Webunsicherheit 2.0

Problemsensibilisierung, Sicherheitslöcher erkennen, Methoden.

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
  - → **Webanwendungen!**

1. Einleitung, neue Ziele

- Verlagerung Ziele: **Webanwendung**
  - > **Anwendung:**
    - Steht im Netz !
  - > **„Junge" Technologie —> Beherrschbarkeit**
  - > **HTTP: „missbrauchtes Protokoll"**
    - RFC 1945 (1996), seit 1990
    - ursprünglich nur für Datenübertragung
      - HTTP-Authentifizierung: Ergänzung (RFC 2617)
      - Stateless HTTP
        - » Sessionhandling: aufgepropft
        - → Applikation
  - > **Einige Fehler: Google hacking**

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:

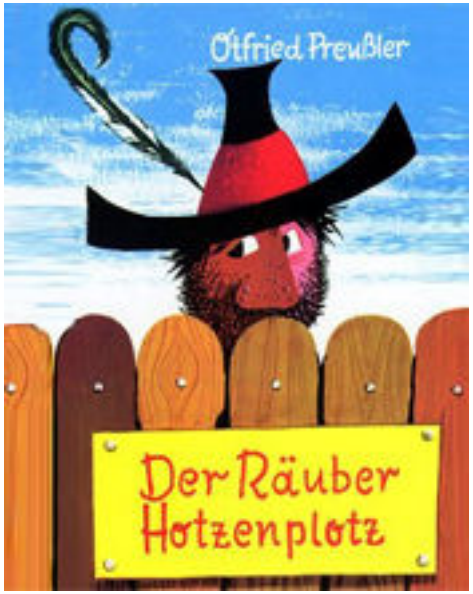
Z1. Zero Day Attacks

Best Practices for Preventing Top 20 Risks

1. Einleitung, neue Ziele

> Cash Cow für mafiöse Strukturen

- Malwareverteilung von Webseiten
- Rekonfigurierung von SoHo-Routern
- Banken
  (Malware im Browser: MITB)
- ebay/Paypal
- ....



1. Einleitung, neue Ziele

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

1. Einleitung, neue Ziele

- Infektion mit Malware: IE, Flash, Realplayer, Acrobat
  - 12/2008: Zero-Day IE (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 !

- Prominente wie
  - Adobe Vlogit! (Asprox Botnet)
  - UN,
  - UK Government
  - DHS?

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
  - CSRF

> Ohne jegliche Eingabefilter

> Aber auch ohne jegliche Bypass-Tricks ;-)

2.1. Typische Gefahren / XSS

- **XSS**
  - **Reflected:**
    Einfachster Fall: im POST-Feld / GET (URL)
    `<script>alert(document.cookie)</script>`
  - **Stored/Persistent:**
    Webseite, wo man JS **speichern** kann (Forum, Seite für den Applikationsadmin, Webmailer, Collab.)
  - **(DOM based, local):**
    lokale Browser-Attacke mit JS

Horde, Cisco IOS 11.0-12.4

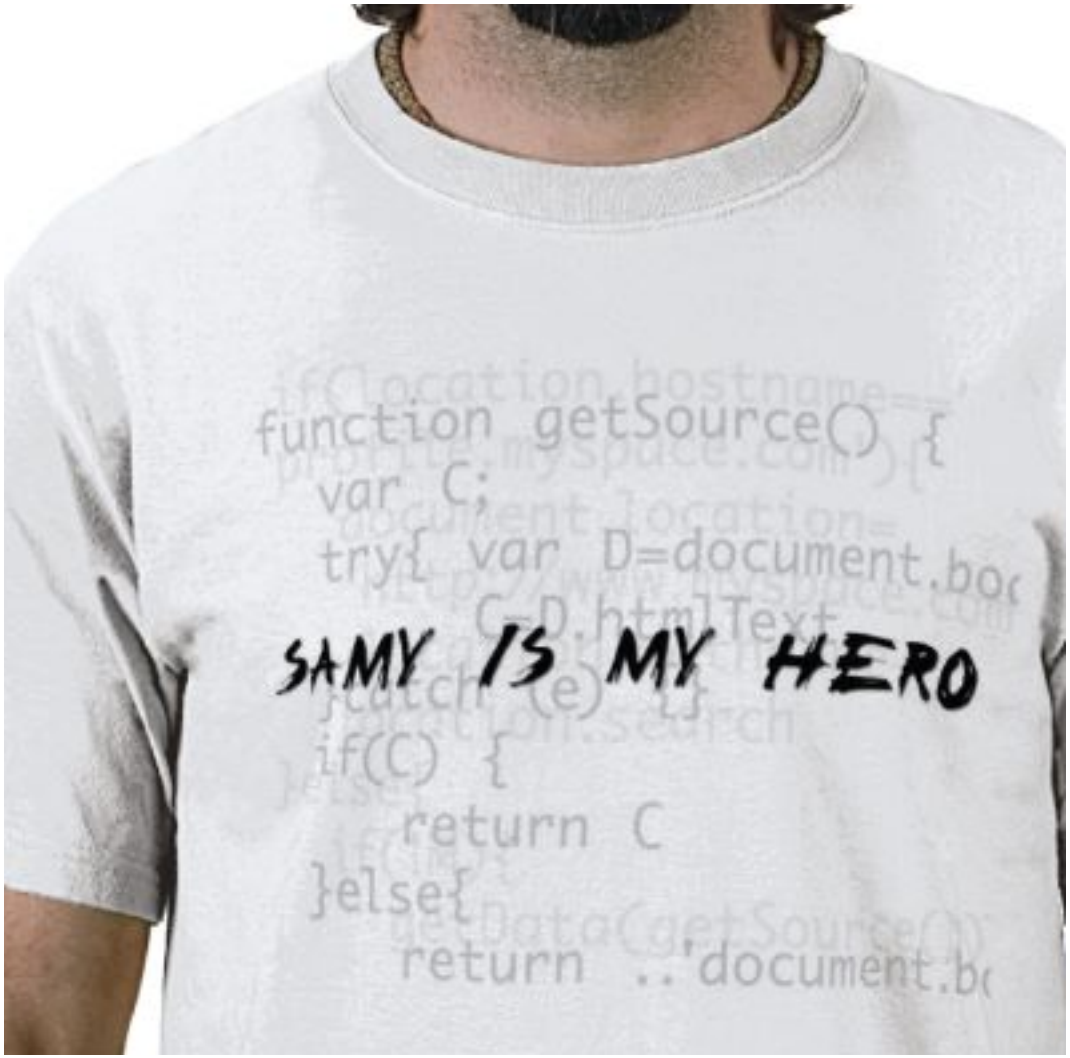
phpBB

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

2.1. Typische Gefahren / XSS

- **XSS: Prominente Beispiele**
  - **4/2009: Twitters Mikeyy/Stalkdaily**
    - 17jähriger wollte stalkdaily.com „bewerben“
    - JavaScript in „Bio“, Stored XSS
  - **10/2005: Samy-Wurm @ MySpace**
    - Experiment eines 19jährigen „Samy is my Hero“
    - In CSS eingebettetes JS –> XSS
    - 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

2.1. Typische Gefahren / XSS

- **XSS: Krasses Beispiel**

  > **IE bis exkl. Version 8:**
  - MIME-Sniffing für z.B. Bilder (vgl. Unix "file" ;-))
  - D.h. IE vertraut ausliefernden Typ nicht
  - Schöne Möglichkeit von Stored XSS

  > **Erinnerung (Botnetze):**
  - PHP LFI/RFI
  - Hinterlegen eines Bildes mit PHP Code am Anfang
  - Wird (im Ggs. zu oben) in WebApp ausgeführt

2.2. Typische Gefahren / Injections

- **Injections**

usa.kaspersky.com
TPB, M$ UK

  - **SQL Injections** (Ziel: DB):
  - ✗ SELECT * FROM u WHERE usr='feldA' AND pw='feldB'

    Szenario: Login-Feld Übergabe von `admin'--` / `' OR 1=1--`

    WHERE usr='**admin'**--' AND pw='any'

    WHERE usr='**' OR 1=1**--' AND pw='any'

Linksys WAG54G2
Santy.A/viewtopic in phpBB
Spreadfirefox/Twiki
Cacti

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

    Szenario: Appl. übergibt `zkette` an definiertes CMD1:
  - ✗ `http://beispl.de/suche.ext?q=zkette`
  - ➔ `http://beispl.de/suche.ext?q=zkette<sep><cmd2>`

    `<sep>`: Separator wie `;` `|` `&` `%0a` `` `` `` `||` `&&`

    `<cmd2>`: Unix: cat, ls, wget, tftpclient, netcat

2.3. Typische Gefahren / CSRF

> CSRF (XSRF, Session Riding)

- **Angriff auf bestehende Browser-Session**

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

```
<img src="http://www.bank.com/transfer.do?
        frmAcct=document.form.frmAcct&
        toAcct=31337&amount=3434.43">
```

  - HTML-Mail: passiv ; aktiv: Phishing (klicke!)
  - Per 2. Webseite (defaced)

2.3. Typische Gefahren / CSRF

> CSRF

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

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

  - Oder einfach GET statt POST versuchen
  - Kombination:
    - » CSRF in XSS-Lücke

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
  - 1/08 Mexiko DSL-Router:

    Drive-by Pharming (DNS verbiegen)
  - Cisco IOS (2008/9): HTTP-Interface zu Routern
  - 10/2007 @ Gmail: E-Mail Klau (Weiterleitungsfilter)
  - 1/2009 @ Novell GroupWise WebAccess

2.3. Typische Gefahren / CSRF

> Abhilfe

- Keine implizierte Authentifizierung durch Session-Cookie
  - Applikationslogik: Seite 1 –> Seite 2 –> Seite 3 *
  - (Kein XSS, kein GET)
  - (Per-)Page Tokens/Cookie 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

3. Maßnahmen

- **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
      - Viel mehr Vektoren/Möglichkeiten
        - » des Blockens
        - » des Umgehens der ***Filterung***

3. Maßnahmen

- **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-

Ein Auto kann lenken + bremsen und damit zuverlässig Unfälle vermeiden

3. Maßnahmen

> **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“
  - dir 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 der 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

> Für „<“ 70 Möglichkeiten (ohne z.B. UTF-7):

%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 (Darf User X Daten v. User Y sehen?)
- Applikationslogik, Designprobleme:
  - Kaputte Authentifizierung (WAF kann's nicht selbst)
    - » Unbegrenzte Anzahl failed logons
    - » 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 (user-id)
- 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!
- 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
    - Selbst: Möglichkeit, aufgedeckte Probleme zu fixen (an der Wurzel, WAF)
  - > **Experte:**
    - Muss Tester/Auditor speziell für Webapplikationen sein
    - Völlig andere Baustelle als Infrastruktursicherheit

3b. Maßnahmen, Audit (extern)

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

3b. Maßnahmen, Audit (extern)

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

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 Felder=Variablen pro Seite
      - » 5 URLs
      - » Annahme: 400 Möglichk. Eingabe: XSS/SQLi
    - 4*5*400=8000
    - 0,5 Minuten: 66,6h, ab 6660 €

3b. Maßnahmen, Audit (extern)

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

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

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

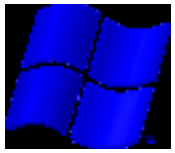
→ **Kombination manuell/automatisch**

3b. Maßnahmen, Audit (extern)

- **Werkzeuge**

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

Inter-cepting Proxys

Kommerziell + 

3c. Maßnahmen, safer programming

- **Des Pudels Kern!**
- **SCA-Tools: Preis!**
  - bislang eher nur kommerzielle brauchbar
  - Fortify, Ounce Labs, Coverity Prevent
- **Wann**
  - > **Währenddessen**
  - > **Danach?**
- **Wie**
  - > **Automatisch (SCA=Statische Code Analyse)**
  - > **Manuell: Code Review**

3c. Maßnahmen, safer programming

- **Wie?**
  - > **Automatisch / Manuell ?**
    - Manueller Code Review + große Applikationen = großer Aufwand
    - Rein automatisch mit SCA-Tool
      - A fool with a tool...
      - Erfordert Verständnis! => „halbautomatisch“

3c. Maßnahmen, SCA

- **Wann?**
  - > **Währendessen / Danach**
    - Vorsorge besser als Nachsorge:
      - Von vornherein sicher Programmieren (schulen!)
        - » SCA als Unterstützung in IDE benutzen
        - » Bibliotheken zur Eingabeüberprüfung (ESAPI, AntiXSS ,...)
      - Externe:
        - » Code + Code-Doku!
        - » PCI DSS v1.2:
          - Überprüfung 6.3.7
          - Codierungsrichtlinien 6.5

3c. Maßnahmen, SCA

- **Währendessen/Danach**
  - **> Tick später: Im QA-Prozess**
    - SCA-Tool + Mensch
    - Ggf: Begleitend: (Interner) Web-Vuln.-Scan
  - **> Code-Sicherheit posthum:**
    - Teurer, höherer Aufwand
    - Technisch (je nach Geld/Zeit): „Sicherheit obendrauf“

4. Zusammenfassung/Ausblick

- **Webanwendungen im Fadenkreuz**
  - > **Schutz nicht allein durch Technik**
    - Die Axt im Hause ersetzt den Zimmermann nicht: WAF, BBVS, SCA
      - – inhouse wo sinnvoll
      - – Audit besser extern

4. Zusammenfassung/Ausblick

- **Webanwendungen im Fadenkreuz**
  - > **Bündel von Maßnahmen, Best Practice:**
    - Sicher Programmieren von Anfang an!
    - Überprüfen durch
      - SCA + Mensch
      - Externer Audit/Scan
      - WAF, wo nötig+erfolgversprechend

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

dirk.wetter@sogeti.de / dirk bei drwetter punkt de