

TSIT01 Datasäkerhetsmetoder

Föreläsning 3: Webtillämpningar, databaser, mer
webbsäkerhet, labbarna

Ingemar Ragnemalm

01001001 01000011 01000111

Förra gången

Lite mer om risk och skada: Att mäta säkerhet

Projekten

Exempel på säkerhetsanalys

Denna föreläsning

Webbtillämpningar

Databaser och annan webbsäkerhet

Labbarna

01001001 01000011 01000111

Att mäta säkerhet

Ett ständigt uppskattande!

Uppskatta sannolikheten för en attack

Uppskatta/mät skadan

Uppskatta effekten av åtgärder

Beräkna kostnaden av åtgärden

Bedöm vilka åtgärder som har bäst effekt, kostnad för åtgärd relativt skadan

Prioritera!

01001001 01000011 01000111

Projekten

Analysera ett av de givna fallen

Finn hot, svagheter, skada, hotagenter

Riskkostnad = skada * sannolikhet

Finn åtgärder, bedöm effekten

Baserat på riskmåttan ovan, prioritera!

01001001 01000011 01000111

Exemplen

Två exempel. Jag benade ut dem till en skiss på sekvens av analys.

Vad var det som var viktigt? Metodisk genomgång, numeriska värden att jämföra för att prioritera.

BILD BORTTAGEN

01001001 01000011 01000111

Projekten

Kan jag börja direkt? Ja, det kan du. Tycker du att jag varit tydlig så är det utmärkt.

Många attackmetoder och säkerhetsmetoder kommer senare men du kan komplettera om de är relevanta.

Hur hittar jag en projektpartner? Förslag: Samlas här framme i pausen!

01001001 01000011 01000111

Föreläsningsfrågor

1. Vad för webläsesanrop syns i adressraden?
2. Vad är det som lagras för en session?
3. Hur gör man databasangrepp och hur hindras de?
4. Vad är ofta principen, attackmetoden, för XSS och CSRF?
5. Hur löste jag "Broken session management"?
6. Vad skall man med en proxy till?

Webtillämpningar

01001001 01000011 01000111

Webtillämpningar

Webben brukade vara så mycket enklare:

Mest statiska HTML-sidor, mest en fråga om öppen information, indata genom enkla formulär

Idag är det mycket annorlunda:

- Dynamiska websidor
- Webtillämpningar
- Accesskontroll

01001001 01000011 01000111

HTTP: Grunden för websidor

Hyper-Text Transfer protocol (HTTP) är ett protokoll i tillämpningslagret som levererar data till websidor

Uppfanns i CERN 1989

Klienten skickar POST- och GET-anrop

Websidan tolkar anropet, extraherar parametrar från av användaren

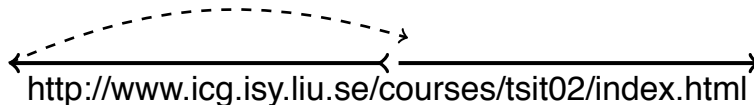
Servern svarar med status och sidan i HyperText Markup Language (HTML)

01001001 01000011 01000111

Uniform Resource Locator/Identifier/Namn

URL-delen (adressen) skickas genom DNS-uppslagning

URN (filnamnet) används för att identifiera filen (eller dataelementet) som önskas av klienten



POST och GET

Det finns två typer av begäran, POST och GET

GET lägger begäran synligt i URL-raden (mindre säkert, loggas och är lätt att manipulera)

POST lägger begäran i requestanropet, syns inte i anropsraden men kan avlyssnas och manipuleras

För bättre säkerhet bör SSL (https) användas så din begäran inte kan avlyssnas

01001001 01000011 01000111

Klientens GET-anrop

Enkelt exempel: Be om en HTML-sida:

Du skriver: `http://www.example.com/index.html`

Detta delas upp till:

`GET /index.html HTTP/1.1`

`Host: www.example.com`

01001001 01000011 01000111

Servrens svar

```
GET /index.html HTTP/1.1
Host: www.example.com
HTTP/1.1 200 OK
Date: Mon, 9 Nov 2016 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2016 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Accept-Ranges: bytes
Connection: close
```

```
<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

Klientens GET-anrop med PHP

PHP-exempel (från W3schools): Be om data från ett PHP-script

```
<?php  
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];  
?>
```

hämtar data och presenterar det som:

Study PHP at W3schools.com

men detta gör att anropet syns:

https://tryphp.w3schools.com/showphp.php?filename=demo_global_get

01001001 01000011 01000111

POST-anrop

Exempel: Mitt *quiz*-system.

Frågenumret måste hela tiden skickas vidare till nästa steg.
För detta använder jag POST:

```
echo '<form action = IngisQuizForm.php method="post">';  
echo '<input type="hidden" name="questionNumber" value="' .  
    $questionNumber . '">';  
echo '<input type="submit" value="Continue" class="button">';  
echo "</form>";
```

och hämtar ut det på nästa sida:

```
$questionNumber = $_POST['questionNumber'];
```

Detta syns *inte* i webadressraden! Därmed är det svårare att avlyssna.

HTML, hypertext markup language

HTML är ett markup language (textbaserat filformat) som beskriver websidor

Element inkluderar formulär, ramar (frames), iframes, bilder, applets och scripts

Kombineras ofta med JavaScript och/eller PHP för att skapa dynamiska websidor

Aktiveras ofta med musklick på en knapp som aktiverar ett GET- eller POST-anrop

Andra händelser är mouseup, onmouseover mm

01001001 01000011 01000111

Cookies

Cookies, "kakor", är små mängder data som sätts av serversvar med headerfältet "Set-Cookie"

Innehåller nyckel+värde-par, domän, utgångsdatum, eventuell sökväg, samt flaggorna "secure" och "HTTP only"

"secure" påtvingar HTTPS-överföring

"HTTP only" förbjuder script-access från klienten

BILD BORTTAGEN

01001001 01000011 01000111

Sessions och cookies

HTTP saknar tillstånd! All information måste överföras från sida till sida eller sparas någonstans!

Sessions: Information (som identitet) sparas i servern (begränsad tid)

Cookies: Information (identitet) sparas i klienten

När en session skapas skickar servern ett ID till klienten som sparas som en cookie

Autentisering av sessions är ett separat problem som kan utföras i olika lager av nätverkssystemet.

01001001 01000011 01000111

Angrepp via webbläsare

Databasangrepp

Cookiestöld

Scriptangrepp

Mer om detta nästa föreläsning!

01001001 01000011 01000111

Ett känt problembarn för säkerhet:

Databaser

01001001 01000011 01000111

Databaser

Webtillämpningar behöver någonstans att lagra data

Detta görs ofta med relationsdatabaser

ISBN_NO	SHORT_DESC	AUTHOR	PUBLISHER	PRICE
0201703092	The Practical SQL, Fourth Edition	Judith S. Bowman	Addison Wesley	39
0471777781	Professional Ajax	Jeremy McPeak, Joe Fawcett	Wrox	32
0672325764	Sams Teach Yourself XML in 21 Days, Third Edition	Steven Holzner	Sams Publishing	49
0764557599	Professional C#	Simon Robinson and Jay Glynn	Wrox	42
0764579088	Professional JavaScript for Web Developers	Nicholas C. Zakas	Wrox	35
1861002025	Professional Visual Basic 6 Databases	Charles Williams	Wrox	38
1861006314	GDI+ Programming: Creating Custom Controls Using C#	Eric White	Wrox	29

Diagram illustrating the structure of a relational database table. The table has 5 columns (ISBN_NO, SHORT_DESC, AUTHOR, PUBLISHER, PRICE) and 7 rows of data. Arrows point from the labels "Columns" and "Rows" to the respective parts of the table.

01001001 01000011 01000111

SQL-språket

Databaser accessas normalt med SQL (*Structured Query Language*)

Exempel:

```
SELECT * from users WHERE userName='admin';
```

Detta returnerar alla rader i databasen `users` som matchar förfrågan.

Alla SQL-förfrågningar avslutas med semikolon.

SQL-servrar tillåter flera rader i en enda förfrågan, uppdelade med semikolon.

01001001 01000011 01000111

SELECT på urval av kolumner

Du kan också fråga efter vissa kolumner.

```
SELECT CustomerName, City from Customers  
WHERE OrderID='1045';
```

Detta visar enbart CustomerName och City

Jokertecknet (wildcard) * låter dig välja alla.

DROP och UNION

SQL-språket innehåller fler direktiv

Till exempel DROP DATABASE users; som raderar en databas vid namn users.

Ledtråd inför labben: Det finns ett SQL-kommando som heter UNION ALL som kan kombinera flera SELECT-kommandon

```
SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;
```

Databasangrepp

01001001 01000011 01000111

SQL-injektion

Antag att en webbtillämpning använder en SQL-databas för att visa ordrar

Användaren anger objektnummer i ett formulär

Webbtillämpningen tar detta ID och bygger följande SQL-query:

```
SELECT * FROM orders WHERE itemID='ID';
```

Detta returnerar alla rader med detta ID.

SQL-injektion

Nu lägger angriparen in följande ID:

1 ' OR 'a'='a

Detta resulterar i följande SQL-query:

```
SELECT * FROM orders WHERE itemID='1' OR 'a'='a';
```

Observera OR-uttrycket. Eftersom 'a'='a' alltid är sann, detta returnerar *alla* rader!

SQL-injektion: Avancerade attacker

En angripare kan använda SQL-kommentarer för mer avancerade attacker.

Två bindestreck är en kommentar: --

Antag att detta är normalt:

```
SELECT MyRecord FROM MyTable WHERE  
MyEmail='$email' AND MyPassword='foo';
```

SQL-injektion: Avancerade attacker

Antag att detta är normalt:

```
SELECT MyRecord FROM MyTable WHERE  
MyEmail='$email' AND MyPassword='foo';
```

Med en falsk E-postadress:

```
SELECT MyRecord FROM MyTable WHERE MyEmail='';  
DROP TABLE MyTable; --' AND MyPassword='foo';
```

All data efter de två bindestrecken ignoreras!

Håll dina parametrar rena

BILD BORTTAGEN

Inte ett vanligt problem idag eftersom alla har sett denna!

01001001 01000011 01000111

Mer websäkerhet

01001001 01000011 01000111

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.

01001001 01000011 01000111

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.

01001001 01000011 01000111

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

01001001 01000011 01000111

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)

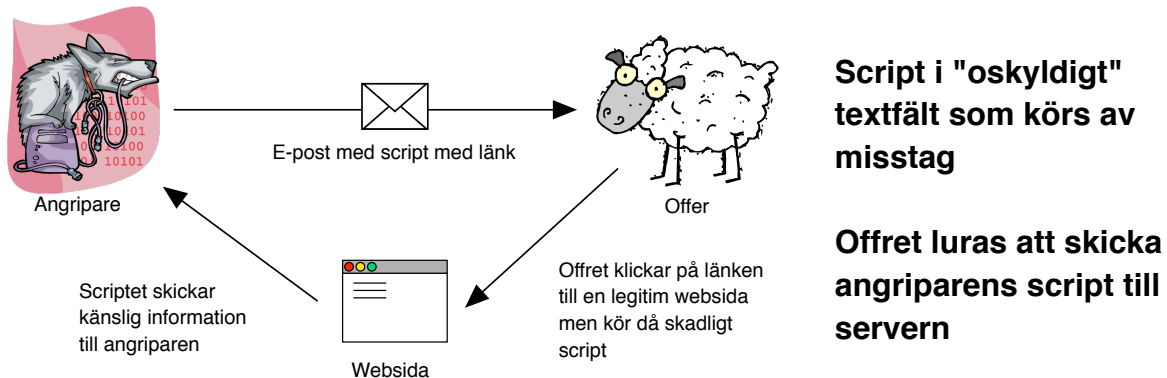
Tekniken kallas *cross-site scripting*

01001001 01000011 01000111

Cross-site scripting, XSS

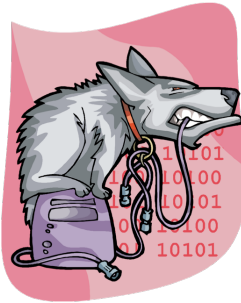
Detta är en typ av tekniker som används för att få angriparens scripts inkluderade i sidor från betrodda servrar

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

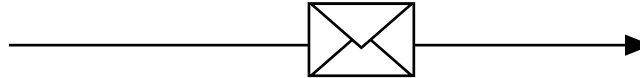


01001001 01000011 01000111

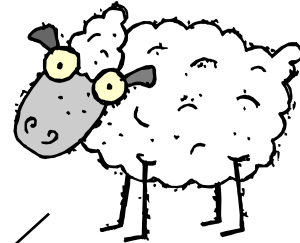
E-post: Besök snälla hemsidan LÄNK + script
aHR0cHM6Ly93d3cueW91dHViZS5jb20vd2F0Y
2g/dj1kUXc0dzlXZ1hjUQ==



Angripare

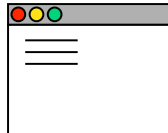


E-post med script med länk



Offer

scriptet skickar data
som fiskades ut med
användarens rättigheter



Websida

Klart jag går till
snälla hemsidan!

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

01001001 01000011 01000111

Cross-site scripting, XSS

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

Min webbläsare vägrar öppna sådana länkar av säkerhetsskäl.

Gissa vilka websidor som gör sådant hela tiden?

01001001 01000011 01000111

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

Angriparen skriver
in ett script i ett
textfält



Det sparas inte men
ekas på sidan och
exekveras då med
serverns rättigheter



Data fiskas ut och
returneras

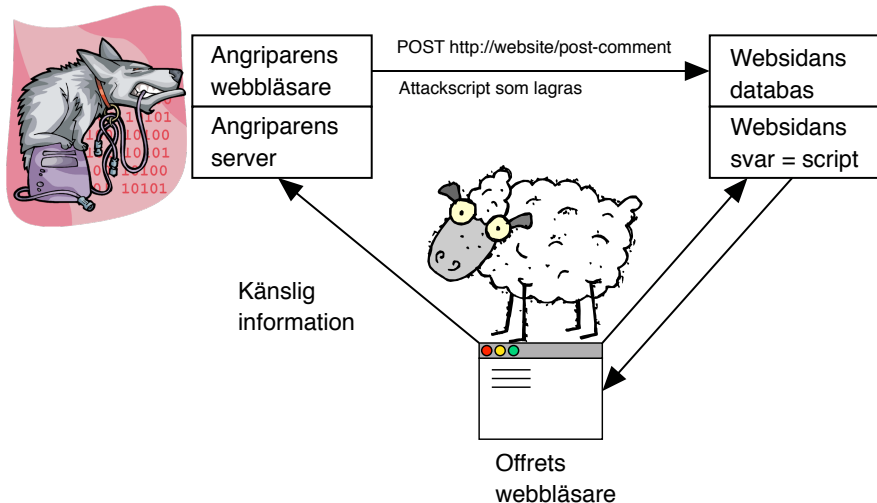
Attack mot serverns
information!

01001001 01000011 01000111

Lagrade (persistent) XSS

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

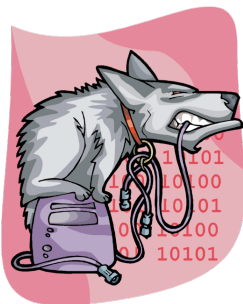
I en lagrad XSS-attack sparar angriparen sitt script på den betrodda servern, till exempel på en diskussionssida.



01001001 01000011 01000111

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

Scriptet lagras som en kommentar. Offret öppnar sidan.



Angriparens
webbläsare

Angriparens
server

POST http://website/post-comment

Attackscript som lagras

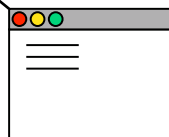
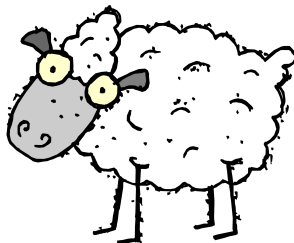
Websidans
databas

Websidans
svar = script

Scriptet skickar data till angriparen.

Offret visar kommentaren

varpå scriptet körs



Offrets
webbläsare

01001001 01000011 01000111

Försvar mot XSS

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

Rengör indata väl!

Förbättra autentiseringen

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

01001001 01000011 01000111

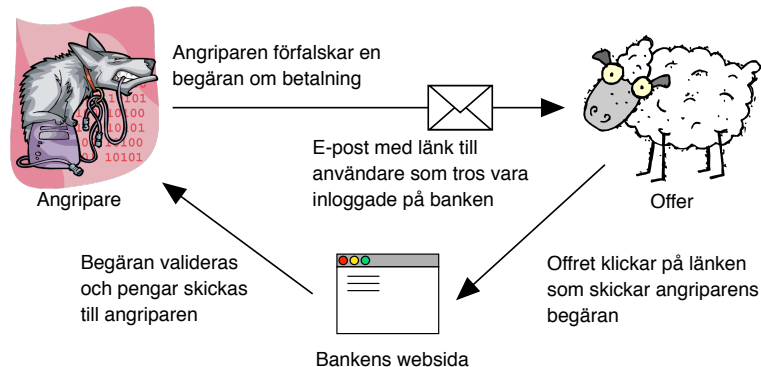
Cross-Site Request Forgery (CSRF)

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

Motsatsen till en XSS-attack

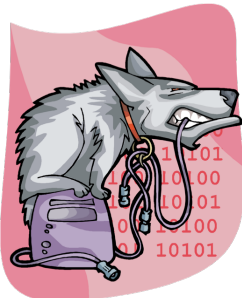
En XSS-attack använder klientens förtroende för att exekvera kommandon hos klienten med serverns rättigheter

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



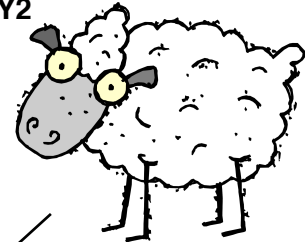
01001001 01000011 01000111

Cross-Site Request Forgery (CSRF)



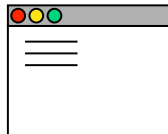
Angripare

E-post: Besök snälla hemsidan LÄNK + script
aHR0cHM6Ly93d3cueW91dHVhZS5jb20vd2F0Y2
g/dj1kUXc0dzlXZ1hjUQ==



Offer

begäran bedöms vara
äkta och pengar överförs



Bankens websida

Klick på länken!
Scriptet körs (med användarens
rättigheter) utan att det märks och
skickar begäran
aHR0cHM6Ly93d3cueW91dHVhZS5jb20vd2F0Y2g/dj1kUXc0dzlXZ1hjUQ==

01001001 01000011 01000111

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.

01001001 01000011 01000111

CSRF: Exempel

Vi vill skapa en websida där autentiserade användare kan rösta:

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

Problem: En angripare lägger en länk i en bildfil: ``

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.

01001001 01000011 01000111

Försvar mot CSRF

När du skapar en sida

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

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.

01001001 01000011 01000111

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!

01001001 01000011 01000111

Validera osäkra data

Allt som skickas till servern kan manipuleras!

Man kan inte lita på webbläsaren!

Dålig validering: Exempel

En webbtillämpning tillåter enbart siffror att skrivas

Backend-programmet kraschar om den får bokstäver

JavaScript-validering säkrar korrekt format

01001001 01000011 01000111

Validera osäkra data

Allt som skickas till servern kan manipuleras!

Man kan inte lita på webbläsaren!

Dålig validering: Exempel

En webbtillämpning 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*.

01001001 01000011 01000111

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.

LABBARNA

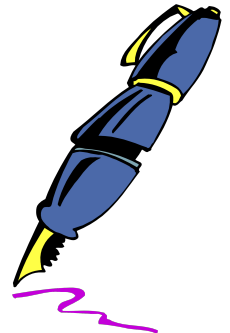
01001001 01000011 01000111

Pentesting

Websäkerhet testas med *penetration testing*, "pentesting".

Detta är metoder för att testa möjligheten, och därmed också risken, för intrång.

Fokus för labbarna!



"Pentesting" har inget med pennor att göra.

01001001 01000011 01000111

Labborganisation

Labbuppgifterna är obligatoriska och utförs individuellt

Du utför dem på egen tid innan deadline. Du loggar in på vår server med ditt LiU-ID.

Vi har tider då du kan få hjälp. Använd dem väl om du behöver dem.

Du får godkänt när du har klarat de obligatoriska uppgifterna.

Server och PM

Servern är en typ som kallas "Security Shepherd", ett system för datasäkerhetsträning

Serveradressen för labbarna är *snickerboa.it.liu.se*

Lab-PM finns på kurssidan, med instruktioner, hur du loggar in mm. (Behöver uppdateras.)

Utmaningskategorier

- CSRF
- XSS
- SQL Injection
- Osäker krypterad lagring
- Osäkta direkta objektreferenser
- Dålig datavalidering

01001001 01000011 01000111

Lektioner och labbuppgifter

För varje kategori finns en lektion och ett antal labbuppgifter

Lektionerna introducerar ämnet

Använd lektionen för att lära dig hur just denna mekanism fungerar

Lektionerna kan också ge ledtrådar efter några misslyckade försök

När du känner dig säker på ett ämne, fortsätt med uppgifterna

Du får godkänt efter att ha fullbordat samtliga 21 uppgifter

01001001 01000011 01000111

Webgränssnitt för Security Shepherd

TOPDOG CHALLENGE

Guilherme B Xavier | Logout

Admin

Scoreboard

Lessons

Assignments (0/21)

CSRF

Failure to Restrict URL Access

Injection

Insecure Cryptographic

Storage

Insecure Direct Object

References

Poor Data Validation

Session Management

XSS

Challenges

Search Modules...

You have the option to change your username here. This username will be shown publicly on the scoreboard.

New username:

Change Username

01001001 01000011 01000111

62(80)

Målet är att få ut resultatnyckeln

TOPDOG CHALLENGE

Guilherme B Xavier | Logout

Admin

Scoreboard

Cheat

Lessons

XBroken Session Management
XCross Site Request Forgery
XCross Site Scripting
XFailure to Restrict URL Access
XInsecure Cryptographic Storage
XInsecure Direct Object References
XPoor Data Validation
XSQL Injection
XUnvalidated Redirects and Forwards

Assignments (0/21)

Challenges

Search Modules...

Submit Result Key Here...

Submit

What is Cross Site Scripting (XSS)?

Cross-Site Scripting, or XSS, issues occur when an application uses untrusted data in a web browser without sufficient validation or escaping. If untrusted data contains a client side script, the browser will execute the script while it is interpreting the page.

Attackers can use XSS attacks to execute scripts in a victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. Anyone that can send data to the system, including administrators, are possible candidates for performing XSS attacks in an application.

According to OWASP, XSS is the most widespread vulnerability found in web applications today. This is partially due to the variety of attack vectors that are available. The easiest way of showing an XSS attack executing is using a simple alert box as a client side script pay load. To execute a XSS payload, a variety of an attack vectors may be necessary to overcome insufficient escaping or validation. The following are examples of some known attack vectors, that all create the same alert pop up that reads "XSS".

For more information please visit OWASP Guide to XSS

```
<script>alert("XSS")</script>

<input type="button" onclick="alert('XSS')" />
<iframe src="javascript:alert('XSS');"></iframe>
```

Hide Lesson Introduction

The following search box outputs untrusted data without any validation or escaping. Get an alert box to execute through this function to show that there is an XSS vulnerability present.

Please enter the Search Term that you want to look up

Get This User

01001001 01000011 01000111

63(80)

Hur resultatnyckeln fungerar

När en uppgift är klar får du resultatnyckeln (result key)

Klistra in den i boxen överst och klicka på submit

Oftast (men inte alltid) ser nyckeln ut något i stil med:

`3c17f6bf34080979e0cebda5672e90...`

Resultatnyckeln är unik för varje användare och varje modul

Varning: Du får poängavdrag om du försöker med brute force-lösningar. Detta syns i våra adminloggar.

01001001 01000011 01000111

Tillgängliga verktyg

Pentesting kräver ett antal verktyg

Du kan använda onlinekalkylatorer, som base64-encoders/decoders (Googla)

En sidas källkod kan avslöja mycket.

Webläsare har utvecklarverktyg.

Du behöver också en attackproxy som låter dig modifiera HTTP-data skickat mellan server och klient.

01001001 01000011 01000111

Websidans källkod

Firefox: Högerklicka/ctrl-klicka websidan och välj "View page source"

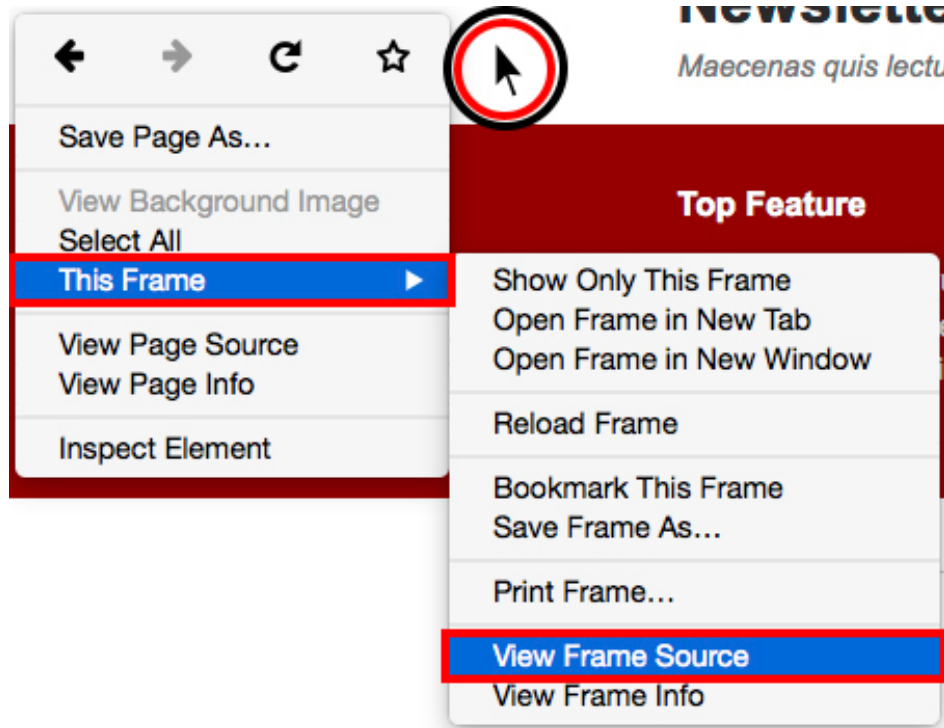
Detta visar källkoden för nuvarande sida.

I Security Shepherd behöver du se källkoden för uppgiften, dess iFrame.

Annars ser du bara källkoden för Security Shepherd.

01001001 01000011 01000111

Visa källkod för en frame



01001001 01000011 01000111

Exempel på källkod för websida

```
<h2 class="title">Failure To Restrict URL Access Challenge 1</h2>
<p>
  To recover the result key for this challenge you need to obtain the current server status message from an administrator
<br/>
  Use this form to view the status of the server <!-- from the point of view of a peasant or guest -->
<br/>
  <form id="leForm" action="javascript:;">
    <table>
      <tr><td>
        <div id="submitButton">
          <input type="submit" value="Get Server Status"/></div>
          <p style="display: none;" id="loadingSign">Loading</p>
          <div style="display: none;" id="hintButton"><input type="button" value="Would you like a hint?" id="theHint" /></div>
        </td></tr>
      </table>
    </form>

    <div id="resultsDiv"></div>
  </p>
```

Figur: Källkoden för en av utmaningarna

01001001 01000011 01000111

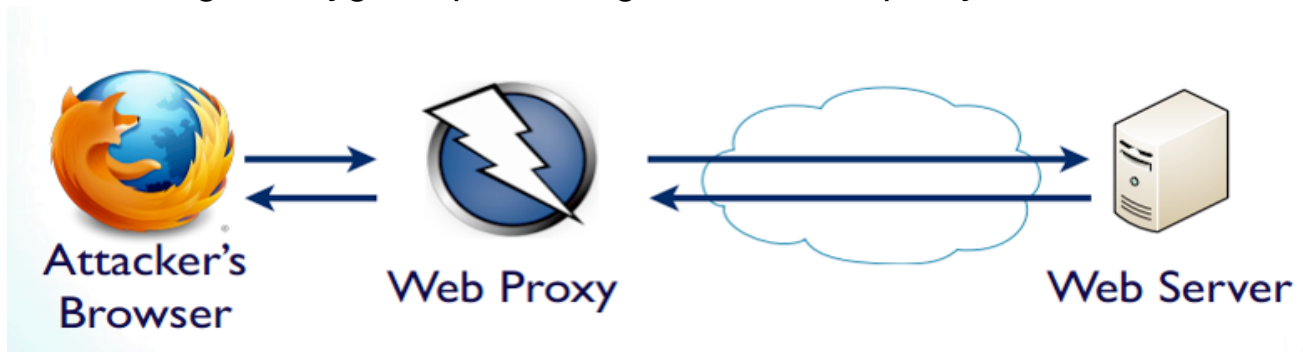
Proxy

En proxy är ett mellanlager, en server som ligger mellan användaren och den server användaren accessar.

En proxy kan göra olika saker, anonymitet, säkerhet, konvertering av data (t.ex. översättning)...

Använda en proxy

Viktigt verktyg för "pentesting" är en attack proxy



En proxy kan modifiera data som skickas genom den!

Du kan använda en proxy för att "pausa" en begäran medan du ändrar det.

01001001 01000011 01000111

ZAP-proxyn

Vi rekommenderar ZAP-proxyn

Finns för Linux, Window och OSX

Open source

Ladda ner här:

<https://www.zaproxy.org/>

Att använda proxyn

När du installerar ZAP så är dess standardvärde localhost:8080

Konfigurera webbläsaren så den skickar alla HTTP-data genom den

Detta skickar all trafik genom proxyn, vilket är irriterande.

Installera en andra webbläsare som kör genom proxyn. Då kan du köra din vanliga webbläsare som vanligt.

Du kan till exempel ladda ner Firefox eller Chrome, som finns för de flesta plattformar. Eller Firefox Developer!

01001001 01000011 01000111

Var inte vårdslös!

Attackera bara uppgifterna!

Du får inte manipulera Shepherd! Försök att göra det detekteras och rapporteras som fusk! Respektive uppgifts egen ram (iFrame) kan du dock gärna öppna och undersöka.

Din uppgift är att knäcka uppgifterna, inte att fuska dig genom labben!

Labkursen lyder under samma regler som andra kurser och vi har order att anmäla fusk till disciplinnämnden!

01001001 01000011 01000111

Du kan jobba på egen tid

Labbarna är öppna från 15/11.

Du kan göra dem från godtycklig dator.

Det betyder, från din egen dator, kan vara hemma, eller från labbdatorer, när det passar dig.

MEN du måste vara klar senast under tentaperioden i januari!

01001001 01000011 01000111

Fullborda labben

Inga labbrapporter

När du har klarat en uppgift så **registrerar systemet detta automatiskt**. Mer instruktioner kommer i PM när labbarna startar.

01001001 01000011 01000111

Fullborda labben

Inga labbrapporter

När du har klarat en uppgift så **registrerar systemet detta automatiskt**. Mer instruktioner kommer i PM när labbarna startar.

Förutom detta finns en publik topplista!

01001001 01000011 01000111

Topplista och bonuspoäng

Vi ser fortfarande vilka uppgifter du har klarat, och labben är godkänd när alla uppgifter är fullbordade

Topplistan har hittills inte varit relaterad till den övriga examinationen... men jag överväger starkt att införa det nu när jag har ett poängsystem!

Topplistan är bara en kul grej, för att utmana varandra.

Alla kurser (TSIT01, TSIT02, 726G81) har gemensam topplista

Tävlingsdelen avslutas 24/11 om inget annat meddelas.

01001001 01000011 01000111

Bra metoder

SKRIV NER dina framsteg. När servern lagrar dina resultat så finns alltid risken att databasen kraschar på vägen.

Då kan du snabbt komma tillbaka där du var.

Jo, vi har också backuper, men ha en egen för säkerhets skull.

Dubbla system är bra för säkerheten!

01001001 01000011 01000111

DEMO

01001001 01000011 01000111

79(80)

Slutord

Vi hoppas att detta blir en kul och intressant upplevelse.

Nej, labben är inte speciellt lätt

men du har tid på dig att förstå och klara av den i tid.

01001001 01000011 01000111