FRO CYBERLOV

Webhacking

Från nyfikenhet till metod

Vad är webhacking?

Att bryta sig in	Utan tillstånd, mot någon annans system. Olagligt, oavsett avsikt.
Att testa med tillstånd	Pentesting, etisk hackning. Samma tekniker, men med ägarens godkännande.
Idag	Vi lär oss om teknikerna. Senare testas de i en anpassad miljö, ingen annanstans.

FRO

Innan vi börjar

Tillstånd först

Testa aldrig något du inte uttryckligen har fått lov att testa.

Idag: bara vår egen miljö

Allt vi visar och allt ni gör sen sker i en miljö byggd för det här.

Varför lära sig det här?

Ett helt yrkesfält	Pentestare, säkerhetsanalytiker, bug bounty-jägare: alla bygger på det här.
Defensivt tänk	Att förstå attacken är hur man bygger försvaret. Svårt att skydda från det man inte förstår.
Överallt	Nästan allt ni använder dagligen är en webbapplikation någonstans i kedjan.

FRO

OWASP Top 10

Branschstandardens karta över de vanligaste sårbarhetsklasserna.

Open Worldwide Application Security Project, Spana in owasp.org!

Hur en webbsida faktiskt fungerar

Klient & server	Din webbläsare (klient) pratar med en dator någon annanstans (server).
HTTP-förfrågan → svar	Du frågar efter en sida, servern svarar med HTML, CSS, JavaScript.
Var koden körs	HTML/CSS/JS körs i din webbläsare. Databaser och känslig logik körs på servern, dold för dig.

DEL 2

Rekognosering (Recon)

Hitta det som inte annonseras

Varför recon kommer först

Du kan inte attackera det du inte känner till

Vilken teknik körs? Vilka sidor finns? Vad är faktiskt exponerat?

Passivt före aktivt

Börja med information som redan finns tillgänglig, innan du knackar på själv.

whois

Vem äger domänen, sedan när, och ofta var den är registrerad.

```
$ whois exempel.se
Domain Name: exempel.se
Registrant: Exempel AB
Created: 2019-03-14
Name Server: ns1.exempel-dns.se
```

dig / nslookup

Visar vilka servrar som faktiskt svarar för domänen, och kan avslöja subdomäner.

```
$ dig exempel.se +short
185.34.12.9

$ dig test.exempel.se +short
185.34.12.44
# en testmiljö ingen länkade till, men som ändå svarar
```

robots.txt & sitemap.xml

Filer webbplatser lägger ut frivilligt för sökmotorer, en instruktion, inte ett skydd.

```
exempel.se/robots.txt

User-agent: *
Disallow: /intern-admin/
Disallow: /gammal-backup/
```

Wappalyzer

Läser en sida och listar vilken teknik den bygger på, ibland ner till versionsnummer.

```
exempel.se

WordPress 6.2
PHP 7.4
jQuery 1.12
nginx
```

nmap

Portskanning. Vad som faktiskt lyssnar på servern, inte bara det som är tänkt att vara publikt.

```
$ nmap -sV exempel.se
PORT     STATE  SERVICE    VERSION
22/tcp   open   ssh
80/tcp   open   http       nginx
443/tcp  open   https      nginx
8080/tcp open   http-proxy
# en port ingen nämnde, men som ändå svarar
```

gobuster / ffuf / dirb

Provar tusentals namn ur en ordlista mot en server och rapporterar vad som faktiskt finns.

```
$ ffuf -u https://exempel.se/FUZZ -w ordlista.txt
admin        [Status: 200]
backup       [Status: 200]
login        [Status: 200]
test         [Status: 403]
```

Sammanfattning: recon-verktygslådan

Passivt	whois, dig/nslookup, information som redan finns, utan att röra målet.
Aktivt	nmap, gobuster/ffuf/dirb, skickar egna förfrågningar för att hitta det som inte annonseras.

DEL 3

Klientsidan

Vad webbläsaren faktiskt visar

Visa källkod / DevTools

Ctrl+U visar hela sidans HTML. F12 öppnar DevTools: element, nätverk, lagring, allt.

```
exempel.se/kontakt

<!-- TODO: ta bort test-inloggning
     innan lansering: /test-login -->
```

Cookies & sessioner

En sessions-cookie är ofta det enda som säger "jag är inloggad", synlig och redigerbar av vem som helst.

```
DevTools → Application → Cookies

session_id
a3f9c1e7b2d84f...
HttpOnly: nej
```

JavaScript på klientsidan

Klientsidig "säkerhet" är ingen säkerhet. Allt en JS-fil kollar kan läsas, ändras eller hoppas över.

```
exempel.se/js/login.js

if (password === "sommar2024") {
  loggaIn();
}
```

Iframes

Ett fönster inuti ett annat, kan missbrukas: en osynlig iframe ovanpå en knapp (clickjacking).

```
exempel.se

<iframe src="bank.se/overfor"
  style="opacity:0">
```

Sammanfattning: klientsidan ljuger aldrig helt

Allt är synligt	Källkod, cookies, JavaScript: allt går att läsa i webbläsaren.
Kontroll måste ske på servern	Det klienten "bestämmer" kan alltid kringgås. Riktig säkerhet lever på serversidan.

FRO

DEL 4

Åtkomstkontroll

Vem är du, och vad får du göra

Autentisering vs auktorisering

Autentisering

— "Vem är du?"

— Inloggning: användarnamn och lösenord

— Bevisar identitet

Auktorisering

— "Vad får du göra?"

— Kontrolleras efter inloggning

— Den viktigaste distinktionen i avsnittet

IDOR (Insecure Direct Object Reference)

Servern kollar att du är inloggad, men inte att just den resursen tillhör dig.

```
exempel.se/order?id=1042

id=1042 → din order
id=1043 → någon annans order
```

Trasig åtkomstkontroll

"Ingen länk dit" är inte samma sak som "skyddad".

```
exempel.se/admin-panel

Ingen inloggning krävd.
Rakt in, om man känner till adressen.
```

Sammanfattning

Kontrollera alltid på servern

Aldrig bara klienten, och aldrig bara "öppen om ingen hittar den".

Varje resurs, varje gång

Kontrollen måste ske på just den resursen, för just den användaren, vid varje förfrågan.

DEL 5

Injektionsattacker

När indata blir kod

Vad är en injektionsattack?

Grundidén

Ett program bygger en kommando- eller frågesträng genom att klistra ihop text, inklusive det användaren skrev in.

När det går fel

Om indata inte hanteras säkert kan den ändra hela innebörden av kommandot, inte bara fylla i ett värde.

SQL-injektion, grunderna

Rätt hanterat blir värdet bara data. Fel hanterat blir det en del av frågans logik.

```
exempel.se/login

SELECT * FROM users WHERE
user='admin' AND pass='
' OR '1'='1
'  →  alltid sant
```

SQL-injektion, lite längre

UNION-baserad injektion kombinerar resultatet från två frågor, vilket kan extrahera data ur helt andra tabeller.

```
Inmatning i sökfältet
' UNION SELECT username, password
  FROM users --

→ hela användartabellen i sökresultatet
```

sqlmap

Automatiserar det du precis gjorde för hand: hittar injekterbara fält, identifierar databastyp, extraherar data.

```
$ sqlmap -u "https://exempel.se/login" --forms
[INFO] testing parameter 'user'
[INFO] parameter 'user' is injectable
[INFO] back-end DBMS: MySQL 8.0
do you want to dump table 'users'? [y/N]
```

Kommandoinjektion

Samma grundidé, men mot serverns kommandorad. Vanligt i verktyg som anropar systemkommandon åt dig.

```
Ett formulärfält som pingar en adress
Inmatning: 127.0.0.1; whoami

PING 127.0.0.1: 56 data bytes
www-data
```

FRO

Cross-Site Scripting (XSS)

Samma grundidé som SQL-injektion, men målet är andra användares webbläsare, inte servern.

```
exempel.se/kommentarer

<script>
  stjalCookie(document.cookie)
</script>
```

Sammanfattning: lita aldrig på indata

Samma mönster, olika mål

SQL-injektion mot databasen, kommandoinjektion mot skalet, XSS mot andra användares webbläsare.

Lösningen är alltid densamma

Behandla användarens indata som data, aldrig som kod eller kommandon.

DEL 6

Verktygslådan

Resten av det ni behöver känna till

Burp Suite

Fångar upp varje förfrågan mellan webbläsare och server innan den skickas. Läs den, ändra den, skicka om den.

```
Burp Suite → Proxy → Intercept
POST /login HTTP/1.1
Host: exempel.se

user=admin&pass=test123

[Forward]  [Drop]  [Edit]
```

curl, snabb repetition

Samma verktyg ni redan kan från terminal-kursen, nu i pentestkontext.

```
$ curl -s https://exempel.se/api/status
{"status": "ok"}

Ingen rendering, inget klick.
Rådata rakt i terminalen.
```

Automatiserade skannrar

Nikto / OWASP ZAP / nmap vulnscan	Skannar automatiskt efter kända, vanliga sårbarheter över hela sidan.
Snabbt, men ytligt	Bra för att hitta det uppenbara. Ersätter aldrig att faktiskt förstå vad man tittar på.

Hela verktygslådan, en bild

1

Recon

nmap, gobuster/ffuf, whois, dig

2

Analys

DevTools, Burp Suite

3

Exploatering

sqlmap, manuell injektion

DEL 7

Fler sårbarhetsklasser

Sådant ni bör känna igen

Säkerhetskonfiguration

Standardlösenord, exponerade .git/.env-filer, pratsamma felmeddelanden som avslöjar mer än de borde.

```
exempel.se/api/anvandare/99999

Error: Unknown column 'anvandare_id'
in /var/www/api/db.php:142
```

Föråldrade komponenter

Kända sårbarheter, kända lösningar

Gamla bibliotek och ramverk har ofta publika CVE:er, med publika exempel på hur de utnyttjas.

Uppdateringar är säkerhetsarbete

Att inte uppdatera är att medvetet lämna kända hål öppna.

Svaga kryptografiska val

Kryptering som ser säker ut, men inte är det	Gamla algoritmer, korta nycklar, egenhändigt uttänkta "krypteringar" som bara är obfuskering.
Alltid en risk	Svag kryptografi kan dyka upp varsomhelst i kedjan.

DEL 8

Från webben till servern

Ett första steg är ofta bara ett första steg

En webbsårbarhet är ofta bara ett första steg

Initial åtkomst

En webbsårbarhet ger dig ofta ett litet fotfäste, inte hela systemet direkt.

Vad händer sen?

Erfarna angripare frågar alltid: vad mer kan jag nå härifrån?

Rättigheter & privilege escalation, kort repetition

Kopplar till terminal-kursen

chmod, sudo, läsa/skriva/köra: samma rättighetsmodell gäller på servern du landar i.

Nästa steg efter att man kommit in

Ett fotfäste med låga rättigheter är sällan slutmålet. Frågan blir vad man kan nå härifrån.

DEL 9

Etik & avslutning

Vad man gör med det man hittar

Ansvarsfull rapportering

Hittar du en riktig sårbarhet	Rapportera till ägaren, ge dem tid att fixa den innan något offentliggörs.
Disclosure-processen	De flesta seriösa organisationer har en tydlig väg in för säkerhetsrapporter. Använd den.

Bug bounties

Ett helt yrkesfält

Företag betalar för att hitta hål innan någon annan gör det, ofta via plattformar med tydliga regler.

Det ni lärt er idag är grunden

Samma tekniker, samma verktyg, bara med tillstånd och en tydlig process.

Reglerna för CTF:et

Bara vår egen miljö

Allt ni testar härnäst sker i en miljö byggd specifikt för det.

Fråga om något är oklart

Bättre att fråga en gång för mycket än att gissa fel.

TACK

Lycka till!

Frågor innan vi sätter igång?