.0/8 {
    secret      = pw4radius
    shortname   = test
}
```

users:

```
droid1      Auth-Type := EAP
```

eap.conf:

```
eap {
            default_eap_type = tls
            timer_expire     = 60
            ignore_unknown_eap_types = no
            cisco_accounting_username_bug = no

            md5 {
```

```
        }
        leap {
        }
        gtc {
              auth_type = PAP
        }
        tls {
              private_key_password = pw4server
              private_key_file = ${raddbdir}/certs/serverkey.pem
              certificate_file = ${raddbdir}/certs/servercert.pem
              CA_file = ${raddbdir}/certs/cacert.pem
              dh_file = ${raddbdir}/certs/dh
              random_file = ${raddbdir}/certs/random
              fragment_size = 1024
        }
        mschapv2 {
        }
}
```

D.) Fazit und Ausblick

Hier, zur Demonstration der praktischen Implementierung, ein Screenshot vom Authentifizierungs-Handshake der mittels Wireshark am Authenticator (genauer an der virtuellen Bridge br0) gecaptured wurde (MAC-Adressen aus Sicherheitsgründen unkenntlich gemacht). Die Motorola MAC gehört zum Supplicant (Android-Smartphone), die Netgear MAC zum Wireless Interface des Authenticators, welcher die IP-Adresse 192.168.1.103 hat, der RADIUS-Server hingegen die IP 192.168.1.101:

239	319.416161	Motorola_f8:[redacted]	Netgear_5d:[redacted]	EAP	Response, Identity [RFC3748]
240	319.416352	192.168.1.103	192.168.1.101	RADIUS	Access-Request(1) (id=16, l=150)
241	319.417345	192.168.1.101	192.168.1.103	RADIUS	Access-challenge(11) (id=16, l=64)
242	319.431592	Motorola_f8:[redacted]	Netgear_5d:[redacted]	TLSv1	Client Hello
243	319.431796	192.168.1.103	192.168.1.101	RADIUS	Access-Request(1) (id=17, l=249)
244	319.476678	192.168.1.101	192.168.1.103	RADIUS	Access-challenge(11) (id=17, l=1090)
245	319.489235	Motorola_f8:[redacted]	Netgear_5d:[redacted]	EAP	Response, EAP-TLS [RFC5216] [Aboba]
246	319.491878	192.168.1.103	192.168.1.101	RADIUS	Access-Request(1) (id=18, l=163)
247	319.493085	192.168.1.101	192.168.1.103	RADIUS	Access-challenge(11) (id=18, l=1090)
248	319.504866	Motorola_f8:[redacted]	Netgear_5d:[redacted]	EAP	Response, EAP-TLS [RFC5216] [Aboba]
249	319.505040	192.168.1.103	192.168.1.101	RADIUS	Access-Request(1) (id=19, l=163)
250	319.506232	192.168.1.101	192.168.1.103	RADIUS	Access-challenge(11) (id=19, l=553)
251	319.660731	Motorola_f8:[redacted]	Netgear_5d:[redacted]	TLSv1	Certificate, Client Key Exchange, Certificate Verify, Change Cipher
252	319.660864	192.168.1.103	192.168.1.101	IP	Fragmented IP protocol (proto=UDP 0x11, off=0, ID=e317)
253	319.660877	192.168.1.103	192.168.1.101	RADIUS	Access-Request(1) (id=20, l=1489)
254	319.697337	192.168.1.101	192.168.1.103	RADIUS	Access-challenge(11) (id=20, l=127)
255	319.701764	Motorola_f8:[redacted]	Netgear_5d:[redacted]	EAP	Response, EAP-TLS [RFC5216] [Aboba]
256	319.701906	192.168.1.103	192.168.1.101	RADIUS	Access-Request(1) (id=21, l=163)
257	319.702836	192.168.1.101	192.168.1.103	RADIUS	Access-Accept(2) (id=21, l=168)
258	319.717128	Motorola_f8:[redacted]	Netgear_5d:[redacted]	EAPOL	Key
259	319.723522	Motorola_f8:[redacted]	Netgear_5d:[redacted]	EAPOL	Key

Wie ist nun das Ergebnis unserer Beschäftigung mit 802.1X und welche Themen wären noch zu behandeln gewesen?

Nach den Einblicken in die Theorie hatten wir genügend Respekt vor 802.1X um die praktischen Versuche nicht auf die leichte Schulter zu nehmen, Zitat Geier:

“802.1X port-based authentication is much more difficult to understand and implement than is commonly thought. In fact, the majority of administrators and engineers have difficulty making

these systems work properly. A lack of understanding and experience with 802.1X solutions is primarily why these troubles exist." [14]

Die Implementierung unseres Testsystems erwies sich dann nicht als so schwierig wie befürchtet, vor allem weil wir auf sehr gute Anleitungen wie die von Hrn Rinta-aho zurückgreifen konnten, hauptsächlich aber weil unsere wenigen, aufs Wesentlichste reduzierten Komponenten wohl in keiner Weise mit einem professionell betriebenen 802.1X System zu vergleichen ist, weder von der Anzahl der Komponenten, noch der Komplexität der Konfiguration.

Eine von FreeRadius gemachte Umfrage hat beispielsweise ergeben dass 50% der mit FreeRadius implementierten Systeme zwischen 100 und 10000 authentifizierte Nutzer haben. [15]

Einen kleinen Vorgeschmack der Probleme die auftreten können haben wir ja schon in unserem kleinen System bekommen, als wir Schwierigkeiten hatten den Windows-Supplicant ad-hoc zum Laufen zu bringen, und wir sind uns bewusst dass bei einer Skalierung um einige Größenordnungen ein ungleich höherer Aufwand zu betreiben wäre.

Einige Fragestellungen die man evt. noch weiter erforschen hätte können:

- Wie sicher ist unser System wirklich, versuchen es zu knacken (Stichwort "Man in the Middle"-Attacke), bei EAP-TLS angeblich unmöglich, aber z.b. bei anderen EAP-Methoden.

- Wie schauen die harten Fakten bzgl. der Performance unseres Systems aus, z.b. auch im Vergleich mit dediziertem Switch/Router als Authenticator statt unserer "simulierten" Lösung.

- Welche Authentifizierungsmethoden gibt es sonst noch, wie schneidet 802.1X im Vergleich ab (Aufwand der Implementierung, Komplexität der Wartung, Sicherheit...)

Insgesamt glauben wir ein recht positives Fazit bezüglich des Eindrucks den 802.1X auf uns hinterlassen hat ziehen zu können, und dass sich die Erfahrungen die wir im Laufe der Übung sammeln durften für uns persönlich auf jeden Fall gelohnt haben.

Referenzen

(1) Geier, Jim: “Implementing 802.1X Security Solutions for Wired and Wireless Networks”, *Wiley Publishing Inc*, 2008, S.51

(2) Geier, ebd.

(3) Geier, ebd.

(4) Geier, a.a.O., S. XXI

(5) Geier, ebd.

(6) Geier, a.a.O., S. 42ff

(7) Geier, a.a.O., S. 40ff

(8) Congdon, Paul: "IEEE 802.1X Overview" (Präsentation), http://www.ieee802.org/1/files/public/docs2000/P8021XOverview.PDF, Letzter Zugriff: 10.01.2011

(9) http://openwrt.org/, Letzter Zugriff: 10.01.2011

(10) http://hostap.epitest.fi/hostapd/, Letzter Zugriff: 10.01.2011

(11) http://www.rinta-aho.org/docs/wlan/wlan.html, Letzter Zugriff: 10.01.2011

(12) http://wireless.kernel.org/en/users/Documentation/hostapd, Letzter Zugriff: 10.01.2011

(13) http://www.linux.com/archive/feed/55617, Letzter Zugriff: 10.01.2011

(14) Geier, a.a.O., S. XXIII

(15) http://freeradius.org/press/survey.html, Letzter Zugriff: 11.01.2011

(16) Geier, a.a.O., S.50

(17) http://www.eduroam.org/, Letzter Zugriff 12.01.2011

(18) http://freeradius.org/download.html, Letzter Zugriff 12.01.2011

(19) http://open1x.sourceforge.net/, Letzter Zugriff: 12.01.2011

(20) wpa_supplicant.conf manpage