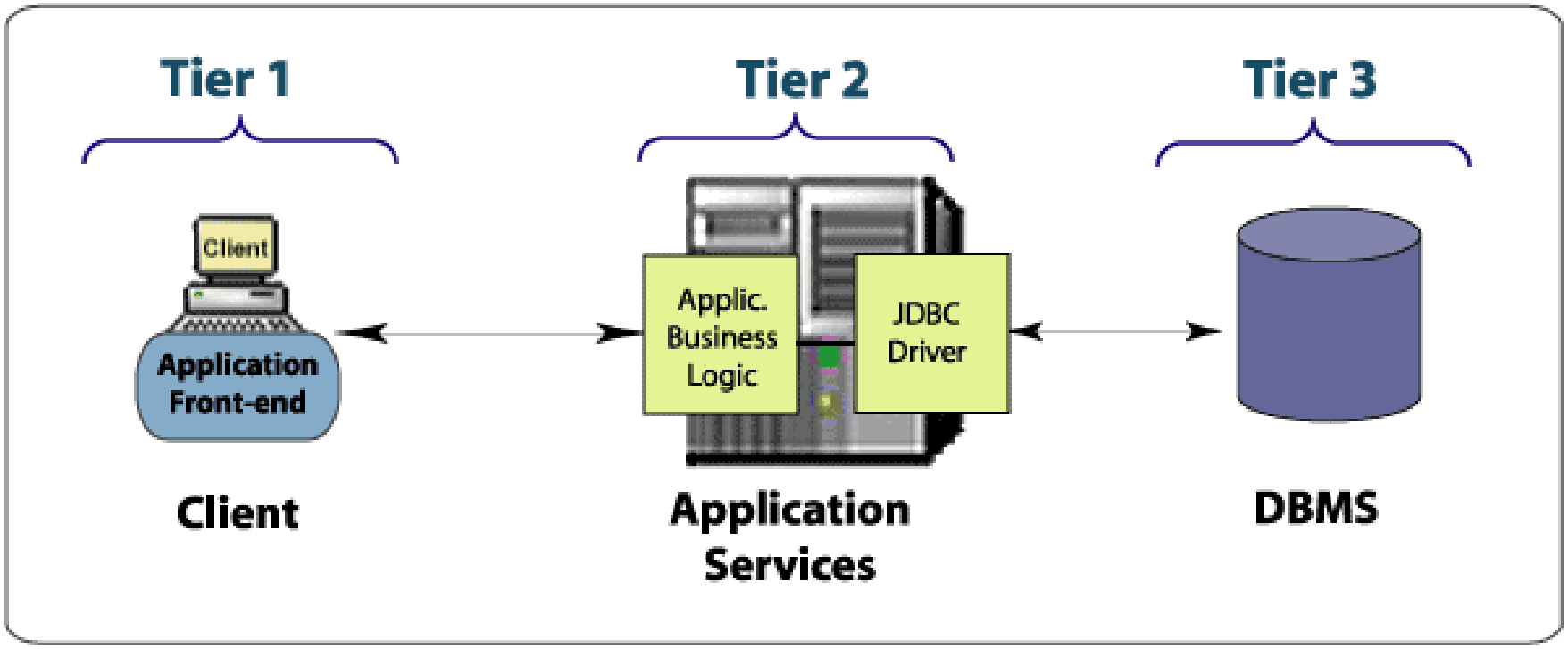
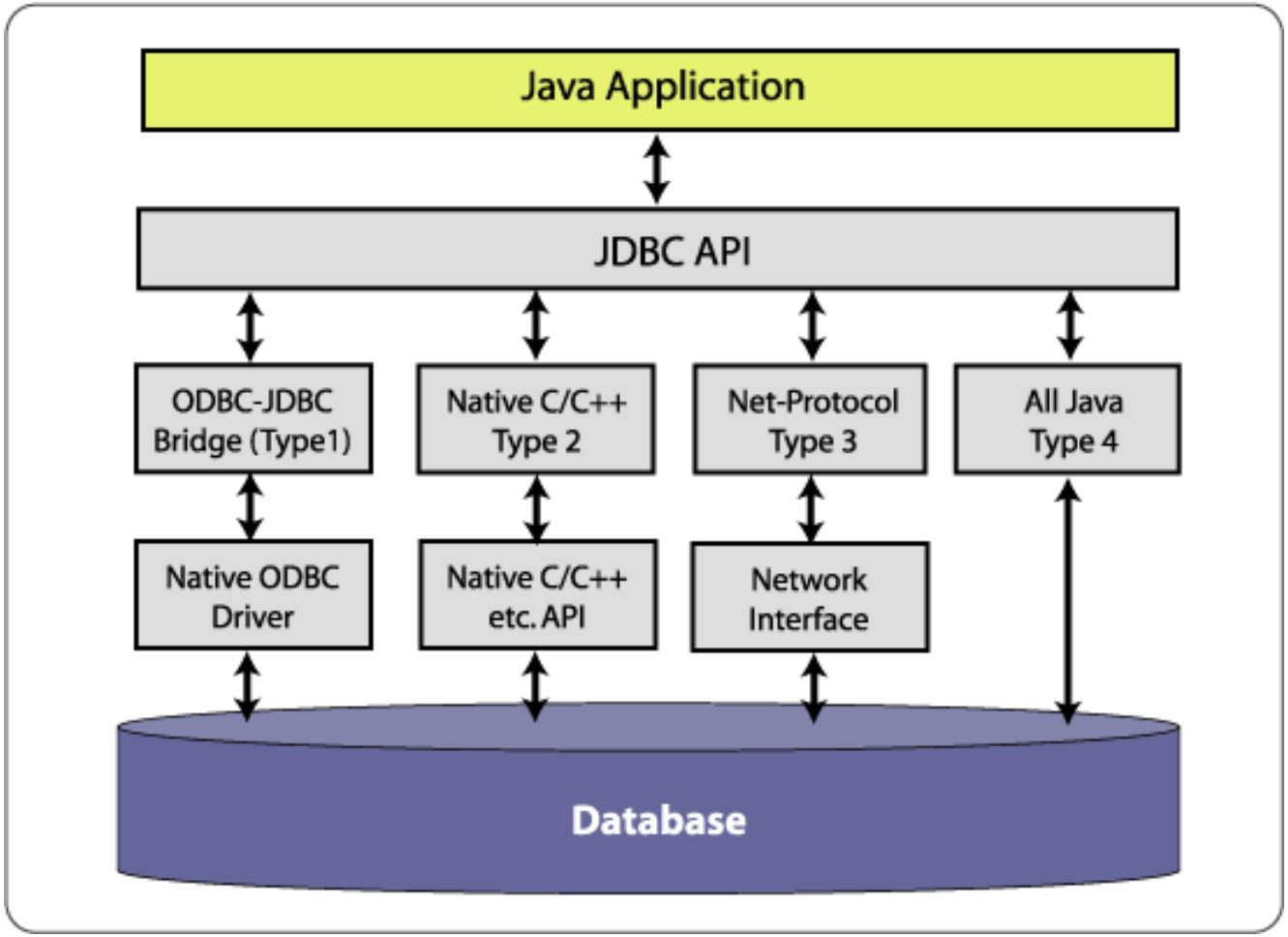
1. Database Connection Pooling

JDBC 3-Tier Model



JDBC Drivers



Long or short lived connections?

- Problems
  - It takes a relatively long time to open / close a connection
    - Naive solution: Keep the connection open "forever"
  - A DBMS can have *N* open connections
    - Naive solutions: Close connections as soon as possible
  - Naive solutions are conflicting
  - Combined solution
    - Connection pool!

Connection pool

- Ideas
  - The application (server) allocates a pool of connections to the database (e.g. 10 connections)
  - Applications programmers don't create connections, but borrows a connection the from the pool.
- Advantages
  - Connections are "recycled"
  - Few physical connections
- Implemented by driver
  - implemented by application programmer

```
var mysql = require('mysql');
var connection = mysql.createConnection({
   host : 'localhost',
   user : 'me', password : 'secret' });

connection.connect();
connection.query('SELECT name from student
where id = 11111 AS names', function(err,
rows, fields) {
if (err) throw err;
console.log('The student is: ', rows[0].names);
});

connection.end();
```

- var mysql = require('mysql');
  var pool = mysql.createPool(...);

  pool.getConnection(function(err, connection) {
  // Use the connection
  connection.query('SELECT name FROM student', function(err, rows) {
  // finish with the connection.
  connection.release();
  // the connection has been returned to the pool.
  }); });

Connection Pool Manager Pseudo Code

```
Class ConnectionManager() {
ArrayList availconn; <datastructure to keep a list of available connections>
<constructor of ConnectionManager>
Constructor() {

For(‘ MAX Pool Size’ times)
{
<Create a new connection and store it in some Data structure>
DriverManager.getConnection(url, user, pw);
availconn.add(new Connection( ) ) ;
}
}
```

Manage Function

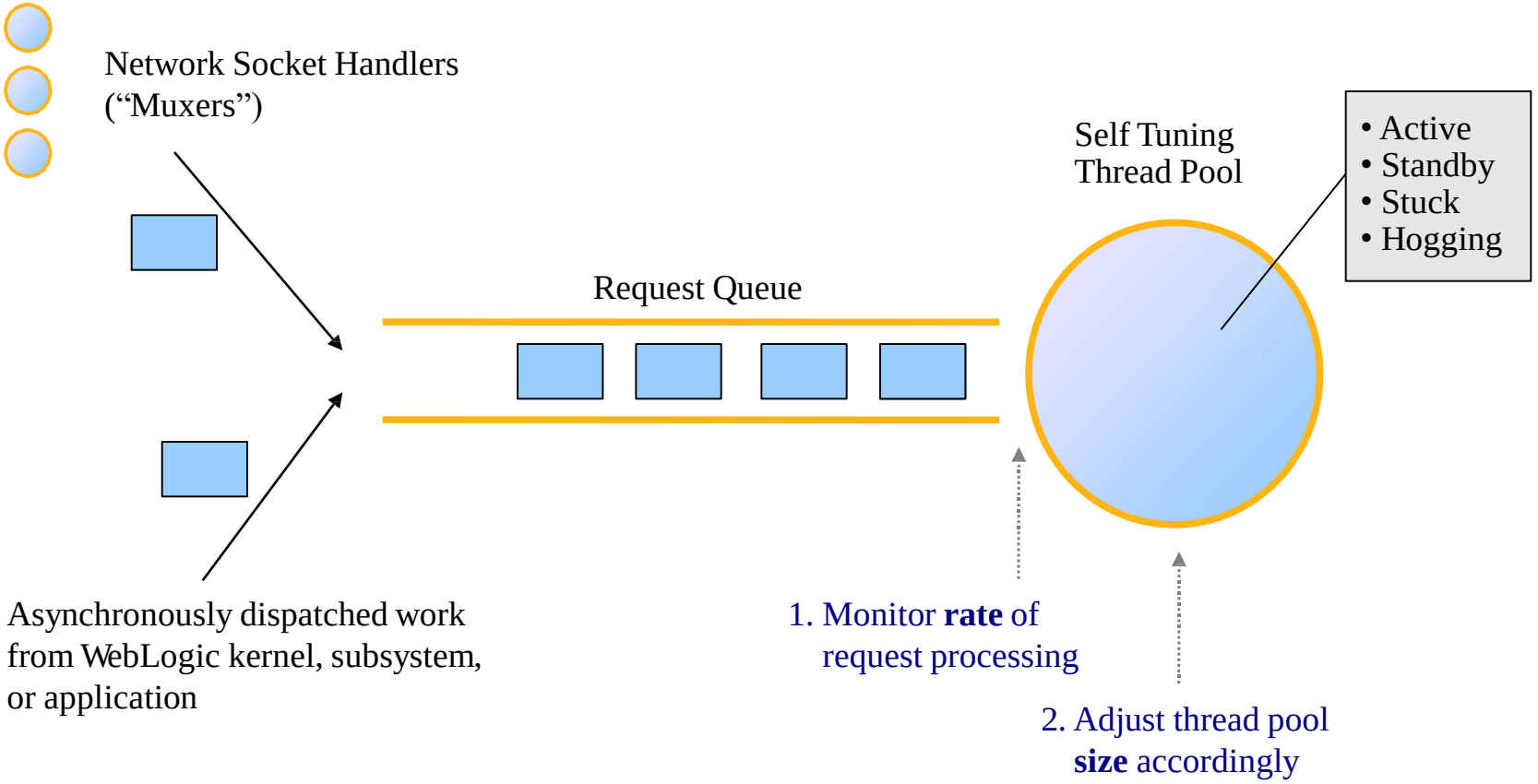
< keep track of the number of connections >
ConnectionManager connection = availconn.get(CURRENTINDEXOFAVAILCONN) ;

Use it here …

availconn.close(connection) // return connection back to the pool when done

Self-Tuning and Work Managers

WebLogic's Self-Tuning Thread Pool



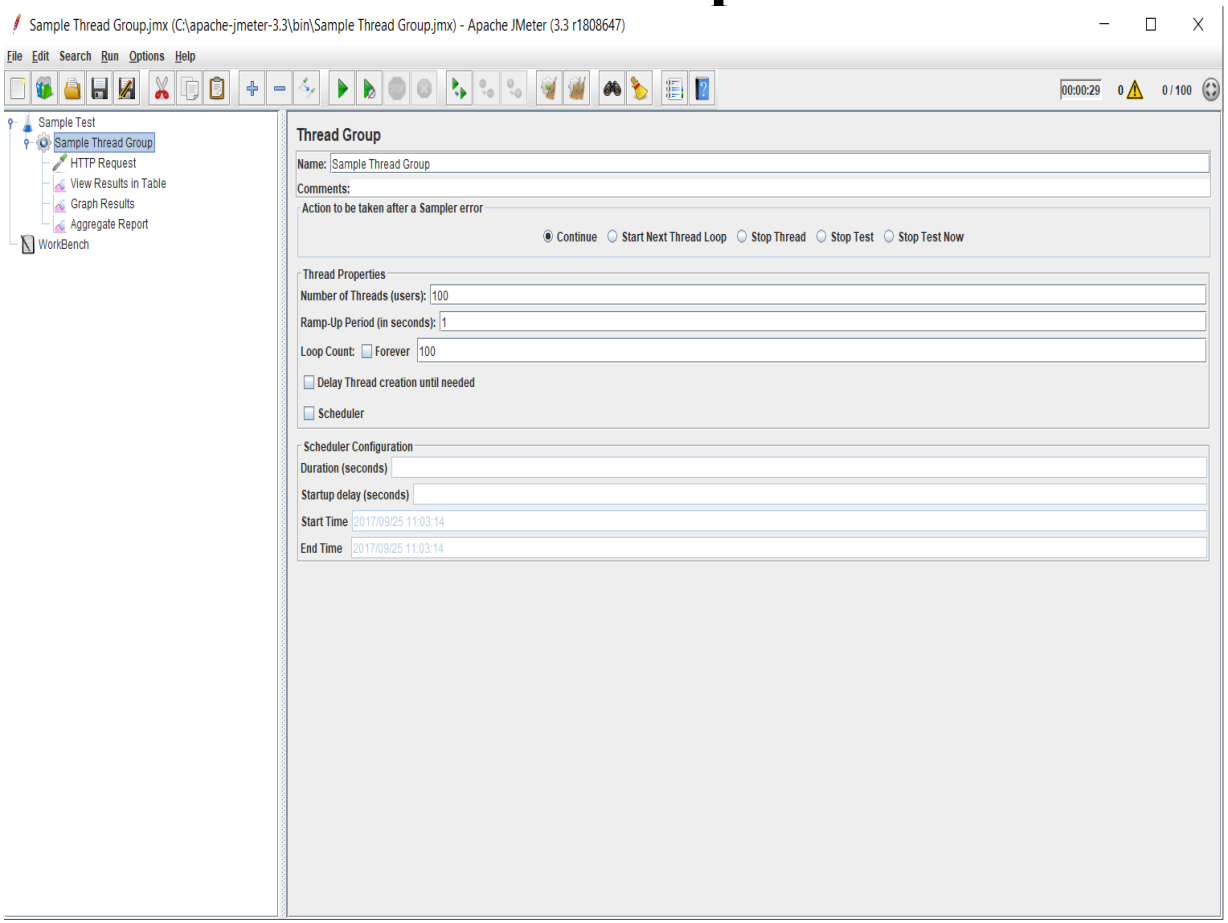
JMeter

How to install?

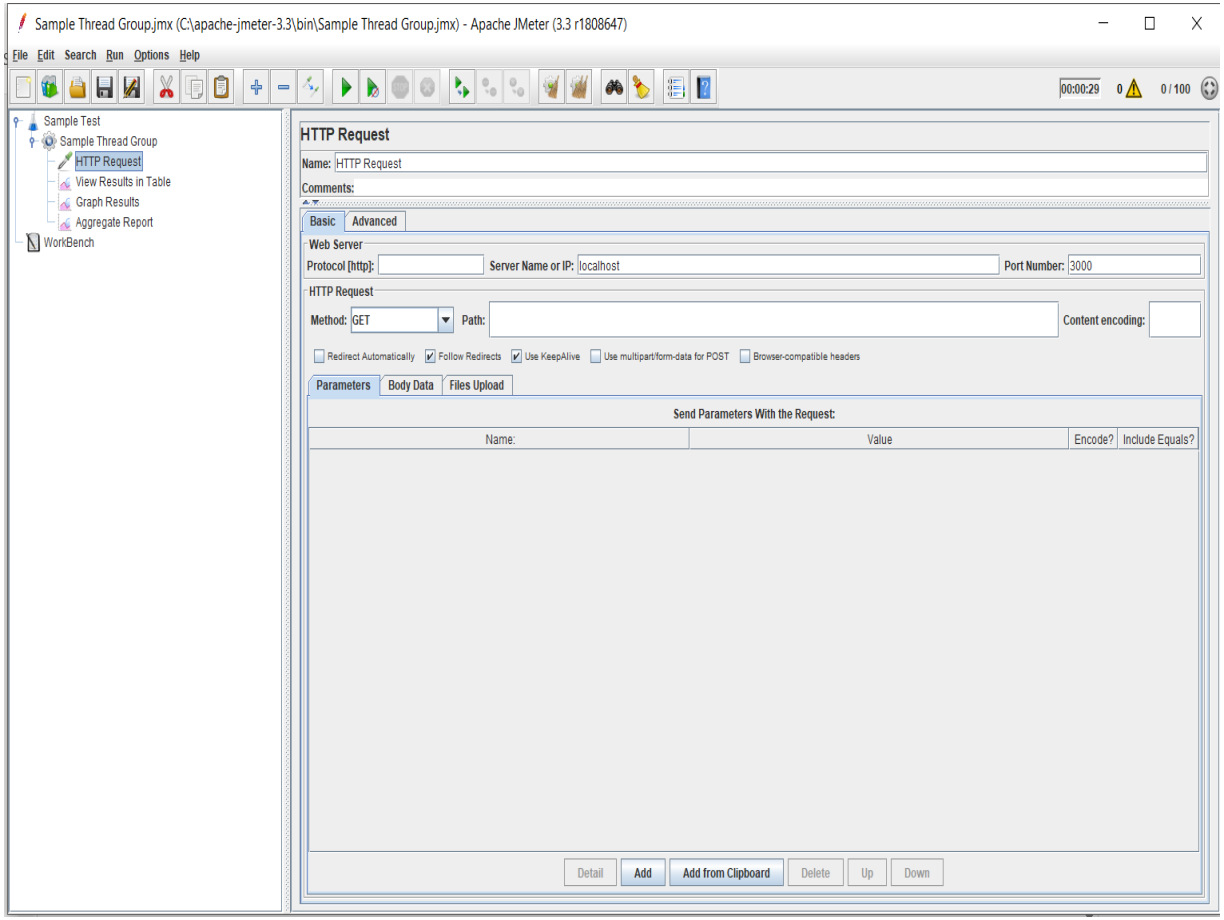
- Download the latest version of JMeter from http://jmeter.apache.org/download_jmeter.cgi.
- Extract the zip/tar file
- Go to jmeter/bin folder and run below file.

OS	Output
Windows	jmeter.bat
Linux	jmeter.sh
Mac	jmeter.sh

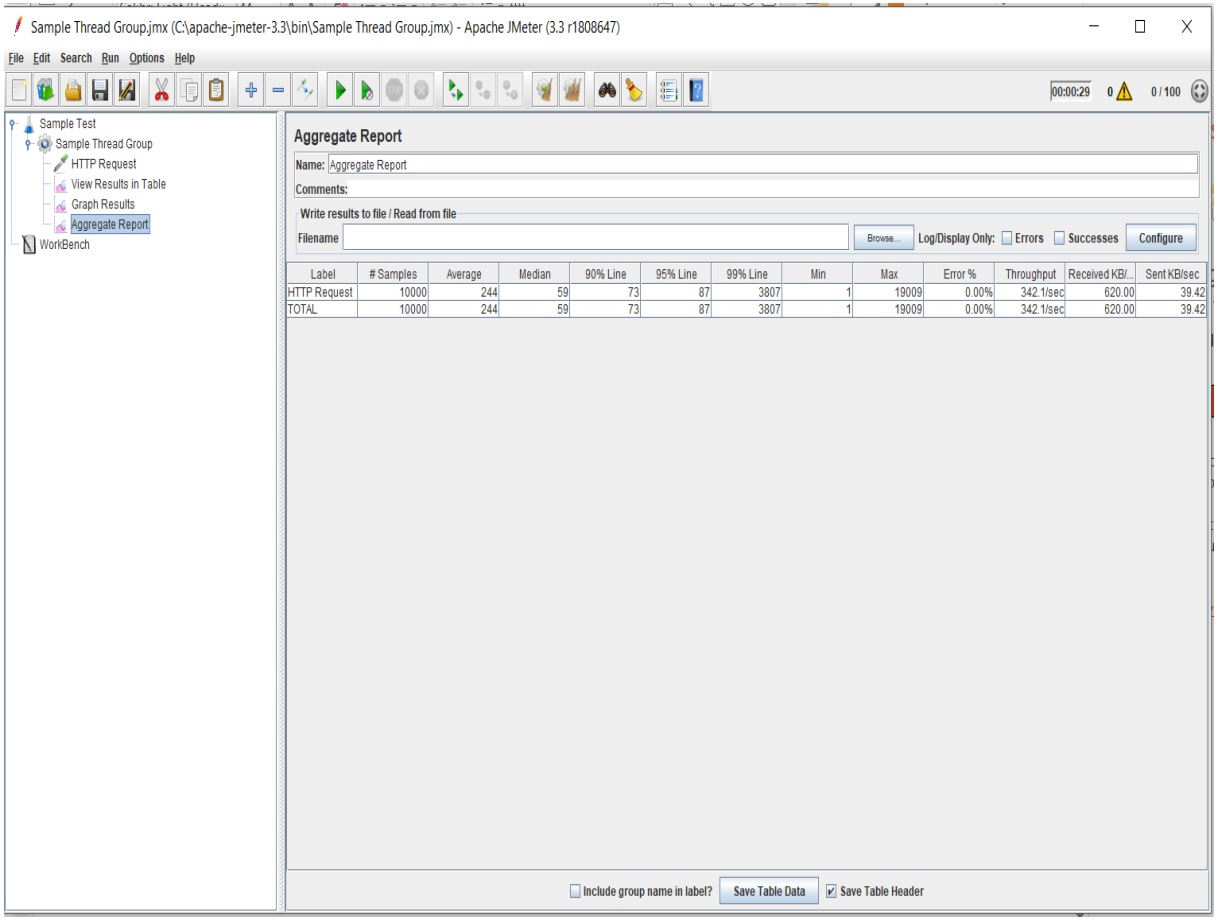
Sample Test Plan with Thread Group



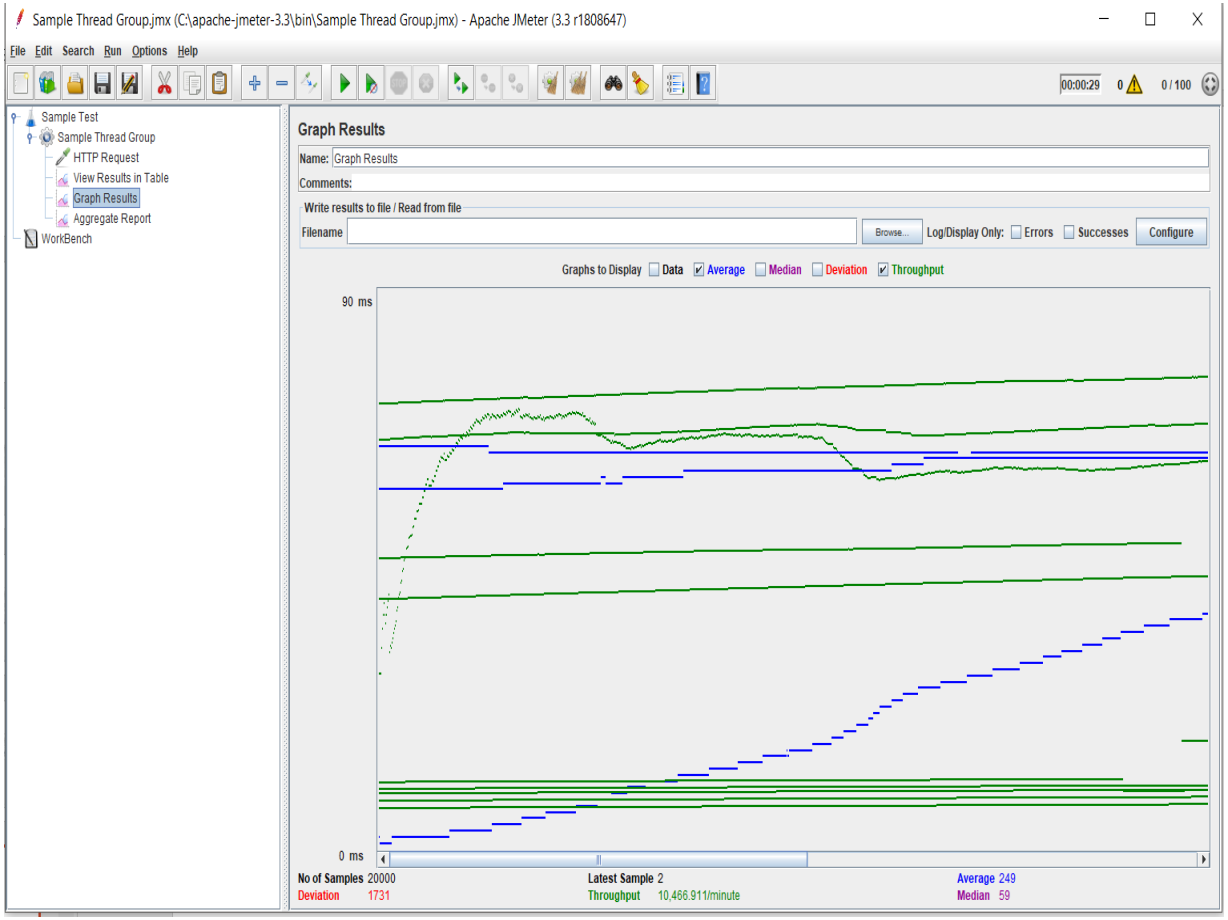
Sample HTTP Request



Sample Report



Sample Graph



Tutorial

- http://jmeter.apache.org/usermanual/
- http://www.tutorialspoint.com/jmeter/

Mocha

How to install?

- npm install mocha
- mkdir test
- Create test.js inside test folder
- Write you test code in js
- Add “test”:”mocha” in scripts in package.json
- Run using npm test

Sample Code

```
describe('http tests', function(){


        it('should return the login if the url is correct',
function(done){
                http.get('http://localhost:3000/', function(res) {
                        assert.equal(200, res.statusCode);
                        done();
                })
        });
});
```

Sample Output

```
D:\demos\MySQLConnection>npm test

> NodeExpress@0.0.1 test D:\demos\MySQLConnection
> mocha



  http tests
    √ should return the login if the url is correct
    √ should not return the home page if the url is wrong
    √ should login


  3 passing (42ms)
```

SQL Injection

- is a technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed.

CMPE 273

SQL Injection

- Username = ' or 1=1 --
  - The original statement looked like:
    'select * from users where username = '" +
    username + '" and password = '" + password + "'
    The result =
    select * from users where username = '' or 1=1 --'
    and password = ''

CMPE 273

Escaping query values

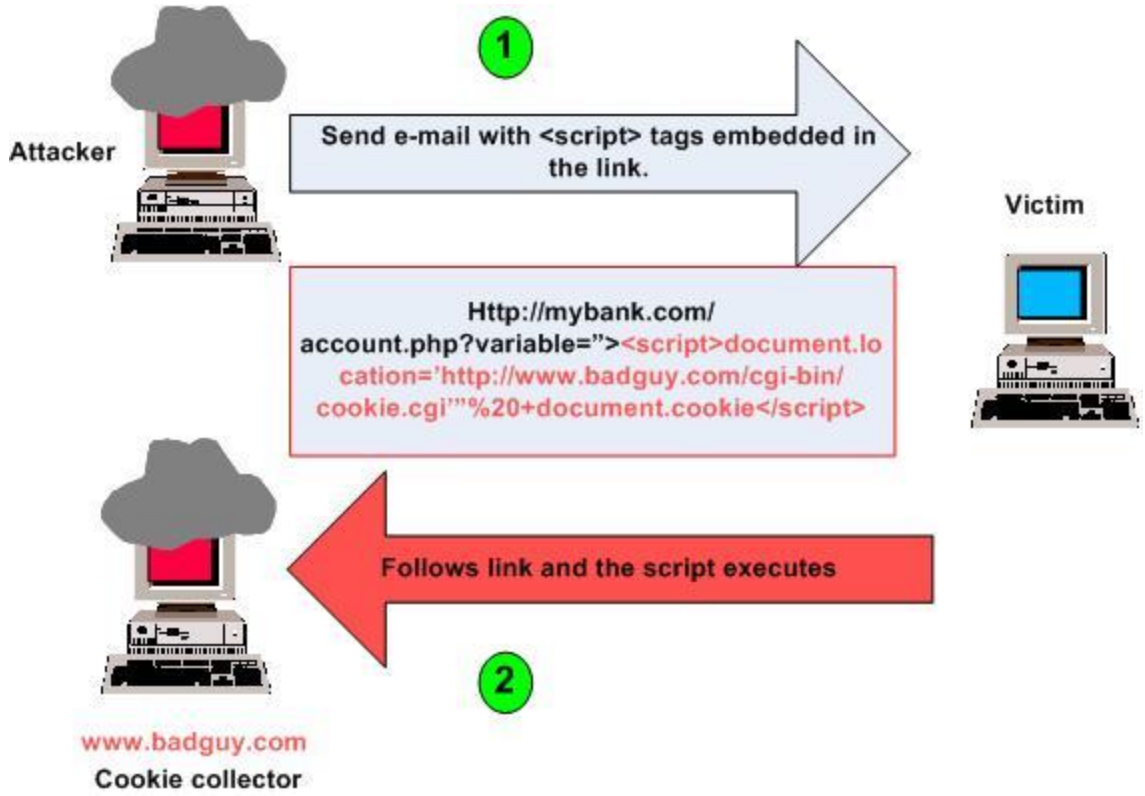
- var userId = ‘student id provided by user';
  var sql = 'SELECT * FROM studnetsWHERE id = ' + ***connection.escape(userId***);

  connection.query(sql, function(err, results) { // ... });

Definition

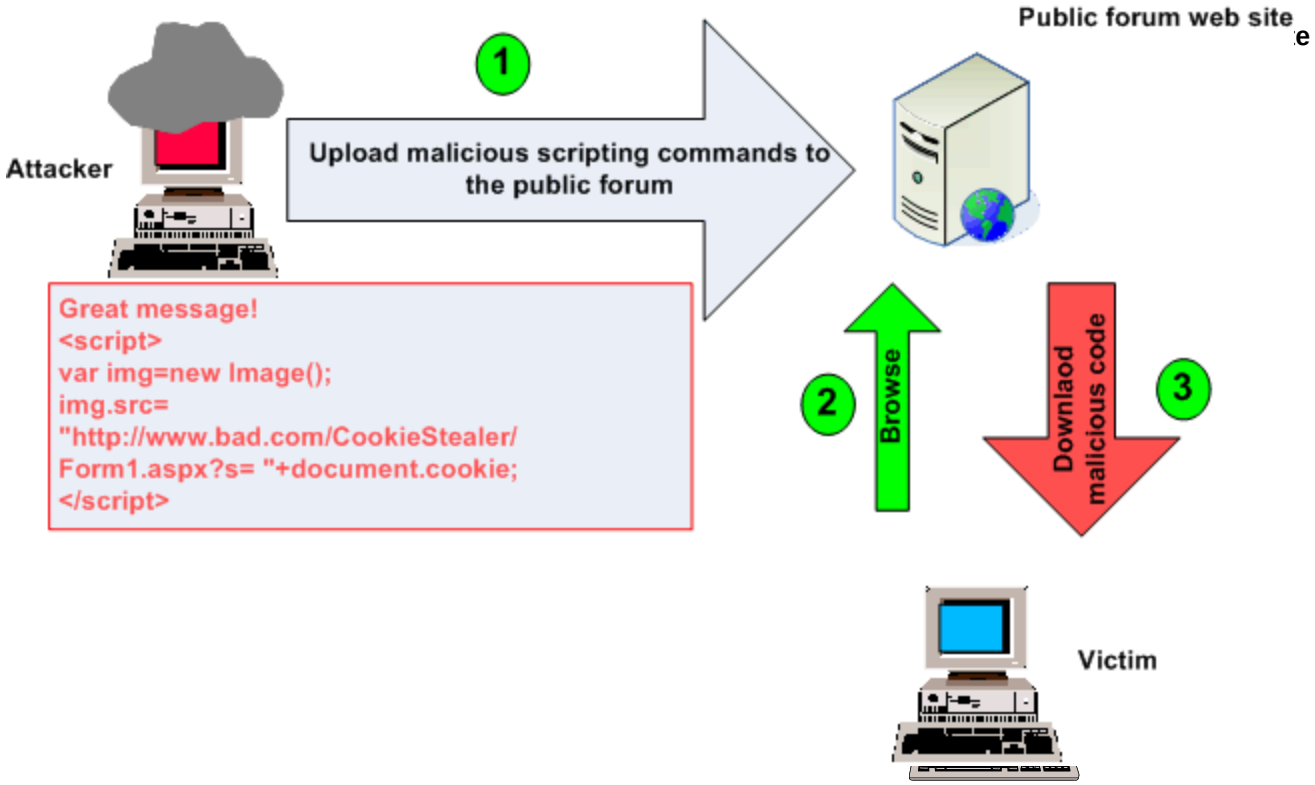
- Cross Site Scripting (XSS) is a type of computer security exploit where information from one context, where it is not trusted, can be inserted into another context, where it is
- The trusted website is used to store, transport, or deliver malicious content to the victim
- The target is to trick the client browser to execute malicious scripting commands
- JavaScript, VBScript, ActiveX, HTML, or Flash
- **Caused by insufficient input validation.**

Reflected (Non-Persistent)



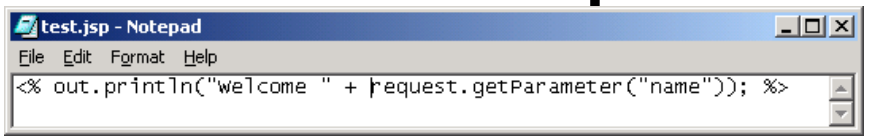
- Malicious content dose not get stored in the server
- The server bounces the original input to the victim without modification

Stored (Persistent)

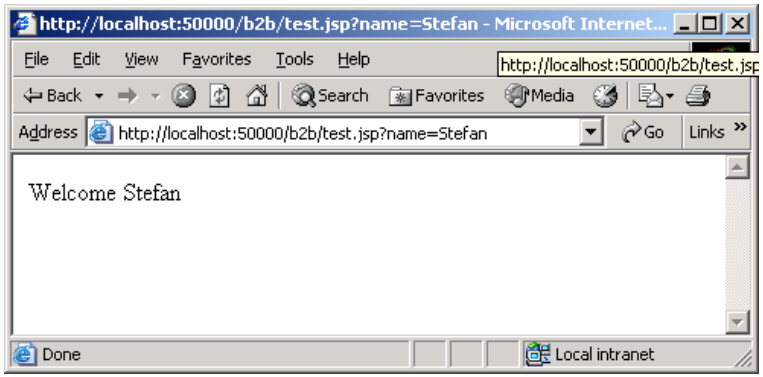


- The server stores the malicious content
- The server serves the malicious content in its original form

Simple XSS Attack

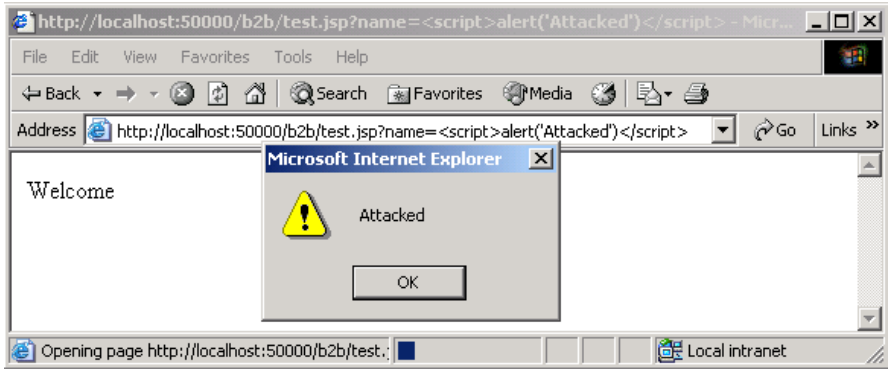


http://myserver.com/test.jsp?name=Stefan




```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</script>




```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</scrip
t>
</Body>
</HTML>
```


Source: 2005, EUROSEC GmbH Chiffriertechnik & Sicherheit

Impact of XSS-Attacks

Access to authentication credentials for Web application

- Cookies, Username and Password
  - XSS is not a harmless flaw !
- Normal users
  - Access to personal data (Credit card, Bank Account)
  - Access to business data (Bid details, construction details)
  - Misuse account (order expensive goods)
- High privileged users
  - Control over Web application
  - Control/Access: Web server machine
  - Control/Access: Backend / Database systems

Source: 2005, EUROSEC GmbH Chiffriertechnik & Sicherheit

Cross Site Scripting Defense

- Clint side
  - **Verify email**

XSS (Cross Site Scripting) Prevention Cheat Sheet

- Server side
  - **Input validation** (Black listing VS White listing)
  - **Encode** all meta characters send to the client (& : &
    " : &quot, ' : &#x27, / : &#x2F
  - **Sanitize**: <script>alert(1)</script> :
    "><script>prompt(1)</script>
  - Web application firewall
  - Always test
  - Use validator: var validator = require('validator');
    var escaped_string = validator.escape(someString);

Cross Site Scripting: References

- RSnake, XSS Cheat Sheet
http://ha.ckers.org/xss.html
- XSS Attack information
http://xssed.com/
- OWASP – Testing for XSS
http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- Klein, A., DOM Based Cross Site Scripting
http://www.webappsec.org/projects/articles/071105.shtml
- Acunetix web application security
http://www.acunetix.com
- N-stalker
http://www.nstalker.com
- How to use XSS ME
http://a4apphack.com/index.php/featured/secfox-xssme-automated-xss-detection-in-firefoxpart-3
- SANS Web Application Security Workshop

How to store Password?

(required in 2nd Lab)

```
alpha:Alfred Phangiso:A4AF8E1F5D6D15F7
bravo:David Bravo:B55D407B780C812EECC7D7D9310235F9
charlie:Charles Windsor:E97F444398BB107A
duck:Philip Ducklin:E97F444398BB107A
echo:Eric Cleese:85E3D442133F57A5E8528559FE21D853
```

- Plain text?

```
alpha:Alfred Phangiso:Alfie99
bravo:David Bravo:aprilVII2004
charlie:Charles Windsor:password
duck:Philip Ducklin:password
echo:Eric Cleese:norwegianBlue
```

- Encrypt?
- Hash (**MD5, SHA-1, and SHA-256**)

```
alpha:Alfred Phangiso:D5D459FFDFCE..7DCF3651919B
bravo:David Bravo:4620F0E4F362..9C88A6B3BD09
charlie:Charles Windsor:5E884898DA28..EF721D1542D8
duck:Philip Ducklin:5E884898DA28..EF721D1542D8
echo:Eric Cleese:89E1D86C63B8..6D0CC7424EDC
```

- Salt and Hash

```
#username:realname:salt:hash

alpha:Alfred Phangiso:0050B9..D970C4:1DC87318B512..A338DC5543EB
bravo:David Bravo:B5916E..325460:B954EF627298..3D1B21FC9DD0
charlie:Charles Windsor:49C20B..78418B:9A0A75EAB9B5..30A0253B6137
duck:Philip Ducklin:71E831..166D6A:D721A297603F..723B175381E4
echo:Eric Cleese:864E2A..A346B7:BF19240CE02E..D45DEFDB952B
```

- Hash stretch:PBKDF2 with HMAC-SHA-256
  - **Take a random key or salt K, and flip some bits, giving K1.**
  - **Compute the SHA-256 hash of K1 plus your data, giving H1.**
  - **Flip a different set of bits in K, giving K2.**
  - **Compute the SHA-256 hash of K2 plus H1, giving the final hash, H2.**

```
#username:realname:iterations:salt:hash

alpha:Alfred Phangiso:10000:0050B9..D970C4:63E75CA4..3AF24935
bravo:David Bravo:10000:B5916E..325460:53149EAE..7545E677
charlie:Charles Windsor:10000:49C20B..78418B:86B2D4AD..CD917089
duck:Philip Ducklin:10000:71E831..166D6A:585B8490..3D68A8E5
echo:Eric Cleese:10000:864E2A..A346B7:F8908212..C0D84C6C
```

Reference

- Beginners guide to a secure way of storing passwords
- **Serious Security: How to store your users' passwords safely**