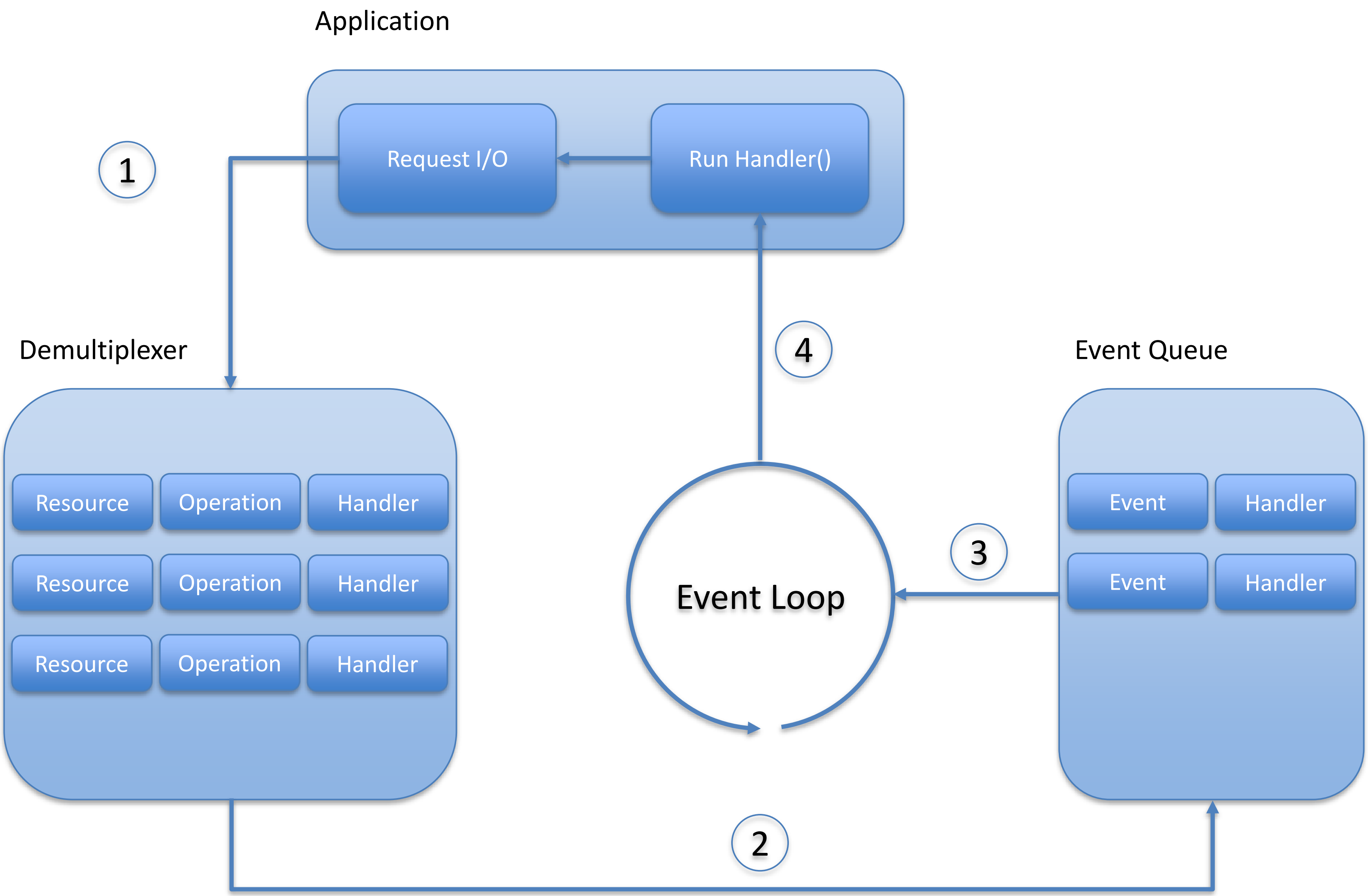
1. Provide services to another system
2. Respond to multiple overlapping requests
3. Allow and restrict user access
4. Protect my business from harm (exploits)
5. Protect user data from being harvested
6. Make changes persistent
7. Consume services from another system
8. Modularize my application
9. Display information from a source
10. Synch information shown on different views
11. Maximize responsiveness
12. Adapt to different devices and screen sizes

API+Node	1. Provide services to another system
Reactor	2. Respond to multiple overlapping requests
Express	3. Allow and restrict user access
OWASP	4. Protect my business from harm (exploits)
OWASP	5. Protect user data from being harvested
State + DB	6. Make changes persistent
API calls	7. Consume services from another system
Architecture	8. Modularize my application
Next term…	9. Display information from a source
	10. Synch information shown on different views
	11. Maximize responsiveness
	12. Adapt to different devices and screen sizes

Last week

1. How does an application implement an API?
2. Concurrency (vs Parallelism)
3. Inside the box – Node.js and ES2015



Node.js Design Patterns, 2nd edition, Casciaro and Mammino, 2016, Packt Publishing

“Closures are functions that refer to independent (free) variables. In other words, the function defined in the closure 'remembers' the environment in which it was created.”

Corollary of lexical scoping – functions are executed in scope in which they are defined (created), not scope in which executed (dynamic scope)

9.2.5 FunctionCreate (kind, ParameterList, Body, Scope, Strict, prototype)

The abstract operation FunctionCreate requires the arguments: *kind* which is one of (Normal, Method, Arrow), a parameter list production specified by ParameterList, a body production specified by Body, a Lexical Environment specified by Scope, a Boolean flag Strict, and optionally, an object *prototype*. FunctionCreate performs the following steps:

1.If the *prototype* argument was not passed, then

    a) Let *prototype* be the intrinsic object %FunctionPrototype%.

2.If *kind* is not Normal, let *allocKind* be "non-constructor".

3.Else let *allocKind* be "normal".

4.Let *F* be FunctionAllocate(*prototype*, *Strict*, *allocKind*).

5.Return FunctionInitialize(*F*, *kind*, *ParameterList*, *Body*, *Scope*).

Set the [[Environment]] internal slot of *F* to the value of *Scope*.

8.1.2.4 NewFunctionEnvironment (F, newTarget)

…

12. Set the outer lexical environment reference of *env* to the value of *F’s* [[Environment]] internal slot.

This week

1. Inside the box – Node.js and ES2015
    1. Promises (and Async/Await)
2. State and statelessness
3. Information Security

The Promise object is used for deferred and asynchronous computations.

A Promise represents an operation that hasn't completed yet, but is expected in the future.

- *pending*: initial state, not fulfilled or rejected.
- *fulfilled*: meaning that the operation completed successfully.
- *rejected*: meaning that the operation failed.

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise

```
// Normal callback usage => PYRAMID OF DOOOOOOOOM
asyncOperation(function(data){
    // Do some processing with `data`
    anotherAsync(function(data2){
        // Some more processing with `data2`
        yetAnotherAsync(function(){
            // Yay we're finished!
        });
    });
});
```

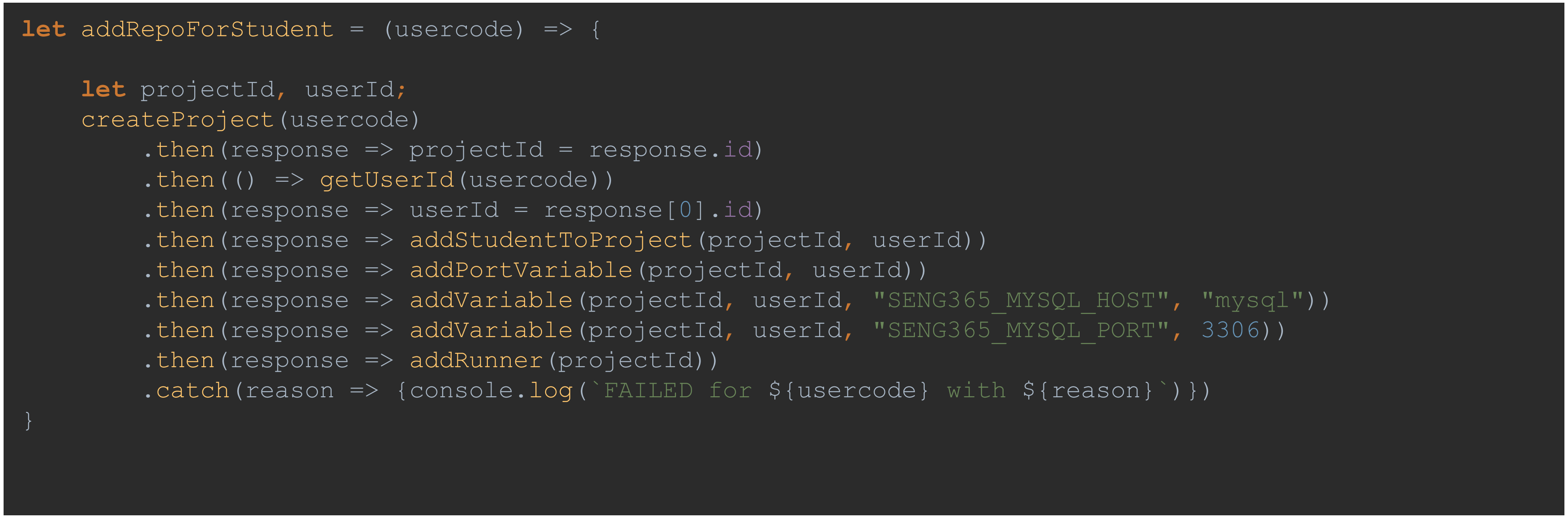
```
// Let's look at using promises
asyncOperation()
.then(function(data){
    // Do some processing with `data`
    return anotherAsync();
})
.then(function(data2){
    // Some more processing with `data2`
    return yetAnotherAsync();
})
.then(function(){
    // Yay we're finished!
});
```

http://www.telerik.com/blogs/what-is-the-point-of-promises

```
let myFirstPromise = new Promise((resolve, reject) => {
  // We call resolve(...) when what we were doing made async successful, and
reject(...) when it failed.
  // In this example, we use setTimeout(...) to simulate async code.
  // In reality, you will probably be using something like XHR or an HTML5 API.
  setTimeout(function(){
    resolve("Success!"); // Yay! Everything went well!
  }, 250);
});
```

```
myFirstPromise.then((successMessage) => {
  // successMessage is whatever we passed in the resolve(...) function above.
  // It doesn't have to be a string, but if it is only a succeed message, it
probably will be.
  console.log("Yay! " + successMessage);
});
```

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise



```
let addRepoForStudent = (usercode) => {

    let projectId, userId;
    createProject(usercode)
        .then(response => projectId = response.id)
        .then(() => getUserId(usercode))
        .then(response => userId = response[0].id)
        .then(response => addStudentToProject(projectId, userId))
        .then(response => addPortVariable(projectId, userId))
        .then(response => addVariable(projectId, userId, "SENG365_MYSQL_HOST", "mysql"))
        .then(response => addVariable(projectId, userId, "SENG365_MYSQL_PORT", 3306))
        .then(response => addRunner(projectId))
        .catch(reason => {console.log(`FAILED for ${usercode} with ${reason}`)})
}
```

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, 'foo'); // fn,delay,param
});

Promise.all([p1, p2, p3]).then(values => {
  console.log(values); // [3, 1337, "foo"]
});
```

…also `Promise.race([p1, p2, p3])`

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

1. Strict mode always
2. Use *let* and *const*, not *var* (or nothing)
3. Understand *this*, and be careful about context
   - – ES5: use .bind or var self = this
   - – ES6: =>
4. Promises in preference to callbacks

Unearthing the excellence in JavaScript

JavaScript: The Good Parts

O'REILLY® | YAHOO! PRESS

Douglas Crockford

By Robert Claypool (Own work) [CC0], via Wikimedia Commons

https://github.com/douglascrockford/JSLint

You can always get better…

1. Local groups and meetups - chc.js, APN, http://www.meetup.com/Functional-Christchurch/, http://canterburysoftware.org.nz/
2. Github for samples
3. Style guides
   - https://github.com/airbnb/javascript
   - https://google.github.io/styleguide/javascriptguide.xml
   - JSLint, ESLint
   - JSRC – preset styles
4. Patterns - e.g., https://github.com/tfmontague/definitive-module-pattern
5. Blogs, e.g.,
   - http://jrsinclair.com/articles/2016/gentle-introduction-to-functional-javascript-functions
   - https://github.com/ericelliott/essential-javascript-links#essential-javascript-links
6. ECMAScript 2018 - https://tc39.github.io/ecma262/

State and Statelessness

Memory of preceding "events"
Set of bindings

CHAPTER IV
CHAPTER V
CHAPTER VI

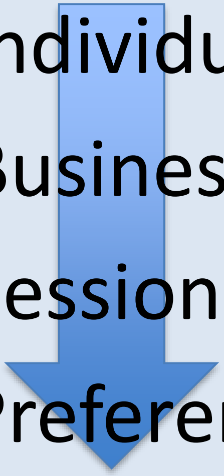
CHAPTER I

THE FAMILY OF Dashwood had been long settled in Sussex. Their estate was large, and their residence was at Norland Park, in the centre of their property, where for many generations, they had lived in so respectable a manner as to engage the general good opinion of their surrounding acquaintance. The late owner of this estate was a single man, who lived to a very advanced age, and who for many years of his life, had a constant companion and housekeeper in his sister. But her death, which happened ten years before his own, produced a great alteration in his home; for to supply her loss, he invited and received into his house the family of his nephew Mr Henry Dashwood, the legal inheritor of the Norland estate, and the person to whom he intended to bequeath it. In the society of his nephew and niece, and their children, the old Gentleman's days were comfortably spent. His attachment to them all increased. The constant attention of Mr and Mrs Henry Dashwood to his wishes, which proceeded not merely from interest, but from goodness of heart, gave him every degree of solid comfort which his age could receive; and the cheerfulness of the children added a relish to his existence.

By a former marriage, Mr Henry Dashwood had one son: by hi

https://www.flickr.com/photos/wastes/6119262807/in/photolist-ajJPDa

State timescales



Individual HTTP request (stateless)

Business transaction

Session

Preferences

Record state

Session (state) information

For *web applications* (in contrast to public websites or webpages) there is a need to maintain some stateful information about the client

State may be held:

- Client-side
- Server-side
- Hybrid

GET ? parameters

Maintain some kind of session variable in the parameter to the HTTP request

Variable does not contain the username and password, but a unique (‘random’) identifier

Include variable as a parameter in **each** network requests, e.g.

```
   GET www.example.com?sessionid=<var>
```

Why is this ‘bad practice’?

Why may something like this be needed, at times?

Cookie

Use cookies to maintain session information

The server issues a unique (‘random’) identifier in the cookie to the client

Client sends back the cookie with **each** network request to the server

- e.g. in the POST data

Cookies sent in HTTP headers:

- Set-Cookie: from server to client
- Cookie: from client to server

Cookies

A small piece of data initially sent by the server to the client.

- Comprises name-value pairs
- Also has attributes (that are not sent back to the server)

Used to maintain state information

- e.g. items in a shopping basket
  (although this example may be better kept on the server)
- e.g. browser activity such as a ‘path’ through a registration process

Sequence of cookie-ing

Request from browser

```
GET /index.htm HTTP/1.1
Host: www.example.com
```

Response from server

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: sessionToken=a1b2c3; Expires = [dat]
```

Follow-up request from browser

```
GET /profile.htm HTTP/1.1
Host: www.example.com
Cookie: sessionToken=a1b2c3
```

Information security (InfoSec)

The practice of defending information from **unauthorized access**, **use**, disclosure, disruption, modification, perusal, inspection, recording or destruction. It is a general term that can be used regardless of the form the data may take (e.g. electronic, physical).

http://en.wikipedia.org/wiki/Information_security

https://owasp.org
https://www.meetup.com/OWASP-New-Zealand-Chapter-Christchurch/
CHCon (October 2017 @ UC)

Open Web Application Security Project

Top 10 security problems (2013)
A1-Injection
A2-Broken Authentication and Session Management
A3-Cross-Site Scripting (XSS)
A4-Insecure Direct Object References
A5-Security Misconfiguration
A6-Sensitive Data Exposure
A7-Missing Function Level Access Control
A8-Cross-Site Request Forgery (CSRF)
A9-Using Components with Known Vulnerabilities
A10-Unvalidated Redirects and Forwards

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Top 10 Cheat Sheet

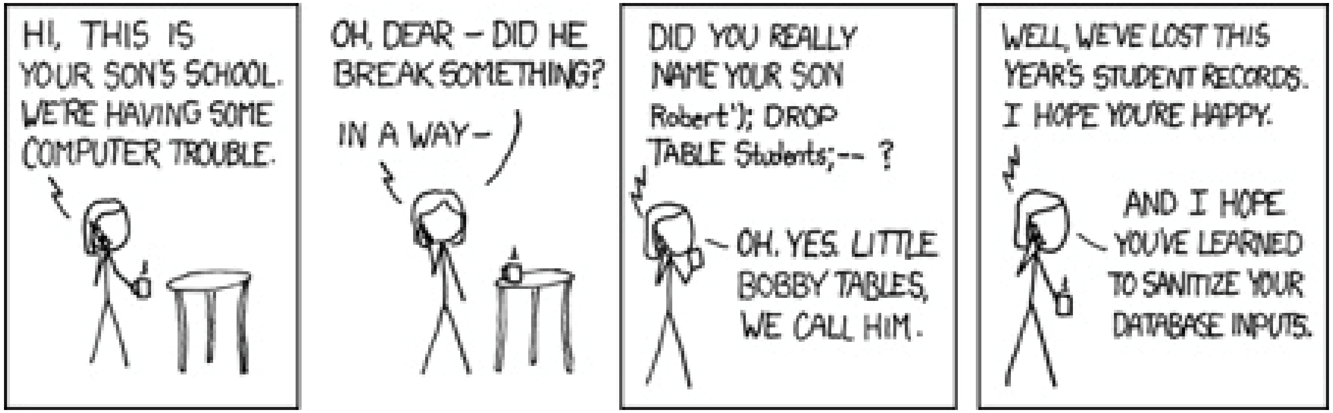
- https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet#OWASP_Top_Ten_Cheat_Sheet

- https://www.owasp.org/index.php/Category:Attack

Injection

Injection flaws allow attackers to relay malicious code through an application to another system e.g. SQL injection.

~ https://www.owasp.org/index.php/Injection_Flaws


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Injection

Any time an application uses **an interpreter of any type** there is a danger of introducing an injection vulnerability.

When a web application passes information from an **HTTP request** through as part of an external request, it must be carefully scrubbed

- why?

SQL injection is a particularly widespread and dangerous form of injection…

Command injection

Assume that we have a Java class (on the server) that gets input from the user via a HTTP request, and that class goes on to use the Java Runtime object to make an MS-DOS call e.g.

```
Runtime rt = Runtime.getRuntime();
// Call exe with userID
rt.exec("cmd.exe /C doStuff.exe " +”-“ +myUid);
```

Command injection

```
Runtime rt = Runtime.getRuntime();
// Call exe with userID
rt.exec("cmd.exe /C doStuff.exe " +”-“ +myUid);
```

When `myUid = Joe69`, we’d get the following OS call:

```
> doStuff.exe -Joe69
```

When `myUid = Joe69 & netstat –a`, we’d get:

```
> doStuff.exe -Joe69
> netstat –a // “&” is command appender in MS-DOS
```

https://www.owasp.org/index.php/Reviewing_Code_for_OS_Injection

Basis of all injections…

All injection flaws are input-validation errors.

- – i.e. you're not checking the input properly

Input is not just text fields

All external input is a source of a threat.

- – The input contains the data with the threat
- – Examples: text fields, list boxes, radio buttons, check boxes, cookies, HTTP header data, HTTP post data, hidden fields, parameter names and parameter values

Validate… and re-validate?

An input field is likely to be validated on the client side e.g. that an IDNumber textfield contains a number rather than a number and malicious code.

- What happens to that data between the client and the server?

Sometimes it’s hard to validate.

Sometimes there are multiple clients e.g. you’re offering a public web service to clients

Should you safely assume that the input data is valid because it was previously validated?

Authentication etc.

Definitions

- Authentication: establish claimed identity
- Authorisation: establish permission to act
- Authentication *precedes* authorisation

Why authenticate?

How can we authenticate?

Three factors…

HTTP is a “stateless” protocol

- Means credentials have to go with every request
- Should use SSL for everything requiring authentication

Session management flaws

- SESSION ID used to track state since HTTP doesn’t
  - and it is just as good as credentials to an attacker
- SESSION ID is typically exposed on the network, in browser, in logs, …

Beware the side-doors

- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

Typical Impact

- User accounts compromised or user sessions hijacked

Verify your architecture

- Authentication should be simple, centralized, and standardized
- Use the standard session id provided by your container
- Be sure SSL protects both credentials and session id at all times

Verify the implementation

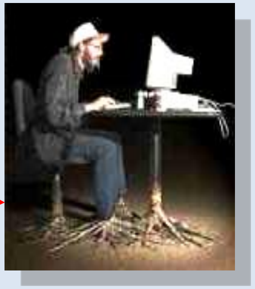
- Forget automated analysis approaches
- Check your SSL certificate
- Examine all the authentication-related functions
- Verify that logoff actually destroys the session
- Use OWASP’s WebScarab to test the implementation

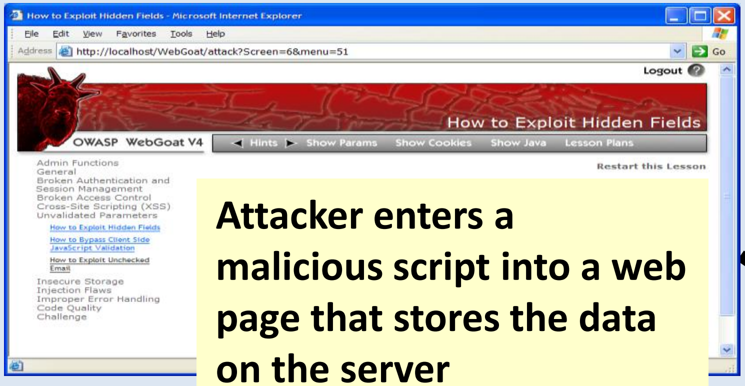
Follow the guidance from

- https://www.owasp.org/index.php/Authentication_Cheat_Sheet

Cross-Site Scripting (XSS)

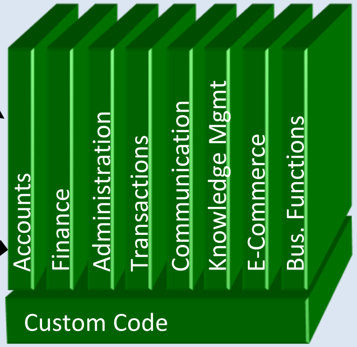
1 Attacker sets the trap – update my profile



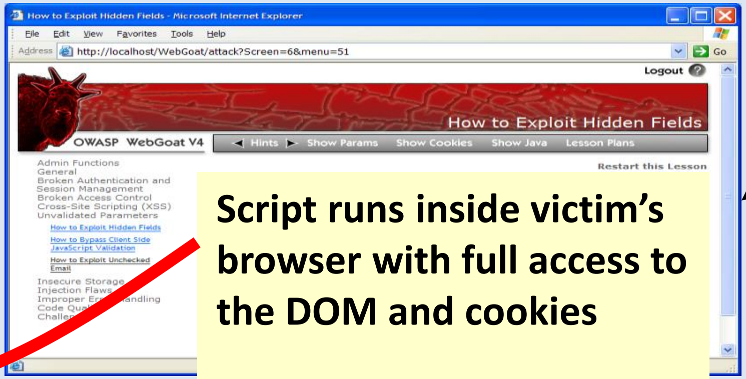

How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51 Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Hints Show Params Show Cookies Show Java Lesson Plans
Restart this Lesson
Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
How to Exploit Hidden Fields
How to Bypass Client Side JavaScript Validation
How to Exploit Unchecked Email
Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge

Attacker enters a malicious script into a web page that stores the data on the server

Application with stored XSS vulnerability



2 Victim views page – sees attacker profile


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51 Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Hints Show Params Show Cookies Show Java Lesson Plans
Restart this Lesson
Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
How to Exploit Hidden Fields
How to Bypass Client Side JavaScript Validation
How to Exploit Unchecked Email
Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge

Script runs inside victim's browser with full access to the DOM and cookies

3 Script silently sends attacker Victim's session cookie

DOM-based XSS Injection

DOM Based XSS allows an attacker to use the Document Object Model (DOM) to introduce hostile code into vulnerable client-side JavaScript embedded in many pages.

Browser interprets .js, HTML, the DOM etc

DOM-based XSS Injection

DOM based XSS is extremely difficult to mitigate against because of its large attack surface and lack of standardization across browsers.

1. Untrusted data should only be treated as displayable text. Never treat untrusted data as code or markup within JavaScript code.
2. Always JavaScript encode and delimit untrusted data as quoted strings when entering the application (Jim Manico and Robert Hansen)

Securing HTTP

HTTP is a stateless protocol

- (yeah, we know, you keep telling us…)

Each request contains all the information needed for the server to service that request

- Remember: GET, POST, PUT, DELETE etc.

Authentication precedes authorisation

- Authenticate: establish identity
- Authorise: grant permissions to act

Each request needs authenticating before authorising

- e.g. establish identify before serving up a resource

Other approaches (not prioritised)

Hash username and password

Require users to change their passwords regularly

Use multi-factor authentication

- Username & password
- Code sent by phone

Salt the username and password

- Add additional elements to the ID information

Use HTTPS (HTTP + TLS)

Next week

How do I…

1. Consume services from another system?
2. Make things persistent?
3. Modularize my application?

But now:

OWASP workshop in Lab 2