Application Security Audits – Applikationen auf den Zahn gefühlt

EUROFORUM
SecurITy Forum 2006, 05.04.2006, Christoph Baumgartner

…mit Sicherheit bessere Lösungen

Agenda

- Fakten
- Grundsatzfragen
- Angreifertypen
- Application Security Audit
- Zweck und Nutzen
- Knackpunkte
- Praxisbeispiel
- Applikatorische Sicherheit
- Techniken
- Typische Schwachstellen
- Empfehlungen
- Kosten und Aufwand
- Beantwortung von Fragen

Diverse Anhänge zu OSSTMM

Fakten

- Standardapplikationen werden ausgiebig getestet, aber
  - bestehende Applikationen/Systeme können nicht ohne weiteres abgelöst werden
  - Applikationen müssen oft an den unternehmensspezifischen Kontext angepasst werden
- Verteilte Systemkomponenten bergen aufgrund der Komplexität im Vergleich zu Stand-alone-Systemen per se ein erhöhtes Gefährdungspotential
- Software ist Hauptursache für technische Sicherheitsprobleme. Komplexität nimmt stetig zu:
  - Unix-Betriebssystem Solaris 7 umfasste ca. 400'000 Codezeilen
  - Boeing 777 fliegt mit ca. 7 Millionen Codezeilen
  - Microsoft Windows XP umfasst ca. 40 Millionen Codezeilen

Grundsatzfragen

- Schutzbedarf der Applikation und deren Daten
  - Was schützen (Systemabgrenzung)?
  - Vor was/wem (Bedrohungs- und Angriffsszenarien)?
  - Wie lange (Anlaufzeit Gegenmassnahmen, «Halbwertszeit» der Daten)?
- Wie/womit sollen Applikation und zugehörige Daten geschützt werden (Massnahmendefinition)?

Angreifertypen

- Ein **User** ist ein Anwender, welcher über das nötige Wissen verfügt, um die **Informatikmittel funktionsgerecht zu nutzen**.
- Als «**Skript Kiddie**» wird eine Person bezeichnet, welche über relativ **wenig Computer- und Netzwerkfachwissen** verfügt, aber unbedarft (vollautomatisierte) **Hackertools einsetzt**.
- Ein «**Hacker**»/«**Cracker**» **bricht ohne Erlaubnis** des Systemeigners meist via Computernetze **in Computersysteme ein** oder knackt das Lizenzsystem von Computerprogrammen. Dafür umgeht er bewusst Sicherheitsmechanismen. Die Beweggründe sind Ehrgeiz, möglicher finanzieller Profit, Geltungssucht, Idealismus oder Zerstörungswille.
- Ein «**Ethical Hacker**» ist ein Computer- und Netzwerkspezialist, welcher **im Auftrag des Systemeigners** nach **Systemverwundbarkeiten sucht**, welche ein «Hacker»/«Cracker» ausnutzen könnte.
- Ein «**Social Engineer**» versucht mittels Ausnutzens menschlicher Schwächen an vertrauliche Daten zu gelangen.
- **Malware** (Viren, Würmer und Trojaner) ist (teil-)automatisiert und versucht verschiedenartige Sicherheitsziele zu unterwandern.

Application Security Audit

Ein *Application Security Audit* berücksichtigt beispielsweise:

- Technische Aspekte, wie
  - Architektur
    - Infrastruktur
    - Vertrauensstellungen
    - Authentisierungsmechanismen
  - Sicherheitslücken
- Organisatorische Aspekte, wie
  - Compliance
  - Verantwortlichkeiten
  - User Management
  - Dokumentationspflege
  - Umgang mit Source Code
  - Patching-Prozess

Zweck und Nutzen

- **Qualitätssicherung** dank (unabhängiger) IT Security-Analyse
- **Compliance**-Nachweis bezüglich gesetzlicher Rahmenbedingungen und Vorgaben
- **Prävention** ermöglicht (in der Zukunft) direkte und indirekte Kosteneinsparungen
- **Awareness Building** auf allen Stufen
- **Know-how Transfer**
- **Argumentationsgrundlage** für zukünftige IT Security-Projekte bzw. -Aktivitäten

Knackpunkte

- **Fach-** und **Sozialkompetenz** der Tester und der am Projekt beteiligten Mitarbeiter
- **Vergleichbarkeit** und **Nachvollziehbarkeit** von
  - Offerten
  - Vorgehen
  - Resultaten und Dokumentationen
- **Compliance** zu Gesetzen, Standards und Vorgaben erwünscht, aber:
  - Nur rudimentäre Behandlung von technischen Audits in Standards (z.B. ISO/IEC 27001/17799)
  - Keine offiziellen Checklisten oder Guidelines verfügbar

Praxisbeispiel: Ausgangslage

- International tätige **Versicherung**
- **Web-Applikation**: Multi-Tier-Applikation, welche es Kunden und Partnern ermöglicht, Daten via Internet einzusehen und teilweise zu mutieren
- *Application Security Audit* als **Qualitätssicherungs-massnahme** vor offizieller Inbetriebnahme der Lösung

Praxisbeispiel: Tasks

- Durchführung des *Application Security Audits* in Form von
  - *Penetration Tests* der Basissysteme
    - **Via Internet** (unprivilegierte Tests aus Sicht «Hacker/Cracker»)
    - **Via LAN** mit gültigen Benutzernamen (privilegierte Tests aus Sicht «User»)
  - Analyse der Systemarchitektur
  - *Document Review*
  - Partieller *Code Review* (Authentisierungskomponenten)
- Auflistung der Schwachstellen
- Beurteilung der Risiken
- Erstellung des Massnahmenkatalogs
- Verfassen des Schlussberichts und der Projektpräsentation

Praxisbeispiel: angewandte Methoden

- *Open Source Security Testing Methodology Manual* (**OSSTMM**) [betreffend technischer Audits kompatibel zu ISO 17799, IT Grundschutzhandbuch, ITIL und SOX; -> siehe Anhang], bezüglich
  - Unprivilegierter Tests
  - Risikokategorisierung
- *Secure Programming Standards Methodology Manual* (**SPSMM**), bezüglich
  - Applikationsdesign
  - Privilegierter Tests
- White-box-Approach (Offenlegung des Untersuchungsobjekts)
- Iterativer Testzyklus
  - Sicherheitslücken aufdecken und Massnahmen definieren
  - Massnahmen umsetzen
  - Nachkontrolle

Praxisbeispiel: Fazit, Sicht Auftraggeber

- Erhöhte Sicherheit: Sicherheitsniveau des Untersuchungsobjekts während des Projekts kontinuierlich gesteigert
  - Betriebssysteme
  - Eigentliche Applikation
  - Drittapplikationen
- Gesteigerte Awareness: Standpunkt «durch die Brille eines Hackers/illoyalen Mitarbeiters»
- Know-how Transfer: Coaching der involvierten Mitarbeiter

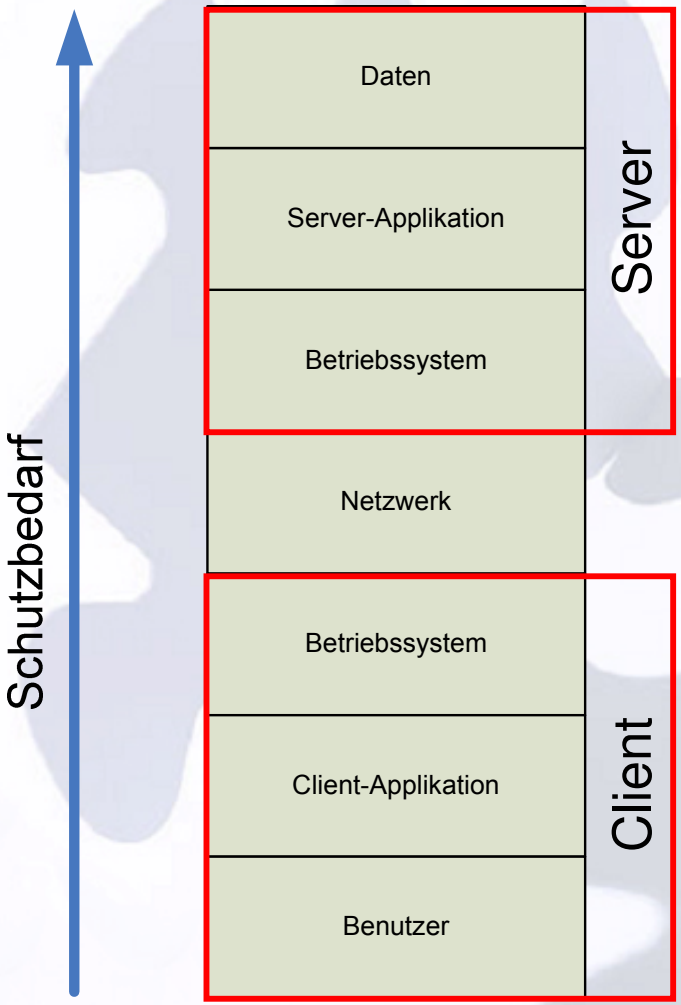
Applikatorische Sicherheit: 1. Teil

- Eine Applikation gilt als sicher, wenn keine involvierte Komponente dazu missbraucht werden kann, die definierten Sicherheitsziele negativ zu beeinträchtigen

  → Eine Applikation kann höchstens so sicher sein, wie das Umfeld, in dem sie betrieben wird

  - Betriebssysteme (Client, Server und zwischengeschaltete Netzwerkkomponenten)
  - Webserver
  - Datenbanken
  - etc.

Applikatorische Sicherheit: 2. Teil



- Generell gilt das Misstrauens-prinzip
- Jede Komponente (Hard- und Software) soll so abgesichert sein, dass sie die Sicherheits-anforderungen der darüber-liegenden Komponente erfüllt

Beispiele:

  - Benutzer kann nicht direkt mit Betriebssystem(diensten) interagieren
  - Netzwerk lässt nur Kommunikation von/mit zugelassenen Clients zu

Techniken (Auswahl): 1. Teil

Technik	Beschreibung	Zweck
Document Review	Analyse von Applikations-/Daten-bankdesign und Konzeptions-vorgaben	SOLL-Vorgaben eruieren und hinterfragen
Input Validation	Bewusster «Missbrauch» der Ein-gabefelder durch Eingabe von Malicious Code-Sequenzen (z.B. Buffer Overflow oder SQL Injection)	Prüfen, ob sich die Applika-tion in einen undefinierten Zustand bringen lässt oder ob der Programmverlauf oder das -verhalten verändert werden kann
Network Sniffing	Mitschnitt und Analyse des Netz-werkverkehrs zwischen verschie-denen Komponenten	Prüfen, ob sich (sicherheits-) relevante Informationen (User-ID, Passwort oder vertrauliche Daten) erlangen lassen

Techniken (Auswahl): 2. Teil

Technik	Beschreibung	Zweck
API Monitoring	Analyse des Datenverkehrs zwischen Applikation und Betriebssystem-komponenten	Suche nach Schwach-stellen in interner Kommunikation
Reverse Engineering / Debugging	Analyse von Systemaufrufen (z.B. Verschlüsselungsaufrufe) im lauf-fähigen Programm	Suche nach systemaufruf- und applikationsbasierten Sicherheitslücken
Decompilation	Versuch, den zu Grunde liegenden (Human readable) Source Code aus dem kompilierten Programm zu regenerieren	Konvertierung zwecks vereinfachter Schwach-stellensuche im Source Code
Code Review	Analyse des Original-Source Codes	Suche nach Schwach-stellen im Source Code

Typische Schwachstellen

- Dokumentierte Funktionalität widerspricht (teilweise) der implementierten
- Bei fehlerhaftem Login wird explizit angegeben, ob User und/oder Passwort falsch sind
- Standard-Datenbankuser-Account mit Defaultpasswort
- Proprietäre Verschlüsselungsalgorithmen und -routinen
- Buffer Overflow-Anfälligkeiten aufgrund veraltetem Patch Level
- Hardcodierte User-ID-Passwortkombination
- Fehlermeldungen erlauben Rückschlüsse auf Architektur
- Direkte Kommunikation mit Datenbank möglich
- Passwort-Policy für Benutzer nicht umgesetzt

Empfehlungen: 1. Teil

- Sicherheitsaspekte bereits in Designphase berücksichtigen
- Zugriffsrelevante Funktionen ausschliesslich Server-Komponentengesteuert (zentraler Ansatz)
- Server-Komponente sollte auch einem versierten Angreifer trotzen können, der bereits im Besitz eines gültigen User-Logins mit eingeschränkter Berechtigung ist
- Kommunikation zwischen Komponenten verschlüsseln
- Auditing (Logging) sämtlicher Aktivitäten
- Verfremdete Test-Daten für externe Audits bereitstellen

Empfehlungen: 2. Teil

- Management Attention
- Präzise Projektplanung und -durchführung:
  - Rahmenbedingungen (inkl. Kontrolle über deren Einhaltung)
  - Rollendefinition (inkl. Notfallplanung)
  - Zeitplanung (min. 2 Kalendermonate Projektdauer einplanen)
  - Voraborientierung der Mitarbeiter sinnvoll?
  - Protokollierung aller Testaktivitäten und Mitschnitt des Netzwerk-verkehrs
- White-Box-Approach (Offenlegung des Untersuchungsobjekts) anstatt Black-Box-Approach
- Nachvollziehbare Methodik
- Regelmässige Überprüfungen und Nachkontrollen
- Einbezug externer Spezialisten sichert Objektivität

Kosten und Aufwand

- Externe Kosten:
  - Minimum: CHF 12‘000
  - Maximum: offen
  - Durchschnitt: CHF 25‘000 – 40‘000
- Zeitaufwand seitens Auftraggeber bzw. Betreuer:
  - Minimum: 30 Minuten pro Testtag
  - Besser: Bereitschaftsdienst Kontaktperson und System-administratoren während der Durchführung aller Tests
  - Optional: gemeinsame Arbeit mit Auftragnehmer
- Zusätzlich: Umsetzung Massnahmen

Beantwortung von Fragen

Besten Dank für Ihre Aufmerksamkeit!
Gerne beantworte ich Ihre Fragen:

Christoph Baumgartner
lic. oec. publ., OPST
CEO / Senior Consultant

info@oneconsult.com
+41 (0)79 421 20 01

OneConsult GmbH
Zürcherstrasse 73
8800 Thalwil
Schweiz

http://www.oneconsult.com
info@oneconsult.com
Tel. +41 (0)43 443 52 52
Fax +41 (0)43 443 52 62

OSSTMM: Überblick

- Abkürzung von «**Open Source Security Testing Methodology Manual**»
- **Erstausgabe 2001**, entwickelt und kontinuierlich weiterentwickelt unter der Leitung **von ISECOM** (Institute for SECurity and Open Methodologies), http://www.osstmm.org
- **Offene**, und **frei verfügbare Methodik zur**
  - **Planung**
  - **Durchführung**
  - **Dokumentation** (mit Zielgruppe: IT Spezialisten)

  von technischen Sicherheitsüberprüfungen
- **Verhaltenskodex für Tester**

OSSTMM: Aufbau

- OSSTMM besteht aus
  - der **Methodologie**
  - den «**Security Metrics**» (Risk Assessment Values, RAVs)
  - den **Formularvorschlägen** (welche die Minimalanforderungen an den Informationsgehalt / Dokumentationsgrad eines OSSTMM-konformen Tests beinhalten bzw. darstellen)
- Methodologie ist **hierarchisch gegliedert**
  - Besteht aus sich überlappenden «**Channels**» (Sektionen), welche **sämtliche Sichten** der Informations-, System-, Kommunikations-, Prozesssicherheit und der physischen Sicherheit **abdecken**
  - **Channels** sind wiederum **in Module gegliedert**, welche in Form von verschiedenen Tasks durchlaufen werden

OSSTMM: Security Metrics (Risk Assessment Value (RAV)), 1. Teil

- «Risk Assessment Value» (RAV) stellt **Sicherheits-niveau des Untersuchungsobjekts als Zahl** dar und wird anhand dreier Variablen ermittelt:
  - **Operative Sicherheit** (OpSec): bewertet Sichtbarkeit, Vertrauensstellungen und Zugriffspunkte (Interaktions-möglichkeiten)
  - **Kontrollausgleich** (Loss Control (LC)): berücksichtigt implementierte Sicherheitsmechanismen und hat positiven Einfluss («Bonuspunkte») auf RAV
  - **Aktuelle Sicherheit** (ActSec): detektierte Risiken («Security Limitations») werden hinsichtlich ihres Bedrohungspotentials gewichtet

OSSTMM: Security Metrics (Risk Assessment Value (RAV)), 2. Teil

$$OpSec_{base} = 100 - \frac{OpSec_{sum}}{(Scope + OpSec_{sum})}$$

$$LC_{base} = Scope \times \frac{LC_{sum} \times 0.1}{(Scope + OpSec_{sum})}$$

$$ActSec_{base} = \frac{ActSec_{sum}}{Scope}$$

$$RAV = OpSec_{base} - \left(\frac{OpSec_{base} \times ActSec_{base}}{100}\right) + \left(\frac{OpSec_{sum}}{Scope + OpSec_{sum}} \times \frac{LC_{base}}{100}\right)$$

	Scope		Loss Controls
Scope	10	Authentication	5
		Non-Repudiation	2
	Operational Security	Confidentiality	2
Visibility	10	Privacy	0
Access	14	Indemnification	1
Trust	1	Integrity	0
Op Sec Δ	-25	Safety	2
Op Sec Total	25	Usability	1
Op Sec % of Scope	97.5	Continuity	2
		Alarm	1
		Loss Controls Δ	1.6
		Loss Controls Total	16
		Loss Controls % of Op Sec	6.4
		Loss Controls % of Scope	16

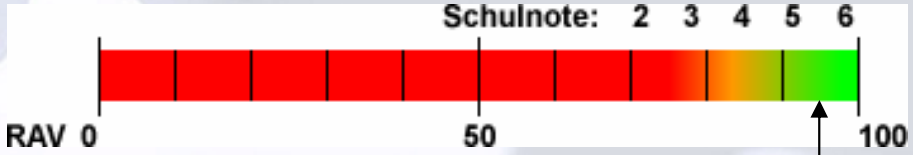
Security Limitations Values

	Verified	Identified
Vulnerability	1.34615385	1.35996055
Weakness	1.33234714	1.34601224
Concern	1.31868204	1.33220699
Exposure	1.30515710	1.31854332
Anomaly	1.29177087	1.30501980

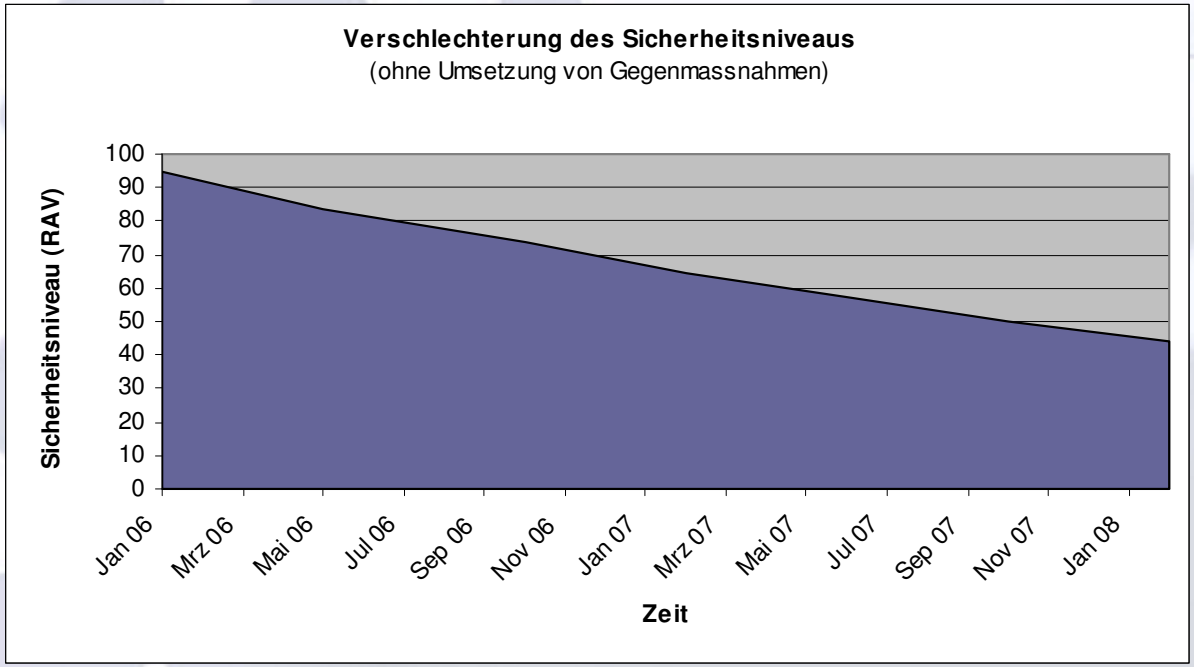
	verified	identified	total
Vulnerabilities	8	0	10.76923077
Weaknesses	7	0	9.32642998
Concerns	8	0	10.54945633
Exposures	10	0	13.05157097
Anomalies	0	0	0
		Security Limitations Δ:	43.69668805
		Security Limitations Total:	4.36966881
		Actual Delta:	-67.09668805
		Actual Security:	99.72230331

RAV Calculation	
Opsec base	99.28571429
LC base	0.457142857
ActSec base	4.369668805
RAV	94.95052271

Nicht Bestandteil des OSSTMM



OSSTMM: Zeitabhängige Verschlechterung des Sicherheitsniveaus



Degradation	
Cycle (days)	128
Degradation (%)	12
Date:	19.01.2006

Helper table for degradation	
19.01.2006	94.95052271
27.05.2006	83.55645998
02.10.2006	73.52968478
07.02.2007	64.70612261
15.06.2007	56.94138790
21.10.2007	50.10842135
26.02.2008	44.09541079

Bemerkungen:

- Laufend werden neue Schwachstellen entdeckt und neue Angriffsmethoden entwickelt
- Falls keine Massnahmen (z.B. Security Patching) umgesetzt werden, verschlechtert sich das Sicherheitsniveau im Lauf der Zeit

OSSTMM: Compliance

Kompatibel hinsichtlich der **Durchführung von technischen Sicherheitsüberprüfungen** zu diversen Gesetzen und «Best Practices» (bis Version 2.1 nur für Remote Tests):

- SOX (Sarbanes-Oxley Act)
- ISO/IEC 17799 (Code of Practice)
- IT GSHB (BSI IT Grundschutzhandbuch)
- ITIL (IT Information Library)
- SET (Secure Electronic Transactions)
- FISCAM (Federal Information System Control Audit Manual)
- etc.

Die komplette Liste kann dem OSSTMM entnommen werden.

OSSTMM: Zertifizierungsmöglichkeiten

Zertifizierungen, welche (unabhängig voneinander) die Listung als **offizielle OSSTMM Auditoren** auf der OSSTMM/ISECOM-Website ermöglichen:

- OPSE (**OSSTMM Professional Security Expert**): belegt, dass der/die Zertifizierte genaue theoretische Kenntnisse des OSSTMMs hat (wie, warum und wann das OSSTMM angewandt wird)
- OPST (**OSSTMM Professional Security Tester**): belegt, dass der/die Zertifizierte über die nötigen Fähigkeiten verfügt, um als professioneller Security Tester nach OSSTMM zu arbeiten
- OPSA (**OSSTMM Professional Security Analyst**): belegt, dass der/die Zertifizierte über die nötigen Fähigkeiten verfügt, Testresultate richtig zu interpretieren und Testerteams zu koordinieren

OSSTMM: Nutzen

- **Quantifizierbarkeit** – Zahlenwerte statt «Bauchgefühl» ermöglichen den Vergleich
- **Konsistenz** – Replizierbarkeit der Ergebnisse
- **Gültigkeit über** das «**Jetzt**» **hinaus** – proaktives statt reaktives Handeln
- **Basierend auf Leistung von Tester** bzw. Analyst statt auf «Brands»
- **Vollständigkeit** und **Genauigkeit**
- **Gesetzes-** und **Standardkonformität**
- **Zertifizierungsmöglichkeit** (Tester und Projekt)