Sicherheit von Anwendungssoftware

Hans Weibel
Prof., dipl. El.-Ing. ETH
Leiter Fachgruppe Kommunikation
Zürcher Hochschule Winterthur
hans.weibel@zhwin.ch

Einordnung

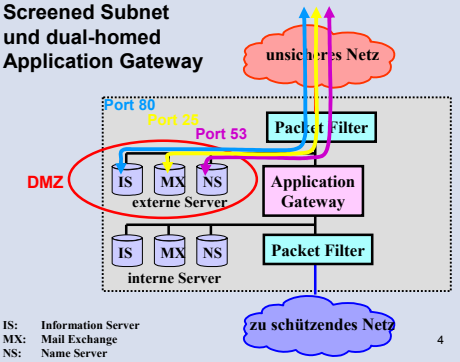
- Ziele aller Sicherheitsmassnahmen sind:
  - Schutz der Daten (Speicherung, Übertragung)
    - Vertraulichkeit
    - Integrität
    - Verfügbarkeit
  - Nachvollziehbarkeit von Transaktionen
  - Authentizität des Kommunikationspartners sicherstellen
- Sicherheit resultiert aus dem Zusammenspiel verschiedener Faktoren:
  - Physische Sicherheit
  - Betriebssystemsicherheit
  - Netzwerksicherheit
  - Anwendungssicherheit

Massnahmen und Wirkung

- Firewall (Paketfilter, Proxy)
- Intrusion Detection / Response System
- Virenscanner
- Kryptographische Verfahren
  - Authentication
  - Encryption
- Applikation
  - Web-Applikationen und Mail-Systeme bringen die klassische IT-Sicherheit an ihre Grenzen
    - Angriffe erfolgen über HTTP, HTTPS, SMTP, DNS
    - HTTP als zustandsloses Protokoll schafft besondere Probleme
  - Firewalls und IDS können viele Angriff auf Applikationsebene weder erkennen noch verhindern

Die perfekte Firewall

Screened Subnet und dual-homed Application Gateway



IS: Information Server
MX: Mail Exchange
NS: Name Server

Schwachstellen ...

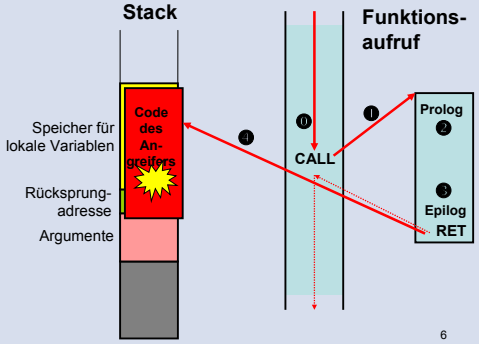
... am Beispiel Web Applikation

Die Angriffsfläche besteht aus Design-, Programmier- und Konfigurationsfehlern

- Buffer Overflow
- Unzureichend validierte Parameter
- Durchbrochene Zugangskontrolle
- Durchbrochenes Session Management
- Cross-Site Scripting
- Command Injection (z.B. SQL)
- Probleme mit Fehlerbehandlung
- Unsichere Implementierung von Kryptographie
- Unsichere Remote Administration
- Fehlkonfiguration von Server bzw. Applikation

Quelle: The Open Web Applications Security Project, www.owasp.org

Buffer Overflow



Buffer Overflow

- Mögliche Folgen?
  - Angreifer kann seinen eingeschleusten Code ausführen, mit den Privilegien der betroffenen Anwendung.
- Gegenmassnahmen?
  - In erster Linie: Sorgfältige Programmierung mit konsequenter Validierung von Parametern.
  - Durch entsprechende Codegenerierung verhindern, dass die Rücksprungadresse überschrieben werden kann:
    - StackShield
    - StackGuard: Modifikation von gcc (Prolog legt Canary auf Stack, Epilog prüft Canary)
    - `/GS` Option von Microsoft's C/C++ Compiler
    - Diese Gegenmassnahmen erschweren ein Exploit. Es gibt jedoch Umgehungsmöglichkeiten.
  - Durch Hardware verhindern, dass Code im Stack ausgeführt wird:
    - Speichersegmentierung des Pentium ermöglicht dies, aber weder Windows noch Linux nutzen diese Funktion.

1. Beispiel: Code Red

- Am 18. Juni 2001 warnt Microsoft vor einer Schwachstelle in IIS's Index Server.
  - Es handelt sich um eine „Buffer Overflow Vulnerability".
  - Gleichzeitig wird ein Patch bereitgestellt.
- Mögliche Auswirkungen sind:
  - Änderung von Web-Inhalten
  - Ausführung von System-Kommandos
  - Server-Rekonfiguration
  - Laden und Ausführen beliebiger Software auf dem Server
  - etc.
- Am 19. Juli 2001 meldet CERT die Verbreitung des Wurmes ***Code Red***, der genau diese Schwachstelle ausnutzt.

Code Red – Technische Details

- Der Angreifer etabliert, wie normale Benutzer auch, eine Verbindung auf Port 80 des Opfers.
- Er sendet einen „gebastelten" GET-Request, der einen Buffer Overflow im Indexing Service bewirkt.
- Ist das Opfer ein IIS mit der entsprechenden Schwachstelle, so wird es übernommen.
  - Danach greift es Rechner mit zufällig ausgewählter IP-Adresse nach dem selben Muster an.
  - Nebst IIS werden auch Cisco's DSL-Router der Serie 600 befallen, die in der Folge keine Pakete mehr weiterleiten (sich aber nicht an der Weiterverbreitung beteiligen).

Code Red - Lehren

- Bulletins werden unterschiedlich beachtet:
  - Es gibt System-Administratoren, die sie nicht lesen bzw. nicht befolgen.
  - Es gibt Cracker, die sich inspirieren lassen.
  - Mittlerweile haben schon sogen. Zero Day Attacks stattgefunden (d.h. Exploit an dem Tag, an dem die Schwachstelle bekannt gegeben wurde).
- Firewalls helfen hier gar nichts.
- Auch andere Systeme können betroffen werden (z.B. Geräte mit Web-Interface wie Router): -> Auswirkung auf Verfügbarkeit von Netzelementen und Services.
- Der Wurm ist speicherresident: im File System ist nichts feststellbar (Integrität ist nicht verletzt).

2. Beispiel: SQL Slammer

- Im Juli 2002 informiert Microsoft über eine Schwachstelle im SQL Server.
  - Es handelt sich um eine „Buffer Overflow Vulnerability".
  - Gleichzeitig wird ein Patch bereitgestellt.
- Mögliche Auswirkungen sind:
  - Zugriff auf Inhalte des Severs
  - Ausführung von System-Kommandos
  - Server-Rekonfiguration
  - Laden und Ausführen beliebiger Software auf dem Server
- Am 25.1.2003 schlägt der ***SQL Slammer*** zu:
  - Viele Netze brechen völlig zusammen.
  - Die Säuberung dauert Tage.

SQL Slammer – Techn. Details

- Der Angreifer sendet ein „gebasteltes" UDP-Paket an das Monitor Port 1434 des SQL Servers
- Ist das Opfer ein Server mit der entsprechenden Schwachstelle, so führt es den im Paket mitgeführten Code aus.
  - Dieser sendet gleichartige Pakete an zufällig gewählte IP-Adressen (dies geht sogar durch non-stateful Firewalls hinaus, weil das Paket als DNS-Response mit Quellport 53 getarnt ist).
  - Netzkomponenten wie Switches werden überlastet (Zieladresse nicht im Cache, Zieladresse ist Multicast, TTL ist 1 -> ICMP).
  - Netzwerke sind längere Zeit in einem unbenutzbaren Zustand.

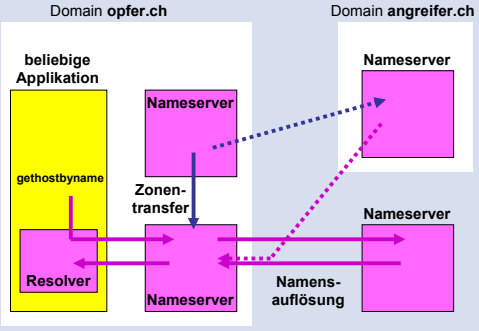
SQL Slammer - Lehren

- Bulletins wurden auch in diesem Fall nicht gebührend beachtet.
- Eine stateful Firewall hätte die weitere Verbreitung nach aussen gestoppt.
- Das Netz wird so stark überlastet, dass ein Inband-Management nicht mehr funktioniert.
- Einmal im Intranet, befällt der Wurm sehr schnell die weniger geschützten Server (z.B. Testumgebung).
- Wie auch bei Code Red: Es gibt keine „uninteressanten“ Opfer!

3. Beispiel: DNS

- Das ganze Internet hängt von DNS ab.
- Viele Nameserver beruhen auf BIND „named“:
  - BIND ist über lange Zeit gewachsen
  - Viele Versionen sind in Gebrauch.
  - Viele Schwachstellen sind bekannt.
  - BIND 8 ist die verbreitete Version.
  - BIND 9 ist von Grund auf neu konzipiert und ist bezüglich DNS Sicherheit zu bevorzugen.
- Schwachstellen
  - Nameserver antwortet mit falscher Information: die erste Antwort gewinnt (Cache Poisoning)
  - Einbruch auf Nameserver (unerlaubter Zonentransfer)
  - DNS Resolver mit Buffer Overflow Vulnerability

DNS – Schwachstellen



DNS – Nameserver BIND

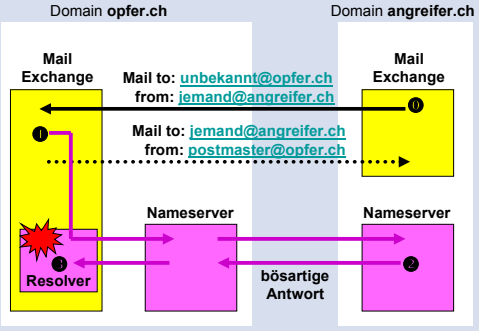
Domains .ch/.li

15'664 Nameserver, 683'441 Domains, davon 8,779 sicher mit BIND (nicht alle Server geben NS-Software preis, doch ca. 80% aller Nameserver sind BIND).

Quelle:
Swiss Internet Analysis, Diplomarbeit ZHW, November 2002

Rang	Version	Anzahl	Anteil
1	BIND 8.2.3-REL	2301	26.2%
2	BIND 9.2.1	1618	18.4%
3	BIND 8.2.4-REL	554	6.3%
4	BIND 9.1.3	546	6.2%
5	BIND 8.3.3-REL	541	6.2%
6	BIND 9.2.0	301	3.4%
7	BIND 8.3.3-REL-NOESW	257	2.9%
8	BIND 8.3.1-REL	249	2.8%
9	BIND 9.1.0	227	2.6%
10	BIND 8.2.2-P5	192	2.2%
11	BIND 8.2.3-REL-NOESW	175	2.0%
12	BIND 9.1.2	167	1.9%
13	BIND 8.1.2	153	1.7%
14	BIND 8.2.5-REL	109	1.2%

DNS – Boshafter Nameserver



DNS – Schwachstelle Resolver

- DNS Resolver mit Buffer Overflow Vulnerability
  - Applikation kann dazu gebracht werden, einen „bösartigen“ Name Server zu kontaktieren
  - Angreifer sendet „bösartige“ DNS Response
- Resolver ist eine Bibliotheksfunktion
  - Statische Library: Alle Applikationen müssen mit der verbesserten Library neu gelinkt werden
  - Shared Library: durch verbesserte Version ersetzen
- Tausende von Applikationen betroffen
  - Solche Applikationen laufen oft als root (Bsp. Mail)

Wer weiss, auf welchem Stand seine Applikationen sind?

Was tun?

- Sichtweise
  - Sicherheit als ist mehr als ein Kostenfaktor und eine Versicherung(sprämie).
  - Sicherheit hat auch einen Nutzen.
  - Sicherheit ist eine Daueraufgabe für Profis (u.U. Externe beiziehen -> Vertrauenssache)
- Security ist relativ
  - Risiken abschätzen
  - Massvollen und wohlabgestimmten Einsatz der Mittel
  - Was man tut, richtig tun (keine Alibiübung)
- S3D: Security by ...
  - ... Design
  - ... Defaults
  - ... Deployment

Was tun? ➔ Entwicklung

- Entwicklungsprozess
- „penetrate and patch“ versus „secure Software Engineering Practices“
- Security als Designaspekt: funktionale und sicherheitsbezogene Anforderungen
- Architektur sicherer Anwendungen, Patterns
  - Server-Sicherheit: Serverseitige Programmierung, Server Hardening, Session Management, DB-Ankopplung
  - Client-Sicherheit: aktive Inhalte (auf Browser ausgeführt), Applets, Scripts, ActveX
  - Sicherheit der Backend-Systeme
- Usability, Dokumentation, „sichere“ Defaults

Lit: Michael Howard, David LeBlanc: „**Writing Secure Code**“, Microsoft Press

Was tun? ➔ Konfiguration

- Ziel ist die Minimierung der Angriffsfläche!
- Sorgfältige und defensive Konfiguration der SW
  - So wenig wie nötig, so einfach wie möglich.
  - Rechtevergabe: wenn möglich nicht als root (chroot mit User Privileg)
- Konfiguration des Netzwerkes:
  - Web-Server in der DMZ
  - Nameserver in verschiedenen Subnetzen, evtl. verschiedene Plattformen verwenden
  - Split Horizon DNS (public und private Nameserver)
  - Keine anderen Dienste auf dem Web- bzw. Nameserver
  - Periodische (tägliche) Integritätsprüfung des File Systems
  - Produktive Umgebung getrennt von Testumgebung/Spielwiesen (interne Firewalls)

Was tun? ➔ Betrieb

- Prinzip: Jeder versteht seinen Job, weiss, was er tut und verfügt über die notwendigen Ressourcen
  - Technik verwenden, die man beherrscht (Einsatz neuer Technologien muss begründbar sein).
  - Informiert bleiben: Bulletins lesen, Mail Lists abonnieren, Systeme pflegen (Patches)
  - Ausbildung und Dokumentation (wo ist was installiert, welche Version, welche Patches?)
  - Organisat. Regelungen, Policies, Verantwortlichkeiten
  - Option: Outsourcing an starke Partner wie Hoster, ISP, ASP (Konzentration auf das eigene Geschäft)
- Audits
  - Überprüfung der korrekten Umsetzung
  - Identifikation von Schwachstellen
  - Unterstützung durch vertrauten neutralen Berater

Fazit: Wirksamkeit und Aufwand

Das berühmte schwächste Glied!