Gerätekommunikation via Virtual Private Networks (VPN)

Ein „Stand alone“-Protokollstapel für den Embedded-Bereich

Von Nathan Braun, Thomas Gillen und Axel Sikora

Die Einbindung von Embedded-Systemen in Netzwerke bringt zahlreiche Vorteile mit sich. Damit jedoch die Endsysteme und das Gesamtsystem nicht kompromittiert werden können, müssen verschiedene Sicherheitsaspekte berücksichtigt werden. Dieser Beitrag beschäftigt sich mit grundsätzlichen Architekturen und konkreten Lösungen zu diesem Problem.

Embedded-Systeme werden zunehmend vernetzt; hierbei tritt neben die spezifischen Feld- und Industriebusse verstärkt die Nutzung des Internet-Protokolls, da dieses die Verwendung standardisierter Schnittstellen und Applikationen eines praktisch überall verfügbaren Netzes auf kostengünstige Weise erlaubt. Hierdurch eröffnet sich aber auch für Embedded-Systeme ein Sicherheitsrisiko, das mit einer Anbindung an das öffentliche Internet steigt:

- Mit der Nutzung standardisierter Internet-Protokolle steigt die Wahrscheinlichkeit systematischer Angriffe auf das System, z.B. durch Port-Scanner, Viren oder „Denial of Service“-Attacken.
- Die Vertraulichkeit der Daten ist im öffentlichen Internet nicht gegeben.
- Drahtlose Netze lassen sich auch physikalisch einfach abhören und angreifen. Vor dem Hintergrund der immer weiter vordringenden drahtlosen Netze auch im Embedded-Bereich [1] wird ein umfassendes Sicherheitskonzept immer wichtiger.

Problematisch erscheint in diesem Zusammenhang, dass Embedded-Systeme in der Regel über eine sehr viel geringere Leistungsfähigkeit verfügen als PC-basierte Systeme [2]. Zudem werden diese oft über Jahre hinweg betrieben. Dies stellt besondere Anforderungen an die Architektur, da heutige Implementierungsentscheidungen noch in Jahrzehnten Bestand haben müssen. Darüber hinaus arbeiten Embedded-Systeme autonom und in der Regel ohne die permanente Betreuung durch einen Administrator. Embedded-Systeme verwalten dabei nicht nur Daten, sondern häufig auch Geräte und Anlagen. Die zunehmende Verbreitung und die eingeschränkten Schutzmechanismen machen Embedded-Systeme zu attraktiven Angriffszielen. Es wäre aber falsch, den Weg der über das Internet-Protokoll gekoppelten verteilten Embedded-Systeme nicht weiter zu verfolgen. Ein sicherer Betrieb auch von vernetzten Embedded-Systemen ist durchaus möglich. Allerdings ist hierfür eine Reihe von Regeln zu beachten.

Sicherheitsziele

Sicherheit kann beschrieben werden als „Zustand des Nichtvorhandenseins von oder Schutz vor Gefahren und Risiken“. Sicherheit ist damit aber eine nur subjektiv wahrnehmbare Größe, die weder direkt sichtbar noch messbar ist. Dabei umfasst der deutsche Begriff „Sicherheit“ zwei Aspekte:

- „Safety“ bezieht sich auf die Zuverlässigkeit eines Systems, z.B. auf dessen Ablauf- und Ausfallsicherheit.
- „Security“ bezieht sich auf den Schutz eines Systems vor beabsichtigten Angriffen.

Bild 1. Parameter der Sicherheit und ihre wechselseitigen Abhängigkeiten. Das Diagramm ist folgendermaßen zu lesen: Bei Verlust der Vertraulichkeit kann auch die Authentifizierung verloren gehen. Bei Verlust der Authentifizierung kann auch die Autorisierung verloren gehen.

Eine sichere Kommunikation von Geräten in Netzwerken schließt im Sinne von „Security“ mindestens die folgenden Aspekte ein, deren wechselseitige Abhängigkeit in Bild 1 dargestellt ist. Vertraulichkeit (Confidentiality, Privacy) bezeichnet die Sicherheit gegen den unerlaubten Zugriff auf Informationen, der zu einem Informationsgewinn beim Abhörenden führt. Integrität (Integrity) hingegen umfasst den Schutz gegen die meist partielle Veränderung von Informationen. In Bezug auf die Funktion eines Netzwerks kann sich die Integrität sowohl auf die Inhalte von Datenpaketen als auch auf deren Steuerinformationen und hierbei insbesondere auf die Adressierung beziehen. In diesem Sinne ist Integrität mit der Authentifizierung (Authentication) eng verbunden. Dabei wird überprüft, ob ein Sender wirklich derjenige ist, der dieser zu sein vorgibt. Die Autorisierung (Autorisation) hingegen beschreibt die Zugangssteuerung, so dass bestimmte Informationen oder Dienste nur einem beschränkten Kreis von authentifizierten Nutzern zur Verfügung stehen. Verbindlichkeit oder Unabstreitbarkeit (Non-Repudiation) gewährleistet, dass ein Subjekt die durchgeführten Aktionen nicht abstreiten kann, während Verfügbarkeit (Availability) und Zugang (Access) besagen, dass Informationen nur dort und dann zugänglich sind, wo und wann sie von Berechtigten gebraucht werden. Unberechtigte haben weder einen Zugang zu den Informationen, noch sind sie in der Lage, die Verfügbarkeit der Informationen zu stören.

Embedded Virtual Private Networks

Eine zentrale Bedeutung bei den Sicherheitsmaßnahmen weisen die Verfahren des „Virtual Private Networking“ (VPN) auf, wie sie aus der Bürowelt bekannt sind. Dabei wird ein kryptographisch gesicherter „Tunnel“ durch das unsichere Netz für die Datenübertragung genutzt. Auf diese Weise werden in der Regel Vertraulichkeit und Integrität geschützt. Hierfür steht eine Vielzahl von

Protokollen zur Verfügung, die sich je nach Anwendungsfall nutzen lassen (Bild 2). Grundsätzlich werden temporäre und statische Tunnel unterschieden.

Statische Tunnel werden für längere Zeit zwischen zwei Netzen oder Systemen aufgebaut. Hier kommen symmetrische Verschlüsselungsverfahren zum Einsatz, d.h., an beiden Endpunkten des Tunnels werden geheime Schlüssel hinterlegt. Solche Tunnel lassen sich realisieren mit Hilfe des Secure Internet Protocol (IPsec) auf der Vermittlungsschicht (Network, OSI-Schicht 3) oder mit Hilfe von PPTP oder L2TP auf der Sicherungsschicht (OSI-Schicht 2, Data Link Layer).

Bild 2. Virtuelle Private Netze (VPN) können mit einer Vielzahl von Protokollen auf den unterschiedlichen Netzwerk-Ebenen unterstützt werden.

Während die statischen Verfahren für die Kopplung von Standorten sowie zur Anbindung abgesetzter Stationen eingesetzt werden können, sind sie für eine sichere Verbindung zwischen wechselnden Partnern weniger geeignet. Hier bieten sich Protokolle an, die auf hybriden Verfahren beruhen. Diese kombinieren asymmetrische kryptographische Verfahren zum Schlüsselaustausch mit symmetrischen Verfahren für den Datenaustausch. Solche temporären Tunnel werden z.B. mit Hilfe des „Secure Socket Layer"-Protokolls (SSL) auf der Transportschicht (OSI-Schicht 4) oder mit Hilfe von Secure Shell (SSH) oder daraus abgeleiteten Tunneln auf der Anwendungsschicht (OSI-Schicht 7) realisiert.

Authentifizierung

Viele Embedded-Systeme werden heute noch ohne Authentifizierung betrieben. Wird diese eingesetzt, so ist die Standardmethode bei Embedded-Systemen immer noch die Eingabe von Benutzernamen und Passwort; dabei wird dort in der Regel wegen des Pflegeaufwands statt individueller Benutzernamen häufig ein funktionaler „Username" von mehreren Nutzern gemeinsam verwendet, wie z.B. „Administrator", „Factory" oder „Service". Damit verbunden sind einheitliche Passworte, die einem mitunter weiten Personenkreis bekannt sind und so trotz der Verschlüsselung der Verbindung ein erhebliches Sicherheitsrisiko darstellen. Nachteilig ist auch, dass die an sich geheimen Passworte auf vielen Systemen mehr oder weniger offen gespeichert werden müssen. Ein Implementierungsbeispiel für eine solche – kryptographisch nicht abgesicherte – HTTP-Authentifizierung von Webseiten ist in [3] vorgestellt; dort werden die Login-Daten im Klartext über das Netz übertragen. Verbesserung bringen hier schon die auf Hash-Code beruhenden Authentifizierungsverfahren, bei denen für die Authentifizierung nur verschlüsselte Informationen über das Netz übertragen werden.

Ein noch höheres Sicherheitsniveau bringt die zertifikatsbasierte Authentifizierung. Ein Zertifikat ist ein öffentlicher Schlüssel, der zusammen mit den Informationen über den Zertifkats-inhaber, die Gültigkeit oder den Verwendungszweck durch eine Zertifizierungstelle (Certification Authority, CA) signiert ist. Jeder „User" verfügt dabei über ein individuelles Zertifikat mit dem dazugehörigen privaten Schlüssel. Das Embedded-System benötigt für die Verifikation der „User"-Zertifikate nun, unabhängig von der Zahl der ausgegebenen „User"-Zertifikate, nur die Zertifikate aller akzeptierten Zertifizierungsstellen. Bei entsprechender Gestaltung der „User"-Zertifikate können eine oder mehrere „User"-Rollen zusätzlich integriert werden; dies erleichtert die Auswertung auf dem Embedded-System. Entscheidend ist, dass auf dem Embedded-System keine weitere geheim zu haltende Information zur Authentifizierung gespeichert werden muss; aus Sicht des Embedded-Systems erfolgt diese vollständig mit dem öffentlichen Material.

Da die Client-Authentifizierung auf dem gleichen RSA-Mechanismus basiert, der bei SSL ohnehin verwendet wird, hält sich der Mehraufwand für die Software in Grenzen. Verglichen mit dem Aufwand für den Verbindungsaufbau ist der zusätzliche Zeitbedarf für die Authentifizierung recht gering.

Firewall

Ein Paketfilter (Firewall) beschränkt den Verkehr zu einem Knoten oder Netz auf den vermutlich sinnvollen Verkehr. Hierdurch soll die Verfügbarkeit der Systeme erhöht werden, indem nur erlaubte Dienste von außen angesprochen und Denial-of-Service-Angriffe verhindert werden können. Embedded-Systeme können nur in den seltensten Fällen mit einem separaten Paketfilter ausgestattet werden, da von vorneherein klar ist, welche Dienste unterstützt und welche Ports dazu geöffnet werden müssen. Daher spielt hier die Qualität der TCP/IP-Implementierung die entscheidende Rolle, die nur den erlaubten Verkehr annehmen darf und alle anderen Anfragen ignorieren muss. Das Sicherheitskonzept steht und fällt mit dieser Komponente.

Weitere Funktionen zur Erkennung oder Vermeidung von Angriffen durch „Intrusion Detection"-Systeme (IDS) bzw. „Intrusion Prevention"-Systeme (IPS) scheitern in der Regel an dem zu hohen Implementierungsaufwand bei der Abspeicherung statistischer Daten.

Protokoll-Beschreibung SSL

Von den in Bild 2 dargestellten Protokollen wird hier das „Secure Socket Layer"-Protokoll (SSL) ausführlich beschrieben, da es eine Reihe von Vorteilen bietet. Es unterstützt die flexible Kommunikation zwischen wechselnden Partnern, da Schlüssel über ein unsicheres Medium ausgetauscht werden können, und bietet die wechselseitige Authentifizierung im Zusammenspiel mit einer „Public Key"-Infrastruktur (PKI) durch die Nutzung von Zertifikaten. Hierbei ist der Aspekt der wechselseitigen Authentifizierung hervorzuheben, weil beide Kommunikationspartner entsprechend überprüft werden. Darüber hinaus erlaubt das Verfahren die effiziente Nutzung gängiger symmetrischer Verschlüsselungsverfahren wie RC-4, 3DES oder AES; es ist daher besonders geeignet für die im Internet typische Kommunikation zwischen PC-basiertem Client und Servern auf einem Embedded-System. SSL ist zudem in vielen Endsystemen und Anwendungsprotokollen bereits integriert. Dies gilt insbesondere für den Web-basierten https-Verkehr, der von vielen Web-Clients (Browsern) unterstützt wird.

Es stehen aber auch Implementierungen anderer Anwendungsprotokolle zur Verfügung: z.B. Secure Copy (SCP)

als FTP-Ersatz oder Secure SMTP (SSMTP) für den sicheren E-Mail-Verkehr. Zudem stellt SSL mit der Socket-Schnittstelle eine Programmierschnittstelle (API) zur Verfügung (Bild 3), die auch die einfache Einbindung von eigenen Anwendungsprotokollen erlaubt, was im Embedded-Bereich recht verbreitet ist. Schließlich kann SSL auch für Transportprotokolle eingesetzt werden, die nicht auf TCP/IP basieren, vorausgesetzt, diese stellen einen zuverlässigen Transportmechanismus bereit.

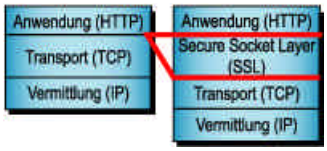

Bild 3. SSL stellt eine eigene sichere Socketschnittstelle oberhalb der Transportschicht zur Verfügung.

Das SSL-Protokoll wurde seit 1994 gemeinsam von Netscape und RSA Security entwickelt. Es soll einen privaten Kanal zwischen Kommunikationsanwendungen bereitstellen, durch den sich Vertraulichkeit und Integrität der Daten gewährleisten lassen und eine Authentifizierung der Partner möglich wird. Hierzu implementiert SSL den asymmetrischen RSA-Algorithmus für das Management des Sitzungsschlüssels und die Authentifizierung von Server und Client sowie einen symmetrischen Algorithmus zur Verschlüsselung der Nutzdaten. Hierbei können sowohl „Streaming"-basierte Verschlüsselungsverfahren wie RC-4 als auch blockbasierte Verschlüsselungsverfahren wie 3DES oder AES verwendet werden.

1996 gründete die IETF die Arbeitsgruppe Transport Layer Security (TLS), um ein an SSL angelehntes Protokoll zu standardisieren. Damit sollten Ansätze von Netscape (SSLv3) und Microsoft (STLP) harmonisiert werden. Nach langen Diskussionen und Wirren konnte dann im Januar 1999 TLS als RFC 2246 verabschiedet werden. TLS und SSLv3 sind in fast allen Bereichen identisch. Viele Werkzeuge unterstützen TLS, darunter auch der Internet Explorer von Microsoft.

Bild 4. SSL ist modular aufgebaut. Auf der übergreifenden „Record"-Schnittstelle bauen die einzelnen Protokollbestandteile auf.

SSL besteht aus einer Reihe von verschiedenen Nachrichtenarten, die mit Hilfe eines einheitlichen „Record Layers" übertragen werden (Bild 4). Dieser modulare Aufbau verbindet die einheitliche Verarbeitung im Protokollstapel mit einer effizienten Systemeinbindung. SSL ist ein verbindungsorientiertes Protokoll. Damit gliedert sich jede Kommunikation in die drei Phasen Verbindungsaufbau, Datenübertragung und Verbindungsabbau. Für den Verbindungsaufbau wird ein mehrstufiger Handshake-Prozess definiert, dessen grundlegender Ablauf im Kasten „Ablauf eines SSL-Handshake" beschrieben ist. Ein Beispiel für eine solche Übertragung, das mit dem Programm „ssldump" [4] aufgezeichnet wurde, ist im Listing eines SSL-Handshake auf S. 61 wiedergegeben. Mit dem Handshake werden der Server authentifiziert (optional auch der Client) und die Schlüssel ausgehandelt, die bei der nachfolgenden verschlüsselten Datenübertragung eingesetzt werden sollen. Eine detaillierte Beschreibung des Protokolls und seiner Implementierung findet sich in [5].

Kryptographie

Zur Absicherung von Integrität und Authentifizierung kommen kryptographische Algorithmen zum Einsatz [6, 7]. Mit ihrer Hilfe lassen sich Informationen so verändern (verschlüsseln), dass sie nur von Befugten verstanden werden. Befugte sind hierbei Personen oder Geräte, die über eine geheime weitere Information verfügen: den Schlüssel. Zudem muss eindeutig nachzuweisen sein, dass eine Veränderung oder Ergänzung nur von einer Person oder einem Gerät durchgeführt wurde, die/das im Besitz einer weiteren geheimen Information ist.

Anforderungen

Die Umsetzung kryptographischer Algorithmen stellt in der Regel hohe Anforderungen an die Rechenleistung der Prozessoren. Dies gilt insbesondere für die asymmetrischen Algorithmen, deren Berechnung auf kleinen Mikrocontrollern durchaus mehrere Sekunden in Anspruch nehmen kann. Zum Glück werden aber diese Operationen nur einmal benötigt: beim Aufbau der Verbindung. Aber auch die symmetrischen Verfahren, die in der zweiten Phase der Datenübertragung zum Einsatz kommen, stellen immer noch hohe Anforderungen. Zwar sind die Grundfunktionen moderner symmetrischer Verfahren relativ einfache Operationen wie 16/32-bit-EXOR- und Schiebe-Operationen oder Tabellenfunktionen, doch werden diese zur Verschlüsselung eines 128 oder 256 bit langen Blocks in unterschiedlicher Abfolge sehr oft durchgeführt. Obwohl neuere Verfahren wie AES mit besonderer Berücksichtigung von Software-Implementierungen entwickelt wurden, erfordern diese eine gewisse Rechenleistung. So kann z.B. ein ARM7TDMI bei 20 MHz mit AES256 pro Sekunde etwa 20 kbyte verschlüsseln [8].

Für die asymmetrischen Verfahren müssen komplexe Operationen wie die modulare Exponentiation mit sehr langen Integer-Zahlen (512 bis 2048 bit) durchgeführt werden. Ohne ein spezielles Rechenwerk müssen diese Berechnungen in Software nachgebildet werden, sie sind daher sehr rechenzeitaufwendig und können auch schnelle Prozessoren über längere Zeit beschäftigen. Ein ARM7TDMI bei 20 MHz benötigt z.B. für eine 1024-bit-RSA-Operation mit dem privaten Schlüssel etwa 2 s [8]. Mit zunehmender Schlüssellänge wachsen diese Zeiten weiter an, bei 2048 bit benötigt die gleiche Operation bereits knapp 14 s [8].

Software oder Hardware

Die Umsetzung der Algorithmen in Software ist insbesondere dann vorteilhaft, wenn nur geringe Datenmengen ver- oder entschlüsselt werden müssen. Hier stehen mittlerweile diverse Bibliotheken zur Verfügung, wie die kommerzielle WAKAN-Cryptolib [9] oder die freie LibTomCrypt [10].

Für den Fall, dass erhebliche Datenmengen verarbeitet werden müssen, kann eine Hardware-basierte Lösung deutliche Vorteile bieten. So werden Mikrocontroller mit Krypto-Coprozessoren ausgestattet, z.B. für Smart Card Controller und für Netzwerk-Anwendungen. Beispiele hierfür sind der „Kirin-IIE" von Freescale [13] oder der „SAMS" von Atmel. Beim Einsatz eines PLD-basierten Prozessors lassen sich die zusätzlichen Rechenoperationen in der programmierbaren Hardware umsetzen. Dabei können unterschiedliche Optimierungsrichtungen beschritten werden: Symmetrische, blockorientierte Verschlüsselungsverfahren wie DES/3DES oder AES lassen sich entweder mit höherem

Flächenaufwand auf minimale Rechenzeit in Pipelines parallelisieren oder mit geringerem Flächenaufwand in mehreren Zyklen betreiben [11]. Nachteilig ist, dass die Hardware-Einbindung von den für Embedded-Systeme geeigneten Bibliotheken nur selten direkt unterstützt wird. Dies bleibt den Integratoren der Protokoll-Software überlassen.

Implementierungen des SSL-Protokolls

Die Umsetzung der kryptographischen Algorithmen ist die eine Seite der Medaille. In einem zweiten Schritt müssen diese in den Protokollablauf eingebunden werden. Auch hierfür gibt es verschiedene Möglichkeiten.

Offene Stacks

SSL liegt in einer Reihe offener Implementierungen vor. Hierzu zählen die Bibliotheken von OpenSSL [12], die alle notwendigen Routinen zur Erstellung eigener Programme und Werkzeuge zur Generierung asymmetrischer Schlüssel und X.509-Zertifikate enthalten. OpenSSL hat für den PC-Bereich im Windows- und Linux-Umfeld große Verbreitung gefunden. Für diesen Beitrag ist auch der Einsatz in Embedded-Linux-Systemen interessant. Eine direkte Übernahme in Embedded-Systeme ist allerdings problematisch. Zu beachten ist die dynamische Speicherverwaltung, die nicht nur bei kleinen Systemen zu einer Segmentierung des Speichers führen kann. Dies kann bei Systemen mit kleinem Speicher und langen Laufzeiten ohne „Reboot“ zu Stabilitätsproblemen führen, die auch durch die Verwendung einer Memory Management Unit (MMU) nicht zu beheben sind.

Embedded Stacks

Die ausschließlich auf Mikrocontrollern basierten Implementierungen erfordern daher spezifische Umsetzungen. Kostengünstige Embedded-Sys-teme verfügen nicht über ausreichende Ressourcen, um darauf ein „Embedded Linux“ zu betreiben. Auch die meisten anderen Portierungen für Embedded-Systeme basieren auf der OpenSSL-Implementierung. Sie beziehen häufig alle Funktionen mit ein, ohne Rücksicht darauf, ob diese auf einem Embedded-System sinnvoll oder notwendig sind. Zusätzlich zu der Qualität der Protokoll-Implementierung ist die Sicherheit einer SSL-verschlüsselten Kommunikation abhängig von zwei weiteren Aspekten, die weitgehend in der Verantwortung des Programmierers und des Anwenders liegen:

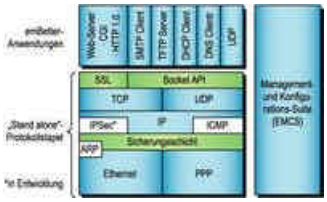

Bild 5. Aufbau des „Embedded TCP/IP“-Protokollstapels „emBetter“.

- Erstens hängt die Sicherheit der SSL-Verbindung entscheidend von der Qualität des Zufallszahlen-Generators auf dem Embedded-System ab. Allerdings ist die Erzeugung guter Zufallszahlen auf einem Embedded-System dann schwierig, wenn nicht spezielle Hardware-Zufallszahlengeneratoren verwendet werden. Die in einigen Prozessoren integrierten Zufallszahlen-Generatoren auf Schieberegisterbasis sind für die Erzeugung kryptographisch sicherer Zufallszahlen ungeeignet. In Frage kommen daher nur Software-Zufallszahlen-Generatoren; diese müssen mit einer ausreichenden Menge zufälliger Ereignisse versorgt werden, die von außen nicht beobachtbar oder vorhersagbar sind.
- Zweitens ist die Geheimhaltung des privaten Schlüssel des Servers von entscheidender Bedeutung. Dieser muss zwangsläufig in irgendeiner Form auf dem Server gespeichert werden. Wenn dieser Schlüssel einem Angreifer in die Hände fällt, kann dieser die gesamte auf diesem Schlüssel basierende Kommunikation entschlüsseln. Daher ist es zwingend erforderlich, dass jeder Server mit einem individuellen Schlüssel ausgestattet wird, damit der Schaden im Falle eines Falles möglichst auf ein System begrenzt bleibt.

„emBetter“-SSL

Am STZEDN (siehe Kasten) wird seit Jahren der TCP/IP-Protokollstapel „emBetter“ entwickelt. Hierbei handelt es sich um einen extrem skalierbaren TCP/IP-Stack für kleine 16- und 32-bit-Mikrocontroller, der entweder für geringen Speicherbedarf oder für hohe Systemleistung konfiguriert werden kann (Bild 5). Für diesen Stack steht nun neben anderen Erweiterungen auch ein SSL-Modul zur Verfügung, das die sichere Kommunikation bei moderatem Ressourcenverbrauch und guter Systemleistung erlaubt.

Steinbeis Transferzentrum STZEDN

Das Steinbeis-Transferzentrum für Embedded Design und Networking (STZEDN – www.stzedn.de) ist an die Berufsakademie Lörrach angegliedert. Es beschäftigt sich mit allen Aspekten des Entwurfs von Embedded-Systemen – und insbesondere mit deren drahtloser und drahtgebundener Vernetzung – sowie den Aspekten des Hardware-Software-Co-Designs. Das Team fest angestellter Ingenieure rund um Prof. Dr. Sikora bietet Beratungs- und Entwicklungsdienstleistungen sowie Soft- und Hardware-Produkte an.

Ablauf eines SSL-Handshake

1. Handshake HelloRequest: Der Aufbau einer SSL-Verbindung wird stets vom Client initiiert. Mit dem ServerHello Request kann jedoch ein Server einen Client veranlassen, einen SSL-Handshake zu initiieren, z.B. für einen Rehandshake.

2. GET HTTPS://: Der eigentliche Ablauf auf der Client-Seite beginnt mit dem Aufruf eines geschützten Dokuments auf dem Server, das durch eine URL mit dem Protokollbezeichner HTTPS:// gekennzeichnet ist. Üblicherweise erkennt die Client-Applikation anhand der URL, dass es sich um eine geschützte URL handelt, und startet den SSL-Handshake durch Senden eines Handshake ClientHello an den sicheren Port (z.B. Port 443 für HTTPS) des Servers (ab Punkt 3. siehe Listing).

3. Handshake ClientHello: Der Client sendet dem Server eine Liste von Algorithmen, die von ihm unterstützt werden, sowie eine Zufallszahl, die bei der Erzeugung des Schlüssels verwendet wird. SSL und TLS unterstützen dabei eine Vielzahl von unterschiedlichen Verschlüsselungsverfahren verschiedener Qualität und Komplexität.

4. Handshake ServerHello: Der Server sucht einen Algorithmus aus den vom Client angebotenen Verfahren aus und informiert den Client hierüber. 5. Handshake Certificate: Der Server sendet seinen öffentlichen RSA-Schlüssel in einem gültigen X.509-Zertifikat an den Client. Falls eine Client-Authentifizierung per Zertifikat gefordert ist, sendet der Server zusätzlich eine Liste der akzeptierten Zertifizierungsstellen als Zertifikat-anforderung.

6. Handshake ServerHelloDone: Zusätzlich versendet auch der Server eine Zufallszahl, die ebenfalls bei der Erzeugung des Schlüssels benötigt wird.

7. Handshake ClientKeyExchange: Der Client überprüft das Zertifikat des Servers. Dann erzeugt er eine zufällige Zeichenfolge (pre master secret) und sendet diese mit dem öffentlichen Schlüssel des Servers verschlüsselt an den Server. Auf der Grundlage der beiden von Server und Client erzeugten Zufallszahlen und des pre master secret berechnen Client und Server unabhängig voneinander die Schlüssel für die Verschlüsselung und die Signaturen (MAC). Der Server kann dies nur korrekt durchführen, wenn er im Besitz des zum Server-Zertifikat passenden privaten Schlüssels ist. Im Falle der Client-Authentifizierung sendet der Client zusätzlich ein passendes Client-Zertifikat sowie eine mit dem privaten Schlüssel signierte Prüfsequenz, die Client und Server im bisherigen Protokollablauf gemeinsam berechnet haben. Der Server überprüft das Client-Zertifikat und überprüft die Prüfsequenz mit Hilfe des öffentlichen Schlüssels aus dem Client-Zertifikat.

8. ChangeCipherSpec: Der Client informiert danach den Server, dass alle weiteren Pakete verschlüsselt übertragen werden.

9. Handshake Finished: Das Finished-Paket enthält ein MAC über alle übertragenen Handshake-Nachrichten. Auf diese Weise kann der Server herausfinden, ob ein „Man in the Middle"-Angriff die originalen Pakete des Clients verändert hat.

10. ChangeCipherSpec: Der Server informiert dann den Client, dass alle weiteren Pakete verschlüsselt übertragen werden.

11. Handshake Finished: Das Finished-Paket des Servers enthält ebenfalls ein MAC über alle übertragenen Handshake-Nachrichten. Auf diese Weise kann auch der Client herausfinden, ob ein „Man in the Middle"-Angriff die originalen Pakete des Servers verändert hat.

Literatur

[1] [1] -Braun, N.; Sikora, A.: Vergleich verschiedener Mikrocontroller-Plattformen für den Einsatz in Kommunikationsanwendungen. Teil 1: Elektronik 2005, H. 20, S. 54ff.; Teil 2: Elektronik 2005, H. 21, S. 49ff

[2] Straub, T.; Sikora, A.: Cryptography on Embedded Systems. Embedded World Conference 2005, Nürnberg, S. 517 – 536.

[3] Erschig, M.; Sikora, A.: Administration and Security Aspects for Embedded Webserver Design. Embedded World Conference 2006, Nürnberg, 14. bis 16. 2. 2006

[4] http://sourceforge.net/projects/ssldump

[5] Rescorla, E.: SSL and TLS – Designing and Building Secure Systems. Addison-Wesley, 2001, ISBN 0-2016-1598-3

[6] Ertel, W.: Angewandte Kryptographie. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2. Auflage 2003

[7] Menezes, A.; van Oorschot, P.; Van-stone, S.: Handbook of Applied Cryptography. CRC Press, 1997

[8] Wakan Crypto-Software-Toolkit Release 2.1 Technical Information. Ascom AG, 2002

[9] Steinbeis-Transferzentrum für Embedded Design und Networking – www.stzedn.de

[10] LibTomCrypt http://libtomcrypt.bytemine.net/libtomcrypt.org_80/

[11] Sikora, A.: Hard- und Softwarelösungen für sichere Embedded Systeme. Design-&-Elektronik-Entwicklerforum „Drahtlose & drahtgebundene Netzwerke für Industrie & Automotive". München, 7. Juli 2004

[12] OpenSSL Project – www.openssl.org

[13] http://www.freescale.com/files/ftf_2005/doc/reports_presentations/MZ123_B.pdf?srch=1

Autoren

Dipl.-Ing. (FH) Nathan Braun hat nach einer Ausbildung zum Elektriker das Studium der Trinationalen Ingenieurausbildung in der Fachrichtung Mechatronik an der Berufsakademie Lörrach, der Fachhochschule beider Basel (FHBB) und der Université de Haute Alsace in Mulhouse absolviert. In seiner Diplomarbeit legte er die Grundlagen für den TCP/IP-Stack des „emBetter". Seit Anfang 2004 arbeitet er im Steinbeis-Transferzentrum für Embedded Design und Networking, wo er sich im Wesentlichen mit Embedded Kommunika-tionslösungen rund um den „emBetter" befasst.
braun(at)stzedn.de

Dr.-Ing. Thomas Gillen befasst sich seit mehreren Jahren mit der Hard- und Softwareentwicklung für industrielle Applikationen sowie mit der Systemsicherheit und der Implementation von kryptographischen Verfahren und sicheren Übertragungsprotokollen für Embedded-Systeme. Seit 2004 ist er bei Art of Technology unter anderem für die sichere Kommunikation und die sichere Authentifizierung bei kleinen Embedded-Systemen zuständig.
thomas.gillen(at)art-of-technology.com

Prof. Dr.-Ing. Axel Sikora leitet den Studiengang Informationstechnik, das Steinbeis-Transferzentrum für Embedded Design und Networking und das Steinbeis-Forschungsinstitut für Drahtlose Kommunikation an der Berufsakademie Lörrach. Schwerpunkte seiner Arbeit sind Hardware-Software-Co-Design für vernetzte Systeme sowie die Protokollentwicklung für Embedded-Netzwerke. Er ist Autor, Co-Autor und Herausgeber mehrerer Fachbücher, sowie zahlreicher Konferenzbeiträge und Fachartikel zu diesem Themenkreis.
sikora(at)ba-loerrach

Jens Würtenberg, Elektronik