Mer websäkerhet

Attackmetoder

Cookie poisoning

Cookiestöld

Cross-site scripting

Cross-site request forgery

Cookie poisoning

"Session hijacking"

Session-IDs skapas av servern och skickas med ett set cookie-anrop

Angripare modifierar cookie-data för att försöka öka rättigheter, "cookie poisoning"

Session IDs (SIDs) skall vara svåra att förutse, och cookies skall lagras säkert

Cookies avlyssnas eller sätts "brute force", chansa på troliga värden.

Manipulera cookies

Cookies ligger på din dator, så du kan avkoda och ändra.

Cookies kodas ofta med base64

Data kodat i base64 sparas enbart som ASCII-text. Detta gör att det inte går att stoppa in "farliga" tecken på fel platser.

Exempel: `Hello world!` blir `SGVsbG8sIHdvcmxkISA=`

Dock: Betrakta inte detta som kryptering. Det är en *kodning*.

Mer om kryptering senare.

Dålig sessionhantering

Sessions styrs av cookies

Farliga scenarier:

• Användarinformation (för inloggning) lagras med otillräcklig kryptering

• Användarinformation kan gissas och ändras på grund av dålig kontohantering

• Tillämpningen använder inte säker överföring (som HTTPs eller sFTP).

• Sessionsparametrar kan ändras manuellt av användaren

Cookiestöld

Cookies skall bara skickas till matchande domän

Det är ett fall av "same-origin policies"

Scripts lever också under dessa regler

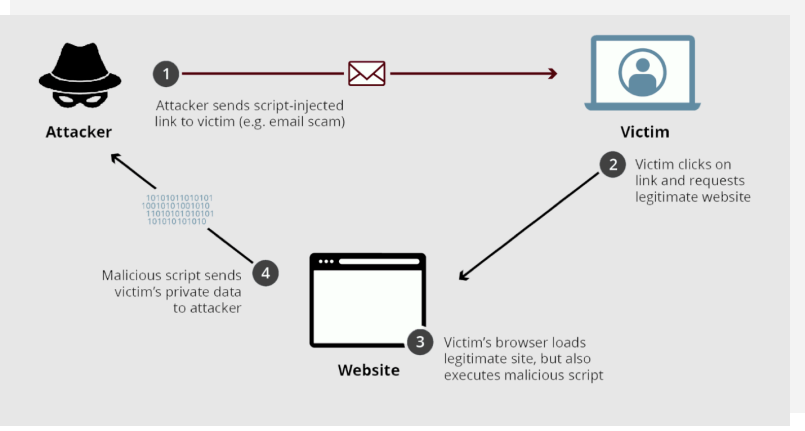
Tricket är att få ditt script inkluderat i svaret från en betrodd källa (som ger dig cookien)

Tekniken kallas *cross-site scripting*

Cross-site scripting, XSS

Detta är en typ av tekniker som används för att få angriparens scripts inkluderade i sidor från betrodda servrar

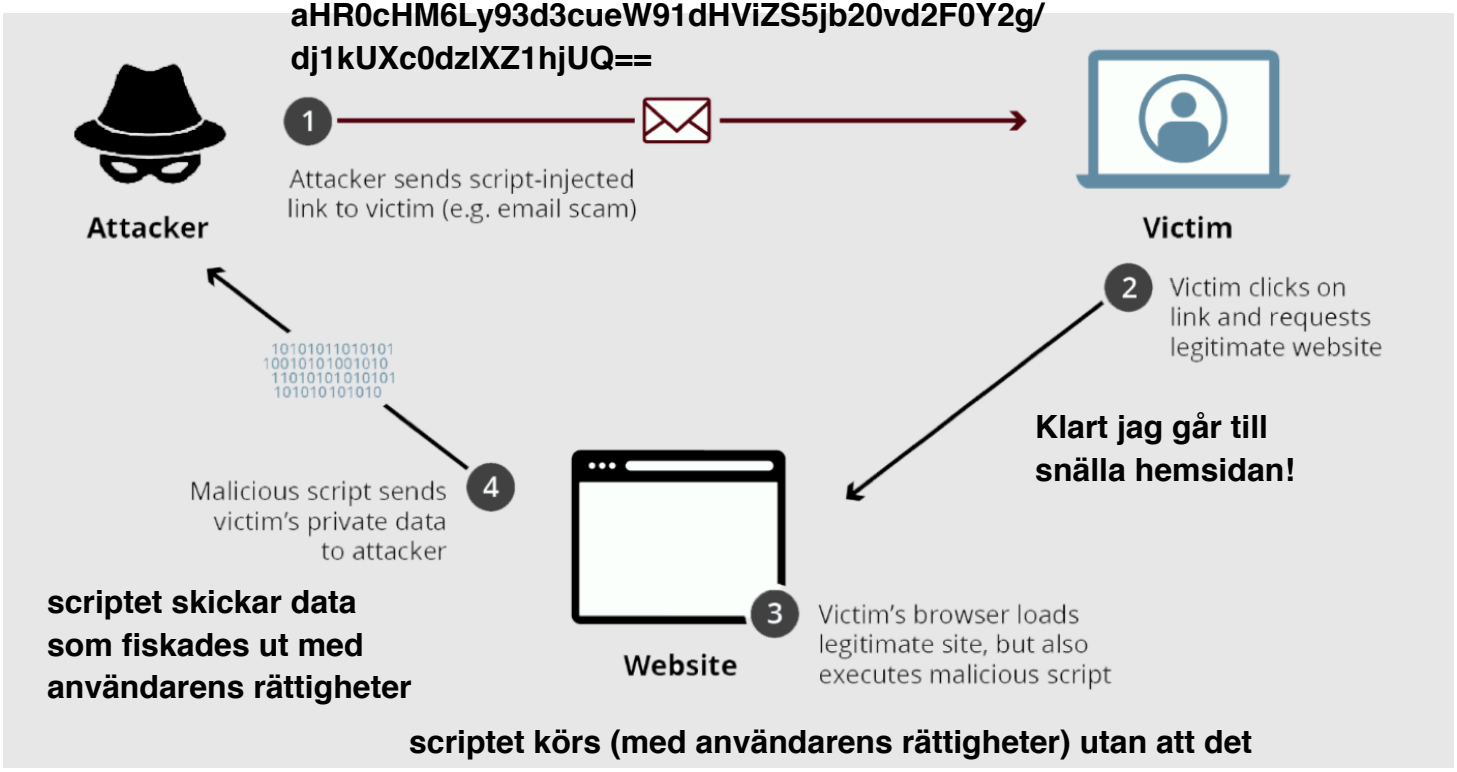
Det handlar om att hitta en öppning där man kan skicka in ett script i en websida



Script i "oskyldigt" textfält som körs av misstag

Offret luras att skicka angriparens script till servern

E-post: Besök snälla hemsidan LÄNK + script aHR0cHM6Ly93d3cueW91dHViZS5jb20vd2F0Y2g/dj1kUXc0dzlXZ1hjUQ==



scriptet skickar data som fiskades ut med användarens rättigheter

scriptet körs (med användarens rättigheter) utan att det märks aHR0cHM6Ly93d3cueW91dHViZS5jb20vd2F0Y2g/dj1kUXc0dzlXZ1hjUQ==

Cross-site scripting, XSS

Länkar med en massa base64 påhakat är farligt! Du kör ett script som du inte vet vad det är.

Min webläsare vägrar öppna sådana länkar av säkerhetsskäl.

Gissa vilka websidor som gör sådant hela tiden?

Returnerade (non-persistent) XSS

Script i "oskyldigt" tillfälligt textfält som körs av misstag för att det ekas

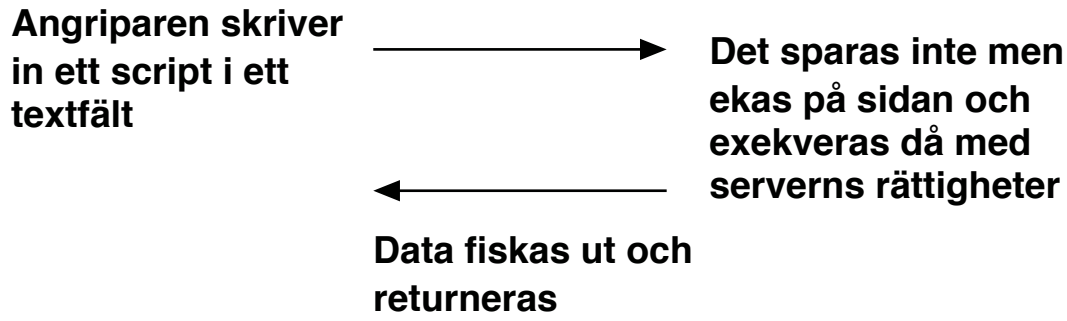
Det finns flera tekniker, exempel:

```
<A HREF="http://trusted.com/comment.cgi
mycomment=<SCRIPT alert('XSS!')></
SCRIPT>">Clickhere</A>
```

Om sidan med kommentaren ekar argumentet så exekveras scriptet av kommentarsidan, med de rättigheter sidor har på den betrodda servern.

Inte bara kommentarssidor, men även sökmotorer, 404-sidor...

Returnerade (non-persistent) XSS

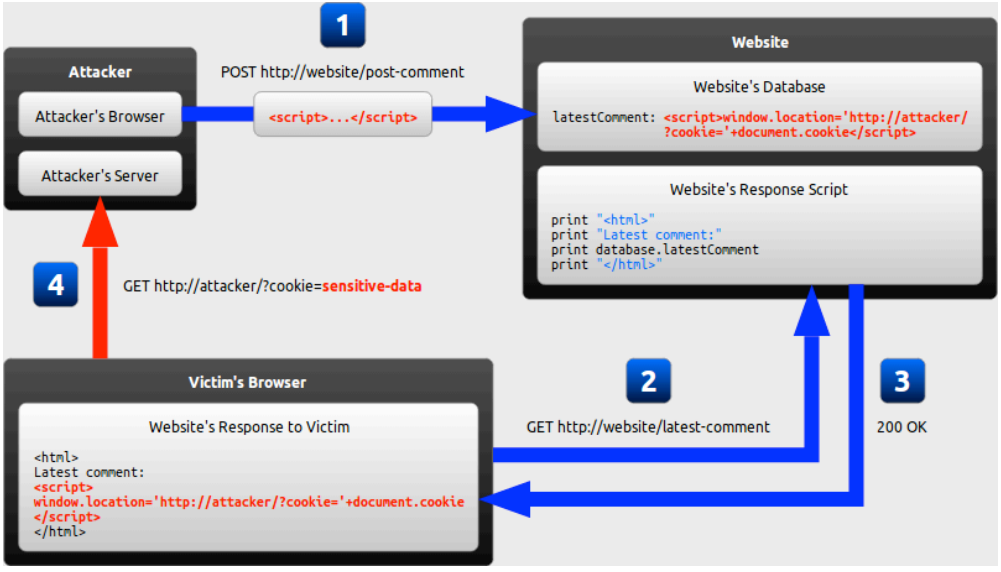


Attack mot serverns information!

Lagrade (persistent) XSS

Script i "oskyldig" text som körs av misstag när det visas för andra användare

I en lagrad XSS-attack sparar angriparen sitt script på den betrodda servern, till exempel på en diskussionssida.



Angriparen skriver in ett script i ett textfält som sparas på servern.

Scriptet lagras som en kommentar. Offret öppnar sidan.

Försvar mot XSS

Stäng av scripting (eller snarare, tillåt enbart från betrodda platser, i.e. använd Noscript)

Rengör indata väl!

Förbättra autenticeringen

Förbättra åtkomstkontrollen, så det blir svårare att stjäla data genom same-origin-policy

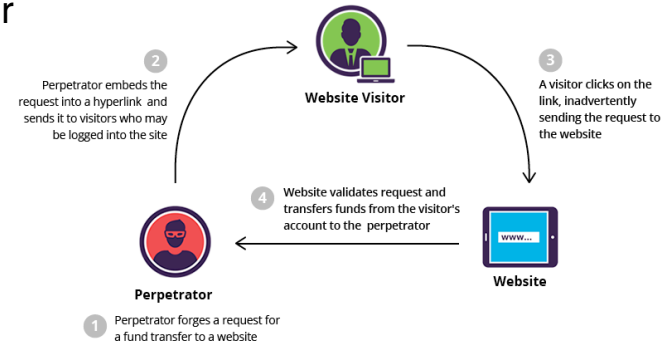
Cross-Site Request Forgery (CSRF)

Till synes legitim men falsk begäran som skickas från användare som får göra transaktionen

Motsatsen till en XSS-attack

En XSS-attack använder klientens förtroende för att exekvera kommandon hos klienten med serverns rättigheter

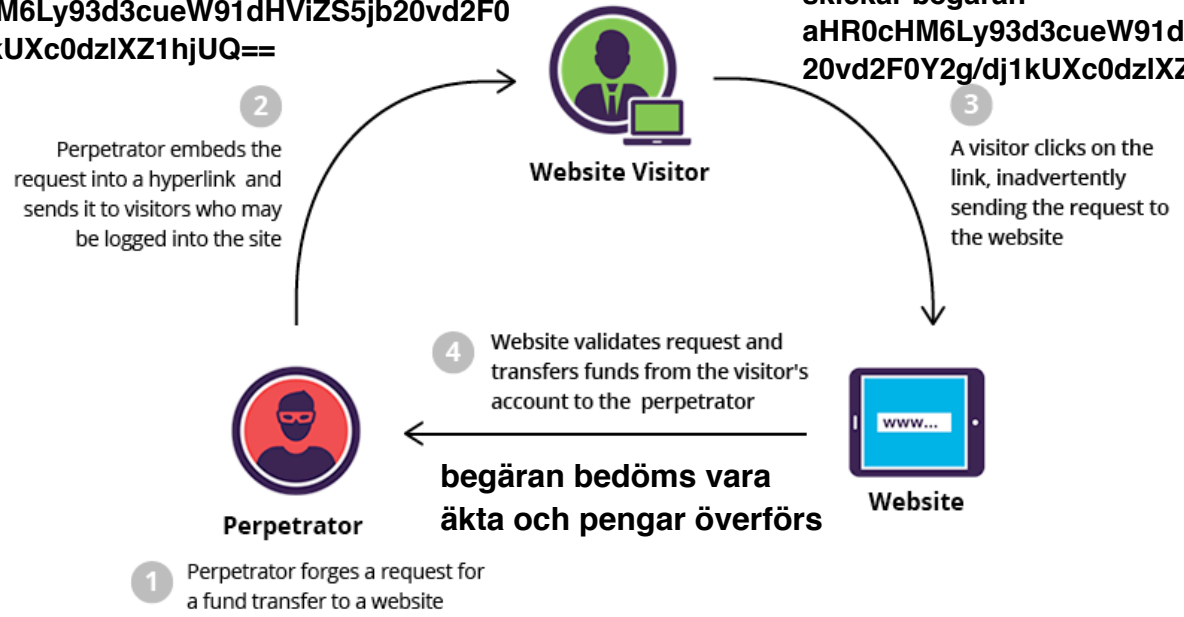
En CSRF-attack använder serverns förtroende hos klienten för att lura klienten att begära utförande av kommandon på servern med klientens rättigheter



Cross-Site Request Forgery (CSRF)

E-post: Besök snälla hemsidan LÄNK + script aHR0cHM6Ly93d3cueW91dHViZS5jb20vd2F0Y2g/dj1kUXc0dzlXZ1hjUQ==

Klick på länken!
Scriptet körs (med användarens rättigheter) utan att det märks och skickar begäran aHR0cHM6Ly93d3cueW91dHViZS5jb20vd2F0Y2g/dj1kUXc0dzlXZ1hjUQ==



Vanliga CSRF-scenarier

Tillagda sidor som utför (oönskade) gärningar å klientens vägnar

En oärlig handlare som använder PayPal kan normalt inte se kreditkortsnummer hos användaren

Handlaren låter kunden logga in på PayPal, och re-autenticerar sedan sig själv

Om kunden nu för in ett kreditkortsnummer så kan handlaren se det.

CSRF: Exempel

Vi vill skapa en websida där autenticerade användare kan rösta:

```
http://mysite.com/vote/25
```

Problem: En angripare lägger en länk i en bildfil: `<img src="http://mysite.com/vote/30" />`

Användare som öppar denna HTML-kod röstar nu på alternativ 30!

CSRF i labbarna

I labbarna kommer ni att göra CSRF mot varandra.

Det finns en räknare som ni skall räkna upp....

...men du får inte öka din egen räknare.

I stället skall du göra en attackkod mot kurskamrater. När de besöker websidan körs din kod, som ökar din räknare.

Försvar mot CSRF

Din röst kan bara göras från din session, som har ditt ID-nummer

När du skapar en sida

Skapa ett unikt element (token)
Lagra det i användarens session
Placera det i länkarna från sidan:

```
http://mysite.com/vote/30?token=AZERTYUHQNWGST
```

När röstning sker

- testa om elementet finns på URLen
- testa om det finns i användarens session

Om inte, räkna inte rösten.

CSRF-försvar, grundprincip

Tokens har kort livslängd och är svåra att gissa

Detta ger angriparen ett fönster på några få minuter för att kodinjektionen skall vara giltig

Angriparen måste gissa bra!

Angriparen måste skapa unika websidor för varje användare!

Validera osäkra data

Allt som skickas till servern kan manipuleras!

Man kan inte lita på webläsaren!

Dålig validering: Exempel

En webtillämpning tillåter enbart siffror att skrivas

Backend-programmet kraschar om den får bokstäver

JavaScript-validering säkrar korrekt format

Validera osäkra data

Allt som skickas till servern kan manipuleras!

Man kan inte lita på webläsaren!

Dålig validering: Exempel

En webtillämpning tillåter enbart siffror att skrivas

Backend-programmet kraschar om den får bokstäver

JavaScript-validering säkrar korrekt format ...hoppas vi.

Angriparen kan antingen stänga av JavaScript eller använda en *proxy*.

Lät det komplicerat?

I "lessons" i labbsystemet finns exempel på många av dessa attacker inklusive CSRF och XSS.

Gör dem för att få en känsla för hur varje attacktyp kan fungera.