Advanced Internet Security
Zusammenfassung SS06

Nicolas Lanquetin (nl013)

LastChangedDate: 2006-11-21 17:16:33 +0000 (Tue, 21 Nov 2006)
LastChangedRevision: 14

Inhaltsverzeichnis

1 Security as a System

- **Prevention** → **Detection** → **Response**
- Security as a Problem
  - **Usability** → meistens: hohe Security = niedrige Usability
  - **Kosten** → Höchste Kosten sind: education, help desk, processes etc. Niedrige Kosten sind eigentlich die Hardware
  - **Komplexität** → Zusammenspiel aus Technik und Sozialem
  - **Wartbarkeit** → updates, patches etc
  - **Privacy** → Loggen von privaten Daten
- Security Pinzipien
  - **Default is deny**
  - **Defense in depth** (lines of defense)
  - **Least Privilege** (Need-to-know, Need-to-do)
  - **Protection, Detection, Response** (Security Probleme erwarten und Pläne parat haben)
  - **Fail save stance** (Bei Einbruch, nichts mehr im System möglich)
  - **No Security by Obscurity**
  - **Simplicity**
- Step-up authentication (TODO)
- Wichtig: nicht nur nach aussen sicher, sondern **innen** auch sicher! (Beispiel: Wireless Kommunikation zw. Aussen und Innen)
- **Security Analyses**
  - neue Service = neue Gefahren
  - Wie reagieren auf Schwachstellen
  - Notfall-Antwort auf Security-Ereignisse
  - Standard-Software untersuchen auf Security-Fragen
  - Risiko-Analysen
- Security-Probleme = Chance für Business (Banken z.B.)
- **Data-Classification Schemes**
  - **Confidentiality**: public, private, restricted
  - **Storage**: daily backup, daily backup with weekly transfer to long-term storage, additional non-it-storage

- Problem mit **root**: root kann jeder sein (keine user ID). root hat alle Rechte (Verstoss gegen Need-to-know, Need-to-do)
- **Pyramide**: **Usabilty, Security, Performance**. Meistens sind immer nur 2 der Punkte erfüllt. (z.B. Applet: usability und security, aber nicht performance)

2 A framework for Security

http://wssg.berkeley.edu/SecurityInfrastructure/reports/framework.html
http://www.apa.state.va.us/info_sys_security/security_framework.htm

- **Keine Klare Trennung**: Internet - Intranet
- Jede Security Analyse braucht ein Security **Framework** mit **Regeln und Policies**
- **Security Framework**: Risk Management Prinzipien, Verantwortungen, Technische Implementationen, etc.
- **Security Policy**: Framework für das Treffen von Entscheidungen, wie "defense mechanisms", "how to configure services", "secure programming guidelines", "procedures for users and admins"
  - mandatory **asset ownership** (Alle Daten haben einen Eigentümer)
  - mandatory **data classification**
  - **Confidentiality**, **Integrity**, **Availability**, **Non-Repudiation** (S. 11)
  - **Need-to-know, Need-to-do** (Separation of power)
  - **Risk Management** / **Analysen**
  - **Monitoring**
  - **Incident handling** (vorbereitet sein)
  - **Security Organisation** (**wer** ist verantwortlich für **was**)
  - **Hardware / Software** Security (Trusted Computing Base, Access Control, Encryption Management, Password Handling, SSO, Virus Handling, Logging, Backup)
  - **Network / Internet Services** Security (VPN, private VLAN, ..., aber nicht Security an der Steckdose)
  - **Sign-Off process** (Garantiert dass Software im Einklang mit der Policy steht. Schon vor der Implementierung überprüft.)
  - **Education**
  - **Disaster** Handling
- **Access Control**: **Hierarchical** vs **Role Based** (Role Based = sehr viel besser, weil Need-to-Know/Do)
- **Trusted Computing Base**: Bearbeitet und speichert classified data (logging, auditing, monitoring, hardware based authentication, special OS, etc)

- Certificate Authority (TODO)
- **Trennung** von **Daten und Verarbeitung**: Customer data zone — Processing Zone — Untrusted clients
- **Keinen direkten Zugriff auf DB** (alles muss durch ein application interface)
- **SSO** (Einmalige Authentisierung gegenüber SSO Layer) (S. 30)
- **DMZ** (Demilitarized Zones) (Kein System is direkt am Internet angebunden. Bereiche sind durch Packet Filter getrennt) (S. 31)
- Standard Software sollte **keine eigene Authentisierung/Authorisierung** implementieren, sondern auf externe APIs/Interfaces zurückgreifen.
- Software Design: **Integration** in bestehende Infrastruktur **durch Mapping**: Authent-, Access-, Log-Mechanismen usw. (Vorteil: Austauschbare Komponenten) (S. 35)

3 Firewalls: Types and Architectures

http://de.wikipedia.org/wiki/Firewall
http://wiki.hackerboard.de/index.php/Firewall
http://de.wikipedia.org/wiki/Proxy

- Arbeitet nach einer bestimmten **Policy** (Regeln)
- Möglichkeiten:
  - **block**
  - **weiterleiten** zu anderen Systemen (transparent proxy)
  - internes **Netz verstecken**
  - Traffic **überprüfen/loggen**
  - **Authentisierung** erzwingen
- Probleme:
  - Komplizierte **Konfiguration**
  - Filtern kostet **Performance**
  - Insider können leicht **piercen**
  - kein Schutz gegen **DOS**
- **IP** = unsicher: Schutz gegen **Sniffing/Spoofing**: **Switches** einbauen (S. 11-13)
- **TCP** = unsicher: Auf **well-known Ports** können ganz andere Service laufen, **TCP SYN-Flooding**, **TCP Probing** (was ist auf welchem Port), TCP **Sequence Number Guessing** (schlechte Algorithmen bei einigen OS') (S. 14-18)
- **ICMP** = unsicher: mit ping netzwerk topology herausfinden, DOS mit echo-reply requests, etc

- Probleme gibt es wenn: **verbindungslos** (UDP), **multi-port** (p2p), **keine standard ports**, **keine klare Client/Server-Trennung**
- Network Firewalls: (Unbedingt **nachlesen** auf S. 40-43) (S. 40-43)
  - **Static packet filter** (stateless, billig u. schnell)
  - **Dynamic packet filter** (stateful, mächtig)
  - **Application Gateway** (Proxies)
  - **Screened Host** (S. 40)
  - **Screened Subnet** (S. 41)
  - **DMZ** (S. 42)
  - **Multiple DMZ**. (S. 43)
  - **Transparent Firewalls**: Keine IP-Adresse, trennt jedoch 2 Netze wie eine bridge
- Application Gateways:
  - PRO: **logging**, **authentication/authorization**, **true network separation**, **caching**, **intelligent filtering**
  - CONTRA: **performance** (bottleneck), **special software** proxies, **Target for attackes**
- **Encrypted End-to-End** (zb VPN) = gefährlich, setzt firewall ausser gefecht (piercing = kinderspiel)
- **Personal Firewall**: Problem: beschützt das OS, braucht es aber gleichzeitig. User muss wissen was er tut.
- **Host based Firewalls**: eine Personal FW auf jedem Host drauf
- **Network Edge Firewalls**: In jeder Nic der Hosts eingebaut, also **OS-unabhängig** (besser als Host Based FW)
- **Limits** of Firewalls: mit Buffer-overflow wird ein Proxy ("**Zecke**") mitübertragen und bei der Applikation ausgeführt.

4 Packet Filtering

http://de.wikipedia.org/wiki/Paketfilter
http://www.tldp.org/HOWTO/IPCHAINS-HOWTO.html (Kapitel 7 lesen!)

- Auswerten der Headers:
  - **IP**: **Protocol**, **Source/Destination Address**
  - **TCP**: **Source/Destination Port**, **Flags** (SYN, ACK)
  - **UDP**: **keine Connection information**
  - **ICMP**: **Grösse überprüfen**!

- Standard Regeln **unbedingt lesen** (S. 12) (S. 12)
- Bei **ipchains** ist es am besten für jede mögliche Verbindung eine user chain zu machen (siehe http://www.tldp.org/HOWTO/IPCHAINS-HOWTO-7.html#ss7.4)
- Im **unix kernel** sind die **netfilter/iptables module** installiert (schnell, muss aber stabil sein). Im **user level** werden diese nur **angewandt**

5 Stateful Filtering (iptables)

http://de.wikipedia.org/wiki/Iptables
http://www.pl-forum.de/t_netzwerk/iptables.html

- **iptables** vs **ipchains**: iptables ist **stateful**, ipchains ist **stateless** (S. 6)
- iptables: 3 **tables** (nat, mangle, filter), 5 built-in **chains** (Siehe S. 8), etliche **targets** (Siehe S. 18) (S. 8f, 18)
- `RELATED` verstehen! Siehe FTP Beispiel der Übungsklausur.
- **SNAT** (pre-routing), **DNAT** (post-routing)

6 Intermediaries and Services

- **Intermediaries**: Komponenten die zwischengeschalten werden und sowohl als **Client** als auch **Server** fungieren können (z.B. Proxy)
- `connect`-Befehl: Wird gebraucht bei Verschlüsselung. Darüber weiss der Proxy zu welchem Host er verbinden soll. (Hier kann der Proxy noch filtern) (S. 13, 26)
- HTTP Service with **transparent Proxy** (S. 16) und **visible Proxy** (S. 17) (S. 17f)
- **RPC based Services** sollten gesperrt werden: ports ändern sich zu oft (S. 23)
- **Distributed Objects** sollten gesperrt werden (Ausnahme: Corba über SSL) (S. 24)
- **Netbios over TCP/IP, SMB** sollten gesperrt werden: werden nur netzintern gebraucht (S. 25)
- **Security Analyse**: Welche **Sicherheits-Fragen** müssen gestellt werden (als wiederholung auf S. 39) (S. 39)
- **Evaluation Framework for Services**:
  - authentication, non-repudiation
  - confidentialitiy, integrity
  - proxying capabilities
  - NAT capabilities
  - protocol semantics (separation of concerns)

7 Buffer Overflow Attacks - How they work

- Kein echter Schutz möglich
  - Aber Verminderung durch **software upgrades** (subscription to **CERT**)
  - Anwenden des **defense-in-depth**-Prinzip
- Buffer Overflow sind tödlich, weil Programme mit allen Rechten laufen (Verstoss gegen POLA)
- Beispiel: Ein einfacher Stack Buffer Overflow
  - Problem: Variablen und Rücksprungadressen werden beide auf dem Stack gespeichert
  - Angriffsziel: Input-Variablen, deren Länge nicht geprüft wird
  - Überschreibe Variable mit bösartiger Funktion
  - Überschreibe weiter mit Rücksprungadresse zu dieser Funktion
  - Irgendwann findet Programm diese Rücksprungadresse und springt zur bösartigen Funktion

8 Attacking Web Applications

- **SQL-Injection**: Wenn Request-Parameter einfach in die SQL-Syntax mit übernommen werden, kann der Attacker sql-statements mit einschleusen.
- **Directory Traversal Attack**: zB bei IIS. Dieser Server validiert den Input in der **falschen Reihenfolge**. Man muss **erst normaliesieren**, **dann filtern**. (S. 9)
- **Session Takeover**: zB über Session Token (Session ID).
  Möglichkeiten für Session: 1) **Cookies**, 2) **URL-Rewriting**, 3) **SSL Session ID**
  Zu 3): ist am besten, weil es nicht gestohlen werden kann. Zudem kann beim Proxy zwischen Session ID und SSL ID **gemappt** werden; damit ändert sich für die Applikation nichts. (S. 12)
- **Cross Site Scripting** (**Script Injection**): Ziel: Skript einfügen über HTML-Forms, die dann auf Seiten durch andere Benutzer aufgeruft werden.
- **Email based semantic attacks**: fishing usw. Lösung: Client-Side Transactionsignierung: Keine Änderung möglich, Empfänger kennt Sender

9 XML Security - Secure XML documents and communication

- **malicious documents**: Entities mit externen Referenzen werden u.U. vom Parser automatisch referenziert, was für DOS-Angriffe ausgenutz werden kann (tausende Referenzen in einem Dokument)
- **Validierung unterdrücken**: Auf eigenes Schema verweisen, welches Validierung für alle anderen Namespaces unterdrückt
- **XML Security bisher**: lediglich **Transport** via SSL verschlüsseln/signieren
- **Problem bei XML**: Logische und physische Struktur von XML ist nicht das gleiche

- **Lösung - Canonical XML**: Nun können Signaturen auf die kanonisierten Dokumente angewendet werden (logische und physische Repräsentation ist nun gleich)
- **XML DSIG**: Zu signierende XML-Node wird über deren URI referenziert oder umhüllt; DigestMethod, SignatureValue, KeyInfo zum Verifizieren der Signatur werden mit angegeben.
- **Encryption**: Verschiedene Teile können unterschiedlich verschlüsselt werden (multi-party communication system encryption) - sehr kompliziert zu realisieren

10 XML and Web Services Security

- **Szenario 1 - Direct Trust**: UserId/Passwort, dann SSL-Session. Lediglich Transport ist gesichert
- **Szenario 2 - Issued Security Token**: Trusted third party, Service muss lediglich third party trauen, Authentifizierungsmechanismus wird von third party geleistet
- **Szenario 3 - Federation**: Ein Identity-Service A mappt Anfragen eines weiteren Identity-Services B. So kann ein Web-Service S, der nur A vertraut auch Anfragen eines Clients von B beantworten. (wahrscheinlich über Zertifikats-Tausch)
- **Szenario 4 - Crossing Security Domain**: Mix aus PKI Infrastructure auf der einen und Kerberos auf der anderen Seite. Trotzdem sollen Services überall funktionieren. (mutual trust)
- **Szenario 5 - Token based delegated authorization**: roadmap for ws-security. Tokens verfallen nach gewisser Zeit. Client muss von Zeit zu Zeit neue Tokens anfordern.
- **Szenarion 6 - Endpoint based authorization**: Client braucht nun nicht mehr ständig neue Tokens.

11 Web Application Security

http://wwwcsif.cs.ucdavis.edu/ wassermg/research/webapps.pdf

TODO