

thethe**code**campus</>

Penetrationstests für Entwickler

w1k

Philipp Burgmer

<> Software-Entwickler, Trainer
Web-Technologien, Sicherheit
burgmer@w11k.de </>

w11k GmbH - The Web Engineers

<> Individual Software für Geschäftsprozesse
Entwicklung, Wartung & Betrieb
Consulting </>

TheCodeCampus - Weiter.Entwickeln.

<> Technologie Schulungen
Projekt Kickoff, Code Reviews
Angular, TypeScript </>

- <> Entwickler einer Anwendung mit streng vertraulichen Daten
- <> Ankündigung Security Audit
- <> Eigene Analyse aus Interesse noch vor Audit
- <> Ergebnisse Audit verstehen und Fehler beheben
- <> Weitere eigene Analyse anhand des Quellcodes: weitere Fehler entdeckt

- <> Audits von Experten vermeiden? **Nein!**
- <> Zeit von Experten nicht mit "Kinderkram" vergeuden
- <> Viele Hacks viel zu einfach: AshleyMadison, Bundestag, ...
- <> **Low Hanging Fruits** erkennen und beheben
- <> Bessere Entwickler werden
- <> Sichere Software entwickeln

<> Komplexität von Software

- viele Technologien
- vieles gewachsen
- komplexes Zusammenspiel
- komplizierte Architekturen

<> Flüchtigkeitsfehler

- <> HTML / DOM / JavaScript
- <> HTTP / Browser Security Model
- <> Java / .NET / PHP / ...
- <> SQL / NoSQL
- <> SMTP / IMAP / Betriebssystem
- <> Alle bis ins Detail bekannt?

You cannot build secure web applications unless
you know how they will be attacked!

Simon Bennetts, Mozilla Security Team

- <> Seitenwechsel: Hacker werden oft Security Experten
- <> Entwickler können ein bisschen (White-Hat-)Hacker werden
 - Technologien, Sprachen, APIs wirklich verstehen lernen
 - Werkzeuge & Vorgehen kennenlernen
 - Kreativ werden!
 - Um die Ecke denken

<> [OWASP Top10 für Entwickler](#)

<> [OWASP WebGoat](#)

<> [hackthissite.org](#)

<> [pentesterlab.com](#)

<> Tangled Web - Michal Zalewski

<> The Web Application Hacker's Handbook - Dafydd Stuttard, Marcus Pinto

<> Blackbox Tests

<> Code Review

<> Mischung aus beidem

Blackbox Tests

- <1> Analyse
- <2> Zugangsbeschränkungen
- <3> Eingabe Validierung
- <4> Logikfehler in Anwendung
- <5> Betrieb / Laufzeitumgebung
- <6> SSL Zertifikate
- <7> Same Origin Policy Konfiguration

- <> Linux Distribution speziell für Penetrationstests
- <> Sehr viele Tools vorinstalliert
- <> Virtuelle Maschinen
- <> Host-Only Network

<> [PortSwigger Burp Suite \(Free / Pro\)](#)

<> [OWASP WebScarab](#)

<> [OWASP ZAP](#)

<> Intercepting Proxy: Schneidet sämtlichen HTTP(S) Verkehr auf

<> Spidering Tool: Analysiert besuchte URLs und Antworten; erstellt Sitemap / Context

<> Test Suite: Tools für Angriffe

<> Sollte nur mit Browser für Testen verwendet werden

<> Browser nicht zum Surfen verwenden

- <> Manuelle Konfiguration: aufwändig (Proxies, SSL Zertifikate, ...)
- <> Automatische Konfiguration über Plug-n-Hack
 - Netter Ansatz, aber nicht zuende gebracht
 - Addon nicht signiert, keine einfache Installation
- <> In `about:config` `xpinstall.signatures.required` auf `false` (nur bis Firefox 46 möglich)
- <> ZAP Welcome Page -> Button 'Configure Your Browser'
- <> Addon installieren

Demo

Setup

Blackbox Tests

Analyse

- <> Ziel: So viele Informationen über die Anwendung gewinnen wie möglich
 - Verwendete Technologien
 - Inhalt und Funktionsweise der Anwendung
 - Möglichkeiten für Benutzereingaben
 - Verzeichnisstruktur und Dateien

<> Informationen:

- Betriebssystem
- Server
- Sprachen
- Frameworks

<> Quellen:

- HTTP Header
- Quellcode im Client (HTML, JavaScript, CSS)
- URLs
- Exception provozieren (z.B. SQL Syntax über SQL Injection)

<> [CVEDetails.com](https://www.cvedetails.com)

- <> Verstehen für was die Anwendung da ist (fachlich)
- <> Login / Logout / Password-Reset Mechanismus ausprobieren
- <> Was wird im Client gemacht, was im Server
- <> Wie werden Daten ausgetauscht (Schnittstelle, Datenformat, ...)
- <> Wie werden Session Informationen im Client gespeichert
- <> ...

- <> Alle Formulare ausfüllen
- <> Alle Wizards durchgehen
- <> Auf Versteckte Formular-Felder prüfen
- <> URL Parameter beobachten
- <> Request Body anschauen

- <> Allen Links auf allen Seiten folgen
- <> Verhalten bei ungültigen Aufrufen
- <> Dateien für Tools: robots.txt, sitemap.xml
- <> Links im HTML
- <> Referenzen im Quellcode

<> Passive Spidering

- Sammeln der Informationen durch manuelles Browsen der Seite

<> Active Spidering

- Automatisches Folgen aller Links
- Ausfüllen von Formularen mit Zufallswerten
- Suche nach häufig verwendeten Wörtern / Pfaden
- Suche nach häufig verwendeten Tools / Bibliotheken

<> Automatische Analyse

- Nikto sammelt Informationen über Server

<> Für Tester:

- Anwendung kennen lernen
- Angriffspunkte ausfindig machen

<> Für Entwickler:

- Sollten eigentlich alles schon wissen
- Analysieren was wir preisgeben

Demo

Analyse

- <> Nur allgemeine, keine technischen Fehlermeldungen an den Client
- <> Keine Stacktraces bei Exceptions
- <> Directory-Listings wirksam verhindern
- <> HTTP-Header einschränken?
 - X-Powered-By
 - Server
 - X-Runtime, X-Version, X-AspNet-Version
- <> Sicherheit durch Verschleiern?

Blackbox Tests

Zugriffsbeschränkungen

- <> Zugriff auf die Anwendung als ein fremder Benutzer bekommen
- <> Ideal: Zugriff als Admin
- <> Aktionen als anderer Benutzer ausführen
- <> Größere Angriffsfläche haben
- <> Tiefer ins System eindringen

Zugriffsbeschränkungen - Ziele

- <> Authentifizierung
- <> Session Management
- <> Authorisierung

- <> Mechanismus verstehen
- <> Passwort Qualität testen
- <> 'Benutzername ausprobieren' testen
- <> 'Passwort raten' testen
- <> 'Passwort vergessen' Funktionalität testen
- <> 'Eingeloggt bleiben' Funktionalität testen
- <> 'Eindeutiger Benutzername' testen

- <> Mechanismus verstehen
- <> Token identifizieren
- <> Token verändern
- <> Token auf Vorhersagbarkeit prüfen
- <> Inhalt des Tokens prüfen
- <> Sichere Übertragung prüfen
- <> Beendung der Session prüfen
- <> CSRF testen

- <> Mechanismus verstehen
- <> Privilegierte Funktionen mit unprivilegiertem Benutzer aufrufen
- <> URL Parameter überprüfen und verändern (`edit`, `access`, ...)

- <> Benutzername raten
- <> Password Brute-Force-Attack
- <> Session ID Analyse
- <> Autorisierungsprüfung testen

- <> Möglichst keinen eigenen Login-Session-Mechanismus verwenden
- <> Konto sperren / Zugriff verlangsamen bei x ungültigen Passwort Eingaben
- <> 'Benutzernamen ausprobieren' verhindern
- <> Passwörter nicht im Klartext abspeichern (sichere Hash-Funktion mit Salt)
- <> Keine Login und Session Informationen in Logs

Blackbox Tests

Benutzereingaben

<> Was sind Benutzereingaben?

- Formular
- URL Parameter
- HTTP Header

<> Alles was der Benutzer / Angreifer beeinflussen kann

<> Alles was beim Server von außen ankommt

<> Alles was im Client von außen ankommt

- <> Cross-Site-Scripting (XSS)
- <> SQL Injection
- <> Command Injection
- <> Path Traversal
- <> ...

- <> Fuzzing Tool / Intruder / Active Scanner
- <> Zufällige Werte für Input um Reaktion Server zu beobachten
- <> Gezielte Werte für verschiedene Angriffsarten

Demo

Benutzereingaben

- <> Immer allen 'Benutzereingaben' misstrauen
- <> An jeweiligen Stellen validieren (Übergang Daten zu Code)
- <> *Web Application Firewall* kann helfen
- <> Bibliotheken / sichere APIs verwenden, z.B.
 - Prepared Statements für SQL in Java
 - Template Engine mit XSS Schutz
- <> Fehler finden:
 - Automatische Attacke
 - Code Review + manuelles Testen

Code Review

- <> **Kein Ersatz für Blackbox Tests**
 - Weniger Automatisierung
 - Hohe Komplexität
 - Was passiert wirklich? Ausführen!
- <> **Gefahren für Entwickler**
 - Schauen zu schnell über eigenen Code
 - Verlassen sich auf Bibliotheken

- <> Potenzielle Schwachstellen schneller finden, dann per Blackox ausgiebig testen
- <> Stellen finden, die von bestimmten Eingaben / Zustand abhängen

- <> Verarbeitung und Datenfluss von Benutzereingaben
- <> Verwendung von potenziell gefährlichen APIs
- <> Kommentare von Entwicklern
- <> Debug Parameter und Funktionalität

- <> IDE mit Datenfluss Analyse (z.B. IntelliJ für Java, [LightTable](#))
- <> Statische Code Analyse (z.B. FindBugs für Java)
- <> Auf Verständnis ausgelegte Editoren (z.B. Source Insight)

- <1> Datenfluss von Benutzereingaben analysieren
- <2> Code Signaturen suchen
- <3> Intensive Untersuchung kritischer Komponenten
 - Authentifizierung, Session Management, Authorisierung
 - Schnittstellen zu anderen Systemen
 - Aufruf nativer Code

Code Review - Datenfluss Benutzereingaben

<> Potenzielles Problem: Übernahme von Benutzereingaben in Code für anderes System (Injection)

<> Suche nach API Aufrufen zum Auslesen des HTTP Requests

Java: `getParameter`, `getQueryString`, `getHeader`, `getCookies`, ...

<> Datenfluss verfolgen

```
1 private void setTitle(final HttpServletRequest request) {  
2     final String requestType = request.getParameter("type");  
3     if ("history".equals(requestType) && request.getParameter("title") != null)  
4         title = request.getParameter("title");  
5     else  
6         title = DEFAULT_TITLE;  
7 }
```


<> Potenzielles Problem: String-Konkatenation mit Benutzereingaben

<> Wird oft von FindBugs gefunden

<> Suche nach hart codierten Strings

```
"SELECT, "INSERT, "DELETE, " AND, " OR, " WHERE  
(case-insensitive, Varianten mit/ohne Leerzeichen, ...)
```

- <> Potenzielles Problem: Benutzereingaben als Teil eines Pfads für Dateizugriff
- <> `../` kann für Zugriff auf andere Verzeichnisse verwendet werden
- <> Suche nach Verwendung von File APIs
 - Java: `getInputStream`, `getReader`, `FileInputStream`,
`FileOutputStream`, `FileReader`, `FileWriter`

- <> Problem: Entwickler vergessen etwas bekanntes zu beheben
- <> Suche nach *bug, problem, fix, crash, hope, bad, ...*

Zusammenfassung

- <> Technologien besser kennen lernen
- <> Angreifer-Denkweise aneignen: Ablauf Blackbox Tests
- <> Angreifern Leben schwer machen: möglichst wenig Informationen preis geben
- <> Blackbox Tests und Code Review durchführen: Projekt interner und externer Entwickler
- <> Professionelles Audit durchführen lassen

Philipp Burgmer
burgmer@w11k.de
@philippburgmer

www.thecodecampus.de
@theCodeCampus